Stereo Correspondence for Slanted Surfaces: Local and Global Methods

Bachelor's Thesis

Jean Marc Roth

Supervisor:
Prof. Dr. Joachim Weickert

Advisors:
Dr.-Ing. Andrés Bruhn
Natalia Slesareva, M.Sc.

Reviewers:
Prof. Dr. Joachim Weickert
(Saarland University, Saarbrücken, Germany)
Dr.-Ing. Bodo Rosenhahn
(Max-Planck-Institute for Computer Science, Saarbrücken, Germany)

Saarland University
Faculty of Natural Sciences and Technology I
Departments of Mathematics
and Computer Science
Mathematical Image Analysis Group

Saarbrücken, March 2008

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, March 4, 2008

J. M. Roth

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, March 4, 2008

J. M. Roth

Abstract

The usual stereo correspondence problem compares regions of images that are supposed to be frontal-parallel planes, which introduces systematic errors. In this thesis, we will investigate special stereo correspondence particularly able to correctly detect slanted surfaces by explicitly modeling first order geometry. This goes toward a more general and smart way of modeling what the real world actually consists of.

First, we show a local method in which we build in support for first order geometry by means of disparity derivatives, which help to account for perspective distortion of first order surfaces encountered from different views. This results in the use of a deformed window for the local region matching method and also the possibility to reconstruct the scene using surfaces described by their normal vectors. We then show how these normal vectors can be used to build a smoothness assumption that can invalidate what probably are bad matches using an iterative procedure and provide some sort of filling-in.

Afterwards, in order to contrast the local method, we develop a global method, into which we build a similar constraint via smoothness assumptions. We first model the problem the classical way using an energy functional containing data and smoothness terms. Then, we enhance the classic Horn/Schunck method to account for first order surfaces by adding a higher-order smoothness assumption. After that, we discretize this energy functional and minimize it using differential calculus. We show complete stencils and boundary conditions.

Finally, a comparison of the local and global methods with several parameters is performed using a synthetic and a real-world scene, where one contains many and the other one few slanted surfaces. We give both quantitative and qualitative evaluations of the results by means of objective error measures and subjective visual inspection, respectively.

Acknowledgments

I would like to thank Natalia Slesareva and Andrés Bruhn for taking their time to answer and discuss all my questions in-depth. I would also like to thank Prof. Weickert for the freedom he left me starting this thesis at my own pace.

Thanks go out to Andrés Bruhn for providing a framework and Stephan Didas for providing hints for Chapter 4.

Contents

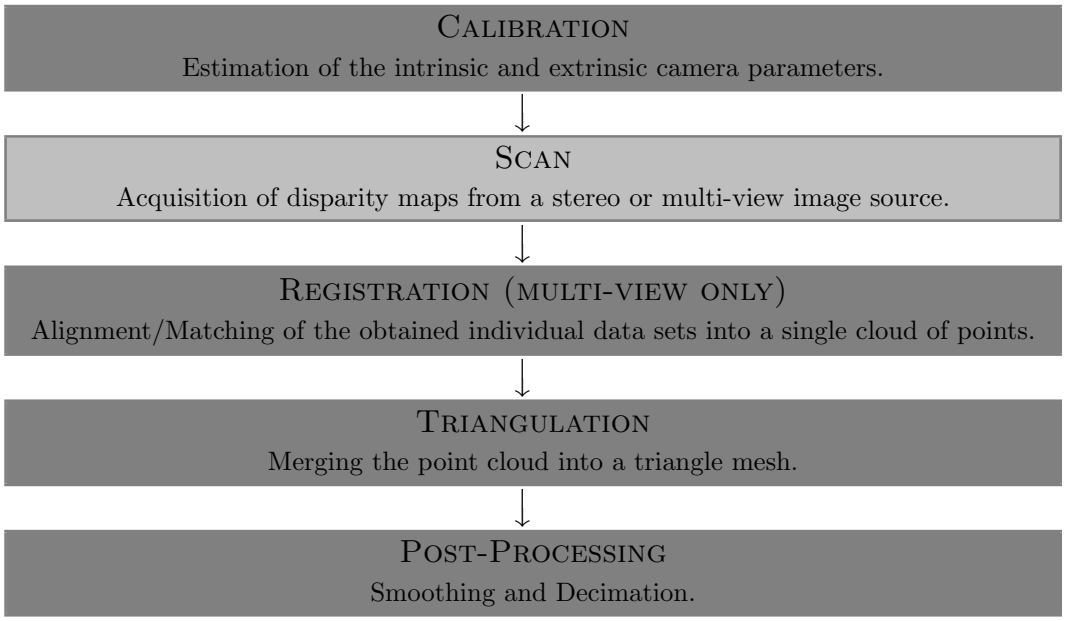
Chapter 1

Introduction

1.1 Motivation

The reconstruction of a three-dimensional object from two-dimensional images has always been a daunting task for machines, since they lack the a priori knowledge indigenous to human beings. As such, the ultimate goal of computer vision is to extract information about a 3-D world from 2-D images. There even exists a method that uses only one (!) 2-D image [HEH05].

The standard way of constructing a 3-D model from 2-D imagery consists of the following steps, which are called the 3-D scanning pipeline [Goe03].



This work will focus on step two: the acquisition of high quality disparity maps.

1.2 Existing Methods

Most computer vision problems are matching problems, and stereo correspondence makes no exception. There exist a wide range of procedures, from very simple local over global methods, to any combination or sequence of local and global ones, so-called cooperative methods [SS02].

Generally, stereo algorithms perform subsets of the following tasks: matching cost computation and aggregation, disparity computation, optimization and refinement.

The matching cost computation is done by methods like sum of squared differences (SSD) [Ana87] or normalized cross-correlation (NXC); other well-known methods are squared intensity differences and absolute intensity differences [BBH93, Han74]. There also exist binary match costs, using features like edges [Can86] or the sign of the Laplacian; they are however not very well applicable to dense stereo methods [Nis87].[1] These methods can be extended to SSSD (sum of sum of squared differences) and SSAD (sum of sum of absolute differences) in a multi-view setting and can be generalized to arbitrary camera configurations (converging cameras) [OK91, Col96].

Local and window-based methods perform the matching cost aggregation by summing or averaging over a support region in the disparity space image (x, y, d), where (x, y) denotes the position in the image (usually the left image is the reference) and $d(x, y)$ the disparity obtained at that pixel. This support region can be two-dimensional at a fixed disparity, or three-dimensional in disparity space. The first favors frontal parallel surfaces (surfaces that are parallel to the camera's image plane) and uses square windows, Gaussian convolution, shiftable windows, windows with adaptive size [KO94], etc. [Vek01]. The second (3-D support region) supports slanted [BT99, LZ05] and curved surfaces [LZ06]. An alternative way of aggregating is diffusion, i.e. repeatedly adding to each pixel's cost the weighted values of its neighboring pixels' costs [SS96].

Disparity refinement allows for example to extend disparity information from a small quantized to a more continuous-appearing set of values (sub-pixel refinement), for example by interpolating between two integer values. Depending on the method used, edges might be blurred, if two adjacent pixels belong to two different surfaces. Furthermore, "holes" in the information can be filled in using such interpolation techniques. A median filter might help removing obvious outliers of small size.

As far as disparity computation is concerned, stereo algorithms can be broken down into three major categories, each of which we will now visit in more detail.

[1] *dense stereo method*: a stereo matching method that produces valid output for every input pixel (there is no "don't know")

Local methods focus on the matching cost computation and aggregation step and many times include smart disparity refinement techniques, which make them however less transparent and more difficult to prove whether they are mathematically sound.

For each pixel, the disparity with the highest or lowest cost value (depending on the method) is chosen. Drawbacks are that all other matches (even if they are only slightly from the maximum/minimum) are thrown away. Also, each pixel in the reference image is matched to one pixel in the other image, but the inverse is not enforced, which can lead to occlusions[2] generating matching problems. There exist plain local and local differential methods, like the Lucas/Kanade method [LK81].

Global optimization methods (variational methods) mainly consist of the disparity computation and optimization step. They are formulated in an energy minimization framework:

$$E(d) = \int D(d) + \sum_i \alpha_i S_i(d)\, dx\, dy\,. \tag{1.1}$$

E is a so-called functional (a function taking function(s) as argument); it contains a data term D which measures how well the disparity function $d(x, y)$ agrees with the input image pair, as well as smoothness terms S_i which encode the smoothness assumptions made by the individual algorithm, i.e. trying not to deviate "too much" from the neighborhood. It does this in order to help solve the aperture problem, which arises if we only have information about parts of a scene (through an imaginary aperture) and therefore cannot always tell the actual direction in which things are moving, since many directions are possible considering the little information we have.[3] The regularization parameters α_i state how important the terms are with regard to each other.

The Horn/Schunck method is one example of such a method [HS81]. Using more elaborate smoothness terms can help preserving edges, this is called discontinuity preservation. There also exist Bayesian interpretations of this problem as well as Markov networks. The smoothness term can also be made dependent, not only on the disparity, but on the intensity difference to neighboring pixels, or both. The main distinction is in the minimization method used: graph cuts [BG07, BVZ01], diffusion [SS96], belief propagation, genetic algorithms, simulated annealing, etc. [Dev74]. A different class of global optimization algorithms are those based on dynamic programming.

[2] *occlusion*: something which was visible in one image, but is no longer visible in another image, because of its new angle to the camera, or because of the line-of-sight being blocked by another object closer to the camera

[3] We rarely have knowledge of the entire "world".

Global methods always have the advantage that one does not have to deal with outliers as in local methods. Additionally, global methods give a filling-in effect for small data terms, which is when the smoothness term takes over and propagates information from areas where more information is available. This results in dense flow fields without performing subsequent interpolation steps (no subpixel refinement).

Cooperative algorithms are inspired by the human stereo system itself [SS02]. These algorithms perform (iteratively) local computations, however they use nonlinear operations that, in the end, perform similar to global optimization. Some operate in a segmented image, others use a scale-space approach, i.e. find results on a coarse scale and then refine these to a finer scale, in order to correctly determine the global minimum of the non-convex functional [BAHH92].

In general, occlusions pose problems, as do texture-less regions and repeating patterns. In practice, it can be very useful to be able to obtain the information where there is no information available (no solution, or worse, an infinite number of solutions), however this is mostly a feature of global methods.

Finally, there are a variety of real-world and synthetic sequences to test stereo algorithms on. There does not yet seem to exist an artificial intelligence capable to cope with all kinds of situations, and our overview is all but exhaustive. One may very well conclude that there is always one certain combination of methods that performs best for one specific application and that the ultimate goal of computer vision is to unite those methods.

1.3 Problem Formulation

A general problem with many traditional stereo algorithms is the explicit or implicit use of the frontal parallel plane assumption, i.e. the assumption that all surfaces in the scene are parallel to the image plane of the camera. This is however not the general case — consider for example the case of slanted or curved surfaces, in which this assumption no longer holds, see Figure 1.1.

Therefore, in the local method, disparity derivative information is used, which essentially indicates where the search window has to be deformed from the reference image to the other image [LZ05].

As far as the global method that we are proposing is concerned, we will model the slanted surfaces using higher-order regularization, thus only penalizing surfaces which are strictly of higher order than first order.

While adjusting our implementations for the deficiency of implicitly only considering frontal-parallel planes, the acquisition of the initial disparity map via simple area-based matching, like sum of squared differences or cross correlation, is very error-prone in itself. This is probably due to its simplicity:

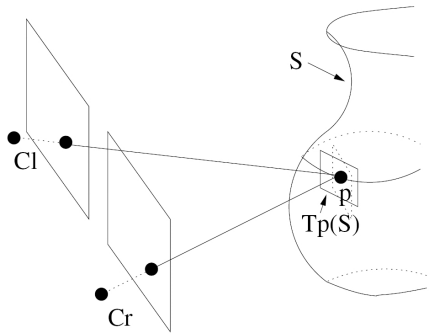


FIGURE 1.1: An example how the frontal parallel plane assumption usually is not fulfilled. Note the tangent plane $\mathcal{T}_{\mathbf{p}}(S)$ at point $\mathbf{p}(x, y, z)$, which is not parallel to either image plane. (The corresponding parallel plane at point $\mathbf{p}$ is shown with dotted lines.)

it only considers intensity values, which might match regions (assumed projections of the same scene point into both images) that are in fact not related at all, except sometimes by chance.

By letting a human observer give a rough estimate of the maximum disparity of the scene, one can already restrict the search space a lot, giving no rise to the most impossible outliers. As a further measure, we will (in the local method) not only remember one match per pixel for ongoing computations, but several ones. Evaluating the geometry of the local neighborhood is then going to tell us which is the correct one, by using an iterative procedure to fill-in the information from trustworthy neighboring pixels.

The global method already comes with the filling-in feature with no additional effort. Its smoothness term provides nothing else than diffusion from neighboring areas in areas where the data term is at a loss providing a correspondence.

1.4 Overview

In Chapter 2, we will introduce our syntax and remind the reader of some basic operations and geometry that is used throughout this work. In Chapter 3, we will introduce already existing algorithms that we have used as a foundation for our implementation of the local method. Appendix A shows an important modification that we have made to one of those algorithms. In Chapter 4, we will present our novel variational approach where we build in similar constraints. Chapter 5 will then give qualitative measurements and compare the local and global method systematically. Finally, Chapter 6 will summarize our work and give an outlook of possible related work in the future.

Chapter 2

Preliminaries

2.1 Basic Operations and Algorithms

A discrete grayscale image is defined as a mapping from a (usually rectangular and two-dimensional) domain $\Omega_2 = \{0, 1, \ldots, n_1\} \times \{0, 1, \ldots, n_2\} \ni (x, y)$, sometimes just noted Ω, to a co-domain $N \subset \mathbb{N}$:

$$\begin{array}{rcl} I : \mathbb{N}^2 \supset \Omega_2 & \rightarrow & N \subset \mathbb{N} \\ (x, y) & \mapsto & I(x, y) = I_{xy} \in \{0, 1, \ldots, 2^n\} \, . \end{array} \tag{2.1}$$

In our case, the co-domain specifies a gray value, which usually is an element of $\{0, 1, \ldots, 2^n\}$, $n, n_1, n_2 \in \mathbb{N}$. In the field of stereo vision, we often use a disparity space image, i.e. the co-domain encodes the disparity (displacement) of a certain pixel as gray value, i.e. $I_{xy} = cd$ where c denotes a scaling constant and d the disparity at (x, y).

2.1.1 Derivatives, Approximations and Notation

Derivatives and Vector-Valued Functions

In the continuous setting, we will denote derivatives of a 1-D function $f(x)$ as $\frac{df}{dx}(x)$ or $f'(x)$, depending on the brevity required. In the higher-order case we write: $f^{(n)}(x)$ or $\frac{d^n}{dx^n} f(x)$ as the n^{th} order derivative of f at the point x.

A normal character with a subscript like h_x does however not indicate the derivative of a function but only a fixed scalar value of some sort (e.g. element of a vector value) — its meaning will be available from the context.

In a multi-dimensional setting we will use partial derivatives like $\frac{\partial}{\partial x_i} f = \partial_{x_i} f$. In the higher-order multi-dimensional case, the notation used is $\frac{\partial^n f}{\partial x_i^n} = \partial_{x_i}^n [f]$, or for example $\frac{\partial^{n_1+n_2} f}{\partial x_i^{n_1} \partial x_j^{n_2}} = \partial_{x_i}^{n_1} \partial_{x_j}^{n_2} [f]$ for mixed derivatives, with $f = f(x_1, \ldots, x_n)$, $i, j \in \{1, \ldots, n\}$, $n, n_1, n_2 \in \mathbb{N}$.

The set of all first order derivatives (the gradient) is usually compactly expressed as $\nabla_n f = \left[\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n} \right]^{\mathrm{T}}$ and the matrix containing all second order

derivatives, the Hessian, consequently as $\nabla^2 f$ or $\mathcal{H}(f)$.

$$\mathbf{x_R}^{\mathrm{T}} = \begin{bmatrix} x_1 \cdots x_n \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \mathbf{x_C} \text{ and } \mathbf{x_C}^{\mathrm{T}} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} x_1 \cdots x_n \end{bmatrix} = \mathbf{x_R}$$

denote the transpose of the row and column n-vector $\mathbf{x_R}$ and $\mathbf{x_C}$, respectively.

We write vectors in bold lowercase letters ($\mathbf{v}$), matrices or vector fields in bold uppercase letters ($\mathbf{N}$), and important matrices that usually have their own name in calligraphic letters ($\mathcal{H}$).

Discrete versions of these derivatives (finite difference approximations) will be introduced as they are needed.

Taylor Series

The Taylor series is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point.

Definition 2.1.1 *The Taylor series of a 1-D function f which is infinitely many times differentiable in a neighborhood of a real (or complex) number a, is the power series*

$$T(x) = f(a) + \frac{f'(a)}{1!}(x-a)^1 + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots , \quad (2.2a)$$

which in a more compact form can be written

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n , \quad (2.2b)$$

where $n!$ is the factorial of n, $f^{(0)} = f$, $(x-a)^0 = 1$ and $0! = 1$.

Its general formula in a multi-dimensional setting is:

$$\begin{aligned} T(x_1, \cdots, x_d) &= \sum_{n_1=0}^{\infty} \cdots \sum_{n_d=0}^{\infty} \frac{\partial^{n_1}}{\partial x_1^{n_1}} \cdots \frac{\partial^{n_d}}{\partial x_d^{n_d}} \frac{f(a_1, \cdots, a_d)}{n_1! \cdots n_d!} (x_1 - a_1)^{n_1} \cdots (x_d - a_d)^{n_d} \\ &= f(\mathbf{a}) + \nabla f(\mathbf{a})^T (\mathbf{x} - \mathbf{a}) + \frac{1}{2}(\mathbf{x} - \mathbf{a})^T \nabla^2 f(\mathbf{a})(\mathbf{x} - \mathbf{a}) + \cdots . \end{aligned} \quad (2.3)$$

Taylor Series expansion up to first order is called linearization.

Vector Fields

Definition 2.1.2 *A vector field* $\mathbf{V}$ *on an open set* $O \subset \mathbb{R}^n$ *is a function which associates a vector* $\mathbf{V}(\mathbf{x})$ *to each point* $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ *of* O*:*

$$\begin{aligned} \mathbf{V} : O \subset \mathbb{R}^n &\to \mathbb{R}^n \\ \mathbf{V}(\mathbf{x}) &\mapsto \begin{bmatrix} V_1(\mathbf{x}) \\ V_2(\mathbf{x}) \\ \vdots \\ V_n(\mathbf{x}) \end{bmatrix} = \mathbf{v} \,, \end{aligned} \tag{2.4}$$

with $V_i : O \to \mathbb{R}$ *and* $i \in \{1, \ldots, n\}$*.*

In a practical example, vector $\mathbf{x}$ can be seen as the location in Euclidean space where one wants to "extract" the corresponding result vector $\mathbf{v}$ from. V_i then extracts the ith coordinate of $\mathbf{v}$ at that location.

2.1.2 Numerical Solvers

Solving linear systems of equations fast and efficiently is a field of its own. In this subsection we give an overview of the most important numerical solvers and we then describe the one we use in a little more detail [Bru07]. We however omit proofs of convergence etc., as they are beyond the scope of this thesis. We consider stationary iterative methods only [Saa03].

Formulation as Fixed Point Iteration

A system of equations (or a single equation) can often be recast as a fixed-point problem for a related function. Usually, iterative schemes of the form $x_{k+1} = g(x_k)$ are used, where g is a suitably chosen function whose fixed points are the solutions for $f(x) = 0$, starting with an initial guess x_0 [Hea01].

Definition 2.1.3 *Given a function* $g(x)$*, a value* x *such that*

$$x = g(x) \tag{2.5}$$

is called a fixed point of the function g*, since* x *is unchanged when* g *is applied to it.*

Geometrically, the fixed points of a function are the point(s) of intersection of the curve of the function and the line $y = x$. If $g(x)$ is a continuous function, $\{x_n\}_{n=0}^{\infty}$ is a sequence generated by fixed point iteration and $\lim_{n\to\infty} x_n = x$, then the fixed point iteration is said to converge, otherwise it diverges.

The question now is: how can we solve the linear system of equations $\mathbf{Ax} = \mathbf{b}$? We use matrix notation to compactly describe the linear system

of equations. Let us quickly show an example of a system of equations with two equations and two unknowns:

$$\underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}}_{\mathbf{b}} . \tag{2.6a}$$

This is equivalent to the following system of equations:

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 &= b_1 , \\ A_{21}x_1 + A_{22}x_2 &= b_2 . \end{aligned} \tag{2.6b}$$

To solve such a system of equations, the idea is to find a "cheap" but accurate approximation of $\mathbf{A}^{-1}$, i.e. the inverse of the matrix $\mathbf{A}$, via the decomposition $\mathbf{A} = \mathbf{A_1} + \mathbf{A_2}$ and then introduce the fixed point iteration

$$\begin{aligned} & (\mathbf{A_1} + \mathbf{A_2})\,\mathbf{x} = \mathbf{b} \\ \Leftrightarrow\ & \mathbf{A_1}\mathbf{x} + \mathbf{A_2}\mathbf{x} = \mathbf{b} \\ \overset{\text{FP}}{\Leftrightarrow}\ & \mathbf{A_1}\mathbf{x}^{k+1} = \mathbf{b} - \mathbf{A_2}\mathbf{x}^{k} \\ \Leftrightarrow\ & \mathbf{x}^{k+1} = {\mathbf{A_1}}^{-1}\left(\mathbf{b} - \mathbf{A_2}\mathbf{x}^{k}\right) . \end{aligned} \tag{2.7}$$

k indicates the respective step in the iteration. Such a method is called stationary because $\mathbf{A_1^{-1}}$, $\mathbf{A_2}$ and $\mathbf{b}$ do not change over the iterations. Formally, this is a fixed point iteration with iteration function $\mathbf{g}(\mathbf{x}) = {\mathbf{A_1}}^{-1}\left(\mathbf{b} - \mathbf{A_2}\mathbf{x}^{k}\right)$.

A frequent approach is to do the decomposition like

$$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U} , \tag{2.8}$$

where $\mathbf{D}$ is the diagonal part, $\mathbf{L}$ the strictly lower triangular part and $\mathbf{U}$ the strictly upper triangular part of the matrix $\mathbf{A}$, respectively.

The Jacobi Method

In the Jacobi method one sets $\mathbf{A_1} = \mathbf{D}$ (diagonal matrices are simple to invert), and $\mathbf{A_2} = -\mathbf{L} - \mathbf{U}$. This results in

$$\mathbf{x}^{k+1} = \mathbf{D}^{-1}\left(\mathbf{b} + (\mathbf{L} + \mathbf{U})\mathbf{x}^{k}\right) . \tag{2.9a}$$

Often, one uses the element-based notation. This gives in the i^{th} equation (see (2.6a)):

$$\begin{aligned} & \sum_{j=1}^{n} A_{ij}x_j = b_i \\ \Leftrightarrow\ & x_i^{k+1} = \frac{1}{A_{ii}}\left(b_i - \sum_{j \neq i} A_{ij}x_j^{k}\right) , \end{aligned} \tag{2.9b}$$

where n is the number of unknowns and $i = 1, 2, \cdots, n$. We have solved for the value of x_i while the other entries of $\mathbf{x}$ remained fixed.

The Gauss-Seidel Method

The Gauss-Seidel method is an improved variant of the Jacobi method. In the Gauss-Seidel method one sets $\mathbf{A_1} = \mathbf{D} - \mathbf{L}$ (this triangular matrix is a better approximation to $\mathbf{A}$ than $\mathbf{D}$ alone), and $\mathbf{A_2} = -\mathbf{U}$. This results in

$$\mathbf{x}_{\mathrm{GS}}^{k+1} = (\mathbf{D} - \mathbf{L})^{-1} \left(\mathbf{b} + \mathbf{U}\mathbf{x}_{\mathrm{GS}}^{k}\right) . \tag{2.10a}$$

The element-based notation gives here:

$$x_{\mathrm{GS},i}^{k+1} = \frac{1}{A_{ii}} \left(b_i - \sum_{j<i} A_{ij} x_{\mathrm{GS},j}^{k+1} - \sum_{j>i} A_{ij} x_{\mathrm{GS},j}^{k} \right) . \tag{2.10b}$$

As one can see, this method has the advantage that it uses previously obtained results as soon as they are available. As a consequence, one also needs less storage space as the array containing $\mathbf{x}$ can be overwritten and does not need explicit copying as in the Jacobi method.

The SOR Method

The Successive Over-Relaxation Method (SOR) is an extension of the Gauss-Seidel method [You50]. The idea behind it is to extrapolate the result of the Gauss-Seidel method, which in the element-based notation reads

$$x_i^{k+1} = (1-\omega) x_i^k + \omega \underbrace{\frac{1}{A_{ii}} \left(b_i - \sum_{j<i} A_{ij} x_j^{k+1} - \sum_{j>i} A_{ij} x_j^k \right)}_{x_{\mathrm{GS},i}^{k+1}} , \tag{2.11}$$

with the overrelaxation parameter $\omega \in [0, 2[$ guaranteeing convergence. The idea is to choose ω such that it will accelerate the convergence towards the solution. For $\omega = 1$ it is exactly the Gauss-Seidel method.

Usually, the SOR method is two to three orders of magnitude faster than the Gauss-Seidel method, therefore, for our needs, it will suffice, as it provides a good compromise between implementation complexity and performance.

2.2 Camera and Epipolar Geometry

At the heart of stereo vision are the cameras and their positioning with respect to each other and the geometry describing both the cameras' properties as well as the relation between the pictures they take.

In the search for a depth (equally disparity) map, we are interested in finding conjugated points, i.e. two points in different images that result from the same 3-D scene point. The problem here lies in the estimation of the disparity, i.e. the distance between two conjugated points. The main idea is

that objects closer to the camera move faster than objects further away from the camera, when the camera is moved.[1]

When two or more cameras come into play, we talk about epipolar geometry, otherwise about monocular geometry.

Even though we assume the simplest of camera setups in this work and we might well do without this section, we nevertheless feel that at least a quick glimpse into this matter is necessary for the sake of completeness.

2.2.1 Pinhole Camera and Perspective Projections

The process of forming a 2-D image from a 3-D world is called projection. The process of projecting can be performed in a lot of ways like perspective, orthographic, oblique, anamorphic, etc..

To keep the theoretical approach simple, the so-called pinhole camera model is usually used. Such a camera contains no lens and only an infinitesimally small aperture through which all the incoming rays of light pass. This is a fairly realistic model for a camera system, and it can be described by a perspective projection. A perspective projection is also called central projection or pinhole model. The "pinhole" is also known as focal center or center of projection in literature [TV98].

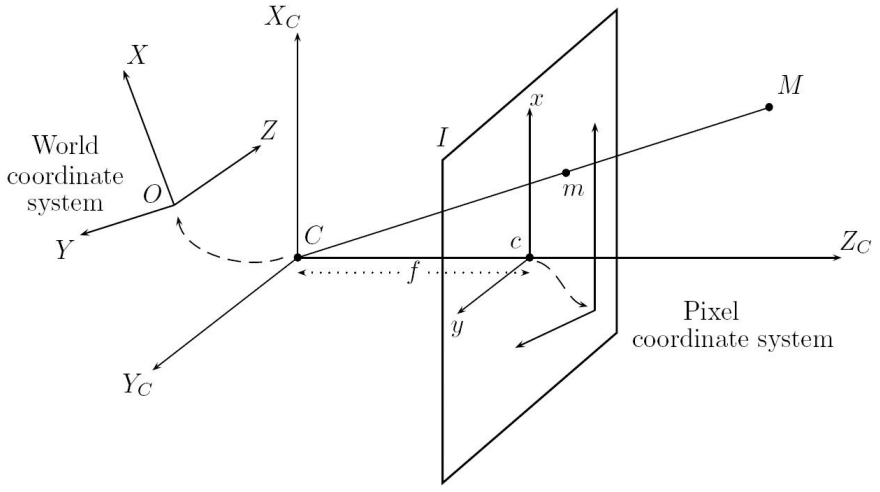


FIGURE 2.1: Complete pinhole camera model with world, camera, image and pixel coordinate systems. (Taken from [Bru07])

There exist several coordinate systems, the link between which is made by the camera parameters, which consist of the so-called intrinsic and extrinsic

[1] Obviously, it doesn't matter if we consider one camera that is moved to a new location while time is "frozen", in order to take a second picture, or if we just consider two cameras taking pictures at the same time.

parameters and a projection matrix, which we will however not elaborate upon. All four coordinate system together are depicted as an overview in Figure 2.1.

A perspective projection maps a point $M = (X_C, Y_C, Z_C)^{\mathrm{T}}$ from the 3-D camera coordinate system to a point $m = (x, y)^{\mathrm{T}}$ in the 2-D image coordinate system. In the case of monocular vision, i.e. in the presence of one camera only, projective geometry is used to describe the scenario.

2.2.2 Multiple Cameras and Epipolar Geometry

When two cameras (or more) view a 3-D scene, there are certain rules that dictate the relationship between these views. These rules are called epipolar geometry [Wik08].

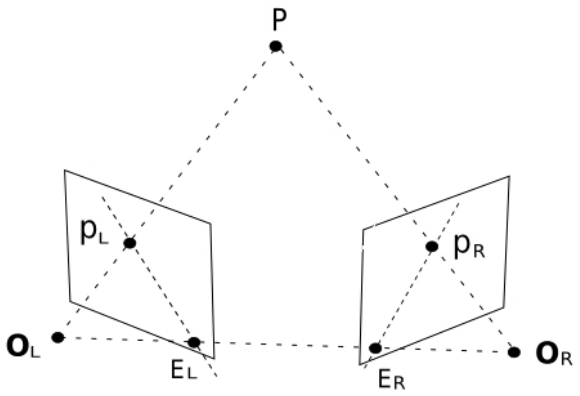


FIGURE 2.2: An example of epipolar geometry with two cameras. (Taken from [Wik08])

Figure 2.2 contains an example of epipolar geometry. In it we see:

▷ P**:** a point $(x, y, z)^{\mathrm{T}}$ in $\mathbb{R}^3$.

