Technische Universität München

Fakultät für Informatik
Lehrstuhl für Bildverarbeitung und Mustererkennung

Efficient Algorithms for Large-Scale Correspondence Problems in Computer Vision

Frank Thomas Steinbrücker

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften
(Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr.-Ing. Darius Burschka
Prüfer der Dissertation:	1. Prof. Dr. Daniel Cremers
	2. Prof. Dr. Bastian Goldlücke Universität Konstanz

Die Dissertation wurde am 07.08.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 28.11.2018 angenommen.

Abstract

In this thesis, we investigate several aspects of inferring the content of a scene depicted in a sequence of images.

We start with the ill-posed problem of dense optical flow and disparity estimation on brightness intensity images, where we introduce a new method for efficiently estimating the optical flow of small-scale structures moving over large distances. Given a sequence of images with both dense depth estimates and brightness intensity, we continue with the problem of estimating the trajectory of a moving camera in a static scene. Finally, we address the problem of reconstructing the surface geometry and texture from a sequence of RGB-D images taken from a camera with a known trajectory. Our focus lies on dense methods, taking into account all the information contained in the images. Unlike previous methods, we are able to recover fine details, while still correlating all the given data.

A common aspect of these problems, and of computer vision problems in general, is the involvement of high-dimensional input data. To make processing of this data feasible, we need efficient algorithms. Depending on the problem, efficiency can mean both a feasible asymptotic behavior and an optimized implementation yielding real-time capability.

The former is a typical aspect of ill-posed problems, where we are given insufficient information and have to rely on additional priors in order to estimate a large number of variables. This is the case for dense optical flow estimation, where our proposed method is an alternative to the state-of-the-art approximation of a computationally infeasible problem.

The latter is typical for problems involving large amounts of input data provided in real-time, where we require our algorithms to run in real-time as well. This is the case for camera pose estimation and dense environment reconstruction in robotic applications.

Acknowledgements

Firstly, I want to thank Prof. Daniel Cremers for giving me the opportunity to work at one of the best research groups in the field of computer vision. Daniel taught me most of what I know about scientific work and about focusing on the relevant issues in my work while simultaneously seeing beyond my own nose.

I am also indebted to my coauthors Thomas Pock, Christian Kerl, and Jürgen Sturm for their fruitful and persistent collaboration.

Finally, I want to thank my colleagues Eno Töppe, Martin R. Oswald and Mohamed Souiai for proofreading my thesis.

Contents

1. Introduction and Overview

Computer vision comprises various tasks dealing with deriving digital representations of an environment observed by one or more imaging sensors. Depending on the task, these representations can live in different domains. For example, the shape of an object can be represented as a 3D surface or a 2D silhouette, and a 3D surface itself can have different representations such as a polygonal mesh, a collection of simple primitives, control points of a 3D spline etc. Motion in the environment can be represented as the 2D motion of each pixel in an image from one camera frame to the next, the 3D motion of an object in 3D space, or the ego-motion of the camera itself. Higher-level tasks include the semantic classification of objects in the environment or detection and classification of behavioral patterns in image sequences. Besides the derivation of environment representations, another set of computer vision tasks is targeted at estimating the parameters of the imaging pipeline itself, i.e. sensor calibration.

The applications of these digital representations are manifold: Knowing the per-pixel motion from one image to the next in an image sequence allows us to interpolate between images and create artificial slow motion. Knowing both the ego-motion of a camera and a digital representation of its environment enables robots to autonomously navigate through an unknown terrain and send the environment representation to a base station, where it can be used for virtual walks, distance measurements, inference of material properties, etc.

All computer vision tasks involving a sensor have in common that their input data is the output of an imaging pipeline, while their own output is a "clean" digital representation of parts of the input to that pipeline. Computer vision is therefore often regarded as solving the inverse problem of computer graphics, where the objective is to synthesize images from digital representations of the environment and imaging pipeline.

Among the central challenges in this context are that the imaging pipeline is lossy and that the input data is high-dimensional.

The lossiness of the imaging pipeline is usually caused by the presence of noise, the incapability of the sensor to represent high-frequency environment data with a fixed and comparably low sample rate, and, in the case of a projective camera, the loss of 3D information under the projection function.

The presence of high-dimensional, large-scale input data is the central topic of this thesis, which explores several tasks in image sequence processing.

In order to achieve feasible runtimes, state-of-the-art approaches tackling these tasks either rely on focusing only on a sparse subset of the data, or on downsampling the problem to a feasible scale. Both approaches are prone to neglect valuable details in the data, and the downsampling itself can be a non-trivial and costly task as well.

This thesis deals with three computer vision tasks subject to large-scale input data:

- Computing the per-pixel displacement between two images over large inter-pixel distances (Chapter 3).
- Estimating the six-degree-of-freedom ego-motion of a camera, given sequences of both texture and depth images (Chapter 4).
- Computing a joint surface representation from a sequence of depth images at large scales on commodity hardware (Chapter 5).

It is structured as follows:

Mathematical and Algorithmic Concepts

In Chapter 2 we introduce the camera model, photoconsistency, and motion models used throughout this thesis, and explain the challenges in applying energy minimization models for dense correspondence estimation under different motion models. We cover the task of stereo disparity and optical flow estimation, i.e. the problem of computing either an arbitrary displacement vector (optical flow) or the scalar disparity along an epipolar line (stereo) that matches a pixel from one image to another image.

Sparse approaches for stereo and optical flow only compute these displacement estimates at locations of sufficient spatial and temporal intensity variation of the image. In contrast, dense energy approaches assign a displacement to every pixel in the image, according to a data term describing how well the displacement matches the two input images, and correlate the estimates with a regularity term to both avoid an underconstrained problem and to robustify the approach against noise, outliers, and missing or ambiguous data.

For dense approaches, the main challenge regarding large-scale data lies in the high dimensionality of the output data. We show that with two variables estimated for every pixel in the image and a spatial coupling of the variables, the current standard formulation of the optical flow problem becomes computationally intractable.

Dense Motion Estimation Without a Coarse-to-fine Scheme

As a remedy, most previous approaches downsample the problem to a coarser scale, compute coarse displacements, and subsequently refine the displacements on finer scales. This strategy has the problem that fine-scaled structures moving over a large distance from one image to the next are neglected due to the downsampling.

In contrast, we propose a method for dense optical flow estimation in Chapter 3, that is based on decoupling the regularity term from the data term. We published our method in [4]. Since it is not relying on downsampling, our method is capable of accurately estimating the displacements of fine-scaled structures. Furthermore, it allows us to use arbitrary data terms, as we have demonstrated in [3].

Camera Pose Estimation on RGB-D Sequences

In Chapter 4, we cover the topic of camera tracking on images containing both a brightness intensity value as well as a depth value in every pixel. The depth estimates can be provided either by a passive photometric stereo approach as described in Chapter 3, by a photometric stereo method aided by an illuminator in a sensor based on structured light patterns, or by a sensor based on the time-of-flight principle. Publications based on recently developed active sensors in the consumer market [42, 83] have coined the term RGB-D cameras and RGB-D images (Red, Green, Blue, and Depth) for such cameras and images. Tracking the camera on a sequence of such images means estimating the relative 6-degree-of-freedom pose transformation the camera has undergone in the time from capturing one image to the next.

This problem is closely related to optical flow and stereo disparity estimation. Instead of estimating a displacement for every pixel, we estimate a camera movement producing all the displacements in the image. The problem can also be regarded as the inverse problem to stereo disparity estimation: For stereo disparity estimation, we are given the relative camera pose transformation and want to estimate the disparity (and therefore the depth) in every pixel. For camera tracking on RGB-D sequences, we are given the depth and want to estimate the relative camera pose transformation. Both problems can co-occur, for example if we want to estimate the trajectory of a binocular camera from a sequence of calibrated stereo images.

The challenge regarding large-scale data lies in the real-time requirements of many applications using novel active RGB-D cameras. Cameras like Microsoft's 1st generation Kinect or Asus' Xtion Pro capture RGB-D images at 30 frames per second. Being able to track the camera with the same speed on commodity hardware opens up a wide field of applications such as robotic navigation, simultaneous localization and mapping (SLAM), or real-time online reconstruction of 3D environments. Most previous approaches for camera tracking either use a sparse subset of visual features based purely on the color information of the images, or try to align the depth maps by an iterative closest point (ICP) method. Both approaches suffer from computationally costly correspondence estimation and outlier rejection.

In contrast, we propose a method for camera tracking using both dense image intensity and depth information, and outperforming a comparable ICP implementation by means of both accuracy and speed. We published this approach in [5].

Large-Scale 3D Reconstruction from RGB-D Sequences

In Chapter 5, we assume to be given an image sequence of a static scene with both dense depth information and a camera pose for every image. Our goal is to densely reconstruct the joint surface geometry in real-time. State-of-the-art approaches rely on an implicit volumetric representation of the surface geometry that can handle arbitrary changes in topology. However, the worst-case memory demand increases cubically with the number of integrated RGB-D pixels. The challenge regarding large-scale data in this case is the sheer amount of voxels that need to be updated for the integration of every RGB-D image. On the one hand, all previous approaches for real-time depth map fusion require parallelization of the algorithm on a graphics processing unit (GPU). On the other hand, the comparatively small amount of available memory on GPUs is an additional limiting factor with respect to the size and resolution of the reconstructed scene.

In contrast to previous approaches, we propose a method that only stores and updates those voxels in the volume actually bearing information about the surface geometry. Furthermore, we represent the geometry adaptively on different scales, according to the resolution we get from the camera observing it at different distances. As a result, our method is able to store large scenes in the limited amount of available GPU memory, as we demonstrated in [1]. Moreover, we show that with a number of optimizations for serial processing and Single-Instruction-Multiple-Data (SIMD) parallelism, which is supported by all central processing units (CPUs) in current commodity desktops and laptops, we are able to perform our dense RGB-D image fusion method on a single laptop CPU core.

In addition, we address the problem of map transmission and visualization. Previous approaches involving an implicit volumetric surface representation use raycasting for visualization. Raycasting heavily relies on GPU parallelization, and it requires a new image to be rendered for every virtual camera pose observing the geometry. Therefore, it is not well suited for decoupling the visualization of the observed geometry from the machine performing the volumetric geometry fusion, and running the volumetric geometry fusion on an embedded platform.

Different to previous approaches, we extract a triangle mesh of the geometry surface incrementally on a second CPU core. The triangle mesh has a comparatively low memory footprint and it can be transferred to a different machine and be used there as both a map for tasks like path planning or obstacle avoidance and for visualization. We published both the optimization of our approach for running on a CPU and the incremental mesh extraction in [6].

Conclusion and Outlook

In Chapter 6 we summarize our work and give an outlook on future work, illustrating how some our work can be improved.

2. Mathematical and Algorithmic Concepts

2.1. Notation

Scalars, Vectors, and Matrices

- We do not distinguish points from vectors in our notation.
- We use normal lower case letters for scalars such as temporal indices $t, t_0, t_1, ... \in \mathbb{R}$.
- We use bold lower case letters for 2D vectors such as pixel indices $\mathbf{p} \in \Omega \subset \mathbb{R}^2$. Depending on the context, we use either simple letters such as x and y, or subscripted letters such as f_x, f_y, c_x, c_y to denote the elements of different 2D vectors.
- For 3D and higher-dimensional vectors and matrices we use bold upper case letters. Similar to 2D vectors, we use either simple or subscripted upper case letter for the elements of 3D vectors.
- For 2D and 3D points $\mathbf{p} \in \mathbb{R}^2$ and $\mathbf{P} \in \mathbb{R}^3$, we use the notation $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{P}}$ for their corresponding homogeneous coordinates in the 2D and 3D projective spaces $\mathbb{R}P^2$ and $\mathbb{R}P^3$.

Coordinate Transformations

- When we describe coordinate transformations in the form of matrix multiplications, like rotating the point $\mathbf{P} \in \mathbb{R}^3$ around the origin with the rotation matrix $\mathbf{R}$, we always multiply the operator from the left, i.e. $\mathbf{RP}$.
- When we refer to the rotation/orientation and translation/position of a camera $\mathbf{R}$ and $\mathbf{T}$, we describe the transformation of a point from camera coordinates to “world” coordinates. This infers that $\mathbf{T}$ is the position of the camera center in “world” coordinates.

Derivatives

- For compositions of functions, with one or more variables, we use the total differential notation to describe the derivative of the composition function. A 1D example is the composition $g(f(x))$, where we write $\frac{\mathrm{d}f(g(x))}{\mathrm{d}x}$ for the derivative of the composite function $h := (f \circ g)$ with respect to x, without explicitly defining h. Similarly, if we have a multivariable function $f(g(x), h(x), x)$, where f is explicitly dependent on x both explicitly and implicitly through g and h, which themselves are - potentially implicitly - dependent on x, we write $\frac{\partial f}{\partial x}$ to describe the partial derivative with respect to the explicit dependency, whereas we write $\frac{\mathrm{d}f(g(x),g(x),x)}{\mathrm{d}x}$ to describe the derivative of the composite function.

- We often use the short form ∂_x for the partial derivative $\frac{\partial}{\partial x}$.
- For the point at which we evaluate a derivative, we choose the notation

$$\left(\frac{\partial f(x_1, x_1, x_3, ...)}{\partial x_{\{1,2,3,...\}}}\right)\Bigg|_{y_1,y_2,y_3,...} \quad \text{or} \quad \left(\frac{\mathrm{d}f(g_1(x), g_2(x), g_3(x), ...)}{\mathrm{d}x}\right)\Bigg|_{y_1,y_2,y_3,...}$$

to describe that the derivatives are evaluated at the location $x_1 = y_1, x_2 = y_2, x_3 = y_3, ...$, or $g_1(x) = y_1, g_2(x) = y_2, g_3(x) = y_3, ...$, respectively. If we have introduced a function before and the order of arguments is known from context, we might choose to omit the arguments in the numerator for better readability, for example like

$$\left(\frac{\partial f}{\partial x}\right)\Bigg|_{y_1,y_2,y_3} .$$

Predicates

In some of the formulas we use Iverson's notation to convert predicates to binary values, analogously to standard digital comparator arithmetic [50]:

$$[P] = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{else} \end{cases} . \tag{2.1}$$

2.2. Camera Model

Throughout this thesis we assume the input images were acquired by a pinhole camera. In the pinhole camera model, a ray of light passes on a straight line from the surface through the camera center onto the image plane, without being refracted by a camera lens. Figure 2.1 is a 2D schematic of the perspective projection of the pinhole camera model. A 3D point $\mathbf{P}$ is projected on a straight line through the camera center $\mathbf{C} \in \mathbb{R}^3$ onto the rectangular image plane $\Omega' \subset \mathbb{R}^2$ behind the camera center on the optical axis. For convenience, we use the virtual image plane $\Omega \subset \mathbb{R}^2$ in front of the camera center instead of the real image plane Ω' in this thesis, and without loss of generality, we use

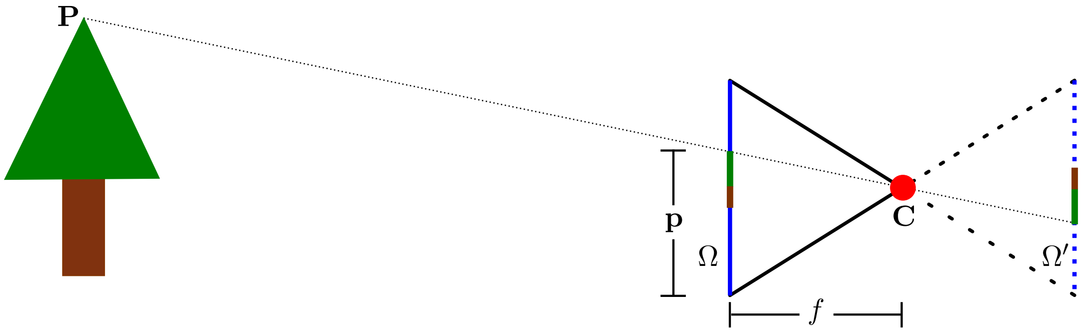


Figure 2.1.: Pinhole camera model

a right-handed camera coordinate system, where X-values increase from left to right, Y-values increase from top to bottom, and Z-values increase from near to far.

The focal length of the pinhole camera is the distance $f \in \mathbb{R}_{>0}$ of the image plane to the camera center. To model the process of creating a pixel image, the focal length f in Meters is multiplied with the pixel size in x- and y-direction, yielding the scaled focal lengths f_x and f_y with a unit of measure in pixels. Furthermore, we represent the position of the image plane Ω in the XY-plane in camera coordinates by the offset $\mathbf{c} = \begin{bmatrix} c_x & c_y \end{bmatrix}^\mathsf{T} \in \mathbb{R}^2$ of the optical axis in the image plane in pixels.

The pixel position $\mathbf{p} = \begin{bmatrix} x & y \end{bmatrix}^\mathsf{T} \in \Omega$ of a 3D point $\mathbf{P} = \begin{bmatrix} P_X & P_Y & P_Z \end{bmatrix}^\mathsf{T} \in \mathbb{R}^3$ can now be computed as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{P_X}{P_Z} f_x + c_x \\ \\ \frac{P_Y}{P_Z} f_y + c_y \end{bmatrix}. \tag{2.2}$$

In the following, we will denote the projection of a 3D point $\mathbf{P}$ into a camera with some intrinsic parameters $\mathbf{K}$ by the function

$$\pi : \mathbb{R}^3 \to \mathbb{R}^2, \mathbf{P} \mapsto \pi(\mathbf{P}). \tag{2.3}$$

Using 2D homogeneous coordinates for $\mathbf{p}$, this projective transformation can be expressed as a multiplication with the intrinsic camera matrix $\mathbf{K}$:

$$\tilde{\mathbf{p}} \in \mathbb{R}P^2 = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{KP} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_X \\ P_Y \\ P_Z \end{bmatrix} \tag{2.4}$$

If $\mathbf{P}$ is not given in camera coordinates, but in world coordinates, and if we want to project $\mathbf{P}$ into a camera rotated with $\mathbf{R} \in \mathrm{SO}(3)$ and translated with $\mathbf{T} \in \mathbb{R}^3$ (see 2.1), we have to apply the inverse transformation to $\mathbf{P}$ beforehand

$$\mathbf{p} = \pi(\mathbf{R}^\mathsf{T}\mathbf{P} - \mathbf{R}^\mathsf{T}\mathbf{T}). \tag{2.5}$$

Representing $\mathbf{P}$ in 3D homogeneous coordinates as $\tilde{\mathbf{P}} = \begin{bmatrix}\mathbf{P} & 1\end{bmatrix}^\mathsf{T}$ and using the matrix $\mathbf{\Pi}$ to project back to Euclidean coordinates, we can rewrite Equation (2.5) with the homogeneous transformation matrix $\mathbf{M}$ as

$$\begin{aligned}
\mathbf{M} &= \begin{bmatrix}\mathbf{R}^\mathsf{T} & -\mathbf{R}^\mathsf{T}\mathbf{T}\\ 0 & 1\end{bmatrix} = \begin{bmatrix}R_{11} & R_{21} & R_{31} & -R_{11}T_1 - R_{21}T_2 - R_{31}T_3\\ R_{12} & R_{22} & R_{32} & -R_{12}T_1 - R_{22}T_2 - R_{32}T_3\\ R_{13} & R_{32} & R_{33} & -R_{13}T_1 - R_{23}T_2 - R_{33}T_3\\ 0 & 0 & 0 & 1\end{bmatrix}\\
\mathbf{\Pi} &= \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\\
\tilde{\mathbf{p}} &= \mathbf{K\Pi M}\tilde{\mathbf{P}} \qquad (2.6)
\end{aligned}$$

Finally, we will describe the image captured by a camera as a real-valued function on the rectangular image plane,

$$\mathcal{I} : \Omega \to \mathbb{R}, \quad \mathbf{p} \mapsto \mathcal{I}(\mathbf{p}) \qquad (2.7)$$

$$\mathcal{I}^c : \Omega \to \mathbb{R}^3, \quad \mathbf{p} \mapsto \mathcal{I}^c(\mathbf{p}) = \begin{bmatrix}\mathcal{R}\\ \mathcal{G}\\ \mathcal{B}\end{bmatrix}(\mathbf{p}). \qquad (2.8)$$

where $\mathcal{I}$ in Equation (2.7) refers to gray-valued intensity images and $\mathcal{I}^c$ in Equation (2.8) refers to RGB-color images. Naturally, there are many other types of images, for example range or disparity images, which will play a role in Chapters 4 and 5. However, unless explicitly stated otherwise, we consider gray-valued intensity images in the rest of this thesis when referring to an image.

2.3. Photoconsistency

In this thesis, we deal with correspondence problems in a twofold way: First, Chapters 3, 4 and the last sections of this chapter address the problem of putting parts of different images into correspondence with one another. Second, Chapter 5 deals with the problem of applying correspondences found between images to reconstruct a geometry model. For the first task, finding the correspondences, we require that the scene geometry we observe in different images looks the same in these images, even though taken from different angles. A common premise for this is that the geometry captured by a camera has a surface subject to an isotropically diffuse or Lambertian reflectance model. In short, this implies that the texture observed on the surface by a camera is independent of the angle spanned by the camera center, the surface point, and the surface normal. This assumption is a premise for the notion of photometric consistency, or photoconsistency.

For a set of n cameras $\{C_i\}_{i=0}^{n-1}$ with camera poses $\mathbf{M}_i$, capturing images $\mathcal{I}_i$, a point $\mathbf{P} \in \mathbb{R}^3$ is photoconsistent, if

$$\forall i, j \in \{0, ..., n-1\} : \quad \mathcal{I}_i(\pi_i(\mathbf{M}_i\mathbf{P})) - \mathcal{I}_j(\pi_j(\mathbf{M}_j\mathbf{P})) = 0 \tag{2.9a}$$

and a trivial pairwise photoconsistency score (the lower the better) for two cameras and images i and j is the absolute difference

$$|\mathcal{I}_i(\pi_i(\mathbf{M}_i\mathbf{P})) - \mathcal{I}_j(\pi_j(\mathbf{M}_j\mathbf{P}))| \,. \tag{2.9b}$$

A common assumption in computer vision is that 3D points lying on a Lambertian surface are photoconsistent. The thus defined concept of photoconsistency can be used for the following scenarios: On the one hand, for known camera poses and intrinsic parameters, one can infer the surface geometry of the scene with this assumption, provided that one has a strategy for removing false correspondences. This will be detailed in the next section. On the other hand, for a known surface geometry and intrinsic camera parameters, one can infer the camera poses. This case will be covered in Chapter 4.

Naturally, the assumption of a Lambertian surface is an idealized one and proves true only theoretically. Other issues corrupting photoconsistency are various forms of noise, occlusions, and illumination changes. The latter will be covered in Section 3.5.

2.4. Motion Models

2.4.1. Rigid Motion Model

When a scene is observed concurrently by at least two cameras C_0 and C_1 with known poses $\mathbf{M}_0$ and $\mathbf{M}_1$, we can use the photoconsistency assumption of Equation (2.9a) to infer which points $\mathbf{P}$ lie on a 3D surface.

While many methods in computer vision involve computing a photoconsistency score for all points on a Euclidean 3D grid [88, 52, 84], the surfaces in this thesis are parametrized as range images or depthmaps in the projective volume of one camera. The projective volume is the preimage of the image plane Ω under the perspective projection.

Figure 2.2 shows an example of a depthmap. Instead of sampling the photoconsistency on a 3D grid, we sample the photoconsistency on the ray going from the camera center through each pixel $\mathbf{p_0}$ in the image plane Ω_0 of camera C_0. Assuming C_0 is rotated with $\mathbf{R}_0$ and translated with $\mathbf{T}_0$, and that it has the intrinsic parameters $\mathbf{K}_0$, then this ray is the set of points $\mathbf{P}$ that fulfill

$$\mathbf{P} = \mathbf{R}_0\mathbf{K}_0^{-1}h\tilde{\mathbf{p_0}} + \mathbf{T}_0 \tag{2.10}$$

for a positive $h \in \mathbb{R}_{\geq 0}$. Projecting this ray into a second camera C_1, we get

$$p_1 = \pi_1\left(\mathbf{R}_1^\top\mathbf{R}_0\mathbf{K}_0^{-1}h\tilde{\mathbf{p_0}} + \mathbf{R}_1^\top\mathbf{T}_0 - \mathbf{R}_1^\top\mathbf{T}_1\right), \tag{2.11}$$

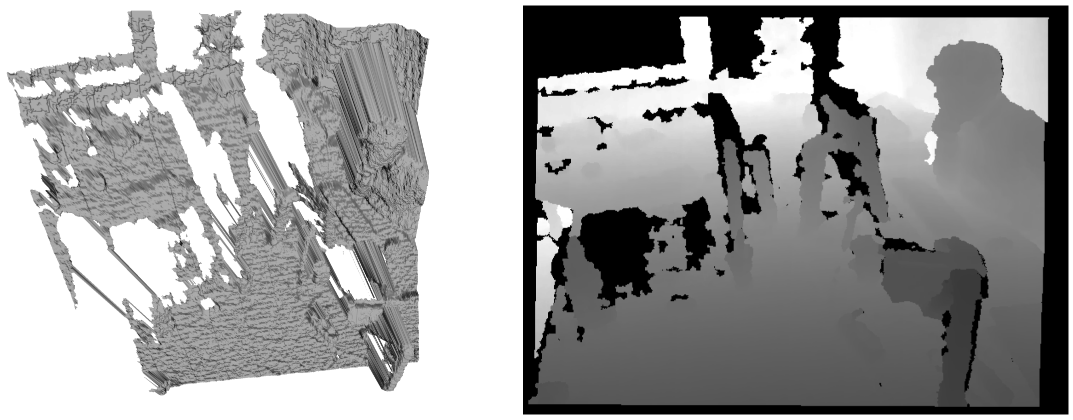


Figure 2.2.: 2D depthmap (left) and the corresponding surface visualized in 3D (right). Bright intensity values in the image imply a large distance of the surface point from the camera center.

which is the line equation of the epipolar line of point $\mathbf{p_0}$ in the image plane Ω_1.

It is the projection of the epipolar plane defined by the two camera centers $\mathbf{T}_0$, $\mathbf{T}_1$ and the true surface point $\mathbf{P}$, and it has the projected camera center

$$\mathbf{e}_1 = \pi_1\left(\mathbf{R}_1^\top(\mathbf{T}_0 - \mathbf{T}_1)\right) \tag{2.12}$$

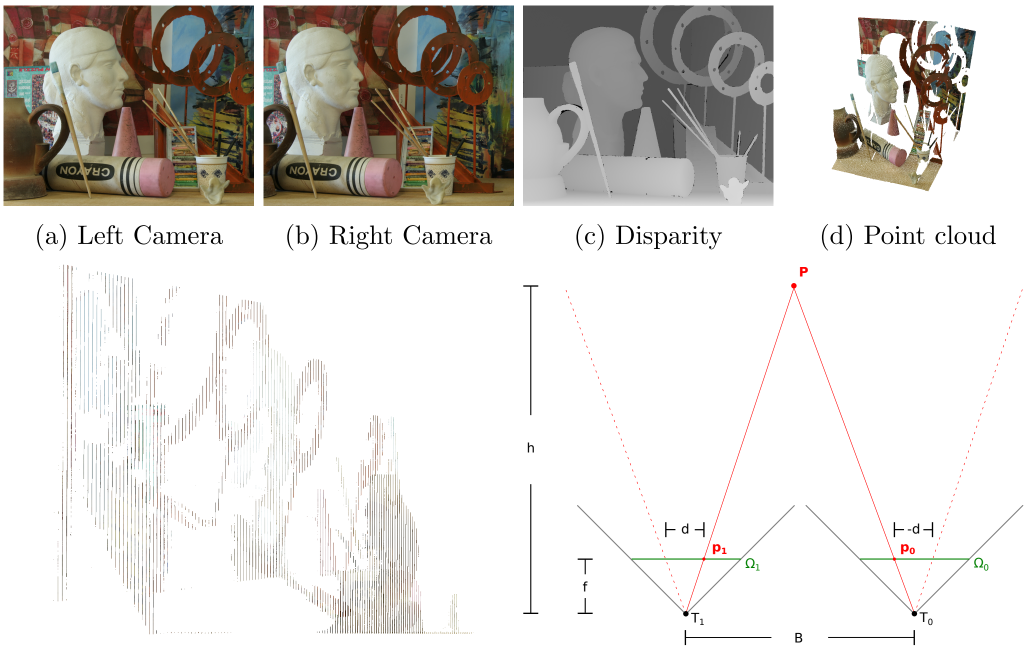
as one of its epipoles. Along the ray defined in Equation (2.10), only the 3D surface point $\mathbf{P}^*$ that is closest to the camera center at $\mathbf{T}_0$ is visible in $\mathbf{p_0}$, and we parametrize it by its depth h^*, i.e. its displacement to the camera center projected onto the optical axis of the camera. If $\mathbf{P}^*$ is not occluded in $\mathcal{I}_1$, we can assume it to be photoconsistent in $\mathcal{I}_0$ and $\mathcal{I}_1$. Combining the epipolar geometry with the photoconsistency definition in Equation (2.9a), we get

$$\mathcal{I}_1\left(\pi_1\left(\mathbf{R}_1^\top\mathbf{R}_0\mathbf{K}_0^{-1}h^*\tilde{\mathbf{p_0}} + \mathbf{R}_1^\top\mathbf{T}_0 - \mathbf{R}_1^\top\mathbf{T}_1\right)\right) - \mathcal{I}_0\left(\mathbf{p_0}\right) = 0. \tag{2.13a}$$

Moreover, an intuitive assumption is that $\mathbf{P}^*$ is the point with the best photoconsistency score along the ray:

$$h^*(\mathbf{p_0}) = \arg\min_h \left\{\left|\mathcal{I}_1\left(\pi_1\left(\mathbf{R}_1^\top\mathbf{R}_0\mathbf{K}_0^{-1}h\tilde{\mathbf{p_0}} + \mathbf{R}_1^\top\mathbf{T}_0 - \mathbf{R}_1^\top\mathbf{T}_1\right)\right) - \mathcal{I}_0\left(\mathbf{p_0}\right)\right|\right\}. \tag{2.13b}$$

A special case is given, if two cameras have the same intrinsic parameters, the same rotation matrix $\mathbf{R}$, and a horizontal relative translation, i.e. $\mathbf{R}^\top(\mathbf{T}_0 - \mathbf{T}_1) = B\begin{bmatrix}1 & 0 & 0\end{bmatrix}^\top$, with the baseline $B \in \mathbb{R}$. In this case, the denominator in the perspective projection equation (2.2) for the epipole (2.12) is zero, yielding horizontally parallel epipolar lines



(a) Left Camera (b) Right Camera (c) Disparity (d) Point cloud

(e) Orthographic side view (f) Rectified stereo schematic view

Figure 2.3.: Image disparity caused by a horizontally shifted viewpoint. 2.3a-2.3e: Middlebury “Art” sequence. 2.3f: Schematic view of the relation between depth, focal length, baseline and disparity in a rectified stereo setting.

and epipoles at $x = \pm\infty$. In this case of rectified images, the depth h of a 3D point $\mathbf{P}$ in the cameras yields an inversely proportional disparity d in the images, which is the displacement between the two projected points $\mathbf{p_0}$ and $\mathbf{p_1}$. If B denotes the length of the relative baseline as introduced above, i.e. $B = \left\langle \begin{bmatrix}1 & 0 & 0\end{bmatrix}^\mathsf{T}, \mathbf{R}^\mathsf{T}(\mathbf{T}_0 - \mathbf{T}_1) \right\rangle$, then the disparity is given as

$$d = \mathbf{p_1}_x - \mathbf{p_0}_x = B\frac{f_x}{h}. \tag{2.14}$$

Figure 2.3f shows a schematic view of this relationship. Figures 2.3a - 2.3e show an example of such a setup from the Middlebury “Art” sequence. For rectified images, Equation (2.13b) can be reformulated as

$$h^*(\mathbf{p_0}) = B\frac{f_x}{d^*(\mathbf{p_0})}, \quad \text{and} \quad d^*(\mathbf{p_0}) = \arg\min_d \left\{ \left| \mathcal{I}_1(\mathbf{p_0}_x + d, \mathbf{p_0}_y) - \mathcal{I}_0(\mathbf{p_0}) \right| \right\}, \tag{2.15}$$

where the search space for a point of optimal photoconsistency is 2D instead of 3D. Iterating over points in image space and re-projecting only those with optimal correspondence into 3D space is computationally slightly less expensive than iterating over

3D points and projecting every point onto the image plane, as it omits many projection operations. Moreover, it has the advantage that a uniform sampling for correspondence points along the epipolar line coincides with the sampling distribution of discrete digital images. For example, all the points $h \geq Bf_x$ correspond to disparities of $|d| \leq 1$. If we were to compute the photoconsistency of points uniformly sampled on the 3D ray, the points of large depth h would have a very similar photoconsistency score. If we compute the photoconsistency for uniformly sampled disparity values, e.g. for every full pixel position in a given range, this makes better use of the given data and leads to a reciprocal sample distribution on the 3D ray. Figure 2.3e demonstrates this effect. It shows an orthographic projection onto the YZ-plane of the point cloud created from the disparity image in 2.3c. For uniformly sampled disparity values, the depth resolution decreases for lower disparity / higher depth values. In order to adequately estimate large depth values at disparity values close to zero, we need both a large baseline and a large focal length, which in our model represents both the zoom-level and the image resolution. However, because digital images do not only have a discrete sampling rate, but also a discrete quantization, the more samples we have for a given range of depth values, the more likely it is that two samples correspond to the same image value. This makes the disparity value maximizing the photoconsistency in Equation (2.15) not unique and renders the assumption that the correct depth value minimizes Equation (2.13b) incorrect.

To overcome this problem and to get a well-defined solution for the depth, there exist two widely used strategies in computer vision. The first one is the use of advanced photoconsistency assumptions, which we will explore in Section 2.5, and the second one is the use of regularity priors, which will be explained in Section 2.6. In Section 3.5, we present a combination of the two models.

2.4.2. Dynamic Motion Model

Photoconsistency plays a role not only for a static geometry observed by several cameras at once, but also for a dynamic geometry with moving objects and a single camera capturing image streams. In this case, the images captured by the camera also depend on a temporal index:

$$\mathcal{I} : \Omega \times \mathbb{R}_{\geq 0} \to \mathbb{R}, (\mathbf{p}, t) \mapsto \mathcal{I}(\mathbf{p}, t) \tag{2.16}$$

If the entire observed scene moves rigidly, i.e. the motion of every point in the scene can be represented by the same six parameters for rotation and translation, the captured image stream is identical to an image stream of a static scene with the camera moving in an inverse way. Using the same argument, the camera motion can always be interpreted inversely as motion of single objects in the scene and we can reduce problems involving both a moving camera and scene to a problem where only the scene is moving whereas the camera remains static. In other words, it does not matter which coordinate frame we chose as a reference, so we can just as well chose the one with the camera pose in the origin.

The most general way to describe the motion of a 3D scene in every point is defining a 3D mapping or "warping" function

$$\begin{aligned}&\mathbf{W} : \mathbb{R}^3 \times \mathbb{R}_{\geq 0} \to \mathbb{R}^3,\\ &(\mathbf{P}, t) \to \mathbf{W}(\mathbf{P}, t),\\ &\mathbf{W}(\mathbf{P}, 0) = \mathbf{P} \quad \forall \mathbf{P} \in \mathbb{R}^3\end{aligned} \tag{2.17}$$

for every point in 3D space and time. Combined with the perspective projection of Equation (2.2) and knowledge about which 3D points lie on a surface visible to the camera, this induces a 2D mapping/warping function

$$\begin{aligned}&\mathbf{w} : \Omega \times \mathbb{R}_{\geq 0} \to \mathbb{R}^2,\\ &(\mathbf{p}, t) \to \mathbf{w}(\mathbf{p}, t),\\ &\mathbf{w}(\mathbf{p}, 0) = \mathbf{p} \quad \forall \mathbf{p} \in \Omega\end{aligned} \tag{2.18}$$

in the image plane. With this motion model, the photoconsistency assumption can be written as

$$\forall t_0, t_1 \in \mathbb{R}_{\geq 0}, \forall \mathbf{p} \in \Omega : \quad \mathcal{I}(\mathbf{w}(\mathbf{p}, t_1), t_1) - \mathcal{I}(\mathbf{w}(\mathbf{p}, t_0), t_0) = 0. \tag{2.19}$$

Assuming that the motion of the scene is differentiable (see Section 4.2.2 for a detailed explanation of differentiable camera motion), we can rewrite Equation (2.19) as

$$\forall t \in \mathbb{R}_{\geq 0}, \forall \mathbf{p} \in \Omega : \quad \frac{\mathrm{d}\mathcal{I}(\mathbf{w}(\mathbf{p}, t), t)}{\mathrm{d}t} = 0. \tag{2.20}$$

and by applying the chain rule we get

$$\forall t \in \mathbb{R}_{\geq 0}, \forall \mathbf{p} \in \Omega : \quad \left(\frac{\partial \mathcal{I}(\mathbf{p}, t)}{\partial \mathbf{p}}\right)\Bigg|_{\mathbf{w}(\mathbf{p},t),t} \left(\frac{\partial \mathbf{w}(\mathbf{p}, t)}{\partial t}\right)\Bigg|_{\mathbf{p},t} + \left(\frac{\partial \mathcal{I}(\mathbf{p}, t)}{\partial t}\right)\Bigg|_{\mathbf{p},t} = 0. \tag{2.21}$$

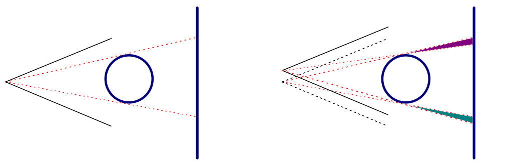
Finally, if we have $t = 0$ and $\forall \mathbf{p} : \ \mathbf{w}(\mathbf{p}, 0) = \mathbf{p}$, and define the velocity

$$\nu : \Omega \times \mathbb{R}_{\geq 0} \to \mathbb{R}^2 \text{ as } \nu(\mathbf{p}, t) = \left(\frac{\partial \mathbf{w}(\mathbf{p}, t)}{\partial t}\right)\Bigg|_{\mathbf{p},t} \tag{2.22}$$

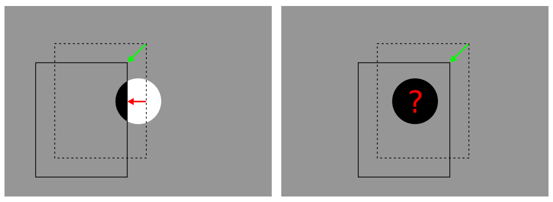
we get the well known optical flow equation

$$\nabla \mathcal{I}^{\mathsf{T}} \nu + \partial_t \mathcal{I} = 0 \tag{2.23}$$

with $\nabla \mathcal{I} = (\partial_{\mathbf{p}} \mathcal{I})^{\mathsf{T}}$. The 2D velocity field ν, the optical flow, can be regarded as the 2D projection of the 3D scene flow, i.e. the temporal derivative of $\mathbf{W}$ on the points of a 3D surface visible to the camera.



(a) Occlusions. Left: Frame 1. Right: Frame 2



(b) Aperture problem

Figure 2.4.: Challenges in optical flow estimation

The problem of estimating the optical flow field from two or more images has been studied extensively for more than 30 years. This is rooted in three major challenges, giving rise to a variety of other problems.

Firstly, the correct optical flow field stemming from the scene flow of a 3D surface is generally not continuous, due to occlusions. An example is shown in Figure 2.4a, where an upward-moving camera and a discontinuous surface cause the cyan regions to be occluded in frame 2, while the magenta ones appear without any correspondence in frame 1. This directly violates the photoconsistency assumption in Equation (2.19).

Secondly, the photoconsistency assumption is not sufficient to determine the motion. As we have two unknowns but only one equation for every point in Equation (2.23), we can only determine the velocity magnitude in direction of the image gradient, i.e. the normal flow

$$\nu(\mathbf{p}, t) = \frac{\partial_t \mathcal{I}}{|\nabla \mathcal{I}|^2}. \tag{2.24}$$

This is commonly known as the aperture problem, which is illustrated in Figure 2.4b. In the left image the only motion observable through the aperture is the horizontal component parallel to the image gradient, while the true motion includes a vertical component as well. The right image shows the case where $\nabla \mathcal{I} = 0$, and no motion at all can be determined. In this case, the normal flow is undefined. Using vector-valued images such as color images can help overcome this problem, if the gradients in the color channels do not coincide, but can also render the resulting equation system overdetermined.

The third major challenge in determining the optical flow is that digital images are neither continuous functions, nor are they captured in continuous streams, but as frames in a discrete image sequence with a measurable time between them. Therefore, for two images taken at time t_0 and t_1, we actually want to estimate a displacement field

$$\mathbf{v}(\mathbf{p}, t_0) = \int\limits_0^{t_0} \nu(\mathbf{p}, t)\, \mathrm{d}t. \tag{2.25}$$

Plugging this into Equation (2.19) and assuming that $\mathbf{v}(t_0) \equiv 0$, we get the standard optical flow constraint formulated on displacement fields:

$$\mathcal{I}(\mathbf{p} + \mathbf{v}(\mathbf{p}, t_1), t_1) - \mathcal{I}(\mathbf{p}, t_0) = 0. \tag{2.26}$$

In contrast to Equation (2.23), this formulation can be evaluated on discrete image sequences. We use the common notation $\mathcal{I}_k(\mathbf{p}) := \mathcal{I}(\mathbf{p}, t_k)$ for the k-th image, and in the case of only two images, we simply write $\mathbf{v}(\mathbf{p})$ for $\mathbf{v}(\mathbf{p}, t_1)$.

Depending on the parametrization of the displacement, the formulation can be ill-defined. If we want to estimate the underlying rigid motion of a known 3D surface, as we will describe in Chapter 4, the combination of Equation (2.26) for all pixels in the image can yield a well- or over-determined system of equations, depending on the surface structure and surface texture. If we want to estimate an independent displacement for every pixel, i.e.

$$\mathbf{v}^*(\mathbf{p}) = \arg\min_{\mathbf{v}} \left\{ |\mathcal{I}_1(\mathbf{p} + \mathbf{v}) - \mathcal{I}_0(\mathbf{p})| \right\}, \tag{2.27}$$

there can be many possible displacement vectors fulfilling Equation (2.26), or none at all, due to noise or occlusions, just as in the case of the rectified stereo correspondence search in Equation (2.15). Similarly to this case, the two standard strategies to overcome this ambiguity are the use of advanced photoconsistency assumptions and regularity priors.

2.5. Advanced Photoconsistency Assumptions

A common assumption to overcome the inherent ambiguity of finding the disparity value or optical flow vector optimizing photoconsistency in a pixel $\mathbf{p}$ is that not only the intensity value $\mathcal{I}_0(\mathbf{p})$ in the first image should be consistent with the corresponding value $\mathcal{I}_1(\mathbf{p}+d(\mathbf{p}))$ or $\mathcal{I}_1(\mathbf{p}+\mathbf{v}(\mathbf{p}))$ in the second image, but that the disparity or optical flow in $\mathbf{p}$ should also match all pixels $\mathbf{p}'$ in a local patch $\mathcal{N}_{\mathbf{p}}$ centered at $\mathbf{p}$:

$$\begin{aligned} \forall \mathbf{p}' \in \mathcal{N}(\mathbf{p}) \quad &: \\ 0 \quad &= \quad \mathcal{I}_1(x' + d(\mathbf{p}), y') - \mathcal{I}_0(\mathbf{p}') && (2.28) \\ 0 \quad &= \quad \mathcal{I}_1(\mathbf{p}' + \mathbf{v}(\mathbf{p})) - \mathcal{I}_0(\mathbf{p}') && (2.29) \end{aligned}$$

This patch-based photoconsistency constraint only holds for surfaces parallel to the image plane and translational motions with a noiseless image acquisition and without occlusions. Nonetheless it has proven a good approximation for many real-world scenarios. Finding a disparity or optical flow estimate optimizing this patch-based photoconsistency assumption can be regarded as a local energy model, with a weighted contribution to the energy of each pixel in the patch. The general weighted sum energy formulation for a pixel $\mathbf{p}$ reads as follows:

$$E_{\text{Patch,Stereo}}(d,\mathbf{p}) = \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{p}') \left|\mathcal{I}_1(x'+d,y') - \mathcal{I}_0(\mathbf{p}')\right|^{\alpha_{\mathcal{N}}} \,\mathrm{d}\mathbf{p}' \tag{2.30a}$$

$$E_{\text{Patch,Flow}}(\mathbf{v},\mathbf{p}) = \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{p}') \left|\mathcal{I}_1(\mathbf{p}'+\mathbf{v}) - \mathcal{I}_0(\mathbf{p}')\right|^{\alpha_{\mathcal{N}}} \,\mathrm{d}\mathbf{p}' \tag{2.30b}$$

The exponent $\alpha_{\mathcal{N}}$ is typically chosen as either 1 or 2, yielding a sum of absolute differences (SAD) or a sum of squared differences (SSD). The non-negative weight function w controls the influence of each pixel depending on its distance to the center pixel. Typical choices are uniform weighting $w \equiv \frac{1}{\|\mathcal{N}(\mathbf{p})\|}$ or a truncated Gaussian weighting

$$w_\sigma(\mathbf{p},\mathbf{p}') = \exp\left(-\frac{\|\mathbf{p}-\mathbf{p}'\|^2}{2\sigma^2}\right) \tag{2.31}$$

The reconstructed disparity value or optical flow vector for pixel $\mathbf{p}$ is then computed as the minimizer of the local energy

$$d^*(\mathbf{p}) = \arg\min_d \left\{E_{\text{Patch,Stereo}}(d,\mathbf{p})\right\}, \tag{2.32a}$$

$$\mathbf{v}^*(\mathbf{p}) = \arg\min_{\mathbf{v}} \left\{E_{\text{Patch,Flow}}(\mathbf{v},\mathbf{p})\right\} \tag{2.32b}$$

Normalizing the energy measures of (2.30) with the norm of the patch

$$\|w\| = \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{p}') \tag{2.33}$$

does not change the minimizing argument of (2.32). However, combining patch-based energies with different patch sizes can merit a normalization.

For the rest of this section, we only list the formulations for optical flow estimation, as the formulations for disparity estimation in the stereo setting are analogous.

A desired property of a photoconsistency assumption is invariance against changes in illumination, which the discussed image intensity constancy assumption does not account for. We can differentiate between two kinds of illumination changes: local and global. Local illumination changes can appear if the Lambertian surface model is violated, while global illumination changes are usually related to different camera calibrations or, in the case of dynamic motion, changes in the light sources of the observed scene. In the context of patch-based photoconsistency assumptions, an invariance to additive illumination changes can be achieved by subtracting in each patch an additive offset $\bar{\mathcal{I}}_1(\mathbf{p})$ and $\bar{\mathcal{I}}_0(\mathbf{p})$:

$$E_{\text{Diff-Patch}}(\mathbf{v},\mathbf{p}) = \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{p}') \left|\mathcal{I}_1(\mathbf{p}'+\mathbf{v}) - \bar{\mathcal{I}}_1(\mathbf{p}+\mathbf{v}) - \left(\mathcal{I}_0(\mathbf{p}') - \bar{\mathcal{I}}_0(\mathbf{p})\right)\right|^{\alpha_{\mathcal{N}}} \,\mathrm{d}\mathbf{p}'. \tag{2.34}$$

Typical choices for the offsets $\bar{\mathcal{I}}_1(\mathbf{p})$ and $\bar{\mathcal{I}}_0(\mathbf{p})$ are either the weighted average intensity values of the patch

$$\bar{\mathcal{I}}(\mathbf{p}) = \frac{\int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{p}')\mathcal{I}(\mathbf{p}')\,\mathrm{d}\mathbf{p}'}{\int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{p}')\,\mathrm{d}\mathbf{p}'} \tag{2.35}$$

or the center values ($\bar{\mathcal{I}}(\mathbf{p}) = \mathcal{I}(\mathbf{p})$). Other photoconsistency assumptions invariant to additive illumination changes are the constancy of derivative measures such as the image gradient or the image gradient magnitude

$$E_{\text{Grad}}(\mathbf{v},\mathbf{p}) = |\nabla\mathcal{I}_1(\mathbf{p}+\mathbf{v}) - \nabla\mathcal{I}_0(\mathbf{p})| \tag{2.36a}$$
$$E_{\text{Grad-Mag}}(\mathbf{v},\mathbf{p}) = ||\nabla\mathcal{I}_1(\mathbf{p}+\mathbf{v})| - |\nabla\mathcal{I}_0(\mathbf{p})||\,. \tag{2.36b}$$

Examples of photoconsistency assumptions invariant to multiplicative changes in illumination are the normalized cross-correlation (NCC)

$$E_{\text{NCC}}(\mathbf{v},\mathbf{p}) = 1 - \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} \frac{\left(\mathcal{I}_1(\mathbf{p}'+\mathbf{v}) - \bar{\mathcal{I}}_1(\mathbf{p}+\mathbf{v})\right)}{\text{Norm}_1(\mathbf{p}+\mathbf{v})} \frac{\left(\mathcal{I}_0(\mathbf{p}') - \bar{\mathcal{I}}_1(\mathbf{p})\right)}{\text{Norm}_0(\mathbf{p})}\,\mathrm{d}\mathbf{p}', \tag{2.37}$$

the census transform

$$E_{\text{Census}}(\mathbf{v},\mathbf{p}) = \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} \left|\left[\mathcal{I}_1(\mathbf{p}'+\mathbf{v}) > \mathcal{I}_1(\mathbf{p}+\mathbf{v})\right] - \left[\mathcal{I}_0(\mathbf{p}') > \mathcal{I}_1(\mathbf{p})\right]\right|\,\mathrm{d}\mathbf{p}', \tag{2.38}$$

or the rank transform

$$E_{\text{Rank}}(\mathbf{v},\mathbf{p}) = \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} \left[\mathcal{I}_1(\mathbf{p}'+\mathbf{v}) > \mathcal{I}_1(\mathbf{p}+\mathbf{v})\right]\,\mathrm{d}\mathbf{p}' - \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} \left[\mathcal{I}_0(\mathbf{p}') > \mathcal{I}_1(\mathbf{p})\right]\,\mathrm{d}\mathbf{p}', \tag{2.39}$$

For the NCC definition $\bar{\mathcal{I}}$ denotes the average as in (2.35) and $\text{Norm}_i(\mathbf{p})$ is the Euclidean norm of differences to the average

$$\text{Norm}_i(\mathbf{p}) = \sqrt{\int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} \left(\mathcal{I}_i(\mathbf{p}') - \bar{\mathcal{I}}_i(\mathbf{p})\right)^2\,\mathrm{d}\mathbf{p}'}. \tag{2.40}$$

Which photoconsistency assumption proves most accurate is depends on the data. Whenever a filter is invariant to a certain attribute, it might be just that attribute that proves useful on some data sequences, while it is a disturbance value on others. For example, for locally constant image regions all the photoconsistency assumptions mentioned above yield highly ambiguous results. Other factors determining the usefulness of these photoconsistency assumptions are computational stability and efficiency.

The NCC has a higher descriptive power than the census or rank transforms, because the binary terms of the latter neglect any absolute scale of the image patch. However, the NCC becomes unstable for constant image patches where the denominator tends to zero, and it comes at a higher computational cost than the census or rank transform. In Chapter 3.5 we perform a thorough evaluation of these photoconsistency assumptions in the context of dynamic motion estimation with energy models.

2.6. Energy Models

For many correspondence problems in computer vision, energy models have become the method of choice. They combine several possibly conflicting assumptions for an optimal solution and allow to abstract the solution from the algorithm used to compute it. While Equations (2.15) and (2.27) already represent energies, the incorporation of additional assumptions can help overcome their inherent ambiguity. For an overview of energy models for optical flow estimation we refer to Section 3.2.

A very popular assumption for an optimal solution is spatial regularity. In the context of optical flow, this means that neighboring pixels should move in a similar direction with a similar speed. In the context of disparity estimation, it means that neighboring pixels should have a similar disparity and depth. This assumption is combined with the photoconsistency assumption by means of two soft constraints usually referred to as the data term and regularity term. The data term penalizes the sum of the absolute distance to the photoconsistency assumption for the stereo and optical flow case in every pixel:

$$D_{\text{stereo}}(d) = \int_{\Omega} \phi_D \left(|\mathcal{I}_1(x + d(\mathbf{p}), y) - \mathcal{I}_0(\mathbf{p})|^2 \right) \mathrm{d}\mathbf{p} \tag{2.41a}$$

$$\begin{aligned} D_{\text{flow}}(\mathbf{v}) &= \int_{\Omega} \phi_D \left(|\mathcal{I}_1(x + v_1(\mathbf{p}), y + v_2(\mathbf{p})) - \mathcal{I}_0(\mathbf{p})|^2 \right) \mathrm{d}\mathbf{p} \\ &= \int_{\Omega} \phi_D \left(|\mathcal{I}_1(\mathbf{p} + \mathbf{v}(\mathbf{p})) - \mathcal{I}_0(\mathbf{p})|^2 \right) \mathrm{d}\mathbf{p} \end{aligned} \tag{2.41b}$$

The first-order regularity term penalizes the magnitude of the derivative of the disparity or optical flow field in every pixel:

$$R_{\text{stereo}}(d) = \int_{\Omega} \phi_R \left(|\nabla d(\mathbf{p})|_{\beta}^2 \right) \mathrm{d}\mathbf{p} \tag{2.42a}$$

$$R_{\text{flow}}(\mathbf{v}) = \int_{\Omega} \phi_R \left(|J_{\mathbf{v}}(\mathbf{p})|_{\beta,\gamma}^2 \right) \mathrm{d}\mathbf{p} \tag{2.42b}$$

We use the parameter β in our notation to define the norm with respect to columns of the Jacobian matrix J, and the parameter γ to define the norm with respect to its rows:

$$|\nabla d|_\beta = \begin{cases} |\partial_x d| + |\partial_y d| & \beta = 1 \qquad (2.43) \\ \sqrt{(\partial_x d)^2 + (\partial_y d)^2} & \beta = 2 \qquad (2.44) \end{cases}$$

$$|J_\mathbf{v}|_{\beta,\gamma} = \begin{cases} |\partial_x v_1| + |\partial_y v_1| + |\partial_x v_2| + |\partial_y v_2| & \beta = 1, \quad \gamma = 1 \qquad (2.45) \\ \sqrt{(\partial_x v_1)^2 + (\partial_y v_1)^2} + \sqrt{(\partial_x v_2)^2 + (\partial_y v_2)^2} & \beta = 2, \quad \gamma = 1 \qquad (2.46) \\ \sqrt{(\partial_x v_1)^2 + (\partial_x v_2)^2} + \sqrt{(\partial_y v_1)^2 + (\partial_y v_2)^2} & \beta = 1, \quad \gamma = 2 \qquad (2.47) \\ \sqrt{(\partial_x v_1)^2 + (\partial_y v_1)^2 + (\partial_x v_2)^2 + (\partial_y v_2)^2} & \beta = 2, \quad \gamma = 2 \qquad (2.48) \end{cases}$$

The non-decreasing loss functions ϕ_D and ϕ_R in (2.41) and (2.42) control the influence of each pixel to the global energy. The most widely used loss functions correspond to quadratic penalization ($\phi(s) = s$) and penalization of absolute values ($\phi(s) = \sqrt{s+\epsilon}$), where the ϵ-term is added to provide differentiability, depending on the method for minimization. The energy equations (2.15) and (2.27) with a first order regularity prior read as follows:

$$E_{\text{stereo}}(d) = \int_\Omega \phi_D\left(|\mathcal{I}_1(x + d(\mathbf{p}), y) - \mathcal{I}_0(\mathbf{p})|^2\right) \,\mathrm{d}\mathbf{p} + \lambda \int_\Omega \phi_R\left(|\nabla d(\mathbf{p})|^2_\beta\right) \,\mathrm{d}\mathbf{p} \qquad (2.49a)$$

for the stereo case, and

$$E_{\text{flow}}(\mathbf{v}) = \int_\Omega \phi_D\left(|\mathcal{I}_1(x + v_1(\mathbf{p}), y + v_2(\mathbf{p})) - \mathcal{I}_0(\mathbf{p})|^2\right) \,\mathrm{d}\mathbf{p} + \lambda \int_\Omega \phi_R\left(|J_\mathbf{v}(\mathbf{p})|^2_{\beta,\gamma}\right) \,\mathrm{d}\mathbf{p} \qquad (2.49b)$$

for the optical flow case. In the case of monomial loss functions, data and regularity terms can be written with the exponents α_D and α_R as in the expressions below.

$$E_{\text{stereo}}(d) = \int_\Omega |\mathcal{I}_1(x + d(\mathbf{p}), y) - \mathcal{I}_0(\mathbf{p})|^{\alpha_D} \,\mathrm{d}\mathbf{p} + \lambda \int_\Omega |\nabla d(\mathbf{p})|^{\alpha_R}_\beta \,\mathrm{d}\mathbf{p} \qquad (2.50a)$$

$$E_{\text{flow}}(\mathbf{v}) = \int_\Omega |\mathcal{I}_1(x + v_1(\mathbf{p}), y + v_2(\mathbf{p})) - \mathcal{I}_0(\mathbf{p})|^{\alpha_D} \,\mathrm{d}\mathbf{p} + \lambda \int_\Omega |J_\mathbf{v}(\mathbf{p})|^{\alpha_R}_{\beta,\gamma} \,\mathrm{d}\mathbf{p} \qquad (2.50b)$$

The minimizing arguments h^* and $\mathbf{v}^*$ are disparity or flow fields, that adhere to both the photoconsistency assumption as well as to a regularity assumption. The scalar weight parameter λ weights the regularity assumption against the photoconsistency assumption. Larger values of λ result in smoother minimizing disparity or flow fields. The exponents α_D and α_R determine the extent to which outliers affect the solution. The larger α_D is, the more will noisy pixels affect the result. The larger α_R is, the more will the resulting field be blurred around strong edges, as one large irregularity in the field will yield a larger cost than several smaller ones.

The above formulations yield a regularity in both x and y direction of the 2-dimensional image domain Ω. While we will focus on two dimensional image domains in this thesis, stemming from the pinhole camera model described in the last section, the energy model can also be applied in different ways. For example, magnetic resonance tomography provides volumetric 3D images. If the patient moves between several scans, the optical flow energy (2.50b) can be reformulated for a 3D domain to estimate a 3D motion of the volume [2]. In the 2D domain, we can consider a formulation in which the regularity term is not formulated fully on all dimensions of $\Omega \in \mathbb{R}^n$, but only on a discrete subset of directions $\mathcal{P} \subset \mathcal{S}^{n-1}$,

$$E_{\text{reg-1D}} = \sum_{\mathbf{n}\in\mathcal{P}} \int_{\Omega} |J(\mathbf{p})\mathbf{n}|_{\beta}^{\alpha_R} \, \mathrm{d}\mathbf{p}, \tag{2.51}$$

where J is either the $2{\times}2$ Jacobian $J_{\mathbf{v}}$ as in the optical flow equation, or the transposed gradient $(\nabla d)^{\mathsf{T}}$ as in the stereo equation. This formulation allows to find a minimizing argument much easier than the original problems do, as we will see in the next section.

A combined problem of the two Equations (2.50) is the 2.5-dimensional scene flow problem of estimating the 3D motion of points on a surface. In contrast to 3D shape matching, this surface is parametrized as a 2D disparity image d that is either given or has the be estimated as well. Given an image $\mathcal{I}_0(t_0)$ and two images $\mathcal{I}_0(t_1)$ and $\mathcal{I}_1(t_1)$, the scene flow can be represented by a disparity field d and an optical flow field $\mathbf{v}$ that match the three input images:

$$\forall \mathbf{p}: \quad \mathcal{I}_0(x+v_1(\mathbf{p}), y+v_2(\mathbf{p}), t_1) = \mathcal{I}_1(x+v_1(\mathbf{p})+d(\mathbf{p}), y+v_2(\mathbf{p}), t_1) = \mathcal{I}_0(x, y, t_0). \tag{2.52}$$

Wedel et al. [89] use a disparity field d_0, matching $\mathcal{I}_0(t_0)$ to the image $\mathcal{I}_1(t_0)$ of the same binocular stereo system, and parametrize d by its difference to d_0,

$$d = d' + d_0, \tag{2.53}$$

in a joint energy formulation:

$$
\begin{aligned}
& E_{\mathrm{sf}}(d', \mathbf{v}) \\
= & \int_\Omega \phi\left(\left|\mathcal{I}_0(x + v_1(\mathbf{p}), y + v_2(\mathbf{p}), t_1) - \mathcal{I}_0(\mathbf{p}, t_0)\right|^2\right) \mathrm{d}\mathbf{p} \\
+ & \int_\Omega \phi\left(\left|\mathcal{I}_1(x_d + d'(\mathbf{p}) + v_1(\mathbf{p}), y + v_2(\mathbf{p}), t_1) - \mathcal{I}_1(x_d, y, t_0)\right|^2\right) \mathrm{d}\mathbf{p} \\
+ & \int_\Omega \phi\left(\left|\mathcal{I}_1(x_d + d'(\mathbf{p}) + v_1(\mathbf{p}), y + v_2(\mathbf{p}), t_1) - \mathcal{I}_0(x + v_1(\mathbf{p}), y + v_2(\mathbf{p}), t_1)\right|^2\right) \mathrm{d}\mathbf{p} \\
+ & \int_\Omega \phi\left(\lambda \left|J_{\mathbf{v}}(\mathbf{p})\right|_{2,2}^2 + \gamma \left|\nabla d'(\mathbf{p})\right|_\beta^{\alpha_R}\right) \mathrm{d}\mathbf{p},
\end{aligned}
\tag{2.54}
$$

with $x_d = x + d(\mathbf{p})$ and $\phi(s) = s$. Using $\mathcal{I}_1(x_d, y, t_0)$ instead of $\mathcal{I}_0(\mathbf{p}, t_0)$ in the second term accounts for potential errors in the original disparity estimate d.

The parametrization of the disparity field by d' instead of d in the energy formulation enforces spatial regularity of the change in disparity, which is a representation of the motion in Z-direction, not of the disparity itself, which is a representation of the surface.

In the case of binocular stereo, the first order regularity term favors surfaces parallel to the image plane. In the case of both optical flow and scene flow, it favors motions that appear rigid under the camera projection. Although this represents only a small subset of all possible surfaces and motions, the first order regularity term has been employed in a large number of energy-based approaches addressing the correspondence problems introduced in this thesis, mainly due to its simplicity. Before we give an overview of those approaches in Chapter 3, we want to demonstrate that even this relatively simple regularity term can impose significant challenges with respect to the runtime complexity of algorithms for minimizing the energies (2.50a) and (2.50b).

2.6.1. Discrete Formulations and Runtime Complexity

As all digital computers work on discrete values, any notion of computational feasibility is referring to the number of elementary compute operations an algorithm is spending on an input of integer length. In the context of finding a continuous function minimizing a given functional, the input length is a combination of the sampling resolution and the bit depth used for quantization. In computer science, “feasible time” generally refers to an asymptotically polynomial runtime complexity. For many problems in computer vision, a polynomial runtime complexity does not guarantee practicality, as the large number of input variables usually renders any method with a quadratic or higher runtime complexity impractical. However, for the rest of this section, we concern ourselves with the distinction between problems of polynomial and non-polynomial runtime complexity.

To compute a minimizer of the functional, we can either discretize the functional itself and then formulate an algorithm to minimize the discrete energy, or we can formulate an optimization strategy in the continuous domain, for example gradient descent, and afterwards discretizes this strategy. In this section, we are making statements about the bounds in runtime complexity for any algorithm minimizing a particular energy, therefore we discretize the energy functional itself. The discretized versions of Equations (2.50a) and (2.50b), on a discrete image domain $\Omega = \{0, ..., w-1\} \times \{0, ..., h-1\}$, and with a forward-difference approximation of the spatial derivatives, can be formulated as follows:

$$
\begin{aligned}
E_{\text{Stereo}}(d) &= \sum_{x=0}^{w-1}\sum_{y=0}^{h-1}\Bigg(|\mathcal{I}_1(x+d(x,y),y) - \mathcal{I}_0(x,y)|^{\alpha_D} && (2.55a)\\
&+ \lambda_F \big(|d(x+1,y) - d(x,y)|^{\beta}\,[x<w-1] \\
&+ \quad |d(x,y+1) - d(x,y)|^{\beta}\,[y<h-1]\big)^{\frac{\alpha_R}{\beta}}\Bigg) \\
E_{\text{Flow}}(\mathbf{v}) &= \sum_{x=0}^{w-1}\sum_{y=0}^{h-1}\Bigg(|\mathcal{I}_1(x+v_1(x,y),y+v_2(x,y)) - \mathcal{I}_0(x,y)|^{\alpha_D} && (2.55b)\\
&+ \lambda_F\Bigg(\Big(|v_1(x+1,y) - v_1(x,y)|^{\beta} + |v_2(x+1,y) - v_2(x,y)|^{\beta}\Big)\,[x<w-1] \\
&+ \quad \Big(|v_1(x,y+1) - v_1(x,y)|^{\beta} + |v_2(x,y+1) - v_2(x,y)|^{\beta}\Big)\,[y<h-1]\Bigg)^{\frac{\alpha_R}{\beta}}\Bigg)
\end{aligned}
$$

The boundary treatment of both formulations implies that $\forall (x,y) \in \Omega : d(w,y) = d(w-1,y) \wedge d(x,h) = d(x,h-1)$. The same applies to v_1 and v_2. This is the discrete version of the Neumann boundary conditions

$$\forall \mathbf{p} \in \partial\Omega : \langle \nabla d(\mathbf{p}), \mathbf{n}(\mathbf{p}) \rangle = 0 \tag{2.56}$$

where $\mathbf{n}$ is the normal pointing out of the image on the image boundary $\partial\Omega$. Since these boundary conditions are necessarily fulfilled by any minimizer as part of the Euler-Lagrange equations, assuming this in the energy formulation instead of setting an explicit boundary value constitutes the smallest bias with respect to the minimizer. Note that these formulations are the discrete versions of (2.43) and (2.45) for $\beta = 1$, and of (2.44) and (2.48) for $\beta = 2$. The discrete versions of (2.47) and (2.46) are analogous. Additionally, one can also use central or backward differences for the approximations of the derivatives.

Depending on the given input data, the choice of regularity formulations and parameter values discussed in the last section can have a huge impact in terms of the ability to find a minimizing argument for the energy in feasible time. In the following, we will distinguish different cases under which the problem of estimating a minimizer either becomes feasible or hard.

Without the regularity term ($\lambda = 0$), both energies (2.50a) and (2.50b) can be trivially minimized in feasible time. As mentioned before, the problem with a sole data term is not the computational effort, but the fact that a minimizer is not unique. Naturally, this problem does not vanish with a regularity term. A default degenerate example for all first and higher order regularity terms are two completely constant images $\mathcal{I}_0 \equiv \mathcal{I}_1 \equiv c$. An efficient minimization of the data term can be performed by a complete search for the best correspondence for every pixel independently.

The regularity terms by themselves can be trivially minimized as well, since all constant functions are minimizers. While the minimizers of the regularity terms are not well defined either, for exponents $\alpha_R \geq 1$, the regularity terms in (2.50a) and (2.50b) are convex:

$$\forall f_1, f_2 \in \mathcal{F}, \forall \lambda \in [0,1]: \; R\left(\lambda f_1 + (1-\lambda) f_2\right) \leq \lambda R\left(f_1\right) + (1-\lambda) R\left(f_2\right). \tag{2.57}$$

This implies that all sublevel sets $\mathcal{S}_\alpha(R) = \{f | R(f) \leq \alpha\}$, and in particular the set of minimizers, are convex sets:

$$\forall f_1, f_2 \in \mathcal{S}_\alpha(R), \forall \lambda \in [0,1]: \lambda f_1 + (1-\lambda) f_2 \in \mathcal{S}_\alpha(R). \tag{2.58}$$

Besides having a convex set of minimizers, convex functionals defined on convex sets can be feasibly minimized by various algorithms for convex optimization, such as gradient descent, finding a solution of the Euler-Lagrange equations, the FISTA-algorithm of Beck and Teboulle [16], or the first-order primal-dual algorithm of Chambolle and Pock [26]. Moreover, convex functionals are closed under linear combination with positive coefficients:

$$\forall E_1, E_2 \text{ convex}, \lambda_1, \lambda_2 \in \mathbb{R}_{\geq 0}: \quad \lambda_1 E_1 + \lambda_2 E_2 \text{ convex} \tag{2.59}$$

Therefore, the combination of a convex regularity term with a convex data term again yields a convex energy functional. However, the data terms are highly non-convex, since the flow fields are index fields for arbitrary images. For exponents $\alpha_R < 1$, the regularity terms are non-convex as well, and currently no general solutions for finding the global optimum of energies with non-convex regularity-terms are known, if the dimensionality of Ω is at least 2. For $\alpha_R \to 0$ the regularity term becomes the Potts prior

$$E_{\text{Potts}}(v) = \int\limits_{\Omega} \left[\,|\nabla d(\mathbf{p})| \neq 0\right] \mathrm{d}\mathbf{p}. \tag{2.60}$$

With this prior, the discrete stereo energy (2.55a) becomes NP-hard. The discrete optical flow energy (2.55b) is NP-hard even for convex regularity terms.

In the following, we will sketch an NP-hardness proof for the optical flow energy with $\alpha_D = 1$, $\alpha_R = 1$ and $\beta = 1$. To this end, we will reduce the NP-hard graph labeling problem with three labels and Potts prior to the discrete optical flow problem.

The graph labeling problem with Potts prior is given as follows: Given a graph $\mathcal{G} = \left(\mathcal{V}, \mathcal{E} \subseteq \begin{bmatrix}\mathcal{V}\\2\end{bmatrix}\right)$ and a discrete set of labels $\mathcal{L}$, does a labeling function $f : \mathcal{V} \to \mathcal{L}$ exist, for which the energy

$$E_{\text{Potts}} = \sum_{v \in \mathcal{V}} \rho(v, f(v)) + \sum_{\{v,v'\} \in \mathcal{E}} \lambda_P \delta\left(f(v) \neq f(v')\right) \tag{2.61}$$

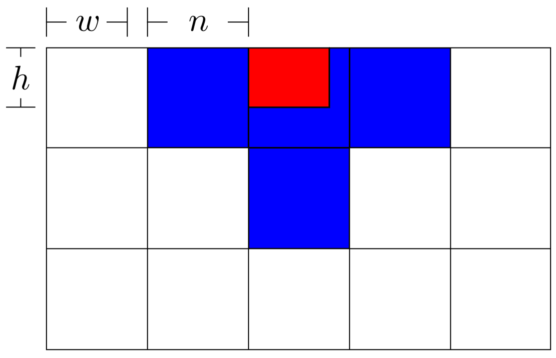
is smaller than a constant $C_P \in \mathbb{R}_{>0}$ Here, the function $\rho : \mathcal{V} \times \mathcal{L} \to \mathbb{R}_{\geq 0}$ is an arbitrary unary cost function. Veksler [87] has sketched a proof that this problem is NP-hard on planar grids, by constructing a polynomial time reduction of the NP-hard multiway cut problem to this problem. Since this reduction does not use more labels in the labeling problem than there are terminal vertices in the multiway cut problem, and the multiway cut problem is NP-hard for three labels, we can assume the labeling problem with Potts prior to be NP-hard on a planar grid with three labels.

If the graph labeling problem with Potts prior is NP-hard for a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ with $\mathcal{V} = \{0, ..., w-1\} \times \{0, ..., h-1\}$ and $\mathcal{E} = \{\{(x,y),(x+1,y)\} |\ x < w-1\} \cup \{\{(x,y),(x,y+1)\} |\ y < h-1\}$, then the decision problem of whether and argument $\mathbf{v} = \begin{bmatrix} v_1 & v_2 \end{bmatrix}^\mathsf{T} : \{0, ..., w-1\} \times \{0, ..., h-1\} \to \mathbb{Z}^2$ exists, for which for the discrete optical flow energy (2.55b) with $\alpha_R = 1$ and $\beta = 1$ is below a constant C_F is NP-hard as well. The main idea for our proof is to embed the three labels of the graph labeling problem in the 2D label space of $\begin{bmatrix} v_1 & v_2 \end{bmatrix}^\mathsf{T}$ of the optical flow problem. For a grid of size $w \times h$ we construct two images $\mathcal{I}_0$ and $\mathcal{I}_1$of size $5n \times 3n$, where $n := \max\{w, h\}$. The pixels of the top middle area of $\mathcal{I}_0$ will be matched to either a pixel in the top left, top right, or bottom middle area. At first, $\mathcal{I}_0$ is 0 everywhere, and $\mathcal{I}_1$ contains images of the three labels in those areas:

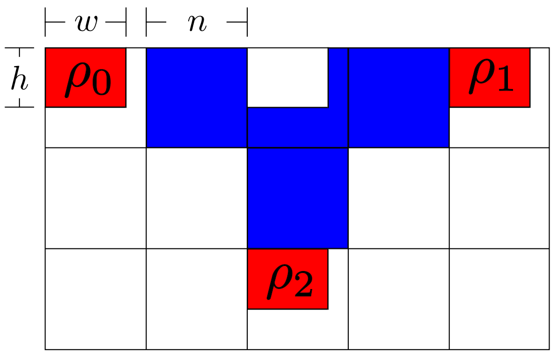
$$\begin{aligned} \mathcal{I}_1(x,y) \leftarrow \rho((x,y),0) \quad &: \quad (x,y) \in \{0, ..., w{-}1\} {\times} \{0, ..., h{-}1\} \\ \mathcal{I}_1(x,y) \leftarrow \rho((x,y),1) \quad &: \quad (x,y) \in \{4n, ..., 4n{+}w{-}1\} {\times} \{0, ..., h{-}1\} \\ \mathcal{I}_1(x,y) \leftarrow \rho((x,y),2) \quad &: \quad (x,y) \in \{2n, ..., 2n{+}w{-}1\} {\times} \{2n, ..., 2n{+}h{-}1\} \\ \mathcal{I}_1(x,y) \leftarrow 0 \quad &: \quad \text{else} \end{aligned} \tag{2.62}$$

Optical flow vectors of the form $\begin{bmatrix} -2n & 0 \end{bmatrix}^\mathsf{T}$, $\begin{bmatrix} 2n & 0 \end{bmatrix}^\mathsf{T}$, or $\begin{bmatrix} 0 & 2n \end{bmatrix}^\mathsf{T}$ of pixels in the domain $\{2n, ..., 2n{+}w{-}1\} {\times} \{0, ..., h{-}1\}$ in $\mathcal{I}_0$ now add the unary label costs of the graph labeling problem to the energy of an optimal solution of the optical flow problem. We set $\lambda_F := \frac{\lambda_P}{2n}$ and the regularity prior adds the binary label costs. To ensure that optical flow vectors from $\mathcal{I}_0$ to $\mathcal{I}_1$ in the pixel domain $\{2n, ..., 2n{+}w{-}1\} {\times} \{0, ..., h{-}1\}$ only have the form $\begin{bmatrix} -n & 0 \end{bmatrix}^\mathsf{T}$, $\begin{bmatrix} n & 0 \end{bmatrix}^\mathsf{T}$, or $\begin{bmatrix} 0 & n \end{bmatrix}^\mathsf{T}$, we add the double maximal cost

$$E_{\max} \leftarrow 2|\mathcal{V}| \max_{v,l} \{\rho(v,l)\} + |\mathcal{E}|\lambda_P \tag{2.63}$$



(a) $\mathcal{I}_0$



(b) $\mathcal{I}_1$

Figure 2.5.: Areas of the two images used for the reduction of the NP-hard graph labeling problem with Potts prior to the optical flow problem.

of the problem in intervals onto every pixel, to ensure that matches adding an energy less than $\frac{1}{2}E_{\max}$ to E_{Flow} correspond to flow vectors as described above:

$$\begin{aligned}
\forall(x,y) \in \{0,...,w-1\}\times\{0,...,h-1\} \quad &: \\
E_{\text{int}} &\leftarrow E_{\max}(yw+x) \\
\mathcal{I}_0(2n+x,y) &\leftarrow \mathcal{I}_0(2n+x,y) + E_{\text{int}}, \\
\mathcal{I}_1(x,y) &\leftarrow \mathcal{I}_1(x,y) + E_{\text{int}}, \\
\mathcal{I}_1(4n+x,y) &\leftarrow \mathcal{I}_1(4n+x,y) + E_{\text{int}}, \\
\mathcal{I}_1(2n+x,2n+y) &\leftarrow \mathcal{I}_1(2n+x,2n+y) + E_{\text{int}} \qquad (2.64)
\end{aligned}$$

In the same manner we force the flow vectors of all pixels of the boundary of domain $\{2n,...,2n+w-1\}\times\{0,...,h-1\}$ to be $\begin{bmatrix}0 & 0\end{bmatrix}^\mathsf{T}$. We enumerate them, starting with hw, and add the product of $E_{\max}$ and the pixel number to $\mathcal{I}_0$ and $\mathcal{I}_1$ in that pixel. This creates an additional cost of $2n\lambda_F$ for every boundary pixel, and the sum can be added to $E_{\max}$ as well. All remaining pixels in $\mathcal{I}_0$ and $\mathcal{I}_1$ that have not been enumerated yet are independent of the given instance of $\mathcal{G}$, apart from the given grid dimensions. To ensure that the remaining pixels in $\mathcal{I}_0$ do not match to pixels depending on the instance in $\mathcal{I}_1$ and that remaining pixels in $\mathcal{I}_1$ are not being matched to by dependent pixels in $\mathcal{I}_0$, we give them all the same value of a multiple of $E_{\max}$ that no other pixel has been given yet. We apply the same value to all regions outside of the image domain.

Figure 2.5 illustrates the two images used in the proof. Pixels in the red area of image $\mathcal{I}_0$ can be matched to one of the pixels in each red areas of image $\mathcal{I}_1$. Pixels in the blue area are matched onto themselves. This is actually only necessary for a one pixel boundary around the red area in image $\mathcal{I}_0$. Pixels in the white area of $\mathcal{I}_0$ can be matched anywhere in the white area of $\mathcal{I}_1$ and are equal for all instances of the problem. Finding an optimal solution for the pixels in the white area of $\mathcal{I}_0$ is possible in polynomial time, since the energy for them is convex and only depends on the boundary values. We can put the optimal energy into the constant C_F together with the energy for the boundary of the red and blue area in $\mathcal{I}_0$ and have the equivalence of the two problems. We should note that the optical flow problem of Equation (2.55b) with $\alpha_D = 1$, $\alpha_R = 1$, and $\beta = 1$ is just one instance in the large family of variational optical flow formulations. We also do not allow the flow vectors to point at sub-pixel positions, and we use forward differences for the regularity term, while many discrete formulations use central differences. Therefore, there might exist efficient algorithms for other instances of these problems. However, the existence of an algorithm generic enough to be applied to this formulation would prove P=NP.

For the stereo problem with $\alpha_R = 1$ and $\beta = 1$ as above, the reduction of the graph labeling problem with three or more labels is not analogous to the one demonstrated above for the optical flow problem. As the label range is one-dimensional, and with the convex regularity term we cannot find three labels with equal pairwise non-zero distances, which is the simulation of the Potts problem. However, for $\alpha_R = 0$, we effectively have a Potts prior for the stereo problem as well, and the proof for the stereo problem is analogous. With the Potts prior we do not have to distribute the three labels of the graph labeling problem in three regions in two dimensions to have a pairwise equal distance for any two regions. Instead, we can put the three regions all in one horizontal line in accordance with the stereo problem.

Due to the non-convex data term, the stereo energy in (2.50a) is non-convex for any regularity term. However, for convex regularity terms, it can be reformulated into a convex problem of finding a minimal surface partitioning a 3D volume. Pock et al. [70] showed this for the case of a Total Variation (TV) regularity term, and for general convex regularity terms [69]. This method works for problems with a domain of arbitrary finite dimensionality, as long as the co-domain of the function is one-dimensional.

Also, if the image domain or the regularity prior is one-dimensional as in (2.51), a minimizer can be found in feasible time for arbitrary data terms and for arbitrary parameters of the regularity term. The case of a one-dimensional image domain is equivalent to a string matching problem [17], while the case of a one-dimensional regularity term is equivalent to finding a shortest path in a k-partite graph [43]. Both problems can be solved efficiently with dynamic programming approaches.

In summary, the energy formulation is solvable in feasible time if the domain or the co-domain of the estimated function is one-dimensional, or if both data and regularity term are convex. Otherwise, no feasible solution is known and some cases of the problems are NP-hard.

3. Dense Motion Estimation Without a Coarse-to-fine Scheme

3.1. Linearization

In the last chapter we have seen that, due to the non-convexity of the data term, finding a minimizer of the standard optical flow energy (2.50b) is generally intractable, or even NP-hard in special cases. Therefore, the vast majority of papers on optical flow in the last 30 years use a linearized version of the data term. To that end, the image indexed with the flow function is approximated by the first two terms of a Taylor series. The expansion point can either be at t_0

$$\mathcal{I}(\mathbf{p}+\mathbf{v}(\mathbf{p},t_1),t_1) \approx \mathcal{I}(\mathbf{p}+\mathbf{v}(\mathbf{p},t_0),t_0) + \left(\frac{\mathrm{d}\mathcal{I}(\mathbf{p}+\mathbf{v}(\mathbf{p},t),t)}{\mathrm{d}t}\right)\Bigg|_{\mathbf{p},t_0} (t_1 - t_0) \tag{3.1a}$$

or at t_1

$$\mathcal{I}(\mathbf{p}+\mathbf{v}(\mathbf{p},t_1),t_1) \approx \mathcal{I}(\mathbf{p},t_1) + \mathbf{v}(\mathbf{p},t_1)\left(\frac{\partial\mathcal{I}(\mathbf{p},t)}{\partial\mathbf{p}}\right)\Bigg|_{\mathbf{p},t_1}. \tag{3.1b}$$

We can transform Equation (3.1a) so that it has a similar for as (3.1b). To that end, we split the total differential in Equation (3.1a) into partial derivatives with respect to space and time by means of the chain rule:

$$\frac{\mathrm{d}\mathcal{I}}{\mathrm{d}t} = \frac{\partial\mathcal{I}}{\partial\mathbf{p}}\frac{\partial\mathbf{v}}{\partial t} + \frac{\partial\mathcal{I}}{\partial t}. \tag{3.2}$$

If we make the usual assumption that $\mathcal{I}_0$ is not warped, i.e. $\mathbf{v}(t_0) \equiv 0$, we get $\mathcal{I}(\mathbf{p}+\mathbf{v}(\mathbf{p},t_0),t_0) = \mathcal{I}(\mathbf{p},t_0)$. We can further assume that the optical flow is constant between t_0 and t_1, as we are not given any contradictory data between t_0 and t_1. From Equation (2.25) we then get

$$\mathbf{v}(\mathbf{p},t_0) + \left(\frac{\partial\mathbf{v}}{\partial t}\right)\Bigg|_{\mathbf{p},t_0} (t_1 - t_0) = \mathbf{v}(\mathbf{p},t_1). \tag{3.3}$$

In the same manner, we can assume a linear transition between the images $\mathcal{I}_0$ and $\mathcal{I}_1$, from which we get

$$\mathcal{I}(\mathbf{p},t_0) + \left(\frac{\partial\mathcal{I}}{\partial t}\right)\Bigg|_{\mathbf{p},t_0} (t_1 - t_0) = \mathcal{I}(\mathbf{p},t_1). \tag{3.4}$$

Equation (3.1a) is now reduced to

$$\mathcal{I}(\mathbf{p}+\mathbf{v}(\mathbf{p},t_1),t_1) \approx \mathcal{I}(\mathbf{p},t_1) + \mathbf{v}(\mathbf{p},t_1)\left(\frac{\mathrm{d}\mathcal{I}}{\mathrm{d}\mathbf{p}}\right)\Bigg|_{\mathbf{p},t_0} . \tag{3.5}$$

The only difference to Equation (3.1b) is the spatial image derivative taken at t_0 instead of t_1. Since both approximations are equally valid under the given assumptions, many implementations use their arithmetic mean, which we will denote from now on when using $\frac{\mathrm{d}\mathcal{I}}{\mathrm{d}\mathbf{p}}$ or $\nabla\mathcal{I}$ without a temporal index. Plugging the approximations of Equations (3.1a) or (3.1b) together with the assumptions of Equations (3.3) and (3.4) into the photoconsistency assumption in Equation (2.19), we get the differential optical flow equation (2.23).

In the case of disparity estimation in a rectified stereo setting we can apply the spatial linearization of (3.1b) with the directional derivative along the epipolar line. For a rectified stereo image pair, we get the approximation

$$\mathcal{I}(x+d(\mathbf{p}),y) \approx \mathcal{I}(\mathbf{p}) + \partial_x\mathcal{I}(\mathbf{p})d(\mathbf{p}) \tag{3.6}$$

Using this approximation for the patch-based data terms in (2.30b) together with quadratic penalization, we get a closed-form solution for the minimizer in each pixel by setting the derivatives with respect to $\mathbf{v}$ and d to zero:

$$\begin{aligned} d^*(\mathbf{p}) &= \arg\min_d \left\{ \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{p}') \left|\partial_x\mathcal{I}(\mathbf{p})d + \mathcal{I}_1(\mathbf{p}') - \mathcal{I}_0(\mathbf{p}')\right|^2 \mathrm{d}\mathbf{p}' \right\} \\ &= \frac{\left(K_w * (\partial_x\mathcal{I})^2\right)(\mathbf{p})}{\left(K_w * (\partial_x\mathcal{I}\,(\mathcal{I}_1 - \mathcal{I}_0))\right)(\mathbf{p})} \end{aligned} \tag{3.7}$$

$$\begin{aligned} \mathbf{v}^*(\mathbf{p}) &= \arg\min_{\mathbf{v}} \left\{ \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} w(\mathbf{p},\mathbf{p}') \left|\nabla\mathcal{I}(\mathbf{p}')^\mathsf{T}\mathbf{v} + \mathcal{I}_1(\mathbf{p}') - \mathcal{I}_0(\mathbf{p}')\right|^2 \mathrm{d}\mathbf{p}' \right\} \\ &= \left(\left(K_w * \left(\nabla\mathcal{I}\nabla\mathcal{I}^\mathsf{T}\right)\right)(\mathbf{p})\right)^{-1} \left(K_w * (\nabla\mathcal{I}\,(\mathcal{I}_1 - \mathcal{I}_0))\right)(\mathbf{p}) \end{aligned} \tag{3.8}$$

Here, K_w is a symmetric convolution kernel with support $\mathcal{N}(0)$ defined by the weight w. For a uniform weighting we get a box kernel, for a Gaussian weighting as in (2.31), we get a truncated Gaussian kernel. Equation (3.8) gives us information about where in the image the linearized patch-based approach is useful: The matrix $\nabla\mathcal{I}\nabla\mathcal{I}^\mathsf{T}$ has at most rank 1, and is therefore not invertible. However, due to the convolution with K_w it can get rank 2. In constant regions of the image with $\nabla\mathcal{I} = 0$, any flow vector is a correct solution, in regions close to an image edge, only the normal flow (2.24) can be estimated, and in regions close to an image corner, the full 2D flow vector can be estimated. The matrix $K_w * \left(\nabla\mathcal{I}\nabla\mathcal{I}^\mathsf{T}\right)$ is commonly known as the structure tensor and it was used as an edge detector by Förstner and Gülch [37] and by Harris and Stephens [41]. Lucas and Kanade used Equation (3.8) with a uniform weight to compute sparsely distributed optical flow vectors as well as stereo depth estimate refinements for pixels where the structure tensor is invertible, or in the case of stereo disparity estimation, where the image gradient is non-vanishing along the epipolar line [57]. Based on this approach of Lucas & Kanade, Tomasi and Kanade propose a system for feature tracking in [85], where the feature selection is based explicitly on the eigenvalues of the structure tensor.

3.2. Previous Strategies for Variational Optical Flow

Using the linearization of the data term in the variational optical flow energy formulation in (2.50b) and the analogous approximation for the stereo formulation in (2.50a), we get

$$E_{\text{stereo}}(d) = \int_\Omega \left|\partial_x\mathcal{I}(\mathbf{p})d(\mathbf{p}) + \mathcal{I}_1(\mathbf{p}) - \mathcal{I}_0(\mathbf{p})\right|^{\alpha_D} \mathrm{d}\mathbf{p} + \lambda \int_\Omega \left|\nabla d(\mathbf{p})\right|_\beta^{\alpha_R} \mathrm{d}\mathbf{p} \tag{3.9a}$$

$$E_{\text{flow}}(\mathbf{v}) = \int_\Omega \left|\nabla\mathcal{I}(\mathbf{p})^\mathsf{T}\mathbf{v}(\mathbf{p}) + \mathcal{I}_1(\mathbf{p}) - \mathcal{I}_0(\mathbf{p})\right|^{\alpha_D} \mathrm{d}\mathbf{p} + \lambda \int_\Omega \left|J_\mathbf{v}(\mathbf{p})\right|_{\beta,\gamma}^{\alpha_R} \mathrm{d}\mathbf{p}. \tag{3.9b}$$

In contrast to the formulations with a non-linearized data term, these equations are now convex in d and $\mathbf{v}$, if the exponents α_D and α_R are greater or equal to 1. This convex formulation has brought forth the majority of research on variational optical flow in the past three decades. In the following, we give a brief summary of the most decisive ones. We will omit the pixel index $\mathbf{p}$ for the sake of readability.

In their seminal paper "Determining Optical Flow" [44], Horn and Schunck propose this formulation with exponents $\alpha_D = \alpha_R = 2$ and the approximation $\partial_t\mathcal{I} \approx \mathcal{I}_1 - \mathcal{I}_0$:

$$E_{\text{HS}}(\mathbf{v}) = \int_\Omega \left|\nabla\mathcal{I}^\mathsf{T}\mathbf{v} + \partial_t\mathcal{I}\right|^2 \mathrm{d}\mathbf{p} + \lambda \int_\Omega \left|J_\mathbf{v}\right|_{2,2}^2 \mathrm{d}\mathbf{p}. \tag{3.10}$$

They minimize it by finding a solution of its corresponding Euler-Lagrange equations

$$\forall\mathbf{p}\in\Omega: \quad 0 = \partial_x\mathcal{I}\left(\nabla\mathcal{I}^\mathsf{T}\mathbf{v} + \partial_t\mathcal{I}\right) - \lambda\Delta v_1 \tag{3.11a}$$

$$0 = \partial_y\mathcal{I}\left(\nabla\mathcal{I}^\mathsf{T}\mathbf{v} + \partial_t\mathcal{I}\right) - \lambda\Delta v_2 \tag{3.11b}$$

$$\forall\mathbf{p}\in\partial\Omega: \quad 0 = \mathbf{n}^\mathsf{T}\nabla v_1 \tag{3.11c}$$

$$0 = \mathbf{n}^\mathsf{T}\nabla v_2. \tag{3.11d}$$

where $\mathbf{n}$ is the normal orthogonal to the image boundary. The Euler-Lagrange equations are linear in $\mathbf{v}$ and their discrete counterparts can be solved very efficiently by iterative methods. Horn and Schunck solve them with the Gauss-Seidel method. While being easy to optimize, the quadratic loss terms have the problem of over-penalizing outliers of the photoconsistency and regularity assumptions. Outliers in the data term occur if a pixel in the first image has no reasonable match in the second image, possibly due to noise or occlusions. Outliers in the regularity term occur mostly at occlusion boundaries, where the correct optical flow is discontinuous. The quadratic loss function in both terms penalizes few large deviations harder than many small ones. For the data term, this leads to re-occurrences of the outliers in the minimizing flow field, while for the regularity term, this leads to over-smoothing at image and flow edges.

An approach for overcoming the latter problem is incorporating image-driven regularity terms. They rely on the heuristic that strong edges in image brightness coincide with discontinuities in the resulting disparity or optical flow field. Alvarez et al. [8] propose to weight the regularity term differently in each pixel with a strictly positive, decreasing function $g : \mathbb{R} \rightarrow \mathbb{R}$, dependent on the gradient magnitude in image $\mathcal{I}_0$:

$$E_{\text{Alv}}(\mathbf{v}) = \int_\Omega \left|\nabla\mathcal{I}^{\mathsf{T}}\mathbf{v} + \partial_t\mathcal{I}\right|^2 \mathrm{d}\mathbf{p} + \lambda \int_\Omega \left|g\left(\left|\nabla\mathcal{I}_0(\mathbf{p})\right|\right) J_{\mathbf{v}}\right|^2 \mathrm{d}\mathbf{p} \tag{3.12}$$

A popular choice for g is

$$g(s) = \frac{1}{\sqrt{s+\epsilon^2}} \tag{3.13}$$

where ϵ is a small constant to prevent unbounded values in constant image regions.

While the scalar function g imposes a low weight on the regularity term in pixels with a high image gradient magnitude, the weight is the same for all directions, independent of the direction of the image gradient. Nagel and Enkelmann [61] replace the squared isotropic norm $|J_{\mathbf{v}}|^2 = \operatorname{trace}\left(J_{\mathbf{v}}J_{\mathbf{v}}{}^{\mathsf{T}}\right)$ of the Horn & Schunck formulation with an anisotropic regularity term:

$$E_{\text{NE}}(\mathbf{v}) = \int_\Omega \left|\nabla\mathcal{I}^{\mathsf{T}}\mathbf{v} + \partial_t\mathcal{I}\right|^2 \mathrm{d}\mathbf{p} + \lambda \int_\Omega \operatorname{trace}\left(J_{\mathbf{v}} D\left(\nabla\mathcal{I}_0(\mathbf{p})\right) J_{\mathbf{v}}^{\mathsf{T}}\right) \mathrm{d}\mathbf{p} \tag{3.14}$$

The diffusion tensor D is a symmetric, positive semi-definite 2×2 matrix, which, like g, only depends on the input images. Therefore, the corresponding Euler-Lagrange equations remain linear in $\mathbf{v}$:

$$\begin{aligned} \forall\mathbf{p}\in\Omega: \quad 0 &= \partial_x\mathcal{I}\left(\nabla\mathcal{I}^{\mathsf{T}}\mathbf{v} + \partial_t\mathcal{I}\right) - \lambda\operatorname{div}\left(D\nabla v_1\right) \\ 0 &= \partial_y\mathcal{I}\left(\nabla\mathcal{I}^{\mathsf{T}}\mathbf{v} + \partial_t\mathcal{I}\right) - \lambda\operatorname{div}\left(D\nabla v_2\right) \\ \forall\mathbf{p}\in\partial\Omega: \quad 0 &= \mathbf{n}^{\mathsf{T}}\nabla v_1 \\ 0 &= \mathbf{n}^{\mathsf{T}}\nabla v_2. \end{aligned} \tag{3.15}$$

For $D = \mathbf{I}$, we get the Horn & Schunck formulation (3.10), for $D = g\mathbf{I}$, we get the formulation of Alvarez et al. (3.12). In [61, 60], Nagel & Enkelmann propose several variants of a diffusion tensor D_{NE}, which only penalizes gradients of the optical flow vector perpendicular to the image gradient:

$$D_{\text{NE}} = \begin{bmatrix} (\partial_y \mathcal{I})^2 & (\partial_x \mathcal{I})(\partial_y \mathcal{I}) \\ (\partial_x \mathcal{I})(\partial_y \mathcal{I}) & (\partial_x \mathcal{I})^2 \end{bmatrix} = \nabla \mathcal{I}_\perp \nabla \mathcal{I}_\perp^\mathsf{T}, \tag{3.16}$$

where $\nabla \mathcal{I}_\perp$ denotes the 2D-vector orthogonal to the gradient $\nabla \mathcal{I}$.