▷ O_L, O_R**:** the left and right optical center, respectively.

▷ p_L, p_R**:** the (perspective) projections of P onto the image planes of left and right cameras, respectively.

▷ E_L**:** the epipole of the left camera: the projection of the optical center of the right camera onto the image plane of the left camera.

▷ E_R**:** the epipole of the right camera: the projection of the optical center of the left camera onto the image plane of the right camera.

The plane that goes through the point of interest P and the optical centers of both cameras is called epipolar plane. The line where an epipolar plane

intersects the image plane of a camera is called epipolar line. The epipolar line in the left camera ($E_L - p_L$) is the projection of the optical ray of the right camera ($O_R - P$) onto the image plane of the left camera, and vice-versa. Only one epipolar line goes through any image point, with the exception of the epipole, through which all of them pass. The epipole need not lie inside the visible image plane of a camera; indeed it does not lie inside the visible image if the cameras do not "see" each other.[2]

Epipolar Constraint, Orthoparallel and Converging Cameras

For the stereo correspondence problem, using the so-called epipolar constraint reduces the search space from 2-D (the entire second image) to 1-D (only a line in the second image). The conjugated points in two stereo images are located on the epipolar lines.

The simplest but in reality most unusual case is the one of orthoparallel cameras, i.e. two identical cameras on the same height with parallel optical axes. More generally, cameras can be found in arbitrary positions and orientations, called converging cameras.

After finding the disparity/displacement d of two objects in a stereo pair of images, depth z in case of an orthoparallel setup can easily be related to it by

$$z(x, y) = \frac{fb}{d(x, y)}\,, \tag{2.12}$$

where f is the focal length of the lens and b the stereo baseline (distance between the two cameras). This equation results from the properties of perspective projections and especially the Intercept Theorem.

The process of creating a stereo pair with horizontal epipolar lines (like if there had been an orthoparallel camera pair) out of a converging setup is called rectification (Figure 2.3).

In this work, we always assume a rectified stereo pair, therefore we need not give details on the converging setup. It is however worth saying that in converging camera setups the so-called fundamental matrix comes into play — it gives the relation between a point in the one image and the (epipolar) line containing its correspondence in the other image. With orthoparallel cameras, epipolar lines in both images are at the same height and horizontal.

[2] do not have each other in their respective FOV (field of view)

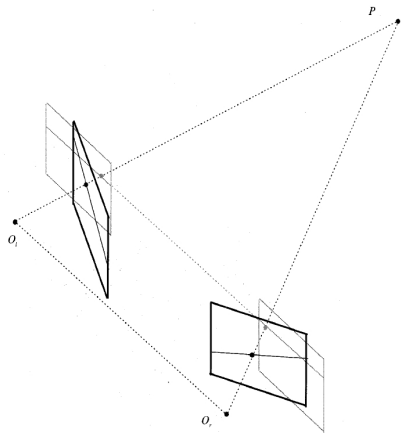


FIGURE 2.3: A rectified stereo pair with converging cameras' image planes are depicted in bold. Rectified images are shown with gray lines. (Taken from [TV98])

Chapter 3

Local Method

3.1 Region-Based Matching Metrics

Block matching methods compare regions in one image to regions in another image in order to find similarities. In the following, let us assume a square window of border size $2m + 1$. The distance from the center pixel to one of the boundaries will therefore be m.

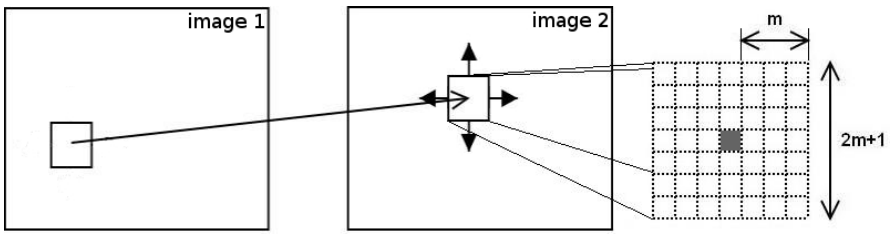


FIGURE 3.1: A block matching method and its window ($m = 3$).

3.1.1 Sum of Squared Differences

The sum of squared differences (SSD) method consists of the quadratic cost function $\mathrm{SSD}(x, y)$

$$\arg\min_{d_x,d_y} \sum_{(\Delta x,\Delta y)\in\mathcal{N}_{xy}} \Big(I_-(x+\Delta x, y+\Delta y) - I_+\big((x+d_x)+\Delta x, (y+d_y)+\Delta y\big)\Big)^2, \tag{3.1}$$

where I_- is the one (usually the left), I_+ the other (usually the right) image and $\mathcal{N}_{xy}$ is the neighborhood around the position (x, y), for example a square of size $(2m+1)\times(2m+1)$ pixels (see Figure 3.1). Δx and Δy are small step sizes in x and y directions respectively, usually $\Delta x, \Delta y \in \{\cdots, -1, 0, 1, \cdots\}$. d_x and d_y are the disparities in x and y direction, respectively (not their

derivatives). In practice, one also limits the search space of these disparities by a maximum disparity.

3.1.2 Cross Correlation

Simple cross correlation (XC) is defined as the maximization of the function $\mathrm{XC}(x, y)$

$$\arg\max_{d_x,d_y} \sum_{(\Delta x,\Delta y)\in\mathcal{N}_{xy}} \Big(I_-\left(x+\Delta x, y+\Delta y\right) \cdot I_+\left((x+d_x)+\Delta x, (y+d_y)+\Delta y\right)\Big). \tag{3.2}$$

It is derived from the idea to use only the mixed term from SSD

$$\sum (I_- - I_+)^2 = \sum I_-^2 - 2\boxed{\sum I_- I_+} + \sum I_+^2 \tag{3.3}$$

for a better measure of similarity, as the other two terms are more or less constant.

With normalized cross correlation (NXC) in mind, let us first write down the mean value of an image window:

$$\overline{I(x,y)} = \frac{1}{(2m+1)^2} \sum_{(\Delta x,\Delta y)\in\mathcal{N}_{xy}} I(x+\Delta x, y+\Delta y)\,. \tag{3.4a}$$

The deviation from this mean of a point $(x+\Delta x, y+\Delta y)$ in a neighborhood $\mathcal{N}_{xy}$ is expressed as follows:

$$\widehat{I}(x,y,\Delta x,\Delta y) = I(x+\Delta x, y+\Delta y) - \overline{I(x,y)}\,. \tag{3.4b}$$

Finally, expression (3.5) denotes mean compensated cross correlation including normalization, usually just called normalized cross correlation, $\mathrm{NXC}(x, y)$

$$\arg\max_{d_x,d_y} \frac{\sum \left(\widehat{I_-}(x,y,\Delta x,\Delta x)\cdot \widehat{I_+}(x+d_x, y+d_y, \Delta x, \Delta y)\right)}{\sqrt{\sum \widehat{I_-}(x,y,\Delta x,\Delta y)^2}\sqrt{\sum \widehat{I_+}(x+d_x, y+d_y, \Delta x, \Delta y)^2}}\,, \tag{3.5}$$

where $\sum := \sum_{\Delta x,\Delta y\in\mathcal{N}_{xy}}$. NXC provides a more robust matching criterion: invariance with regard to global (linear) changes in illumination between the patches[1]. This can however also be considered as a deficiency because information is lost — it is a matter of perspective.

Note how the argument needs to be maximized (in SSD it was minimized) for an optimal fit using cross correlation. This has to do with the fact that

[1] *patch*: local neighborhood/surface, we also call it "image window"

NXC can be viewed as an inner product $\frac{\mathbf{a}^{\mathrm{T}}\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos(\angle(\mathbf{a}, \mathbf{b}))$ between vectors $\mathbf{a}$ and $\mathbf{b}$, e.g.

$$\mathbf{a} = \begin{bmatrix} \widehat{I_-}(x,y,-3,-3) \\ \widehat{I_-}(x,y,-3,-2) \\ \widehat{I_-}(x,y,-3,-1) \\ \widehat{I_-}(x,y,-3,0) \\ \widehat{I_-}(x,y,-3,1) \\ \widehat{I_-}(x,y,-3,2) \\ \widehat{I_-}(x,y,-3,3) \\ \widehat{I_-}(x,y,-2,-3) \\ \widehat{I_-}(x,y,-2,-2) \\ \vdots \\ \widehat{I_-}(x,y,3,2) \\ \widehat{I_-}(x,y,3,3) \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \widehat{I_+}(x+d_x,y+d_y,-3,-3) \\ \widehat{I_+}(x+d_x,y+d_y,-3,-2) \\ \widehat{I_+}(x+d_x,y+d_y,-3,-1) \\ \widehat{I_+}(x+d_x,y+d_y,-3,0) \\ \widehat{I_+}(x+d_x,y+d_y,-3,1) \\ \widehat{I_+}(x+d_x,y+d_y,-3,2) \\ \widehat{I_+}(x+d_x,y+d_y,-3,3) \\ \widehat{I_+}(x+d_x,y+d_y,-2,-3) \\ \widehat{I_+}(x+d_x,y+d_y,-2,-2) \\ \vdots \\ \widehat{I_+}(x+d_x,y+d_y,3,2) \\ \widehat{I_+}(x+d_x,y+d_y,3,3) \end{bmatrix} \tag{3.6}$$

for the half window size $m = 3$, i.e. the vectors containing all the differences from the mean of all the pixels for each image's window. Because of the properties of the cosine the result lies in the range $[-1, 1]$. 1 herein denotes a perfect match, 0 means no correlation at all (orthogonal vectors) and -1 means anti-correlation.

3.2 Deformed Window SSD

If the frontal parallel plane assumption is fulfilled, i.e. every object's faces are parallel to the image plane of the cameras, then the previously mentioned formulation of the problem is correct: for a point (x, y) in the left image we can compute an exact disparity estimate d based on the point $(x - d, y)$ in the right image.[2]

As we have seen before, traditional region-based matching metrics compare small windows in left and right image in order to find matches. By adding first order disparity information to the problem, these windows are deformed: we account for perspective distortion. As a matter of fact, adding first order disparity information also adds information about the surface normal to our problem statement, i.e. how surfaces are slanted, all of which we are going to discuss now [LZ05].

Let us now consider a first order approximation of the correspondence of $(x + \Delta x, y + \Delta y)$ from the left image in the right image:

$$x \longrightarrow x - d \overset{x=x+\Delta x}{\Leftrightarrow} x + \Delta x \longrightarrow x + \Delta x - d_{\mathrm{FO}} \,, \tag{3.7a}$$

[2] In the right image, the correspondences are located more left than in the left image, therefore the subtraction, as we would like our disparities to have positive sign.

with $\Delta x = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$.

The first order approximation d_{FO} of $d(x, y)$ is obtained using a Taylor series expansion up to first order (linearization)

$$d_{\mathrm{FO}}(x, y) = d + \frac{\partial}{\partial x} d \cdot \Delta x + \frac{\partial}{\partial y} d \cdot \Delta y \,. \tag{3.7b}$$

This results in the following expression for the first order approximation of the correspondence to be found in the right image

$$x + \Delta x \quad \longrightarrow \quad x + \Delta x - d - \frac{\partial}{\partial x} d \cdot \Delta x - \frac{\partial}{\partial y} d \cdot \Delta y \,. \tag{3.7c}$$

Using this reformulated similarity measure yields the deformed window SSD (dwSSD):

$$\boxed{\begin{aligned} \underset{\{d, \frac{\partial}{\partial x} d, \frac{\partial}{\partial y} d\}}{\arg\min} \sum_{(x+\Delta x, y+\Delta y) \in N_{xy}} \Bigg(& I_{-}(x + \Delta x, y + \Delta y) \\ & - \widetilde{I_{+}}(x + \Delta x - d - \frac{\partial d}{\partial x} \Delta x - \frac{\partial d}{\partial y} \Delta y, y + \Delta y) \Bigg)^2 \end{aligned}}, \tag{3.8}$$

where N_{xy} is the window centered at pixel (x, y).

As the first order approximation probably lies "between" pixels, we are working with floating point pixel positions: $\widetilde{I_{+}}$ denotes the (linearly) interpolated floating point intensity of the two nearest integer index positions in the right image. For a 1-D signal $I_{\mathrm{1D}}(x)$, linear interpolation is defined as

$$\widetilde{I_{\mathrm{1D}}(x_f)} = (1 - \alpha)\, I_{\mathrm{1D}}(x_L) + \alpha I_{\mathrm{1D}}(x_R), \text{ with } \alpha = \frac{x_f - x_L}{x_R - x_L} \,, \tag{3.9}$$

where x_f denotes a floating point pixel position and x_L and x_R the left and right pixels respectively, relatively to it. In other words, the resulting intensity is a sum of weighted neighboring pixels, the weight depending on the distance of their centers to the floating point position given.

To find the argument that produces the minimum in expression (3.8), in practice we use Powell's Direction Set Method, a multi-dimensional minimization algorithm [PTVF92]. We have actually modified it such that once it has found a value of 0 (which in our case is the best possible), it stops searching.

3.3 Disparity Derivatives Demystified

We have seen disparity derivatives introduced in the last section. But what do they mean intuitively?

First of all, it is important to understand that we are inversely trying to guess disparity derivatives by looking at how a square surface element in the one image is best deformed to fit in the other image. Indeed, the algorithm used to minimize expression (3.8) knows nothing about derivatives of disparity, as the values $\frac{\partial d}{\partial x}$ and $\frac{\partial d}{\partial y}$ are only given implicitly. For the minimizer, these values are just scalar unknowns, not functions or alike. As such, the minimizer deforms the window more and more, until it fits best [LZ05].

In the classical sense, one can interpret the scalar value $\frac{\partial d}{\partial x}$ as "the rate of change of d as we move in the direction of x". Indeed, this is true: if one moves from left to right in image space, disparity either increases, decreases or stays the same. Accordingly, the disparity derivative $\frac{\partial d}{\partial x}$ becomes positive, negative, or zero, respectively. As disparity is related directly to depth (Expression (2.12)) we can also say that if $\frac{\partial d}{\partial x} > 0$, depth decreases and so on. Analogous explanations hold for $\frac{\partial d}{\partial y}$.

The meaning of disparity derivatives in the context of the minimization problem in (3.8) is slightly more complex, but not less evident. Essentially, disparity derivatives compensate for the perspective distortion encountered when comparing a window of data in the left image to a window of data in the right image. Since the camera has moved right, all objects have moved left. But: the objects that are nearer to the camera have moved more left than the objects far away. Remember: this was the initial idea of relating the movement of objects in the two images and extract information from it, i.e. depth.

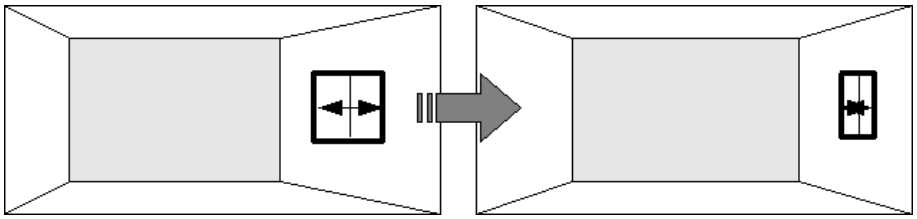


FIGURE 3.2: Two perspectives of a room and a deformed window with $\frac{\partial d}{\partial x} > 0$.

Now take a look at Figure 3.2. You are looking at a room from two perspectives. The right perspective has had the point of view moved to the right (compared to the left perspective). Therefore, less of the right wall is visible in the right perspective, or more correctly: the projection of the right wall onto the image plane has changed shape. The search window of our region matching function therefore would make a mistake if it would use a window of the same size and shape in the right image than in the left image in order to look for a correspondence: it could never cover the same region as in the left image.

The two ways that were just presented to perceive disparity derivatives

are indeed equivalent. As an example consider the right wall in Figure 3.2, which has positive disparity derivative $\frac{\partial d}{\partial x}$: when we move to the right in image space from the center of the image, we come closer to the camera, because the wall moves towards the camera. The correspondence of the point x in the left image becomes

$$x - d + \left(1 - \tfrac{\partial d}{\partial x}\right) \cdot \Delta x - \tfrac{\partial d}{\partial y} \Delta y$$

in the right image (remember Expression (3.8)), hence the matching window is compressed in x-direction: moving one step right in the left window moves a little less than one step right in the right window (see the part of the expression highlighted in light gray color).

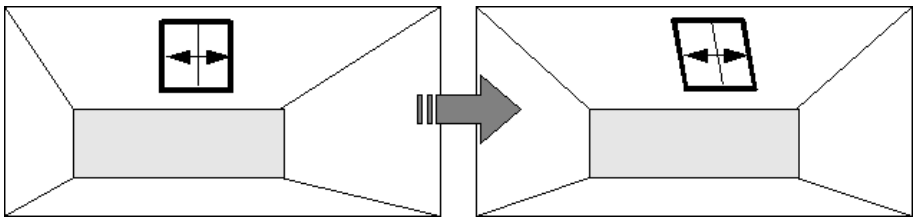


FIGURE 3.3: Two perspectives of a room and a deformed window with $\frac{\partial d}{\partial y} < 0$.

A similar thing happens in y-direction (see Figure 3.3), however, since we are working with a rectified stereo pair, i.e. epipolar lines are horizontal, the y-coordinate in the right image is not affected but its disparity derivative $\frac{\partial d}{\partial y}$ nevertheless has an influence on x (since d is a function of (x, y)). See the dark gray highlight in the previously shown expression.

We, as a human observer, intuitively know about such properties of 3-D space and perspective, but the machine would always look for correspondences in a frontal parallel plane, hence systematic error would occur, if the problem formulation i.e. the window shape was not improved. For all possible combinations of disparity derivatives, see Table 3.1.

See Table 3.2 for the values of the various disparity derivatives as they would be for the walls of our room example in Figures 3.2 and 3.3. Also see Figure 3.4 for a summary of possible deformations and their respective disparity derivatives. Intuitively, for negative $\frac{\partial d}{\partial x}$, the window expands in x-direction, for positive $\frac{\partial d}{\partial x}$ it is compressed in x-direction, and $\frac{\partial d}{\partial y}$ defines how it is slanted.

Since disparity derivatives encode a plane, they automatically encode normal vectors, which is one way of defining a plane, as the following definition will show.

Definition 3.3.1 *Let* $\mathbf{p} \in \mathbb{R}^3$ *be the point we wish to lie in the plane, and* $\mathbf{n} \in \mathbb{R}^3$ *a nonzero normal vector to the plane. The desired plane is the set of*

all points $\mathbf{r} \in \mathbb{R}^3$ *such that*

$$\mathbf{n} \cdot (\mathbf{r} - \mathbf{p}) = 0\,. \tag{3.10}$$

Computing the normalized surface normal is straightforward: disparity derivatives $\left\{\frac{\partial d}{\partial x}, \frac{\partial d}{\partial y}\right\}$ at (x, y) are a basis of the tangent plane at (x, y). The direction of the tangent plane's normal vector therefore is $\frac{\partial d}{\partial x} \otimes \frac{\partial d}{\partial y}$ and its magnitude is $\|\frac{\partial d}{\partial x} \otimes \frac{\partial d}{\partial y}\|$, where $\otimes$ denotes the vector cross product. This yields for all available points the unit normal vector field:

$$\begin{aligned}
\mathbf{N} &= \frac{\frac{\partial d}{\partial x} \otimes \frac{\partial d}{\partial y}}{\|\frac{\partial d}{\partial x} \otimes \frac{\partial d}{\partial y}\|} \\
&= \frac{1}{\sqrt{(-\frac{\partial d}{\partial x})^2 + (-\frac{\partial d}{\partial y})^2 + 1^2}} \begin{bmatrix} 0 \cdot \frac{\partial d}{\partial y} - \frac{\partial d}{\partial x} \cdot 1 \\ \frac{\partial d}{\partial x} \cdot 0 - 1 \cdot \frac{\partial d}{\partial y} \\ 1 \cdot 1 - 0 \cdot 0 \end{bmatrix} \\
&= \frac{\left(-\frac{\partial d}{\partial x}, -\frac{\partial d}{\partial y}, 1\right)^{\mathrm{T}}}{\sqrt{\frac{\partial d}{\partial x}^2 + \frac{\partial d}{\partial y}^2 + 1}}\,.
\end{aligned} \tag{3.11}$$

$\frac{\partial d}{\partial x}$	$\frac{\partial d}{\partial y}$	3-D surface
$= 0$	$= 0$	Frontal Parallel
$\neq 0$	$= 0$	Horizontally Slanted
$= 0$	$\neq 0$	Vertically Slanted
$\neq 0$	$\neq 0$	General Surface

TABLE 3.1: Configuration of 3-D geometry and their corresponding first order disparities.

$\frac{\partial d}{\partial x}$	$\frac{\partial d}{\partial y}$	Location
< 0	$= 0$	Left Wall
> 0	$= 0$	Right Wall
$= 0$	< 0	Ceiling
$= 0$	> 0	Floor
$= 0$	$= 0$	Frontal Parallel Wall (Light Gray)

TABLE 3.2: 3-D geometry and their first order disparities wrt. room (Figs. 3.2—3).

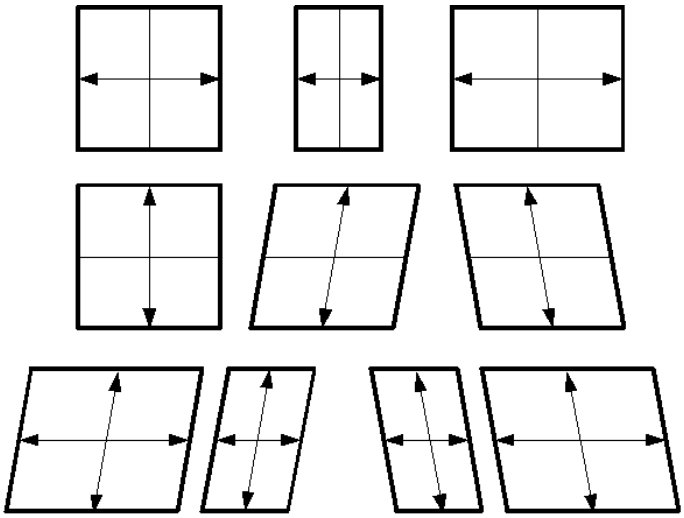


FIGURE 3.4: First order disparity derivatives and their corresponding deformed region matching windows. **1st row (from left to right)**: zero, positive and negative $\frac{\partial d}{\partial x}$ ($\frac{\partial d}{\partial y} = 0$) **2nd row (from left to right)**: zero, positive and negative $\frac{\partial d}{\partial y}$, ($\frac{\partial d}{\partial x} = 0$) **3rd row (from left to right)**: $\{\frac{\partial d}{\partial x} < 0, \frac{\partial d}{\partial y} > 0\}$, $\{\frac{\partial d}{\partial x} > 0, \frac{\partial d}{\partial y} > 0\}$, $\{\frac{\partial d}{\partial x} > 0, \frac{\partial d}{\partial y} < 0\}$ and $\{\frac{\partial d}{\partial x} < 0, \frac{\partial d}{\partial y} < 0\}$.

3.4 Geometric Contextual Information

In this section, the method is further extended to include geometric information from a local neighborhood, since plain region matching metrics for obtaining depth information usually do not provide unambiguous results: the traditional region based matching method may have found several good correspondences for a surface patch **i**, the best of which we remember. Hence, in what follows, we have several possible matches ("candidate matches") in each point to choose from [LZ05].

The principle of geometric consistency means that, if two points are on the same surface, the known position and normal measurements at point **i** and the computed approximation of the local neighborhood around **i** should agree with the actual data (one of the best fits obtained) at the computed neighboring point, which we will call **j**. Figure 3.5 shows possible matches that are not consistent, i.e. have the wrong orientation or lie off the tangent plane (dashed lines).

The geometric compatibility between two matching pairs can therefore be expressed as:

$$g_{\mathbf{ij}} = 1 - \frac{1}{m}\Big(|\mathbf{v_{ij}} \cdot \mathbf{N_i}| + |\mathbf{v_{ji}} \cdot \mathbf{N_j}|\Big)\,, \tag{3.12}$$

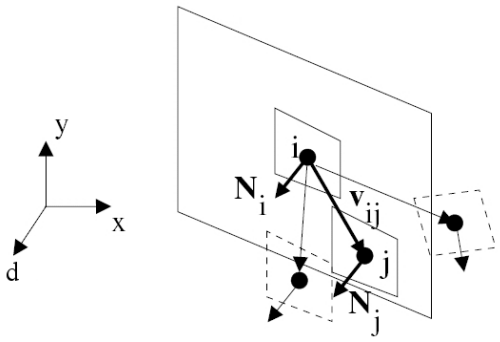


FIGURE 3.5: A surface patch with consistent and inconsistent correspondences.

where m is a normalization constant.[3] By that it follows that $0 \leq g_{\mathbf{ij}} \leq 1$.

Let us now take a closer look at the metric in (3.12): The term penalizes deviations of the neighboring match's depth from the current point's depth. It simultaneously penalizes deviations of the neighboring match's normal from the current point's normal. Here, the dot product does not consider the angle between the two vectors, but rather the length of the projection of one onto the other, since $\mathbf{v_{ij}}$ and $\mathbf{v_{ji}}$ are not normalized. Obviously, perpendicular vectors still give the best result, i.e. zero in the dot product; additionally, a vector $\mathbf{v}$ with a large magnitude penalizes an off-zero angle even more.

The expected depth and normal is supposed not to deviate too much from the current position (smoothness constraint). A resulting value of g close to 1 denotes a good match, and a value close to 0 a bad match. We currently ignore the orientation of the projection of one vector onto the other, therefore the absolute value is used.

With this measure in mind, the following iterative update scheme, which we call geometric support space iterative diffusion (GSID) is then used to compute the geometric support

$$s_{\mathbf{i}}^{t} = \begin{cases} 1 - \frac{d_{\mathbf{i}}}{d} & , \text{if } t = 0 \\ X \cdot \sum_{\mathbf{j} \in \mathcal{N}_{\mathbf{i}}} g_{\mathbf{ij}} s_{\mathbf{j}}^{t-1} & , \text{if t} > 0 \end{cases} \tag{3.13}$$

that a point receives from its neighboring candidate matches, where $d_{\mathbf{i}}$ is the initial disparity estimate found at point $\mathbf{i}$ using a region matching metric of choice (see section 3.1), d is a normalization constant, which we choose to be the maximum disparity of the scene (that is usually defined manually), and t denotes the time, respectively the iteration number.

The term in the case $t > 0$ is not the original term from [LZ05]; why we choose to modify it and what X exactly is, is explained in Appendix A.

[3] If we do not have a general value of m suitable to normalize, i.e. at least as large as the maximum value to be normalized, we could also set this as a threshold and ignore values exceeding it. In our experiments, $m = 50$, which encompasses all values obtained.

From all of this, and with the assumption that X has normalizing behavior[4], i.e. $X \leq \frac{1}{|\mathcal{N}_\mathbf{i}|}$, it again follows that $0 \leq s_\mathbf{i}^t \leq 1, \forall t$, because $|\mathcal{N}_\mathbf{i}| \geq \sum s_\mathbf{i} = \sum 1 s_\mathbf{i} \geq \sum g_\mathbf{i} s_\mathbf{i}$, $s_\mathbf{i} \in [0,1]$ and $g_\mathbf{i} \in [0,1]$. Considerably less than ten iterations seem to be enough for our method to converge (see Chapter 5 for results).

This approach reproducibly eliminates many more bad matches (visible as noise) from the intermediate result, which was simply the best fit obtained by deformed window SSD inside of the manually defined disparity range.

Note that we have obtained several candidate matches per pixel, therefore in our implementation we need to iterate over all pixels and all matches in (3.13) and also consider *every* match of every pixel in the sums herein. $s_\mathbf{i}$ and $s_\mathbf{j}$ denote these sets of fits (there is no further indexing in this mathematical formulation). As such, the GSID happens over a 3-D (x, y, d) geometric support region (GSR).

The surface normals are also updated based on the normals of neighbors that deviate less than $\pm\frac{\pi}{4}$ radians from the current normal by a least square fit, in order to reduce the effect of local noisy measurements:

$$\mathbf{N}_\mathbf{p}^{t+1} = \frac{\sum_{\mathbf{q}\in\mathcal{N}_\mathbf{p}} \mathbf{N}_\mathbf{q}^t}{\sqrt{(\sum_{\mathbf{q}\in\mathcal{N}_\mathbf{p}} \mathbf{N}_{\mathbf{q}_x}^t)^2 + (\sum_{\mathbf{q}\in\mathcal{N}_\mathbf{p}} \mathbf{N}_{\mathbf{q}_y}^t)^2 + (\sum_{\mathbf{q}\in\mathcal{N}_\mathbf{p}} \mathbf{N}_{\mathbf{q}_z}^t)^2}} \tag{3.14}$$

where $\mathbf{N}_{\mathbf{q}_*}^t$ denotes an individual component of the three-vector

$$\mathbf{N}_q^t = \left[\mathbf{N}_{\mathbf{q}_x}^t \mathbf{N}_{\mathbf{q}_y}^t \mathbf{N}_{\mathbf{q}_z}^t\right]^\mathrm{T}$$

which is the vector $\mathbf{N}(\mathbf{q})$ at time step t.