Image-driven approaches as (3.12) and (3.14) allow discontinuities in the minimizing disparity or flow field and still yield linear Euler-Lagrange equations, which are easy to solve. However, the heuristic that image edges coincide with disparity or motion edges does not hold for many well-textured surfaces, where the data term might get too large a weight, leading to oversegmentation of the flow field. A different approach, which is not depending on heuristics of the image, is replacing the quadratic loss functions of Horn & Schunck with more robust loss functions. Using the non-monomial formulation of Equation (2.49b) in the formulations based on Horn & Schunck, we have $\phi_D(s) = \phi_R(s) = s$. Among several loss functions proposed in [19], Black and Anandan [20] evaluated the loss function

$$\phi(s) = \log\left(1 + \frac{1}{2}\left(\frac{s}{\sigma^2}\right)\right) \tag{3.17}$$

for the regularity term as well as for the data term. Similar to sub-quadratic monomial loss functions, it gives a comparatively smaller weight to noisy outlier pixels in the data term and preserves strong edges in the flow field. The loss function of Black & Anandan makes the energy non-convex in d and $\mathbf{v}$. Thus, its Euler-Lagrange equations can have multiple solutions not yielding a convex set. Therefore, Black & Anandan search for a local minimum of the energy with a gradient descent method.

In [22], Bruhn et al. combine the robustness to noise of the method of Lucas & Kanade (3.8) with dense variational optical flow estimation in a combined local-global (CLG) framework. The functional of Horn & Schunck in (3.10) with generic loss functions as in (2.49b) and an expanded square in the data term can be rewritten as

$$E_{\text{HS}}(\mathbf{v}) = \int_\Omega \phi_D\left(\mathbf{v}^\mathsf{T} \nabla \mathcal{I} \nabla \mathcal{I}^\mathsf{T} \mathbf{v} + \mathbf{v}^\mathsf{T} \partial_t \mathcal{I} \nabla \mathcal{I} + (\partial_t \mathcal{I})^2\right) \mathrm{d}\mathbf{p} + \lambda \int_\Omega \phi_R\left(|J_\mathbf{v}|_{2,2}^2\right). \tag{3.18}$$

By convolving the factors dependent on the image gradient with a Gaussian kernel, Bruhn et al. integrate the potentially invertible structure tensor of the Lucas-Kanade method into a variational framework (the constant term from the previous formulation can be omitted):

$$E_{\text{CLG}}(\mathbf{v}) = \int_\Omega \phi_D\left(\mathbf{v}^\mathsf{T}\left(G_\sigma * \left(\nabla \mathcal{I} \nabla \mathcal{I}^\mathsf{T}\right)\right)\mathbf{v} + \mathbf{v}^\mathsf{T}\left(G_\sigma * (\partial_t \mathcal{I} \nabla \mathcal{I})\right)\right) \mathrm{d}\mathbf{p} + \lambda \int_\Omega \phi_R\left(|J_\mathbf{v}|_{2,2}^2\right). \tag{3.19}$$

Without this convolution and without the terms from the regularity term, the two Equations (3.11a) and (3.11b) are linearly dependent, reflecting the aperture problem. With the convolution, the solution for the CLG problem can be regarded as a regularized version of the Lucas-Kanade flow (3.8) rather than a regularized version of the normal flow (2.24). For the loss functions ϕ_D and ϕ_R they use the function proposed by Charbonnier et al. in [27]:

$$\phi_{\text{Charbonnier}}(s) = 2\beta^2 \sqrt{1 + \frac{s}{\beta^2}} \tag{3.20}$$

In [29], Cohen proposes a regularity term based on the Lasso loss function $\phi_R(s) = \sqrt{s}$, combined with a square loss function for the data term:

$$E_{\text{Cohen}} = \int_\Omega \left|\nabla\mathcal{I}^\mathsf{T}\mathbf{v} + \partial_t\mathcal{I}\right|^2 \mathrm{d}\mathbf{p} + \lambda \int_\Omega |J_\mathbf{v}|_{1,2}\ \mathrm{d}\mathbf{p} \tag{3.21}$$

With the Lasso loss function the functional remains convex in $\mathbf{v}$, but edges in the minimizing flow field are much better preserved. Aubert et al. also uses this loss function for the L^1-norm of the data term ($\phi_D(s) = \sqrt{s}$) in [11], to give a smaller weight to outliers than the L^2-norm used by Horn & Schunck.

In addition to using the Lasso loss function for both data term and regularity term, Papenberg et al. incorporate various photoconsistency assumptions other than image intensity consistency into the variational framework in [68]. In contrast to Cohen, who uses a separate penalization of v_1 and v_2 ($\gamma = 1$), they use a rotationally invariant joint penalization ($\gamma = 2$). The Euler-Lagrange equations of the optical flow formulation with a linearized data term and generic loss functions ϕ_D and ϕ_R as in (3.18) read as follows:

$$\begin{aligned}
\forall\mathbf{p}\in\Omega:\ 0 &= \phi'_D\Big(\left|\nabla\mathcal{I}^\mathsf{T}\mathbf{v} + \mathcal{I}_1 - \mathcal{I}_0\right|^2\Big)\,\partial_x\mathcal{I}\Big(\nabla\mathcal{I}^\mathsf{T}\mathbf{v} + \partial_t\mathcal{I}\Big) - \lambda\mathrm{div}\Big(\phi'_R(|J_\mathbf{v}|^2_{2,2})\nabla v_1\Big)\\
0 &= \phi'_D\Big(\left|\nabla\mathcal{I}^\mathsf{T}\mathbf{v} + \mathcal{I}_1 - \mathcal{I}_0\right|^2\Big)\,\partial_y\mathcal{I}\Big(\nabla\mathcal{I}^\mathsf{T}\mathbf{v} + \partial_t\mathcal{I}\Big) - \lambda\mathrm{div}\Big(\phi'_R(|J_\mathbf{v}|^2_{2,2})\nabla v_2\Big)\\
\forall\mathbf{p}\in\partial\Omega:\ 0 &= \mathbf{n}^\mathsf{T}\nabla v_1\\
0 &= \mathbf{n}^\mathsf{T}\nabla v_2.
\end{aligned} \tag{3.22}$$

For Lasso loss functions $\phi_D(s) = \phi_R(s) = \sqrt{s}$, the corresponding derivatives in the Euler-Lagrange equations are $\phi'_D(s) = \phi'_R(s) = \frac{1}{2\sqrt{s}}$. Therefore, the Euler-Lagrange equations are non-linear. To solve these non-linear equations, Cohen employs an implicit gradient descent scheme, Aubert et al. introduce dual variables for the loss functions and alternatingly minimize the energy for $\mathbf{v}$ and for the dual variables, and Papenberg et al. employ a fixed-point iteration scheme with lagged diffusivity values ϕ'_D and ϕ'_R. Since the diffusivity values are unbounded in constant regions of the flow field with $|J_\mathbf{v}| \equiv 0$, Papenberg et al. smooth the loss functions by a small value $\epsilon \approx 10^{-3}$:

$$\phi(s) = \sqrt{s + \epsilon^2} \qquad \Rightarrow \qquad \phi'(s) = \frac{1}{2\sqrt{s + \epsilon^2}} \tag{3.23}$$

Zach et al. [94] propose an approach for minimizing the optical flow formulation with Lasso loss functions for data and regularity term based on the Legendre-Fenchel transformation. They use the separate penalization of ∇v_1 and ∇v_2 and reformulate the gradient magnitude by means of its convex biconjugate:

$$\int_\Omega |\nabla v_i| \, \mathrm{d}\mathbf{p} = \sup_{\xi \in \mathcal{C}_c^\infty(\Omega,\mathbb{R}^2)} \int_\Omega \langle \nabla v_i, \xi \rangle - \infty\big[\, |\xi| \leq 1 \big] \, \mathrm{d}\mathbf{p} \qquad i \in \{1,2\}\,. \tag{3.24}$$

Furthermore, they introduce the auxiliary variable $\mathbf{u}$ and decouple the data term from the regularity term:

$$E_{\text{Zach}}(\mathbf{u},\mathbf{v}) = \frac{1}{\lambda} \int_\Omega \left| \nabla \mathcal{I}^\mathsf{T} \mathbf{v} + \mathcal{I}_1 - \mathcal{I}_0 \right| \mathrm{d}\mathbf{p} + \frac{1}{2\theta} \int_\Omega |\mathbf{v} - \mathbf{u}|_2^2 \, \mathrm{d}\mathbf{p} + \int_\Omega |J_\mathbf{u}|_{1,2} \, \mathrm{d}\mathbf{p} \tag{3.25}$$

where the coupling parameter θ is a small constant. With this decoupling scheme, the functional becomes easy to minimize for $\mathbf{u}$ and $\mathbf{v}$ with the respective other variable kept fixed. In $\mathbf{u}$, the functional is now equivalent to the one proposed for image denoising by Rudin et al. in [74]. Zach et al. minimize it with the fixed-point iteration scheme for the dual variable ξ proposed by Chambolle in [25],

$$\xi_i^{k+1} = \frac{\xi_i^k + \tau \nabla \left(\operatorname{div}\left(\xi_i^k\right) - \frac{v_i}{\theta} \right)}{1 + \tau \left| \nabla \left(\operatorname{div}\left(\xi_i^k\right) - \frac{v_i}{\theta} \right) \right|} \qquad i \in \{1,2\}\,, \tag{3.26}$$

with a time-step $\tau \leq \frac{1}{8}$. When ξ reaches the steady-state

$$\nabla \left(\theta \operatorname{div}\left(\xi_i\right) - v_i \right) = \left| \nabla \left(\theta \operatorname{div}\left(\xi_i\right) - v_i \right) \right| \xi_i,$$

$\mathbf{u}$ can then be computed as

$$u_i = v_i - \theta \operatorname{div}\left(\xi_i\right) \qquad i \in \{1,2\}\,. \tag{3.27}$$

The minimization of E_{Zach} in $\mathbf{v}$ can be performed by a simple thresholding:

$$\mathbf{v} = \mathbf{u} + \begin{cases} \frac{\theta}{\lambda} \nabla \mathcal{I} & \text{if } \left| \nabla \mathcal{I}^\mathsf{T} \mathbf{v} + \mathcal{I}_1 - \mathcal{I}_0 \right| < -\frac{\theta}{\lambda} \left| \nabla \mathcal{I} \right|^2 \\ -\frac{\theta}{\lambda} \nabla \mathcal{I} & \text{if } \left| \nabla \mathcal{I}^\mathsf{T} \mathbf{v} + \mathcal{I}_1 - \mathcal{I}_0 \right| > \frac{\theta}{\lambda} \left| \nabla \mathcal{I} \right|^2 \\ -\frac{\nabla \mathcal{I}}{|\nabla \mathcal{I}|^2} \left| \nabla \mathcal{I}^\mathsf{T} \mathbf{v} + \mathcal{I}_1 - \mathcal{I}_0 \right| & \text{if } \left| \nabla \mathcal{I}^\mathsf{T} \mathbf{v} + \mathcal{I}_1 - \mathcal{I}_0 \right| \leq \frac{\theta}{\lambda} \left| \nabla \mathcal{I} \right|^2 \end{cases} \tag{3.28}$$

Both parts of the optimization can easily be parallelized on graphics processing units (GPUs), and Zach et al. achieved real-time frame rates for QVGA images on commodity graphics hardware at the time of publication.

For the rest of this thesis we will concentrate on the variational optical flow model with Lasso loss functions for data and regularity term, and this separate penalization of

v_1 and v_2 in the regularity terms, i.e. a penalization of the total variation (TV) of each of the two flow components.

For a more complete overview over the variety of optical flow and stereo methods we refer to various works of Barron et al. [13, 15], Scharstein et al. [76, 78, 12], Butler et al. [23], and to the popular benchmark websites at

- Middlebury Collge (http://vision.middlebury.edu)
- MPI Tübingen (http://sintel.is.tue.mpg.de)
- Karlsruhe (http://www.cvlibs.net/datasets/kitti).

3.2.1. Coarse-to-Fine Warping

The optical flow formulations presented in the last section have the common drawback of relying on the linearization of the data term as in Equations (3.1) and (3.9). Since this first-order approximation of the non-convex data term is only valid for small displacements in the sub-pixel range, the approaches will produce incorrect results in regions of high texture frequencies and large motion.

The standard solution to overcome this problem and to allow large displacements is performing the Taylor expansion in (3.1) iteratively at different expansion points corresponding to incremental stereo or optical flow estimates. To this end, the optical flow field $\mathbf{v}$ in Equation (3.1b) is split into an accumulate field $\mathbf{v}^k_{\text{acc}}$ and an incremental field $\mathbf{v}^k_{\text{inc}}$ in iteration k, and linearization is performed at $\mathbf{p}+\mathbf{v}^k_{\text{acc}}$:

$$\begin{aligned} & \mathcal{I}(\mathbf{p}+\mathbf{v}^k_{\text{acc}}(\mathbf{p},t_1)+\mathbf{v}^k_{\text{inc}}(\mathbf{p},t_1),t_1) \\ \approx\; & \mathcal{I}(\mathbf{p}+\mathbf{v}^k_{\text{acc}}(\mathbf{p},t_1),t_1)+\mathbf{v}^k_{\text{inc}}(\mathbf{p},t_1)\left(\frac{\mathrm{d}\mathcal{I}}{\mathrm{d}\mathbf{p}}\right)\Bigg|_{\mathbf{p}+\mathbf{v}^k_{\text{acc}}(\mathbf{p},t_1),t_1} \end{aligned} \tag{3.29}$$

Indexing the images with a rough displacement estimate $\mathbf{v}_{\text{acc}}$ is commonly referred to as "warping" ([68]). After computing $\mathbf{v}^k_{\text{inc}}$ with a given $\mathbf{v}^k_{\text{acc}}$, it is added to $\mathbf{v}^k_{\text{acc}}$ for the next iteration,

$$\mathbf{v}^{k+1}_{\text{acc}} = \mathbf{v}^k_{\text{acc}} + \mathbf{v}^k_{\text{inc}}, \tag{3.30}$$

and $\mathbf{v}^0_{\text{acc}}$ is typically set to 0. Averaging the image gradient of $\mathcal{I}_1$ with the one of $\mathcal{I}_0$ as a result of averaging Equations (3.1b) and (3.5) requires to also index the gradient of $\mathcal{I}_0$ with $\mathbf{v}_{\text{acc}}$, which is typically not done in implementations involving warping. Instead, only the gradient of $\mathcal{I}_1$ is used for every refinement of the flow field.

To overcome the restriction of the optical flow to small displacements due to linearization, this approach is usually embedded into a hierarchical system of images at different resolutions. To this end, the images are downsampled to a coarser resolution, the stereo or flow field is computed for the coarse resolution, upsampled, and used as an initial estimate for the subsequent linearization on a finer resolution. On the coarse resolution, linearization of the data term is justified because larger displacements on the original resolution now become small enough to be in the sub-pixel vicinity. On subsequent finer scales, linearization of the data term is justified as only increments in a sub-pixel vicinity have to be estimated. Several of the works referenced in the last section use this coarse-to-fine warping strategy [57, 10, 19, 20, 68, 94]. A theoretical justification and a quantitative evaluation of coarse-to-fine warping on the Yosemite sequence (see [13, 15]) can be found in [68]. Given a ground truth disparity or optical flow field d_{GT} or $\mathbf{v}_{\mathrm{GT}}$, usually computed from synthetic data like the Yosemite sequence and the Sintel benchmark [23] or from real-world sequences carefully captured under laboratory conditions as in the Middlebury benchmark [12], the standard discrete error metrics for optical flow evaluation are the average end-point error (AEE) and the average angular error (AAE), defined as

$$AEE(\mathbf{v}) = \frac{1}{\|\Omega\|} \sum_{\mathbf{p}\in\Omega} |\mathbf{v}(\mathbf{p}) - \mathbf{v}_{\mathrm{GT}}(\mathbf{p})|_2 \tag{3.31a}$$

$$AAE(\mathbf{v}) = \frac{1}{\|\Omega\|} \sum_{\mathbf{p}\in\Omega} \arccos\left(\frac{\langle \mathbf{v}(\mathbf{p}), \mathbf{v}_{\mathrm{GT}}(\mathbf{p})\rangle + 1}{\sqrt{|\mathbf{v}(\mathbf{p})|_2^2 + 1}\sqrt{|\mathbf{v}_{\mathrm{GT}}(\mathbf{p})|_2^2 + 1}} \right). \tag{3.31b}$$

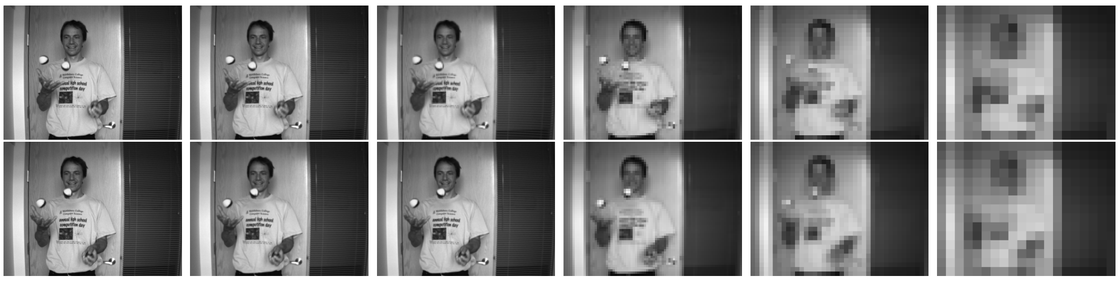
For disparity fields, which are typically quantized at a coarser scale than optical flow fields, the standard error metric is the number of falsely classified pixels with an AEE greater than a given threshold T:

$$ERR(d) = \sum_{\mathbf{p}\in\Omega} \left[\, |d(\mathbf{p}) - d_{\mathrm{GT}}(\mathbf{p})| > T \right] \tag{3.31c}$$

While coarse-to-fine warping helps to improve the quality of the estimated optical flow fields on many image sequences containing large motions, this strategy entails two significant problems:

- Small structures moving over large distances between two frames of an image sequence vanish on coarser image scales. Figure 3.1 demonstrates this effect on frames 10 and 11 of the Middlebury Beanbags sequence at various resolutions. As a result, the optical flow of pixels belonging to these structures cannot be estimated correctly, if the objects move over distances greater than their size from one frame to the next.

- While a large variety of data terms can be applied (see [68] for a comparison), all of them have to be spatially differentiable. Moreover, some of the discriminative power of the data terms can get lost under the linearization, as we will show in Section 3.5.



(a) 640×480 (b) 320×240 (c) 160×120 (d) 80×60 (e) 40×30 (f) 20×15

Figure 3.1.: **Middlebury "Beanbags" sequence at different resolutions.** Top row: frame 10. Bottom row: frame 11.

In the following, we propose an alternative method of optical flow and disparity estimation that does not rely on linearization and can therefore cope with large displacements of small objects as well as arbitrary data terms.

3.3. A Formulation without Linearization or Coarse-to-Fine Warping

To introduce our alternative solution we have a look at the decoupled functional (3.25) of Zach et al., combined with the original non-linearized data term in (2.50b):

$$\begin{aligned} E_{\text{Dec}}(\mathbf{u},\mathbf{v}) &= \frac{1}{\lambda}\int\limits_{\Omega} \phi_D\left(\left|\mathcal{I}_1(\mathbf{p}+\mathbf{v}(\mathbf{p})) - \mathcal{I}_0(\mathbf{p})\right|^2\right) \mathrm{d}\mathbf{p} \\ &+ \frac{1}{2\theta}\int\limits_{\Omega} |\mathbf{v}(\mathbf{p}) - \mathbf{u}(\mathbf{p})|_2^2 \,\mathrm{d}\mathbf{p} \\ &+ \int\limits_{\Omega} |J_{\mathbf{u}}(\mathbf{p})|_{1,2} \,\mathrm{d}\mathbf{p}. \end{aligned} \tag{3.32}$$

We make three observations:

- Firstly, for $\theta \to 0$, the functional becomes a generalized continuous formulation of (2.55b), therefore it cannot be optimized globally with respect to both $\mathbf{u}$ and $\mathbf{v}$ for an arbitrary θ.
- Secondly, just as in (3.25), the functional is convex in $\mathbf{u}$ and equivalent to the image denoising functional proposed by Rudin et al. in [74]. Therefore, for a fixed $\mathbf{v}$, it can be globally optimized in $\mathbf{u}$ with an arbitrary method for convex optimization.

- Thirdly, for a fixed $\mathbf{u}$, the functional can be optimized in $\mathbf{v}$ independently for every pixel $\mathbf{p}$. The dependence of the minimizer in pixel $\mathbf{p}$ on neighborhing pixels of $\mathbf{p}$, which makes the original functional (2.50b) NP-hard, comes from the regularity term. By decoupling the data from the regularity term, the data term can be minimized in $\mathbf{v}(\mathbf{p})$ for every pixel $\mathbf{p}$ by an exhaustive search around $\mathbf{u}(\mathbf{p})$.

Combining these observations, we propose an iterative method for optimizing (3.32) by alternating the optimization in $\mathbf{u}$ and $\mathbf{v}$. We start with a relatively large θ and successively decrease θ after each pair of minimizations for $\mathbf{u}$ and $\mathbf{v}$. This allows $\mathbf{u}$ and $\mathbf{v}$ to be significantly different at the beginning, and successively "anneals" the optimization process by more and more limiting the difference between $\mathbf{u}$ and $\mathbf{v}$ for smaller θ.

The proposed decoupling scheme can also be used for disparity estimation in the case of two-view or multi-view stereo. As disparity estimation can be performed globally optimal in feasible time, the benefit of the decoupling scheme is not theoretical in the sense of an approximation of an NP-hard problem, but rather practical in comparison to the globally optimal method of Pock et al. in [70, 69]. While their approach of functional lifting requires to store fields in the 3D volume spanned by the image domain Ω and the range of disparity values, the optimization in the proposed decoupling scheme remains in the 2D domain, yielding a lower memory demand and shorter computation time.

The exhaustive search is the most computationally expensive part of the optimization, especially for the case of optical flow. In the context of implementation, we want to emphasize that the proposed approach is easily parallelizable on modern GPUs, and doing so yields a significant speedup. The exhaustive search still prevents our approach from achieving real-time frame rates for optical flow estimation on recent commodity hardware. However, compared to all formulations for optical flow estimation mentioned in the last section, our proposed model has two significant advantages:

- Since the algorithm does not rely on image coarsening, there is no issue with small-scale structures being lost on coarser scales that warping schemes require for estimating large motions. As a consequence, our algorithm provides better motion fields for small scale structures undergoing large displacements.

- Without the linearization of the data term in previous approaches, we remove the constraint on the data term regarding linearizable photoconsistency assumptions. Without this constraint, we can incorporate local color values or patch-based and correlation-based photoconsistency assumptions as in Section 2.5 into our energy.

In the context of algorithmic optimization with respect to runtime performance, a trivial strategy for accelerating the exhaustive search for $\mathbf{v}$ is the limitation of the search space for $\mathbf{v}(\mathbf{p})$ in each pixel $\mathbf{p}$ to a window around $\mathbf{u}(\mathbf{p})$. Qualitatively, this hyperparameter corresponds to the number of pyramid levels specified in warping schemes.

A different strategy is limiting the number of samples in the search space. While most stereo approaches work with a coarse quantization of a fraction of a pixel, optical flow is generally computed with floating-point precision. While the optimal solution for $\mathbf{u}$ has inherent floating-point precision, the precision of the optimal solution for $\mathbf{v}$ is limited to fractions of pixels, because we only take an integer amount of samples.

When sampling the data term at sub-pixel values, we have to do so according to a given method for sub-pixel interpolation of the images $\mathcal{I}_1$, and according to a given choice of a photoconsistency assumption. In the next section, we present analytic solutions for computing optical flow and stereo fields with sub-pixel precision for an underlying bilinear interpolation, which is the standard model implemented in GPU hardware besides nearest-neighbor interpolation with a trivial sub-pixel solution. As the data term we use the L^1 data term. We will show that an analytic sub-pixel optimization is well suited for 1D disparity estimation with a standard image intensity photoconsistency assumption, but rather ill-suited for 2D optical flow estimation and more complex photoconsistency assumptions.

3.3.1. Sub-pixel Accuracy and Parallelization

The Rectified Stereo Case

In the one-dimensional case of rectified stereo, sub-pixel optimization comes down to finding the optimal floating-point increment α for any integer disparity value a:

$$\begin{aligned} &\operatorname*{arg\,min}_{d\in\mathbb{R}_{\geq 0}} \left\{ \frac{1}{\lambda} \left| \mathcal{I}_1(x+d,y) - \mathcal{I}_0(\mathbf{p}) \right| + \frac{1}{2\theta} (d-d_0)^2 \right\} \\ = \quad &\operatorname*{arg\,min}_{a\in\mathbb{N}_0, \alpha\in[0,1]} \left\{ \frac{1}{\lambda} \left| \mathcal{I}_1(a+\alpha,y) - \mathcal{I}_0(\mathbf{p}) \right| + \frac{1}{2\theta} (a+\alpha-x-d_0)^2 \right\} \end{aligned} \tag{3.33}$$

Here the value d_0 represents the auxiliary variable $\mathbf{u}$ of the decoupled optical flow model. Using linear image interpolation in the 1D case, we can rewrite the value of $\mathcal{I}_1$ at sub-pixel position as

$$\mathcal{I}_1(a+\alpha,y) = \mathcal{I}_1(a,y) + (\mathcal{I}_1(a+1,y) - \mathcal{I}_1(a,y))\,\alpha. \tag{3.34}$$

With the equality

$$|\sigma\alpha + \mu_0| = \left| \sigma \left(\alpha + \frac{\mu_0}{\sigma} \right) \right| = |\sigma|\,|\alpha+\mu| \qquad \text{for } \sigma \neq 0, \quad \mu = \frac{\mu_0}{\sigma}$$

we can rewrite the data term as

$$\begin{aligned} &\frac{1}{\lambda} \left| (\mathcal{I}_1(a{+}1,y) - \mathcal{I}_1(a,y))\,\alpha + \mathcal{I}_1(a,y) - \mathcal{I}_0(\mathbf{p}) \right| \\ = \quad &\underbrace{\frac{1}{\lambda} \left| \mathcal{I}_1(a{+}1,y) - \mathcal{I}_1(a,y) \right|}_{\sigma} \left| \alpha - \underbrace{\left(\frac{\mathcal{I}_0(\mathbf{p}) - \mathcal{I}_1(a,y)}{\mathcal{I}_1(a{+}1,y) - \mathcal{I}_1(a,y)} \right)}_{\mu} \right| \end{aligned} \tag{3.35}$$

Combined with the coupling term, the search for the optimal sub-pixel increment α is now the problem of finding an α with minimal absolute distance to the offset μ and minimal quadratic distance to the offset η:

$$\alpha^* = \arg\min_{\alpha} \left\{ \sigma \left| \alpha - \mu \right| + \frac{1}{2\theta} \left(\alpha - \underbrace{(x + d_0 - a)}_{\eta} \right)^2 \right\} \tag{3.36a}$$

$$\Rightarrow \quad \alpha^* = \eta - \sigma\theta \ \mathrm{sgn}(\alpha^* - \mu) \tag{3.36b}$$

Obviously, the minimizing α^* lies between η and μ. If $|\eta - \mu| \leq \theta\sigma$, the absolute distance to μ outweighs the quadratic distance to η, and the minimum is found at μ. Otherwise, the minimum is found according to (3.36b) with $\mathrm{sgn}(\alpha^* - \mu) = \mathrm{sgn}(\eta - \mu)$. Finally, the minimum of the convex energy in (3.36a) has to be projected into the interval $[0, 1]$. We get the closed-form solution

$$\alpha^* = \min \left\{ 1, \max \left\{ 0, \left(\begin{cases} \min \{\eta + \sigma\theta, \mu\} & \text{if } \mu \geq \eta \\ \min \{\eta - \sigma\theta, \mu\} & \text{if } \mu \leq \eta \end{cases} \right) \right\} \right\} . \tag{3.37}$$

The Optical Flow Case

The splitting of the optical flow vector into an integer and a floating-point part is analogous to the 1D case in (3.33). For clarity we refrain from the vectorial notation and write the energy depending on the two separate optical flow components v_1 and v_2:

$$\begin{aligned} & \arg\min_{v_1, v_2 \in \mathbb{R}} \left\{ \frac{1}{\lambda} \left| \mathcal{I}_1(x{+}v_1, y{+}v_2) - \mathcal{I}_0(\mathbf{p}) \right| + \frac{1}{2\theta} \left((v_1{-}u_1)^2 + (v_2{-}u_2)^2 \right) \right\} \\ = & \arg\min_{a_1, a_2 \in \mathbb{Z}, \alpha_1, \alpha_2 \in [0,1]} \left\{ \frac{1}{\lambda} \left| \mathcal{I}_1(a_1{+}\alpha_1, a_2{+}\alpha_2) - \mathcal{I}_0(\mathbf{p}) \right| \right. \\ + & \left. \frac{1}{2\theta} \left((a_1{+}\alpha_1{-}x{-}u_1)^2 + (a_2{+}\alpha_2{-}y{-}u_2)^2 \right) \right\} \end{aligned} \tag{3.38}$$

In the 2D case we assume a bilinear model for image interpolation. The L^1 difference of the data term in every pixel can be reformulated as

$$
\begin{aligned}
& \frac{1}{\lambda}\left|\mathcal{I}_1\left(a_1{+}\alpha_1, a_2{+}\alpha_2\right) - \mathcal{I}_0(\mathbf{p})\right| \\
= & \left|\underbrace{\frac{1}{\lambda}\left(\mathcal{I}_1(a_1{+}1, a_2) - \mathcal{I}_1(a_1, a_2)\right)}_{\mathcal{I}_x}\alpha_1 + \underbrace{\frac{1}{\lambda}\left(\mathcal{I}_1(a_1, a_2{+}1) - \mathcal{I}_1(a_1, a_2)\right)}_{\mathcal{I}_y}\alpha_2\right. \\
+ & \underbrace{\frac{1}{\lambda}\left(\mathcal{I}_1(a_1{+}1, a_2{+}1) + \mathcal{I}_1(a_1, a_2) - \mathcal{I}_1(a_1{+}1, a_2) - \mathcal{I}_1(a_1, a_2{+}1)\right)}_{\mathcal{I}_{xy}}\alpha_1\alpha_2 \\
+ & \left.\underbrace{\frac{1}{\lambda}\left(\mathcal{I}_1(a_1, a_2) - \mathcal{I}_0(\mathbf{p})\right)}_{\mathcal{I}_r}\right| \\
= & \left|\mathcal{I}_x\alpha_1 + \mathcal{I}_y\alpha_2 + \mathcal{I}_{xy}\alpha_1\alpha_2 + \mathcal{I}_r\right|. \qquad (3.39)
\end{aligned}
$$

In the following, we abbreviate the term $\mathcal{I}_x\alpha_1 + \mathcal{I}_y\alpha_2 + \mathcal{I}_{xy}\alpha_1\alpha_2 + \mathcal{I}_r$ with $\rho\left(\alpha_1, \alpha_2\right)$. It vanishes on the hyperbola

$$
\alpha_2 = -\frac{\mathcal{I}_x}{\mathcal{I}_{xy}} + \frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2}\frac{1}{\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}}\quad, \qquad (3.40)
$$

which has the asymptotes $\alpha_1 = -\frac{\mathcal{I}_y}{\mathcal{I}_{xy}}$, and $\alpha_2 = -\frac{\mathcal{I}_x}{\mathcal{I}_{xy}}$, and that is stretched by the factor of $\frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2}$ (see Appendix B.1). We can easily test whether this hyperbola intersects the search space $[0,1]^2$ by comparing the sign of the data term at the four corner pixels. Depending on the two regions of $\rho\left(\alpha_1, \alpha_2\right) \lessgtr 0$, we have to minimize the energy including the coupling term:

$$
\begin{aligned}
E(\alpha_1, \alpha_2) &= \pm\mathcal{I}_x\alpha_1 \pm \mathcal{I}_y\alpha_2 \pm \mathcal{I}_{xy}\alpha_1\alpha_2 \pm \mathcal{I}_r \\
&+ \frac{1}{2\theta}\left(\left(\alpha_1 - \underbrace{(x + u_1 - a_1)}_{\eta_1}\right)^2 + \left(\alpha_2 - \underbrace{(y + u_2 - a_2)}_{\eta_2}\right)^2\right) \qquad (3.41)
\end{aligned}
$$

The partial derivatives of the energy are

$$
\partial_x E = \pm\mathcal{I}_x \pm \mathcal{I}_{xy}\alpha_2 + \frac{1}{\theta}\left(\alpha_1 - \eta_1\right) \qquad (3.42a)
$$

$$
\partial_y E = \pm\mathcal{I}_y \pm \mathcal{I}_{xy}\alpha_1 + \frac{1}{\theta}\left(\alpha_2 - \eta_2\right) \qquad (3.42b)
$$

and in the minimum we have for α_1 and α_2

$$\alpha_1 = \frac{\eta_1 + \mathcal{I}_y \mathcal{I}_{xy} \theta^2 \mp \mathcal{I}_{xy} \theta \eta_2 \mp \mathcal{I}_x \theta}{1 - \mathcal{I}_{xy}^2 \theta^2} \tag{3.43a}$$

$$\alpha_2 = \frac{\eta_2 + \mathcal{I}_x \mathcal{I}_{xy} \theta^2 \mp \mathcal{I}_{xy} \theta \eta_1 \mp \mathcal{I}_y \theta}{1 - \mathcal{I}_{xy}^2 \theta^2} \tag{3.43b}$$

The eigenvalues of the Hessian matrix are $\frac{1}{\theta} \mp \mathcal{I}_{xy}$ for the eigenvectors $\begin{bmatrix}\sqrt{2} & \mp\sqrt{2}\end{bmatrix}^\mathsf{T}$. If $\frac{1}{\theta} = |\mathcal{I}_{xy}|$, the energy does not have a unique minimum in the open sets of $\rho > 0$ and $\rho < 0$. Therefore, the minimum in $[0,1]^2$ lies either on the boundary or on the hyperbola where $\rho = 0$. The same is true for $\frac{1}{\theta} < |\mathcal{I}_{xy}|$, where the energy is concave in one direction. If $\frac{1}{\theta} > |\mathcal{I}_{xy}|$, the energy is convex in (α_1, α_2) and has a unique minimum $(\alpha_1{}^*, \alpha_2{}^*)$ for each of the two regions. If the minimum of each of the two regions lies in the respected region itself, we have to project them into $[0,1]^2$ and choose the minimum. If not, we have to find the minimum on the hyperbola or on the boundary of $[0,1]^2$. To this end, we have to find the minimum of the coupling term for α_1, where α_2 is substituted as in (3.40). This involves finding the zeros of a polynomial of degree 3 (see Appendix B.2). If the projection of the analytic minimum onto $[0,1]^2$ does not lie in the same region of $\pm\rho \leq 0$, we have have to find the minimum on the boundary of $[0,1]^2$. An algorithm for analytic sub-pixel optimization in each pixel with the L^1 data term and image intensity photoconsistency assumption can be found in Appendix B.3, for the case that $\frac{1}{\theta} \neq |\mathcal{I}_{xy}|$. While it is possible to perform an analytic sub-pixel optimization for optical flow in principle, the following observations lead us to refrain from doing so:

- The algorithm is sufficiently complex with respect to case distinctions to produce highly divergent code not suitable for GPU parallelization.
- The same case distinctions are required for the optimization on the boundary of $[0,1]^2$.
- The analytic solution on the hyperbola is also too complex for GPU implementation.
- The degenerate case $\frac{1}{\theta} = |\mathcal{I}_{xy}|$ is not covered yet. Neither are the degenerate cases for the analytic solution on the hyperbola.

Furthermore, the presented methods only work for an L^1 data term with image intensity photoconsistency assumption and a bilinear image interpolation model, for the optical flow case as well as for the stereo case. Using patch-based data terms as in Section 3.5 would further complicate the computation by a large degree. Therefore, we refrain from using an analytic sub-pixel optimization and rather evaluate supersampling the energy at various resolutions in Section 3.4.2.

In addition to supersampling the energy at interpolated image positions, we can fit a paraboloid through 5 adjacent energy measurements: For a discretely sampled flow vector $\mathbf{v}^s = (v_1^s, v_2^s)$ and the corresponding center position $\mathbf{x}_c = (x_c, y_c) = (x + v_1^s, y + v_2^s)$, let $\mathbf{x}_l$, $\mathbf{x}_r$, $\mathbf{x}_u$, and $\mathbf{x}_d$ denote the image positions $(x_c - s, y_c)$, $(x_c + s, y_c)$, $(x_c, y_c - s)$, and $(x_c, y_c + s)$ adjacent to $\mathbf{x}_c$ with a supersampling step size s. Then the paraboloid fitted through the 5 energy values $E(\mathbf{x}_l)$, $E(\mathbf{x}_r)$, $E(\mathbf{x}_u)$, $E(\mathbf{x}_d)$, and $E(\mathbf{x}_c)$ is given by

$$E(x,y) = a\,(x - x_c)^2 + b\,(x - x_c) + c\,(y - y_c)^2 + d\,(y - y_c) + e \tag{3.44a}$$

with

$$\begin{aligned} a &= \frac{E(\mathbf{x}_l) - 2E(\mathbf{x}_c) + E(\mathbf{x}_r)}{2s^2}, \quad & b &= \frac{E(\mathbf{x}_r) - E(\mathbf{x}_l)}{2s} \\ c &= \frac{E(\mathbf{x}_u) - 2E(\mathbf{x}_c) + E(\mathbf{x}_d)}{2s^2}, \quad & d &= \frac{E(\mathbf{x}_d) - E(\mathbf{x}_u)}{2s} \\ e &= E(\mathbf{x}_c) \end{aligned} \tag{3.44b}$$

and an extremum at

$$0 = 2a\,(x - x_c) + b = 2c\,(y - y_c) + d \tag{3.44c}$$

for $a \neq 0$ and $c \neq 0$. In the iterative search for the point $\mathbf{x}_c$ of minimal energy, we can buffer at least two, and, depending on the amount of available cache, possibly all four adjacent energy values. Computing the sub-pixel refinement $\mathbf{v}^{\text{sub}} = (\mathbf{x} - \mathbf{x}_c)$ is equivalent to performing a Newton step

$$\mathbf{v}^{\text{sub}} = \mathbf{x}_c - HE(\mathbf{x}_c)^{-1} \nabla E(\mathbf{x}_c) \tag{3.45}$$

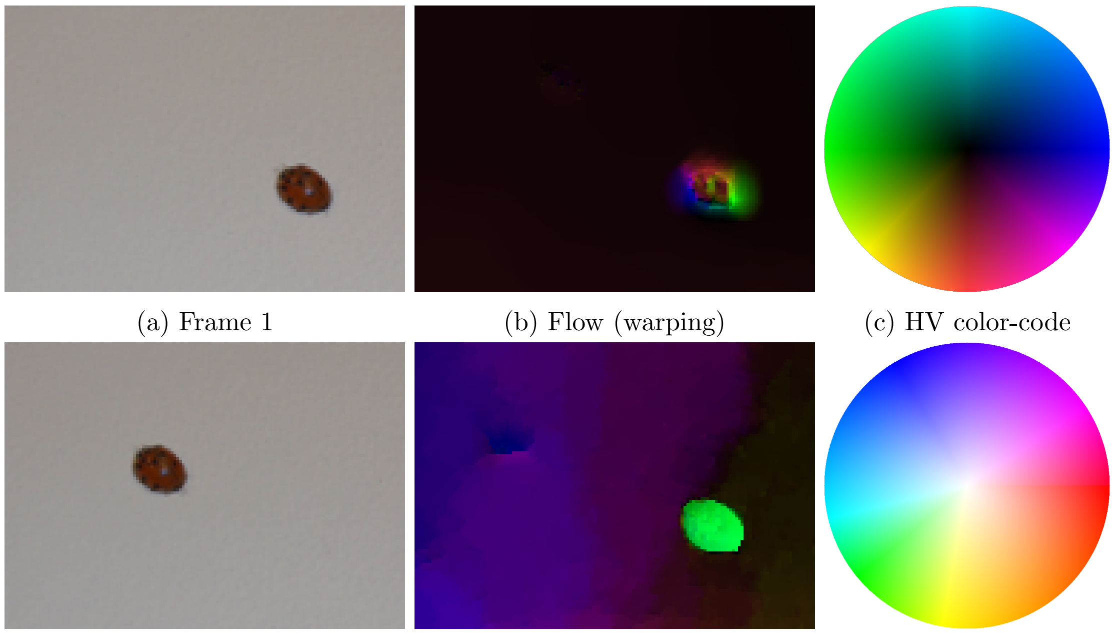
with gradient $\nabla E(\mathbf{x}_c) = \begin{bmatrix} b \\ d \end{bmatrix} \mathbf{x}_c$, and Hessian $HE(\mathbf{x}_c) = \begin{bmatrix} 2a & 0 \\ 0 & 2c \end{bmatrix}$. If $\mathbf{v}^{\text{sub}}$ lies in the interval $[-s, s]^2$, we include the sub-pixel value $\left(\mathbf{v}_s + \mathbf{v}^{\text{sub}}\right)$ in the exhaustive search. In the next section we include sample results of this method.

3.4. Experimental Evaluation

In the following, we are going to evaluate our method of optical flow computation in two ways: First, we demonstrate the effectiveness of our method qualitatively on several real-world images showing large displacements of small-scale objects, and compare it against the linearization-based method of Zach et al. [95]. Second, we perform a quantitative parameter evaluation on images of the Middlebury benchmark [12].

3.4.1. Large Motion

Since real-world images typically do not come with a ground truth flow field, we evaluate the results in a two-fold manner:



(a) Frame 1 (b) Flow (warping) (c) HV color-code

(d) Frame 2 (e) Flow (proposed) (f) HS color-code

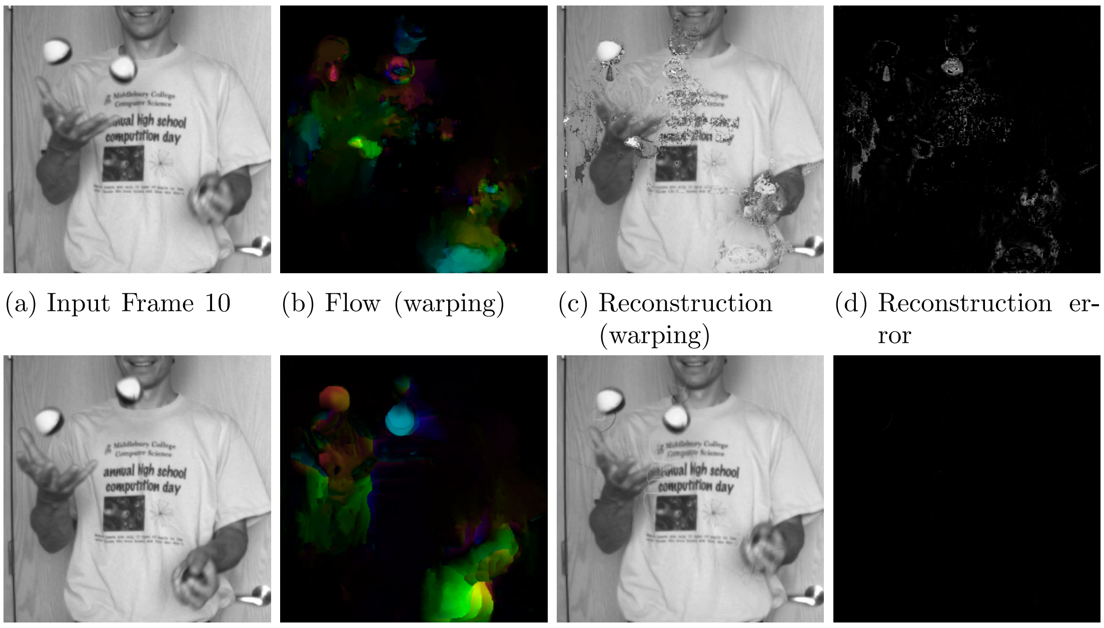
Figure 3.2.: **Large displacement of small-scale structures.** For two images of a lady bug taken seconds apart, in contrast to the coarse-to-fine warping schemes, the proposed approach allows to accurately estimate the correspondence. The flow is visualized with an HSV color-code using the hue and value channel.

- We verify qualitatively whether the computed and color-coded flow is meaningful.
- We check the consistency of the flow field by reconstructing the first of the two frames $\mathcal{I}_0^r$ using the second frame $\mathcal{I}_1$ and the estimated flow field $\mathbf{v}$ according to

$$\mathcal{I}_0^r(\mathbf{p}) := \mathcal{I}_1(\mathbf{p} + \mathbf{v}(\mathbf{p})). \tag{3.46}$$

  If the flow field is correct, then the reconstructed first frame $\mathcal{I}_0^r$ is identical to the observed one $\mathcal{I}_0$. We can quantify this error by plotting the difference image $|\mathcal{I}_0^r - \mathcal{I}_0|$.

Both evaluations are purely qualitative. A quantitative assessment of the difference image alone is meaningless, since it exactly represents an instance of the data term of Equation (2.41b), and without a regularity term, it can be trivially by a single exhaustive search for a possibly chaotic flow field. Therefore, we also evaluate the flow field qualitatively and check whether it matches the visually observed motion in the images.



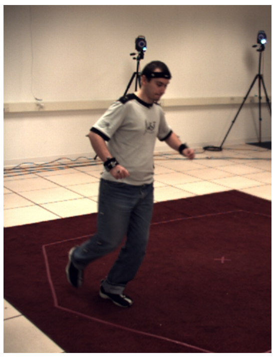
(a) Input Frame 10 (b) Flow (warping) (c) Reconstruction (warping) (d) Reconstruction error

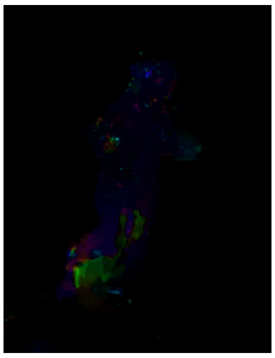
(e) Input Frame 11 (f) Flow (proposed) (g) Reconstruction (proposed) (h) Reconstruction error

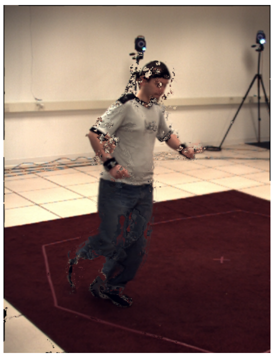
Figure 3.3.: **Comparison of reconstructed images from flow fields computed with and without warping.** The experiments show the flow fields and reconstructions of frame 10 computed from frame 11 of the Middlebury Beanbags sequence. While the warping scheme (above) clearly loses small scale structures such as the fast moving ball, these are appropriately preserved with the proposed algorithm (below). As a consequence, we obtain a substantially smaller reconstruction error.

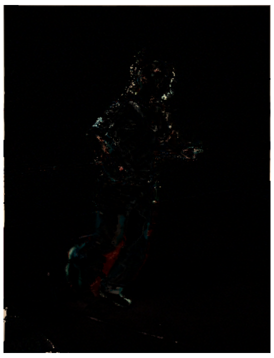
Figures 3.2, 3.3, and 3.4 show comparisons of the proposed method and the method of Zach et al. [95]. The flow fields are color-coded using the Hue-Saturation-Value (HSV) colorcode shown in Figures 3.2c and 3.2f. The orientation of the flow vector is encoded in the hue channel, with linear interpolation in 6 sections between the colors red, magenta, blue, cyan, green, and yellow. In Figure 3.2c its magnitude is encoded in the value channel, and the hue sections have equal sizes. In Figure 3.2f its magnitude is encoded in the saturation channel, and the hue sections have different sizes based on perceptual dissimilarity [12].