For an overview of the algorithm, see Figure 3.6.

[4] which it does, see Appendix A

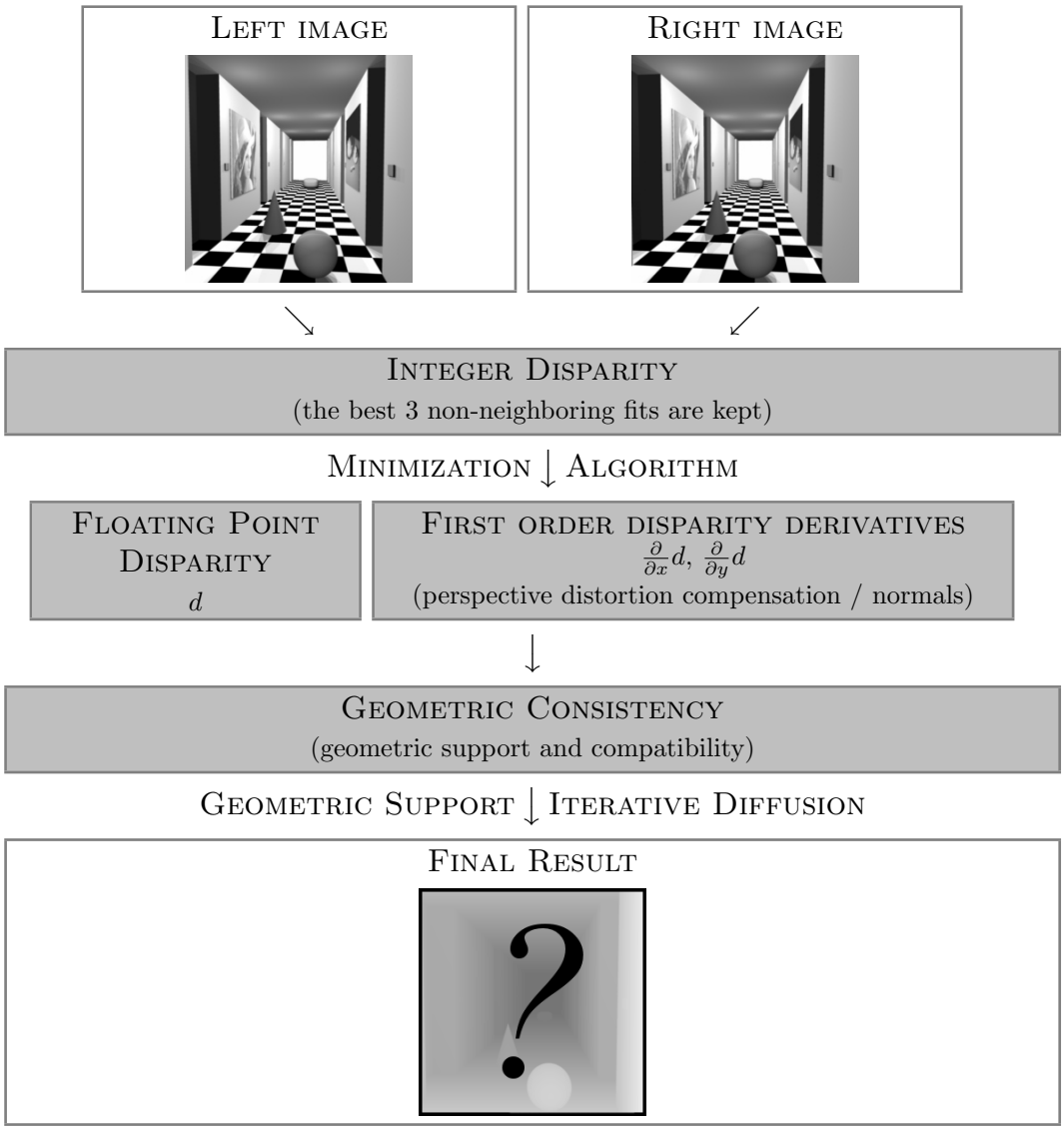


Figure 3.6: Algorithm in a Nutshell. Grey background indicates data that is computed and kept for each of the initial fits (local minima). Starting from the first minimization, all values have at least single floating point precision. Each box also explains the intuitive meaning and purpose of the respective value in parentheses.

Chapter 4

Variational Method

Variational methods are global differential methods. They rely on the solution of partial differential equations (PDEs) to perform their job. A variational method offers transparent modeling and provides dense flow fields, unlike local methods where one usually needs intermediate and post-processing steps. This disparity refinement (filling-in and sub-pixel accuracy) comes for free due to the continuous modeling of the problem.

We are designing a solution to the problem at hand by embedding it into a so-called optic flow problem. This will result in dense flow fields, but the complicated interactions between pixels herein require to solve large linear or non-linear systems of equations. Therefore we have introduced the SOR method in Chapter 2. Additionally, we will use a coarse-to-fine approach, which makes linearization a valid tool, even in the presence of large displacements.

4.1 Modeling

We would now like to find an energy minimization formulation similar to the local method used in the previous chapter and show how we tackle the problem step by step.

Data Term: From Local To Local Differential Method

We only consider orthoparallel setups, i.e. the complete displacement $\mathbf{d} = \begin{bmatrix} d_x & d_y \end{bmatrix}^{\mathrm{T}}$ happens in x-direction only, i.e. $d_y = 0$. The data term must penalize deviations from the disparity constancy assumption, we have seen squared differences before:

$$\left(I_-(x,y) - I_+(x+d_x,y)\right)^2 \ . \tag{4.1}$$

We recast the stereo problem as an optic flow problem: we now understand I_- to be the image at time t and I_+ the one at time $t+1$, i.e. we

pretend the two images were recorded by the same camera at two points in time instead of two different cameras at the same point in time but different locations, which is a different formulation of the same problem. An approach where the generalized epipolar constraint has been integrated into an optic flow problem can be found in [SBW05].

The image domain is increased by one dimension (now $\Omega \subset \mathbb{R}^3$), the term inside the square of (4.1) can be rewritten as

$$I(x + d_x, y, t + 1) - I(x, y, t) = 0 \,, \tag{4.2}$$

which says that both images should be more or less similar. This very popular data term is also called the brightness constancy constraint equation. Linearizing the part of the term from time $t + 1$ gives

$$I(x + d_x, y, t + 1) \approx I(x, y, t) + \partial_x I d_x + \partial_t I \,. \tag{4.3}$$

Expression (4.2) then becomes

$$I + \partial_x I d_x + \partial_t I - I = \partial_x I d_x + \partial_t I = 0 \,. \tag{4.4}$$

The previous expression is called the linearized optic flow constraint.

As one can see, what once was the disparity map has now become a 3-D vector field (where the third dimension is time), also called optic flow:

$$\begin{aligned} &\mathbf{W} : \mathbb{R}^3 \to \mathbb{R}^3 \\ &\mathbf{W}(\mathbf{w}) \mapsto \begin{bmatrix} W_1(\mathbf{w}) \\ W_2(\mathbf{w}) \\ W_3(\mathbf{w}) \end{bmatrix} = \begin{bmatrix} u(\mathbf{w}) \\ v(\mathbf{w}) \\ 1 \end{bmatrix} \,, \end{aligned} \tag{4.5}$$

with $\mathbf{w} = (x, y, t)^{\mathrm{T}}$ (remember Section 2.1.1 for how vector fields exactly work). We denote the horizontal flow component with u, and the vertical flow component with v. One obtains nothing else than a simplified Lucas/Kanade optic flow method [LK81] with horizontal flow only ($v = 0$ always). The "distance" between time steps always is 1 (W_3) [Luc84].

Figure 4.1 depicts a simple example of optic flow: at $(0, 0, t)$ we have displacement $\mathbf{W}(0, 0, t) = (0, 1, 1)^{\mathrm{T}}$.

Motion Tensor Notation

We can now reformulate (4.1) using the linearized form (4.4) (with $u = d_x$), which yields a quadratic form

$$\begin{aligned} (I_-(x, y) - I_+(x + d_x, y))^2 &\to (I(x + d_x, y, t + 1) - I(x, y, t))^2 \\ &\overset{\text{LIN.}}{=} (\partial_x I d_x + \partial_t I)^2 \\ &\overset{u=d_x}{=} \left(\begin{bmatrix} u & 0 & 1 \end{bmatrix} \begin{bmatrix} \partial_x I \\ \partial_y I \\ \partial_t I \end{bmatrix} \right)^2 \\ &= (\mathbf{W}^{\mathrm{T}} \nabla_3 I)^2 \,. \end{aligned} \tag{4.6a}$$

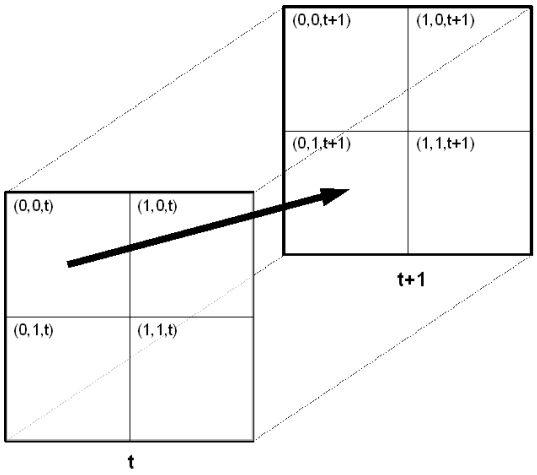


FIGURE 4.1: Optic flow between two frames at times t and $t+1$ (frame size 2×2).

We can then express this quadratic form in motion tensor notation [Bru06]:

$$(\mathbf{W}^{\mathrm{T}}\nabla_3 I)^2 = (\mathbf{W}^{\mathrm{T}}\nabla_3 I)(\nabla_3 I^{\mathrm{T}}\mathbf{W}) = \mathbf{W}^{\mathrm{T}} \underbrace{(\nabla_3 I \nabla_3 I^{\mathrm{T}})}_{=:\mathbf{J}(\nabla_3 I)} \mathbf{W} \,. \tag{4.6b}$$

$\mathbf{J}(\nabla_3 I)$ is a symmetric positive semi-definitive 3×3 matrix called motion tensor.

$$\begin{aligned} \mathbf{W}^{\mathrm{T}}\mathbf{J}\mathbf{W} &= \begin{bmatrix} W_1 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \partial_x I \\ \partial_y I \\ \partial_t I \end{bmatrix} \odot \begin{bmatrix} \partial_x I & \partial_y I & \partial_t I \end{bmatrix} \right) \begin{bmatrix} W_1 \\ 0 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} W_1 & 0 & 1 \end{bmatrix} \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix} \begin{bmatrix} W_1 \\ 0 \\ 1 \end{bmatrix} , \end{aligned} \tag{4.7a}$$

with

$$\begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix} = \begin{bmatrix} (\partial_x I)^2 & \partial_x I \partial_y I & \partial_x I \partial_t I \\ \partial_y I \partial_x I & (\partial_y I)^2 & \partial_y I \partial_t I \\ \partial_t I \partial_x I & \partial_t I \partial_y I & (\partial_t I)^2 \end{bmatrix} , \tag{4.7b}$$

where $\odot$ here denotes the tensor product (matrix multiplication).

Amongst others, the motion tensor notation has the advantage that only one implementation for all constancy assumptions is required, therefore reducing the workload for experiments, in case of multiple data terms to be investigated.

Smoothness: From Local Differential To Global Differential Method

We additionally enforce smoothness constraints mentioned in the beginning of this section. Not only do we enforce smoothness of the flow field gradient,

we additionally penalize locally affine transformations (instead of only locally constant ones). It is in essence an extended variant of the Horn/Schunck method [HS81].

This means we have to penalize second order derivatives of the flow field. Consequently, functions of the type $u(x, y) = ax + by + c$ are mapped to zero by the regularizer: those are exactly affine transformations.

All of this makes our method a global differential method with filling-in effect, the filling-in happening via diffusion from the neighborhood. The resulting continuous energy functional looks like:

$$\boxed{E(\mathbf{W}) = \int_y \int_x \mathbf{W}^{\mathrm{T}} \mathbf{J} \mathbf{W} + \alpha \underbrace{|\nabla \mathbf{W}|^2}_{F_1} + \beta \underbrace{\|\mathcal{H}(\mathbf{W})\|_{\mathrm{Fr}}^2}_{F_2} \, dx \, dy} \tag{4.8a}$$

With $\nabla \mathbf{W} = (\nabla u, 0, 0)$, we have

$$|\nabla \mathbf{W}| = \sqrt{|\nabla u|^2} = \sqrt{(\partial_x u)^2 + (\partial_y u)^2} \, . \tag{4.8b}$$

Analogously, the Frobenius norm (entry-wise norm) of the Hessian of $\mathbf{W}$ is

$$\|\mathcal{H}(\mathbf{W})\|_{\mathrm{Fr}} = \sqrt{\sum_{i=1}^{2} \sum_{j=1}^{2} |H(u)_{i,j}|^2} = \sqrt{(\partial_{xx} u)^2 + 2(\partial_{xy} u)^2 + (\partial_{yy} u)^2} \, , \tag{4.8c}$$

with $\mathcal{H}(u)_{i,j}$ being the entry in row i and column j in $\mathcal{H}(u)$. We will similarly address specific elements of $\mathbf{J}$ using $J_{i,j}$ in the upcoming sections. The constants α and β indicate the importance of the respective smoothness assumption; they are called regularization parameters.

We show the assumptions for the global method in comparison with what the local method did in Table 4.1.

The Warping Strategy: From Small To Large Displacements

If the displacements are small (i.e. in the order of one pixel), linearization gives a valid approximation (Expression (4.9a)). However, when we start to deal with large displacements, this formulation is no longer valid. In this case we postpone the linearization of the constancy assumptions to the numerical scheme.

$$E(\mathbf{W}) = \int_\Omega (\partial_x I u + \partial_t I)^2 + S_{\alpha\beta}(u, v) \, dx \, dy \tag{4.9a}$$

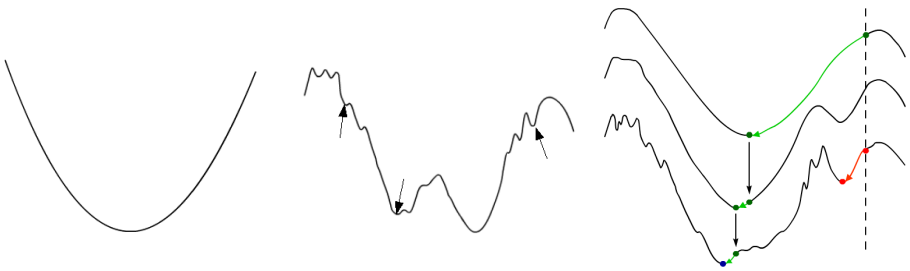
$$E(\mathbf{W}) = \int_\Omega \left(I(x + u, y, t + 1) - I(x, y, t)\right)^2 + S_{\alpha\beta}(u, v) \, dx \, dy \tag{4.9b}$$

This however makes the non-linearized functional (4.9b) non-convex, i.e. it contains multiple local minima and its Euler-Lagrange (E-L) equations

Local Method	Global Differential Method
SSD fit yielding disparity d	differential formulation yielding displacement vector field $\mathbf{W}$ (data term)
geometric support: neighboring pixels should be close to each other in (x, y, d)-space	constancy of the displacement field gradient (smoothness term I)
geometric support: choose best neighboring fit based on similarity of normal vectors (bias towards slanted surfaces)	constancy of the variation of the displacement field gradient, i.e. the Hessian, allowing affine motion (smoothness term II)

TABLE 4.1: Comparison between approaches in local method and in our global differential method.

have multiple solutions (see Figure 4.2ab). This non-convex optimization problem will then be approximated by a series of convex optimization problems, that is, embedded in a coarse-to-fine hierarchy and a convex problem will be solved at each hierarchy level (see Figure 4.2c).



(a) convex (b) nonconvex (c) hierarchy

FIGURE 4.2: Convex and non-convex functions. For the non-convex function we indicate several local minima. We also show the coarse-to-fine hierarchy. (Taken from [Bru07])

In such cases the warping strategy comes into play: it is an hierarchical incremental fixed point iteration. We use the flow field obtained at coarse scale (downsampled image) at the next finer scale (determined by scaling parameter η) to solve the difference problem: technically, we warp the second image in order to compensate for this estimated motion. We do this until the finest scale has been reached and sum up all the flow contributions from all scales [BBPW04, Bru07].

On the coarse grid, linearization is indeed a valid approximation: the distance from one pixel to the next in the coarse grid is however a larger

distance than the one in the finer grid. This approach might lead to small objects being unnoticed, since they are not present at all in the coarse grid.

The Euler-Lagrange equation for the non-linearized functional is given by

$$\begin{aligned} 0 = \partial_x I(x+u, y, t+1) \left(I(x+u, y, t+1) - I(x, y, t)\right) \\ - \alpha \Delta u + \beta \Delta^2 u \, . \end{aligned} \tag{4.10}$$

We introduce the fixed point iteration step

$$\begin{aligned} 0 = \partial_x I(x+u^k, y, t+1) \left(I(x+u^{k+1}, y, t+1) - I(x, y, t)\right) \\ - \alpha \Delta u^{k+1} + \beta \Delta^2 u^{k+1} \, . \end{aligned} \tag{4.11a}$$

We further introduce an incremental computation by the splitting

$$u^{k+1} = u^k + du^k \, . \tag{4.11b}$$

We then linearize the data term with respect to du^k

$$\begin{aligned} I(x+u^{k+1}, y, t+1) &= I(x+u^k+du^k, y, t+1) \\ &= I(x+u^k, y, t+1) + \partial_x I(x+u^k, y, t+1) du^k \, , \end{aligned} \tag{4.11c}$$

yielding the linearized step at fixed point iteration k:

$$\begin{aligned} 0 =& \partial_x I(x+u^k, y, t+1) \\ & \left(\partial_x I(x+u^k, y, t+1) du^k + I(x+u^k, y, t+1) - I(x, y, t)\right) \\ & - \alpha \Delta (u^k + du^k) + \beta \Delta^2 (u^k + du^k) \, . \end{aligned} \tag{4.12a}$$

This can again be related to a convex problem and has therefore a unique solution. In motion tensor notation we can write:

$$0 = J_{11}^k du^k + J_{13}^k - \alpha \Delta u^k - \alpha \Delta du^k + \beta \Delta^2 u^k + \beta \Delta^2 du^k \, , \tag{4.12b}$$

where $J^k = \nabla_3 I(x+u^k, y, t+1) \nabla_3 I(x+u^k, y, t+1)^{\mathrm{T}}$.

This fixed point iteration is then embedded in the previously mentioned coarse-to-fine strategy.

4.2 Discretization and Minimization

There are two methods for finding a minimum of an energy functional in a discrete setting. One is writing down its Euler-Lagrange (E-L) equation(s), setting them to zero and then discretizing them.

However, obtaining the correct boundary conditions can be difficult with this approach. Especially the formulation using E-L equations usually contains a normal vector to account for the condition at the boundaries, which

is especially hard to discretize later on. Correct boundary conditions are however important, since e.g. the E-L equations resulting from an energy functional containing the Laplacian and another one containing the Hessian as smoothness terms are both the same, only their boundary conditions differ.

Therefore, we discretize the energy functional directly, which is the more difficult way of obtaining a discretization, but which on the other hand automatically contains the correct boundary conditions. We use the necessary condition for obtaining a minimum of the energy functional,

$$0 \stackrel{!}{=} \frac{\partial E}{\partial u} \approx \left(\frac{\partial E}{\partial u_{1,1}}, \ldots, \frac{\partial E}{\partial u_{N,M}} \right) . \tag{4.13}$$

We use the $\approx$ sign to denote that we are going from the continuous to the discrete setting, i.e. we are approximating.

4.2.1 Discretization

We assume a rectangular grid with grid size h_x in x-direction and h_y in y-direction and perform the following discretizations, step by step:

▷ **image** : discretization of I,

▷ **image derivatives** : discretization of $\partial_x I$, $\partial_y I$, $\partial_t I$,

▷ **motion tensor** : the previous two discretizations yield the discretization of $\mathbf{J}$ (in our case only its entries J_{11} and J_{13} are relevant),

▷ **flow field** : discretization of u, yielding discretization of W,

▷ **flow derivatives - 1st order** : discretization of $\partial_x u$ and $\partial_y u$,

▷ **flow derivatives - 2nd order** : discretization of $\partial_{xx} u$, $\partial_{xy} u$ and $\partial_{yy} u$.

Discretization of Data Term

Based on the grid spacings h_x and h_y we denote the image value at position (i, j, t) (t indicates which image we are in) by

$$I_{i,j,t} = I\left((i-1)h_x, (j-1)h_y, t\right) , \tag{4.14}$$

for $i = 1, \ldots, N$, $j = 1, \ldots, M$ and $t = 1, 2$. N and M are the sizes of the images in x and y directions, respectively.

By that, we obtain for the derivatives, using averaged central differences for the space, and forward differences for the time part:

$$(\partial_x I)_{i,j} = \frac{1}{2} \left(\frac{I_{i+1,j,t+1} - I_{i-1,j,t+1}}{2h_x} + \frac{I_{i+1,j,t} - I_{i-1,j,t}}{2h_y} \right) , \tag{4.15a}$$

$$(\partial_y I)_{i,j} = \frac{1}{2}\left(\frac{I_{i,j+1,t+1} - I_{i,j-1,t+1}}{2h_x} + \frac{I_{i,j+1,t} - I_{i,j-1,t}}{2h_y}\right) , \tag{4.15b}$$

$$(\partial_t I)_{i,j} = \frac{I_{i,j,t+1} - I_{i,j,t}}{h_t} = I_{i,j,t+1} - I_{i,j,t} . \tag{4.15c}$$

h_t is usually set to 1 and the image data is mirrored at the boundaries, i.e. $I_{i+1,j}$ exists even for $i = N$, and consequently the derivative at the boundary is zero.

The discretization of the motion tensor is now straightforward:

$$\begin{aligned}
(J_{11})_{i,j} &= (\partial_x I)^2_{i,j} , & (J_{12})_{i,j} &= (\partial_x I \partial_y I)_{i,j} = (J_{21})_{i,j} , \\
(J_{22})_{i,j} &= (\partial_y I)^2_{i,j} , & (J_{13})_{i,j} &= (\partial_x I \partial_t I)_{i,j} = (J_{31})_{i,j} , \\
(J_{33})_{i,j} &= (\partial_t I)^2_{i,j} , \quad \text{and} & (J_{23})_{i,j} &= (\partial_y I \partial_t I)_{i,j} = (J_{32})_{i,j} .
\end{aligned}$$

Similarly to the image, we discretize the flow field as

$$u_{i,j} = u\left((i-1)h_x, (j-1)h_y\right) , \tag{4.16}$$

for $i = 1, \ldots, N$ and $j = 1, \ldots, M$, resulting in $W_{i,j} = [u_{i,j} \quad 0 \quad 1]^{\mathrm{T}}$.

Discretization of Smoothness Term I

We now focus on the term F_1 for which we simply use forward finite differences

$$\begin{aligned}
F_1 &= (\partial_x u)^2 + (\partial_y u)^2 \\
&\approx \left(\frac{u_{i+1,j} - u_{i,j}}{h_x}\right)^2 + \left(\frac{u_{i,j+1} - u_{i,j}}{h_y}\right)^2 .
\end{aligned} \tag{4.17}$$

Discretization of Smoothness Term II

For the components of the term F_2 with respect to one variable only, we use standard second differences

$$\begin{aligned}
F_{2_1} &= (\partial_{xx} u)^2 + (\partial_{yy} u)^2 \\
&\approx \left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2}\right)^2 + \left(\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2}\right)^2 .
\end{aligned} \tag{4.18}$$

For the mixed derivatives of the term F_2, we use averaged backward/forward differences, resulting in central differences

$$\begin{aligned}
F_{2_2} &= (\partial_{xy} u)^2 \\
&\approx \left(\frac{1}{4}\left(\frac{\frac{u_{i+1,j+1} - u_{i,j+1}}{h_x} - \frac{u_{i+1,j} - u_{i,j}}{h_x}}{h_y} + \frac{\frac{u_{i+1,j} - u_{i,j}}{h_x} - \frac{u_{i+1,j-1} - u_{i,j-1}}{h_x}}{h_y}\right.\right. \\
&\quad \left.\left. + \frac{\frac{u_{i,j+1} - u_{i-1,j+1}}{h_x} - \frac{u_{i,j} - u_{i-1,j}}{h_x}}{h_y} + \frac{\frac{u_{i,j} - u_{i-1,j}}{h_x} - \frac{u_{i,j-1} - u_{i-1,j-1}}{h_x}}{h_y}\right)\right)^2 \\
&= \left(\frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{4h_x h_y}\right)^2 .
\end{aligned} \tag{4.19}$$

The complete discrete energy functional thus looks like:

$$
\begin{aligned}
E(\mathbf{W}) = &\sum_{i=1}^{N}\sum_{j=1}^{M} W_{i,j}^{\mathrm{T}} J_{i,j} W_{i,j} \\
&+ \alpha \left(\left(\frac{u_{i+1,j} - u_{i,j}}{h_x} \right)^2 + \left(\frac{u_{i,j+1} - u_{i,j}}{h_y} \right)^2 \right) \\
&+ \beta \Bigg(\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} \right)^2 \\
&\quad + 2 \left(\frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{4h_x h_y} \right)^2 \\
&\quad + \left(\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} \right)^2 \Bigg)
\end{aligned}
\tag{4.20}
$$

Alternative Discretization of Smoothness Term II

For the mixed derivatives of the term F_2, we have tried the alternative expression

$$
\begin{aligned}
F_{2_2} &= (\partial_{xy} u)^2 \\
&\approx \frac{1}{4} \Bigg(\left(\frac{u_{i+1,j+1} - u_{i,j+1} - u_{i+1,j} + u_{i,j}}{h_x h_y} \right)^2 + \left(\frac{u_{i+1,j} - u_{i,j} - u_{i+1,j-1} + u_{i,j-1}}{h_x h_y} \right)^2 \\
&\quad + \left(\frac{u_{i,j+1} - u_{i-1,j+1} - u_{i,j} + u_{i-1,j}}{h_x h_y} \right)^2 + \left(\frac{u_{i,j} - u_{i-1,j} - u_{i,j-1} + u_{i-1,j-1}}{h_x h_y} \right)^2 \Bigg) .
\end{aligned}
\tag{4.21}
$$

The discrete energy functional will be similar to (4.20), except for the mixed term in its second to last line, which would need to be replaced. There are many possibilities of discretizing a certain expression. Generally, one can say the higher the order of consistency of the approximation, the qualitatively better it will be.

4.2.2 Minimization

Stencil Notation and Scope of Application

A stencil indicates the new value of the center pixel (shown in gray) with regard to the sum of the neighboring pixels with the given coefficients

$(i-1, j-1)$	$(i, j-1)$	$(i+1, j-1)$
$(i-1, j)$	(i, j)	$(i+1, j)$
$(i-1, j+1)$	$(i, j+1)$	$(i+1, j+1)$

Applying a stencil to an image (equally a flow field) means positioning its center (consecutively) onto each pixel of the image and performing the above-mentioned operation.

We use operator overloading for stencils: if their centers overlap then "+" denotes component-wise addition on all overlapping elements, and for "·" there holds some sort of distributivity law.

As mentioned before, boundary conditions are extremely important. If the stencil is completely inside the data, there is no problem. However, when parts of it lie outside the data, special care must be taken to correct the weights of the elements still inside the image — their sum must e.g. still be equal to zero, etc.. To that end we introduce the indicator function

$$\chi_{[x_L,x_U]\times[y_L,y_U]}(i,j) = \begin{cases} 1 & , i \in [x_L, x_U] \text{ and } j \in [y_L, y_U] \\ 0 & , \text{else} . \end{cases} \tag{4.22}$$

which "activates" certain terms in certain regions only. Usually, we use χ inside of a stencil: then, the argument (i, j) is implicitly given by the position of the center element of the stencil. x_L, y_L and x_U, y_U indicate the lower and upper bounds, respectively, for the x and y coordinates.

Minimization of Smoothness Term I

In order to yield the corresponding boundary conditions we differentiate with regard to $u_{k,l}$ which can correspond to either one of the u's in F_1, that is $u_{i+1,j}$, $u_{i,j}$ or $u_{i,j+1}$.

Practically, one can do it this way:

- ▷ differentiate F_1 with regard to each variable therein,
- ▷ perform an index shift setting the center to the variable being differentiated with, revealing the scope of application.