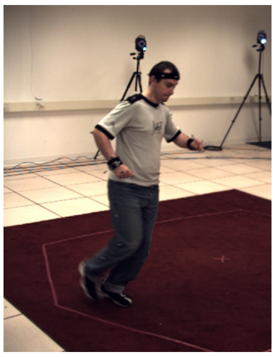
The experiments demonstrate that indeed the motion fields and the reconstructed frames obtained with the proposed approach are more convincing than those obtained with the warping scheme. In all cases, the motion is larger than the size of the moving objects. A closer observation shows that the warping scheme gives rise to flow fields

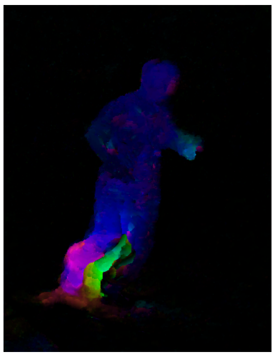

(a) Input frame 546

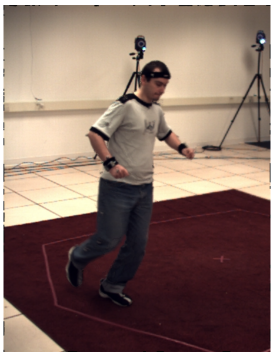

(b) Flow (warping)


(c) Reconstruction (warping)


(d) Reconstruction error


(e) Input frame 550


(f) Flow (proposed)


(g) Reconstruction (proposed)

(h) Reconstruction error

Figure 3.4.: **Performance of the proposed algorithm on color sequences.** The experiments show the flow fields and reconstructions of frame 546 computed from frame 550 and the estimated flow field for two images from the HumanEva-II sequence of Sigal et al. in [80]. In contrast to the warping scheme, the proposed method finds correspondences for fast moving structures as well as for occluded areas.

which tend to shrink the respective structures (to account for their disappearance). This is most prominent in Figure 3.2b, where the motion is the opposite of the color wheel in Figure 3.2c, which yields a convergent motion. In more complex scenes, the warping scheme incorrectly matches small structures to the most similar structures in their vicinity - see Figures 3.3 and 3.4. In contrast, our method provides reliable motion estimates which give rise to faithful reconstructions of the first frame.

3.4.2. Parameter Evaluation

All in all, our proposed method has several free parameters:

- the type of photoconsistency assumption used in the data term with its own parameters,
- the regularity weight λ,
- the initial and final values of the coupling parameter θ,
- the rate at which θ is decreased over time,
- the size of the search window used in the exhaustive search for $\mathbf{v}$,
- the supersampling frequency used in the exhaustive search for $\mathbf{v}$,
- the convex optimization method used for finding a minimizing $\mathbf{u}$ for the energy (3.32), and its parameters.

For an evaluation of the scalar parameters we use the standard image intensity photoconsistency assumption, and deal with more advanced data terms in the next section. The weight of the regularity term λ is a parameter of the energy functional and it is independent of the method used to minimize the energy. It has to be set according to the photoconsistency assumption, the range of image intensity values, and the motion in the scene. In our experiments, we set it to 0.02 for image intensity values between 0 and 1. The size of the search window is also solely dependent on the motion in the images, and it has to be set just large enough to capture the largest vectors in the flow field. In our experiments, we use a window of 41×41 pixels. For the convex optimization of the energy in $\mathbf{u}$ we use the method of Chambolle and Zach et al. according to Equations (3.26) and (3.27).

We evaluate the remaining parameters of our method on the RubberWhale, Hydrangea, Dimetrodon, Grove2, Urban2, and Venus datasets of the Middlebury benchmark [12]. Figure 3.5 shows the images used in our evaluation. The top row shows the 10th frame of the respective image sequence, the middle row shows the 11th frame. The bottom row shows the ground truth flow field from the 10th to the 11th frame. We evaluate one parameter at the time and keep the respective other parameters fixed at λ=50, θ-start=50, θ-stop=0.1, and θ-factor=0.99.

The most prominent parameter determining the quality of the estimated flow field is the supersampling frequency in the exhaustive search. Table 3.1 shows a quantitative evaluation of this parameter. Except for the Grove2 and Urban2 sequences, where the flow error is caused predominantly by large occlusions, both AAE and AEE, defined in Equations (3.31), are monotonically decreasing the more samples we take per pixel. However, the runtime required for the exhaustive search is approximately proportional

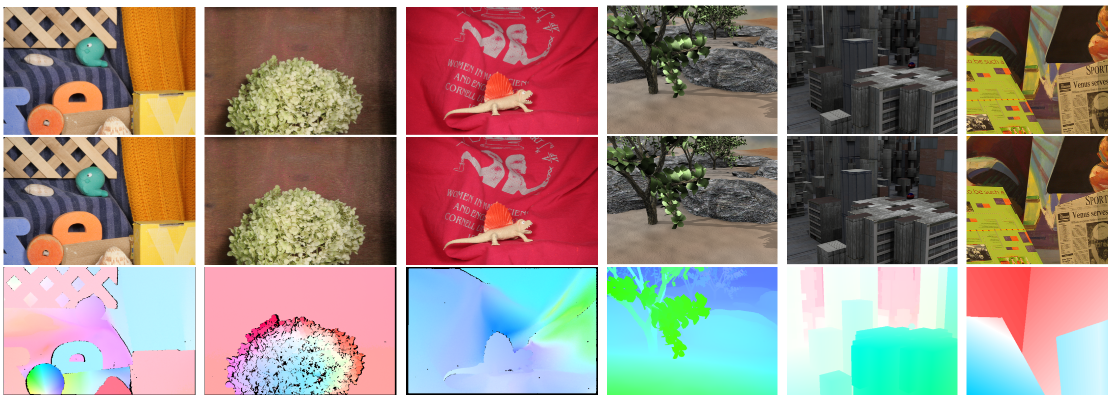
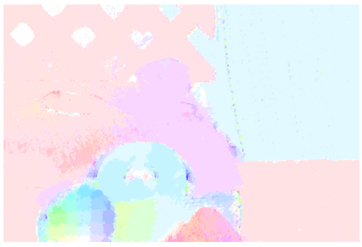

(a) RubberWhale (b) Hydrangea (c) Dimetrodon (d) Grove2 (e) Urban2 (f) Venus

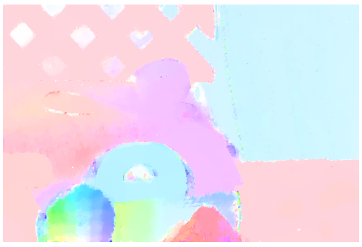
Figure 3.5.: **Middlebury benchmark images.** From top to bottom: Frame 10, frame 11, color-coded ground truth flow field using the color-code in 3.2f.

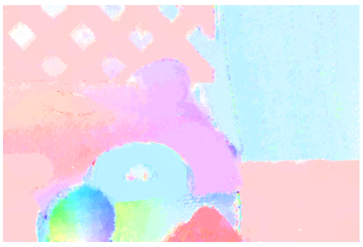
Samples/Pixel		1	4	25	100	400	2500
RubberWhale	AAE	0.181	0.136	0.101	0.097	0.094	**0.093**
	AEE	0.357	0.254	0.186	0.177	0.172	**0.170**
	Time	8	28	158	594	2253	13453
Hydrangea	AAE	0.086	0.070	0.056	0.053	0.052	**0.052**
	AEE	0.385	0.330	0.263	0.251	0.241	**0.241**
	Time	9	28	158	594	2253	13452
Dimetrodon	AAE	0.165	0.096	0.074	0.069	0.068	**0.067**
	AEE	0.441	0.269	0.212	0.200	0.197	**0.196**
	Time	9	28	158	594	2253	13453
Grove2	AAE	0.142	0.083	0.063	0.061	**0.060**	0.061
	AEE	0.607	0.352	0.257	0.247	**0.240**	0.242
	Time	11	37	214	802	3050	18284
Urban2	AAE	0.176	0.122	0.095	0.087	**0.081**	0.083
	AEE	**1.682**	1.842	2.027	1.945	1.724	1.879
	Time	12	37	214	802	3050	18320
Venus	AAE	0.136	0.113	0.102	0.101	0.100	**0.099**
	AEE	0.604	0.453	0.367	0.357	0.350	**0.344**
	Time	6	20	112	423	1608	9603

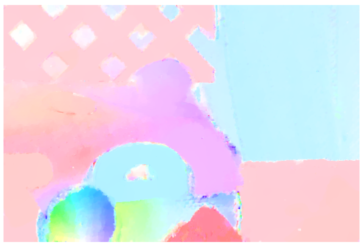
Table 3.1.: Quantitative evaluation of the supersampling frequency.

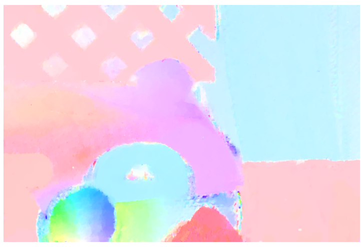
to the number of samples per pixel. We attribute the slightly sublinear increase in

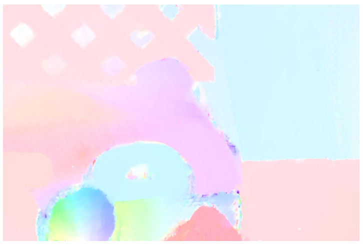

(a) 1 sample

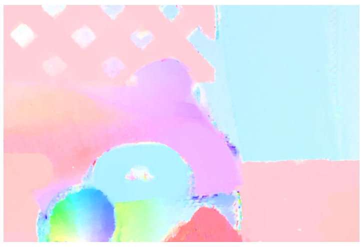

(b) 1 sample,paraboloid

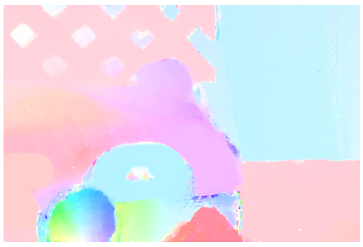

(c) 2×2 samples

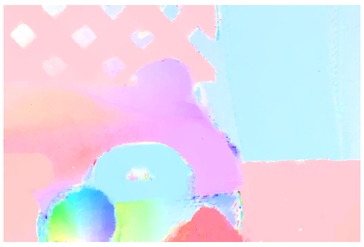

(d) 2×2 samples,paraboloid

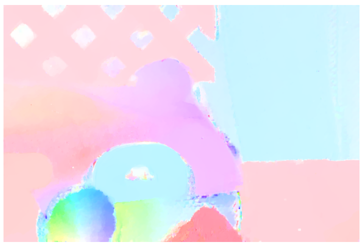

(e) 5×5 samples

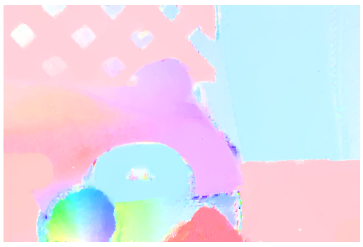

(f) 5×5 samples,paraboloid


(g) 10×10 samples


(h) 10×10 samples, paraboloid


(i) 20×20 samples


(j) 20×20 samples, paraboloid


(k) 50×50 samples

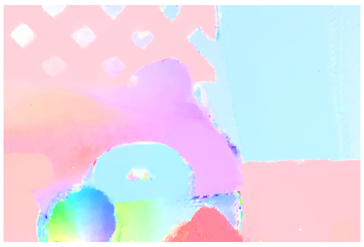

(l) 50×50 samples, paraboloid

Figure 3.6.: **Qualitative evaluation of different resolutions of supersampling the data term.**

runtime in our experiments to having an increasing number of cache hits with densely sampled images on our GPU architecture. Figure 3.6 shows the color-coded flow fields of the RubberWhale sequence sampled with 1 to 2500 samples per pixel. It can be seen particularly well on the wheel structure in the lower left part of the image, that an increasing number of samples reduces structured errors in the flow field.

Samples/Pixel		1	4	25	100	400	2500
RubberWhale	AAE	0.122	0.102	0.095	0.093	**0.092**	0.092
	AEE	0.229	0.188	0.175	0.172	**0.168**	0.169
	Time	44	171	1037	4150	16700	104923
Hydrangea	AAE	0.066	0.056	0.053	0.052	**0.051**	0.051
	AEE	0.305	0.264	0.247	0.242	**0.239**	0.239
	Time	45	172	1037	4150	16691	104868
Dimetrodon	AAE	0.104	0.071	0.067	0.067	0.067	**0.066**
	AEE	0.285	0.205	0.196	0.195	0.196	**0.194**
	Time	45	172	1037	4150	16694	104697
Grove2	AAE	0.087	0.061	0.060	**0.060**	0.060	0.061
	AEE	0.375	0.258	0.243	**0.241**	0.242	0.244
	Time	61	234	1402	5603	22515	142101
Urban2	AAE	0.127	0.086	0.084	0.080	**0.078**	0.084
	AEE	1.611	1.709	1.904	1.827	**1.792**	2.040
	Time	61	232	1401	5604	22534	142273
Venus	AAE	0.109	0.102	0.099	**0.098**	0.099	0.098
	AEE	0.418	0.364	0.344	0.343	0.341	**0.340**
	Time	32	123	741	2963	11921	74790

Table 3.2.: Quantitative evaluation of the supersampling frequency with additional paraboloid fitting.

The effects of fitting a paraboloid through 5 adjacent energy values, as discussed at the end of Section 3.3.1, are demonstrated in Table 3.2. The parameters and hardware are identical to the ones used for Table 3.1. The bold numbers show the best results in each row before rounding to 3 decimal places. We can observe two prominent results in this experiment: Firstly, with paraboloid fitting, the AAE and AEE are not monotonically decreasing with more samples per pixel on every dataset anymore. Secondly, the error values for a certain number of samples with paraboloid fitting are mostly lower or equal than the error values for twice the number of samples without paraboloid fitting. However, also the required runtime values are higher than the ones for doubling the number of samples in each dimension. Therefore, fitting a paraboloid can be a valid alternative to using more samples per pixel, depending on the hardware capabilities.

Tables 3.3, 3.4, and 3.5 show evaluations of the start and stop values for the coupling value θ, as well as the factor multiplied with θ in each iteration. While Table 3.5 shows that in general a slow decrement of θ produces better results, Tables 3.3 and 3.4 show that optimal start and stop values of θ are dependent on the image sequence.

This is not surprising, as our method is dependent on the initial values of $\mathbf{u}$ and $\mathbf{v}$,

θ-start		200	100	50	20	10	1
RubberWhale	AAE	**0.180**	0.181	0.182	0.181	0.181	0.189
	AEE	**0.356**	**0.356**	0.358	0.357	0.357	0.376
Hydrangea	AAE	0.087	0.087	**0.086**	**0.086**	0.087	0.447
	AEE	0.386	**0.384**	0.385	0.386	0.386	1.911
Dimetrodon	AAE	**0.165**	0.166	**0.165**	**0.165**	0.167	0.376
	AEE	**0.441**	0.443	**0.441**	0.443	0.447	0.934
Grove2	AAE	0.143	0.142	0.142	**0.141**	0.142	0.577
	AEE	0.609	0.608	0.607	**0.606**	0.608	1.641
Urban2	AAE	**0.172**	0.175	0.176	0.230	0.362	0.785
	AEE	**1.154**	1.625	1.682	3.070	4.787	7.553
Venus	AAE	0.134	0.136	0.136	**0.130**	0.131	0.565
	AEE	0.597	0.602	0.604	**0.594**	0.637	2.389

Table 3.3.: Quantitative evaluation of the coupling weight start value.

and for an initial $\mathbf{u} \equiv \mathbf{v} \equiv 0$, the combination of λ and the start value θ is a prior on the magnitude of the motion vectors in the first optimization of $\mathbf{v}$. However, Table 3.3 also shows us that our method is fairly robust against different choices for the initial value of θ, as the resulting error values hardly vary for choices between 20 and 200.

Next to λ, the choice of the stop value for θ is a prior on the regularity of the resulting flow as well: For $\theta = 0$ we have $\mathbf{u} \equiv \mathbf{v}$, however, for $\theta > 0$ we have different flow fields for $\mathbf{u}$ and $\mathbf{v}$, and choosing either one for evaluation poses a bias towards flow fields either fulfilling the data or the regularity term. In our experiments we used $\mathbf{u}$ for evaluation, and Table 3.4 shows us that the optimal flow field is slightly smoother than the joint optimum of $\mathbf{u}$ and $\mathbf{v}$.

3.5. Advanced Data Terms

As described in Section 3.2.1, the proposed method of decoupling the data term from the regularity term allows us to incorporate any data term of the form $E(\mathbf{v}, \mathbf{p}) : \mathbb{R}^2 \times \Omega \to \mathbb{R}_{\geq 0}$. The data term is neither required to be convex in $\mathbf{v}$, nor does it have to maintain its discriminative power under linearization. The latter feature is of particular interest, because patch-based data terms are prone to lose a lot of their discriminative power under linearization in the presence of high-frequency image data.

For example, we consider the SSD data term of Equation (2.30b) with $\alpha_D = 2$,

θ-stop		1	0.5	0.2	0.1	0.05	0.02
RubberWhale	AAE	**0.141**	0.148	0.173	0.181	0.185	0.188
	AEE	**0.272**	0.284	0.340	0.357	0.367	0.374
Hydrangea	AAE	**0.058**	0.066	0.080	0.086	0.090	0.093
	AEE	**0.286**	0.315	0.361	0.385	0.399	0.408
Dimetrodon	AAE	**0.092**	0.113	0.157	0.165	0.170	0.173
	AEE	**0.293**	0.313	0.417	0.441	0.455	0.463
Grove2	AAE	0.097	**0.087**	0.124	0.142	0.153	0.160
	AEE	0.430	**0.362**	0.527	0.607	0.653	0.681
Urban2	AAE	**0.137**	0.142	0.171	0.176	0.179	0.181
	AEE	**1.635**	1.577	1.661	1.682	1.694	1.702
Venus	AAE	0.136	**0.122**	0.129	0.136	0.140	0.142
	AEE	0.604	**0.525**	0.572	0.604	0.623	0.634

Table 3.4.: Quantitative evaluation of the coupling weight stop value.

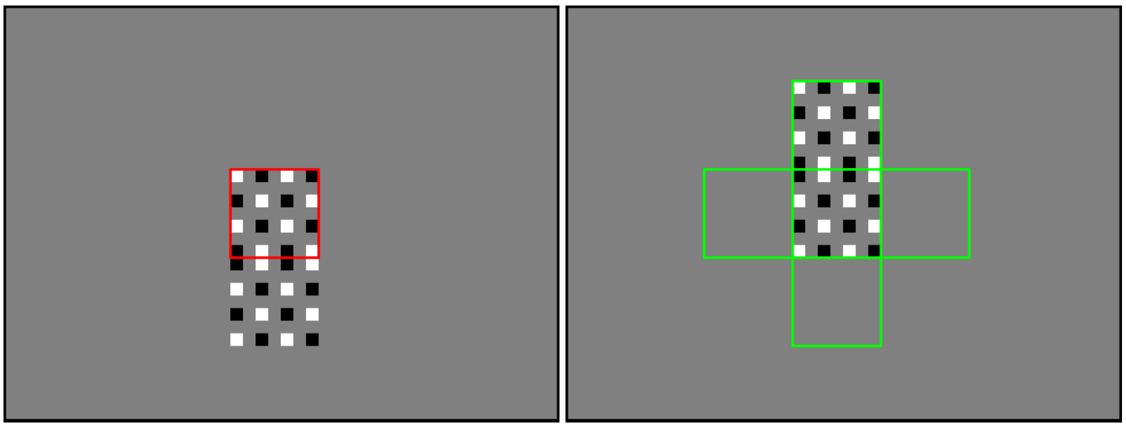
θ-factor		0.5	0.8	0.9	0.95	0.98	0.99
RubberWhale	AAE	0.169	0.107	0.097	0.096	**0.095**	0.097
	AEE	0.328	0.205	0.182	0.177	**0.175**	0.177
Hydrangea	AAE	0.592	0.134	0.061	0.053	**0.053**	**0.053**
	AEE	2.94	0.774	0.308	0.259	**0.251**	**0.251**
Dimetrodon	AAE	0.441	0.174	0.105	0.076	0.072	**0.069**
	AEE	1.145	0.545	0.338	0.223	0.207	**0.200**
Grove2	AAE	0.691	0.289	0.152	0.091	0.067	**0.061**
	AEE	2.065	0.979	0.554	0.344	0.265	**0.247**
Urban2	AAE	0.759	0.517	0.408	0.305	0.148	**0.087**
	AEE	7.400	6.547	5.853	4.970	3.363	**1.945**
Venus	AAE	0.743	0.371	0.220	0.141	0.105	**0.101**
	AEE	2.866	1.885	1.235	0.734	0.426	**0.357**

Table 3.5.: Quantitative evaluation of the coupling weight decay factor.

combined with a Gaussian weighting as in (2.31):

$$E_{\mathrm{SSD,Gauss}}\left(\mathbf{v},\mathbf{p}\right) = \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} \exp\left(-\frac{\left\|\mathbf{p}-\mathbf{p}'\right\|^2}{2\sigma^2}\right)\left(\mathcal{I}_1\left(\mathbf{p}'+\mathbf{v}\right)-\mathcal{I}_0\left(\mathbf{p}'\right)\right)^2 \mathrm{d}\mathbf{p}', \qquad (3.47a)$$

The linearized and normalized version of the data term can be written as a Gaussian



(a) Image $\mathcal{I}_0$ (b) Image $\mathcal{I}_1$

Figure 3.7.: **Example for patch-based optical flow.** Optical flow methods based on linearization have problems determining the motion of the marked patch from frame 1 on the left to frame 2 on the right.

convolution

$$
\begin{aligned}
& E_{\mathrm{SSD,Gauss}}(\mathbf{v},\mathbf{p}) \\
\approx\ & \frac{1}{\|w_\sigma\|} \int\limits_{\mathbf{p}'\in\mathcal{N}(\mathbf{p})} \exp\left(-\frac{\|\mathbf{p}-\mathbf{p}'\|^2}{2\sigma^2}\right)\left(\mathcal{I}_1(\mathbf{p}) + \nabla\mathcal{I}_1(\mathbf{p})^\mathsf{T}\mathbf{v} - \mathcal{I}_0(\mathbf{p})\right)^2 \mathrm{d}\mathbf{p}' \\
=\ & \mathbf{v}^\mathsf{T}\left(G_\sigma * \left(\nabla\mathcal{I}_1\nabla\mathcal{I}_1^\mathsf{T}\right)\right)(\mathbf{p})\mathbf{v} + \left(G_\sigma * \left(\left(\mathcal{I}_1 - \mathcal{I}_0\right)\nabla\mathcal{I}_1\right)\right)(\mathbf{p}) \qquad (3.47\mathrm{b})
\end{aligned}
$$

Combined with the approximation $\partial_t\mathcal{I} \approx \mathcal{I}_1 - \mathcal{I}_0$ of (3.4), a robust loss function ϕ_D, and a regularity term, we get the data term of the CLG method (3.19) of Bruhn et al. [22]. Using the Charbonnier loss function, the CLG method is robust against outliers and efficiently produces accurate flow fields. However, the linearized constancy assumption limits the method to images with dominating low-frequency structure.

Figure 3.7 illustrates problems of the CLG method when applied to high-frequency image structure. The marked patch in image $\mathcal{I}_0$ on the left clearly moves upward in image $\mathcal{I}_1$. However, a linearized data term as in Equation (3.47b) indicates a different motion: Without loss of generality, let the patch center $\mathbf{p}$ be 0. For every image gradient $\nabla\mathcal{I}_1 = \begin{bmatrix}\partial_x\mathcal{I}_1 & \partial_y\mathcal{I}_1\end{bmatrix}^\mathsf{T}$ at point $\mathbf{p}' = \begin{bmatrix}x' & y'\end{bmatrix}^\mathsf{T}$ there exists an orthogonal image gradient $\nabla\mathcal{I}_1^\perp = \begin{bmatrix}\partial_y\mathcal{I}_1 & -\partial_x\mathcal{I}_1\end{bmatrix}^\mathsf{T}$ of equal magnitude at point $\mathbf{p}'_\perp = \begin{bmatrix}-y' & x'\end{bmatrix}^\mathsf{T}$. Since G_σ is point symmetric at 0, we get a multiple of the identity matrix for $G_\sigma * \left(\nabla\mathcal{I}_1\nabla\mathcal{I}_1^\mathsf{T}\right)$:

$$\begin{aligned}
&\int\limits_{\mathbf{p}'\in\mathcal{N}(0)} w_\sigma\left(0,\mathbf{p}'\right)\nabla\mathcal{I}_1(\mathbf{p}')\nabla\mathcal{I}_1(\mathbf{p}')^\mathsf{T}\,\mathrm{d}\mathbf{p}' \\
=& \int\limits_{\substack{\{\mathbf{p}'\in\mathcal{N}(0)|\\ x'y'\geq 0, y'\neq 0\}}} w_\sigma\left(0,\mathbf{p}'\right)\left(\nabla\mathcal{I}_1\left(\mathbf{p}'\right)\nabla\mathcal{I}_1\left(\mathbf{p}'\right)^\mathsf{T} + \nabla\mathcal{I}_1\left(\mathbf{p}'_\perp\right)\nabla\mathcal{I}_1\left(\mathbf{p}'_\perp\right)^\mathsf{T}\right)\mathrm{d}\mathbf{p}' \\
=& \int\limits_{\substack{\{\mathbf{p}'\in\mathcal{N}(0)|\\ x'y'\geq 0, y'\neq 0\}}} w_\sigma\left(0,\mathbf{p}'\right)\left(\begin{pmatrix}\partial_x\mathcal{I}_1\\ \partial_y\mathcal{I}_1\end{pmatrix}\begin{pmatrix}\partial_x\mathcal{I}_1\\ \partial_y\mathcal{I}_1\end{pmatrix}^\mathsf{T} + \begin{pmatrix}\partial_y\mathcal{I}_1\\ -\partial_x\mathcal{I}_1\end{pmatrix}\begin{pmatrix}\partial_y\mathcal{I}_1\\ -\partial_x\mathcal{I}_1\end{pmatrix}^\mathsf{T}\right)(\mathbf{p}')\,\mathrm{d}\mathbf{p}' \\
=& \int\limits_{\substack{\{\mathbf{p}'\in\mathcal{N}(0)|\\ x'y'\geq 0, y'\neq 0\}}} w_\sigma\left(0,\mathbf{p}'\right)\left(\left(\partial_x\mathcal{I}_1\right)^2 + \left(\partial_y\mathcal{I}_1\right)^2\right)(\mathbf{p}')\mathbf{I}\,\mathrm{d}\mathbf{p}' \qquad (3.48)
\end{aligned}$$

Moreover, for every image gradient $\nabla\mathcal{I}_1$ at point $\mathbf{p}'$, there is an opposing image gradient $-\nabla\mathcal{I}_1$ at point $-\mathbf{p}'$ with an equal temporal derivative $(\mathcal{I}_1 - \mathcal{I}_0)$. In $G_\sigma * \left(\left(\mathcal{I}_1 - \mathcal{I}_0\right)\nabla\mathcal{I}\right)$ these sum up to 0, leaving

$$E_{\mathrm{SSD,Gauss}}(\mathbf{v},\mathbf{p}) = \frac{1}{2}\left(\int\limits_{\mathbf{p}'\in\mathcal{N}(0)} \left|\nabla\mathcal{I}(\mathbf{p}')\right|^2\,\mathrm{d}\mathbf{p}'\right)\mathbf{v}^\mathsf{T}\mathbf{v} \qquad (3.49)$$

Without the influence of the regularity term, the data term suggests that the patch does not move at all, which is one of the worst possible motions for the patch in this example. This stems from the fact that linearization discards the high image frequencies in the patch, and that the patch moves over a relatively large distance. For this example, the standard solution of embedding linearization of the data term into a coarse-to-fine hierarchy fails as well, since on coarser scales of the Gaussian scale-space the intensity of a patch converges to the average gray value of the patch. The average gray values of all the marked patches in image $\mathcal{I}_1$ in Figure 3.7 are the same, rendering each patch an equally good match for the center patch in image $\mathcal{I}_0$.

3.5.1. Experimental Evaluation

In the following, we evaluate the qualitative and quantitative performance of several data terms in our proposed algorithmic framework. We compare the image intensity data term (2.27) to the patch-based SAD data term (2.30b), the NCC data term (2.37), the census data term (2.38), and the rank data term (2.39). For a quantitative and qualitative evaluation we use the same datasets as in the last section. The quantitative

Dataset	RubberWhale		Hydrangea		Dimetrodon		Grove2		Urban2		Venus	
∼ Time	599 s		600 s		600 s		810 s		810 s		428 s	
$1/\lambda$	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE
5	0.145	0.274	0.042	0.217	0.067	0.205	0.055	0.214	0.135	4.029	0.142	0.487
7	0.127	0.243	0.041	0.207	0.062	0.187	0.052	0.202	0.115	3.260	0.130	0.432
10	0.113	0.218	**0.041**	0.203	0.060	0.176	0.049	0.193	0.097	2.786	0.118	0.389
12	0.106	0.201	0.042	**0.203**	**0.060**	**0.174**	0.049	**0.192**	0.097	2.759	0.115	0.379
15	0.101	0.189	0.042	0.205	0.061	0.180	**0.048**	0.193	0.090	2.580	0.109	0.358
20	0.095	0.176	0.044	0.211	0.063	0.183	0.049	0.198	0.088	2.492	0.108	0.353
25	**0.094**	0.173	0.046	0.218	0.064	0.188	0.050	0.203	0.084	2.113	0.105	0.349
30	0.094	0.173	0.047	0.226	0.065	0.189	0.054	0.217	0.083	2.071	0.104	0.348
35	0.094	**0.172**	0.049	0.231	0.065	0.191	0.057	0.227	**0.081**	1.966	**0.099**	**0.341**
40	0.095	0.174	0.051	0.239	0.067	0.194	0.057	0.231	0.094	2.166	0.100	0.346
45	0.096	0.175	0.052	0.244	0.067	0.196	0.059	0.239	0.090	2.039	0.101	0.349
50	0.097	0.177	0.053	0.251	0.069	0.200	0.061	0.247	0.087	1.945	0.101	0.357
70	0.102	0.190	0.057	0.269	0.073	0.212	0.069	0.278	0.105	**1.909**	0.105	0.381
100	0.110	0.208	0.063	0.295	0.083	0.237	0.080	0.323	0.222	2.371	0.113	0.430
120	0.114	0.217	0.066	0.311	0.088	0.253	0.087	0.349	0.113	1.826	0.120	0.461

Table 3.6.: SAD data term evaluation

Dataset	RubberWhale		Hydrangea		Dimetrodon		Grove2		Urban2		Venus	
∼ Time	6837 s		6830 s		6830 s		9256 s		9272 s		6284 s	
$1/\lambda$	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE
0.7	0.133	0.254	0.040	0.200	0.062	0.185	0.051	0.196	0.075	0.634	0.125	0.413
1	0.113	0.212	0.039	0.193	0.058	0.168	0.047	0.182	0.065	0.539	0.101	0.355
1.5	0.094	0.168	**0.039**	0.188	0.056	0.160	0.045	0.177	0.059	0.512	0.098	0.335
2	0.090	0.158	0.039	**0.186**	0.056	0.159	**0.044**	**0.177**	0.055	0.509	0.085	0.297
2.5	0.089	0.155	0.039	0.187	**0.055**	**0.157**	0.045	0.179	0.053	0.468	0.083	0.289
3	0.085	0.149	0.040	0.191	0.056	0.159	0.045	0.181	0.051	0.461	0.069	0.267
4	0.083	0.146	0.042	0.196	0.056	0.161	0.046	0.187	**0.050**	**0.452**	**0.063**	**0.260**
5	**0.082**	**0.144**	0.043	0.204	0.058	0.165	0.047	0.194	0.051	0.459	0.064	0.265
7	0.082	0.145	0.046	0.217	0.062	0.179	0.049	0.206	0.056	0.502	0.068	0.279
10	0.084	0.150	0.050	0.236	0.066	0.193	0.052	0.221	0.063	0.555	0.071	0.300

Table 3.7.: Patch data term evaluation

results for various regularity weights λ are shown in Tables 3.6 to 3.10. For qualitative evaluation we show the color-coded flow images for the best parameter setup according to the AEE (3.31a) in Figure 3.8, using the Middlebury color-code shown in Figure 3.2f. We sampled all patch-based data terms in a box of 3×3 samples, centered at sub-pixel location and with the space of one pixel between the samples in each direction. For the coupling parameter θ we used a start value of 50, a stop value of 0.1, and a decay factor of 0.99. We used a search radius of 20 pixels, and a sampling frequency of 10×10 samples per pixel.

Among the different data terms, the rank data term clearly performs worst. The information of how many pixels in the patch are brighter or darker than the center pixel is not very strong. If we do not require sub-pixel accuracy, we can use the rank data term as a pre-computed image transform. The other patch-based data terms however compare each sample individually, yielding much better results. When we look at the resulting flow images in Figure 3.8, we see that the strength of the patch-based data terms lies in correctly estimating the motion of small untextured regions, such as the hole in one of the objects of the RubberWhale image, or the gap in the middle of the Venus image. On the other hand, the patch-based data terms are more susceptible to image noise, especially the NCC and census data terms, as they are based on high-frequency image derivatives. In the quantitative results, the patch-based SAD data term is able to outperform the standard SAD data term on every data set. The NCC data term outperforms the SAD data term by large margins on most data sets, while performing slightly worse on the Dimetrodon sequence and the Grove2 sequence in terms of the AAE, and much worse on the Urban sequence. The census data term yields better results than the SAD data term on the RubberWhale and Hydrangea sequences, while yielding worse results on the other data sets. Similar to the rank data term, the census data term can be regarded as an image transform on integer pixel positions. For a 3×3 sample window, the census image simply stores the 8bit bitstring in every pixel. The data term computation then only involves a fast binary Hamming distance computation of two census pixels. This is also a core feature of the closely related BRIEF [24] and ORB [73] descriptors.

Dataset	RubberWhale		Hydrangea		Dimetrodon		Grove2		Urban2		Venus	
$\sim$ Time	7992 s		7983 s		7988 s		10852 s		10973 s		5704 s	
$1/\lambda$	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE
0.1	0.174	0.334	0.051	0.331	0.066	0.197	0.100	0.381	0.517	7.082	0.193	0.873
0.2	0.104	0.201	0.042	0.275	0.062	0.185	0.059	0.227	0.479	6.926	0.121	0.394
0.5	0.072	0.134	0.034	0.196	**0.062**	**0.181**	**0.046**	**0.179**	0.399	6.416	0.073	0.288
0.7	0.065	0.119	0.032	0.178	0.066	0.189	0.045	0.175	0.373	6.253	0.068	0.268
1	0.055	0.102	0.030	0.148	0.068	0.197	0.047	0.180	0.354	6.027	**0.072**	**0.278**
1.5	**0.053**	**0.098**	0.029	0.139	0.072	0.207	0.054	0.206	0.372	6.003	0.080	0.310
2	0.054	0.100	**0.028**	0.136	0.074	0.215	0.059	0.227	0.382	5.996	0.098	0.382
2.5	0.058	0.106	0.028	0.136	0.077	0.224	0.062	0.260	0.379	5.936	0.122	0.494
3	0.061	0.114	0.029	**0.136**	0.081	0.234	0.066	0.286	0.362	5.816	0.143	0.611
4	0.070	0.136	0.029	0.138	0.088	0.253	0.076	0.345	0.331	5.396	0.174	0.790
5	0.078	0.157	0.030	0.141	0.096	0.273	0.087	0.421	0.306	4.401	0.198	0.919
7	0.088	0.192	0.032	0.148	0.112	0.317	0.097	0.484	0.269	3.687	0.231	1.112
10	0.097	0.233	0.034	0.160	0.140	0.393	0.111	0.572	0.273	3.281	0.273	1.376
12	0.102	0.253	0.035	0.166	0.157	0.447	0.114	0.611	**0.260**	2.721	0.304	1.575
15	0.108	0.285	0.037	0.175	0.186	0.536	0.123	0.661	0.270	2.634	0.325	1.766
20	0.118	0.340	0.041	0.189	0.228	0.686	0.137	0.742	0.278	**2.569**	0.359	2.065
25	0.126	0.380	0.044	0.204	0.262	0.834	0.151	0.810	0.288	2.606	0.394	2.319
30	0.133	0.423	0.046	0.218	0.297	0.981	0.162	0.876	0.306	2.689	0.423	2.550

Table 3.8.: NCC data term evaluation

Dataset	RubberWhale		Hydrangea		Dimetrodon		Grove2		Urban2		Venus	
$\sim$ Time	1794 s		1796 s		1785 s		2435 s		2427 s		1278 s	
$1/\lambda$	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE
0.01	0.240	0.446	0.494	2.477	0.087	0.282	0.459	1.374	0.742	7.675	0.799	3.140
0.02	0.170	0.320	0.053	0.348	0.072	0.218	0.101	0.392	0.604	7.340	0.537	2.520
0.05	0.107	0.208	0.039	0.261	**0.070**	**0.209**	0.063	0.251	0.555	7.170	0.208	1.548
0.07	0.090	0.174	0.036	0.238	0.074	0.217	0.055	0.219	0.540	7.109	0.171	1.307
0.1	0.072	0.141	0.034	0.215	0.077	0.226	0.053	0.208	0.542	7.042	0.145	1.080
0.2	0.062	0.118	**0.032**	0.179	0.083	0.240	**0.050**	**0.201**	0.508	6.886	**0.113**	0.656
0.3	**0.060**	**0.113**	0.033	0.173	0.088	0.252	0.051	0.206	**0.497**	6.818	0.117	0.622
0.5	0.063	0.118	0.034	**0.171**	0.096	0.273	0.056	0.229	0.510	6.802	0.121	**0.575**
0.7	0.072	0.133	0.036	0.182	0.103	0.295	0.061	0.251	0.531	6.820	0.130	0.603
1	0.082	0.154	0.040	0.199	0.118	0.334	0.069	0.286	0.541	6.793	0.149	0.676
1.5	0.101	0.191	0.047	0.230	0.147	0.413	0.085	0.352	0.565	6.817	0.174	0.759
2	0.119	0.229	0.053	0.264	0.179	0.506	0.103	0.426	0.576	**6.792**	0.202	0.879
2.5	0.135	0.270	0.061	0.300	0.213	0.602	0.122	0.502	0.599	6.876	0.225	0.969

Table 3.9.: Census data term evaluation

Dataset	RubberWhale		Hydrangea		Dimetrodon		Grove2		Urban2		Venus	
$\sim$ Time	6091 s		6089 s		6194 s		10386 s		8239 s		4336 s	
$1/\lambda$	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE	AAE	AEE
10	0.128	0.251	0.876	3.177	0.625	1.417	1.336	3.157	1.012	8.141	1.098	3.577
12	0.120	0.238	0.861	3.146	0.625	1.414	1.332	3.152	1.002	8.128	1.096	3.573
15	0.114	0.228	**0.860**	**3.133**	0.617	1.398	1.311	3.120	0.986	8.106	1.100	3.575
20	0.108	0.218	0.882	3.137	0.618	1.388	1.299	3.099	0.983	8.099	1.112	3.587
25	0.109	0.218	0.904	3.151	0.617	1.379	1.294	3.087	**0.977**	**8.088**	1.106	3.580
30	0.108	0.216	0.919	3.159	0.618	1.378	1.283	3.068	0.981	8.089	1.115	3.592
35	0.107	0.215	0.937	3.171	0.618	1.377	1.275	3.055	0.981	8.091	1.115	3.593
40	**0.106**	**0.213**	0.953	3.184	0.620	1.377	1.271	3.048	0.986	8.094	1.108	3.586
45	0.108	0.215	0.966	3.194	0.618	1.373	1.267	3.041	0.988	8.095	1.110	3.588
50	0.108	0.217	0.979	3.206	**0.616**	**1.368**	1.268	3.043	0.992	8.101	1.111	3.589
70	0.111	0.221	1.000	3.202	0.620	1.375	1.252	3.020	0.997	8.107	1.107	3.584
100	0.119	0.235	1.018	3.217	0.621	1.380	1.243	3.006	1.003	8.115	1.109	3.588
120	0.120	0.238	1.023	3.226	0.626	1.389	1.231	2.992	1.006	8.119	1.098	3.573
150	0.124	0.244	1.031	3.233	0.630	1.398	1.226	2.985	1.005	8.119	1.101	3.577
170	0.125	0.245	1.032	3.232	0.629	1.397	**1.223**	2.985	1.004	8.117	1.101	3.578
200	0.126	0.249	1.034	3.232	0.629	1.397	1.224	**2.984**	1.002	8.114	1.094	**3.567**
250	0.127	0.252	1.026	3.224	0.628	1.397	1.224	2.985	1.006	8.119	**1.093**	3.568
300	0.128	0.252	1.023	3.223	0.630	1.400	1.224	2.987	1.008	8.125	1.094	3.570
350	0.130	0.256	1.028	3.239	0.628	1.396	1.223	2.985	1.002	8.114	1.106	3.588

Table 3.10.: RANK data term evaluation

3.6. Conclusion

In this chapter we have introduced an alternative minimization strategy for estimating the dense optical flow between two images with a variational method consisting of a data term and a regularity term. Other than the majority of existing methods, our method does not rely on a linear approximation of the non-convex data term. Instead, we optimize the non-linearized energy by decoupling the data term from the regularity term, and perform an alternating optimization in both of these terms. The energy of the data term can be minimized globally by an exhaustive search, while the energy of the regularity term can be minimized globally by a standard method for convex optimization, provided that the regularity term is convex.

As a result, our method can correctly estimate the motion of small objects over large distances, a problem methods based on linearization of the data term struggle with. We evaluated the parameters our approach thoroughly on a standard test data set, and in addition, we proved that we are able to incorporate patch-based data terms in our formulation, which lose much of their discriminative power under linearization.

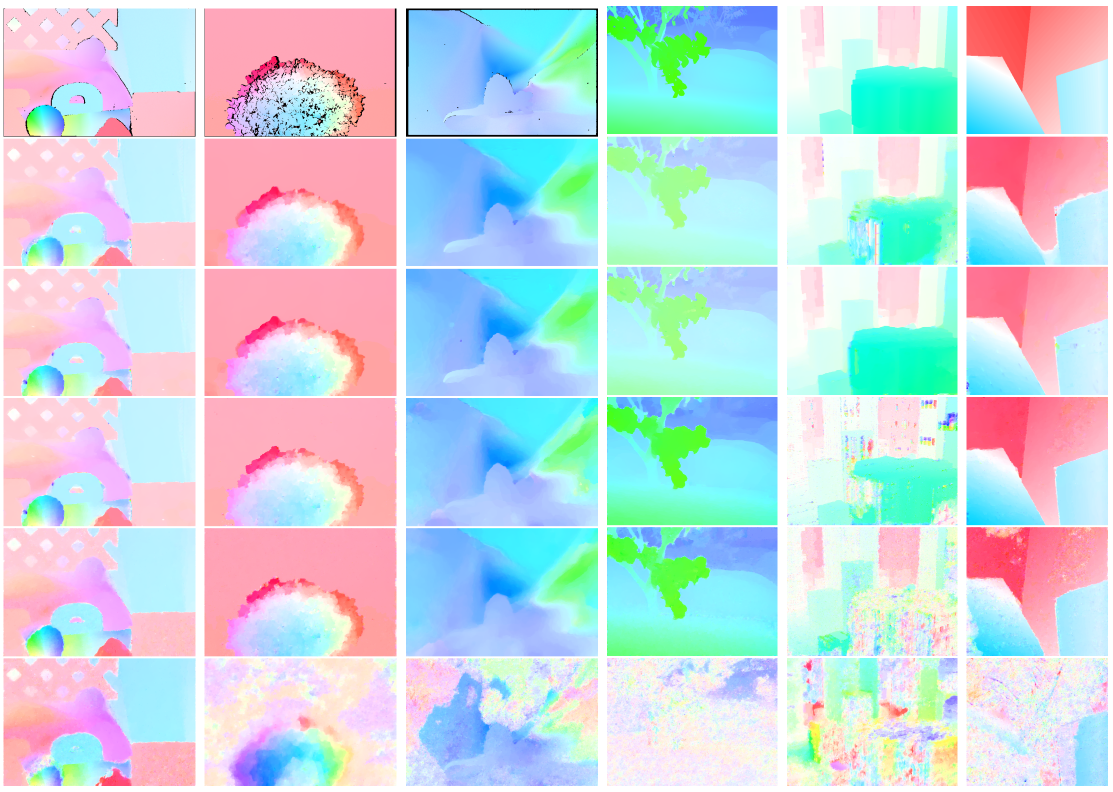


Figure 3.8.: **Qualitative evaluation of advanced data terms.** From top to bottom: Ground truth, SAD dataterm, patch-based SAD dataterm, NCC dataterm, census data term, rank data term.

4. Camera Tracking on RGB-D Sequences

In Chapter 2 we have introduced the concept of photoconsistency in the context of depth/disparity and optical flow estimation. We have proven that optimizing some instances of the standard non-linearized optical flow energy formulation is NP-hard even for convex regularity terms.

In Chapter 3 we have then presented a method for optical flow estimation without the need for linearization or coarse-to-fine warping. While not being able to run in real-time on current commodity hardware, it is able to estimate the optical flow of small-scale structures moving over large distances. Compared to linearization and coarse-to-fine warping based solutions, it therefore serves as an alternative approximation of the original problem, which is computationally infeasible with respect to its asymptotic runtime complexity.

In this chapter we address the problem of visual odometry or camera tracking, i.e. estimating the relative camera pose transformation between each two successive images in a continuous image sequence. As an input we assume a sequence of both intensity and depth images, which can be provided in real-time by recent commodity RGB-D cameras. Other than in the last chapter, the goal is not to find an asymptotically feasible solution to a problem with a variable number of unknowns, but to estimate a fixed number of unknowns in real-time. Mathematically, it is the complementary problem to the one of depth/disparity estimation described in Chapter 2, as we will explain in Section 4.2.1.

4.1. Previous Strategies for Camera Tracking

Visual odometry, i.e. the problem of tracking the pose of a mobile system purely from its visual observations, has a long history in the fields of computer vision and robotics [53, 65]. Previous approaches tackling this problem can be roughly subdivided into two categories:

The first category is comprised of methods using only geometry information. The relative camera pose transformation is estimated as one optimally aligning the geometry observed by the current frame with the geometry of one or several previous frames with respect to some metric. A very popular method for aligning geometry for relative camera pose estimation is the Iterative Closest Point (ICP) algorithm [18, 75], that uses one of the simplest geometry representations, an unoriented point cloud. This point cloud could come for example from triangulation from images, from a LIDAR system, or from a CAD model in a frame-to-model or model-to-model alignment task. ICP finds the closest point in one point cloud for each point in the other, and the estimated camera pose transformation minimizes the alignment error of all point-to-point correspondences. Since this method is highly dependent on the initial estimate of the pose transformation, and the point-to-point correspondences from the initial estimate are not necessarily correct, extensions of this method use point-to-plane assignments rather than point-to-point assignments [77], referred to as Generalized-ICP (GICP). Compared to ICP, GICP needs additional information about the local surface orientation in one of the point clouds. If the point cloud is structured, in that it comes from an dense image (including the spherical image of a LIDAR scan), then the local orientation can be computed from the spatial neighborhood of each point in the image. If not, then the local surface orientation for each point has to be estimated by finding neighbors in the unoriented point cloud itself.

The second category contains methods optimizing photoconsistency in a sparse set of key points or features in the image. The choice of key points is typically driven by a heuristic for detecting image corners, as for example the Shi-Tomasi criterion [79] in [32, 33], the FAST corner detector [71] in [49], or extrema of the trace [56] or the determinant [14] of the image Hessian. The photoconsistency measure, i.e. the descriptor of the key point, is a vectorial measure, usually either an image patch, as for example in [32, 33, 49], or a histogram of gradients in the patch, as for example in [42, 81, 34]. These descriptor-based approaches are less prone to incorrect feature matches than ICP and therefore better suited for relocalization in a global map, or tracking over large distances. However, they are prone to containing many outliers that have to be rejected with methods like Random Sample Consensus (RANSAC) [36]. The dependence on strong outlier rejection can be mitigated by restricting the search for descriptor matches to the local vicinity of the key point location re-projected from one image into the other. However, this re-projection requires an already existing estimate of the 3D point on the geometry surface corresponding to the key point. While the reduction to sparse key points speeds up computation time considerably, much relevant information about the scene is lost. For example, a point on a white wall does not provide sufficient information to be uniquely matched to another point (see Equation (3.8)). However, assuming reasonably good lighting conditions, it bears the information that it should not be matched against a point on a black surface. Using the high computational power of recent GPUs, dense localization methods as in [55] and [63] make use of this information.

4.2. A Direct Dense Method for Camera Tracking on RGB-D Image Sequences

Similar to dense photometric stereo solutions, recent consumer RGB-D cameras provide an RGB color image as well as a dense depth image in real-time by actively illuminating the observed geometry. Most cameras either rely on correspondence estimation in a structured light pattern, as in the popular Kinect camera developed by PrimeSense, or on the time-of-flight principle, where the depth is measured by the time it takes for an emitted light pulse to be reflected into a receiver. In contrast to passive photometric stereo solutions, these active RGB-D cameras are robust against untextured environments, they achieve real-time frame rates, and they require little to no computational overhead from the application using them. In addition they are comparatively mobile - in contrast to many LIDAR systems, for example. On the one hand, camera tracking method based on RGB-D cameras has the benefit of getting both dense texture and geometry information as inputs for free. On the other hand, the real-time and light-weight nature of RGB-D cameras typically impose real-time and light-weight requirements on the tracking method as well. A method estimating the camera trajectory in real-time and on portable hardware has a wide range of applications, such as robotic navigation, simultaneous localization and mapping (SLAM), and the real-time reconstruction of 3D geometry.

In this section we propose such a method, that provides frame-to-frame tracking without building a joint environment model and therefore requires only a constant amount of memory, runs in real-time on a standard desktop CPU, and, in contrast to previous methods described above, uses both dense texture and geometry information in a sequence of RGB-D images.

4.2.1. Energy Formulation

The geometry representation we get from the RGB-D camera is the same we introduced in Equation (2.13b), i.e. a dense depth field $h : \Omega \to \mathbb{R}_{>0}$ describing the projection of the radial distance of the surface point projected onto the optical axis in every pixel. Inverting the pinhole camera projection model of Equation (2.4), we get a 3D surface $S_h : \Omega \to \mathbb{R}^3$

$$S_h(\mathbf{p}) = \mathbf{K}^{-1}h(\mathbf{p})\tilde{\mathbf{p}} = \begin{bmatrix} \frac{x-c_x}{f_x}h(\mathbf{p}) \\ \\ \frac{y-c_y}{f_y}h(\mathbf{p}) \\ \\ h(\mathbf{p}) \end{bmatrix}. \tag{4.1}$$

Given such a surface, an optimal camera pose transformation $(\mathbf{R}^*, \mathbf{T}^*)$ between two images $\mathcal{I}_0$ and $\mathcal{I}_1$ achieves perfect photoconsistency in every pixel,

$$\forall \mathbf{p} \in \Omega : \quad \mathcal{I}_1\left(\pi\left(\mathbf{R}^* S_h(\mathbf{p}) + \mathbf{T}^*\right)\right) - \mathcal{I}_0(\mathbf{p}) = 0. \tag{4.2}$$

The standard approach for finding the pose is the assumption of a Gaussian noise model and a least-squares formulation summing up the constraints of all pixels,

$$E\left(\mathbf{R}, \mathbf{T}\right) = \int_\Omega \phi\left(\left|\mathcal{I}_1\left(\pi\left(\mathbf{R}S_h(\mathbf{p}) + \mathbf{T}\right)\right) - \mathcal{I}_0(\mathbf{p})\right|^2\right) \mathrm{d}\mathbf{p}. \tag{4.3}$$

Equation (4.2) is an instance of the general motion model in Equation (2.19), with $\mathbf{w}(\mathbf{p}, t_0) = \mathbf{p}$ and $\mathbf{w}(\mathbf{p}, t_1) = \pi\left(\mathbf{R}^* S_h(\mathbf{p}) + \mathbf{T}^*\right)$. More precisely, it is the same equation (2.13a) describing image intensity based photoconsistency under a rigid camera motion we introduced in Section 2.4.1. The only difference is that we want to estimate the camera pose transformation instead of the depth/disparity values. Photoconsistency based camera tracking on intensity and depth images is therefore related to both optical flow estimation and stereo estimation. Instead of estimating a displacement vector in every pixel with either one or two degrees of freedom, we are estimating a motion represented with only a few degrees of freedom, that describes the displacement vectors in all pixels matching one image to the next. In the following sections, we provide a detailed derivation of both this representation of the camera motion and the optimization scheme to estimate it.

4.2.2. Minimal Camera Pose Representation

While the translation vector $\mathbf{T}$ in Equation (4.3) compactly describes the three degrees of freedom of the camera pose translation, the rotation matrix $\mathbf{R}$ with its nine values lies on an only three dimensional manifold as well, with two degrees of freedom for the rotation axis and one for the rotation angle. Instead of estimating a full 3×3 matrix and projecting it into the Special Orthogonal Group $SO(3)$, we represent rotations by means of the corresponding Lie algebra $so(3)$. Furthermore, we represent the entire Special Euclidean Group $SE(3)$ by its Lie algebra of twists $se(3)$. In the following, we will give a brief introduction of this concept, derived from the one in [59], to which we refer for a more detailed description.

Instead of representing a 3D rotation by its axis in polar coordinates and a rotation angle around this axis, an equally compact representation is a vector $\omega \in \mathbb{R}^3$ that is the product of the normalized rotation axis in 3D Euclidean coordinates and the rotation angle. We denote the $\hat{\cdot}$-operator by the mapping of $\mathbb{R}^3$ into the Lie algebra of skew symmetric matrices,

$$\hat{} : \mathbb{R}^3 \to so(3): \qquad \begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\end{bmatrix} \mapsto \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & \omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{4.4}$$

The multiplication with this skew-symmetric matrix is equivalent to the cross product in $\mathbb{R}^3$:

$$\forall\ \mathbf{V} \in \mathbb{R}^3: \qquad \hat{\omega}\mathbf{V} = \omega \times \mathbf{V}. \tag{4.5}$$

From this equivalence, the following three equalities for unit vectors $\omega \in \mathcal{S}^2$ can be easily

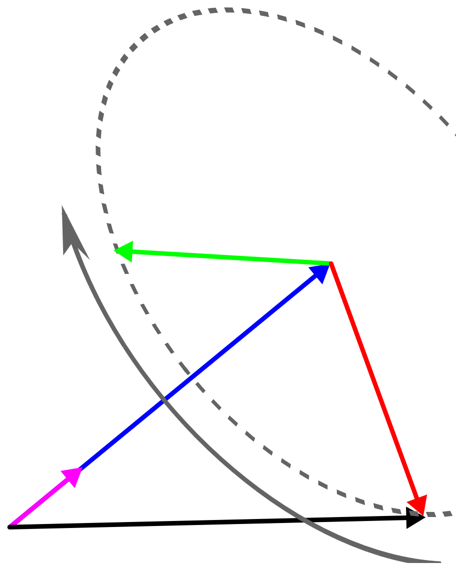


Figure 4.1.: Geometric interpretation of Equation (4.8). Black: Given vector $\mathbf{V}$. Purple: Normalized rotation axis ω. Blue: Projected vector $\omega\omega^{\mathsf{T}}\mathbf{V}$. Red: Cosine part $\left(\mathbf{I}-\omega\omega^{\mathsf{T}}\right)\mathbf{V}$. Green: Sine part $\hat{\omega}\mathbf{V}$.

verified:

$$\hat{\omega}^2 = \omega\omega^{\mathsf{T}} - \mathbf{I}, \tag{4.6a}$$
$$\hat{\omega}^3 = -\hat{\omega}, \tag{4.6b}$$
$$\hat{\omega}^4 = -\hat{\omega}^2. \tag{4.6c}$$

Using these three equalities, we can prove, that for a vector $\omega \in \mathbb{R}^3$ the exponential map of the corresponding skew symmetric matrix, defined by the Taylor series of the matrix exponential,

$$\exp : so(3) \to SO(3), \qquad \hat{\omega} \mapsto \exp(\hat{\omega}) = \sum_{k=0}^{\infty} \frac{\hat{\omega}^k}{k!} \tag{4.7}$$

yields a rotation matrix $\mathbf{R} \in SO(3)$, which rotates any vector $\mathbf{V} \in \mathbb{R}^3$ around ω by an angle of $\|\omega\|$. For sake of readability we write t for $\|\omega\|$ and ω for $\frac{\omega}{\|\omega\|}$. Due to the equalities in (4.6) we can then separate the even and odd indices of the matrix exponential into a sine part and a cosine part:

$$\begin{aligned}\sum_{k=0}^{\infty}\frac{(\hat{\omega}t)^k}{k!} &\overset{(4.6b)(4.6c)}{=} \mathbf{I}+\hat{\omega}\sum_{k=0}^{\infty}\left((-1)^k\frac{t^{2k+1}}{(2k+1)!}\right)-\hat{\omega}^2\sum_{k=1}^{\infty}\left((-1)^k\frac{t^{2k}}{(2k)!}\right)\\ &= \mathbf{I}+\hat{\omega}\sin(t)+\hat{\omega}^2\left(1-\cos(t)\right)\\ &\overset{(4.6a)}{=} \omega\omega^\top+\hat{\omega}\sin(t)+\left(\mathbf{I}-\omega\omega^\top\right)\cos(t). \qquad (4.8)\end{aligned}$$

This closed form of the matrix exponential of skew-symmetric matrices is known as the Rodrigues formula. With this formula, we can verify the exponential map for rotation matrices in a geometric way. Figure 4.1 depicts the composition of the rotation matrix according to the Rodrigues formula: The multiplication of the matrix exponential with a vector $\mathbf{V}\in\mathbb{R}^3$ yields the sum of three vectors defining the rotation plane: $\omega\omega^\top\mathbf{V}$ is the orthogonal projection of $\mathbf{V}$ onto ω and the position vector of the rotation plane. $(\mathbf{I}-\omega\omega^\top)\mathbf{V}$ and $\hat{\omega}\mathbf{V}$ are the two orthogonal vectors spanning the rotation plane with their coefficients $\cos(t)$ and $\sin(t)$. For a non-normalized rotation axis ω, we get the matrix exponential

$$\exp(\hat{\omega})=\mathbf{I}+\hat{\omega}\frac{\sin\left(\|\omega\|\right)}{\|\omega\|}+\hat{\omega}^2\frac{1-\cos\left(\|\omega\|\right)}{\|\omega\|^2}. \qquad (4.9)$$

To compute the rotation axis and angle from an $SO(3)$ rotation matrix, we follow [35], and use the fact that every rotation matrix $\mathbf{R}$ can be uniquely decomposed into a symmetric part $\frac{\mathbf{R}+\mathbf{R}^\top}{2}$ and a skew-symmetric part $\frac{\mathbf{R}-\mathbf{R}^\top}{2}$. In Equation (4.9), both $\mathbf{I}$ and $\hat{\omega}^2$ are symmetric, while $\hat{\omega}$ is skew-symmetric. Therefore, we have

$$\hat{\omega}\frac{\sin(\|\omega\|)}{\|\omega\|}=\frac{\mathbf{R}-\mathbf{R}^\top}{2}. \qquad (4.10a)$$

We can invert the $\hat{\cdot}$-operator and get a vector z:

$$\omega\frac{\sin(\|\omega\|)}{\|\omega\|}=\frac{1}{2}\begin{bmatrix}R_{32}-R_{23}\\ R_{13}-R_{31}\\ R_{21}-R_{12}\end{bmatrix}=:z. \qquad (4.10b)$$

For the identity rotation matrix the rotation axis is undefined. Otherwise, we can factor out the normalized rotation axis $\frac{z}{\|z\|}=\frac{\omega}{\|\omega\|}$ and get

$$\sin(\|\omega\|)=\|z\| \quad\Rightarrow\quad \|\omega\|=\arcsin(\|z\|), \qquad (4.10c)$$

and by substituting $\|\omega\|$ in Equation (4.10b), we get

$$\omega=z\frac{\arcsin(\|z\|)}{\|z\|}. \qquad (4.10d)$$

Similar to rotations $\mathbf{R} \in SO(3)$ we can also represent rigid body motions $\mathbf{M}$ of the special Euclidean group $SE(3)$ by 6D vectors ξ. To that end, we redefine the $\hat{\cdot}$-operator to map a 6D vector ξ into the Lie algebra $se(3)$ of twist coordinates:

$$\hat{\cdot}: \mathbb{R}^6 \to se(3): \qquad \xi \mapsto \hat{\xi} \quad \Leftrightarrow \quad \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mapsto \begin{bmatrix} 0 & -\omega_3 & \omega_2 & v_1 \\ \omega_3 & 0 & -\omega_1 & v_2 \\ -\omega_2 & \omega_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}. \tag{4.11}$$

The exponential map for elements of $so(3)$ can be adapted to elements of $se(3)$ to map into the special Euclidean group $SE(3)$. Given a twist $\hat{\xi} \in se(3)$, the matrix exponential $\exp(\hat{\xi})$ yields a matrix $\mathbf{M}$ representing a rigid body motion in homogeneous coordinates

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ 0 & 1 \end{bmatrix} = \sum_{k=0}^{\infty} \frac{\hat{\xi}^k}{k!},$$

where $\mathbf{T}$ has the closed-form solution

$$\mathbf{T} = \begin{cases} \left(\mathbf{I} + \frac{1-\cos(\|\omega\|)}{\|\omega\|^2} \hat{\omega} + \frac{\|\omega\| - \sin(\|\omega\|)}{\|\omega\|^3} \hat{\omega}^2 \right) v & : \|\omega\| \neq 0 \\ v & : \|\omega\| = 0 \end{cases} \tag{4.12}$$

The exponential of the $\hat{\omega}$ follows from Equations (4.6) to (4.9). Concerning the translation $\mathbf{T}$, we see from splitting the matrix exponential of $\hat{\xi}$ into three parts,

$$\sum_{k=0}^{\infty} \frac{\hat{\xi}^k}{k!} = \mathbf{I} + \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} + \sum_{k=2}^{\infty} \frac{1}{k!} \begin{bmatrix} \hat{\omega}^k & \hat{\omega}^{k-1} v \\ 0 & 0 \end{bmatrix}, \tag{4.13}$$

that in the trivial case of $\|\omega\| = 0$, we have $\mathbf{T} = v$. As in the last proof, we write $t := \|\omega\|$, $\omega := \frac{\omega}{\|\omega\|}$, and $v := \frac{v}{\|\omega\|}$, to normalize ω factor t out of $\hat{\xi}$. Then we note that the following equality holds for any invertible 4×4 matrix g:

$$g^{-1} \exp(g \hat{\xi} g^{-1} t) g = g^{-1} \sum_{k=0}^{\infty} \left(\frac{\left(g \hat{\xi} g^{-1} t \right)^k}{k!} \right) g = g^{-1} \sum_{k=0}^{\infty} \left(\frac{g \left(\hat{\xi} t \right)^k g^{-1}}{k!} \right) g = \exp(\hat{\xi} t). \tag{4.14}$$

With this equality, we can draw a translation orthogonal to ω and v out of the matrix exponential, and find a closed form for $\mathbf{T}$:

$$\begin{aligned}
\exp\left(\begin{bmatrix}\hat{\omega} & v\\ 0 & 0\end{bmatrix} t\right) &\overset{(4.14)}{=} \begin{bmatrix}\mathbf{I} & \hat{\omega}v\\ 0 & 1\end{bmatrix}\left(\sum_{k=0}^{\infty}\frac{t^k}{k!}\left(\begin{bmatrix}\mathbf{I} & -\hat{\omega}v\\ 0 & 1\end{bmatrix}\begin{bmatrix}\hat{\omega} & v\\ 0 & 0\end{bmatrix}\begin{bmatrix}\mathbf{I} & \hat{\omega}v\\ 0 & 1\end{bmatrix}\right)^k\right)\begin{bmatrix}\mathbf{I} & -\hat{\omega}v\\ 0 & 1\end{bmatrix}\\
&\overset{(4.6a)}{=} \begin{bmatrix}\mathbf{I} & \hat{\omega}v\\ 0 & 1\end{bmatrix}\left(\sum_{k=0}^{\infty}\frac{t^k}{k!}\begin{bmatrix}\hat{\omega} & \omega\omega^\intercal v\\ 0 & 0\end{bmatrix}^k\right)\begin{bmatrix}\mathbf{I} & -\hat{\omega}v\\ 0 & 1\end{bmatrix}\\
&\overset{(4.13)}{=} \begin{bmatrix}\mathbf{I} & \hat{\omega}v\\ 0 & 1\end{bmatrix}\begin{bmatrix}\exp(\hat{\omega}t) & \omega\omega^\intercal tv\\ 0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{I} & -\hat{\omega}v\\ 0 & 1\end{bmatrix}\\
&= \begin{bmatrix}\exp(\hat{\omega}t) & \left((\mathbf{I}-\exp(\hat{\omega}t))\,\hat{\omega}+\omega\omega^\intercal t\right)v\\ 0 & 1\end{bmatrix}
\end{aligned} \tag{4.15}$$

Removing the normalization of the rotation axis ω, we get

$$\mathbf{T} = \frac{\left((\mathbf{I}-\exp(\hat{\omega}))\,\hat{\omega}+\omega\omega^\intercal\right)v}{\|\omega\|^2}, \tag{4.16}$$

and with the Rodrigues formula (4.8) and Equations (4.6) we get Equation (4.12).

While $so(3)$-parametrization of the rotation is an intuitive and minimal representation, the parametrization of the rigid body motion ($\mathbf{R}$,$\mathbf{T}$) by a twist at first seems less intuitive than a simple parametrization of the translation $\mathbf{T}$ in a canonical basis. However, with the twist parametrization, we have the concept of a continuous and differentiable rigid body motion. Similar to the general model for dynamic motion introduced in Chapter 2.4.2 we can define a rigid body motion model depending on a temporal parameter t:

$$G : \mathbb{R}^6 \times \mathbb{R}_{\geq 0} \times \mathbb{R}^3 \to \mathbb{R}^3, \quad (\xi, t, \mathbf{P}) \mapsto \mathbf{R}(\xi, t)\mathbf{P} + \mathbf{T}(\xi, t). \tag{4.17}$$

With $\xi = \begin{bmatrix}\omega^\intercal & v^\intercal\end{bmatrix}^\intercal$ as in (4.11), we get

$$\mathbf{R}(\xi, t) = \exp(\hat{\omega}t) \quad \text{and} \quad \mathbf{T} = \begin{cases}\frac{\left((\mathbf{I}-\exp(\hat{\omega}t))\hat{\omega}+\omega\omega^\intercal t\right)vt}{\|\omega\|^2 t} & : \|\omega\|t \neq 0\\ vt & : \|\omega\|t = 0\end{cases}. \tag{4.18}$$

4.2.3. Energy Linearization and Minimization

With the compact representation of camera poses introduced in the last section, we can reformulate the problem of estimating the relative camera pose transformation between the recording of two RGB-D images analogously to the problem of optical flow and disparity estimation in the last chapter. Given two images taken at t_0 and t_1, the problem of camera pose estimation in Equation (4.3) can now be reformulated as

$$\xi^* = \underset{\xi \in \mathbb{R}_{se(3)}}{\arg\min}\left\{\int_{\Omega} \phi\left(\left|\mathcal{I}\left(\pi\left(G\left(\xi, t_1, S_h(\mathbf{p})\right)\right), t_1\right) - \mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_0\right)\right|^2\right) \mathrm{d}\mathbf{p}\right\}, \tag{4.19}$$

where we use the notation $\mathbb{R}_{se(3)}$ for the set

$$\mathbb{R}_{se(3)} = \left\{ \xi = \begin{bmatrix} \omega^\mathsf{T} & v^\mathsf{T} \end{bmatrix}^\mathsf{T} \;\middle|\; \omega, v \in \mathbb{R}^3, \quad |\omega| < 2\pi \right\} \tag{4.20}$$

to get a unique minimum. In the following, we set t_0 to 0, so $\mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_0\right)$ is reduced to $\mathcal{I}\left(\mathbf{p}, t_0\right)$. For optimizing Equation (2.9a) with respect to $\mathbf{R}$ and $\mathbf{T}$ we now have to find an optimal ξ in $B_{\mathbb{R}^3} \times \mathbb{R}^3$. Although the search domain has a fixed dimensionality, a complete search for an optimal transformation in all six degrees of freedom is not tractable, if we require real-time capability of our approach. The photoconsistency measure is neither independent for the elements of ξ, nor is it convex in ξ. Therefore, we rely on a local approach by iteratively linearizing the photoconsistency equation with respect to ξ and therefore approximating Equation (2.20). Furthermore, we use a coarse-to-fine approach, just as it is done for many optical flow implementations. In contrast to the optical flow problem however, we do not care whether the coarse-to-fine approach eliminates fine-scaled structures on coarse scales, since we are not interested in reconstructing fine details but only the global camera motion, which is mainly defined on coarse image scales. However, in cases where the camera motion is only distinguishable from a sparse set of fine-scaled points moving over large distances from one image to the next, our approach is inferior to feature-based methods.

Applying the spatial linearization of the image $\mathcal{I}_1$ with respect to t to Equation (4.19), and applying the chain rule, we get

$$\begin{aligned} & \mathcal{I}\left(\pi\left(G\left(\xi, t_1, S_h(\mathbf{p})\right)\right), t_1\right) \\ \approx \; & \mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_1\right) + \\ & \left(\frac{\mathrm{d}\mathcal{I}\left(\pi\left(G\left(\xi, t, \mathbf{P}\right)\right), t_1\right)}{\mathrm{d}t}\right)\Bigg|_{\pi(G(\xi, t_0, S_h(\mathbf{p})))} \left(t_1 - t_0\right) \\ = \; & \mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_1\right) + \\ & \left(\frac{\mathrm{d}\mathcal{I}(\mathbf{P}, t_1)}{\mathrm{d}\mathbf{p}}\right)\Bigg|_{\pi(G(\xi, t_0, S_h(\mathbf{p})))} \left(\frac{\mathrm{d}\pi}{\mathrm{d}\mathbf{P}}\right)\Bigg|_{G(\xi, t_0, S_h(\mathbf{p}))} \left(\frac{\mathrm{d}G}{\mathrm{d}t}\right)\Bigg|_{\xi, t_0, S_h(\mathbf{p})} \left(t_1 - t_0\right). \end{aligned} \tag{4.21}$$

In an analogous way to the linearization at different points as in Equations (3.1), (3.4), and (3.5), we can also apply a linearization completely at t_0:

$$\begin{aligned}
& \mathcal{I}\left(\pi\left(G\left(\xi, t_1, S_h(\mathbf{p})\right)\right), t_1\right) \\
\approx \; & \mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_0\right) + \\
& \left(\frac{\mathrm{d}\mathcal{I}\left(\pi\left(G\left(\xi, t, \mathbf{P}\right)\right), t\right)}{\mathrm{d}t}\right)\Bigg|_{\pi(G(\xi,t_0,S_h(\mathbf{p}))),t_0} (t_1 - t_0) \\
= \; & \mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_0\right) + \\
& \left(\left(\frac{\mathrm{d}\mathcal{I}\left(\pi\left(G\left(\xi, t, \mathbf{P}\right)\right), t_0\right)}{\mathrm{d}t}\right)\Bigg|_{\pi(G(\xi,t_0,S_h(\mathbf{p})))} + \left(\frac{\partial \mathcal{I}}{\partial t}\right)\Bigg|_{t_0}\right)(t_1 - t_0) \\
\overset{(3.4)}{=} \; & \mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_1\right) + \qquad (4.22) \\
& \left(\frac{\mathrm{d}\mathcal{I}(\mathbf{P}, t_0)}{\mathrm{d}\mathbf{p}}\right)\Bigg|_{\pi(G(\xi,t_0,S_h(\mathbf{p})))} \left(\frac{\mathrm{d}\pi}{\mathrm{d}\mathbf{P}}\right)\Bigg|_{G(\xi,t_0,S_h(\mathbf{p}))} \left(\frac{\mathrm{d}G}{\mathrm{d}t}\right)\Bigg|_{\xi,t_0,S_h(\mathbf{p})} (t_1 - t_0)\,.
\end{aligned}$$

where the only difference to Equation (4.21) is, that the image gradient is now computed on $\mathcal{I}_0$. From the differentiation rule of the exponential function we have the equality

$$\left(\frac{\mathrm{d}G}{\mathrm{d}t}\right)\Bigg|_{\xi,t,\mathbf{P}} = \hat{\omega}\mathbf{R}(\xi, t)\mathbf{P} + \hat{\omega}\mathbf{T}(\xi, t) + v, \qquad (4.23)$$

and applying this equality to the derivative formulation, while omitting the indices in the derivative chain for better readability, we get

$$\begin{aligned}
& \mathcal{I}\left(\pi\left(G\left(\xi, t_1, S_h(\mathbf{p})\right)\right), t_1\right) \qquad (4.24) \\
\approx \; & \mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_1\right) + \nabla\mathcal{I}^{\mathsf{T}} \frac{\mathrm{d}\pi}{\mathrm{d}\mathbf{P}} \left(\hat{\omega}\left(\mathbf{R}(\xi, t_0) S_h(\mathbf{p}) + \mathbf{T}(\xi, t_0)\right) + v\right).
\end{aligned}$$

For $t_0 = 0$, we have $\mathcal{I}\left(\pi\left(G\left(\xi, t_0, S_h(\mathbf{p})\right)\right), t_1\right) = \mathcal{I}\left(\mathbf{p}, t_1\right)$, $\mathbf{R}(\xi, t_0) = \mathbf{I}$, $\mathbf{T}(\xi, t_0) = 0$, and Equation (4.24) is linear in ξ. Therefore, for functions ϕ such as $\phi(s) = s$ or $\phi(s) = \sqrt{s}$, energy (4.19) is convex and it can be globally minimized in ξ.

In our coarse-to-fine scheme, we iteratively estimate incremental motions ξ_{inc} from images warped by the accumulated motion ξ_{acc} from all past iterations. After iteration k, we update the accumulated motion with the estimated increment:

$$\hat{\xi}_{\text{acc}}^{k+1} = \log\left(\exp\left(\hat{\xi}_{\text{inc}}^{k}\right)\exp\left(\hat{\xi}_{\text{acc}}^{k}\right)\right) \qquad (4.25)$$

Given an initial estimate of a pose transformation $(\xi_{\text{acc}}, t_{\text{acc}})$, where we set $t_{\text{acc}} = 1$ for sake of simplicity, we find the incremental transformation ξ_{inc} by minimizing the augmented Equation (4.19):

$$\xi_{\text{inc}}^* = \underset{\xi_{\text{inc}} \in \mathbb{R}_{se(3)}}{\arg\min} \left\{ \int_\Omega \phi \left(\left| \mathcal{I}\left(\pi\left(G\left(\xi_{\text{inc}}, t_1, G\left(\xi_{\text{acc}}, t_{\text{acc}}, S_h(\mathbf{p})\right)\right)\right), t_1\right) - \mathcal{I}\left(\mathbf{p}, t_0\right) \right|^2 \right) \mathrm{d}\mathbf{p} \right\}. \tag{4.26}$$

Using Equation (4.24) with $t_0 = 0$, we get for the linearization

$$\begin{aligned} & \mathcal{I}\left(\pi\left(G\left(\xi_{\text{inc}}, t_1, G\left(\xi_{\text{acc}}, t_{\text{acc}}, S_h(\mathbf{p})\right)\right)\right), t_1\right) \\ \approx\; & \mathcal{I}\left(\pi\left(G\left(\xi_{\text{acc}}, t_{\text{acc}}, S_h(\mathbf{p})\right)\right), t_1\right) \\ +\; & \nabla \mathcal{I}^\mathsf{T} \frac{\mathrm{d}\pi}{\mathrm{d}\mathbf{P}} \left(\hat{\omega_{\text{inc}}} \left(\mathbf{R}(\xi_{\text{acc}}, t_{\text{acc}}) S_h(\mathbf{p}) + \mathbf{T}(\xi_{\text{acc}}, t_{\text{acc}}) \right) + v_{\text{inc}} \right). \end{aligned} \tag{4.27}$$

Using the abbreviations $\mathbf{G}(\mathbf{p}) = G\left(\xi_{\text{acc}}, t_{\text{acc}}, S_h(\mathbf{p})\right)$ and $w(\mathbf{p}) = \pi(\mathbf{G}(\mathbf{p}))$, we get

$$\nabla \mathcal{I}^\mathsf{T}(w(\mathbf{p})) \frac{\mathrm{d}\pi}{\mathrm{d}\mathbf{P}} (\mathbf{G}(\mathbf{p})) \left(\hat{\omega_{\text{inc}}} \mathbf{G}(\mathbf{p}) + v_{\text{inc}} \right). \tag{4.28}$$

This expression is linear in ξ_{inc} and can be written as $\mathcal{C}_{\text{acc}}(\mathbf{p})^\mathsf{T} \xi_{\text{inc}}$ with a 6×1 constraint vector $\mathcal{C}_{\text{acc}}$. For an explicit formulation of this constraint vector we refer to Equation (A.5) in the appendix. With this, Equation (4.3) now becomes

$$(\mathbf{R}^*, \mathbf{T}^*) = \underset{(\mathbf{R}, \mathbf{T}) \in SE(3)}{\arg\min} \left\{ \int_\Omega \phi \left(\left| \mathcal{C}_{\text{acc}}(\mathbf{p})^\mathsf{T} \xi_{\text{inc}} + \mathcal{I}_1(\mathbf{p}) - \mathcal{I}_0(\mathbf{p}) \right|^2 \right) \mathrm{d}\mathbf{p} \right\} \tag{4.29}$$

and the necessary condition for $(\mathbf{R}^*, \mathbf{T}^*)$ to be minimal is given by the normal equation

$$\int_\Omega \phi' \left(\left| \mathcal{C}_{\text{acc}}(\mathbf{p})^\mathsf{T} \xi_{\text{inc}} + \mathcal{I}_1(\mathbf{p}) - \mathcal{I}_0(\mathbf{p}) \right|^2 \right) \left(\mathcal{C}_{\text{acc}}(\mathbf{p}) \mathcal{C}_{\text{acc}}(\mathbf{p})^\mathsf{T} \xi_{\text{inc}} + \mathcal{I}_1(\mathbf{p}) - \mathcal{I}_0(\mathbf{p}) \right) \mathrm{d}\mathbf{p} = 0 \tag{4.30a}$$

or

$$\left(\int_\Omega \phi'_{\text{acc}}(\mathbf{p}) \mathcal{C}_{acc}(\mathbf{p}) \mathcal{C}_{\text{acc}}(\mathbf{p})^\mathsf{T} \,\mathrm{d}\mathbf{p} \right) \xi_{\text{inc}} = \int_\Omega \phi'_{\text{acc}}(\mathbf{p}) \left(\mathcal{I}_0(\mathbf{p}) - \mathcal{I}_1(\mathbf{p}) \right) \mathrm{d}\mathbf{p}. \tag{4.30b}$$

We can now iteratively compute the normal equation and solve the resulting 6×6 equation system iteratively or analytically, resulting in a Gauss-Newton method for relative camera pose estimation.

In the next two sections, we are going to relate method to the ICP method, which is the current standard for depth-based camera tracking. After this, we are going to compare our approach with a quadratic loss function $\phi(s) = s$ against the GICP implementation of Segal et al. [77] on the RGB-D benchmark of Sturm et al. proposed in [83].

4.3. Relation to ICP

Our method for camera pose estimation is related both to the ICP method and to methods based on sparse feature correspondences, as all of them can be formulated using a representation of the camera pose transformation as the exponential of a twist matrix. Given a sequence of depth images, ICP can estimate the camera pose transformation between successive images on a frame-to-frame basis, and it is comparatively free of hyper-parameters. In contrast, odometry methods based on monocular intensity images and feature descriptor matches, as referenced in Section 4.1, have to infer parts of the 3D geometry from several images, and follow a frame-to-model approach, requiring an additional memory and compute overhead. RGB-D image based methods like Endres et al.[34] estimate the camera pose transformation between two point patterns ([86]), just like ICP. However, the estimated point-to-point correspondences are not based on Euclidean distances, but on distances in the descriptor space. Therefore, just like other feature based methods, its performance depends on the correct choice of a feature descriptor, which itself has dependencies on methods for outlier detection and on computational hardware.

The input to ICP is a set of 3D data points, $\{\mathbf{d}_i | 1 \leq i \leq n_d\}$, and a set of model points $\{\mathbf{m}_j | 1 \leq j \leq n_m\}$, where typically n_d is much smaller than n_m. The goal is to find a rigid body motion $(\mathbf{R}, \mathbf{T})$ that optimally maps all data points onto the closest corresponding model points. Under the assumption of a Gaussian error distribution, this yields the energy formulation

$$E_{\text{ICP}}(\mathbf{R}, \mathbf{T}) = \sum_i \min_j \left\{ \langle \mathbf{R}\mathbf{d}_i + \mathbf{T} - \mathbf{m}_j, \mathbf{n}_{ij} \rangle^2 \right\}, \tag{4.31}$$

where different choices of the normal vector $\mathbf{n}_{ij}$ yield different versions of ICP. The choice $\mathbf{n}_{ij} = \frac{\mathbf{R}\mathbf{d}_i + \mathbf{T} - \mathbf{m}_j}{|\mathbf{R}\mathbf{d}_i + \mathbf{T} - \mathbf{m}_j|}$ yields the point-to-point ICP formulation, whereas a normal $\mathbf{n}_{ij}$ corresponding to the local plane around $\mathbf{m}_j$ yields point-to-plane ICP.

ICP alternates the search for point-to-point correspondences and solving for the pose transformation: In every iteration k, it estimates for every data point $\mathbf{d}_i^k$ a corresponding model point $\mathbf{m}_i$ of minimal distance

$$\mathbf{m}_i = \arg\min_j \left\{ \left| \mathbf{d}_i^k - \mathbf{m}_j \right| \right\}, \tag{4.32a}$$

and then updates the pose transformation as

$$(\mathbf{R}^{k+1}, \mathbf{T}^{k+1}) = \arg\min_{(\mathbf{R}, \mathbf{T})} \left\{ \sum_i \left\langle \mathbf{R}\mathbf{d}_i^k + \mathbf{T} - \mathbf{m}_i, \mathbf{n}_i \right\rangle^2 \right\}. \tag{4.32b}$$

Finally, the data points are transformed for the next iteration

$$\mathbf{d}_i^{k+1} = \mathbf{R}^{k+1}\mathbf{d}_i^k + \mathbf{T}^{k+1} \tag{4.32c}$$

After κ alternations, the camera pose transformation $(\mathbf{R}, \mathbf{T})$ is the concatenation of the incremental transformations:

$$\begin{bmatrix} \mathbf{R}^{\kappa} & \mathbf{T}^{\kappa} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}^{\kappa-1} & \mathbf{T}^{\kappa-1} \\ 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} \mathbf{R}^{1} & \mathbf{T}^{1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}^{0} & \mathbf{T}^{0} \\ 0 & 1 \end{bmatrix} \tag{4.33}$$

with $\mathbf{R}^0 = \mathbf{I}$ and $\mathbf{T}^0 = \mathbf{0}$. For step (4.32b) one can use the matrix exponential formulation of Equation (4.12) for $\mathbf{R}$ and $\mathbf{T}$. Using a first-order approximation of the exponential

$$\exp(\hat{\xi}) \approx \mathbf{I} + \hat{\xi}, \tag{4.34}$$

and incorporating an initial motion estimate $(\mathbf{R}_{\text{acc}}, \mathbf{T}_{\text{acc}})$, the energy is now convex in ξ

$$\begin{aligned} &\underset{(\mathbf{R},\mathbf{T})}{\arg\min} \left\{ \sum_i \left\langle (\mathbf{I} + \hat{\omega}) \underbrace{\left(\mathbf{R}_{\text{acc}} \mathbf{d}_i^{k+1} + \mathbf{T}_{\text{acc}} \right)}_{=:\mathbf{G}} + v - \mathbf{m}_i, \mathbf{n}_i \right\rangle^2 \right\} & (4.35) \\ = \; &\underset{(\mathbf{R},\mathbf{T})}{\arg\min} \left\{ \sum_i \left(\begin{bmatrix} (G_Y n_{i3} - G_Z n_{i1}) \\ (G_Z n_{i1} - G_X n_{i3}) \\ (G_X n_{i2} - G_Y n_{i1}) \\ n_{i1} \\ n_{i2} \; n_{i3} \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} - \langle \mathbf{m}_i, \mathbf{n}_i \rangle \right)^2 \right\} & (4.36) \end{aligned}$$

If the data and model point clouds stem from surfaces as in Equation (4.1), we get Equation (4.28) of our formulation by setting $\mathbf{n}_i \equiv \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^{\mathsf{T}}$, projecting both point clouds into their respective images and by penalizing the image intensity differences. To keep the energy convex in ξ, we need to make the same two linear approximations for the image intensity and for the perspective projection.

Though both our method and ICP use the same parametrization of camera motion, our method has a key advantage compared to ICP, since the step of finding the closest model point for each data point in Equation (4.32a) is computationally cumbersome. A naïve implementation has a quadratic runtime complexity, and many implementations rely on space partitioning trees to accelerate this task. However, the tree has to be recomputed for every new image pair, and the runtime complexity of an iteration is linear-logarithmic, while the runtime complexity of our method remains linear, rendering our method faster than ICP.

Another difference between our method and ICP are the different types of failure cases: A trivial failure case of our method is an untextured geometry, where $\nabla \mathcal{I} \equiv 0$. In this case ICP can still find the correct camera pose transformation, if the structure of the geometry contains sufficient information. Our method also relies on an additional linearization of the perspective projection and the image, resulting in tracking failures over large distances, where the linear approximations are incorrect. On the other hand, our method can find the correct camera pose transformation on textured planar surfaces like walls or floors, while ICP can only estimate 4 of the desired 6 degrees of freedom. Also, in many indoor scenes suited for active RGB-D sensors, the high frequency texture information, which causes tracking failures over large distances, enables our method to be more precise for small camera pose transformations, as it is confirmed by our experimental evaluation in the next section.

4.4. Experimental Evaluation

We evaluate our method for camera tracking on the Freiburg RGB-D dataset of Sturm et al [83]. This dataset contains RGB-D images from a Microsoft Kinect sensor with synchronized camera poses from an external motion capture system. Figure 4.2 shows a qualitative assessment of our method on the fr2/desk sequence. The first row shows the reference image in each column for comparison. The second row shows different images that are tracked against the reference image, with increasing time and distance of the camera pose from left to right. The third row shows the reconstructed reference frame using the estimated camera motion. The fourth row shows the difference image between the original and the reconstructed reference frame. As we can see, our method is capable of accurately estimating the camera motion over a substantial distance, but fails if the distance between two images becomes too large, as it gets stuck in a local minimum of the energy.

We also use the Freiburg RGB-D dataset to quantitatively evaluate our method against a reference implementation of the GICP method of Segal et al in [77]. From the large variety of different sequences, we choose the fr1/desk and fr2/desk sequences for our experiments, as they contain both translational and rotational motions in a typical office environment at different speeds. To simulate larger camera velocities, we leave out intermediate frames in the sequence. In particular, we matched $I(t)$ and $I(t+k)$ for different $k = 1, \dots, 20$ and measured for all t the error between our motion estimate and the motion from the ground truth. Figure 4.3 shows the median error with respect to k. In particular, we found that our approach outperforms GICP when k is small, i.e., $k < 5$ for freiburg1/desk and $k < 17$ for freiburg2/desk. Note that the average camera speed in freiburg1/desk is much higher than in freiburg2/desk. From this result, we conclude that our approach is well suited for continuous camera tracking, while GICP can better deal with larger displacements.

The superior performance of our method on smooth and continuous camera trajectories is also reflected in the drift-per-frame evaluation of these two sequences, see Table 4.1 and Figure 4.4b. Our approach has a median translational drift of 5.3mm and 1.5mm while GICP drifts by 10.5mm and 6.2mm per frame, respectively. We analyzed this result further by computing the error histogram shown in Figure 4.4b: This plot confirms

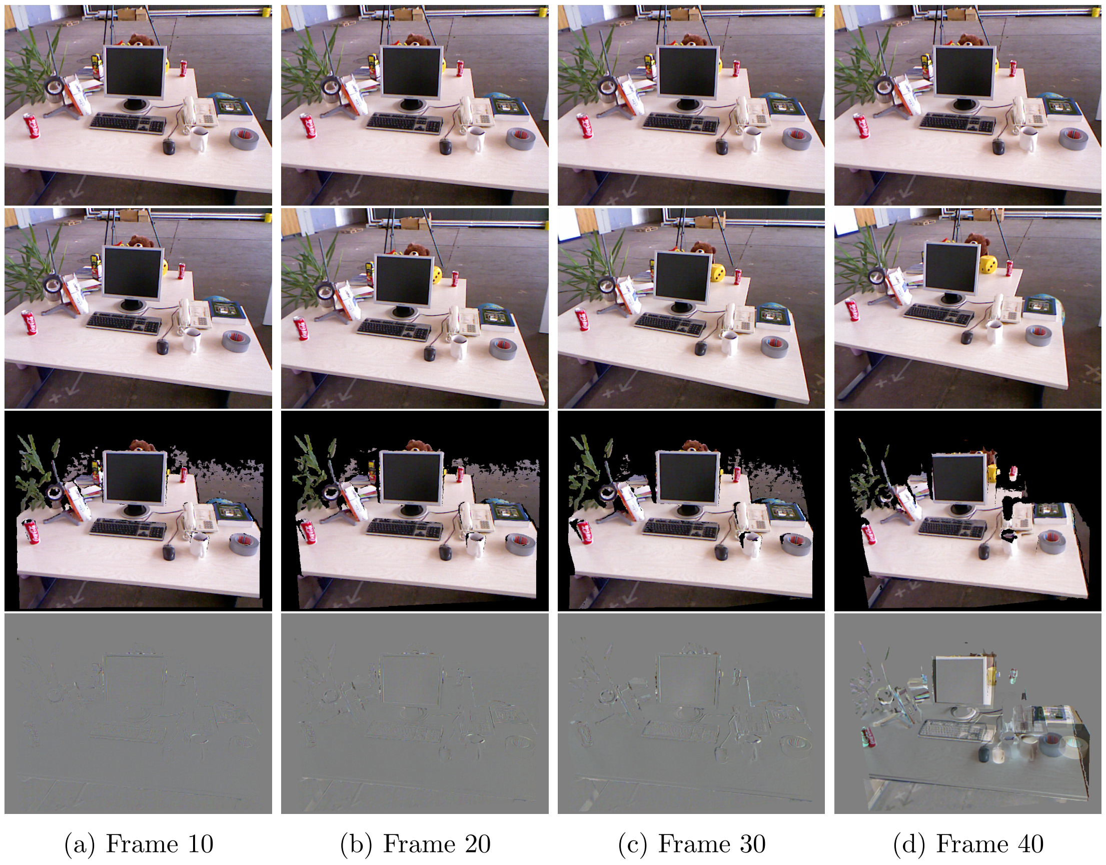
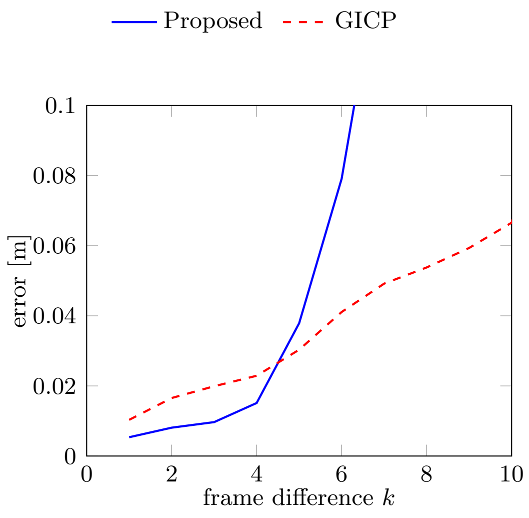


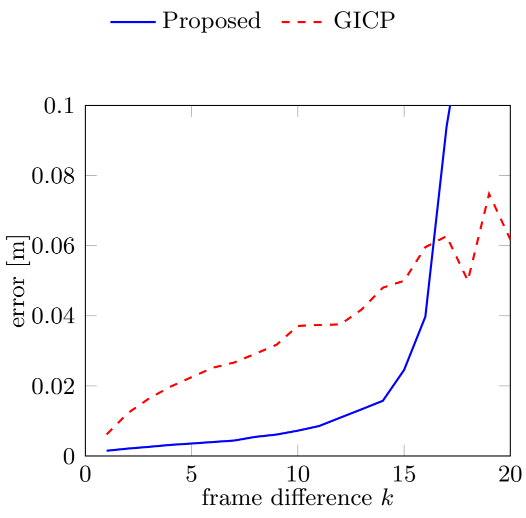
Figure 4.2.: **Camera motion estimation over increasing distances.** From top to bottom: Reference image 1, image tracked against the reference image, reconstructed reference image, difference image. Our algorithm is capable of tracking the camera over a substantial distance/time, but fails if the distance is too large.

Dataset	Proposed	GICP	Improvement
freiburg1/desk	0.0053 m	0.0103 m	1.94x
	0.0065 deg	0.0154 deg	2.37x
freiburg2/desk	0.0015 m	0.0062 m	4.13x
	0.0027 deg	0.0060 deg	2.22x

Table 4.1.: Comparison of the drift per frame of our approach versus GICP on two different datasets. The values give the median. Our approach achieves more than 50% better pose estimates than GICP.



(a) freiburg1/desk dataset



(b) freiburg2/desk dataset

Figure 4.3.: Pose accuracy under increasing frame differences, i.e., we match $I(x,t)$ and $I(x,t+k)$ for all t. For not too large inter frame differences, the proposed method gives more accurate results while GICP is more robust against larger displacements.

that our approach has considerably lower pose errors than GICP. Additionally, our approach has fewer outliers. We found that GICP has in 15.3% of all frames an error larger than 1cm, while our approach exceeds 1cm error only in one of the 2070 frames in the fr2/desk sequence.

Our method also outperforms the reference GICP implementation with respect to speed. On a single Intel Xeon E5520 CPU with 2.27GHz, we measured that our approach takes an average of 8ms per frame (varying slightly with the number of pixels with valid depth values), while the standard GICP implementation takes an average of 7.5s per match. This means that our approach is able to provide visual odometry in real-time at 12.5Hz with our relatively naïve implementation. We suppose that frame rates surpassing the one of the sensor can easily be achieved by using SIMD instructions available on current commodity CPUs, and by possibly trading accuracy for speed by taking into account only a subset of all available pixels with valid depth values.

4.5. Conclusion

We have introduced a direct method for estimating the relative camera pose transformations between two images of a static scene. By optimizing photoconsistency, it is related to the disparity estimation introduced in Section 2.4.1. While previous methods like ICP

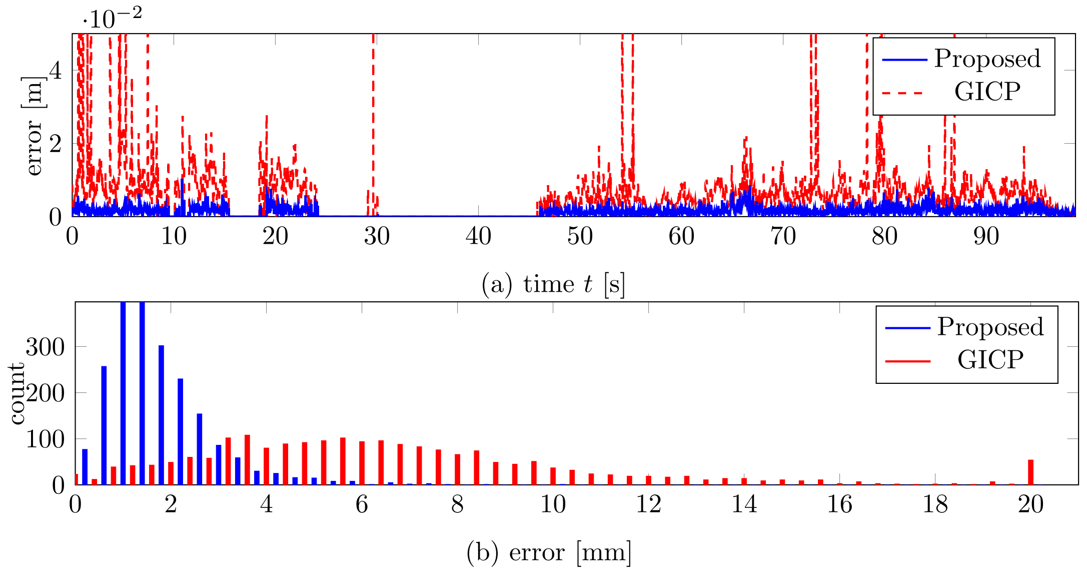


Figure 4.4.: Per-frame error (top) and error histogram (bottom) on the fr2/desk sequence. We found that our approach has both a lower median error and fewer outliers in comparison to GICP. Note that the ground truth information is partially missing due to occlusions in the scene.

or sparse feature matching pre-compute point-to-point correspondences, based on spatial 3D proximity or photoconsistency, our methods estimates the pose transformation directly as a minimizer of a photoconsistency energy. In the context of dense RGB-D image sequences provided by novel commodity sensors, our method is able to use the dense information provided in both texture and depth, and it allows for frame-to-frame tracking without an additional mapping back-end. Without the need for a computationally cumbersome step of point-to-point correspondence estimation, the proposed method is significantly faster than a reference ICP implementation, and it enables real-time camera tracking on a single core of a commodity laptop CPU. Using both depth and detailed texture information, our method outperforms the reference ICP implementation with respect to pose error for slower continuous motions, while losing track earlier then ICP in the presence of large camera motions between consecutive images.