For the first variable, the necessary conditions to be fulfilled for a minimum are:

$$\begin{aligned} 0 = \partial_{u_{i+1,j}} F_1 &= 2\frac{u_{i+1,j} - u_{i,j}}{h_x} \cdot \frac{1}{h_x} \\ &\overset{k=i+1,l=j}{=} \frac{2}{h_x^2} \left(u_{k,l} - u_{k-1,l}\right) . \end{aligned} \tag{4.23a}$$

It is not hard to see that the scope of application of the above expression is $[2, N] \times [1, M]$. This is evident, as $u_{k-1,l}$ is not defined for $k = 1$. There is however no general method to see this, except for discretizing the functional directly. So far, we have the following part of the stencil (we divide by two on-the-fly):

0	0	0
$\chi_{[2,N]\times[1,M]}\frac{-1}{h_x^2}$	$\chi_{[2,N]\times[1,M]}\frac{1}{h_x^2}$	0
0	0	0

$\cdot\, u$

Now we differentiate with regard to the next variable:

$$\begin{aligned} 0 = \partial_{u_{i,j}} F_1 &= \frac{-2}{h_x^2}\left(u_{i+1,j} - u_{i,j}\right) + \frac{-2}{h_y^2}\left(u_{i,j+1} - u_{i,j}\right) \\ &\overset{k=i,l=j}{=} \underbrace{\frac{-2}{h_x^2}\left(u_{k+1,l} - u_{k,l}\right)}_{[1,N-1]\times[1,M]} + \underbrace{\frac{-2}{h_y^2}\left(u_{k,l+1} - u_{k,l}\right)}_{[1,N]\times[1,M-1]} . \end{aligned} \tag{4.23b}$$

The scope of application of the respective terms in the above expression is shown with the underbrace. Up to now, the stencil looks like

0	0	0
$\chi_{[2,N]\times[1,M]}\frac{-1}{h_x^2}$	$\chi_{[2,N]\times[1,M]}\frac{1}{h_x^2}$ $+\chi_{[1,N-1]\times[1,M]}\frac{1}{h_x^2}$ $+\chi_{[1,N]\times[1,M-1]}\frac{1}{h_y^2}$	$\chi_{[1,N-1]\times[1,M]}\frac{-1}{h_x^2}$ $\cdot u$
0	$\chi_{[1,N]\times[1,M-1]}\frac{-1}{h_y^2}$	0

The third and last differentiation is

$$\begin{aligned} 0 = \partial_{u_{i,j+1}} F_1 &= \frac{2}{h_y^2}\left(u_{i,j+1} - u_{i,j}\right) \\ &\overset{k=i,l=j+1}{=} \frac{2}{h_y^2}\left(u_{k,l} - u_{k,l-1}\right) , \end{aligned} \tag{4.23c}$$

with scope of application $[1, N] \times [2, M]$.

After division by two, this results in the following final stencil for the first order terms (inducing a second order diffusion process):

0	$\chi_{[1,N]\times[2,M]}\frac{-1}{h_y^2}$	0
$\chi_{[2,N]\times[1,M]}\frac{-1}{h_x^2}$	$\chi_{[1,N]\times[2,M]}\frac{1}{h_y^2}$ $+\chi_{[2,N]\times[1,M]}\frac{1}{h_x^2}$ $+\chi_{[1,N-1]\times[1,M]}\frac{1}{h_x^2}$ $+\chi_{[1,N]\times[1,M-1]}\frac{1}{h_y^2}$	$\chi_{[1,N-1]\times[1,M]}\frac{-1}{h_x^2}$ $\cdot u$
0	$\chi_{[1,N]\times[1,M-1]}\frac{-1}{h_y^2}$	0

which we call $S_{xx} + S_{yy}$.[1]

Minimization of Smoothness Term II

We proceed as before; the necessary conditions to be fulfilled for having a minimum here are:

$$\begin{aligned} 0 = \partial_{u_{i+1,j}} F_{2_1} &= 2\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} \cdot \frac{1}{h_x^2} \\ \Leftrightarrow 0 &= \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^4}\chi_{[3,N]\times[1,M]} , \end{aligned} \tag{4.24a}$$

[1] We do not show S_{xx} and S_{yy} separately.

$$\begin{aligned} 0 = \partial_{u_{i,j+1}} F_{2_1} &= 2\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} \cdot \frac{1}{h_y^2} \\ \Leftrightarrow 0 &= \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^4} \chi_{[1,N]\times[3,M]} \,, \end{aligned} \tag{4.24b}$$

$$\begin{aligned} 0 = \partial_{u_{i,j}} F_{2_1} &= 2\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} \cdot \frac{-2}{h_x^2} + 2\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} \cdot \frac{-2}{h_y^2} \\ \Leftrightarrow 0 = -2\Big(&\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^4} \chi_{[2,N-1]\times[1,M]} \\ &+ \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^4} \chi_{[1,N]\times[2,M-1]} \Big) \,, \end{aligned} \tag{4.24c}$$

$$\begin{aligned} 0 = \partial_{u_{i,j-1}} F_{2_1} &= 2\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} \cdot \frac{1}{h_y^2} \\ \Leftrightarrow 0 &= \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^4} \chi_{[1,N]\times[1,M-2]} \,, \text{and} \end{aligned} \tag{4.24d}$$

$$\begin{aligned} 0 = \partial_{u_{i-1,j}} F_{2_1} &= 2\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} \cdot \frac{1}{h_x^2} \\ \Leftrightarrow 0 &= \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^4} \chi_{[1,N-2]\times[1,M]} \,. \end{aligned} \tag{4.24e}$$

We have already divided by two, giving the stencils:

$S_{xxxx} =$

$\frac{1}{h_x^4}$	$\chi_{[3,N]\times[1,M]}$	$-2\chi_{[3,N]\times[1,M]}$ $-2\chi_{[2,N-1]\times[1,M]}$	$\chi_{[3,N]\times[1,M]}$ $+4\chi_{[2,N-1]\times[1,M]}$ $+\chi_{[1,N-2]\times[1,M]}$	$-2\chi_{[1,N-2]\times[1,M]}$ $-2\chi_{[2,N-1]\times[1,M]}$	$\chi_{[1,N-2]\times[1,M]}$

$\cdot u$

$$S_{yyyy} = \frac{1}{h_y^4} \begin{array}{|c|} \hline \chi_{[1,N]\times[3,M]} \\ \hline -2\chi_{[1,N]\times[3,M]} \\ -2\chi_{[1,N]\times[2,M-1]} \\ \hline \chi_{[1,N]\times[3,M]} \\ +4\chi_{[1,N]\times[2,M-1]} \\ +\chi_{[1,N]\times[1,M-2]} \\ \hline -2\chi_{[1,N]\times[2,M-1]} \\ -2\chi_{[1,N]\times[1,M-2]} \\ \hline \chi_{[1,N]\times[1,M-2]} \\ \hline \end{array} \cdot u$$

Component-wise addition yields the stencil that corresponds to (part of)

our desired fourth-order diffusion process:

$S_{xxxx} + S_{yyyy} =$

		$\chi_{[1,N]\times[3,M]}\frac{1}{h_y^4}$		
		$\chi_{[1,N]\times[3,M]}\frac{-2}{h_y^4}$ $+\chi_{[1,N]\times[2,M-1]}\frac{-2}{h_y^4}$		
$\chi_{[3,N]\times[1,M]}\frac{1}{h_x^4}$	$\chi_{[3,N]\times[1,M]}\frac{-2}{h_x^4}$ $+\chi_{[2,N-1]\times[1,M]}\frac{-2}{h_x^4}$	$\chi_{[3,N]\times[1,M]}\frac{1}{h_x^4}$ $+\chi_{[2,N-1]\times[1,M]}\frac{4}{h_x^4}$ $+\chi_{[1,N-2]\times[1,M]}\frac{1}{h_x^4}$ $+\chi_{[1,N]\times[3,M]}\frac{1}{h_y^4}$ $+\chi_{[1,N]\times[2,M-1]}\frac{4}{h_y^4}$ $+\chi_{[1,N]\times[1,M-2]}\frac{1}{h_y^4}$	$\chi_{[1,N-2]\times[1,M]}\frac{-2}{h_x^4}$ $+\chi_{[2,N-1]\times[1,M]}\frac{-2}{h_x^4}$	$\chi_{[1,N-2]\times[1,M]}\frac{1}{h_x^4}$
		$\chi_{[1,N]\times[2,M-1]}\frac{-2}{h_y^4}$ $+\chi_{[1,N]\times[1,M-2]}\frac{-2}{h_y^4}$		
		$\chi_{[1,N]\times[1,M-2]}\frac{1}{h_y^4}$		

$\cdot u$

For an example of how these conditions look at the boundaries, see Figure 4.3 for an example in x-direction only.

For the mixed derivatives, differentiation gives:

$$0 = \partial_{u_{i+1,j+1}} F_{2_2} = \frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{8h_x^2 h_y^2} \underset{=}{^{k=i+1,l=j+1}} \frac{u_{k,l} - u_{k-2,l} - u_{k,l-2} + u_{k-2,l-2}}{8h_x^2 h_y^2} \tag{4.25a}$$

with the scope of application $[3, N] \times [3, M]$,

$$0 = \partial_{u_{i-1,j+1}} F_{2_2} = -\frac{u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}}{8h_x^2 h_y^2} \underset{=}{^{k=i-1,l=j+1}} -\frac{u_{k+2,l} - u_{k+2,l-2} - u_{k,l} + u_{k,l-2}}{8h_x^2 h_y^2} \tag{4.25b}$$

with the scope of application $[1, N-2] \times [3, M]$,

$$0 = \partial_{u_{i+1,j-1}} F_{2_2} = -\frac{u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}}{8h_x^2 h_y^2} \underset{=}{^{k=i+1,l=j-1}} -\frac{u_{k,l+2} - u_{k,l} - u_{k-2,l+2} + u_{k-2,l}}{8h_x^2 h_y^2} \tag{4.25c}$$

with the scope of application $[3, N] \times [1, M-2]$, and

$$0 = \partial_{u_{i-1,j-1}} F_{2_2} = \frac{u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}}{8h_x^2 h_y^2} \underset{=}{^{k=i-1,l=j-1}} \frac{u_{k+2,l+2} - u_{k+2,l} - u_{k,l+2} + u_{k,l}}{8h_x^2 h_y^2} \tag{4.25d}$$

with the scope of application $[1, N-2] \times [1, M-2]$.

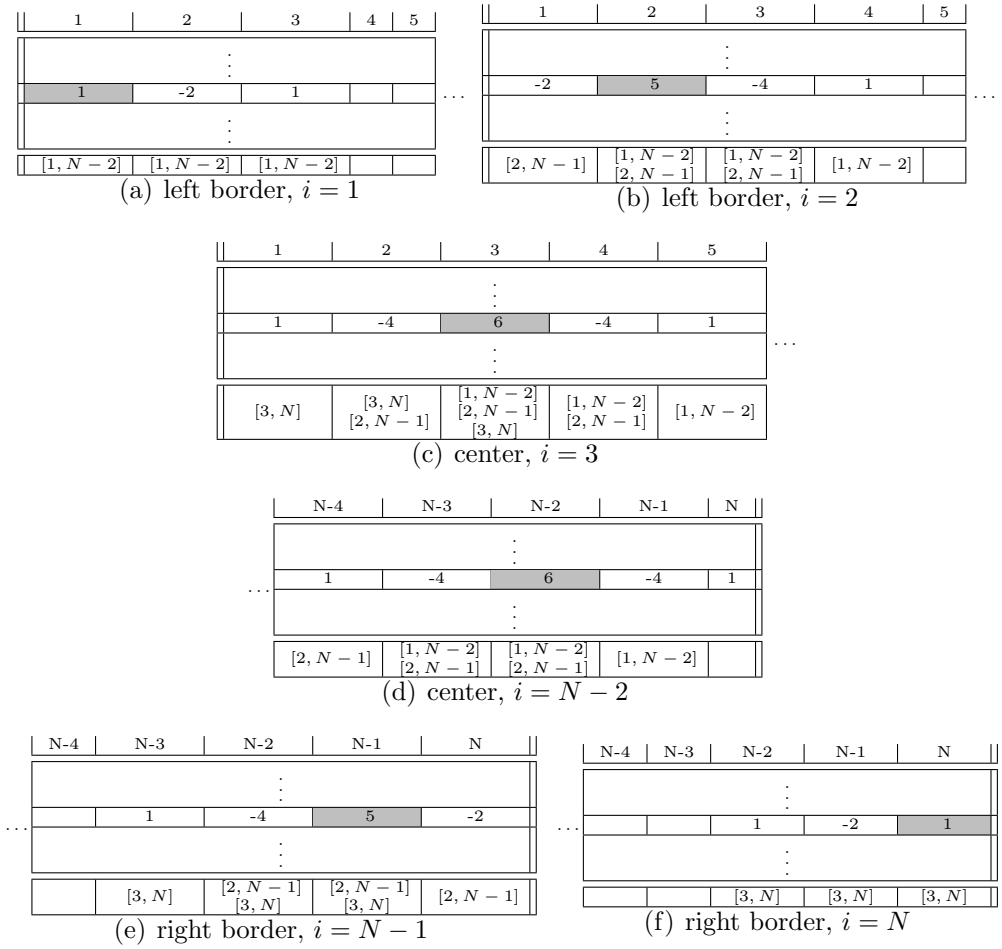


FIGURE 4.3: Illustration of the indicator function χ for the stencil S_{xxxx}. Each table represents part of an image/flow field with differently selected center pixels (location i, in gray). The x-range is numbered $[1, N]$ (top rows). The bottom row indicates the x-ranges of the active χs. (Their y-range is always $[1, M]$ in this example.)

This yields the stencil:

$$\partial_{xxyy}u \approx S_{xxyy} =$$

$$\frac{1}{8h_x^2h_y^2}\begin{array}{|c|c|c|c|c|}\hline \chi_{[3,N]\times[3,M]} & 0 & \begin{array}{c}-\chi_{[3,N]\times[3,M]}\\ -\chi_{[1,N-2]\times[3,M]}\end{array} & 0 & \chi_{[1,N-2]\times[3,M]} \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \begin{array}{c}-\chi_{[3,N]\times[3,M]}\\ -\chi_{[3,N]\times[1,M-2]}\end{array} & 0 & \begin{array}{c}\chi_{[3,N]\times[3,M]}\\ +\chi_{[1,N-2]\times[3,M]}\\ +\chi_{[3,N]\times[1,M-2]}\\ +\chi_{[1,N-2]\times[1,M-2]}\end{array} & 0 & \begin{array}{c}-\chi_{[1,N-2]\times[3,M]}\\ -\chi_{[1,N-2]\times[1,M-2]}\end{array} \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \chi_{[3,N]\times[1,M-2]} & 0 & \begin{array}{c}-\chi_{[3,N]\times[1,M-2]}\\ -\chi_{[1,N-2]\times[1,M-2]}\end{array} & 0 & \chi_{[1,N-2]\times[1,M-2]} \\ \hline\end{array} \cdot u \tag{4.26}$$

In stencil notation, we therefore have the necessary condition for a minimum

$$0 \stackrel{!}{=} J_{11_{i,j}}u_{i,j} + J_{13_{i,j}} + \alpha\left(S_{xx}+S_{yy}\right)_{i,j} + \beta\left(S_{xxxx}+2S_{xxyy}+S_{yyyy}\right)_{i,j}\ , \tag{4.27}$$

for $i \in [1,N], j \in [1,M]$. The complete stencil for the smoothness term (diffusion) is given in Figure 4.4.

As a matter of fact, this coincides perfectly with the Euler-Lagrange equation

$$0 = J_{11}u + J_{13} - \alpha\Delta u + \beta\Delta^2 u\ , \tag{4.28}$$

as our $(S_{xx}+S_{yy})$ is exactly what one would obtain discretizing $-\Delta u$ directly using standard discretizations. Δu is the trace of $\mathcal{H}(u)$ or the Laplacian of u; Δ^2, the square of the Laplacian, is also called the biharmonic operator and discretized via $S_{xxxx}+2S_{xxyy}+S_{yyyy}$.

Minimization of Alternative Discretization of Smoothness Term II

We have previously shown how the calculation works, therefore we just show the stencil for the alternative approximation of $\partial_{xxyy}u$ here:

$\partial_{xxyy}u \approx S_{xxyy} =$

$$\frac{1}{2h_x^2h_y^2}\begin{array}{|c|c|c|c|c|}\hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & \begin{array}{l}\chi_{[2,N]\times[2,M]} \\ +\chi_{[2,N-1]\times[2,M]} \\ +\chi_{[2,N]\times[2,M-1]} \\ +\chi_{[2,N-1]\times[2,M-1]}\end{array} & \begin{array}{l}-\chi_{[1,N-1]\times[2,M]} \\ -\chi_{[2,N]\times[2,M]} \\ -2\chi_{[2,N-1]\times[2,M]} \\ -\chi_{[1,N-1]\times[2,M-1]} \\ -\chi_{[2,N]\times[2,M-1]} \\ -2\chi_{[2,N-1]\times[2,M-1]}\end{array} & \begin{array}{l}\chi_{[1,N-1]\times[2,M]} \\ +\chi_{[2,N-1]\times[2,M]} \\ +\chi_{[1,N-1]\times[2,M-1]} \\ +\chi_{[2,N-1]\times[2,M-1]}\end{array} & 0 \\ \hline 0 & \begin{array}{l}-\chi_{[2,N]\times[1,M-1]} \\ -\chi_{[2,N]\times[2,M]} \\ -\chi_{[2,N-1]\times[2,M]} \\ -\chi_{[2,N-1]\times[1,M-1]} \\ -2\chi_{[2,N]\times[2,M-1]} \\ -2\chi_{[2,N-1]\times[2,M-1]}\end{array} & \begin{array}{l}\chi_{[1,N-1]\times[1,M-1]} \\ +\chi_{[2,N]\times[1,M-1]} \\ +2\chi_{[2,N-1]\times[1,M-1]} \\ +\chi_{[1,N-1]\times[2,M]} \\ +\chi_{[2,N]\times[2,M]} \\ +2\chi_{[2,N-1]\times[2,M]} \\ +2\chi_{[1,N-1]\times[2,M-1]} \\ +2\chi_{[2,N]\times[2,M-1]} \\ +4\chi_{[2,N-1]\times[2,M-1]}\end{array} & \begin{array}{l}-\chi_{[1,N-1]\times[1,M-1]} \\ -\chi_{[2,N-1]\times[1,M-1]} \\ -\chi_{[1,N-1]\times[2,M]} \\ -\chi_{[2,N-1]\times[2,M]} \\ -2\chi_{[1,N-1]\times[2,M-1]} \\ -2\chi_{[2,N-1]\times[2,M-1]}\end{array} & 0 \\ \hline 0 & \begin{array}{l}\chi_{[2,N]\times[1,M-1]} \\ +\chi_{[2,N-1]\times[1,M-1]} \\ +\chi_{[2,N]\times[2,M-1]} \\ +\chi_{[2,N-1]\times[2,M-1]}\end{array} & \begin{array}{l}-\chi_{[1,N-1]\times[1,M-1]} \\ -\chi_{[2,N]\times[1,M-1]} \\ -2\chi_{[2,N-1]\times[1,M-1]} \\ -\chi_{[1,N-1]\times[2,M-1]} \\ -\chi_{[2,N]\times[2,M-1]} \\ -2\chi_{[2,N-1]\times[2,M-1]}\end{array} & \begin{array}{l}\chi_{[1,N-1]\times[1,M-1]} \\ +\chi_{[2,N-1]\times[1,M-1]} \\ +\chi_{[1,N-1]\times[2,M-1]} \\ +\chi_{[2,N-1]\times[2,M-1]}\end{array} & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array} \cdot u \tag{4.29}$$

4.3 Solving

With a 5×5 neighborhood (stencil), this leads to a system of equations with 25 equations and 25 unknowns, that is, a sparse matrix with 25 diagonals which are however "grouped" into blocks of 5 diagonals. These diagonals contain all the interaction with the corresponding neighboring pixels.

The elements of the matrix are now necessarily ordered in a linear fashion, i.e. no longer indexed two-dimensionally with $u_{i,j}$ but as $u_{o(i,j)}$ via a function $o(i,j) = ((N-1)\cdot j) + i$. Therefore, in *each row* of the system matrix one can see the stencil applied. The diagonals that are far away from the main diagonal in the system matrix contain the neighboring pixels from another row (but the same column) of the signal/image, while the diagonals that are adjacent to the center diagonal represent the neighbors from the current row (but the other columns). An exemplary system matrix $\mathbf{A}$ clearly showing

horizontal and vertical interactions for a 4×2 image looks like

$$\underbrace{\begin{bmatrix} c_0 & x_0^+ & 0 & 0 & y_0^+ & 0 & 0 & 0 \\ x_1^- & c_1 & x_1^+ & 0 & 0 & y_1^+ & 0 & 0 \\ 0 & x_2^- & c_2 & x_2^+ & 0 & 0 & y_2^+ & 0 \\ 0 & 0 & x_3^- & c_3 & 0 & 0 & 0 & y_3^+ \\ y_4^- & 0 & 0 & 0 & c_4 & x_4^+ & 0 & 0 \\ 0 & y_5^- & 0 & 0 & x_5^- & c_5 & x_5^+ & 0 \\ 0 & 0 & y_6^- & 0 & 0 & x_6^- & c_6 & x_6^+ \\ 0 & 0 & 0 & y_7^- & 0 & 0 & x_7^- & c_7 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} . \tag{4.30}$$

In it we denote by c_i the center weights for the corresponding stencil application (centered at u_i) and by x_i^+, x_i^-, y_i^+ and y_i^- their right, left, lower and upper neighbor weights, respectively. With the previously mentioned linear ordering, the image would be encoded in a structure like

$u_{o(0,0)} = u_0$	$u_{o(1,0)} = u_1$	$u_{o(2,0)} = u_2$	$u_{o(3,0)} = u_3$
$u_{o(0,1)} = u_4$	$u_{o(1,1)} = u_5$	$u_{o(2,1)} = u_6$	$u_{o(3,1)} = u_7$

.

The stencil with its corresponding weights at element u_3 (upper-right corner of image) would thus look like

0	0	0
x_3^-	c_3	0
0	y_3^+	0

$\cdot\, u_3$.

An exemplary structure of the system of equations $\mathbf{Ax} = \mathbf{b}$ is

$$\underbrace{\left(\underbrace{\begin{bmatrix} J_{11} & & & & \\ & J_{11} & & & \\ & & J_{11} & & \\ & & & J_{11} & \\ & & & & J_{11} \end{bmatrix}}_{\text{data term}} - \alpha \underbrace{\begin{bmatrix} -2 & 1 & & 1 & \\ 1 & -3 & 1 & & 1 \\ & 1 & -2 & 1 & \\ 1 & & 1 & -3 & 1 \\ & 1 & & 1 & -2 \end{bmatrix}}_{\text{smoothness term 1}} + \beta \underbrace{\begin{bmatrix} 17 & -16 & 3 & & -4 \\ -16 & 30 & -16 & 3 & \\ 3 & -16 & 27 & -16 & 3 \\ & 3 & -16 & 30 & -16 \\ -4 & & 3 & -16 & 17 \end{bmatrix}}_{\text{smoothness term 2}} \right)}_{\text{system matrix } \mathbf{A}} \underbrace{\begin{bmatrix} u \\ u \\ u \\ u \\ u \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} -J_{13} \\ -J_{13} \\ -J_{13} \\ -J_{13} \\ -J_{13} \end{bmatrix}}_{\mathbf{b}} . \tag{4.31}$$

In Section 2.1.2 we had introduced the SOR method for solving linear systems of equations and previously in this chapter we have seen the warping strategy. This is where both are used in the implementation.

	$i-2$	$i-1$	i	$i+1$	$i+2$
$j-2$	$\beta\Big(\chi_{[3,N]\times[3,M]}(\frac{1}{4h_x^2h_y^2})\Big)$	0	$\beta\Big(\chi_{[1,N]\times[3,M]}(\frac{1}{h_y^4}) + \chi_{[3,N]\times[3,M]}(\frac{-1}{4h_x^2h_y^2}) + \chi_{[1,N-2]\times[3,M]}(\frac{-1}{4h_x^2h_y^2})\Big)$	0	$\beta\Big(\chi_{[1,N-2]\times[3,M]}(\frac{1}{4h_x^2h_y^2})\Big)$
$j-1$	0	0	$\alpha\Big(\chi_{[1,N]\times[2,M]}(\frac{1}{h_y^2})\Big) + \beta\Big(\chi_{[1,N]\times[3,M]}(\frac{-2}{h_y^4}) + \chi_{[1,N]\times[2,M-1]}(\frac{-2}{h_y^4})\Big)$	0	0
j	$\beta\Big(\chi_{[3,N]\times[1,M]}(\frac{1}{h_x^4}) + \chi_{[3,N]\times[3,M]}(\frac{-1}{4h_x^2h_y^2}) + \chi_{[3,N]\times[1,M-2]}(\frac{-1}{4h_x^2h_y^2})\Big)$	$\alpha\Big(\chi_{[2,N]\times[1,M]}(\frac{1}{h_x^2})\Big) + \beta\Big(\chi_{[3,N]\times[1,M]}(\frac{-2}{h_x^4}) + \chi_{[2,N-1]\times[1,M]}(\frac{-2}{h_x^4})\Big)$	$\alpha\Big(\chi_{[1,N-1]\times[1,M]}(\frac{-1}{h_x^2}) + \chi_{[2,N]\times[1,M]}(\frac{-1}{h_x^2}) + \chi_{[1,N]\times[1,M-1]}(\frac{-1}{h_y^2}) + \chi_{[1,N]\times[2,M]}(\frac{-1}{h_y^2})\Big) + \beta\Big(\chi_{[3,N]\times[1,M]}(\frac{1}{h_x^4}) + \chi_{[2,N-1]\times[1,M]}(\frac{4}{h_x^4}) + \chi_{[1,N-2]\times[1,M]}(\frac{1}{h_x^4}) + \chi_{[1,N]\times[3,M]}(\frac{1}{h_y^4}) + \chi_{[1,N]\times[2,M-1]}(\frac{4}{h_y^4}) + \chi_{[1,N]\times[1,M-2]}(\frac{1}{h_y^4}) + \chi_{[3,N]\times[3,M]}(\frac{1}{4h_x^2h_y^2}) + \chi_{[1,N-2]\times[1,M-2]}(\frac{1}{4h_x^2h_y^2}) + \chi_{[3,N]\times[1,M-2]}(\frac{1}{4h_x^2h_y^2}) + \chi_{[1,N-2]\times[3,M]}(\frac{1}{4h_x^2h_y^2})\Big)$	$\alpha\Big(\chi_{[1,N-1]\times[1,M]}(\frac{1}{h_x^2})\Big) + \beta\Big(\chi_{[1,N-2]\times[3,M]}(\frac{-2}{h_x^4}) + \chi_{[2,N-1]\times[1,M]}(\frac{-2}{h_x^4})\Big)$	$\beta\Big(\chi_{[1,N-2]\times[1,M]}(\frac{1}{h_x^4}) + \chi_{[1,N-2]\times[1,M-2]}(\frac{-1}{4h_x^2h_y^2}) + \chi_{[1,N-2]\times[3,M]}(\frac{-1}{4h_x^2h_y^2})\Big)$
$j+1$	0	0	$\alpha\Big(\chi_{[1,N]\times[1,M-1]}(\frac{1}{h_y^2})\Big) + \beta\Big(\chi_{[1,N]\times[2,M-1]}(\frac{-2}{h_y^4}) + \chi_{[1,N]\times[1,M-2]}(\frac{-2}{h_y^4})\Big)$	0	0
$j+2$	$\beta\Big(\chi_{[3,N]\times[1,M-2]}(\frac{1}{4h_x^2h_y^2})\Big)$	0	$\beta\Big(\chi_{[1,N]\times[1,M-2]}(\frac{1}{h_y^4}) + \chi_{[1,N-2]\times[1,M-2]}(\frac{-1}{4h_x^2h_y^2}) + \chi_{[3,N]\times[1,M-2]}(\frac{-1}{4h_x^2h_y^2})\Big)$	0	$\beta\Big(\chi_{[1,N-2]\times[1,M-2]}(\frac{1}{4h_x^2h_y^2})\Big)$

$\cdot u$

FIGURE 4.4: Stencil for the discretization of the fourth-order filter in (4.28) with complete boundary conditions.

Chapter 5

Experimental Results

In this chapter we present quantitative and qualitative results of the local method from Chapter 3, the standard Horn/Schunck global method, and the variational method from Chapter 4 and its two discretizations.

5.1 Error Measures

For local methods there are several popular methods of indicating some kind of mean computed disparity error. First, there are the average absolute (AA) disparity error and the root mean squared (RMS) error, both in disparity units between the ground truth map d_T and the computed disparity map d_C:

$$e_{\mathrm{AA}}(d_C, d_T) = \frac{1}{|\Omega_2|} \sum_{(x,y)\in\Omega_2} \left| d_C(x,y) - d_T(x,y) \right| , \tag{5.1a}$$

$$e_{\mathrm{RMS}}(d_C, d_T) = \sqrt{\frac{1}{|\Omega_2|} \sum_{(x,y)\in\Omega_2} \left(d_C(x,y) - d_T(x,y) \right)^2} , \tag{5.1b}$$

where Ω_2 denotes the two-dimensional image domain and $|\Omega_2|$ consequently the number of pixels in the image.

The RMS error by design penalizes large differences between the real and the estimated data more than smaller ones, which are even attenuated (if they are smaller than 1). One can argue that this represents a better subjective error measure, as very small errors tend to go unnoticed by humans. The final display of the disparity map as an image forcibly happens with integer precision, therefore one can further argue that such small nuances do not matter anyway, which is why we stick to the AA disparity error (AADE).

For optic flow techniques the error measure of choice is the average angular error (AAE) between the true and the computed displacement vector field [BFB94]. It is an error expressed as angle because optic flow is a generalization of the stereo correspondence problem, in which correspondences can be arbitrary; in stereo they are not, remember the epipolar constraint (Section

2.2). The average difference of the angle between the vectors in the real and in the computed vector field gives the AAE and is expressed in degrees.

In our case however, we know beforehand that we will obtain horizontal flow fields only, therefore we can encode the length of the corresponding vector $u(x, y, t)$ directly as a gray scale image, and use the AA or RMS error instead (after properly scaling). This also has the advantage that our measurements for local and global method are directly comparable.