5. Large-Scale 3D Reconstruction from RGB-D Sequences

In this chapter we cover the problem of dense geometry mapping. Given a sequence of RGB-D images of a static scene, as well as the corresponding camera poses, we want to reconstruct the joint textured geometry observed in these images in global coordinates. The input sequence of RGB-D images can come from dense disparity estimation as outlined in Chapters 2 and 3, or from an active RGB-D sensor. The camera poses can come from inside-out methods similar to the one presented in 4, or they could be provided by an external global motion tracking system as in [83]. Like our camera tracking method in the last chapter, we want our geometry reconstruction method to be real-time capable on consumer hardware - potentially even on mobile or wearable hardware - and we want it to be scalable to large environments beyond the capability of previous methods, while keeping fine-scale details. This limits our choices of input data, and we use an active Kinect1 / Xtion Pro RGB-D sensor in combination with a derivative of the camera tracking method from the last chapter [48, 47] for our experiments. In the following, we abstract the methods used for creating the input data, and just assume it to be given.

In addition to be real-time capable and scalable, we require our geometry reconstruction method to be an online method, meaning that not all the data has to be given at the beginning of our method, and we require a real-time visualization method capable of running on mobile and potentially remote platforms. We are going to explain the former in greater detail in the next three sections, and the latter in Section 5.4.

5.1. Surface Representations and Prior Work

A naïve approach for a geometry reconstruction is creating a joint point cloud of all depth images. Point clouds are one of the most intuitive global map representations. They are featured in many popular approaches not necessarily restricted to those based on dense RGB-D information but rather a sparse set of visual feature points matched by their corresponding descriptors [7]. This sparse set of features can be combined with a volumetric map representation as well, as for example in the work of Endres et al. in [34].

However, point clouds lack information of the local neighborhood of each point, and do not represent dense, continuous surfaces. The most popular dense surface representation is a triangle mesh. When created from a point cloud, every point on the continuous surface is interpolated linearly between 3 points of the cloud. Meshes can be stored easily and efficiently, and they are the standard for 3D geometry visualization running on dedicated graphics hardware.

Reconstructing the joint point cloud from a sequence of depth images creates the problem that the memory required to store this point cloud scales linearly with the number of processed depth images. While this is a problem for all reconstruction methods in the worst case - when all images capture completely disjoint parts of the geometry - the linear scaling is independent of the observed geometry for the naïve approach. For example, if all images and camera poses are identical, because the camera does not move, the resulting point cloud would be highly redundant with multiple points on the same position. With a 3×4 Byte floating-point value for the 3D position and a 3×1 Byte RGB color value for each point, a sequence of 1000 images of VGA resolution would already result in a point cloud of more than 4.2 GB, assuming that all pixels in all images have a valid depth value.

The next two plausible steps would be the removal or the averaging of points that lie either on the same 3D position as other points, or in their close vicinity, and the triangulation of a surface mesh from the remaining points. The naïve implementation for removing redundant points from the cloud involves computing the nearest neighbor for each new 3D point and successively averaging neighboring points until the distance between any two points in the cloud is larger than some threshold. Without any special data structure, the search for nearest neighbors results in a total runtime complexity quadratic in the number of points, which becomes infeasible very fast. Finding neighbors of points for the triangle surface computation has the same problem. To accelerate this approach, the standard solution is a nearest neighbor search accelerated by binary space partitioning tree, typically an axis-aligned k-d tree ($k = 3$), where each leaf contains references to points in the cloud. Such a k-d tree is an example of a volumetric structure, which is required for fast access to geometry elements located at a given 3D position. Other examples of volumetric structures are 3d grids and octrees, that we are going to investigate below.

Even with such a volumetric acceleration structure, computing a triangle mesh representing a 3D surface from a point cloud is not trivial. Greedy algorithms such as the one outlined above can produce ill-formed elongated triangles or triangle crossings. Approaches based on the popular Voronoi diagrams [9] and the geometrically dual Delaunay triangulations [21] can overcome the problems of ill-formed triangles and triangle crossings. However, the Delaunay triangulation can produce mesh edges shared by more than two triangles, which constitutes self-crossing meshes. As described in [21], these meshes can be reduced to valid surface meshes, but neither the Delaunay triangulation nor the reduction to a surface mesh are computationally feasible for large point clouds.

To be a valid surface mesh, we want the mesh to be a collection of non-crossing, orientable polyhedra, which implies that it locally divides the 3D space into an inside and an outside region. We allow the mesh to be non-watertight and to have a boundary, i.e. triangles with less than three neighbors, as we have to account for the fact that there will be regions unobserved by the camera. This definition can allow edges shared by multiple triangles as degenerate cases as well, as outlined in Appendix D.1.

A prominent method for reconstructing orientable polyhedral meshes from a point cloud with normal information is the method of Poisson surface reconstruction of Kazhdan et al. [46]. Internally, this method uses a popular implicit surface representation, an occupancy map. In this volumetric representation, every point $\mathbf{P}$ in a 3D volume $\mathcal{V}$ is assigned a value of an indicator function $\chi : \mathcal{V} \to \{0, 1\}$:

$$\chi(\mathbf{P}) = \begin{cases} 1 & \text{if } \mathbf{P} \text{ lies in the interior of an object} \\ 0 & \text{else}\mathbf{P} \end{cases} \tag{5.1}$$

The surface is then extracted where the indicator function jumps between 0 and 1. Instead of using binary values, or ternary values as in [72], the image of χ is often relaxed to the convex interval $[0, 1]$, to represent the probability that a voxel belongs to the interior of the reconstructed geometry [58]. This convex representation of 3D geometry is very popular in the areas of photoconsistency-based single-view reconstruction [67], 3D reconstruction [52], and 4D reconstruction [66], where no initial depth maps or point clouds are given, and the 3D geometry is estimated entirely based on volumetric photoconsistency estimates and additional cues like silhouette information or color distributions for object and background textures. A commonly used prior here is a minimal surface area of the observed objects, and for a binary indicator function, the surface area is directly given by its total variation. Given an indicator function relaxed to the convex codomain $[0, 1]$, the surface is extracted as an iso-level, either by thresholding the continuous indicator function with a threshold between 0 and 1 and successively extracting the surface at the integer positions of the boundary voxels, or using a method for discretized iso-surface extraction of a continuous occupancy function like the “Marching Cubes” method of Lorensen and Cline in [54].

Compared to photoconsistency-based 3D reconstruction, the fusion of depth maps into a joint geometry model gives us several advantages:

- It allows to abstract the modality of creation of the depth map. Depth cameras using active illumination can be used equally well as photoconsistency-based stereo depth maps. For example, since active sensors like Microsoft’s and PrimeSense’s Kinect get the depth information from disparity estimates based on a structured light pattern, depth maps can also be captured from completely textureless geometry.

- We do not have to focus on the complicated 3D photoconsistency estimation problem. Estimating photoconsistency at global 3D coordinates involves a notion of surface visibility. Combined with the task of surface reconstruction, this becomes a chicken-and-egg problem.

- We get an intuitive volumetric data term. In variational photoconsistency-based approaches the regularity term penalized the gradient of the indicator function less in regions of good photoconsistency than in regions of weak photoconsistency. However, without a data term preferring the indicator function to be positive in some regions, the trivial optimal solution for the indicator function is the constant zero function. To overcome this problem, Vogiatzis et al. propose a "ballooning" term to inflating the geometry in [88], Kolev et al. propose data terms based on color distributions in the images in [52] and propagated photoconsistency estimates in [51], and Töppe et al. impose a volume constraint on the 3D geometry for the case of single-view reconstruction in [84]. The globally optimal method for photoconsistency-based disparity estimation of Pock et al. in [70] can also be regarded as a method for volumetric photoconsistency-based 3D reconstruction. However, here the reconstruction volume does not represent a Euclidean space but a projective space. The indicator function has the constraint of being 1 at infinity and 0 in the camera center. This constraint forces the indicator function to change somewhere to 1, and as in the other aforementioned methods, the location where it should change preferably is steered by the photoconsistency estimates in the regularity term.

  In contrast, every depth map we fuse into a global volumetric geometry representation bears an intuitive information of whether a voxel observed in the camera frustum belongs to the geometry interior or exterior.

On the other hand, a method for depth map fusion is required to have certain features not necessarily required in the area of photoconsistency-based 3D and 4D reconstruction. For the latter, the entire input image sequence is usually known from the start. In the area of RGB-D image fusion, we are usually not given all images from the start, but rather expected to incrementally add one RGB-D image after another. An algorithm not being given all data to work on from the start is commonly referred to as an online algorithm. In most methods for photoconsistency-based 3D reconstruction, the indicator function is optimized iteratively, and a very useful feature of the implicit representation is that it allows for changes in the topology of the reconstructed geometry. This freedom to change topology is also very useful for our purpose, successively fusing RGB-D images into a global geometry model. This suggests using an implicit surface representation as well.

In our approach, we do not use a binary indicator function as it is used in the aforementioned publications on photoconsistency-based 3D reconstruction. Instead, following the work of Curless and Levoy in [31], we store the running average of a signed distance function (SDF). The value $D(\mathbf{P})$ in each voxel $\mathbf{P}$ describes the distance to of that point to the closest surface. Voxels with a negative SDF value represent empty space, while voxels with a positive SDF value lie on the inside of an object. For the t-th RGB-D

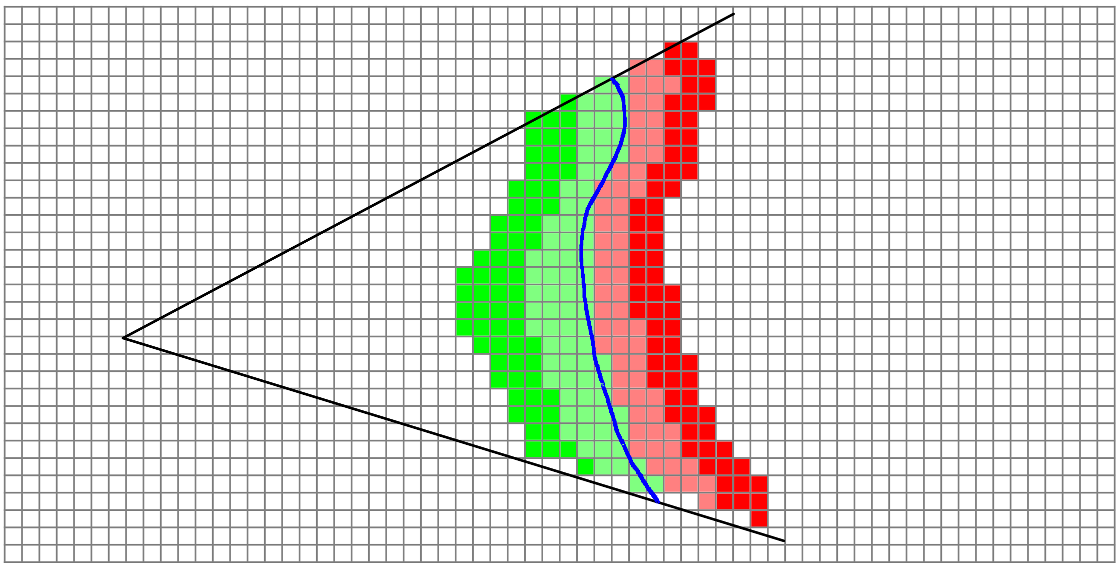


Figure 5.1.: Schematic partition of a voxel volume after the integration of one depth image, with $\delta = 0$. White voxels: $D = 0$, $W = 0$. Dark green voxels: $D = -T_D$, $W = 1$. Light green voxels: $-T_D < D < 0$, $W = 1$. Light red voxels: $0 < D < T_D$, $0 < W < 1$. Dark red voxels: $D = T_D$, $0 < W < 1$.

image, the running average is updated as

$$D_t(\mathbf{P}) = \frac{1}{t}\sum_{j=0}^{t} d_j(\mathbf{P}) = \frac{1}{t}\left(D_{t-1}(\mathbf{P})\,(t-1) + d_t(\mathbf{P})\right) \tag{5.2}$$

with $D_0(\mathbf{P}) = d_0(\mathbf{P}) = 0$.

In addition to the signed distance value, we store a weight value $W(\mathbf{P})$. The weight value fulfills three interlinked purposes:

- It is required for storing a running average in every voxel. Not every voxel is observed from every camera pose, therefore, storing a global counter of the number of integrated RGB-D images as in Equation (5.2) does not suffice to accurately store the denominator of the running average for every voxel $\mathbf{P}$.
- We want to allow non-watertight surfaces. Voxels not observed by any camera need to be assigned a special value. Otherwise, our reconstruction method would hallucinate a surface at the boundary of the observed part of the reconstruction volume. Instead of defining a special distance value, we define that voxels with an accumulated weight value below a certain threshold T_W should not contribute to the reconstructed surface geometry. In Figure 5.1 this is illustrated by the

white voxels outside the camera frustum. There should neither be a surface at the boundary between unobserved voxels and voxels with positive distance (red), nor between unobserved voxels and voxels with negative distance (green).

- For each new RGB-D image, we want to weight distance increments d differently in each voxel depending on how certain we are about the distance. In each new camera frustum, we can be rather sure that there are not any objects in the camera frustum between the camera center and the observed surface. However, we have little prior information about the thickness of the observed geometry behind the visible surface. Therefore, we assign a small or zero weight to the projective distance estimates behind the observed surface in each camera frustum. In Figure 5.1 this is illustrated by the white pixels behind the observed surface.

Given depth images as input, we have a notion of surface visibility, and we can accumulate the color information of the RGB-D image volumetrically as well, storing a running average $C \in \mathbb{N}^3$ in each voxel, and incrementing it by the values c read in the RBG image. The combined update equations of distance, weight, and color read as

$$D_t(\mathbf{P}) = \frac{D_{t-1}(\mathbf{P})W_{t-1}(\mathbf{P}) + d_t(\mathbf{P})w_t(\mathbf{P})}{W_{t-1}(\mathbf{P}) + w_t(\mathbf{P})}, \tag{5.3a}$$

$$W_t(\mathbf{P}) = W_{t-1}(\mathbf{P}) + w_t(\mathbf{P}), \tag{5.3b}$$

$$C_t(\mathbf{P}) = \frac{C_{t-1}(\mathbf{P})W_{t-1}(\mathbf{P}) + c_t(\mathbf{P})w_t(\mathbf{P})}{W_{t-1}(\mathbf{P}) + w_t(\mathbf{P})}. \tag{5.3c}$$

The integration of the t-th RGB-D image with color channel $\mathcal{I}_t$, depth channel h_t, and camerapose $(\mathbf{R}_t, \mathbf{T}_t)$ into a regular voxel grid of n^3 voxels is summarized by Algorithm 1. Following Curless and Levoy in [31], we truncate the SDF values to a threshold T_D to avoid a too large contribution of outlier values in line 7. In the computation of the incremental weight in line 8, the value δ denotes the value of the projective distance estimate, above which the incremental weight w linearly decreases to zero, and represents the assumed minimal object thickness. The weight threshold T_W is the distance value, above which w is set to zero. To avoid incorrect memory accesses and undefined values of d in our program, we explicitly check for both visibility of the voxel in the camera frustum (line 3) and validity of the depth estimate (line 5), instead of simply zeroing w in voxels without a valid projective distance estimate. The term $\mathbf{P}_c\,[3]$ denotes the z-element of the point $\mathbf{P}$. In the context of active RGB-D sensors such as the Kinect, an invalid depth estimate is detected if there is no correct correspondence in the structure light pattern, possibly due to specular reflective surfaces, missing geometry or a geometry to far away from the sensor, or too much ambient light covering the structured light pattern. However, we can also integrate predefined minimum and maximum depth values here.

Algorithm 1 Algorithm for SDF, weight, and color update in a regular voxel grid

```
 1: for $\mathbf{P}_v \in \{0, ..., n-1\}^3$ do
 2:   $\mathbf{P}_c \leftarrow \mathbf{R}_t^\mathsf{T}\mathbf{P}_v - \mathbf{R}_t^\mathsf{T}\mathbf{T}_t$
 3:   if $\mathbf{P}_c[3] > 0$ and $\pi(\mathbf{P}_c) \in \Omega$ then
 4:     $h \leftarrow h_t(\pi(\mathbf{P}_c))$
 5:     if $valid(h)$ then
 6:       $\Delta_D \leftarrow |\mathbf{P}_c| \left(1 - \frac{h}{\mathbf{P}_c[3]}\right)$
 7:       $d_t \leftarrow \max\{\min\{\Delta_D, T_D\}, -T_D\}$
 8:       $w_t(\mathbf{P}_v) \leftarrow \begin{cases} 1 & \text{if } \Delta_D < \delta \\ \frac{T_D - \Delta_D}{T_D - \delta} & \text{if } \Delta_D \geq \delta \text{ and } \Delta_D \leq T_W. \\ 0 & \text{if } \Delta_D > T_W \end{cases}$
 9:       $c_t \leftarrow \mathcal{I}_t(\pi(\mathbf{P}_c))$
10:       $D_t(\mathbf{P}_v) \leftarrow \frac{D_{t-1}(\mathbf{P}_v)W_{t-1}(\mathbf{P}_v) + d_t w_t}{W_{t-1}(\mathbf{P}_v) + w_t}$
11:       $W_t(\mathbf{P}_v) \leftarrow W_{t-1}(\mathbf{P}_v) + w_t$
12:       $C_t(\mathbf{P}_v) \leftarrow \frac{C_{t-1}(\mathbf{P}_v)W_{t-1}(\mathbf{P}_v) + c_t w_t}{W_{t-1}(\mathbf{P}_v) + w_t}$
13:     end if
14:   end if
15: end for
```

The most prominent work based on the Curless and Levoy publication is “KinectFusion” of Newcombe, Izadi et al. [62, 45], where the concept of volumetric depth map integration is applied in a GPU-supported real-time framework based on the Kinect sensor. Another approach worth mentioning here is the one of Zach et al. in [95]. Instead of using a running average of SDF values, they create an L^1 data term by penalizing the sum of absolute differences to all projective distance estimates and combine it with a spatial regularity term. Without this regularity term, the minimizer of the data term yields the median of projective distances, which is more robust than the running average of Curless and Levoy we are using. However, it also much more expensive, since we would have to store all projective distance values to update the median value. In [40] Graber et al. present an online version of this approach using histograms of projective distance values to overcome the large memory demand.

Besides mapping approaches based on point cloud representation and approaches using an implicit volumetric representation in the form of an occupancy or distance function, there also exist other approaches for mapping, such as Multi-Resolution Surfel Maps of Stückler and Behnke in [82].

5.2. Sparse Representation of the Reconstruction Volume

The main problem of dense volumetric surface representation is clearly its limited scalability, mostly due to the large amount of voxel data required to be stored and updated. The asymptotic complexity of the memory demand for representing the entire surface geometry captured by an RGB-D camera is cubic in the length of the camera trajectory, as in the worst case the camera moves on the diagonal of the cuboid reconstruction volume. Analogously, it is also cubic in the resolution of the resulting surface mesh. Furthermore, to be able to run in real-time, newer implementations of volumetric methods typically rely heavily on parallelization, mostly on GPUs. The need for GPUs however further decreases the amount of efficiently accessible memory, since contemporary GPUs usually have a smaller amount of memory available than contemporary CPUs. Therefore, methods like KinectFusion are restricted to small environments. For example, the reconstruction of a house in a predefined volume of 20m×20m×10m at 1cm voxel resolution and a quantization of 4 Bytes for distance, 4 Bytes for weight, and 3*2 Bytes for color would result in a data volume of more than 52 GB.

Currently a memory demand of this size is not met by any consumer grade GPU. While we can certainly expect a significant increase in available GPU memory in the near future, we can also expect an increase in the amount of data provided by RGB-D sensors, related to higher spatial and temporal resolutions. Therefore, an optimized approach fulfilling a given performance requirement on contemporary hardware can still serve its purpose on future hardware in the presence of larger input data.

Another limitation in scalability of a dense reconstruction volume is its limited online capability. We can incrementally integrate RGB-D images inside the reconstruction volume, however, those parts of the camera frustum not intersecting the reconstruction volume are lost. To reconstruct the entire geometry observed in the cameras, we need to know the union of all camera frusta in advance. To have a fully online capable mapping algorithm, we need to be able to dynamically adapt the reconstruction volume to the progressing camera path.

The key observation for overcoming the problem of a large memory demand is the fact that for many scenes the bulk of the volumetric data is constant. Voxels in the volume not observed by any camera have a constant zero weight, voxels lying in empty space, observed close to the camera position far in front the observed surface have a constant weight of 1 and a truncated SDF value of $-T_D$, and voxels lying far inside thick objects have zero SDF and weight values. All these voxels do not contribute to the implicitly represented surface. Therefore, a data structure only storing a narrow band of voxels around the surface geometry can overcome the large memory demand of a dense voxel grid.

Such a data structure needs to fulfill three general requirements: The first two requirements have already been explained above: The amount of memory required should be dependent on the observed geometry, without unnecessary processing of voxels in empty space, and the reconstruction volume should “grow” dynamically with a progressing camera path. The third requirement is that any update of the data structure with a new RGB-D image should be possible in sublinear runtime complexity with respect to the number of voxels already stored in the data structure.

Before we give an overview of different data structures, we want to emphasize that in any data structure it is rather infeasible to store individual voxels, as an implicit surface representation relies on the notion of voxel neighborhoods. Therefore, we group voxels into small cubic mini-volumes or bricks, an approach common in computer graphics, see for example [30].

A simple data structure is a list of bricks, with every brick storing its 3D position. This structure is very similar to the joint point cloud described at the beginning of this chapter. The requirement of a low, geometry-dependent storage cost is met, as we only process bricks in the vicinity of the observed surface geometry. The requirement of being able to efficiently grow the reconstruction volume is also met trivially, since one simply adds new bricks to the list. However, the requirement of being able to efficiently update the band is not met, since for every 3D point from a depth map we have to traverse the complete list to check for already existing bricks in the vicinity of the point, in order to avoid duplicate bricks in the list. This constitutes an $O(n)$ operation, with n being the length of the brick list. Assuming a linear growth of n with respect to the number of integrated RGB-D images, this results in a quadratic runtime complexity for the integration of all RGB-D images. A local reconstruction volume following the camera can be regarded as this approach taken to the extreme, with overlapping bricks in the size of an entire reconstruction volume and only one brick updated for every RGB-D image. Whelan et al. follow this approach in [92, 90, 91], and instead of keeping a list of reconstruction volumes, they extract a polygon mesh for the geometry outside the local reconstruction volume around the camera. Doing so drastically reduces the memory demand, but renders revisiting places of an already compressed reconstruction volume a tricky task.

An approach different to a simple list of bricks is storing a dense grid of brick pointers for which bricks are only allocated when lying close to the surface. Other than with the list of bricks, we can access every brick in $O(1)$ runtime. However, we give up the ability to easily grow the reconstruction volume. Using a moving reconstruction volume, the problem of being able to update parts of the geometry outside of the current reconstruction volume again constitutes an $O(n)$ problem, similar to the one with the brick list described above, only with one or a few larger volumes instead of many smaller bricks.

In general there are two data structures for efficient storage of a sparse set of elements, hash maps and tree structures. In the case of tree structures, the three-dimensional key space in combination with bricks at integer positions suggests using octrees.

Besides recent approaches in voxel hashing [64], most approaches for a sparse representation of a reconstruction volume for implicit surface representation use octrees as a method of choice, such as Fuhrmann and Goesele [38], Chen et al. [28], or Zheng et al. [96]. Octrees are also popular in the context of occupancy maps, the most prominent system is the "Octomap" of Wurm et al. [93].

In our work we also use an octree structure to perform fast and scalable RGB-D image fusion. Compared to the methods above, instead of traversing the entire camera frustum, we only update a narrow band around the surface. In Figure 5.1, this is expressed by the white voxels in the frustum in front of the depth map not storing an explicit representation of empty space ($D = -T_D$, $W = 1$) like the dark green voxels. Instead, they are not stored in the data structure at all ($D = 0$, $W = 0$). On the one hand, this approach has the drawback of not being able to eliminate outlier noise far away from the correct surface. This essentially means that everything visible in the joint point cloud is also visible in our dense reconstruction. On the other hand, this approach is very memory efficient, fast, parallelizable in several ways, and it runs on current CPUs in real-time.

5.2.1. Multiscale Approach

A feature of an octree particularly useful for the fusion of disparity-based depth maps, e.g. the depth maps of first generation Kinect sensors, is that an octree provides an intuitive representation of hierarchical data. According to the intercept theorem, the size of voxels projecting into a single pixel grows linearly with the distance between the voxel and the camera center. A voxel at distance h to the camera center can have a size of $\frac{h}{f}$ to project into a single pixel (assuming square pixels, i.e. $f_x = f_y = f$). Also, as we explained in Section 2.4.1, the distance of a point to the camera h and its corresponding disparity estimate have a reciprocal relationship:

$$h(d) = \frac{Bf_x}{d} \tag{5.4}$$

Assuming that the disparity measurements d of the sensor lie in the interval

$$d \in [d^* - \sigma, d^* + \sigma] \tag{5.5}$$

around the true disparity d^*, substituting (5.4) in (5.5), we get an interval of

$$\left[h(d^*) + \frac{h^2(d^*)\sigma}{Bf_x - h(d^*)\sigma}, h(d^*) - \frac{h^2(d^*)\sigma}{Bf_x + h(d^*)\sigma}\right] \tag{5.6}$$

for the corresponding depth values (see Appendix D.2). Assuming a small disparity error relative to the true disparity, we can approximate the depth values by a Taylor expansion at d^*:

$$h(d^* \pm \sigma) \approx h(d^*) \pm \sigma \left(\frac{\partial h}{\partial d}\right)\Bigg|_{d^*} = h(d^*) \mp \sigma \frac{Bf_x}{d^{*2}} = h(d^*) \mp \sigma \frac{h^2(d^*)}{Bf_x} \tag{5.7}$$

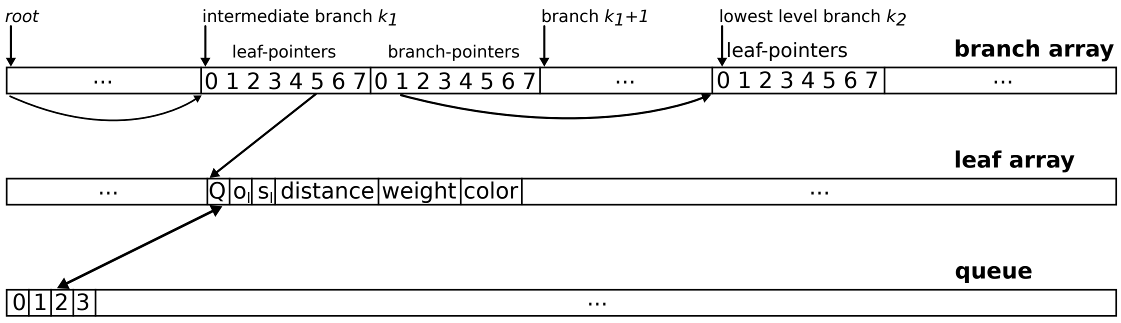


Figure 5.2.: Octree representation in memory. We store all branches in one array. One branch comprises 8 pointers to sub-branches and 8 pointers to its leaves. All leaves are stored in a second array. For fast access during the integration of a new RGB-D image a queue maintains pointers to the leaves that have to be updated.

This means that for disparity-based depth sensors, for a relatively small disparity error uniformly distributed among all disparity values, the error in the corresponding depth values grows quadratically with increasing depth. For more detailed information we refer to Gallup et al. [39]. Therefore, it makes sense to store the volumetric SDF information on different resolutions in the sparse data structure, depending on how close a brick has been to a camera. In our octree data structure this can be done easily by storing bricks as leaves of the tree at different depths from the tree root. While in most descriptions of tree data structures, the term “leaf” denotes a node or branch in the tree without children or subbranches, we use it to denote the brick data stored in the branches at all levels in the tree in the rest of this thesis, i.e. when a branch has a leaf, it means that the leaf pointer in the branch array is not *NULL*.

The further away a brick is stored from the root, the smaller is the size of the voxels in the brick, and the higher is the resolution of the implicit surface in this area. This implies that bricks can lie inside one another, if parts of the volume covered by a coarse brick, resulting from being observed by a camera far away, are also observed by a camera close by, see Figure 5.4c. In this case, the coarse brick is not disregarded or entirely subdivided into bricks of the same resolution as the small brick, but kept in the tree. Extraction of the iso-surface from the volumetric data is then performed from fine to coarse resolutions: If the weight in the voxels of a brick of fine resolution is too low, we proceed to the next coarser resolution. Figure 5.2 shows a schematic overview of the general composition of our octree data structure. It is comprised of an array of branches in the tree, an array of leaves/bricks in the tree, and a queue of bricks to be updated for every RGB-D image.

Instead of storing a pointer to a leaf/brick and 8 pointers to subtrees for every branch, we do not place a leaf at the root of the tree, and directly store 8 pointers to leaves/bricks in the leaf array and 8 pointers to possible subbranches in the branch array for every branch. This way, we can omit pointers to branches for the lowest level of branches, where only a leaf but no further subbranches are stored. For example, if we have an octree representing a volume of 64^3 voxels at the finest resolution/voxel size λ, and a brick size of 8^3 voxels, the tree has 3 layers: For the root and the next finer layer, 8 pointers for subbranches and 8 pointers for leaves are stored in the branch array. For the lowest layer, only the 8 pointers to leaves need to be stored in the branch array (see Figure 5.2). All bricks in the tree have the same size, therefore, the bricks stored in the root branch and the layer of branches below the root also have a size of 8^3, but the voxels have a size of 4λ and 2λ, respectively.

For each leaf/brick in the array, we store the volumetric SDF, weight, and color data. For a brick of 8^3 voxels, as it is sized in our implementation, and each voxel using 4 Bytes for a floating-point SDF value, 4 Bytes for a floating-point weight value, and 2 Bytes for the integer color value of each channel in an RGB texture, this amounts to 7168 Bytes of voxel data. In addition, we store for each brick its integer 3D offset $\mathbf{o}_l$ relative to the root of the tree in the number of voxels on the finest voxel scale, its integer scale s_l in multiples of the finest voxel scale, and an integer position in the brick queue for the current RGB-D image. In our experiments we use 2 Bytes for each dimension in the position value, 2 Bytes for the scale value, and 4 Bytes for the queue index, limiting the maximum voxel resolution of the entire tree to 65536^3. With a minimum voxel size of 5mm, this limits the entire reconstruction volume to roughly $(320\text{m})^3$. While sufficient for our experiments presented in Section 5.5, the extent of the maximum reconstruction volume can be increased exponentially with respect to the size of the octree, by using larger data types for the index arrays. Similar to other data structures like dense voxel grids, we also store a global 3D position of the tree root and its finest voxel scale in floating-point precision.

Since the depth sensor has a fixed maximum range, the geometry that can be observed by the depth sensor lies in a convex frustum of fixed maximum diameter. Given a fixed minimum voxel size, all voxels and bricks intersecting this frustum lie in an axis-aligned cube of fixed maximum side length, and are bounded above by a constant. The number of levels in the octree that lie between the finest level and the level at which no more than 8 voxels and bricks intersect this cube is logarithmic. By means of the geometric series, the sum of all voxels between the leaf level and this level is therefore bounded above by a constant as well. Going higher in the tree, at each level at most 8 voxels and bricks have to be updated. All in all, for each depth image the maximum number of voxels to be updated in the tree is bounded above by a constant plus a negligible logarithmic overhead.

In the next section we will detail the procedure of integrating an RGB-D image in the tree and its implementation on CPU and GPU hardware.

5.3. Implementation

Integrating an RGB-D image into our data structure involves a twofold process:

In the first step we iterate over all valid depth pixels in the current RGB-D image, compute the corresponding 3D points in global coordinates by applying the pose transformation from camera coordinates into global coordinates. For each point $\mathbf{P}$, we create an axis-aligned cube

$$[P_X - T_D, P_X + T_D] \times [P_Y - T_D, P_Y + T_D] \times [P_Z - T_D, P_Z + T_D]\,, \tag{5.8}$$

i.e. a 1-ball around $\mathbf{P}$ with the truncation threshold T_D as its radius. Beyond this cube, the projective SDF update for the current RGB-D image corresponding to $\mathbf{P}$ is constant (compare Figure 5.1). Depending on the distance of $\mathbf{P}$ to the camera center, we compute the voxel resolution of the bricks intersecting the cube, which determines their level in the tree. Next we look up the bricks in the tree. If the bricks for a given position and size have not been already allocated in the tree, we add them to the tree. If not already present from previous operations on the tree, we also add new subbranches along the paths from the root to the bricks. For every RGB-D image we compute a bounding box of all points outside the current reconstruction volume represented by the tree. If any of the points lie outside the current reconstruction volume, we iteratively grow the tree by adding new roots until the reconstruction volume represented by the tree encloses all points of the current RGB-D image.

In the second step we iterate over all bricks previously placed in the brick queue and update the SDF, weight, and color values inside those bricks. The update procedure for the voxels in one brick is similar to the one presented in Algorithm 1 for the case of a dense voxel grid, with the only difference in line 2, where we have to include the position and scale of the tree and the current leaf to transform the point $\mathbf{P}_b$ in brick coordinates to the point $\mathbf{P}_c$ in camera coordinates:

$$\mathbf{P}_c \leftarrow \mathbf{R}_t^\top \left(\mathbf{o}_t + \left(\mathbf{o}_l + \mathbf{P}_b s_l\right) s_t\right) - \mathbf{R}_t^\top \mathbf{T}_t \tag{5.9}$$

5.3.1. Implementation on the GPU

In our implementation on the GPU we parallelize our code at two points: We integrate all pixels of an RGB-D image into the octree in parallel, and we update all voxels in the bricks in parallel. To this end, we need to adhere to certain paradigms of GPGPU computing and slightly adapt our data structure.

The main bottleneck of current GPU computations is the access of global memory accessible by all threads. Since this memory is accessed sequentially over a bus, it is important that we use as much of the bus bandwidth as possible with coalesced memory fetches. This induces the paradigm of using structures of arrays rather than arrays of structures. Other than depicted in Figure 5.2 for the sake of clarity, we do not use a structure for every leaf, comprised of the queue index, position, scale, distance, weight, and color values, and store and array of these structures, but instead store all the elements of the leaf structure in separate arrays.

On GPUs we also do not have trivial support for dynamic arrays or linked lists. Though modern GPUs support allocation of global memory from inside GPU code, an append operation on a dynamic array performed in one thread would void all memory references to the array in other threads, if memory has to be resized. In addition, the array would need to be copied into the newly allocated memory during a resize operation, requiring the old and the new array to exist simultaneously, and therefore, reducing the total available memory on the GPU. Linked lists require memory overhead for pointers and produce fragmented fields of memory not suited for coalesced read and write operations. Therefore, we allocate all arrays before integrating the first RGB-D image, and perform later append operations by incrementing global variables.

Performing the SDF, weight, and color update according to Algorithm 1 in parallel for every voxel is comparatively easy. Performing the tree traversal however is more complicated, as we have a lot more diverging code, atomic append operations and limited local memory.

For intersecting an axis-aligned cube with the volumes covered by branches in the tree, we possibly have to traverse several paths in the tree from the root to all bricks intersecting the cube. With k denoting the number of bricks intersecting the cube, and d denoting their depth (i.e. their distance to the root) in the tree, the naïve strategy of precomputing the positions of all bricks and then traversing the tree for each brick from the root to the branch holding the brick requires $O(kd)$ operations. However, given the cube and any branch in the tree, we can easily deduce which subbranches intersect the cube as well (see Appendix D.3). Therefore, we can reduce the runtime complexity to $O(k + d)$ by employing algorithms like breadth-first search or depth-first search. For breadth-first search we need to store a queue of at least k indices of branches per thread. For depth-first search, we need to store at most d indices per thread, one for each branch on the path from the root to a leaf. Since k is dependent on the truncation threshold T_D, it can get very large for large threshold values. In contrast, d is limited by the maximum depth of the tree. Since a queue of variable and possibly large size could exceed the available amount of local memory for each thread on a GPU, we use the depth-first search in our implementation, and store the indices in the shared memory of the GPU.

In our depth-first search implementation we have to handle mutually exclusive operations explicitly. Most modern GPUs have an intrinsic support for atomic operations on single scalar values, assuring that only one thread at a time accesses a memory cell. However, each branch or a brick should only be added to the tree once, which requires more than one instruction in critical sections.

Algorithm 2 shows the parallel depth-first traversal of the octree in pseudo-code. The atomic append operations for branches, bricks, and brick pointers in the queue are highlighted in purple. As they consist of several instructions and are not natively

Algorithm 2 Algorithm for parallel depth-first traversal of the octree

```
 1: branch ← root
 2: depth ← 0
 3: while depth ≥ 0 do
 4:    for each child intersecting the cube do
 5:       if depth = desired depth then
 6:          if child brick pointer in branch array is NULL then
 7:             try to get access to MUTEX for child brick pointer in branch array
 8:             if got access to MUTEX then
 9:                append brick to brick array
10:                set child brick pointer in branch array to newly allocated brick
11:             end if
12:             goto 6
13:          else
14:             if queue index of brick is NULL then
15:                try to get access to MUTEX for queue index of brick
16:                if got access to MUTEX for queue index of brick then
17:                   append brick pointer to brick queue
18:                   set queue index of brick to newly appended queue index
19:                end if
20:             end if
21:          end if
22:       else
23:          if child branch pointer in branch array is NULL then
24:             try to get access to MUTEX for child brick pointer in branch array
25:             if got access to MUTEX then
26:                append branch to branch array
27:                set child branch pointer in branch array to newly allocated branch
28:             else
29:                goto 23
30:             end if
31:          else
32:             branch ← child branch behind pointer
33:             depth ← depth + 1
34:             goto 3
35:          end if
36:       end if
37:    end for
38:    branch ← parent branch
39:    depth ← depth - 1
40: end while
```

supported by current GPUs, we have to embed them into regions of mutual exclusion (MUTEX). Because we also do not have a native call stack on current GPUs, we have to emulate recursions into subbranches with the branch and depth variables. We do not store a parent branch for every branch in an array in global memory, but rather store one parent branch pointer at every depth level of the tree in shared memory as described above. On current GPUs the computing cores are grouped into sets in which all threads execute the same instructions. In Nvidia's CUDA programming framework these sets are referred to as "warps". Sharing the same instructions, when only some threads in a warp enter a conditional branch, all threads not entering the branch are stalled. For "if-then-else" conditionals, it is also not defined whether the "then"-branch is executed before or after the "else"-branch.

In lines 12 and 29, the threads that did not get the MUTEX to allocate the brick or branch have to wait until the thread that did is done. In the case of an unallocated branch they have to traverse it and in case of an unallocated brick they possibly have to put it into the queue, since queuing bricks and allocating bricks have to be decoupled. However, the threads cannot wait in infinite loops in lines 12 and 29 until another thread has allocated the brick or branch, because it will stall the entire program if the "else"-branch is executed before the "then"-branch. Instead, we have to repeat the loop for all threads, including the allocating thread.

5.3.2. Implementation on the CPU

While an implementation of our approach on a modern GPU yields impressive run time results (see Section 5.5), the use of GPUs limits its applicability. Many mobile platforms do not support high-level GPUs, due to their relatively large weight and power demand. However, they often support modern desktop and laptop CPUs, which require significantly less power and cooling. A good example are recently developed quadrotor platforms like the "Pelican" from Ascending Technologies, supporting Intel Core-i7 CPUs. Though powerful GPUs suited for mobile and embedded devices are in development, for example new Nvidia Tegra devices, being able to perform parts of our approach efficiently on a CPU enables us to distribute the computational workload between both processor types on future architectures. Another benefit we get from having our approach run on a CPU is the larger amount of available memory compared to a GPU, and implementations distributed on both processor types can make efficient use of both types of memory at once.

For the traversal of the tree for allocation of branches and bricks, and for queuing allocated bricks, we can make use of a serial loop over all bricks by eliminating multiple traversals of the tree for allocating the same bricks: If we choose the truncation threshold T_D to be a multiple of the brick length, then the cubes around two points falling into the same brick will intersect the same set of bricks. The two points that are most likely to fall into the same brick are points corresponding to adjacent image pixels. If we iterate

Algorithm 3 Parallelizable algorithm for SDF, weight, and color update in the brick of a multiscale octree

```
 1: for $\mathbf{P}_b \in \{0, ..., s_b - 1\}^3$ do
 2:     $\mathbf{P}_c \leftarrow \mathbf{R}_t^\top (\mathbf{o}_t + (\mathbf{o}_l + \mathbf{P}_l s_l) s_t) - \mathbf{R}_t^\top \mathbf{T}_t$
 3:     if $\mathbf{P}_c[3] > 0$ and $\pi(\mathbf{P}_c) \in \Omega$ then
 4:         $h \leftarrow h_t(\pi(\mathbf{P}_c))$
 5:         if $valid(h)$ then
 6:             $\Delta_D \leftarrow |\mathbf{P}_c| \left(1 - \frac{h}{\mathbf{P}_c[3]}\right)$
 7:             $d_t \leftarrow \max\{\min\{\Delta_D, T_D\}, -T_D\}$
 8:             $w_t \leftarrow \begin{cases} 1 & \text{if } \Delta_D < \delta \\ \frac{T_D - \Delta_D}{T_D - \delta} & \text{if } \Delta_D \geq \delta \text{ and } \Delta_D \leq T_D. \\ 0 & \text{if } \Delta_D > T_D \end{cases}$
 9:             $c_t \leftarrow \mathcal{I}_t(\pi(\mathbf{P}_c))$
10:             $D_t(\mathbf{P}_b) \leftarrow \frac{D_{t-1}(\mathbf{P})W_{t-1}(\mathbf{P}) + d_t w_t}{W_{t-1}(\mathbf{P}_b) + w_t}$
11:             $W_t(\mathbf{P}_b) \leftarrow W_{t-1}(\mathbf{P}_b) + w_t$
12:             $C_t(\mathbf{P}_b) \leftarrow \frac{C_{t-1}(\mathbf{P}_b)W_{t-1}(\mathbf{P}_b) + c_t w_t}{W_{t-1}(\mathbf{P}_b) + w_t}$
13:         end if
14:     end if
15: end for
```

over all depth image pixels from top to bottom and from left to right (without loss of generality), we can check for every pixel whether it falls into the same brick as its top or left neighbor. This check is performed by a simple $O(1)$ rounding operation and it saves an $O(d + k)$ traversal of the tree as described above.

The update of the SDF, the weight, and the color values in each voxel is the part of our algorithm that benefits most from parallelization. Being able to perform all operations in the loop from line 1 to line 15 of Algorithm 1 in parallel enables methods like KinectFusion to run in real-time. However, iterating over all voxels in a serial manner on a CPU enables us to reduce many operations otherwise run in parallel. Moreover, we can make use of the support for SIMD parallelism on modern CPUs. The x86-CPUs used in our experiments provide SIMD parallelism in the form of Streaming SIMD Extensions (SSE). To make use of these vector instructions, we have to replace the conditionals in our code with binary masks. Algorithm 3 shows an implementation of the voxel update in each brick that can run in parallel on a GPU, i.e. a combination of Algorithm 1 and Equation (5.9). The algorithm focuses only on a single brick b and does not take into account the position of b in memory among other bricks. The side length of the brick in voxels is denoted by s_b. Algorithm 4 shows various optimizations regarding the serial iteration over the voxels in the brick.

Since memory is addressed linearly, we use the integer scalar value p_b to address the memory instead of the 3D addresses $\mathbf{P}_v$ and $\mathbf{P}_b$ in Algorithms 1 and 3, according to the mapping rule

$$p_b = \left(\mathbf{P}_b\left[3\right] s_b + \mathbf{P}_b\left[2\right]\right) s_b + \mathbf{P}_b\left[1\right]. \tag{5.10}$$

Algorithm 4 Serialized and SIMD-capable voxel update algorithm in an octree

```
1: $p_b^{\text{SIMD}} \leftarrow s_b^3 b$
2: $\mathbf{P}_c^z \leftarrow \mathbf{R}_t^\mathsf{T}\left(\mathbf{o}_t + \mathbf{o}_l s_t\right) - \mathbf{R}_t^\mathsf{T}\mathbf{T}_t$
3: for $z = 0$ to $s_b - 1$ do
4:   $\mathbf{P}_c^y \leftarrow \mathbf{P}_c^z$
5:   for $y = 0$ to $s_b - 1$ do
6:     $\mathbf{P}_c^x \leftarrow \mathbf{P}_c^y$
7:     for $x = 0$ to $\frac{s_b}{n_{\text{SIMD}}} - 1$ do
8:       for $k \in \{0, ..., n_{\text{SIMD}} - 1\}$ do
9:         $p_b \leftarrow p_b^{\text{SIMD}} + k$
10:        $\mathbf{P}_c \leftarrow \mathbf{P}_c^x + \mathbf{R}^\mathsf{T}\left[:,1\right] k s_l s_t$
11:        $h \leftarrow h_t(\pi_\Omega(\mathbf{P}_c))$
12:        $\Delta_D \leftarrow \left|\mathbf{P}_c\right| \left(1 - \frac{h}{\mathbf{P}_c[3]}\right)$
13:        $d_t \leftarrow \max\{\min\{\Delta_D, T_D\}, -T_D\}$
14:        $w_t \leftarrow \begin{cases} 1 & \text{if } \Delta_D < \delta \\ \frac{T_D - \Delta_D}{T_D - \delta} & \text{if } \Delta_D \geq \delta \text{ and } \Delta_D \leq T_D . \\ 0 & \text{if } \Delta_D > T_D \end{cases}$
15:        $w_t \leftarrow w_t \left[\mathbf{P}_c\left[3\right] > 0 \textbf{ and } \pi(\mathbf{P}_c) \in \Omega \textbf{ and } \mathit{valid}(h)\right]$
16:        $c_t \leftarrow \mathcal{I}_t(\pi(\mathbf{P}_c))$
17:        $D_t(p_b) \leftarrow \frac{D_{t-1}(p_b)W_{t-1}(p_b) + d_t w_t}{W_{t-1}(p_b) + w_t}$
18:        $W_t(p_b) \leftarrow W_{t-1}(p_b) + w_t$
19:        $C_t(p_b) \leftarrow \frac{C_{t-1}(p_b)W_{t-1}(p_b) + c_t w_t}{W_{t-1}(p_b) + w_t}$
20:      end for
21:      $p_b^{\text{SIMD}} \leftarrow p_b^{\text{SIMD}} + n_{\text{SIMD}}$
22:      $\mathbf{P}_c^x \leftarrow \mathbf{P}_c^x + \mathbf{R}_t^\mathsf{T}\left[:,1\right] s_l s_t n_{\text{SIMD}}$
23:    end for
24:    $\mathbf{P}_c^y \leftarrow \mathbf{P}_c^y + \mathbf{R}_t^\mathsf{T}\left[:,2\right] s_l s_t$
25:  end for
26:  $\mathbf{P}_c^z \leftarrow \mathbf{P}_c^z + \mathbf{R}_t^\mathsf{T}\left[:,3\right] s_l s_t$
27: end for
```

Naturally this mapping to linear memory has to be computed in parallelized implementations as well. In a serial iteration over the voxels in a brick however, we can simply increment the memory address from one voxel to the next instead of recomputing the entire mapping.