Another popular error measure is the number of bad matching pixels (BP) [SZ00]

$$e_{\mathrm{BP}}(d_C, d_T) = \frac{100}{|\Omega_2|} \sum_{(x,y)\in\Omega_2} \left(\left| d_C(x, y) - d_T(x, y) \right| > \delta_d \right). \tag{5.2}$$

The part inside the sum is read as a comparison and as such has a boolean result: if the operator $>$ returns "false" we interpret this as the integer 0 and if it returns "true" we interpret it as the integer value 1. We sum up these results and finally display them as the percentage of pixels that are off by more than a certain threshold over the entire region under consideration. For the threshold, we use $\delta_d = 1.0$.

It is important to note that, as indicated above, we do not simply compare the result images that you see in this section to their ground truth images, but we compare (floating point) disparities, as many rounding errors and imprecisions would otherwise be introduced. We use the ground truth images and their known normalization factor used to produce them in order to obtain a floating point disparity. We then compare this value directly to the floating point disparity resulting from our computations.

Additionally we usually do not consider a certain region at the border. Which region that is exactly for the individual dataset and method is mentioned in its context. In the ground truth displayed, such regions are completely black.

5.2 Parameters and Implementation Details

In this section, we show the parameters used for the upcoming experiments and bring up some details about the implementation that are worth mentioning. Some basic data about the scenes used for evaluation can later be found in Tables 5.1 and 5.2 for the local and global method, respectively.

5.2.1 Local Method

Window size

The choice of the window size involves a trade-off between a noisy disparity map and blurring of depth boundaries, which occurs when the support region

(window) spans depth/object boundaries. On the other hand, detailed features in a scene are usually only recognized using a small window. A perfect example for this is the lamp stem in the Tsukuba sequence.

Candidate matches and sorting

It is important to understand that starting from the initial plain SSD fit computation, for *each* pixel in the left image, there are *several* possible matching locations (candidate matches) in the right image that are kept in memory. This is done because region matching like SSD is inherently error-prone: the best fit value need not necessarily be the right match. It is however assumed that one of the best fits is indeed the correct correspondence (at least it is assumed to be in the vicinity of the correct match), which the deformed window SSD will then detect at a later step.

Storing the top n matches for one pixel is not as trivial as it might seem because we have the additional constraint that two good matches must be a certain number of disparity units apart (in order to avoid local minima). In that case, simply sorting the values by best fit, i.e. checking the epipolar line pixel by pixel and checking whether the new fit is better than the previous fit will fail. Consider the following matches along the epipolar line and let us assume that the next (best) fit stored must be at least two positions away from the current best fit:

disparity	1	2	3	4	5	6
fit	5	4	3	2	4	5

We start with the first cell containing the value 5. As it is the first value seen, it automatically is the best value seen so far (green color). The next cell contains the value 4: it is better than 5 (here lower is better, like in SSD), but it is directly adjacent to 5, therefore it is ignored (red color). In the next cell, the value 3 is stored as another best fit etc. Finally, the array storing the top three fits would look like

5	3	X

where X denotes an empty cell. Obviously, we have not obtained the actual best fit, which would have been 2 at disparity four (red background). To tackle this problem algorithmically, one has to store fit-disparity tuples as follows:

1. compute all SSD/NXC fit values for one image pixel (up to a certain maximum disparity), remember (f, d)-tuples where f is the fit value and d the associated disparity,

2. sort the tuples by fit value from best to worst (that is, increasing for SSD / decreasing for NXC); if two tuples have the same fit value, we select the "better" one as the one with the lower disparity,

3. remember the final best n fits:

   (a) add the best tuple to the final array of candidate matches,

   (b) check the next best unseen tuple, and

      ▷ add it to the final array, if it passes the check that the disparity is at least δ pixels apart from each currently stored final tuples, or

      ▷ ignore it, if this is not the case.

   (c) Repeat the last step until we have found n tuples or none are left.

(We choose $n = 3$ and $\delta = 2$, that is $|x_2 - x_1| \geq \delta$).

Considering our previous example, the resulting array of best fit-disparity tuples would be

slot	1	2	3
disparity	4	2	6
fit	2	4	5

The value 4 at disparity five is not added because it is too close to the best value at disparity four. The value 5 at disparity one is not added because it is too close to the value 4 at disparity two.

Note that in our implementation the maximum number of fits kept is set to three[1]. If there are no more free slots to accommodate new values, we overwrite existing values by better ones, but evidently only if the new value is better than the worst one currently stored (the worst value will be replaced).

Restriction of search space

Not only do we remember several matches, we also try to obtain dense and more accurate disparity maps by just accepting the lowest fit value(s) within the manually defined maximum disparity as given by a human observer (we do *not* search the entire epipolar line).

Additionally, in an orthoparallel setup, it suffices to search from right to left in the right image, starting at the x coordinate of the original point in the left image. This becomes evident because in the right image, the valid match must be on the left-hand side of the original coordinate, as *all* objects have moved to the left in the right image, compared to the left image. (Obviously, we must not confound left and right images for this to work, as in the general case we do not talk about "left" or "right" images, but about image 1 and 2.)

[1] otherwise the computational burden for all the subsequent steps would be too high

Deformed Window SSD (dwSSD) and Normalized Cross Correlation (NXC)

Although, as the name indicates, inside the deformed window an SSD is computed, one can however choose the value with which to initialize this minimization problem. It might be easier for the dwSSD to find the actual minimum if the initialization is more correct. That is why we use NXC with dwSSD as an experiment — this is not a mistake.

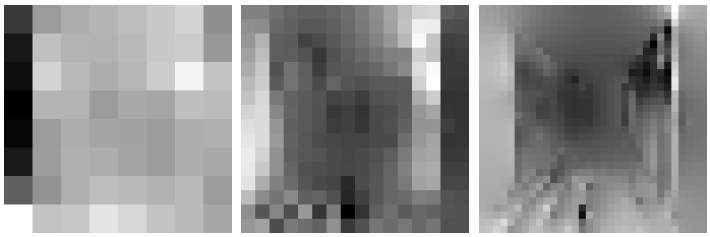
5.2.2 Global Method

The second order regularization term added to the Horn/Schunck method was chosen with the idea in mind to have something similar to slanted surfaces detection in the local method that we presented previously.

Model and Numerical Parameters

α indicates the regularization parameter for the first order and β for the higher order (HO) smoothness term, exactly as they are used in the theoretical section.

The warping parameter η indicates how much the resolution is changed from a fine to a coarser level, i.e. with $\eta = 0.5$ one would have half the resolution of the current level at the next level (see Figure 5.1). The convergence factor ω for the SOR method remains fixed at 1.950.



(a) coarse ... (b) ... to ... (c) ... fine

FIGURE 5.1: Coarse and fine images for $\eta = 0.5$.

Additionally, if a correspondence is found outside of the image (since the image size is known), the software ignores the data term and just uses the smoothness term (i.e. fills in from the neighborhood). 200 iterations are enough in our cases for the filling-in to produce qualitatively good results.

Scaling of HO Smoothness Regularization Parameter

Using higher-order smoothness terms together with the warping strategy requires careful adjustment of the regularization parameter (here β) at the individual warping levels. While the first order smoothness assumption divides by terms of the order of h^2, the second order smoothness assumption divides by terms of the order of h^4, i.e. compared to each other the latter factors become smaller by an order of two compared to the former ones from level to level. In order to counteract that effect we scale the regularization parameter β by $\frac{1}{\eta^2}$ at each level (making it larger, as $0 < \eta < 1$).

window size	Corridor		Tsukuba	
	(9×9)	(21×21)	(9×9)	(21×21)
Images resolution	256×256		384×288	
ground truth resolution	256×256		348×252	
disp. map resolution	248×248	236×236	376×280	364×268
considered region	236×236	236×236	348×252	348×252
# of pixels (all)	55696		87696	
# of pixels (non-occl.)	54768 (98.3%)		85438 (97.4%)	
disp. map scale factor	21.25		16	

TABLE 5.1: Parameters of Local Method

	Corridor	Tsukuba
Images resolution	256×256	384×288
ground truth resolution	256×256	348×252
disp. map resolution	256×256	384×288
considered region	236×236	348×252
SOR iterations	200	
warping levels	10	
η	0.8	
disp. map scale factor	21.25	16

TABLE 5.2: Parameters of Global Method

5.3 Evaluation

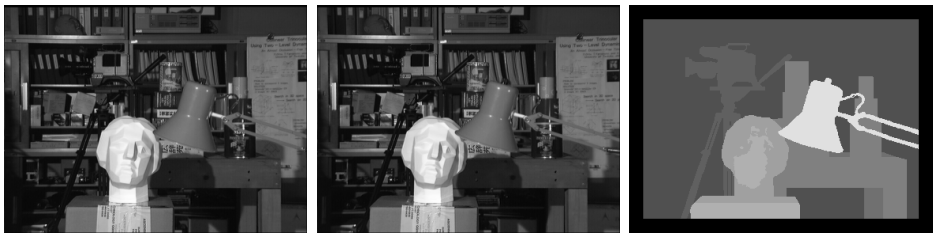
The scenes we use are the University of Bonn Corridor Sequence (Figure 5.2), a synthetic sequence with lots of slanted surfaces, and two frames of

the University of Tsukuba stereo sequence (Figure 5.3), containing complex objects at different depths generating several occlusions, as well as poorly-textured regions in the background.

To be able to provide fair results between the local and global method, it is especially necessary to compare the very same regions of the resulting disparity maps to its ground truth. For the local method, we have the problem with the Corridor Sequence, that using a larger window size decreases the size of the resulting disparity map. We thus compare for both local and global methods only the regions where both methods always provide results. For Tsukuba, there is no such problem, as the available ground truth is the limiting factor here. We have adequately mentioned this in Tables 5.1 and 5.2 as the "considered region".

(a) left camera (b) right camera (c) ground truth

FIGURE 5.2: University of Bonn Corridor Sequence (256×256 px, disparity range 11).



(a) left camera (b) right camera (c) ground truth

FIGURE 5.3: University of Tsukuba stereo sequence (384×288 px, disparity range 14).

Note that we have not applied any pre-smoothing to the images in either method ($\sigma = 0$). "px" stands for the unit pixels.

5.3.1 Local Method

We will use several abbreviations for the individual elements in the local method, we now summarize them:

- ▷ **SSD/NXC**: sum of squared differences/normalized cross correlation: region-based matching metric; first seen in Section 3.1,
- ▷ **IWS**: initial window size: window size for the initial region matching method (SSD/NXC),
- ▷ **dwSSD**: deformed window SSD: region based matching with deformed window (this window size is always 9×9, except for one experiment); first seen in Section 3.2,
- ▷ **GSID**: geometric support space iterative diffusion; first seen in Section 3.4,
- ▷ **GSR**: geometric support region: 3-D (x,y,d) support region (neighborhood) for GSID; first seen in Section 3.4.

The computation of the Corridor Sequence (256×256 pixels) takes approximately 35 minutes on an AMD Athlon64 3700+ CPU (2200 MHz) with 1 MB L2-Cache. Most of the time is spent on the minimization problem for dwSSD (248×248 pixels of information for a 9×9 SSD window size and a 9×9 window size for dwSSD), with about 0.01 second per fit and three fits per pixel, dwSSD takes about 30 minutes. The remaining five minutes are spent on the diffusion in the geometric support space (GSID) with geometric support region (GSR) $5 \times 5 \times 3$. Increasing the GSR to $21 \times 21 \times 7$ increases the running time for this part of the computation from five to at least twenty minutes. This is expected, because in the worst case we have about 40 times more points to consider.

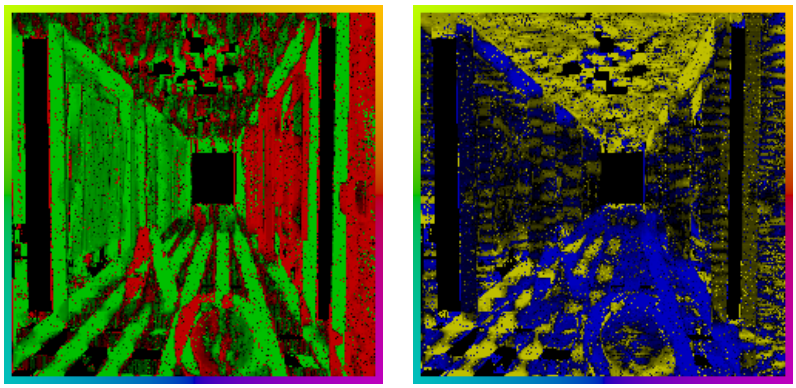
As one can see, for the Corridor Sequence, SSD performs slightly better than NXC, after only a few iterations, but only for the initialization with a large IWS (Figure 5.12). This agrees with the intuition that for scenes with many (and/or large) homogeneous areas, larger windows perform better than smaller ones.

For the Tsukuba sequence, one would first think using a smaller IWS was a more correct approach in order to detect the minute details in the scene. Furthermore, there is varying illumination in the scene, so NXC should do better. This reasoning indeed seems to be correct: the best result is obtained using NXC with a 9×9 window for initial region matching.

For both scenes, a large GSR of $21 \times 21 \times 7$ proves useful, in order to aggregate the most information from the neighborhood.

To get an idea, Figure 5.4 shows what disparity derivatives have been determined for the Corridor Sequence. Green, red, yellow and blue areas indicate negative and positive $\partial_x d$, and negative and positive $\partial_y d$, respectively.

The results agree with our intuitions from Section 3.3, especially Table 3.2, and we can also see that discontinuities and the checkerboard floor pattern pose problems here.



(a) $\partial_x d$ (b) $\partial_y d$

FIGURE 5.4: Corridor Sequence First Order Disparities

5.3.2 Global Method

We can see that using the HO (higher horder) regularizer performs especially well with the Corridor sequence, since it contains many slanted surfaces. On the other hand, it also seems to introduce some visually unappealing artifacts at the end of the corridor in the homogeneous region. Looking at the error measures however proves its superiority.

The Tsukuba sequence also does well with the HO regularizer, however since slanted surfaces are less present here, we must conclude that simply the use of higher order regularization allows for a better disparity map estimation. The best result was obtained using a little of the first order regularizer too, which we determined empirically.

As far as the two discretizations for the mixed term are concerned, one can already conclude that the standard central differences obtained by taking the square of the average of backward/forward differences (4.19) dominates taking the average of the square of backward/forward differences (4.21). This could be due to the first one (Stencil 4.26) considering pixels further away from the center than the second one (Stencil 4.29), and therefore being more robust.

5.3.3 And the Winner is ...

In the local method, it is sometimes unclear when to stop the geometric support space iterative diffusion (GSID), as one can see in the case of 21×21

SSD with $21 \times 21 \times 7$ GSR for the corridor sequence. It reaches the unbeaten value[2] of 0.313 AA disparity error units (for the non-occluded regions) and starts to slightly worsen from there. So, if one were to stop when the results are getting worse again, one would choose this method. However, the real winner is the very similar 21×21 NXC with $21 \times 21 \times 7$ GSR, which continues to work towards always better values, even after the 24th iteration (see Table 5.3), with 0.319 AADE units. The best value obtained for the corridor with the global method is the very close 0.331 AADE units.

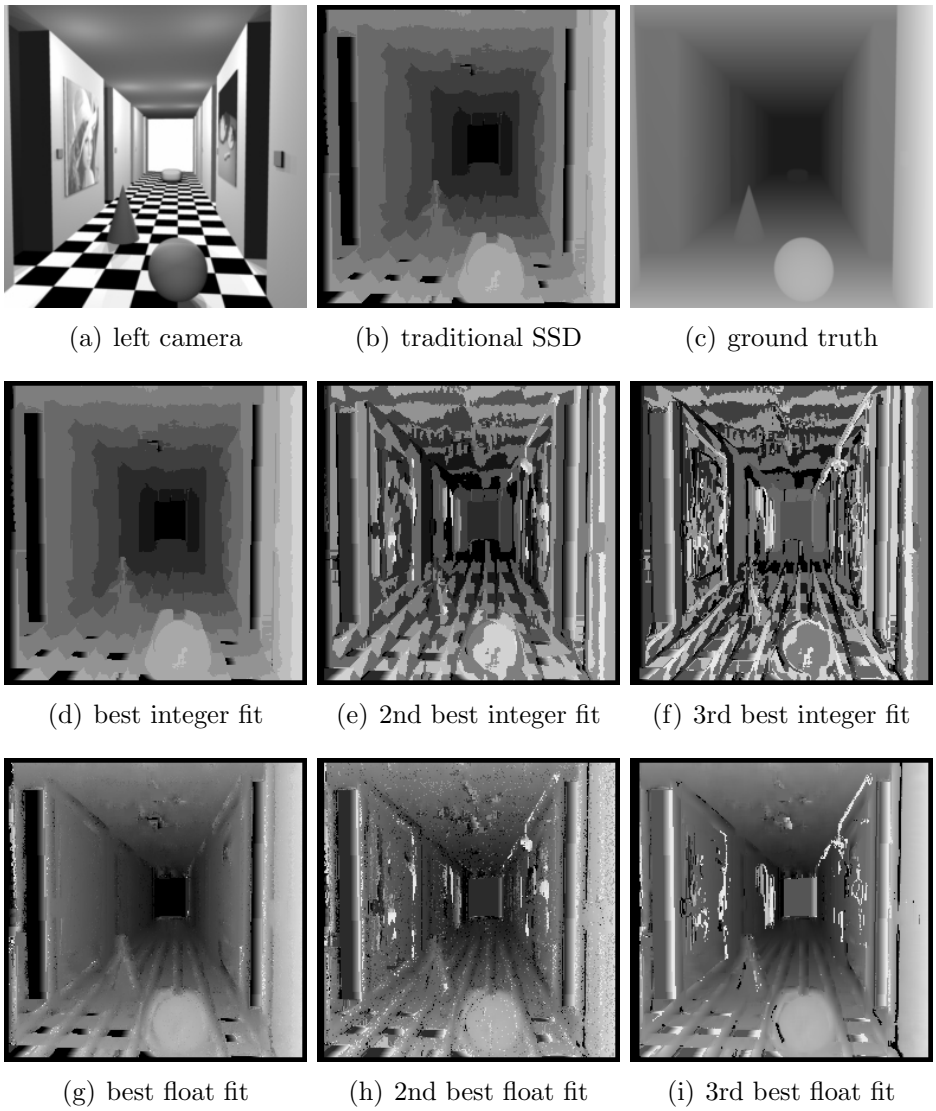
As far as the Tsukuba sequence is concerned, the global method prevails with a best result of 0.539 AADE units (non-occluded). The best result of the local method in this case is the 9×9 SSD with $21 \times 21 \times 7$ GSR with 0.597 AADE units. The best BP error measure for Tsukuba is however obtained with 9×9 NXC with $5 \times 5 \times 3$ GSR with 11.909% (see Table 5.4), which is a sign that the latter method produces more small errors, and the former larger ones.

The intuition that, for scenes with lots of detail (like Tsukuba), a smaller IWS is better, is confirmed (up to 0.25 AADE and 0.8 BP units better than large window for the same method and same GSR); vice versa, for scenes with less detail (like the Corridor), using a larger IWS runs in no danger of blurring the non-existing details/discontinuities and additionally produces more exact results (ca. 0.1-1 AADE and 1-5 BP units better than smaller window for the same method and same GSR).

One can also conclude for the local method that a larger GSR is always better, as it allows to aggregate more information from the neighborhood. It is however not wise to use large windows for everything, including dwSSD, as the experiment with 21×21 NXC with $21 \times 21 \times 7$ GSR and 21×21 dwSSD for Tsukuba has shown; too many details are blurred and this experiment has the worst error statistics for this scene.

We will now show results as pictures with short descriptions, followed by charts and tables with the quantitative data.

[2] as far as our experiments are concerned



(a) left camera (b) traditional SSD (c) ground truth

(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

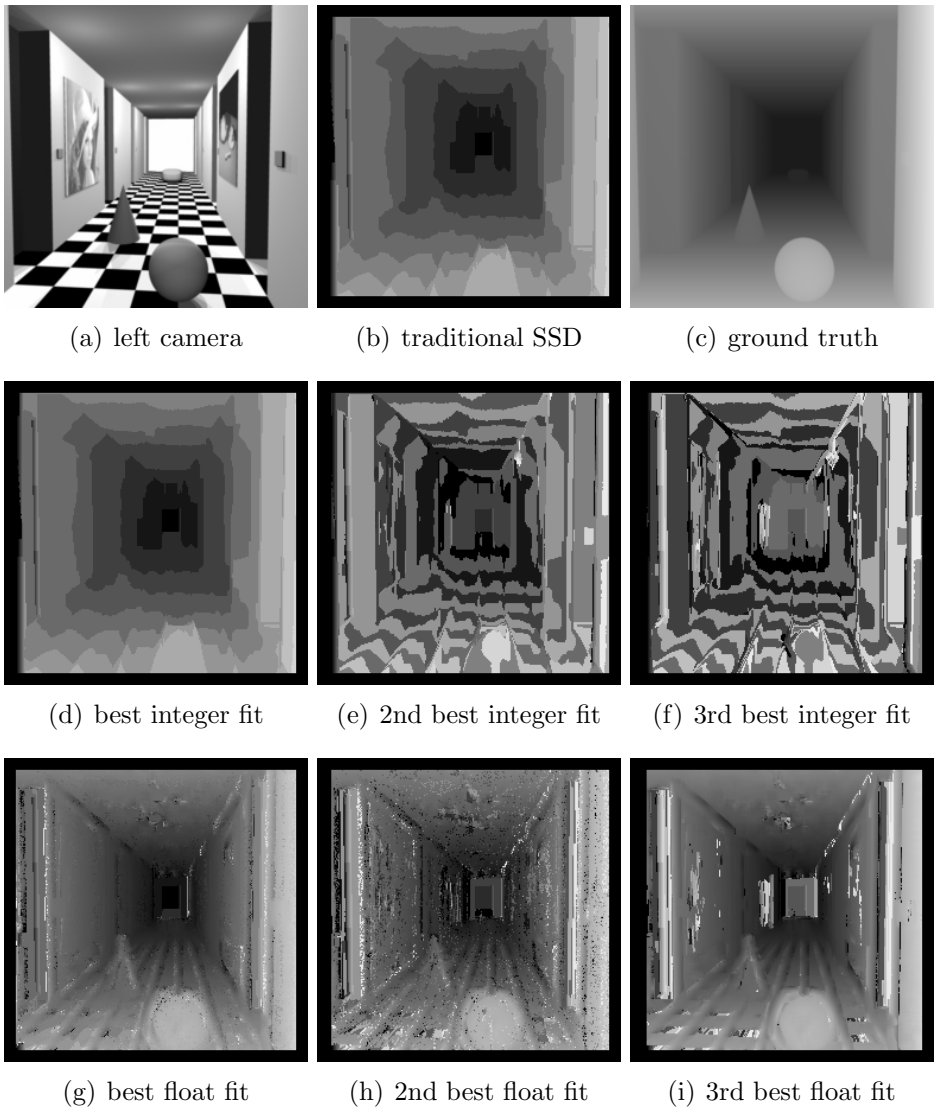
(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

FIGURE 5.5: Corridor disparity maps with 9×9 SSD window before and after dwSSD.

Figure 5.5 shows data for 9×9 SSD. In the second row, we show the three best fits obtained via traditional SSD, that is used to initialize the dwSSD, whose results are displayed in the third row.

Obviously, e.g. Figure 5.5(d) is the same as Figure 5.5(b) as the best match is always the one returned when we choose to have one fit per pixel only.

In this and all following experiments one can see that interpolated floating point disparities clean up the results a lot. The pictures on the left and right walls as well as discontinuities can pose matching problems, here at least for the second and third best fits. Geometric support space diffusion (GSID) will clean this up for the final result.



(a) left camera (b) traditional SSD (c) ground truth

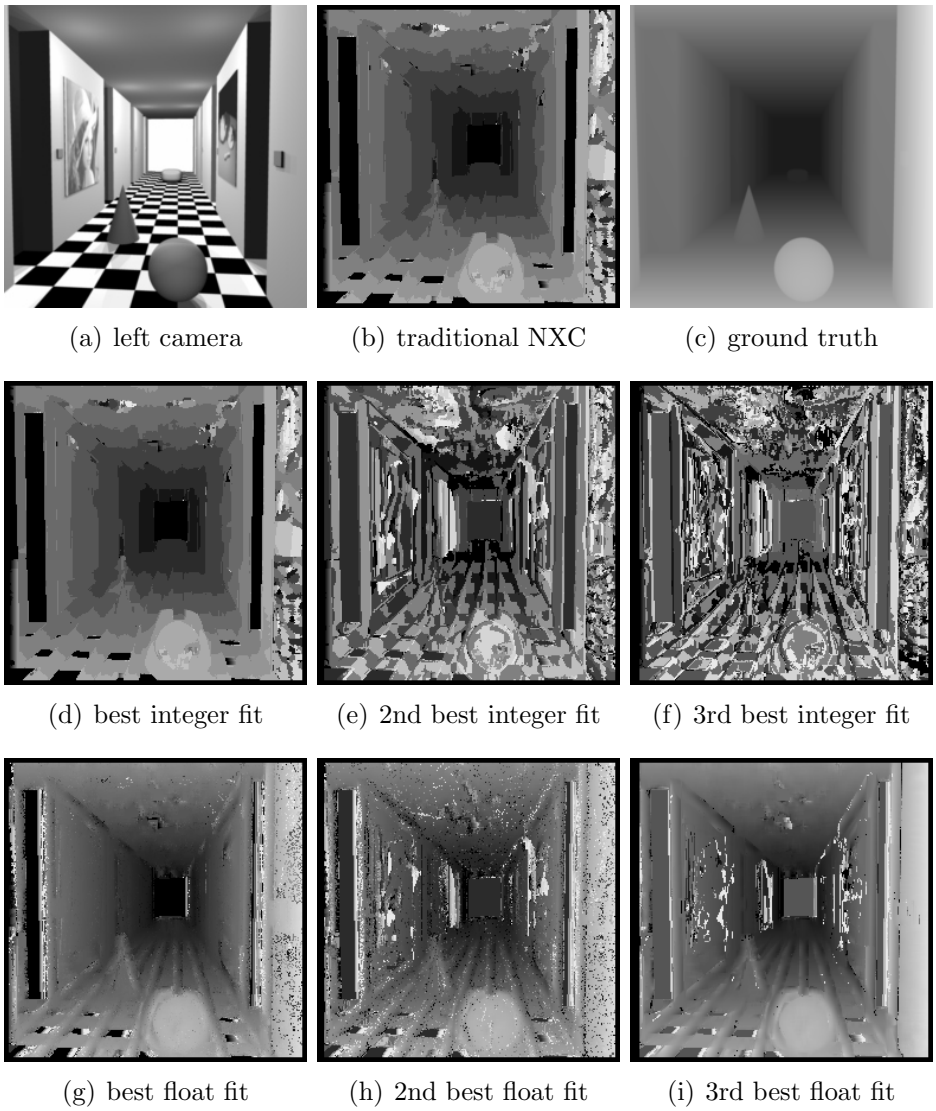
(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

FIGURE 5.6: Corridor disparity maps with 21×21 SSD window before and after dwSSD.

Figure 5.6 shows data obtained using 21×21 SSD. In the second row, we show the three best fits obtained via traditional SSD, that is used to initialize the dwSSD (always a 9×9 window), whose results are displayed in the third row.

In this experiment, as in the previous and following experiments with local region matching, one can see that the homogeneous area at the back of the corridor poses problems to local methods. In the worst case, a seemingly valid disparity could be the distance in pixels from one side of the door at the end of the corridor to the other, which however we avoid by selecting the lowest disparity for the same fit value.



(a) left camera (b) traditional NXC (c) ground truth

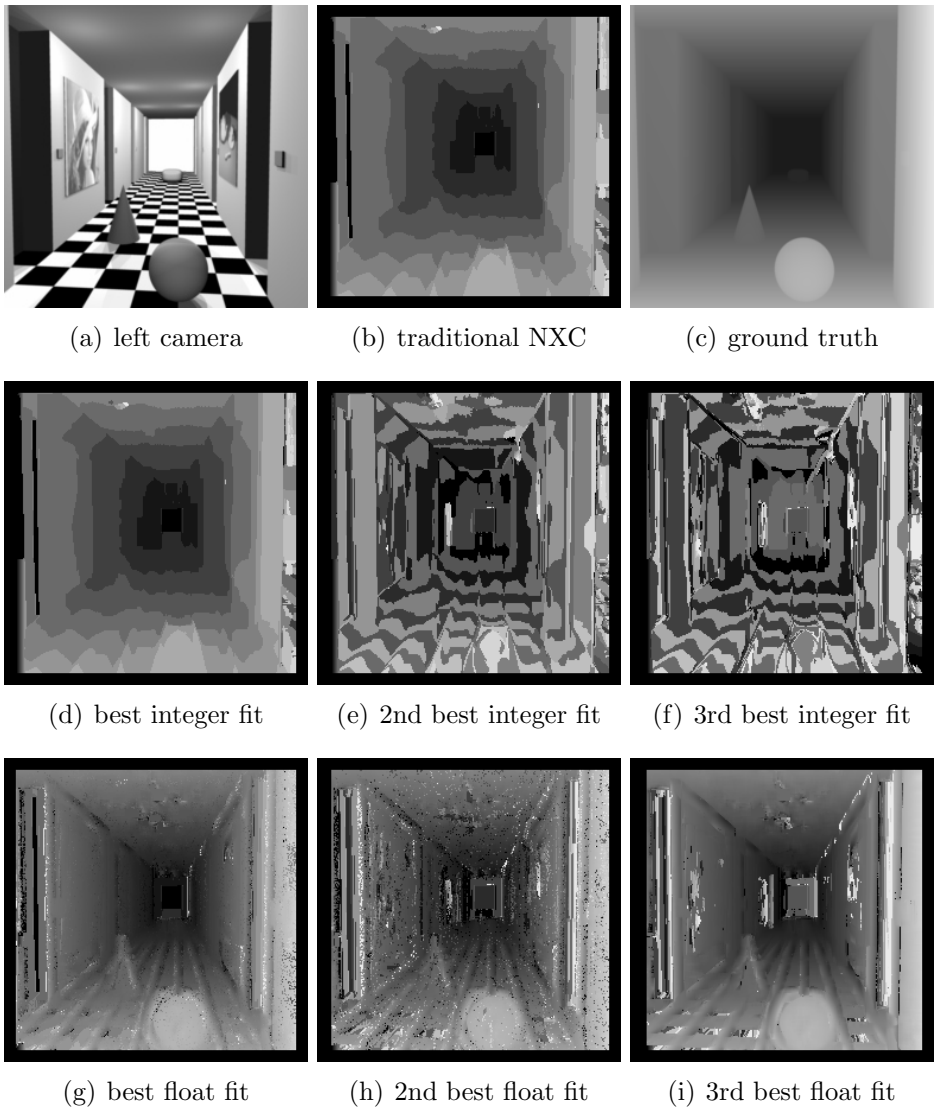
(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

FIGURE 5.7: Corridor disparity maps with 9×9 NXC window before and after dwSSD.