This optimization by itself is rather trivial, and we do not evaluate it specifically, but we do not want to omit mentioning it either, because the similarly iterative computation of the transformed voxel positions $\mathbf{P}_c$ yields a significant increase in performance: For line 2 of Algorithm 3, we need to perform at least one camera pose transformation $\mathbf{RP}+\mathbf{T}$, involving 9 multiplications and 9 additions. In a serial iteration over the voxels in x-, y- and z-direction, as in lines 3, 5, and 6 of Algorithm 4, we can reduce the computational effort to 3 additions for the corresponding columns of the rotation matrix in lines 22, 24, and 26. The scalar multiplications in those lines can be precomputed and the 3D positions of the tree and the brick in the tree only occur before the iteration in line 2. Under the paradigm of using structures of arrays, as mentioned in the beginning of Section 5.3.1, the memory area belonging to the b-th brick starts at address $s_b^3 b$, used in the assignment in line 1.

To be able to use the SIMD capabilities of current CPUs, we split the inner iteration over x into two parts, a serial part in line 6 and a parallel part in line 7. Apart from non-aligned memory accesses in line 11, the code between lines 7 and 20 can be run in parallel. The conditionals in lines 3 and 5 of Algorithm 3 are now encoded in line 15, using Iverson's notation from (2.1). If one of the conditions is false, we add a zero increment to the SDF, weight, and color values. To avoid incorrect memory accesses previously avoided by the conditional in line 3 of algorithm 3, we have to clamp pixel coordinates to the image plane after the pinhole camera projection, denoted by π_Ω in line 11.

5.4. Online Applications Using Dense Geometry Reconstructions

In the last sections we have presented data structures and algorithms for dense surface geometry reconstruction, that run in real-time on consumer hardware due to an efficient usage of memory and computational power, are capable of dealing with changes in topology, and that do not require all the input data to be given at the beginning of the algorithm, therefore constituting an online method. We will prove these claims in the experiments in Section 5.5.

However, the focus lay primarily on the input side - building up a dense geometry representation from input data with the aforementioned attributes. In addition, we have to address the output/consumer side as well - how such a representation can be used in other systems that motivate these attributes, and what kind of additional challenges these systems come with.

For example, large-scale 3D geometry reconstruction methods can be used for new view synthesis or creating virtual tours through prominent landmarks. A popular example is "Building Rome in a Day" by Agarwal et al. [7]. This system can reconstruct a point cloud representation of an entire city from extremely large collections of photos on

Internet photo sharing sites, estimating camera locations and calibrations concurrently. While yielding impressive results, systems like this are clearly neither required to deliver the point cloud reconstruction while the input image are being captured, nor do they have to run in real-time on mobile devices. Instead, the method runs on a cluster of desktop computers and takes up to a day for the reconstruction of a typical scene.

In contrast, two systems using dense environment reconstructions provided in real-time from an online algorithm are mixed reality (MR) applications and robotic exploration.

In MR, virtual objects rendered on see-through devices or on portable screens like smartphones or tablets interact with real objects in their environment. Possible interactions are occlusions of virtual objects by physical objects with respect to the observer, physics simulations like collision detection or casting shadows, or other tasks like path planning. To simulate these interactions, the MR system needs a dense geometry reconstruction of the environment.

In robotic exploration, a mobile robot is navigating through an unknown environment. The navigation can be autonomous, for which the robot has to interpret the reconstructed environment representation itself, or the robot can be controlled remotely from a base station, for which the environment representation has to be transmitted to the base station. Combinations are possible as well, where local obstacle avoidance is performed autonomously by the robot, while larger-scale tasks like path planning are performed remotely, potentially coordinating multiple robots simultaneously.

These two examples reveal additional challenges for applications based on dense environment reconstructions, both with respect to runtime and memory efficiency:

- “Real-time” can mean many different speeds for different applications, both in terms of frame rates and latency. Until now in this thesis, “real-time” meant “as fast as the input sensor, or slower by a constant factor, so that dropped frames to not affect the result”. In MR applications, the frame rate and latency at which the geometry reconstruction is rendered from different perspectives have to be as fast as the display for screen-based systems, or as fast as the human visual system for see-through devices. Especially the latter is usually much faster than the input sensors. On the other hand, the rate and latency at which new observations of the environment are reflected in the virtual reconstruction can be much slower than the rendering rate and the MR system can still deliver a compelling experience in static environments. The same is true for robotic exploration, where the frame rate and latency at which an environment representation is rendered on a base station can differ greatly from the rate and latency at which updates of the geometry representation are sent from the mobile platform to the base station.

- Tasks like transmitting an environment map from one system to another or rendering it in real-time can have even stricter memory constraints than reconstructing a scalable geometry representation itself. However, these tasks can operate on volatile snapshots of the geometry reconstruction that are not required to be adaptable with respect to topology changes over time.

The implicit SDF representation of the surface geometry from the previous sections is already applicable for some robotic tasks such as autonomous path planning or obstacle avoidance. However, for tasks like map transmission or multi-robot coordination, a map representation in form of a triangle mesh is preferable, because it has a much lower memory footprint than an SDF.

For visualization, the method of choice in the approaches cited above is raycasting, where for each pixel in the image of a virtual camera, a ray is traversed through the voxel structure until it passes the iso-level. However, rendering a triangle mesh can be performed much faster than raycasting a volumetric geometry representation of equal size and complexity, as GPUs are highly optimized for triangle rasterization, and raycasting involves both scattered memory read operations and branching code.

For MR applications, extracting a triangle mesh from the SDF instead of raycasting every image benefits the required high rendering frame rates. For robotic exploration, it benefits the rendering rate as well as the memory footprint for map transmission, if the geometry is rendered on the base station. If it is rendered on the mobile platform, a potentially large number of images have to be transmitted to the base station, and the rendering latency includes the transmission time. If the rendering rate is higher than the reconstruction rate, this can impose a significant computational burden on the mobile platform, especially for a computationally expensive method like raycasting. Giving the base station control over the virtual camera used for visualization also requires an uplink from the base station to the mobile platform, adding to the rendering latency. Therefore, this is not a favorable approach.

In the following, we will present a method for efficiently extracting a triangle mesh from the SDF data, that can then be used for fast visualization and incremental map transmission.

5.4.1. Mesh Extraction from Volumetric Data

To extract a mesh at the zero-level of our SDF, we use the "Marching Cubes" method of Lorensen and Cline [54]. Every method for iso-level extraction from voxel data requires to look at a local neighborhood of each voxel. In the case of Marching Cubes, we need to consider groups of 8 adjacent voxels for extracting triangles in the volume between them. For meshing a voxel at position $\begin{bmatrix}x & y & z\end{bmatrix}^\mathsf{T} \in \mathbb{N}^3$, we have to know the SDF values of this voxel as well as its 7 higher neighbor voxels at $\begin{bmatrix}x{+}1 & y & z\end{bmatrix}^\mathsf{T}$, $\begin{bmatrix}x & y{+}1 & z\end{bmatrix}^\mathsf{T}$, $\begin{bmatrix}x & y & z{+}1\end{bmatrix}^\mathsf{T}$, $\begin{bmatrix}x{+}1 & y{+}1 & z\end{bmatrix}^\mathsf{T}$, $\begin{bmatrix}x{+}1 & y & z{+}1\end{bmatrix}^\mathsf{T}$, $\begin{bmatrix}x & y{+}1 & z{+}1\end{bmatrix}^\mathsf{T}$, and $\begin{bmatrix}x{+}1 & y{+}1 & z{+}1\end{bmatrix}^\mathsf{T}$. Based on the sign of the SDF values in those 8 voxels there are 256 different configurations for triangles approximating the zero-level between the voxels. Taking into account the symmetry with respect to inversion of the SDF values

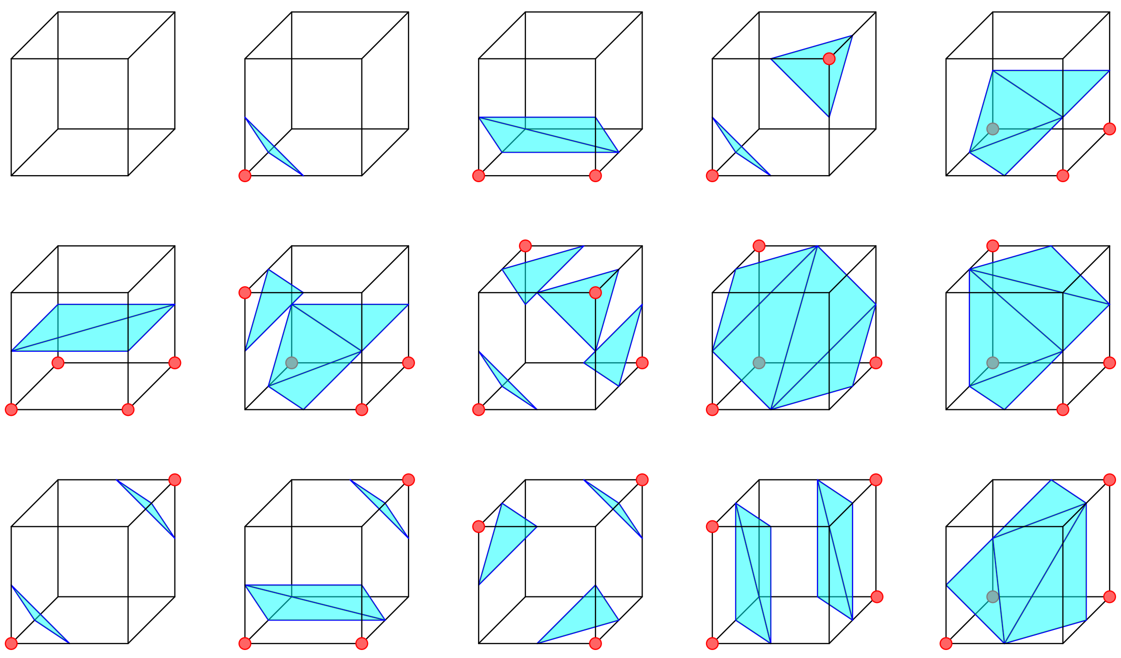
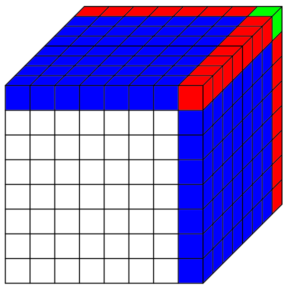
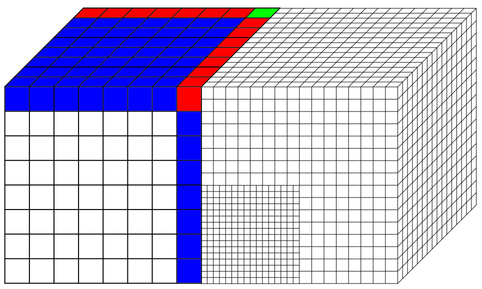
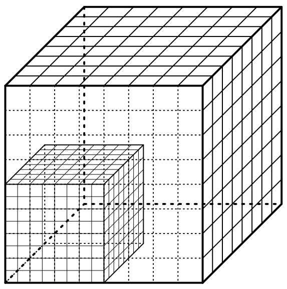
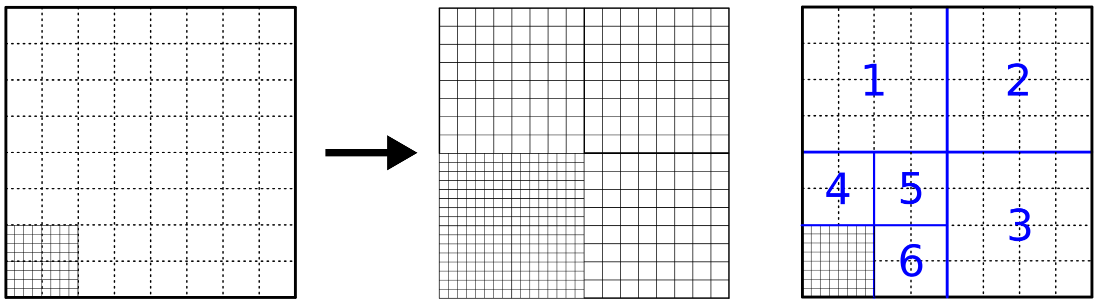


Figure 5.3.: Equivalence classes of Marching Cubes triangulations. Each corner of a cube represents a voxel. The triangles separate the voxels with a negative value, marked with a dot, from the voxels with a non-negative value. The vertices of the triangles lie on the lines between two voxels.


(a) Boundary voxels of a brick


(b) Boundary voxels with a subdivided neighbor brick


(c) Differently sized bricks lying inside one another

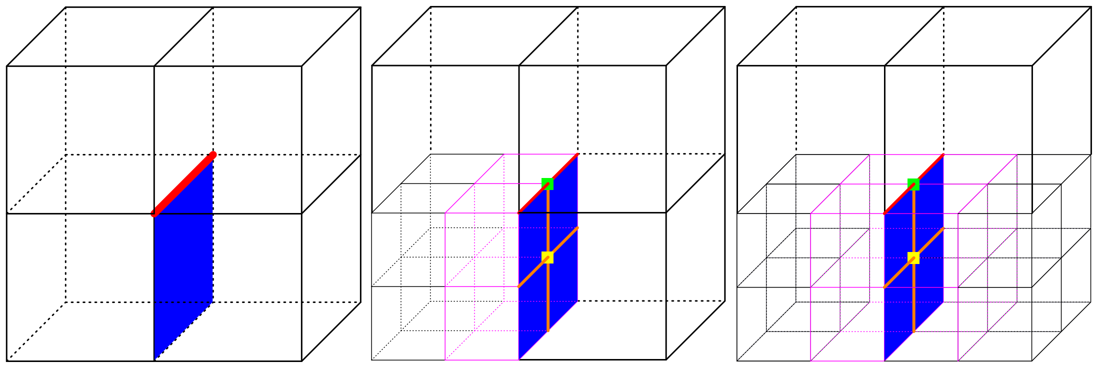


(d) Mesh extraction from partially subdivided bricks. Left: Large branch with brick of low resolution with one allocated brick on a grandchild branch. Middle: Adaptive subdivision of ancestral brick. Right: Different mesh parts associated with branches, while creating the mesh on the resolution of the large brick.

Figure 5.4.: Brick configurations

and to rotation, these 256 configurations reduce to 15 equivalence classes depicted in Figure 5.3. On a dense voxel grid Marching Cubes produces watertight meshes, provided that every voxel has a positive weight, and not taking into account the boundary of the reconstruction volume. In an octree, however, we have the problem that we do not readily know the 7 neighbor voxels for every voxel. Figure 5.4a shows a schematic view of a brick of 8^3 voxels, with voxel coordinates increasing from left to right, bottom to top, and front to back. The voxels are colored differently, depending on their local neighborhood:

- The white colored voxels belonging to the interior of the brick are all inside the brick itself.
- For the blue colored voxels belonging to the faces of the brick, at least one neighboring voxels lies in a different brick.
- For the red colored voxels belonging to the edges of the brick, at least three neigh-



(a) Face and edge without subdivision (b) Face and edge with one subdivided branch (c) Face and edge with 2 subdivided branches

Figure 5.5.: Two different cases of subdivided faces and edges

boring voxels lie in different bricks.

- For the green colored voxels belonging to the corner of the brick, all seven neighboring voxels lie in different bricks.

A naïve approach for computing the mesh from the geometry of a brick would be an iteration over all bricks, looking up the seven higher neighbors for each brick in the tree, and then computing the mesh for the brick. This approach works well if all adjacent bricks have the same size. Otherwise, a brick can have multiple neighbors, as shown in the example in Figure 5.4b. Some of the visible boundary voxels of the left brick have 4 neighbors to the right, others have 16.

Knowing the neighbors of each voxel, the mesh between them can be computed on the finest scale with a linear interpolation of the SDF values of coarser voxels. Since Marching Cubes finds the subvoxel position of the triangle vertices by linear interpolation of the SDF values as well, the resulting mesh would be consistent.

However, the analogous approach to looking up the seven higher neighbors for each brick in the case of uniformly sized bricks is a traversal of the tree for all neighboring voxels of each voxel belonging to a brick boundary in the case of non-uniformly sized bricks. Besides running slowly on current commodity hardware, this approach does not take into account that bricks can also lie into one another, visualized in Figure 5.4c. This is the case if the same surface has been observed from several camera poses at different distances.

Therefore, an iteration over all bricks and the resulting union of the meshes of all bricks would yield a mesh with several instances of the same geometry on top of each other in different resolutions. We could avoid this by adaptively subdividing bricks in the presence of bricks on descendant branches. For example, if a brick is allocated on a grandchild branch, the subdivision of the grandparent brick would create 7 bricks on the grandchild level and 7 bricks on the child level. The middle of Figure 5.4d shows a 2D illustration of this example. While this procedure solves the problem of having parts of the reconstruction volume covered by several voxels, it also wastes memory by supersampling coarse-scale information.

5.4.2. Recursive Formulation

Algorithm 5 traverse_interior(branch b)

```
 1: for all 8 subbranches $b'$ do
 2:   if $b'$ has subbranches then
 3:     traverse_interior(branch $b'$)
 4:   else
 5:     if any brick (also from coarser levels) covers $b'$ then
 6:       extract_mesh_interior($b'$)
 7:     end if
 8:   end if
 9: end for
10: for all 12 pairs of 2 adjacent subbranches $b'$and $b''$ do
11:   traverse_face($b'$,$b''$)
12: end for
13: for all 6 groups of 4 adjacent subbranches $b'$, $b''$, $b'''$ and $b''''$ do
14:   traverse_edge($b'$,$b''$,$b'''$,$b''''$)
15: end for
16: traverse_corner($b'$,...,$b''''''''$)
```

Instead of extracting the mesh brick by brick, we associate mesh parts with branches of the trees. On branches covered by smaller-scaled bricks providing fine-scaled geometry information, a higher-resolution mesh is computed, while larger-scaled bricks provide the rest of the mesh on a smaller resolution. For creating the mesh for the volume of a branch, we start with the smallest brick covering it, and proceed to larger, coarser-scaled bricks whenever the weight values are too small on the finer scale. Instead of computing a mesh for every brick, we compute a mesh for every branch not having a subbranch. However, we cannot simply iterate over all those branches either, as we would have the same problem as with an iteration over all bricks, illustrated in Figure 5.4b. We also have to consider the surface passing through the faces between 2 branches, the edges between 4 branches, and the corners between 8 branches, as shown in Figure 5.5.

Abstracting mesh parts not only from bricks, but also from branches helps us to overcome this problem. The key idea is that while a branch can have common faces and edges with many other branches, every face lies between exactly two branches, and every

Algorithm 6 traverse_face(branches b_1,b_2)

```
1: if $b_1$ or $b_2$ has subbranches then
2:     Let $\{b_1', b_1'', b_1''', b_1''''\}$ and $\{b_2', b_2'', b_2''', b_2''''\}$
       be the branches belonging to the 4 subfaces
3:     traverse_face($b_1'$,$b_2'$)
4:     traverse_face($b_1''$,$b_2''$)
5:     traverse_face($b_1'''$,$b_2'''$)
6:     traverse_face($b_1''''$,$b_2''''$)
7:     traverse_edge($b_1'$,$b_2'$,$b_1''$,$b_2''$)
8:     traverse_edge($b_1'''$,$b_2'''$,$b_1''''$,$b_2''''$)
9:     traverse_edge($b_1'$,$b_2'$,$b_1'''$,$b_2'''$)
10:    traverse_edge($b_1''$,$b_2''$,$b_1''''$,$b_2''''$)
11:    traverse_corner($b_1'$,$b_2'$,$b_1''$,$b_2''$,$b_1'''$,$b_2'''$,$b_1''''$,$b_2''''$)
12: else
13:    if any 2 bricks cover $b_1$ and $b_2$ then
14:        extract_mesh_face($b_1$,$b_2$)
15:    end if
16: end if
```

edge lies between either 3 branches, if the edge lies on a T-junction between one large branch and two small branches, as it is the case for any of the orange edges in Figure 5.5b, or 4 branches in all other cases.

Therefore, we separate the extraction of the mesh into 4 parts: extracting the mesh for the interior of branches, for faces between 2 branches, for edges between 3 or 4 branches, and for corners between 8 branches. By building up the mesh part by part, where each part is a branch interior, a face, an edge, or a corner, instead of brick by brick or branch by branch, we only have to store a constant amount of neighborhood information for each part.

Algorithms 5, 6, 7, and 8 represent a top-down approach for extracting the mesh of the entire tree, if Algorithm 5 is executed on the root branch.

- Algorithm 5 performs the mesh extraction of the 3D interior volume covered by a branch. If it is subdivided, it spawns 8 new 3D volumes, 12 2D faces, 6 1D edges, and one corner point in the middle. When a 2D face between 2 branches is subdivided, it spawns 4 2D faces, 4 1D edges, and one corner point in the middle. This is illustrated in Algorithm 6.

- A face is subdivided if any of its two branches is subdivided. If both branches are subdivided, 4 of the 8 subbranches of each branch share a subface of the original face. This case is illustrated in Figure 5.5c, where the subbranches belonging to the 4 new subfaces are colored pink. If only one branch of a face is subdivided,

Algorithm 7 traverse_edge(branches b_1,b_2,b_3,b_4)

```
1: if $b_1$, $b_2$, $b_3$, or $b_4$ has subbranches then
2:   Let $\{b'_1, b''_1\}$, $\{b'_2, b''_2\}$, $\{b'_3, b''_3\}$, and $\{b'_4, b''_4\}$ be the branches belonging to the 2 subedges
3:   traverse_edge($b'_1$,$b'_2$,$b'_3$,$b'_4$)
4:   traverse_edge($b''_1$,$b''_2$,$b''_3$,$b''_4$)
5:   traverse_corner($b'_1$,$b'_2$,$b'_3$,$b'_4$,$b''_1$,$b''_2$,$b''_3$,$b''_4$)
6: else
7:   if any 4 bricks cover $b_1$, $b_2$, $b_3$ and $b_4$ then
8:     extract_mesh_edge($b_1$, $b_2$, $b_3$,$b_4$)
9:   end if
10: end if
```

the non-divided branch is passed to all recursive calls. For example, if only b_1 in Algorithm 6 is subdivided, then we have $b'_2 = b''_2 = b'''_2 = b''''_2 := b_2$. This case is illustrated in Figure 5.5b. Independent of whether all branches belonging to a face are subdivided, every instance of traverse_face is given two different branches as parameters, albeit possibly of different size. This is not the case for edges.

- When a 1D edge between 4 branches is subdivided within Algorithm 7, it spawns 2 1D edges and one corner point. The central edge in Figure 5.5a has 4 distinct branches, and when subdivided with only one subdivided branch as in Figure 5.5b, each one of its two subedges still has 4 distinct branches, though 3 of them are the same for both subedges. The 4 edges newly created from the subdivision of the face with Algorithm 6 however each have two identical branches, if only one of the branches of a face is subdivided. This also applies to corners created by the subdivision of faces in Algorithm 6, line 11, and by the subdivision of edges in Algorithm 7, line 5.
- A corner point itself is not subdivided, but we have to traverse all branches into the respective 8 smallest branches by tail-recursive calls.

While the four nested recursive algorithms 5 to 8 illustrate the structure of the mesh, the actual computation of the mesh with the Marching Cubes algorithm is performed in separate routines denoted by extract_mesh_interior, extract_mesh_face, extract_mesh_-edge, and extract_mesh_corner in the algorithms. They are only called if the branches belonging to an interior, face, edge, and corner are covered by bricks. In our top-down meshing approach we have to keep track of the smallest bricks covering each branch. Those do not have to be distinct, in fact, all branches of a face, edge or corner can be covered by the same brick. For example, this would be the case in Figure 5.5c, if the entire depicted volume was covered by a brick from a coarser level, and the subdivision of

Algorithm 8 traverse_corner(branches b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8)

```
1: if $b_1$, $b_2$, $b_3$, $b_4$, $b_5$, $b_6$, $b_7$ or $b_8$ has subbranches then
2:     Let $b'_1$, $b'_2$, $b'_3$, $b'_4$, $b'_5$, $b'_6$, $b'_7$, $b'_8$ be the tranches belonging to the corner
3:     traverse_corner($b'_1$, $b'_2$, $b'_3$, $b'_4$, $b'_5$, $b'_6$, $b'_7$, $b'_8$)
4: else
5:     if any 8 bricks cover $b_1$, $b_2$, $b_3$, $b_4$, $b_5$, $b_6$, $b_7$ and $b_8$ then
6:         extract_mesh_corner(branches $b_1$,$b_2$,$b_3$,$b_4$,$b_5$,$b_6$,$b_7$,$b_8$)
7:     end if
8: end if
```

the left and right branch only leads to fine-scaled bricks on the branches not belonging to the blue faces. A coarse brick covering the depicted volume is the smallest brick covering each of the purple-colored branches belonging to the faces, edges, and the corner in the middle.

In addition to the smallest brick for each branch, we also have to keep track of the entire brick hierarchy, to be able to fill holes in the fine-scaled bricks with information from coarse-scaled bricks, if the fine-scaled bricks contain unobserved voxels. This can be done either by keeping a list of bricks for every branch, or by precomputing a brick-tree for the entire reconstruction volume, where a reference to the closest ancestor is stored for every brick. We use the latter one in the data structure described in the next section, enabling us to compute an incremental meshing in real-time.

5.4.3. Incremental Mesh Extraction

The top-down approach described in the last section extracts the mesh of the entire geometry at once. In principle, we could call the traverse_interior routine on the root of our octree after the integration of each RGB-D image. However, as we do not update every brick with the integration of a new RGB-D image, this way of extracting a mesh would result in a redundant extraction of mesh parts belonging to a volumetric representation that has not changed. We would perform an $O(n)$ operation in every mesh update, with n denoting the number of bricks in the tree. If we add an asymptotically constant number of bricks to the tree with each RGB-D image, we would significantly slow down the mesh extraction process over time.

Instead, we propose an incremental method, which only extracts the mesh of those parts of the dense geometry representation changed in the last RGB-D image integration, with a runtime overhead linear in the number of updated bricks, once those bricks have been identified. Combining this mesh with saved mesh of the unchanged parts of the geometry, we have a mesh of the entire geometry as it is after each RGB image integration.

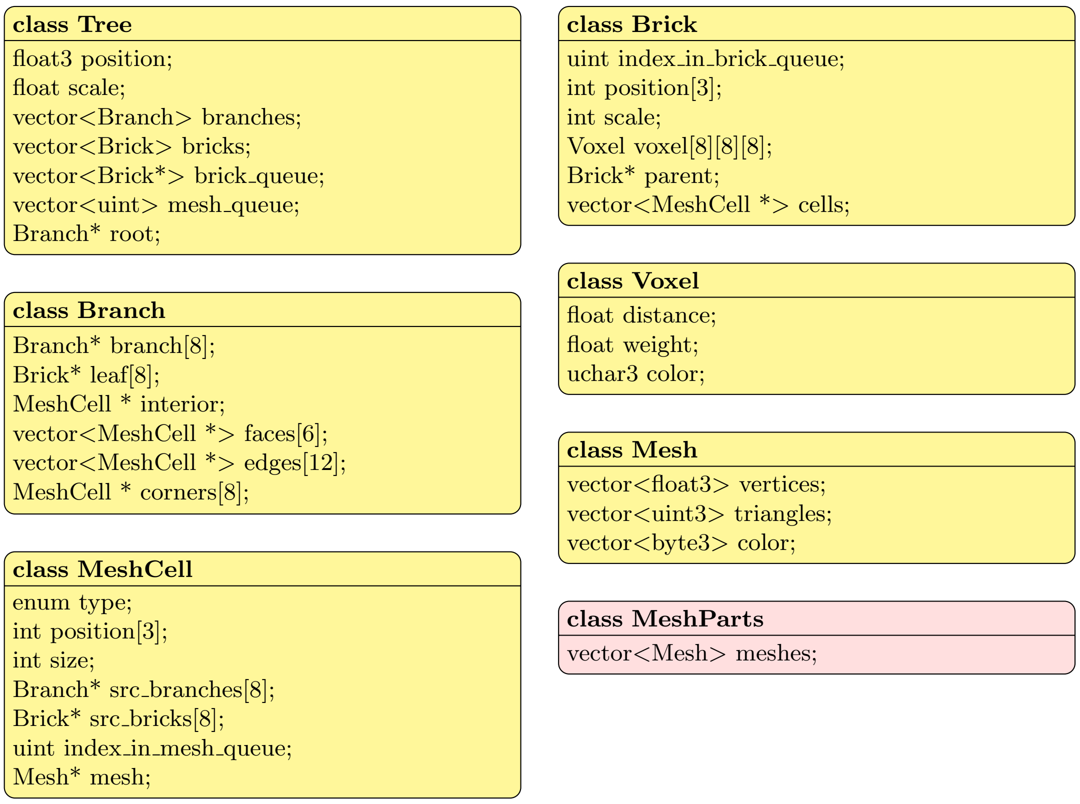
We want to remark that combining old and new parts of the mesh in an array to display a mesh of the entire geometry is always an $O(n)$ operation. However, this combination can take place on a separate machine, limiting the workload of the system performing dense geometry fusion and mesh extraction. The visualization of the mesh of the entire geometry can also be accelerated by methods of space partitioning according to the virtual camera frustum, but we do not further discuss this in this thesis. Instead, we focus on keeping the runtime complexity for mesh extraction with Marching Cubes linear in the number of updated bricks per integrated RGB-D image.

Adding checks on the respective bricks before the calls of the extract_mesh-routines in algorithms 5 to 8 would extract meshes only from updated bricks, but the runtime complexity would still be linear in the number of bricks in the tree, as we would have to look at all branches, and the number of bricks is bounded from above by the number of branches. Furthermore, it does not solve the problem of maintaining the mesh parts extracted at the integration of the last RGB-D image. If we want to update only those parts of the mesh belonging to the updated parts of the volumetric representation, but maintain a consistent partition of the mesh for the entire geometry, we have to keep track of all mesh parts, and we have to be able to efficiently delete those parts of the old mesh that get replaced by updated parts.

We achieve this by explicitly storing the neighborhood relations of every branch in the tree. Instead of storing references to all adjacent branches however, we store references to mesh parts represented by a structure we call “MeshCell”. Each MeshCell stores its type, i.e. if it belongs to the interior of a branch, a face, an edge, or a corner, its position and size in 3D space, references to up to 8 branches and bricks, and a reference to a mesh storing vertices, vertex colors, and triangle indices. In addition to the references to subbranches and bricks we store references to the MeshCells for each branch. Figure 5.6 shows an array-of-structures schematic of our octree data structure including the MeshCell components. As we discussed in Section 5.3.1, we choose structures of arrays over arrays of structures, but the latter are better suited for illustration.

For every branch we store a reference to the MeshCell for the interior of the branch, 6 vectors of references to MeshCells for its 6 faces, 12 vectors with references to MeshCells for its edges, and 8 references to MeshCells for its corners. The vectors for the faces also store references to MeshCells of edges and corners, and the vectors for the edges also store references to MeshCells of corners. An example is the lower right branch in Figure 5.5b, where the vector for its left face contains references to the 4 blue faces, the 4 orange edges, and the yellow corner, and the vector for its upper left edge contains references to the two red edges and the green corner.

In addition to its position, scale, queue index, and voxel data, every brick stores a vector of references to all MeshCells of the branches it covers, and a reference to its parent in the tree, i.e. the next larger brick it is covered by. Each mesh cell stores a reference to the smallest bricks, and with the parent-brick reference, we can traverse the bricks from fine to coarse during the mesh extraction with Marching Cubes. Given the vector of MeshCell references and the brick queue for every RGB-D image, we can



```
class Tree
float3 position;
float scale;
vector<Branch> branches;
vector<Brick> bricks;
vector<Brick*> brick_queue;
vector<uint> mesh_queue;
Branch* root;
```

```
class Branch
Branch* branch[8];
Brick* leaf[8];
MeshCell * interior;
vector<MeshCell *> faces[6];
vector<MeshCell *> edges[12];
MeshCell * corners[8];
```

```
class MeshCell
enum type;
int position[3];
int size;
Branch* src_branches[8];
Brick* src_bricks[8];
uint index_in_mesh_queue;
Mesh* mesh;
```

```
class Brick
uint index_in_brick_queue;
int position[3];
int scale;
Voxel voxel[8][8][8];
Brick* parent;
vector<MeshCell *> cells;
```

```
class Voxel
float distance;
float weight;
uchar3 color;
```

```
class Mesh
vector<float3> vertices;
vector<uint3> triangles;
vector<byte3> color;
```

```
class MeshParts
vector<Mesh> meshes;
```

Figure 5.6.: Proposed data structure. Every branch refers to up to 8 subbranches and 8 bricks. A brick contains 8^3 voxels. A MeshCell stores the triangle mesh of a particular region. Cross references (e.g., from a branch to its MeshCells and vice versa) enable fast traversal during data fusion and meshing.

construct a queue of MeshCells for every RGB-D image in linear time with respect to the number of updated bricks. This MeshCell queue can be processed after each integration of an RGB-D image, it can be processed after the integration of several images since the last mesh extraction, or it can be processed in parallel to the update of the volumetric geometry representation, for which the MeshCell queue is duplicated and one queue is being filled simultaneously while the other one is being emptied. Both the queues for MeshCells and bricks are stored in the tree data structure.

The differently colored MeshParts array containing all meshes can be stored on several platforms. In this case, the platform performing the volumetric fusion and mesh extraction sends the MeshCell queue as well as the updated MeshCells to the other platforms for synchronization of the map, and the MeshCell queue contains references relative to the MeshParts array. In Figure 5.6 we illustrate this fact by storing unsigned integer (UINT) values in the MeshCell queue while storing pointers/references in the brick queue. While it is possible in principle to use global references for all other references, in our structure-of-arrays implementation all references are relative to a predefined global array (see Figure 5.2). This representation is useful if the total amount of available memory is significantly lower than the amount of addressable memory, as it is the case for most 64 Bit architectures, because it reduces the amount of memory necessary for storing the references.

For each MeshCell, the data containing its type, position, size, and references to bricks and to the mesh are used in the methods for mesh extraction. The 8 references to branches are used to efficiently subdivide a MeshCells when one of its branches is subdivided to a size smaller than the MeshCell, as in this case all other branches sharing the MeshCell need to be given references to all new MeshCells stemming from the subdivision.

Compared to storing references to the mesh parts directly with the branches, we get two essential benefits from the MeshCell data structure:

- If the volumetric data of any one of two adjacent branches sharing a face, an edge, or a corner is updated, the mesh for the interior of that branch and the joint mesh parts have to be updated, but not necessarily the interior of the other branch. If the volumetric data of several branches belonging to a face, an edge, or a corner is updated, the joint mesh parts have to be updated only once. Therefore, we have to decouple the update of mesh parts associated with multiple branches from the update of the branches themselves. With the MeshCell data structure, we can create a queue of MeshCells from the queue of updated bricks. This way, every part of the mesh is updated at most once for every RGB-D image.

- In the case of a branch sharing faces or edges with multiple smaller adjacent branches, we do not have to store neighborhood relations of single voxels explicitly. This information is represented by the size and position values stored with each

MeshCell. For each branch belonging to a MeshCell, the voxels that are used for the extraction of the mesh can be easily determined.

5.5. Experimental Evaluation

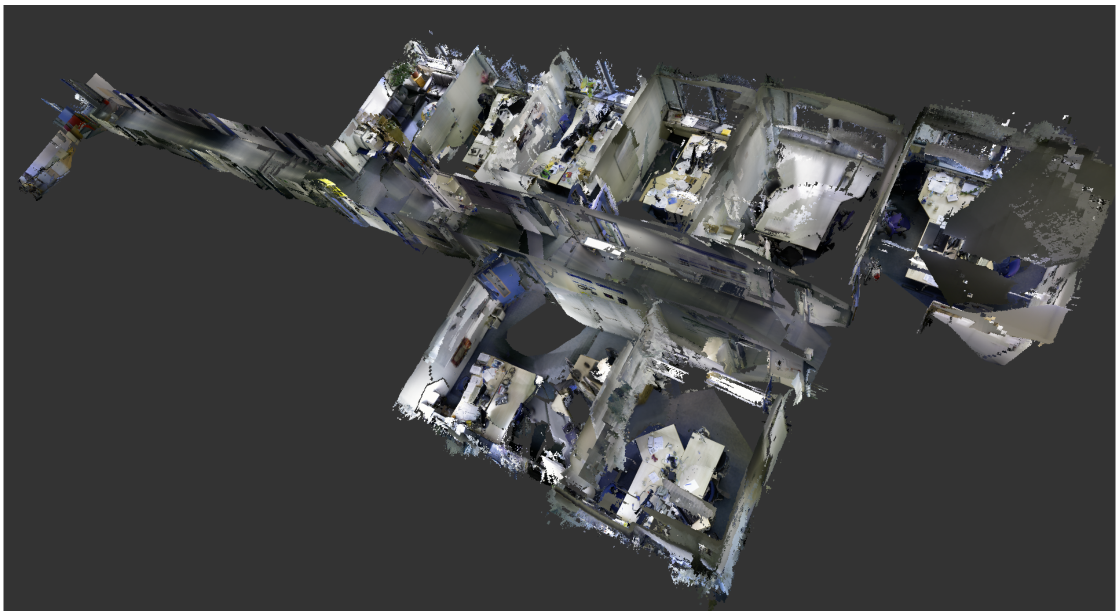
To demonstrate the scalability of our approach, we recorded an office scene of approximately 45m×12m×3.4m in a sequence of 24076 images. We reconstructed the geometry with a maximal voxel resolution of 5mm per voxel. Naturally in our multiscale approach, not all the geometry is stored at the finest resolution. We compute the leaf size at which we stop the tree traversal for a depth value h with the formula

$$s_l = \frac{\left\lfloor \max\left\{\frac{h}{h_{\min}}, 1\right\} s_b \right\rfloor}{s_b}. \tag{5.11}$$

The minimum depth $h_{\min}$ is the depth below which the geometry is reconstructed at maximum resolution, corresponding to a leaf size of $s_l = 1$. For the reconstruction shown below we set $h_{\min}$ to 1m, while we set the truncation threshold T_D to twice the voxel size in every brick. There is no need for specifying the reconstruction volume before starting the integration of depth images, as our implementation supports online growth of the tree by inserting additional branches at the top. Figure 5.7 shows a view of the entire reconstructed geometry and the tree structure it is embedded in. The ceiling of the offices has been removed in a postprocessing step for a better view of the office interior from this position. Figure 5.8 shows RGB images at different camera positions and the corresponding geometry mesh observed from the same positions as well as from different positions. It shows that we are able to reconstruct fine details in the scene where they are available, even though the entire scene fits into approximately 2.5 GB GPU memory, including color. The geometry fusion was performed on an Nvidia GTX680 GPU at an average of 220 frames per second, not including post-processing such as Marching Cubes and visualization.

In Figure 5.10, we illustrate the hierarchical nature of our data structure on the fr3/long_office_household sequence of the Freiburg RGB-D benchmark of Sturm et al. [83]: The left column shows the reconstructed mesh after 50 RGB-D images have been integrated into the SDF representation. The middle column shows the mesh after 170 images, and the right column shows the mesh after 300 images. We see that as the camera observes the geometry from a close-by viewpoint, the resulting geometry gets more detail. This effect is particularly well observable in the two bottom rows. While the two top rows show a flat-shaded mesh representation with and without texture, the two bottom rows show a close-up wireframe representation and a coloring of the different MeshCells as in Figure 5.4.

We evaluate the performance of our method on the GPU quantitatively on the Freiburg RGB-D benchmark as well. The results are shown in Table 5.1, demonstrating that our


(a) Triagle mesh of the reconstructed surface geometry

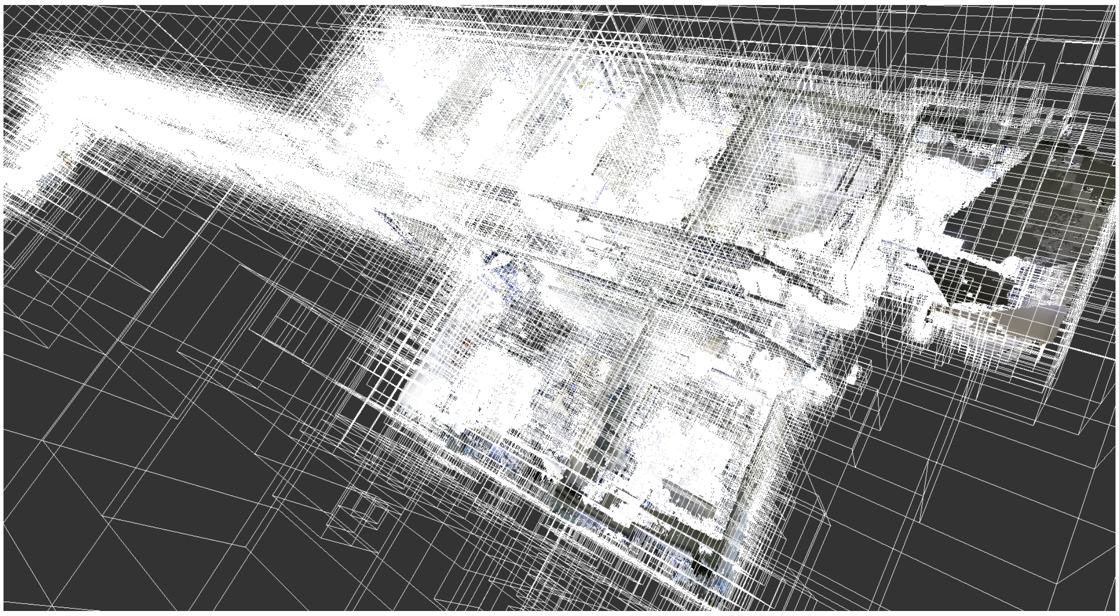
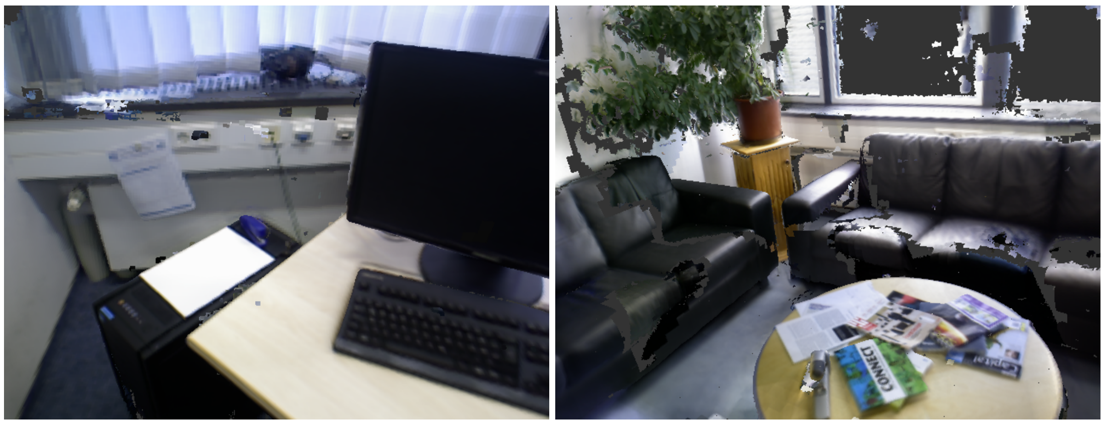

(b) Reconstructed surface geometry embedded in a visualization of the octree branches

Figure 5.7.: Reconstruction of an office floor of approximately 45m×12m×3.4m

(a) Input image


(b) Reconstructed view

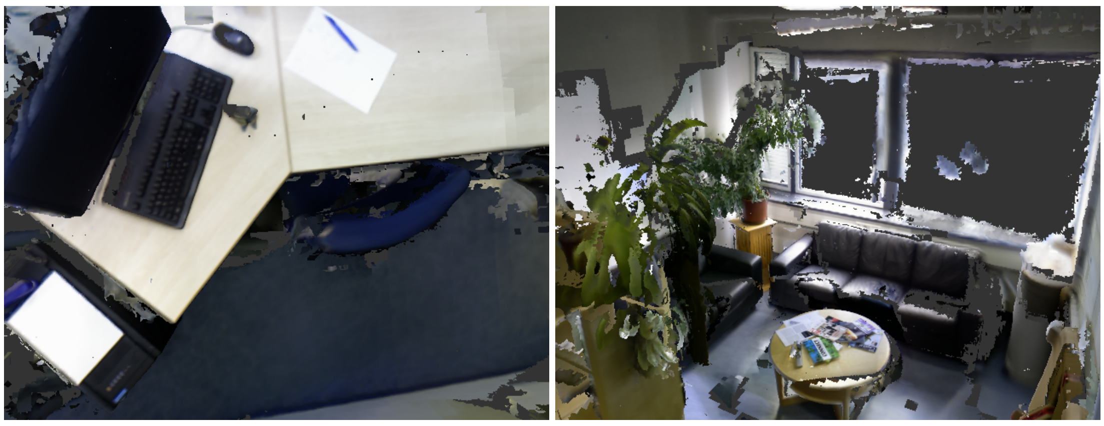

(c) Novel view

Figure 5.8.: Sample reconstructions of two different images from our office floor sequence.

Dataset	Images	Bricks	Processing Time			Memory
			Acquisition	Fusion Total	Fusion Total	
fr1/360	744	917	24s	2.79s	3.7ms	333MB
fr1/desk	5738	987	191s	1.89s	3.3ms	135MB
fr1/desk2	620	1058	20s	2.54s	4.0ms	186MB
fr1/plant	1126	1131	37s	4.15s	3.6ms	221MB
fr1/room	1352	1073	45s	4.96s	3.6ms	453MB
fr1/rpy	694	1049	23s	2.94s	4.2ms	113MB
fr1/teddy	1401	1179	46s	4.78s	3.4ms	248MB
fr1/xyz	792	1054	26s	2.46s	3.1ms	65MB
fr2/desk	2893	1268	96s	12.0s	4.1ms	178MB
fr3/office	2488	1366	82s	10.8s	4.3ms	297MB
average	1785	1108	59s	4.93s	3.7ms	223MB

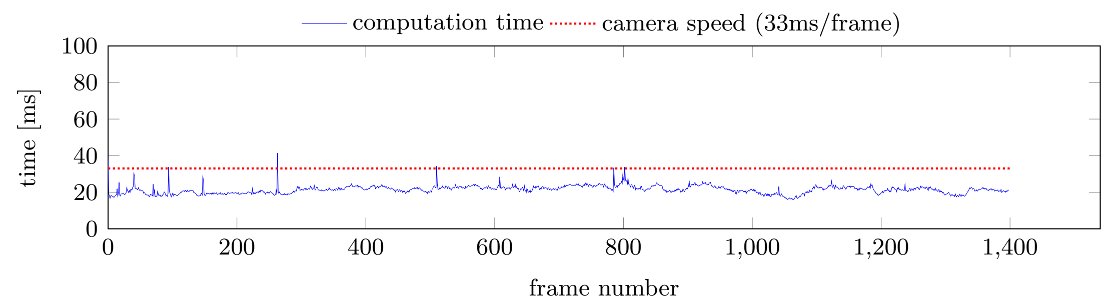
Table 5.1.: Quantitative evaluation of our GPU implementation. From left to right: Dataset, number of images in the dataset, number of bricks used stored in our data structure for the entire dataset, acquisition time of the dataset, total fusion time of the dataset, average fusion time for one frame, memory requirement for the volumetric voxel data.