As a further experiment, Figure 5.7 shows data obtained using 9×9 normalized cross correlation. In the second row, we show the three best fits obtained via NXC, that is used to initialize the dwSSD (always a 9×9 window), whose results are shown in the third row.

One can see that for this scene it provides a seemingly worse result than SSD, especially at the right wall close to the camera (except for the light switch, which is at the vertical center of that wall). This introduces a lot of noise in that location also for the interpolated disparity, as these seemingly random matches are too far from the global minimum for dwSSD.



(a) left camera (b) traditional NXC (c) ground truth

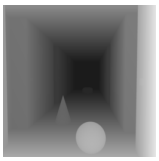
(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

FIGURE 5.8: Corridor disparity maps with 21×21 NXC window before and after dwSSD.

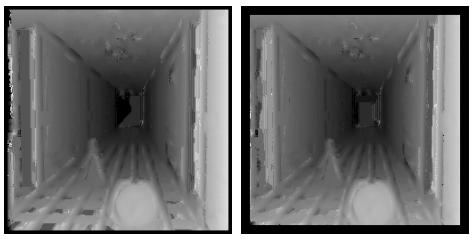
In Figure 5.8, as with 21×21 SSD, we can see that a large window helps somewhat in scenes with lots of homogeneous areas and no small details. It also helps alleviate the problem with the checkerboard pattern on the floor, as it blurs the discontinuities. However, in another scene, where those could have actually been different surfaces, it would have made things worse.

Figure 5.9 shows the finally obtained disparity maps (after GSID) with all methods and all window sizes that we have tried. Figures 5.10 and 5.11 show how GSID proceeds for different initializations (only SSD initialization is shown). As most improvements happen during the first few iterations, not many more changes are visible in those figures, however one can see the floor better filled-in using the larger GSID window.

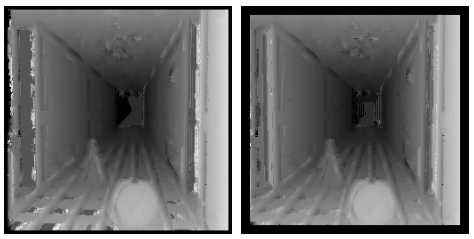


(a) ground truth

— $5 \times 5 \times 3$ Geometric Support Region —

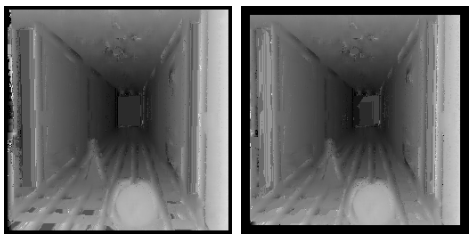


(b) 9x9 SSD (c) 21x21 SSD

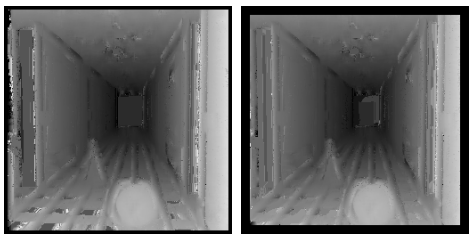


(d) 9x9 NXC (e) 21x21 NXC

— $21 \times 21 \times 7$ Geometric Support Region —

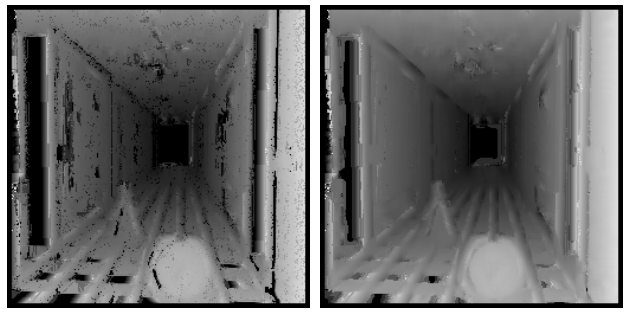


(f) 9x9 SSD (g) 21x21 SSD

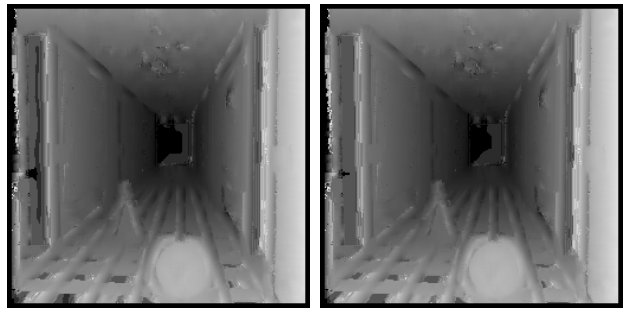


(h) 9x9 NXC (i) 21x21 NXC

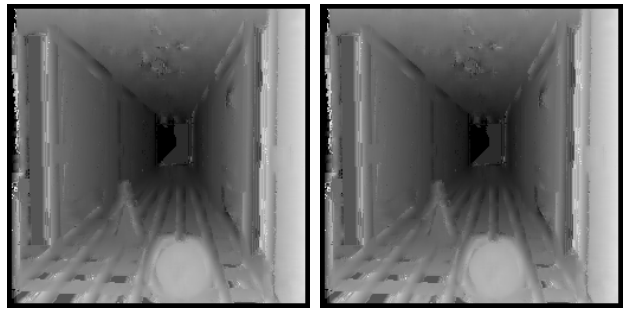
FIGURE 5.9: Corridor final disparity maps after 8 iterations with different initializations and GS window sizes.



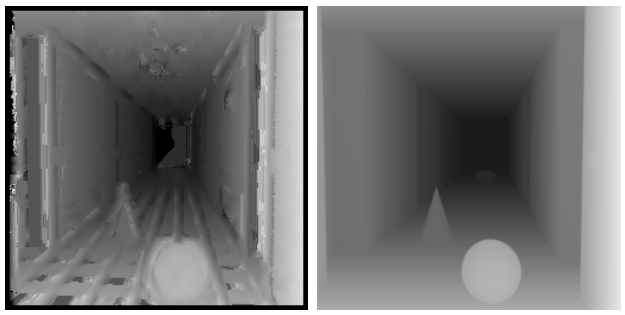
(a) initialization (b) iteration 3



(c) iteration 7 (d) iteration 11

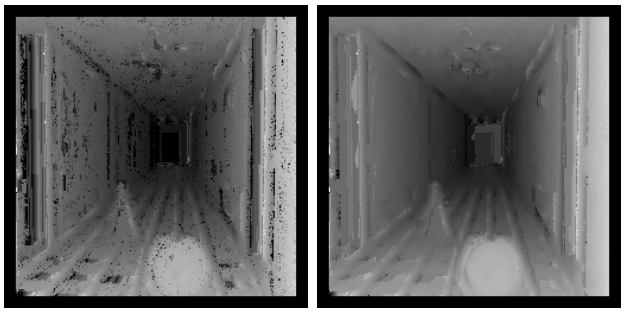


(e) iteration 15 (f) iteration 19

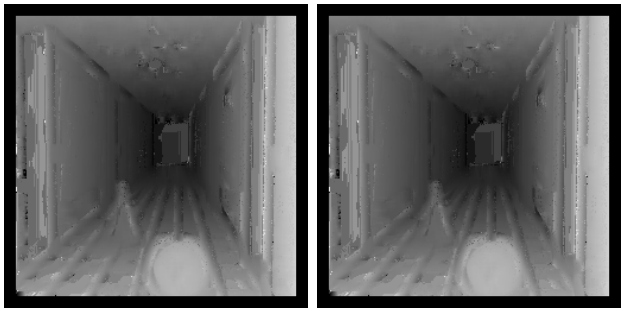


(g) iteration 23 (h) ground truth

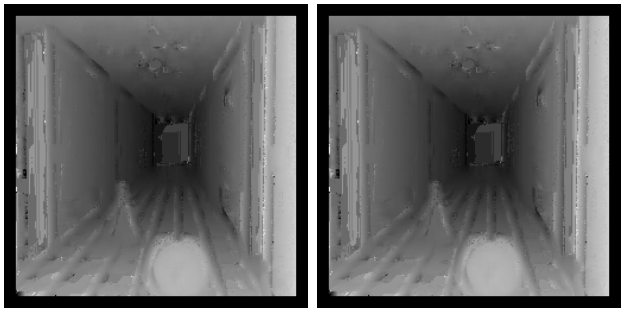
FIGURE 5.10: Corridor sequence for 9×9 SSD with $5 \times 5 \times 3$ GS after the indicated iterations.



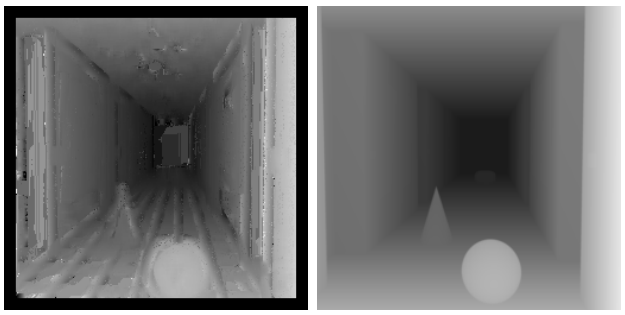
(a) initialization (b) iteration 3



(c) iteration 7 (d) iteration 11

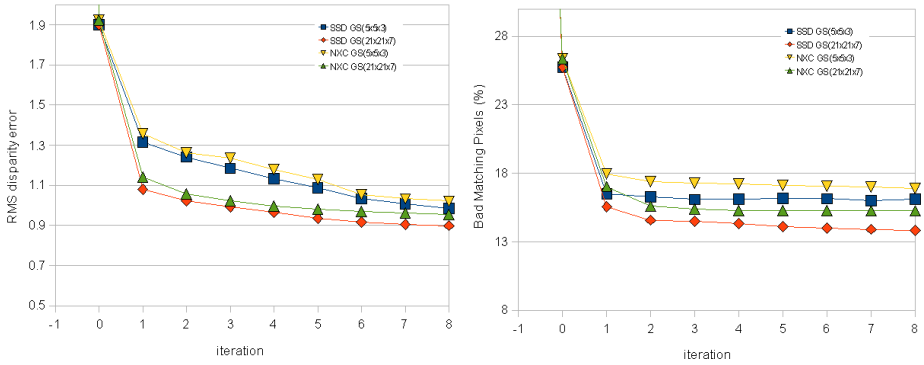


(e) iteration 15 (f) iteration 19

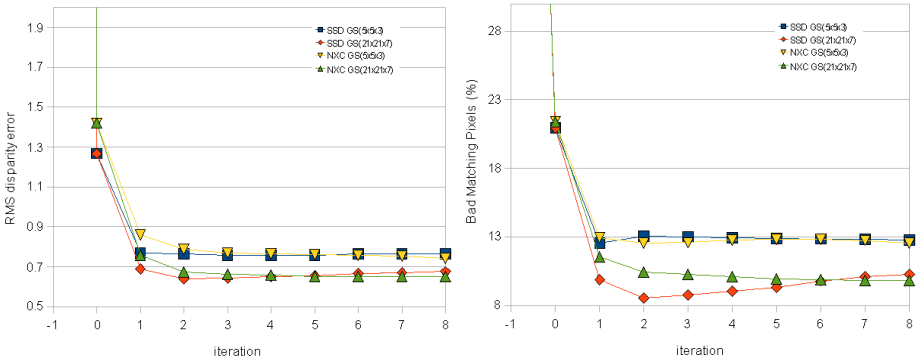


(g) iteration 23 (h) ground truth

FIGURE 5.11: Corridor sequence for 21×21 SSD with $21 \times 21 \times 7$ GS after the indicated iterations.



(a) window size 9×9



(b) window size 21×21

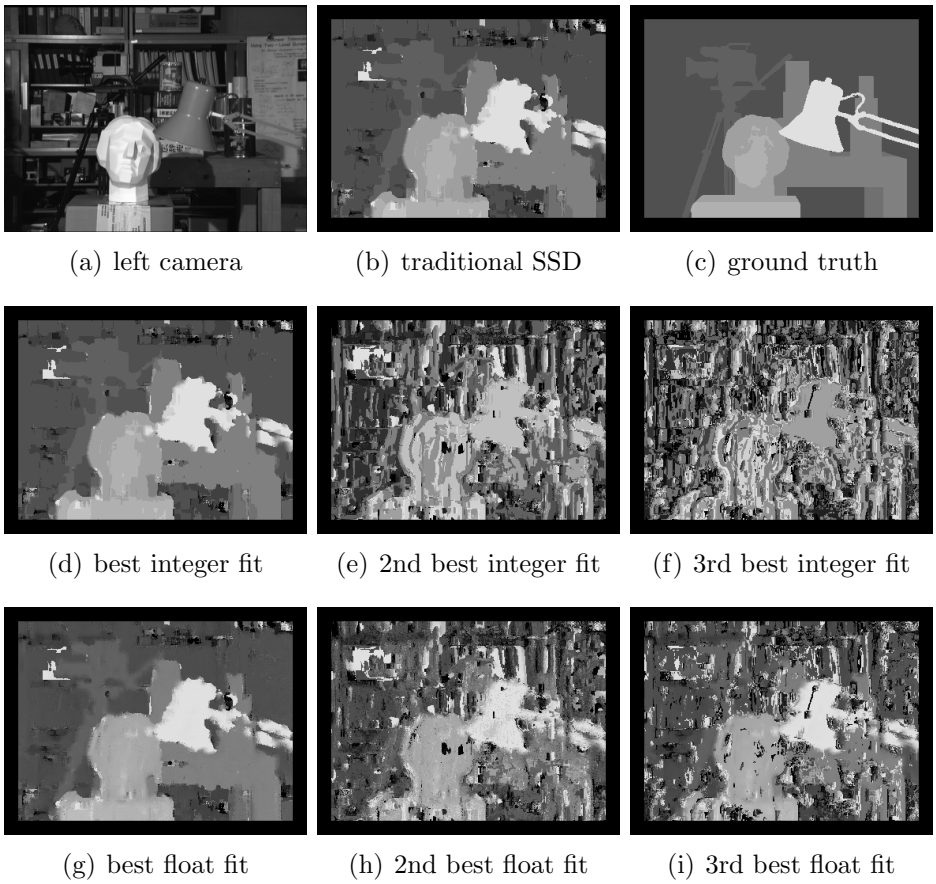
FIGURE 5.12: Corridor GSID error statistics. **First column**: RMS disparity error (disparity units) **Second column**: Bad matching pixels (percent)

As most improvements happen during the first few iterations in Figures 5.10 and 5.11, not many more changes are visible in those figures[3], however one can see the floor better filled-in because of the previously mentioned advantage of a larger SSD window in this case.

Finally, Figure 5.12 shows the RMS error for different initializations and support regions as a chart. Iteration -1 is the value for the traditional method (integer disparity via SSD or NXC) — as expected it is very bad (in the order of 100 RMS disparity units), that is why we don't scale the charts to show it, but we focus on the relevant measurements. Obviously, the larger GSID window always provides better results, as it aggregates more information from a larger neighborhood.

Table 5.3 later in this chapter shows exact AA/BP errors for both all and non-occluded regions only.

[3]using our custom diffusion method presented in Appendix A



(a) left camera (b) traditional SSD (c) ground truth

(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

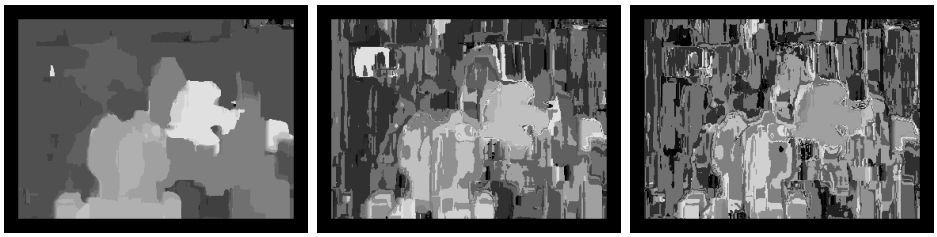
FIGURE 5.13: Tsukuba disparity maps with 9×9 SSD window before and after dwSSD.

Figure 5.13 shows data for 9×9 SSD. In the second row, we show the three best fits obtained via traditional SSD, that is used to initialize the dwSSD, whose results are displayed in the third row.

In this and the following experiments one can see that the background of the Tsukuba scene is particularly challenging due to its homogeniety and repeating patterns, especially in the upper right corner. It is however interesting to note the almost perfect detection of the camera in the background which often even escapes human observers as some parts of it almost blend in completely with the background.

The problematic region at the left of the camera can however be solved by looking at the other best fits, which confirms that most actual correspondences indeed lie near to the very best fit obtained. We show this in more detail in one of the upcoming figures.

(a) left camera (b) traditional SSD (c) ground truth



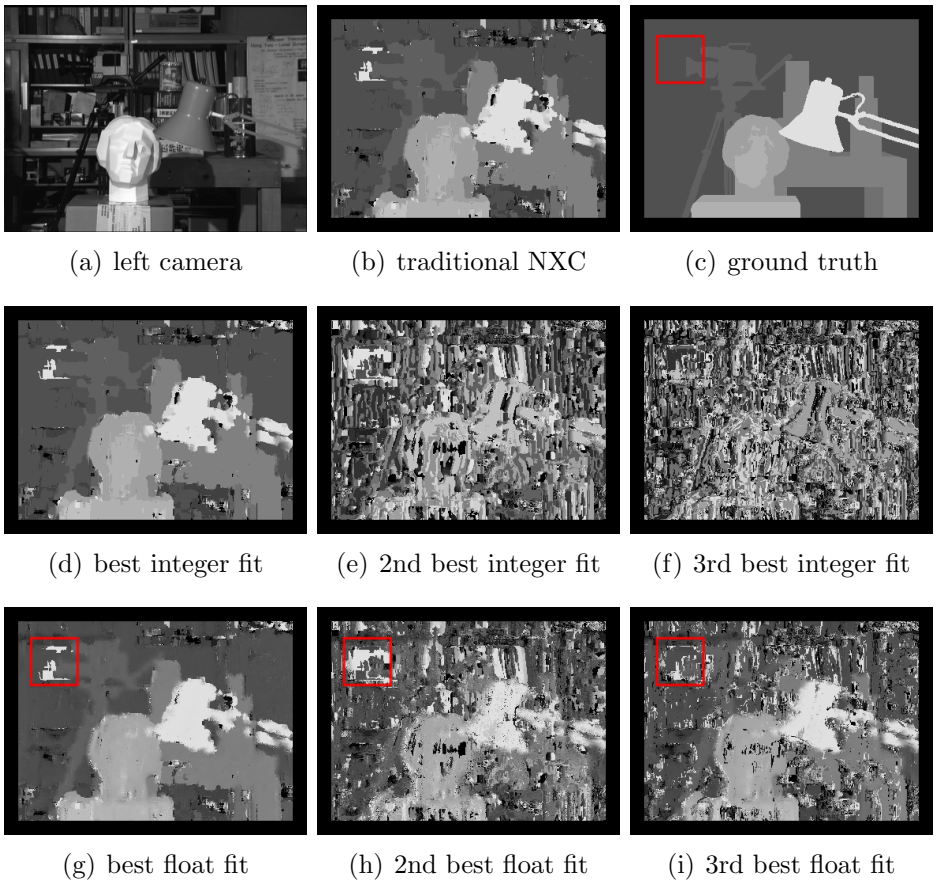
(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

FIGURE 5.14: Tsukuba disparity maps with 21×21 SSD window before and after dwSSD.

Figure 5.14 shows data obtained using 21×21 SSD. In the second row, we show the three best fits obtained via traditional SSD, that is used to initialize the dwSSD (always a 9×9 window), whose results are displayed in the third row.

As stated previously, using a large window size in a scene with lots of detail blurs edges too much. Here we can see this e.g. at the top of the head and also at the lamp stem, which is almost completely lost. However the background is reconstructed well. The Tsukuba sequence would probably yield a better reconstruction not with deformed windows but adaptive window sizes.



(a) left camera (b) traditional NXC (c) ground truth

(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

FIGURE 5.15: Tsukuba disparity maps with 9×9 NXC window before and after dwSSD.

Figure 5.15 shows data obtained using 9×9 normalized cross correlation. In the second row, we show the three best fits obtained via NXC, that is used to initialize the dwSSD (always a 9×9 window), whose results are shown in the third row.

We have highlighted a region, where one can see that intelligently combining information from the three different disparity maps (three best fits), might yield a much better final result. GSID can be considered as a filling-in method for this local method. Remember that disparities are never changed once they have been initially computed — we finally merely select one of several fits that we obtained in the beginning.

(a) left camera (b) traditional NXC (c) ground truth

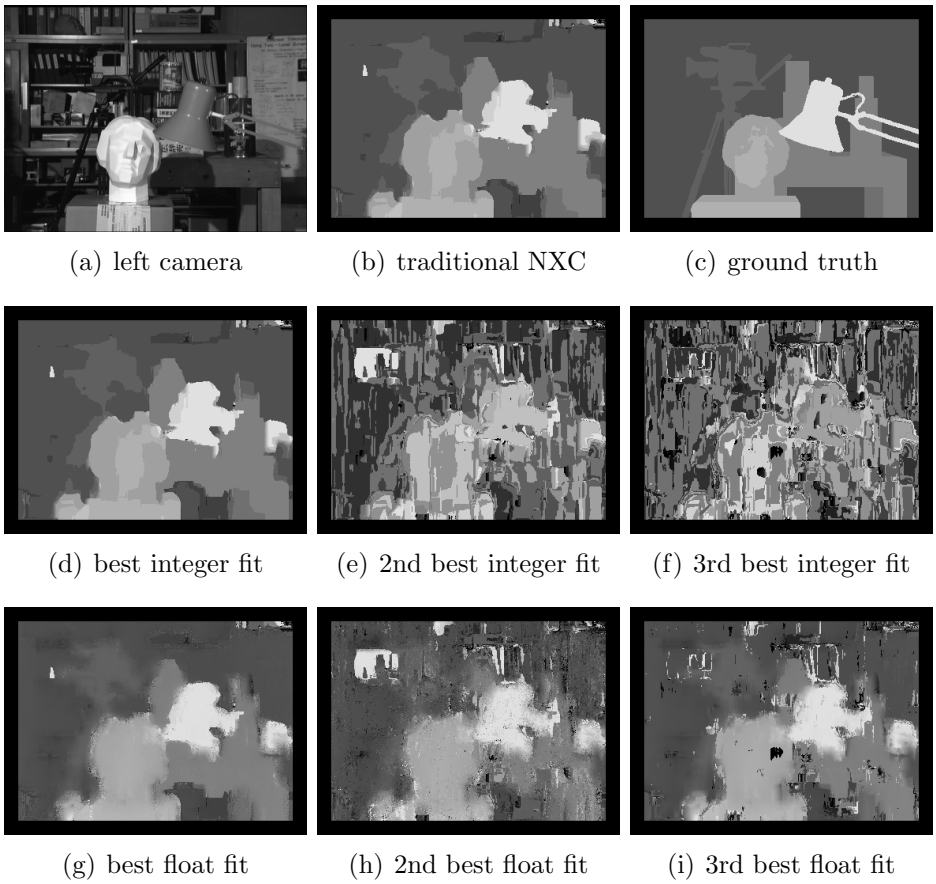
(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

FIGURE 5.16: Tsukuba disparity maps with 21×21 NXC window before and after dwSSD.

Figure 5.16 shows data obtained using 21×21 NXC. In the second row, we show the three best fits obtained, that are used to initialize the dwSSD (a 9×9 window), whose results are displayed in the third row.

For this scene and this window size, NXC seems to perform more or less equally to SSD.



(a) left camera (b) traditional NXC (c) ground truth

(d) best integer fit (e) 2nd best integer fit (f) 3rd best integer fit

(g) best float fit (h) 2nd best float fit (i) 3rd best float fit

FIGURE 5.17: Tsukuba with 21×21 NXC and dwSSD window before and after dwSSD.

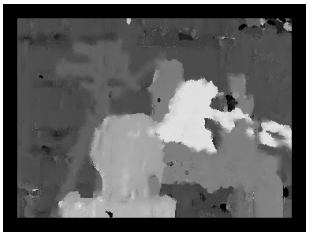
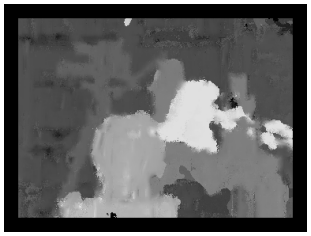
As an extra experiment, we have included for the NXC region matching with window size 21×21 an increased window for dwSSD of 21×21 (in all other cases it was always kept at 9×9) for the Tsukuba sequence. The result is depicted in Figure 5.17 and it is also mentioned as a fifth case in the corresponding error statistics.

As already mentioned in Section 5.3.3, using a large dwSSD window (in addition to the already large NXC window), simply blurs edges too much for a good reconstruction, especially in this scene containing lots of details.

Figure 5.18 shows the finally obtained disparity maps with all methods and all window sizes that we have tried. Figures 5.19 and 5.20 show how the geometric support space diffusion (GSID) evolves for different initializations (only NXC initialization is shown). Note how the filling-in is faster for an initialization with data obtained by using a large GSID window (Figure 5.20): the gap in the head closes more quickly than in Figure 5.19.

(a) ground truth

— $5 \times 5 \times 3$ Geometric Support Region —

(b) 9x9 SSD (c) 21x21 SSD


(d) 9x9 NXC (e) 21x21 NXC

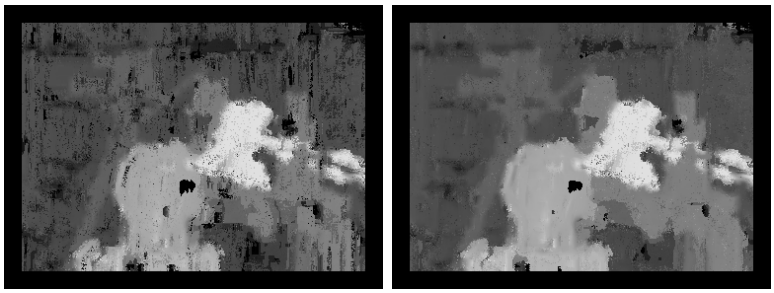
— $21 \times 21 \times 7$ Geometric Support Region —


(f) 9x9 SSD (g) 21x21 SSD

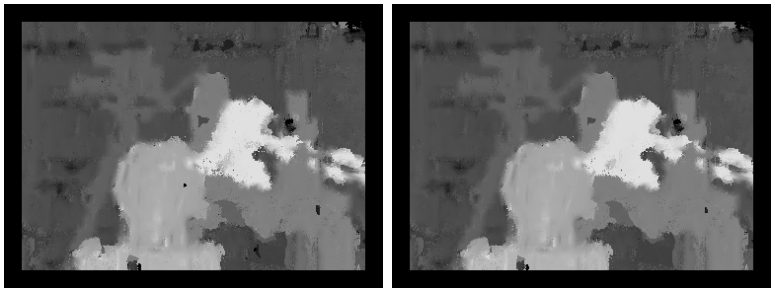


(h) 9x9 NXC (i) 21x21 NXC (j) 21x21 NXC+dwSSD

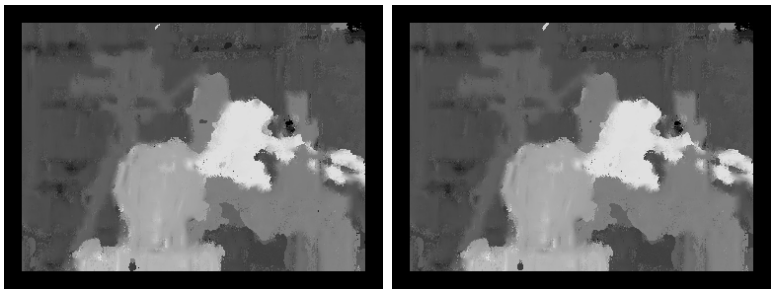
FIGURE 5.18: Tsukuba final disparity maps after 8 iterations with different initializations and GS window sizes.



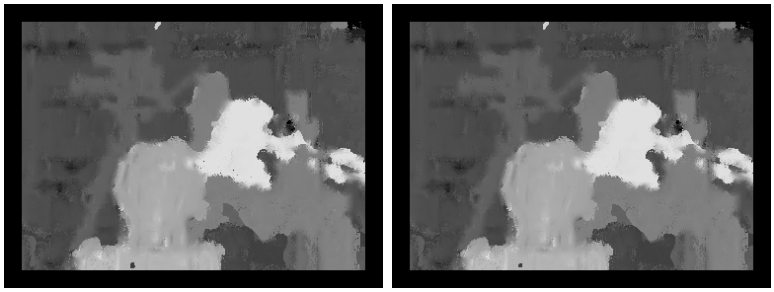
(a) initialization (b) iteration 1



(c) iteration 2 (d) iteration 3

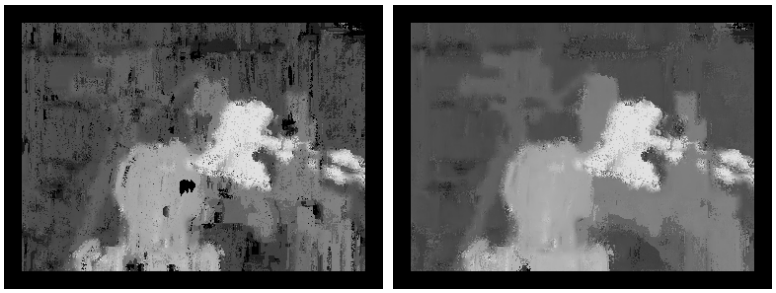


(e) iteration 4 (f) iteration 5

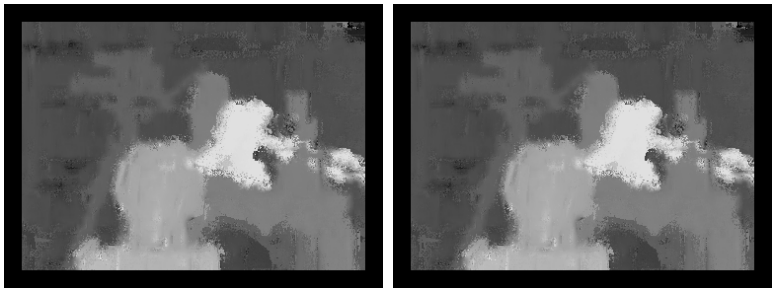


(g) iteration 6 (h) iteration 7

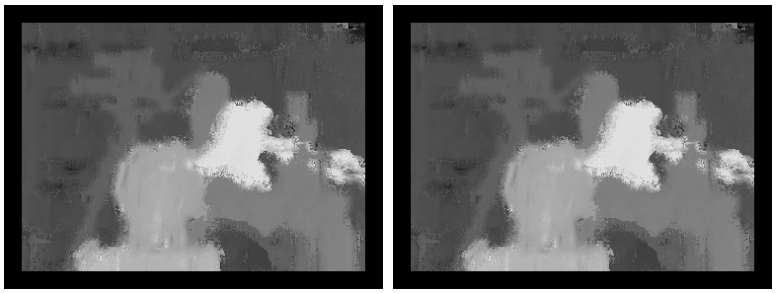
FIGURE 5.19: Tsukuba sequence for 9×9 NXC with $5 \times 5 \times 3$ GS after the indicated iterations.



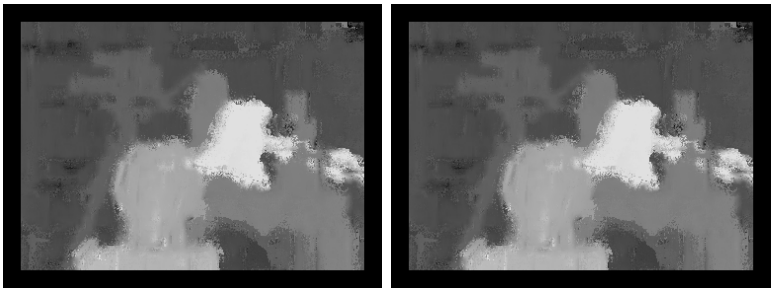
(a) initialization (b) iteration 1



(c) iteration 2 (d) iteration 3

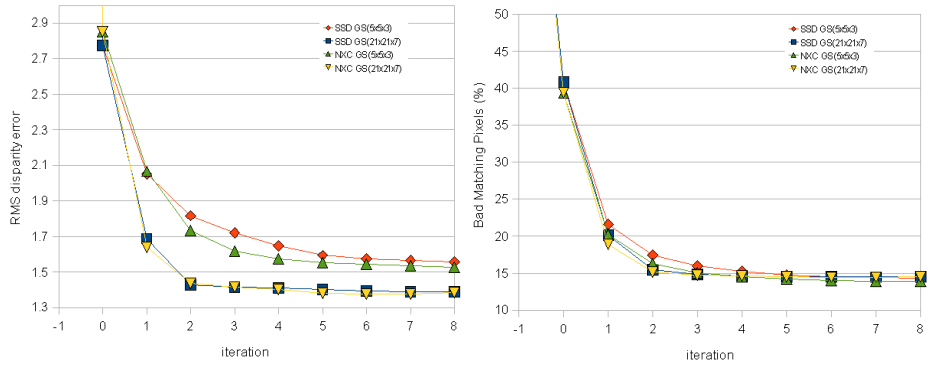


(e) iteration 4 (f) iteration 5

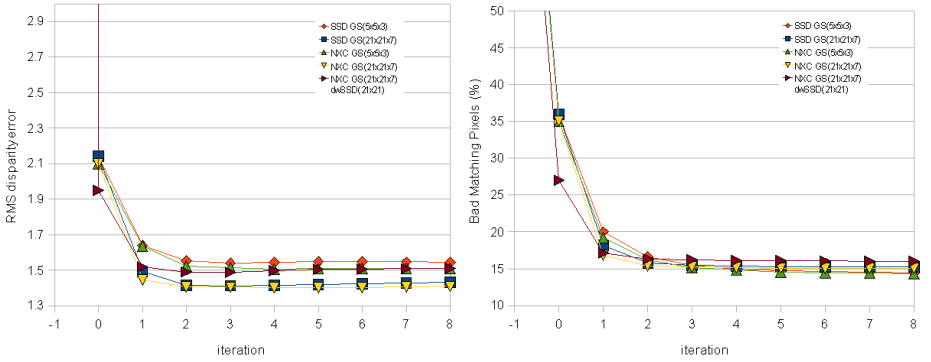


(g) iteration 6 (h) iteration 7

FIGURE 5.20: Tsukuba sequence for 21×21 NXC with $21 \times 21 \times 7$ GS after the indicated iterations.



(a) window size 9×9



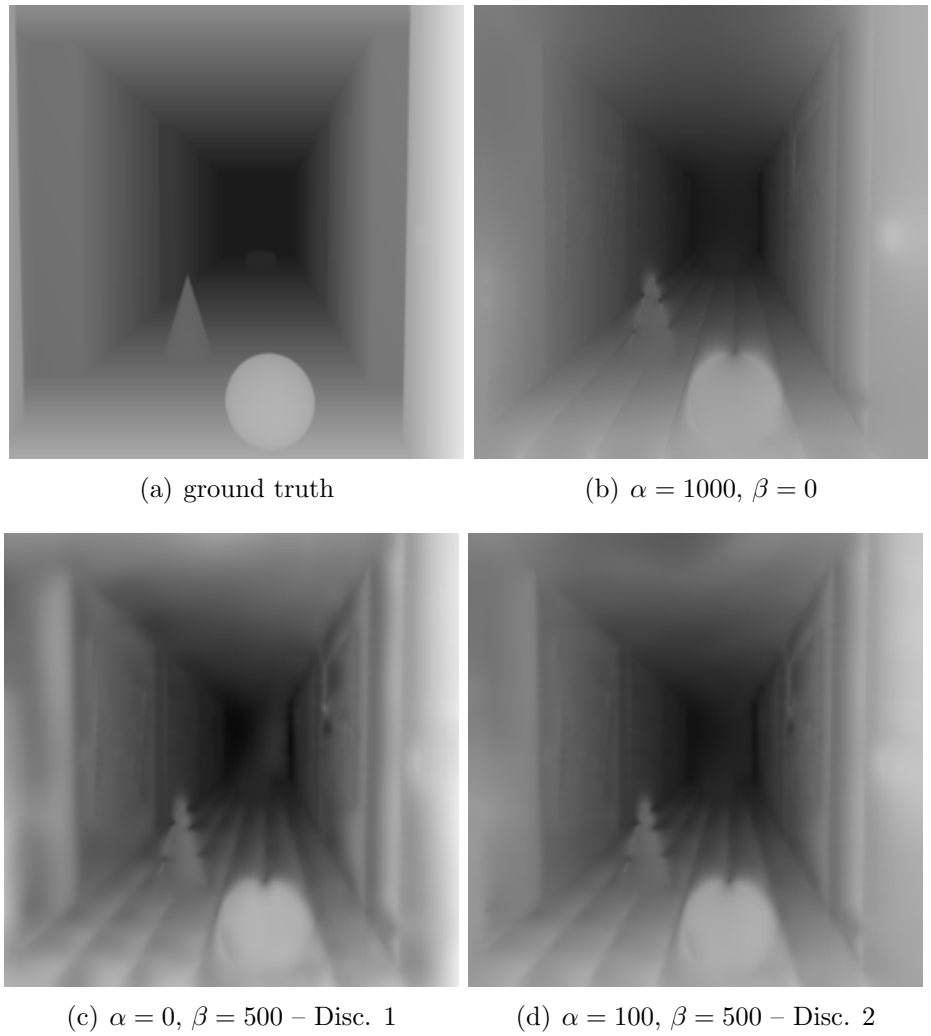
(b) window size 21×21

FIGURE 5.21: Tsukuba GSID error statistics. **First column**: RMS disparity error (disparity units) **Second column**: Bad matching pixels (percent)

Finally, Figure 5.21 shows the RMS error for different initializations and support regions as a chart. Iteration -1 is the value for the traditional method (integer disparity via SSD or NXC). For more info on the diffusion method see Appendix A.

As before, the biggest improvements in the error measure happen during the first few iterations of our diffusion method. Note that for this scene the window size does not seem to matter as much as for the Corridor, as the final result after GSID is more or less the same disparity error. This is probably due to the fact that none of our initializations are really good. (Again, we would probably need at least adaptive window sizes for the local region matching.)

Table 5.4 later in this chapter shows exact AA/BP errors for both all and non-occluded regions only.



(a) ground truth (b) $\alpha = 1000$, $\beta = 0$

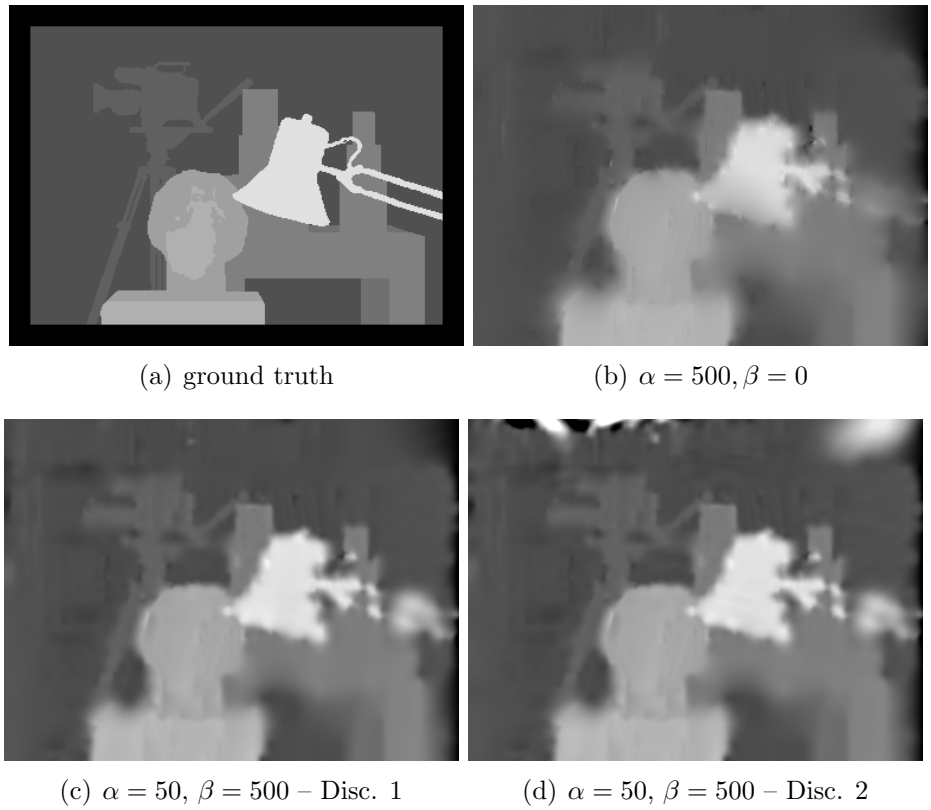
(c) $\alpha = 0$, $\beta = 500$ – Disc. 1 (d) $\alpha = 100$, $\beta = 500$ – Disc. 2

FIGURE 5.22: Corridor disparity map estimation using global method. Model parameters as indicated. Numerical parameters as given in the text.

Figures 5.22 and 5.23 show results for both scenes with the global method with both Horn/Schunck and HO regularizers. Disc. 1 and 2 refer to the discretization for the HO mixed terms (4.19) and (4.21), respectively.

Strangely, using HO regularization looks worse than only Horn/Schunck for the Corridor dataset (especially the floor), however the error measure clearly shows that it improves results.

Note how the HO regularizer with discretization #1 allows for a more correct separation of the head and the lamp in the Tsukuba dataset. Discretization #2 seems inferior, as it produces artifacts at the boundaries (especially the top) with both scenes.



(a) ground truth (b) $\alpha = 500, \beta = 0$

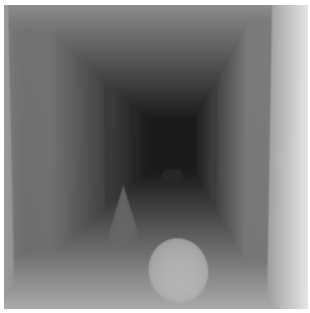
(c) $\alpha = 50$, $\beta = 500$ – Disc. 1 (d) $\alpha = 50$, $\beta = 500$ – Disc. 2

FIGURE 5.23: Tsukuba disparity map estimation using global method. Model parameters as indicated. Numerical parameters as given in the text.

Finally, Figure 5.24 shows the best results side-by-side. Although the local method for the Corridor seems to produce less smooth results than the global method, it nevertheless appears that certain areas are reconstructed more correctly. This could be due to the fact that slanted surfaces are directly integrated into the data term of our local method, whereas in the global method only smoothness is higher-order. For Tsukuba, it is obvious that the global method prevails because the background is filled in more correctly.

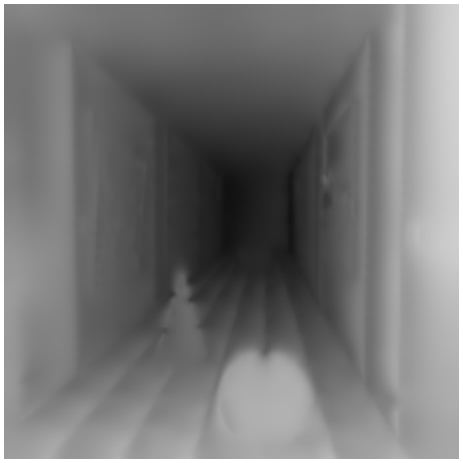
Tables 5.3 and 5.4 show the computed AA disparity and BP errors for the Corridor and Tsukuba scene, respectively. Considering that the Tsukuba dataset consists of about 1.57 times more pixels than the Corridor, it has nevertheless about $0.616 - (0.331 \cdot 1.57) = 0.096 \approx 0.1$ AADE units worse error statistics for all pixels, and $0.539 - (0.313 \cdot 1.57) = 0.049$ in the non-occluded case (comparing the best results). This confirms that occlusions pose a great deal of problems and that the Tsukuba scene can be considered a rather demanding scene for stereo algorithms.

In those tables, *bestint* denotes the best integer fit (the initialization for dwSSD) and *bestfloat* the best fit (in floating point precision) produced by dwSSD.

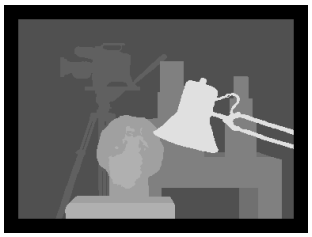


(a) Corridor Ground Truth

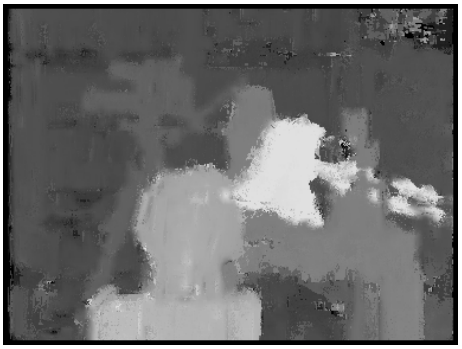
(b) Corridor Local — Winner



(c) Corridor Global



(d) Tsukuba Ground Truth



(e) Tsukuba Local

(f) Tsukuba Global — Winner

FIGURE 5.24: Local/Global Lineup — Best Results

	Local Method — SSD (9 × 9)								Local Method — NXC (9 × 9)							
	GS(5×5×3)				GS(21×21×7)				GS(5×5×3)				GS(21×21×7)			
	AA		BP		AA		BP		AA		BP		AA		BP	
iteration (t)	all	non-occl	all	non-occl	all	non-occl	all	non-occl	all	non-occl	all	non-occl	all	non-occl	all	non-occl
bestint	95.234	94.476	100.000	100.000	95.234	94.476	100.000	100.000	88.978	88.116	100.000	100.000	88.978	88.116	100.000	100.000
bestfloat	0.666	0.652	17.134	16.336	0.666	0.652	17.134	16.336	0.737	0.725	19.364	18.615	0.737	0.725	19.364	18.615
t=0	0.918	0.908	24.740	24.051	0.918	0.908	24.740	24.051	0.909	0.899	25.110	24.443	0.909	0.899	25.110	24.443
1	0.549	0.533	15.457	14.629	0.484	0.468	14.365	13.532	0.571	0.556	16.883	16.081	0.515	0.500	15.771	14.972
2	0.520	0.504	15.274	14.446	0.453	0.436	13.414	12.558	0.534	0.518	16.486	15.677	0.471	0.455	14.482	13.650
3	0.502	0.485	15.055	14.224	0.438	0.421	13.387	12.533	0.524	0.508	16.339	15.529	0.453	0.436	14.335	13.499
4	0.485	0.469	15.091	14.260	0.424	0.407	13.233	12.376	0.503	0.486	16.260	15.451	0.441	0.424	14.254	13.417
5	0.472	0.455	15.152	14.322	0.410	0.392	12.997	12.137	0.484	0.468	16.179	15.368	0.434	0.417	14.231	13.393
6	0.456	0.438	15.150	14.320	0.399	0.381	12.838	11.974	0.460	0.443	16.100	15.288	0.428	0.410	14.207	13.369
7	0.448	0.431	14.999	14.167	0.394	0.376	12.766	11.901	0.453	0.436	16.035	15.224	0.424	0.407	14.202	13.364
8	0.440	0.422	15.121	14.293	0.391	0.373	12.658	11.792	0.452	0.435	15.963	15.151	0.421	0.403	14.197	13.358
9	0.437	0.420	15.125	14.297	0.390	0.372	12.548	11.682	0.452	0.435	15.928	15.115	0.418	0.400	14.152	13.313
10	0.434	0.417	15.006	14.176	0.388	0.370	12.407	11.536	0.452	0.435	15.883	15.069	0.416	0.399	14.145	13.305
11	0.432	0.414	14.798	13.964	0.387	0.369	12.299	11.426	0.452	0.435	15.847	15.033	0.415	0.398	14.132	13.292
12	0.430	0.413	14.748	13.913	0.387	0.369	12.200	11.326	0.451	0.434	15.807	14.992	0.414	0.397	14.120	13.280
13	0.428	0.410	14.629	13.793	0.387	0.369	12.159	11.286	0.451	0.434	15.798	14.983	0.414	0.397	14.114	13.274
14	0.428	0.410	14.624	13.787	0.387	0.369	12.148	11.275	0.450	0.433	15.779	14.963	0.415	0.397	14.120	13.280
15	0.428	0.411	14.628	13.791	0.387	0.369	12.155	11.282	0.450	0.433	15.773	14.958	0.415	0.398	14.121	13.283
16	0.428	0.410	14.626	13.789	0.387	0.369	12.157	11.284	0.450	0.433	15.768	14.952	0.415	0.398	14.125	13.287
17	0.428	0.410	14.629	13.793	0.387	0.369	12.162	11.289	0.450	0.433	15.762	14.947	0.416	0.399	14.132	13.294
18	0.428	0.410	14.622	13.785	0.387	0.369	12.159	11.286	0.450	0.432	15.755	14.939	0.416	0.399	14.137	13.300
19	0.428	0.410	14.624	13.787	0.387	0.369	12.155	11.282	0.449	0.432	15.753	14.938	0.416	0.399	14.132	13.294
20	0.428	0.410	14.619	13.782	0.387	0.369	12.152	11.278	0.450	0.432	15.752	14.936	0.417	0.399	14.132	13.294
21	0.428	0.410	14.611	13.774	0.388	0.370	12.157	11.284	0.449	0.432	15.744	14.928	0.417	0.399	14.139	13.302
22	0.428	0.410	14.601	13.764	0.388	0.370	12.157	11.284	0.449	0.432	15.737	14.921	0.417	0.400	14.146	13.309
23	0.427	0.410	14.590	13.753	0.388	0.370	12.153	11.280	0.449	0.432	15.735	14.919	0.417	0.400	14.148	13.311
24	0.427	0.410	14.586	13.749	0.388	0.370	12.153	11.280	0.449	0.432	15.735	14.919	0.417	0.400	14.150	13.313
	Local Method — SSD (21 × 21)								Local Method — NXC (21 × 21)							
bestint	98.848	98.197	100.000	100.000	98.848	98.197	100.000	100.000	96.339	95.651	100.000	100.000	96.339	95.651	100.000	100.000
bestfloat	0.413	0.395	10.938	10.062	0.413	0.395	10.938	10.062	0.487	0.470	12.753	11.908	0.487	0.470	12.753	11.908
t=0	0.650	0.636	20.949	20.227	0.650	0.636	20.949	20.227	0.716	0.703	21.423	20.711	0.716	0.703	21.423	20.711
1	0.406	0.388	12.531	11.647	0.357	0.339	9.891	8.994	0.430	0.412	12.922	12.045	0.393	0.375	11.530	10.660
2	0.405	0.387	13.049	12.184	**0.331**	**0.313**	**8.550**	**7.629**	0.407	0.389	12.565	11.689	0.353	0.334	10.435	9.542
3	0.403	0.385	12.987	12.120	0.335	0.317	8.769	7.855	0.402	0.384	12.577	11.702	0.348	0.330	10.245	9.354
4	0.402	0.384	12.952	12.086	0.340	0.321	9.031	8.122	0.402	0.384	12.750	11.877	0.346	0.327	10.078	9.184
5	0.403	0.385	12.884	12.016	0.344	0.325	9.322	8.417	0.401	0.383	12.803	11.932	0.343	0.324	9.904	9.007
6	0.404	0.386	12.850	11.981	0.350	0.332	9.787	8.888	0.400	0.382	12.811	11.939	0.343	0.324	9.880	8.983
7	0.404	0.387	12.809	11.939	0.355	0.336	10.076	9.182	0.399	0.381	12.737	11.865	0.342	0.324	9.830	8.932
8	0.405	0.387	12.782	11.914	0.358	0.339	10.270	9.381	0.394	0.376	12.557	11.684	0.342	0.323	9.792	8.894
9	0.406	0.388	12.859	11.992	0.360	0.341	10.412	9.526	0.391	0.372	12.394	11.518	0.342	0.323	9.767	8.868
10	0.408	0.390	12.963	12.098	0.361	0.342	10.475	9.590	0.389	0.371	12.288	11.410	0.342	0.323	9.746	8.846
11	0.410	0.392	13.055	12.191	0.361	0.343	10.512	9.628	0.388	0.370	12.250	11.373	0.341	0.323	9.712	8.812
12	0.411	0.393	13.096	12.233	0.362	0.343	10.550	9.666	0.387	0.369	12.218	11.341	0.341	0.322	9.658	8.757
13	0.411	0.393	13.116	12.254	0.362	0.344	10.572	9.688	0.387	0.368	12.171	11.293	0.341	0.322	9.631	8.730
14	0.412	0.394	13.163	12.301	0.362	0.344	10.566	9.684	0.387	0.368	12.161	11.282	0.340	0.322	9.604	8.702
15	0.413	0.395	13.242	12.381	0.362	0.344	10.541	9.659	0.386	0.368	12.161	11.282	0.340	0.321	9.529	8.625
16	0.414	0.396	13.297	12.438	0.363	0.344	10.521	9.639	0.386	0.368	12.146	11.268	0.340	0.321	9.475	8.571
17	0.414	0.396	13.315	12.456	0.363	0.344	10.496	9.613	0.386	0.368	12.128	11.249	0.339	0.321	9.426	8.521
18	0.415	0.397	13.349	12.491	0.363	0.345	10.494	9.611	0.385	0.367	12.107	11.227	0.339	0.320	9.360	8.454
19	0.415	0.397	13.376	12.520	0.363	0.345	10.484	9.600	0.385	0.366	12.073	11.193	0.339	0.320	9.304	8.397
20	0.415	0.397	13.358	12.502	0.363	0.345	10.475	9.591	0.384	0.366	12.048	11.167	0.338	0.320	9.263	8.355
21	0.414	0.396	13.330	12.473	0.363	0.345	10.468	9.584	0.384	0.366	12.030	11.149	0.338	0.320	9.220	8.311
22	0.414	0.396	13.335	12.478	0.363	0.345	10.453	9.571	0.383	0.365	12.013	11.132	0.338	0.319	9.177	8.268
23	0.414	0.396	13.340	12.484	0.363	0.345	10.444	9.562	0.383	0.364	11.985	11.103	0.338	0.319	9.153	8.244
24	0.414	0.397	13.383	12.527	0.363	0.345	10.421	9.538	0.382	0.364	11.967	11.085	0.338	0.319	9.135	8.226

Variational Method — Discretization 1					
α	β	AA all	AA non-occl	BP all	BP non-occl
1000	0	0.403	0.387	14.017	13.102
50	500	**0.354**	**0.331**	**8.819**	**7.600**

Variational Method — Discretization 2					
α	β	AA all	AA non-occl	BP all	BP non-occl
100	500	0.367	0.346	9.173	7.985
50	500	0.420	0.400	11.906	10.837

TABLE 5.3: Corridor AA and BP disparity error statistics. Best result of each category is in bold. Best overall result is underlined.

	Local Method — SSD (9 × 9)								Local Method — NXC (9 × 9)							
	GS(5×5×3)				GS(21×21×7)				GS(5×5×3)				GS(21×21×7)			
	AA		BP		AA		BP		AA		BP		AA		BP	
iteration (t)	all	non-occl	all	non-occl	all	non-occl	all	non-occl	all	non-occl	all	non-occl	all	non-occl	all	non-occl
bestint	104.334	103.066	100.000	100.000	104.334	103.066	100.000	100.000	104.933	103.824	100.000	100.000	104.933	103.824	100.000	100.000
bestfloat	0.794	0.706	15.826	13.876	0.794	0.706	15.826	13.876	0.843	0.755	16.403	14.467	0.843	0.755	16.403	14.467
t=0	1.690	1.638	40.851	39.693	1.690	1.638	40.851	39.693	1.703	1.655	39.426	38.289	1.703	1.655	39.426	38.289
1	1.002	0.930	21.588	19.825	0.840	0.771	20.181	18.481	0.988	0.917	20.263	18.498	0.810	0.741	18.875	17.150
2	0.851	0.772	17.423	15.528	0.699	0.625	15.476	13.615	0.812	0.732	16.305	14.394	0.698	0.624	15.188	13.322
3	0.795	0.712	15.960	14.014	0.687	0.611	14.858	12.959	0.757	0.674	15.035	13.078	0.686	0.612	14.754	12.853
4	0.764	0.680	15.240	13.275	0.683	0.606	14.672	12.755	0.736	0.651	14.522	12.542	0.681	0.605	14.616	12.693
5	0.743	0.657	14.779	12.799	0.680	0.602	14.574	12.642	0.725	0.639	14.202	12.201	0.675	0.598	14.500	12.556
6	0.734	0.647	14.567	12.578	0.678	0.599	14.502	12.556	0.719	0.632	14.033	12.027	0.673	0.595	14.477	12.521
7	0.728	0.641	14.410	12.408	0.677	0.598	14.486	12.526	0.715	0.628	13.939	11.942	0.674	0.595	14.481	12.513
8	0.723	0.636	14.297	12.293	**0.677**	**0.597**	14.486	12.520	0.713	0.625	**13.908**	**11.909**	0.677	0.598	14.517	12.537
	Local Method — SSD (21 × 21)								Local Method — NXC (21 × 21)							
bestint	105.821	104.697	100.000	100.000	105.821	104.697	100.000	100.000	105.730	104.676	100.000	100.000	105.730	104.676	100.000	100.000
bestfloat	0.760	0.675	15.704	13.835	0.760	0.675	15.704	13.835	0.760	0.677	15.558	13.696	0.760	0.677	15.558	13.696
t=0	1.277	1.221	35.985	34.800	1.277	1.221	35.985	34.800	1.240	1.184	35.057	33.844	1.240	1.184	35.057	33.844
1	0.839	0.767	19.993	18.262	0.757	0.687	18.139	16.350	0.822	0.749	19.247	17.477	0.726	0.655	16.818	14.969
2	0.759	0.680	16.589	14.729	0.698	0.626	15.751	13.863	0.743	0.664	16.151	14.242	0.690	0.618	15.386	13.483
3	0.735	0.653	15.428	13.523	0.694	0.620	15.480	13.562	0.723	0.641	15.196	13.253	0.687	0.613	15.225	13.292
4	0.729	0.646	14.995	13.075	0.693	0.619	15.354	13.430	0.715	0.631	14.795	12.849	0.685	0.610	15.147	13.205
5	0.725	0.641	14.735	12.807	0.694	0.618	15.293	13.364	0.711	0.627	14.557	12.607	0.685	0.609	15.099	13.150
6	0.722	0.639	14.589	12.665	0.695	0.619	15.263	13.332	0.710	0.625	14.460	12.508	0.685	0.609	15.082	13.130
7	0.720	0.636	14.494	12.571	0.696	0.620	15.264	13.329	0.708	0.624	14.377	12.424	0.686	0.609	15.067	13.108
8	0.718	0.634	14.416	12.490	0.698	0.622	15.289	13.351	0.707	0.623	14.310	12.363	0.686	0.610	15.053	13.090

	Local Method NXC (21 × 21) dwSSD(21 × 21) GS(21×21×7)			
	AA		BP	
iteration (t)	all	non-occl	all	non-occl
bestint	105.730	104.676	100.000	100.000
bestfloat	0.763	0.690	15.399	13.627
t=0	1.056	1.002	26.981	25.623
1	0.759	0.693	17.105	15.291
2	0.737	0.669	16.327	14.477
3	0.733	0.665	16.184	14.318
4	0.734	0.665	16.104	14.233
5	0.735	0.666	16.078	14.203
6	0.735	0.666	16.039	14.159
7	0.735	0.666	16.014	14.133
8	0.735	0.665	15.985	14.100

Variational Method — Discretization 1					
α	β	AA all	AA non-occl	BP all	BP non-occl
1000	0	0.666	0.597	16.799	14.799
50	500	**0.616**	**0.539**	**13.488**	**11.392**

Variational Method — Discretization 2					
α	β	AA all	AA non-occl	BP all	BP non-occl
50	500	0.652	0.576	14.413	12.345

TABLE 5.4: Tsukuba AA and BP disparity error statistics. Best result of each category is in bold. Best overall result is underlined.