Dataset	Images	Bricks	Traversal Time		SDF Update Time		
			Naïve	Brick Check	Naïve	Serialized Transform	S.T. + SSE
fr1/360	744	917	30ms	23ms	27ms	22ms	10ms
fr1/desk	5738	987	28ms	23ms	29ms	24ms	11ms
fr1/desk2	620	1058	28ms	24ms	31ms	26ms	11ms
fr1/plant	1126	1131	28ms	23ms	34ms	29ms	12ms
fr1/room	1352	1073	33ms	24ms	35ms	29ms	12ms
fr1/rpy	694	1049	31ms	24ms	34ms	28ms	11ms
fr1/teddy	1401	1179	31ms	23ms	38ms	33ms	13ms
fr1/xyz	792	1054	27ms	24ms	31ms	27ms	11ms
fr2/desk	2893	1268	30ms	23ms	42ms	37ms	14ms
fr3/office	2488	1366	32ms	24ms	44ms	39ms	15ms
Average	1785	1108	30ms	23.5ms	34.5ms	29.4ms	12ms

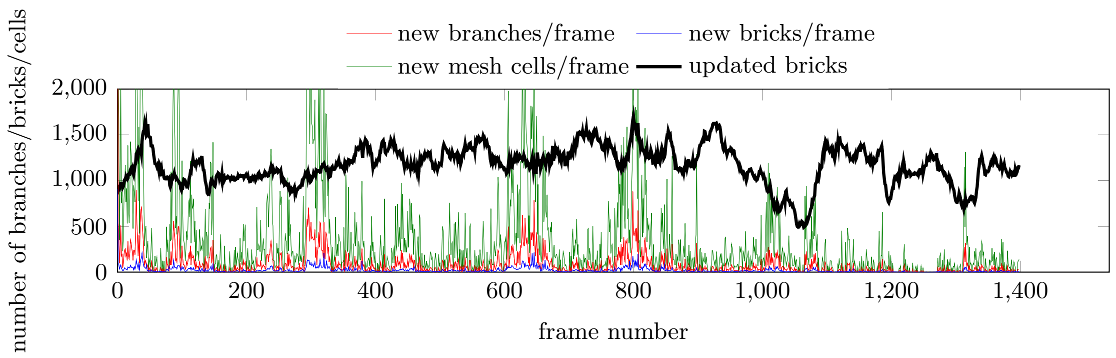
Table 5.2.: Runtime performance gains on the CPU by various optimizations. From left to right: Name of the dataset, number of images in the dataset, average number of updated bricks per frame, average time per frame to find the bricks for a depthmap in the tree without and with checking for double bricks, average time per frame to update the SDF in the bricks performed in a naïve , possibly parallel way, with serially optimized transform, and with serially optimized transform as well as SSE-SIMD instructions.

method is very efficient by means of both runtime and memory demand. On average, our method runs at approximately 270 Hz on the listed benchmark sequences. With an average number of 1785 images per sequence, a joint point cloud with 15 Bytes for each point (12 Bytes for the 3D position and 3 Bytes for RGB color) would require more than 7800 MB for images with a resolution of 640×480 pixels. The average 223 MB required to store a scene with our approach constitute less than 2.9% of this amount of required memory. For most sequences, the final tree had a depth of 10 layers, yielding a volume of 8192^3 voxels on the finest resolution.

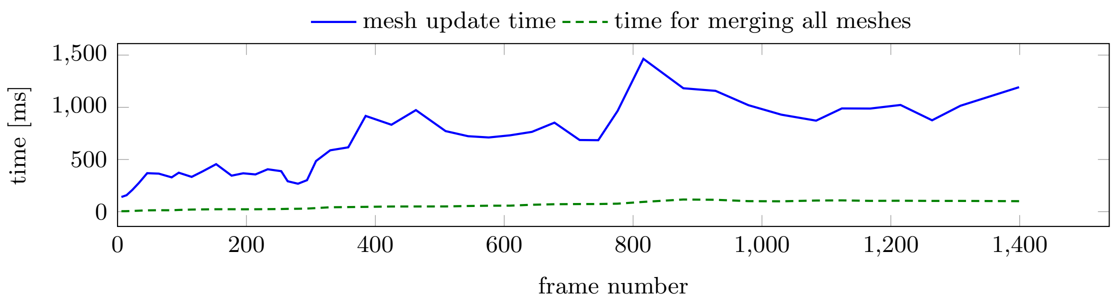
In Table 5.2 we assess the performance gains of the various optimizations discussed in Section 5.3.2. We tested our CPU implementation on the same datasets of the RGB-D benchmark of Sturm et al. [83] as for the runtime assessment of our GPU implementation. We also used the same parameters for voxel size on the finest resolution (5mm) and truncation threshold T_D (twice the voxel size in each brick). We measured the timings on a Laptop CPU of type Intel Core i7-2720QM with 2.2 GHz system clock frequency and 8 GB of system memory. For all sequences, we measured the time for traversing the tree to allocate and queue branches, bricks, and MeshCells, as well as the time for updating



(a) Computation time for data fusion per frame



(b) Number of branch/brick/cell updates per frame



(c) Computation time (latency) for incremental meshing

Figure 5.9.: Evaluation of the runtime on the fr1/teddy sequence on an Intel Core i7-2720QM. (a) computation time for the geometry fusion of every frame, compared to the camera frame rate. (b) amount of updated bricks for every frame as well as the number of newly created branches, bricks, and mesh cells. (c) Evaluation of the runtime for meshing (corresponding to display latency).

the SDF values in the voxels of the queued bricks. For traversing and queuing bricks, the naïve algorithm required an average of 20.7 ms per RGB-D frame, while our optimized versions required 12.9 ms, yielding a speedup of 37%. Updating the SDF values took another 27.6 ms for the naïve implementation, buy only 8.9 ms after optimization using the serialized transform and SSE instructions, i.e. a speedup of 67%. In total, the optimized integration of an RGB-D image requires an average 21.8 ms, corresponding to a processing speed of approximately 45 Hz.

To study the sequential behavior of our CPU implementation in more detail, we also evaluate the computational load over time on the fr1/teddy sequence. Figure 5.9 shows the results. At the top, we plot the total time needed per RGB-D frame for the traversal of the tree to add and queue new branches, bricks, and MeshCells, and for updating the distance, weight, and color values in every voxel, i.e., the work performed by the "main" thread. As can be seen, the processing time stays below 33 ms for almost all frames.

Beneath we show how the number of newly created branches, bricks, and MeshCells varies over time, depending on the camera motion and the amount of newly discovered geometry. A peak in these values is due to the fact, that the camera visits "unknown territory" at this time. In contrast, the number of updated bricks per RGB-D frame remains more or less constant around 1,100, which is closely related to the computation time of that frame.

At the bottom of Figure 5.9 we show the processing time of the meshing queue, and thus, the latency at which the updated mesh becomes available. The latency varies between 0.5 s and 1.5 s, while the final $O(n)$ merging of the meshes of all MeshCells, updated or not, for visualization in OpenGL grows monotonically to 0.1 s after 1,400 frames. The final mesh consists of roughly 3.3 million triangles on 2.8 million vertices.

When assessing the performance of our incremental meshing approach, that is running on a separate thread in parallel to the update of the volumetric data, we have to keep in mind, that updating the mesh parts covered by a brick is generally much slower than updating the distance, weight, and color values in the brick, as the former involves relations between several voxels and append operations on the resulting mesh parts by means of both vertices and vertex indices for the triangles. Both severely limit parallelizability inside the update of a MeshCell. When running the meshing in a separate thread, we fill a queue of MeshCells in the thread performing the volumetric updates and empty the same queue. Therefore, the faster the RGB-D images are integrated into the volumetric data, the more the meshing thread will lag behind. Three things reducing the lag of the meshing thread are limitation of the rate of RGB-D image integration, parallelization of the meshing across MeshCells, and a slow moving camera resulting in several successive RGB-D images queuing the same MeshCells.

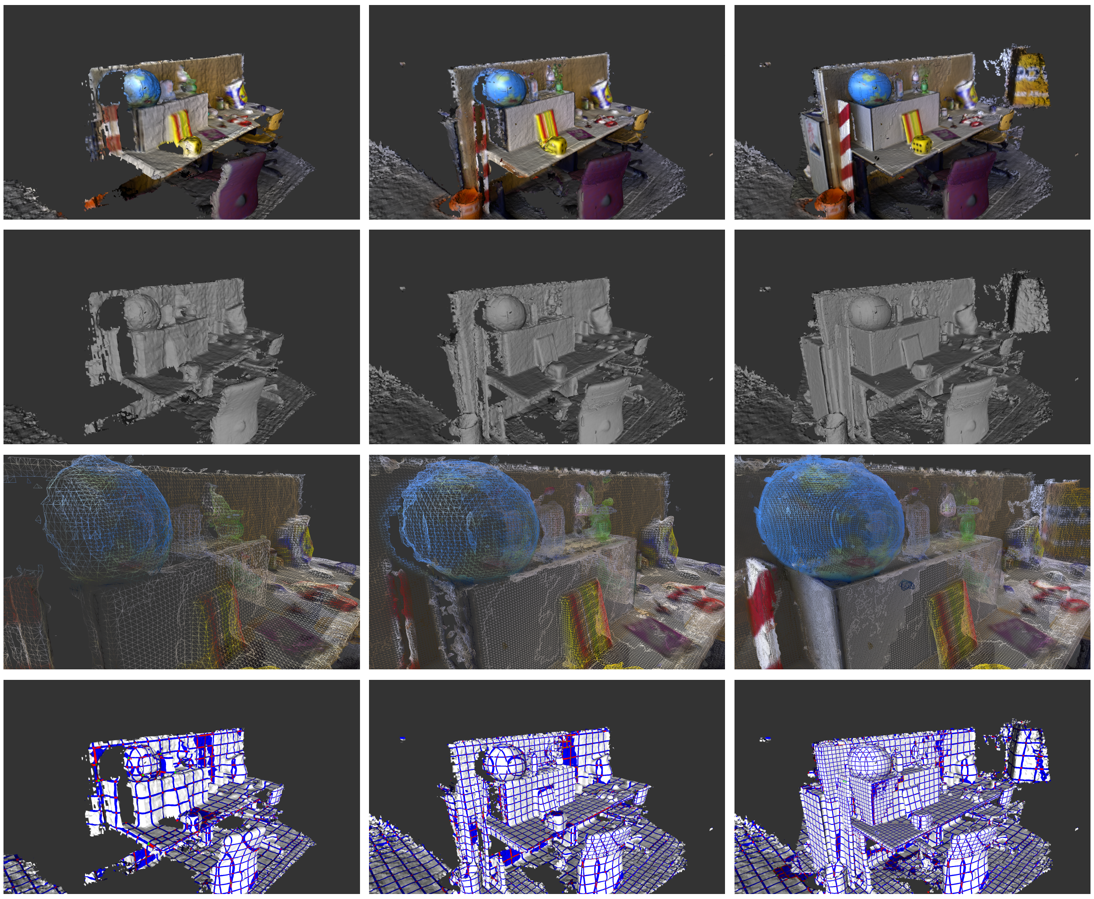


Figure 5.10.: Geometry evolution of the fr3/long_office_household sequence. Left to right: Result after the first 50, 170, and 300 frames. Top to bottom: Geometry and texture, geometry only, textured wireframe, and visualization of different mesh parts as in Figure 5.4. The level of detail in the geometry increases as the camera records more close-up images.

5.6. Conclusion

We have introduced a novel method for reconstructing a 3D surface from a sequence of RGB-D images. Our method represents the surface implicitly as the zero-level of a volumetric signed distance function. Other than previous methods, we choose a sparse representation of the reconstruction volume based on an octree. This enables our method to run in an online setup without a predefined reconstruction volume. The reconstruction of the joint surface follows the camera trajectory, and for each RGB-D image, there is only an asymptotically constant number of voxels, that need to be updated on each level of the octree. Nevertheless, any part of the reconstruction volume can be revisited an arbitrary number of times without loss of reconstruction accuracy. The octree-based method also facilitates a multi-level representation of the geometry, based on the closest distance from a point on the surface to any camera. Finally, due to the compact representation of the geometry, our method outperforms previous methods both by means of computational speed and memory demand. To the best of our knowledge, it is the first published method capable of running in real-time on one core of a standard laptop CPU. This includes storing an RGB color value in each voxel, facilitating an online update of the geometry texture as well.

In addition, we have introduced a method for an incremental mesh extraction, facilitating map transmissions with an even lower memory footprint. The mesh extraction follows the volumetric surface reconstruction in that its computational complexity is asymptotically constant for the geometry update induced by each depth image. Our method is able to extract a mesh of the updated geometry in roughly 1sec on a separate CPU core.

6. Conclusion

In this thesis, we have introduced efficient algorithms for a variety of correspondence problems in geometry and motion estimation from image sequences.

For half of the problems addressed, algorithmic efficiency means finding an approximate solution to a given problem with a better asymptotic runtime or memory complexity than the state of the art. This is the case for the energy-based optical flow computation without linearization of the data term, and the octree-based volumetric geometry fusion from depth image sequences.

For the other half of the problems, efficiency means computing results in online, real-time scenarios on limited commodity hardware. This is the case for estimating the camera motion in 3D space from a sequence of RGB-D images and for computing the joint geometry observed in these images in real-time on a CPU with the ability to compute an incremental light-weight mesh representation for visualization on a remote platform.

For the variational optical flow problem with a non-linearized data term, we proved that it can be NP-hard for some parameter settings, even with convex regularity terms. In order to solve the problem in feasible time, we therefore have to come up with an approximate solution. While most approximate solutions rely on a linearized data term, we proposed an alternative in Chapter 3. By decoupling the data term from the regularity term, we were able to retain the original non-convex dataterm, and we demonstrated that we can overcome the problem of small-scale structures vanishing on coarse resolutions within a coarse-to-fine scheme required to estimate large motions with a linearized data term.

In addition, we have demonstrated that keeping the original non-linearized data term allows us to incorporate patch-based data terms into our variational optical flow scheme, that would otherwise lose much of their descriptive power under linearization. We evaluated the performance of our algorithm with different patch-based difference metrics, such as the sum of absolute differences of a patch, normalized cross correlation, census and rank metrics.

In Chapter 4 we have shown how to estimate the trajectory of a moving RGB-D camera in real-time. The energy formulation used for this motion estimation problem is equivalent to the formulation used for disparity estimation in Chapter 3, except we optimize the energy with respect to a different variable. For slow continuous camera motions, our method proved to be more precise than standard ICP methods, while skipping their computationally costly point-to-point correspondence estimation.

Given both disparity or depth images and their corresponding camera poses, potentially estimated by the methods introduced in Chapters 3 and 4, we have shown how to reconstruct a joint representation of the geometry observed in these images in Chapter 5. We employ an implicit representation capable of capturing changes in topology of the reconstructed surface geometry, however, in contrast to previous methods based on this representation, our method does not require a pre-determined area containing the geometry to be reconstructed, and can therefore be applied in online scenarios where not all data is present up front. By retaining a sparse octree-based representation of the reconstruction volume, we are also able to keep the runtime and memory space complexity linear with respect to the number of integrated images. Additionally, the hierarchical octree representation allows us to represent the geometry on different resolutions based on their distance to the camera.

We have demonstrated that we can reconstruct geometry on large scales on current commodity GPU hardware with a speed many times the image rate of current commodity depth cameras, while additionally being able to reconstruct the RGB surface texture volumetrically, provided the depth camera provides color information as well.

Finally, we have presented how to taylor our approach to serial SIMD architectures, enabling our method to run on current laptop CPUs in real-time. By computing an incremental mesh representation from the reconstruction volume, we are able to abstract the visualization of the surface from its volumetric representation, enabling a visualization on a remote machine.

6.0.1. Outlook and Future Work

All the methods presented in this thesis constitute first proofs of concept in their respective research areas. In our opinion, the most manifest directions for further research are the following:

- **Construction of a joint pipeline including all the proposed methods.** In this thesis, we have evaluated our decoupling scheme for variational optical flow estimation. We can evaluate it for disparity estimation, compare it to methods based on functional lifting with respect to accuracy and hardware demand, and use the resulting disparity images instead of images from and active depth sensor as input to our method for camera motion estimation. Both the estimated disparity images and the camera trajectory can then be used as inputs for our surface reconstruction algorithm.

- **Robust loss functions and sparsification for camera pose estimation.** We can replace the quadratic loss function used in our formulations with more robust loss functions. Robust loss functions would only introduce another factor in the chain of derivatives for the linearized energy. We expect to attain greater robustness against outliers with this formulation. Furthermore, we could restrict the method from all pixels in the image to only a salient subset of pixels, in order to make the method even faster. As a criterion for using a pixel in the formulation we could use the distance to a line in the image. While we would no longer use all the information in the image, we would still use more pixels than state-of-the-art feature-based methods, which are mostly based on corner detection, and avoid computationally cumbersome descriptor matching steps.

- **Computing the intersection of the viewing frustum with the octree for large-scale surface reconstruction.** In this thesis, we have limited the voxels and bricks allocated in the octree to a small band around the observed surface. We could instead use efficient methods from the computer graphics literature for intersecting a camera frustum with an axis-aligned cube in order to allocate all bricks in the camera frustum in front of the observed surface. This method would still remain sparse, only storing what is actually observed. However, it would include a more explicit representation of empty space, similar to those methods based on a dense voxel grid. While this would require more memory than the method proposed here, it would reduce the amount of flying-triangle artifacts in the reconstructed surface.

- **Frame-to-model tracking.** Instead of using a depth or disparity image to compute the 3D surface S in (4.1), we raycast the SDF in the octree, and only use the depth image for integration into the joint geometry. That way, we would expect better tracking results, as the model removes much of the noise present in the individual depth images. Combined with the sub-sampling strategy mentioned above, we expect to still achieve real-time performance.

A. Appendix

A. List of Acronyms

- **AAE** Average Angular Error (3.31b).
- **AEE** Average End-Point Error (3.31a).
- **CPU** Central Processing Unit 1.
- **GPU** Graphics Processing Unit 1.
- **HSV** Hue-Saturation-Value 3.4.1.
- **ICP** Iterative Closest Point 4.1.
- **MR** Mixed Reality 5.4.
- **MUTEX** Mutual Exclusion 5.3.1.
- **NCC** Normalized Cross-Correlation (2.37).
- **RGB-D** Red, Green, Blue, and Depth 1.
- **SAD** Sum of Absolute Differences 2.5.
- **SDF** Signed Distance Function 5.1.
- **SIMD** Single-Instruction-Multiple-Data 1.
- **SSD** Sum of Squared Differences 2.5.
- **SSE** Streaming SIMD Extensions 5.3.2.
- **TV** Total Variation 2.6.1.
- **UINT** Unsigned Integer 5.4.3.

B. Dense Motion Estimation Without a Coarse-to-fine Scheme

B.1. Sub-pixel Optimization - Derivation of the Hyperbola Equation

$$\begin{aligned}
0 &= \mathcal{I}_x\alpha_1 + \mathcal{I}_y\alpha_2 + \mathcal{I}_{xy}\alpha_1\alpha_2 + \mathcal{I}_r \\
\alpha_2 &= -\frac{\mathcal{I}_x\alpha_1 + \mathcal{I}_r}{\mathcal{I}_{xy}\alpha_1 + \mathcal{I}_y} \\
&= -\frac{\mathcal{I}_x\alpha_1 + \mathcal{I}_r}{\mathcal{I}_{xy}\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)} \\
&= -\frac{\mathcal{I}_x\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}} - \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right) + \mathcal{I}_r}{\mathcal{I}_{xy}\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)} \\
&= -\frac{\mathcal{I}_x\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right) + \frac{\mathcal{I}_{xy}\mathcal{I}_r - \mathcal{I}_x\mathcal{I}_y}{\mathcal{I}_{xy}}}{\mathcal{I}_{xy}\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)} \\
&= -\frac{\mathcal{I}_x\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)}{\mathcal{I}_{xy}\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)} + \frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)} \\
&= -\frac{\mathcal{I}_x}{\mathcal{I}_{xy}} + \frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2}\frac{1}{\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)} \qquad \text{(A.1)}
\end{aligned}$$

B.2. Sub-pixel Optimization - Minimization of the Energy on the Hyperbola

On the hyperbola we have

$$\mathcal{I}_x\alpha_1 + \mathcal{I}_y\alpha_2 + \mathcal{I}_{xy}\alpha_1\alpha_2 + \mathcal{I}_r = 0 \qquad \Leftrightarrow \qquad \alpha_2 = -\frac{\mathcal{I}_x}{\mathcal{I}_{xy}} + \frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2}\frac{1}{\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)}$$

Substituting the hyperbola equation into the formulation of the coupling term, we get

$$\begin{aligned}
E(\alpha_1) &= \frac{1}{2\theta}\left((\alpha_1 - \eta_1)^2 + (\alpha_2 - \eta_2)^2\right) \\
&= \frac{1}{2\theta}\left((\alpha_1 - \eta_1)^2 + \left(-\frac{\mathcal{I}_x}{\mathcal{I}_{xy}} + \frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2}\frac{1}{\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)} - \eta_2\right)^2\right), \text{(A.2)}
\end{aligned}$$

which we have to minimize in α_1. The first derivative w.r.t. α_1 is

$$\frac{1}{\theta}\left(\alpha_1 - \eta_1 + \left(-\frac{\mathcal{I}_x}{\mathcal{I}_{xy}} + \frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)} - \eta_2\right)\left(-\frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)^2}\right)\right) \tag{A.3}$$

$$= \frac{1}{\theta}\left(\alpha_1 - \eta_1 + \left(\frac{\mathcal{I}_x}{\mathcal{I}_{xy}} + \eta_2\right)\frac{\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r}{\mathcal{I}_{xy}^2\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)^2} - \frac{\left(\mathcal{I}_x\mathcal{I}_y - \mathcal{I}_{xy}\mathcal{I}_r\right)^2}{\mathcal{I}_{xy}^4\left(\alpha_1 + \frac{\mathcal{I}_y}{\mathcal{I}_{xy}}\right)^3}\right), \tag{A.4}$$

which is a polynomial of degree 3.

B.3. Sub-pixel Optimization - Algorithm for the Analytic Optimization

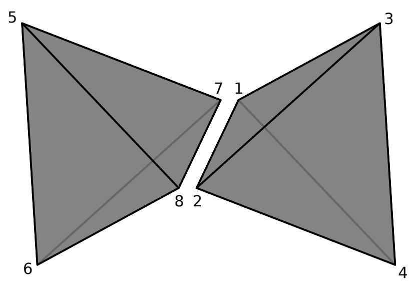
```
if $\frac{1}{\theta} > |\mathcal{I}_{xy}|$ then
  {The energy is convex.}
  Compute the minima $(\alpha_{1+}, \alpha_{2+})$ and $(\alpha_{1-}, \alpha_{2-})$ with Equations (3.43).
  if $\rho(\alpha_{1+}, \alpha_{2+}) < 0$ then
    Compute $(\alpha_{1+}, \alpha_{2+})$ on the hyperbola.
    Project the solution on the hyperbola into $[0,1]^2$.
    if The hyperbola does not intersect $[0,1]^2$ then
      Recompute $(\alpha_{1+}, \alpha_{2+})$ on the boundary of $[0,1]^2$.
    end if
  else
    Compute the projection $\pi(\alpha_{1+}, \alpha_{2+})$ of $(\alpha_{1+}, \alpha_{2+})$ into $[0,1]^2$.
    if $\rho(\pi(\alpha_{1+}, \alpha_{2+})) < 0$ then
      Recompute $(\alpha_{1+}, \alpha_{2+})$ on the boundary of $[0,1]^2$.
    end if
  end if
  if $\rho(\alpha_{1-}, \alpha_{2-}) < 0$ then
    Compute $(\alpha_{1-}, \alpha_{2-})$ on the hyperbola.
    Project the solution on the hyperbola into $[0,1]^2$.
    if The hyperbola does not intersect $[0,1]^2$ then
      Recompute $(\alpha_{1-}, \alpha_{2-})$ on the boundary of $[0,1]^2$.
    end if
  else
    Compute the projection $\pi(\alpha_{1-}, \alpha_{2-})$ of $(\alpha_{1-}, \alpha_{2-})$ into $[0,1]^2$.
    if $\rho(\pi(\alpha_{1-}, \alpha_{2-})) < 0$ then
      Recompute $(\alpha_{1-}, \alpha_{2-})$ on the boundary of $[0,1]^2$.
    end if
  end if
else
```

```
    {The energy is convex-concave and the minimum lies at infinity or on the hyperbola.}
    Compute $(\alpha_{1+}, \alpha_{2+})$ and $(\alpha_{1-}, \alpha_{2-})$ on the boundary of $[0,1]^2$.
    **if** The hyperbola intersects $[0,1]^2$ **then**
      Compute $(\alpha_{1+}, \alpha_{2+})$ and $(\alpha_{1-}, \alpha_{2-})$ on the hyperbola.
      Check for possible better solution in the intersection.
    **end if**
  **end if**
```

C. Camera Tracking on RGB-D Sequences

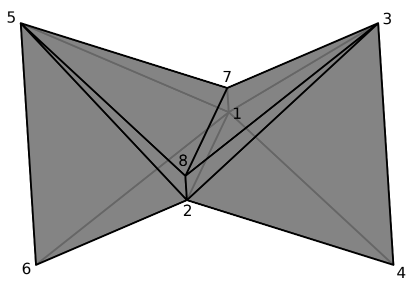
C.1. Explicit Formulation of the 6×1 Constraints for Camera Tracking

$$\begin{aligned}
&\nabla\mathcal{I}^{\mathsf{T}}(w(\mathbf{p}))\frac{\mathrm{d}\pi}{\mathrm{d}\mathbf{P}}(\mathbf{G}(\mathbf{p}))\,(\widehat{\omega_{\text{inc}}}\mathbf{G}(\mathbf{p}) + v_{\text{inc}}) \\
=\;& \begin{bmatrix} \partial_x\mathcal{I}(w(\mathbf{p})) \\ \\ \partial_y\mathcal{I}(w(\mathbf{p})) \end{bmatrix}^{\mathsf{T}}
\begin{bmatrix} \frac{f_x}{G_Z} & 0 \\ 0 & \frac{f_y}{G_Z} \\ -\frac{f_x G_X}{G_Z^2} & -\frac{f_y G_Y}{G_Z^2} \end{bmatrix}^{\mathsf{T}}
\begin{bmatrix} 0 & -G_Z & G_Y(\mathbf{p}) \\ G_Z & 0 & -G_X(\mathbf{p}) \\ -G_Y(\mathbf{p}) & G_X(\mathbf{p}) & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{\mathsf{T}}
\begin{bmatrix} \omega_{1\text{inc}} \\ \omega_{2\text{inc}} \\ \omega_{3\text{inc}} \\ v_{1\text{inc}} \\ v_{2\text{inc}} \\ v_{3\text{inc}} \end{bmatrix} \\
=\;& \begin{bmatrix}
-\partial_y\mathcal{I}(w(\mathbf{p}))f_y - \frac{\partial_x\mathcal{I}(w(\mathbf{p}))f_x G_X(\mathbf{p}) + \partial_y\mathcal{I}(w(\mathbf{p}))f_y G_Y(\mathbf{p})}{G_Z^2(\mathbf{p})} G_Y(\mathbf{p}) \\
\partial_x\mathcal{I}(w(\mathbf{p}))f_x + \frac{\partial_x\mathcal{I}(w(\mathbf{p}))f_x G_X(\mathbf{p}) + \partial_y\mathcal{I}(w(\mathbf{p}))f_y G_Y(\mathbf{p})}{G_Z^2(\mathbf{p})} G_X(\mathbf{p}) \\
\frac{\partial_y\mathcal{I}(w(\mathbf{p}))f_y G_X(\mathbf{p}) - \partial_x\mathcal{I}(w(\mathbf{p}))f_x G_Y(\mathbf{p})}{G_Z(\mathbf{p})} \\
\frac{\partial_x\mathcal{I}(w(\mathbf{p}))f_x}{G_Z(\mathbf{p})} \\
\frac{\partial_y\mathcal{I}(w(\mathbf{p}))f_y}{G_Z(\mathbf{p})} \\
-\frac{\partial_x\mathcal{I}(w(\mathbf{p}))f_x G_X(\mathbf{p}) + \partial_y\mathcal{I}(w(\mathbf{p}))f_y G_Y(\mathbf{p})}{G_Z^2(\mathbf{p})}
\end{bmatrix}^{\mathsf{T}}
\begin{bmatrix} \omega_{1\text{inc}} \\ \omega_{2\text{inc}} \\ \omega_{3\text{inc}} \\ v_{1\text{inc}} \\ v_{2\text{inc}} \\ v_{3\text{inc}} \end{bmatrix} \\
=\;& \mathcal{C}_{\text{acc}}(\mathbf{p})^{\mathsf{T}}\xi_{\text{inc}} \qquad \text{(A.5)}
\end{aligned}$$



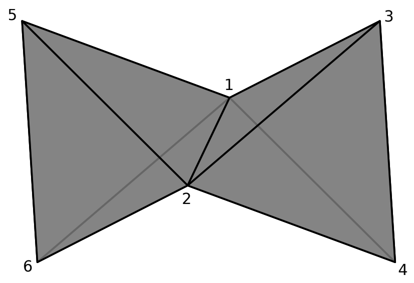
```
OBJ Code
v 0.05 1 0
v 0.05 0 1
v 1 1 1
v 1 0 0
v -1 1 1
v -1 0 0
v -0.05 1 0
v -0.05 0 1
f 1 2 3
f 2 1 4
f 8 7 5
f 7 8 6
f 3 4 1
f 4 3 2
f 6 5 7
f 5 6 8
```

(a) Closed Orientable Surface 1



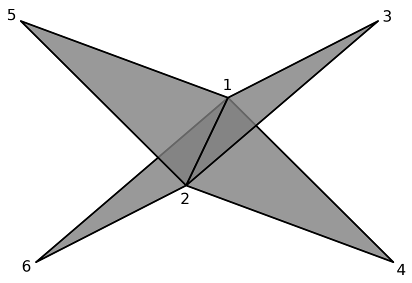
```
OBJ Code
v 0 0.95 -0.05
v 0 -0.05 0.95
v 1 1 1
v 1 0 0
v -1 1 1
v -1 0 0
v 0 1.05 0.05
v 0 0.05 1.05
f 7 8 3
f 2 1 4
f 8 7 5
f 1 2 6
f 3 4 1
f 4 3 2
f 6 5 1
f 5 6 2
f 8 2 3
f 2 8 5
f 1 7 3
f 7 1 5
```

(b) Closed Orientable Surface 2



```
OBJ Code
v 0 1 0
v 0 0 1
v 1 1 1
v 1 0 0
v -1 1 1
v -1 0 0
f 1 2 3
f 2 1 4
f 2 1 5
f 1 2 6
f 3 4 1
f 4 3 2
f 6 5 1
f 5 6 2
```

(c) Degenerate Case



```
OBJ Code
v 0 1 0
v 0 0 1
v 1 1 1
v 1 0 0
v -1 1 1
v -1 0 0
f 1 2 3
f 2 1 4
f 2 1 5
f 1 2 6
```

(d) Degenerate Open Orientable Surface

Figure A.1.: **Orientable surface configurations**. The left side of each subfigure shows an orthogonal projection of the triangle mesh described by the Wavefront OBJ format on the right side. The triangles have a right-handed orientation, in that a triangle **f** a b c has the surface normal $(b-a) \times (c-a)$.

D. Large-Scale 3D Reconstruction from RGB-D Sequences

D.1. Degenerate Case of Edge-Triangle Configurations

In Section 5.1 we have mentioned that orientable triangle meshes can have degenerate cases containing edges shared by more than two triangles. Figure A.1 shows such a case. Figures A.1a and A.1b show two valid orientable surfaces. Figure A.1c can be seen as their degenerate limiting case, where the edges $(1, 2)$ and $(7, 8)$ are fused. Figure A.1d then shows an open surface, that can still be regarded as a valid degenerate case. This example demonstrates the importance of the individual triangle orientations. If either the first and fourth, or the second and third triangle were flipped, the mesh would self-cross in edge $(1, 2)$.

D.2. Error Intervals of Disparity-based Depth Sensors

The disparity interval

$$d \in [d^* - \sigma, d^* + \sigma] \tag{A.6}$$

is mapped to the depth interval

$$[h(d^* - \sigma), h(d^* + \sigma)]\,. \tag{A.7}$$

Using the notation $h^* := h(d^*)$, we get

$$\begin{aligned}
[h(d^* - \sigma), h(d^* + \sigma)] &= \left[\frac{Bf_x}{d^* - \sigma}, \frac{Bf_x}{d^* + \sigma}\right] \\
&= \left[\frac{Bf_x}{d^*} + \frac{Bf_x}{d^* - \sigma} - \frac{Bf_x}{d^*}, \frac{Bf_x}{d^*} + \frac{Bf_x}{d^* + \sigma} - \frac{Bf_x}{d^*}\right] \\
&= \left[h^* + Bf_x\left(\frac{d^* - (d^* - \sigma)}{d^*\,(d^* - \sigma)}\right), h^* + Bf_x\left(\frac{d^* - (d^* + \sigma)}{d^*\,(d^* + \sigma)}\right)\right] \\
&= \left[h^* + \frac{h^*\sigma}{d^* - \sigma}, h^* - \frac{h^*\sigma}{d^* + \sigma}\right] \\
&= \left[h^* + \frac{Bf_x h^*\sigma}{Bf_x d^*\left(1 - \frac{\sigma}{d^*}\right)}, h^* - \frac{Bf_x h^*\sigma}{Bf_x d^*\left(1 + \frac{\sigma}{d^*}\right)}\right] \\
&= \left[h^* + \frac{h^{*2}\sigma}{Bf_x - h^*\sigma}, h^* - \frac{h^{*2}\sigma}{Bf_x + h^*\sigma}\right]. \qquad (A.8)
\end{aligned}$$

D.3. Intersecting a Cube with a Branch in the Tree

Having traversed the tree from its root to a branch b, we know the cubic volume it covers, parametrized by its position in the tree $\mathbf{o}_b$ and its size s_b. Figure A.2 shows a 1D illustration of an axis-aligned cube, parametrized by its lower and upper corners $c^{\min}$ and $c^{\max}$, intersecting the volume covered by a branch having subbranches. The volumes covered by its subbranches in this 1D example go from $\mathbf{o}_b$ to $\mathbf{o}_b + \frac{s_b}{2}$ and from $\mathbf{o}_b + \frac{s_b}{2}$ to $\mathbf{o}_b + s_b$. We assume that the cube intersects the volume covered by b, therefore, cases 1 and 7 in figure A.2 do not happen. In cases 2 to 6, the question whether the cube intersects the lower subbranch is solely depending on whether $c^{\min} < \mathbf{o}_b + \frac{s_b}{2}$. Analogously, the question whether the cube intersects the upper subbranch is depending on whether $c^{\max} > \mathbf{o}_b + \frac{s_b}{2}$. In a 3D octree,the question whether the 3D cube intersects a subbranch is simply a conjunction of the three 1D checks.

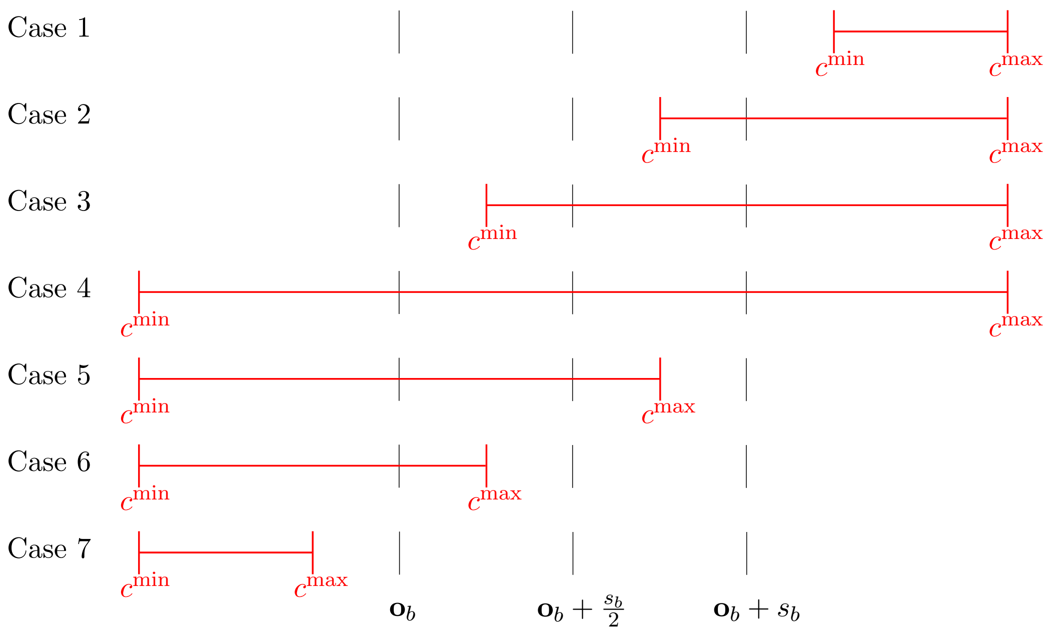


Figure A.2.: 1D example cases of intersections of a cube with a tree branch and sub-branches

Own Publications

[1] Frank Steinbrücker, Christian Kerl, Jürgen Sturm, and Daniel Cremers. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[2] Frank Steinbrücker, Anke Meyer-Bäse, Axel Wismüller, and Thomas Schlossbauer. Application and evaluation of a motion compensation technique to breast mri. In *SPIE Defense, Security, and Sensing*, pages 73470J–73470J. International Society for Optics and Photonics, 2009.

[3] Frank Steinbrücker, Thomas Pock, and Daniel Cremers. Advanced data terms for variational optic flow estimation. In *Proceedings Vision, Modeling and Visualization (VMV)*, Braunschweig, Germany, 2009.

[4] Frank Steinbrücker, Thomas Pock, and Daniel Cremers. Large displacement optical flow computation without warping. In *IEEE International Conference on Computer Vision (ICCV)*, Kyoto, Japan, 2009.

[5] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Real-time visual odometry from dense rgb-d images. In *Workshop on Live Dense Reconstruction with Moving Cameras at ICCV*, 2011.

[6] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Volumetric 3d mapping in real-time on a cpu. In *Proc. International Conference on Robotics and Automation (ICRA)*, Hongkong, China, 2014.

Bibliography

[7] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building rome in a day. In *ICCV*, 2009.

[8] Luis Alvarez, Julio Esclarín, Martin Lefébure, and Javier Sánchez. A PDE model for computing the Optical Flow. In *Proceecings XVI Congreso de Ecuaciones Diferenciales y Aplicaciones*, pages 1349–1356, Las Palmas de Gran Canaria, September 1999.

[9] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 415–421, 1998.

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

[11] Gilles Aubert, Rachid Deriche, and Pierre Kornprobst. Computing optical flow via variational techniques. *SIAM Journal on Applied Mathematics*, 60:156–182, 1999.

[12] Simon Baker, Daniel Scharstein, J.P. Lewis, Stefan Roth, Michael J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *Proc. International Conference on Computer Vision*, 2007.

[13] J.L. Barron, D.J. Fleet, and S.S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[14] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[15] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM Comput. Surv.*, 27(3):433–466, September 1995.

[16] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2:183–202, 2009.

[17] Richard E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

[18] Paul J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239–256, 1992.

[19] Michael J. Black. Robust incremental optical flow. Technical report, 1992.

[20] Michael J. Black and P. Anandan. A framework for the robust estimation of optical flow. In *Proc. International Conference on Computer Vision*, pages 231–236, 1993.

[21] Jean-Daniel Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Trans. Graph.*, 3(4):266–286, October 1984.

[22] Andrés Bruhn, Christoph Schnörr, and Joachim Weickert. Lucas/kanade meets horn/schunck: Combining local and global optic flow methods - updated version with errata. *International Journal of Computer Vision*, 61(3):211–231, 2005.

[23] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part VI*, ECCV'12, pages 611–625. Springer-Verlag, 2012.

[24] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 778–792, 2010.

[25] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20(1-2):89–97, 2004.

[26] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[27] Pierre Charbonnier, Laure Blanc-Fraud, Gilles Aubert, and Michel Barlaud. Two deterministic half-quadratic regularization algorithms for computed imaging. In *ICIP*, pages 168–172, 1994.

[28] Jiawen Chen, Dennis Bautembach, and Shahram Izadi. Scalable real-time volumetric surface reconstruction. In *SIGGRAPH*, 2013.

[29] Isaac Cohen. Nonlinear Variational Method for Optical Flow Computation. In *Proceecings 8th SCIA*, pages 523–530, June 1999.

[30] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering. In *SIGGRAPH*, feb 2009.

[31] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.

[32] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2003.

[33] Andrew J. Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[34] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *Proc. International Conference on Robotics and Automation (ICRA)*, 2012.

[35] Kenth Engø. On the BCH-formula in so(3). Technical report, BIT, 2000.

[36] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. *Comm. Assoc. Comp. Mach.*, 24:381–395, 1981.

[37] Wolfgang Förstner and Eberhard Gülch. A fast operator for detection and precise localization of distinct points, corners and circular features. In *Proc. Intercommission Conf. on Fast Processing of Photogrammetric Data*, pages 281–305, Interlaken, Switzerland, 1987.

[38] Simon Fuhrmann and Michael Goesele. Fusion of Depth Maps with Multiple Scales. *ACM Trans. Graph.*, 30(6):148, 2011.

[39] David Gallup, Jan-Michael Frahm, Philippos Mordohai, and Marc Pollefeys. Variable baseline/resolution stereo. In *Proc. International Conference on Computer Vision and Pattern Recognition*, 2008.

[40] Gottfried Graber, Thomas Pock, and Horst Bischof. Online 3d reconstruction using convex optimization. In *ICCV Workshops*, pages 708–711, 2011.

[41] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proc. of The Fourth Alvey Vision Conference*, pages 147–151, Manchester, 1988.

[42] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *Proc. of the Intl. Symp. on Experimental Robotics (ISER)*, Delhi, India, 2010.

[43] Heiko Hirschmüller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *CVPR*, 2005.

[44] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *A.I.*, 17:185–203, 1981.

[45] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew J. Davison, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 559–568, 2011.

[46] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, 2006.

[47] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, 2013.

[48] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for RGB-D cameras. In *ICRA*, 2013.

[49] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, 2007.

[50] Donald E. Knuth. Two notes on notation. *Am. Math. Monthly*, 99(5):403–422, May 1992.

[51] Kalin Kolev, Thomas Brox, and Daniel Cremers. Propagating photoconsistency in multiview 3d reconstruction. In *Proc. International Conference on Computer Vision and Pattern Recognition*, 2007. submitted.

[52] Kalin Kolev, Maria Klodt, Thomas Brox, Selim Esedoḡlu, and Daniel Cremers. Continuous global optimization in multiview 3d reconstruction. In *Int. Conf. on Energy Minimization Methods for Computer Vision and Pattern Recognition*, 2007. To appear.

[53] Kurt Konolige, Motilal Agrawal, Robert C. Bolles, Cregg Cowan, Martin Fischler, and Brian Gerkey. Outdoor mapping and navigation using stereo vision. In *Intl. Symp. on Experimental Robotics (ISER)*, 2007.

[54] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 21(4):163–169, 1987.

[55] Steven Lovegrove and Andrew J. Davison. Real-time spherical mosaicing using whole image alignment. In *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*, ECCV'10, pages 73–86, 2010.

[56] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[57] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proc.7th International Joint Conference on Artificial Intelligence*, pages 674–679, Vancouver, 1981.

[58] Hans P. Moravec. Robot spatial perception by stereoscopic vision and 3d evidence grids. Technical report, 1996.

[59] Richard M. Murray, S. Shankar Sastry, and Li Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994.

[60] Hans-Hellmut Nagel. On the estimation of optical flow: Relations between different approaches and some new results. *Artif. Intell.*, 33:299–324, 1987.

[61] Hans-Hellmut Nagel and Wilfried Enkelmann. An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(5):565–593, 1986.

[62] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*, 2011.

[63] Richard A. Newcombe, Steven Lovegrove, and Andrew J. Davison. DTAM: dense tracking and mapping real-time. In *Proc. of the Intl. Conf. on Computer Vision (ICCV)*, Barcelona, Spain, 2011.

[64] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6):169:1–169:11, November 2013.

[65] David Nister, Oleg Naroditsky, and James Bergen. Visual odometry. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[66] Martin R. Oswald and Daniel Cremers. A convex relaxation approach to space time multi-view 3d reconstruction. In *ICCV Workshop on Dynamic Shape Capture and Analysis (4DMOD)*, 2013.

[67] Martin R. Oswald, Eno Toeppe, Kalin Kolev, and Daniel Cremers. Non-parametric single view reconstruction of curved objects using convex optimization. In *Pattern Recognition (Proc. DAGM)*, 2009.

[68] Nils Papenberg, Andrés Bruhn, Thomas Brox, Stephan Didas, and Joachim Weickert. Highly accurate optic flow computation with theoretically justified warping. *International Journal of Computer Vision*, 67(2):141–158, April 2006.

[69] Thomas Pock, Daniel Cremers, Horst Bischof, and Antonin Chambolle. Global solutions of variational models with convex regularization. *SIAM Journal on Imaging Sciences*, 3:1122–1145, 2010.

[70] Thomas Pock, Thomas Schoenemann, Gottfried Graber, Horst Bischof, and Daniel Cremers. A convex formulation of continuous multi-label problems. In *European Conference on Computer Vision (ECCV)*, Marseille, France, October 2008.

[71] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ECCV'06, pages 430–443, 2006.

[72] Y. Roth-Tabak and Ramesh Jain. Building an environment model using depth information. *Computer*, 22(6):85–90, June 1989.

[73] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, 2011.

[74] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[75] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Proc. of the Intl. Conf. on 3-D Digital Imaging and Modeling*, Quebec, Canada, 2001.

[76] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, April 2002.

[77] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-ICP. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, 2009.

[78] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR*, 2006.

[79] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proc. International Conference on Computer Vision and Pattern Recognition*, pages 592–600, 1994.

[80] Leonid Sigal, Alexandru O. Balan, and Michael J. Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *Int. J. Comput. Vision*, 87(1-2):4–27, March 2010.

[81] Hauke Strasdat, J.M.M. Montiel, and Andrew J. Davison. Scale drift-aware large scale monocular slam. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, 2010.

[82] Jörg Stückler and Sven Behnke. Integrating depth and color cues for dense multi-resolution scene mapping using rgb-d cameras. In *MFI*, 2012.

[83] Jürgen Sturm, Stéphane Magnenat, Nikolas Engelhard, François Pomerleau, Francis Colas, Wolfram Burgard, Daniel Cremers, and Roland Siegwart. Towards a benchmark for RGB-D SLAM evaluation. In *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS)*, Los Angeles, USA, 2011.

[84] Eno Toeppe, Martin R. Oswald, Daniel Cremers, and Carsten Rother. Image-based 3d modeling via Cheeger sets. In *Asian Conf. on Computer Vision*, 2010.

[85] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, Pittsburgh, PA, 1991.

[86] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380, 1991.

[87] Olga Veksler. *Efficient Graph-based Energy Minimization Methods in Computer Vision.* PhD thesis, Cornell University, July 1999.

[88] George Vogiatzis, Philip H.S. Torr, and Roberto Cippola. Multi-view stereo via volumetric graph-cuts. In *Proc. International Conference on Computer Vision and Pattern Recognition*, pages 391–399, 2005.

[89] Andreas Wedel, Clemens Rabe, Tobi Vaudrey, Thomas Brox, Uwe Franke, and Daniel Cremers. Efficient dense scene flow from sparse or dense stereo data. In *European Conference on Computer Vision (ECCV)*, Marseille, France, October 2008.

[90] Thomas Whelan, Hordur Johannsson, Michael Kaess, John J. Leonard, and John B. McDonald. Robust tracking for real-time dense RGB-D mapping with Kintinuous. Technical report, MIT, 2012.

[91] Thomas Whelan, Hordur Johannsson, Michael Kaess, John J. Leonard, and John B. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *ICRA*, Karlsruhe, Germany, 2013.

[92] Thomas Whelan, Michael Kaess, Maurice F. Fallon, Hordur Johannsson, John J. Leonard, and John B. McDonald. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[93] Kai M. Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation at ICRA*, 2010.

[94] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime TV-L1 optical flow. In *Pattern Recognition (Proc. DAGM)*, LNCS, pages 214–223. Springer, 2007.

[95] Christopher Zach, Thomas Pock, and Horst Bischof. A globally optimal algorithm for robust TV-L1 range image integration. In *ICCV*, 2007.

[96] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. A Memory-Efficient KinectFusion using Octree. In *Computational Visual Media*, volume 7633 of *Lecture Notes in Computer Science*, pages 234–241. Springer Berlin Heidelberg, 2012.