Chapter 6

Summary and Outlook

In this work, we have presented methods for stereo reconstruction from two major categories: local and global. Both methods are designed to be especially apt to reconstruct scenes whose geometry contains slanted surfaces, i.e. allowing linear variations of depth/disparity, whereas the usual methods only consider constancy of those values.

Our work as far as the local method is concerned is based on [LZ05]. However, in [LZ06] the authors additionally use second order properties and chose to work in Euclidean space, which is more complicated and which we chose to ignore after carefully evaluating it, since the simple modification presented here already improves results a lot. Furthermore, it is unclear if that approach would work equally well for slanted surfaces. In theory curved surfaces are a superset of slanted surfaces; however, the implementation would have drawbacks like the inability to cope with planes that are (exactly) orthogonal to the line of sight etc. There may however be more complex scenes from which that approach might be able to extract more information than our method does.

Outliers, thresholding, etc. are a curse of local methods, therefore we have also investigated a global method. In it, we have included a related smoothness constraint, namely allowing locally affine transformations instead of locally constant ones (which is exactly the change from fronto-parallel surfaces to slanted ones). We have started with a classic Horn/Schunck method and supplemented it with a higher order regularizer, the combination of which we have shown to improve results, even with the simplest of discretizations.

Future work may include the investigation, implementation and comparison of a local method that actually can cope with all sorts of geometry or "curved surfaces", as the previously mentioned work in [LZ06] seemed not very feasible. As we show in Appendix A, a careful analysis for the geometric support space diffusion method and its initialization may also be required.

One could even think of integrating such consistency of curved surfaces into a variational model: future work may include the investigation of more complex higher order regularization terms. Such a method in the anisotropic

setting would e.g. permit locally affine variations along edges/structures instead of locally constant ones. This need not be related to stereo reconstruction in particular, but is a general issue regarding diffusion methods that remains fairly uninvestigated, and might prove useful in any number of domains like optic flow, image enhancement, inpainting (equally image compression) or medical imaging.

Last but not least, not only the smoothness assumptions are important, the data term is too. Therefore, future work might also address more complex data terms than the linearized SSD we use in this work. Those could e.g. be inspired from shape-from-shading, that is, taking into account illumination effects or photometric invariants of color images, and embedding those into a variational model. Finally, also occlusion handling might need further investigation as does the investigation of any of these methods in a multi-view, spatio-temporal or real-time scenario.

Appendix A

Diffusion Process in Local Method

A.1 The Problem

In Section 3.4, we have introduced geometric consistency of surfaces. Several iterative procedures were available to sort out which of the several fits per pixel was the right one. What matters is how those results behave. The better a neighborhood, the higher the result should be. The emphasis is on "better": it is ambiguous what is better; better geometric compatibility or better geometric support. What happens when neither one is really convincing or when both are?

We have first experimented with the method proposed in [LZ05], which we show here:

$$s_{\mathbf{i}}^{t} = \begin{cases} 1 - \frac{d_{\mathbf{i}}}{d} & \text{, if } t = 0 \\ n \cdot \frac{\sum_{\mathbf{j} \in \mathcal{N}_{\mathbf{i}}} g_{\mathbf{ij}} s_{\mathbf{j}}^{t-1}}{\sum_{\mathbf{j} \in \mathcal{N}_{\mathbf{i}}} s_{\mathbf{j}}^{t-1}} & \text{, if t} > 0 \end{cases} \tag{A.1}$$

where $n = 1$ and $\mathcal{N}_{\mathbf{i}}$ is a neighborhood around point $\mathbf{i}$ in (x, y, d)-space. We were not satisfied. It is especially unclear why choose a value that is variable across the image like $\sum s$ to normalize. Additionally it does not just normalize — it gives a result close to the average value of g while penalizing values of g that are off its average in the wrong "direction", i.e. away from s. We show its behavior in Table A.1 and Figure A.2. We claim we can do better: a behavior as shown in Table A.2 (symmetric) is preferable.

A.2 Our Improved Method

Our improvement is initially derived from the idea that normalizing as in (A.1) with $n = 1$ we have found does not work well in practice: a neighborhood containing many points has more significance than one with few points

	s good	s bad
g good	very good	good
g bad	bad	bad

TABLE A.1: s and g vs. the result (new s) obtained using Expression (A.1).

which may only provide a small number of (possibly erroneous) matches. One therefore additionally has to account for the average support (using the number of points in the neighborhood $\mathcal{N}_\mathbf{i}$):

$$\frac{\sum_{\mathbf{j}\in\mathcal{N}_\mathbf{i}} g_{\mathbf{ij}} s_\mathbf{j}^{t-1}}{\sum_{\mathbf{j}\in\mathcal{N}_\mathbf{i}} s_\mathbf{j}^{t-1}} \cdot \underbrace{\frac{\sum_{\mathbf{j}\in\mathcal{N}_\mathbf{i}} s_\mathbf{j}^{t-1}}{|\mathcal{N}_\mathbf{i}|}}_{n} = \frac{\sum_{\mathbf{j}\in\mathcal{N}_\mathbf{i}} g_{\mathbf{ij}} s_\mathbf{j}^{t-1}}{|\mathcal{N}_\mathbf{i}|} \tag{A.2a}$$

	s good	s bad
g good	good	bad
g bad	bad	very bad

TABLE A.2: s and g vs. the result obtained using our diffusion method

We take it even one step further and use

$$n = \frac{\sum_{\mathbf{j}\in\mathcal{N}_\mathbf{i}} s_\mathbf{j}^{t-1}}{|\mathcal{N}_\mathbf{i}|} \cdot \frac{|\mathcal{N}_\mathbf{i}|}{N_{\max}} \tag{A.2b}$$

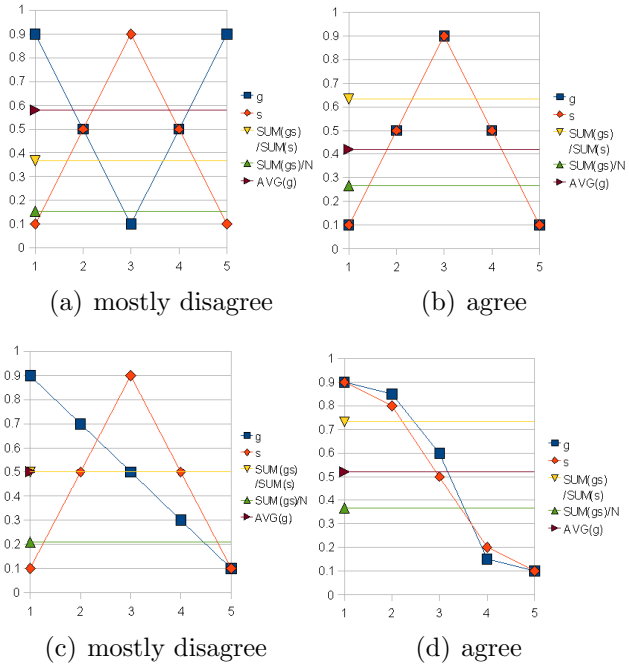
in expression (A.1), which now represents not only the average support value but also the number of neighbors seen in the current pixel ($|\mathcal{N}_\mathbf{i}|$) vs. the highest number of neighbors ever seen from a pixel in the entire image ($N_{\max}$), i.e. a good overall indicator of how representative the neighborhood is in terms of trust (support) and size (number of points).

The second problem was the behavior of the original method as we have shown in Table A.1: it does not penalize enough. It even seems to reward when g and s match in some cases (which makes no sense as they are not related) as one can see in Figure A.1(b) and (d), which we now describe in more detail.

In Figure A.1 we have shown a few examples of combinations of geometric compatibilities g and geometric support s interacting to produce the new support value at the current pixel. The examples are made up of a five-pixel neighborhood (horizontal axis), their respective values of s and g and the results produced by the respective normalizers shown as horizontal lines.

One can see that using our method, good support with good compatibility is rewarded, whereas if either support or compatibility is bad (or both are) then this is penalized.

We emphasize again: whether g disagrees or agrees with s should be unrelated, however in the original method it seems to behave like it was somehow related. Also, if one of both s or g is bad the result should a bad score, see Figure A.2(b) and (d): a good compatibility to a neighborhood we cannot trust means we should not trust it in the future either. The original method's result (support in the current pixel for the next time step) is depicted with a yellow line, our method with a green one. Figure A.2 shows similar examples, only for more extreme cases.



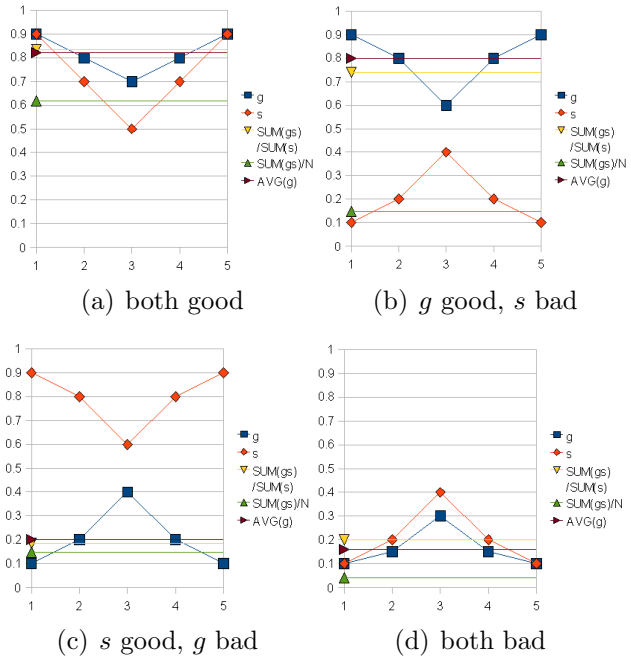
(a) mostly disagree (b) agree

(c) mostly disagree (d) agree

FIGURE A.1: Comparison of penalizations for mixed cases (good and bad values). We indicate how much s and g agree.

We have also compared the methods using combinations of two sets of neighborhood pixels. Listed in Table A.3 are all combinations of two sets of matches with neighborhoods containing 10 respectively 100 points with all combinations of good and bad geometric consistency and geometric support, where good here means 0.9 and bad means 0.1 for s and g alike. Values higher than 0.7 are highlighted in green, values lower than 0.3 in red. As usual we operate on the closed interval $(0, 1)$.

Our method (the column entitled with `(III)`) gives a good compromise between good and bad neighborhoods. The original method is located in column `(I)`; the other columns contain the results of other methods that we



(a) both good (b) g good, s bad

(c) s good, g bad (d) both bad

FIGURE A.2: Comparison of penalizations for extreme cases of s and g.

have tried and are of no importance here. Our method might look a little bit conservative, especially the low values are really very small, but it removes lots of noise (i.e. wrong matches) and converges fast.

In Table A.3, in the row prefixed with `(3)`, we notice the same behavior (deficiency) of the original method as mentioned before. Table A.4 shows the same evaluation but for two sets with the same number of elements; obviously each category degrades to the same results within, but we can still see our method performs as it should, i.e. here it mostly resorts to "don't know" ($0.45 \approx 0.5$) as results are mostly ambiguous. For the cases where really bad things happen, it penalizes further.

`(1)`, `(2)` and `(3)` at the bottom and left of Table A.3 denote what should happen under certain conditions, namely those from Table A.2. We can see those conditions applied and how the different methods behave. Note that our method (the column entitled with `(III)`) fulfills all three conditions.

Finally, in addition to the results in Chapter 5 we show in Figure A.3 the RMS disparity error for the University of Bonn Corridor Sequence with ongoing iterations (up to 24) for the original and for our method, with SSD and NXC as initializations and two geometric support neighborhood sizes: $5 \times 5 \times 3$ and $21 \times 21 \times 7$. One can see that the original method even gets a little worse after a few steps. Our method wouldn't even need that many iterations to converge, and it only gets better.

`orig` denotes the original method, `npn` denotes the method where we

	GOOD s+/g+	BAD s+/g-	BAD s-/g+	VB* s-/g-	(I) SUM(gs)/SUM(s)	(II) SUM(gs)/SUM(g)	N	(III) SUM(gs)/N	(SUM(gs)/SUM(s)) *(SUM(g)/N)	(SUM(gs)/SUM(g)) *(SUM(s)/N)	(SUM(gs)/ SUM(g)*SUM(s))
s	0.9	0.9	0.1	0.1							
g	0.9	0.1	0.9	0.1							
	100	10	0	0	0.83	0.9	110	0.74	0.68	0.81	0.43
	10	100	0	0	0.17	0.9	110	0.16	0.03	0.81	0.14
	100	100	0	0	0.5	0.9	200	0.45	0.25	0.81	0.32
	10	10	0	0	0.5	0.9	20	0.45	0.25	0.81	0.32
	0	100	10	0	0.11	0.52	110	0.09	0.02	0.43	0.09
	0	10	100	0	0.52	0.11	110	0.09	0.43	0.02	0.09
	0	100	100	0	0.18	0.18	200	0.09	0.09	0.09	0.09
	0	10	10	0	0.18	0.18	20	0.09	0.09	0.09	0.09
	0	0	10	100	0.17	0.1	110	0.02	0.03	0.01	0.06
(1)	0	0	100	10	0.83	0.1	110	0.08	0.68	0.01	0.09
	0	0	100	100	0.5	0.1	200	0.05	0.25	0.01	0.08
	0	0	10	10	0.5	0.1	20	0.05	0.25	0.01	0.08
	100	0	10	0	0.9	0.83	110	0.74	0.81	0.68	0.43
	10	0	100	0	0.9	0.17	110	0.16	0.81	0.03	0.14
	100	0	100	0	0.9	0.5	200	0.45	0.81	0.25	0.32
	10	0	10	0	0.9	0.5	20	0.45	0.81	0.25	0.32
	100	0	0	10	0.89	0.89	110	0.74	0.74	0.74	0.45
(2)	10	0	0	100	0.48	0.48	110	0.08	0.08	0.08	0.24
(3)	100	0	0	100	0.82	0.82	200	0.41	0.41	0.41	0.41
	10	0	0	10	0.82	0.82	20	0.41	0.41	0.41	0.41
	0	100	0	10	0.1	0.83	110	0.08	0.01	0.68	0.09
	0	10	0	100	0.1	0.17	110	0.02	0.01	0.03	0.06
	0	100	0	100	0.1	0.5	200	0.05	0.01	0.25	0.08
	0	10	0	10	0.1	0.5	20	0.05	0.01	0.25	0.08

(1) no trust/support but some good compat. -> should not be propagated

(2) many very bad points (no trust and no compat.) -> should not be propagated

(3) many very good and very bad points -> should yield at most “don't know” (0.5)

* VERY BAD

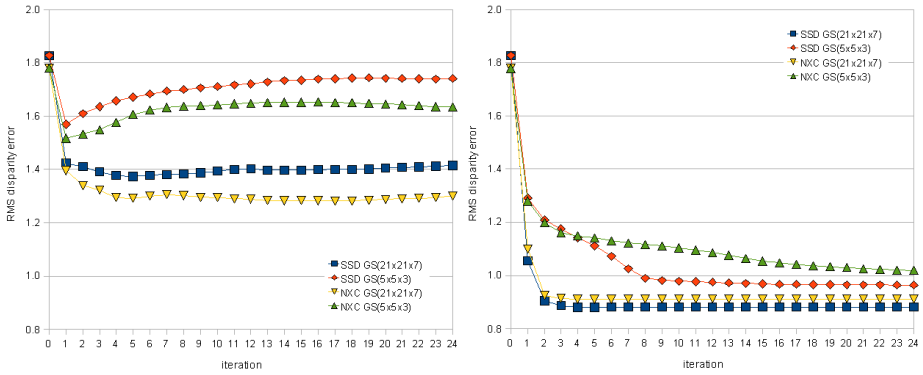
TABLE A.3: Different geometric support iterative procedures. $c_1 = 100$, $c_2 = 10$. See text for details.

	GOOD s+/g+	BAD s+/g-	BAD s-/g+	VB* s-/g-	(I) SUM(gs)/SUM(s)	(II) SUM(gs)/SUM(g)	N	(III) SUM(gs)/N	(SUM(gs)/SUM(s)) *(SUM(g)/N)	(SUM(gs)/SUM(g)) *(SUM(s)/N)	(SUM(gs)/ SUM(g)*SUM(s))
s	0.9	0.9	0.1	0.1							
g	0.9	0.1	0.9	0.1							
	100	100	0	0	0.5	0.9	200	0.45	0.25	0.81	0.32
	100	100	0	0	0.5	0.9	200	0.45	0.25	0.81	0.32
	100	100	0	0	0.5	0.9	200	0.45	0.25	0.81	0.32
	100	100	0	0	0.5	0.9	200	0.45	0.25	0.81	0.32
	0	100	100	0	0.18	0.18	200	0.09	0.09	0.09	0.09
	0	100	100	0	0.18	0.18	200	0.09	0.09	0.09	0.09
	0	100	100	0	0.18	0.18	200	0.09	0.09	0.09	0.09
	0	100	100	0	0.18	0.18	200	0.09	0.09	0.09	0.09
	0	0	100	100	0.5	0.1	200	0.05	0.25	0.01	0.08
	0	0	100	100	0.5	0.1	200	0.05	0.25	0.01	0.08
	0	0	100	100	0.5	0.1	200	0.05	0.25	0.01	0.08
	0	0	100	100	0.5	0.1	200	0.05	0.25	0.01	0.08
	100	0	100	0	0.9	0.5	200	0.45	0.81	0.25	0.32
	100	0	100	0	0.9	0.5	200	0.45	0.81	0.25	0.32
	100	0	100	0	0.9	0.5	200	0.45	0.81	0.25	0.32
	100	0	100	0	0.9	0.5	200	0.45	0.81	0.25	0.32
	100	0	0	100	0.82	0.82	200	0.41	0.41	0.41	0.41
	100	0	0	100	0.82	0.82	200	0.41	0.41	0.41	0.41
	100	0	0	100	0.82	0.82	200	0.41	0.41	0.41	0.41
	100	0	0	100	0.82	0.82	200	0.41	0.41	0.41	0.41
	0	100	0	100	0.1	0.5	200	0.05	0.01	0.25	0.08
	0	100	0	100	0.1	0.5	200	0.05	0.01	0.25	0.08
	0	100	0	100	0.1	0.5	200	0.05	0.01	0.25	0.08
	0	100	0	100	0.1	0.5	200	0.05	0.01	0.25	0.08

* VERY BAD

TABLE A.4: Different geometric support iterative procedures. $c_1 = 100$, $c_2 = 100$. See text for details.

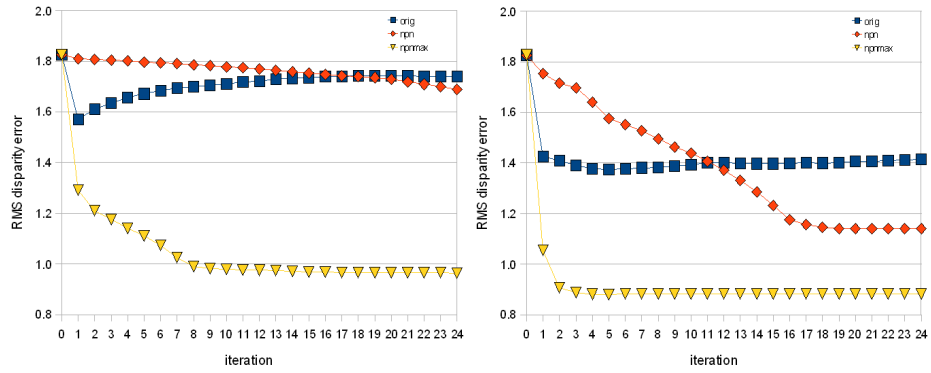
normalize by the number of points in the neighborhood of that pixel, as in (A.2a). `npnmax` denotes the method where we additionally normalize by the maximum number of neighbors seen across the entire image, as in (A.2b). Figure A.4 additionally shows the evolution of the diffusion process and the corresponding RMS error for different initializations and window sizes.



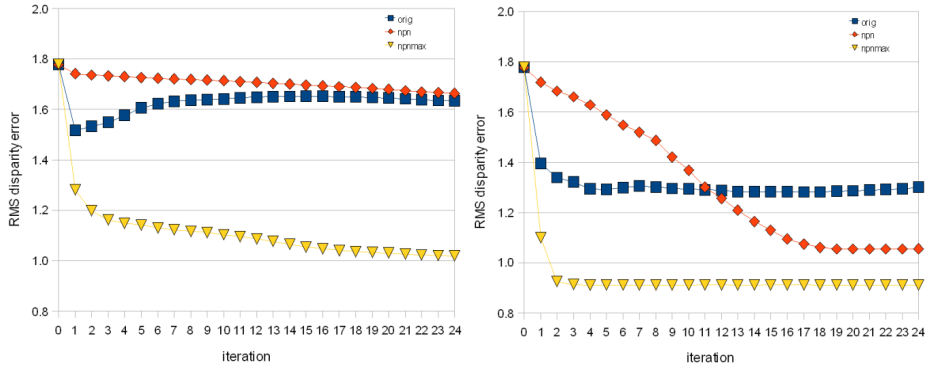
(a) original method (b) our method

FIGURE A.3: RMS disparity error of the Corridor Sequence for the two methods with different initializations and support regions.

We have not gone beyond this empiric investigation, but we wanted however to show what was behind our custom method of aggregating support, which works well in practice.



(a) 9×9 SSD and $5 \times 5 \times 3$ GSR (b) 9×9 SSD and $21 \times 21 \times 7$ GSR



(c) 9×9 NXC and $5 \times 5 \times 3$ GSR (d) 9×9 NXC and $21 \times 21 \times 7$ GSR

FIGURE A.4: Comparison of Corridor Sequence with different initializations, geometric support regions (GSRs) and diffusion methods. (lower is better)

Bibliography

[Ana87] P. Anandan. *Measuring Visual Motion from Image Sequences*. Ph.D. thesis, University of Massachusetts Amherst, Amherst, Massachusetts, USA, 1987.

[BAHH92] J. R. Bergen, P. Anandan, K. J. Hanna and R. Hingorani. Hierarchical Model-Based Motion Estimation. In *Computer Vision – ECCV 1992*, volume 588 of *Lecture Notes in Computer Science*, pages 237–252. Springer, Berlin, 1992.

[BBH93] R. C. Bolles, H. H. Baker and M. J. Hannah. The JISCT Stereo Evaluation. In *Proceedings of the DARPA Image Understanding Workshop*, pages 263–274. 1993.

[BBPW04] T. Brox, A. Bruhn, N. Papenberg and J. Weickert. High accuracy optic flow estimation based on a theory for warping. In T. Pajdla and J. Matas, editors, *Computer Vision – ECCV 2004*, volume 3024 of *Lecture Notes in Computer Science*, pages 25–36. Springer, Berlin, 2004.

[BFB94] J. L. Barron, D. J. Fleet and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):pages 43–77, February 1994.

[BG07] M. Bleyer and M. Gelautz. Graph-cut-based stereo matching using image segmentation with symmetrical treatment of occlusions. *Signal Processing: Image Communication*, 22(2):pages 127–143, 2007.

[Bru06] A. Bruhn. *Variational Optic Flow Computation – Accurate Modelling and Efficient Numerics*. Ph.D. thesis, Department of Mathematics and Comptuer Science, Saarland University, Germany, July 2006.

[Bru07] A. Bruhn. Correspondence Problems in Computer Vision. Lecture at Saarland University, Saarbrücken, Germany, 2007.

[BT99] S. Birchfield and C. Tomasi. Multiway Cut for Stereo and Motion with Slanted Surfaces. *International Conference on Computer Vision*, 1(11):pages 489–495, 1999.

[BVZ01] Y. Boykov, O. Veksler and R. Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):pages 1222–1239, 2001.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):pages 679–698, 1986.

[Col96] R. T. Collins. A Space-Sweep Approach to True Multi-Image Matching. In *Proceedings of the 1996 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '96)*, pages 358–363. IEEE Society Press, Washington, DC, USA, 1996.

[Dev74] P. Dev. Segmentation processes in visual perception: A cooperative neural model. In *Proceedings of the 1974 Conference on Biologically Motivated Automata Theory*. 1974.

[Goe03] M. Goesele. Computer Graphics II — 3-D Scanning. Lecture at Saarland University, Saarbrücken, Germany, 2003.

[Han74] M. J. Hannah. *Computer Matching of Areas in Stereo Images*. Ph.D. thesis, Stanford University, 1974.

[Hea01] M. T. Heath. *Scientific Computing*. McGraw-Hill Higher Education, 2001.

[HEH05] D. Hoiem, A. A. Efros and M. Hebert. Automatic photo pop-up. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 577–584. ACM Press, New York, NY, USA, 2005.

[HS81] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:pages 185–203, 1981.

[KO94] T. Kanade and M. Okutomi. A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):pages 920–932, 1994.

[LK81] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 674–679. Vancouver, Canada, August 1981.

[Luc84] B. D. Lucas. *Generalized Image Matching by the Method of Differences*. Ph.D. thesis, School of Computer Science, Carnegie–Mellon University, Pittsburgh, PA, 1984.

[LZ05] G. Li and S. W. Zucker. Stereo for Slanted Surfaces: First Order Disparities and Normal Consistency. In A. Rangarajan, B. C. Vemuri and A. L. Yuille, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition – EMMCVPR 2005*, volume 3757 of *Lecture Notes in Computer Science*, pages 617–632. Springer, Berlin, 2005.

[LZ06] G. Li and S. W. Zucker. Differential Geometric Consistency Extends Stereo To Curved Surfaces. In A. Leonardis, H. Bischof and A. Pinz, editors, *Computer Vision – ECCV 2006*, volume 3953 of *Lecture Notes in Computer Science*, pages 44–57. Springer, Berlin, 2006.

[Nis87] H. K. Nishihara. PRISM: Practical Real-Time Imaging Stereo Matcher. pages 63–72, 1987.

[OK91] M. Okutomi and T. Kanade. A Multiple Baseline Stereo. In *Proceedings of the 1991 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '91)*, pages 63–69. IEEE Society Press, 1991.

[PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, second edition, 1992.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[SBW05] N. Slesareva, A. Bruhn and J. Weickert. Optic Flow Goes Stereo: A Variational Method for Estimating Discontinuity-Preserving Dense Disparity Maps. In *Pattern Recognition*, volume 3663 of *Lecture Notes in Computer Science*, pages 33–40. Springer, Berlin, 2005.

[SS96] D. Scharstein and R. Szeliski. Stereo Matching with Non-Linear Diffusion. In *Proceedings of the 1996 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '96)*, page 343. IEEE Society Press, Washington, DC, USA, 1996.

[SS02] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1–3):pages 7–42, 2002.

[SZ00] R. Szeliski and R. Zabih. An Experimental Comparison of Stereo Algorithms. In *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 1–19. Springer, Berlin, 2000.

[TV98] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, Englewood Cliffs, 1998.

[Vek01] O. Veksler. Stereo Matching by Compact Windows via Minimum Ratio Cycle. *International Conference on Computer Vision*, 1:page 540, 2001.

[Wik08] Wikipedia. Epipolar geometry — Wikipedia, The Free Encyclopedia, 2008. [Online; accessed 10-January-2008].

[You50] D. Young. *Iterative Methods for Solving Partial Difference Equations of Elliptic Type*. Ph.D. thesis, Harward University, Cambridge, Massachusetts, USA, 1950.

List of Figures

List of Tables

Index