Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehrstuhl für Informatik VIII (Computergraphik und Multimedia)
Prof. Dr. Leif Kobbelt

Institute for Creative Technologies, Graphics Group
Dr. Paul Debevec

Diploma thesis

Depth-Discontinuity Preserving Optical Flow Using Time-Multiplexed Illumination

Malte Weiss
Matriculation number: 243 541

June 2007

First evaluator: Prof. Dr. Leif Kobbelt
Second evaluator: Dr. Paul Debevec, Research Associate Professor

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht zu haben.

Aachen, June 6, 2007

(Malte Weiss)

Abstract

In the last 25 years optical flow has become more and more important in the field of Computer Vision. Many algorithms have been published since the first paper in 1981 and a lot of concepts have been introduced facing problems which occur when the pixel displacements between frames are determined. Even though todays algorithms explicitly consider violations of their model assumptions, they still suffer from two main issues: an imprecise estimation of motion discontinuities, which often causes flow algorithms to oversmooth these boundaries, and the inability to find correct displacements if an object in the scene is weakly textured, for example with a repetitive texture pattern.

We show that the computation of optical flow can be strongly improved if additional geometric information about the respective frames is incorporated: the surface normals of the objects in the scene and the depth discontinuities. The former provides a larger feature vector in order to find more unique displacements, whereas the latter allows the estimation of a piecewise smooth flow field, which preserves discontinuities between object boundaries. Both inputs do not require a 3D representation of the scene but retrieve the information channels by illuminating the scene under different lighting conditions.

This thesis explains how the surface normals and the depth discontinuities of a scene can be obtained using specific lighting conditions and how they are incorporated into an existing optical flow framework. We focus on human performances in the context of Image Based Relighting.

Acknowledgements

This thesis would not have been possible without the support of many people. I thank Prof. Dr. Leif Kobbelt for supporting my idea to write my diploma thesis abroad and Dr. Paul Debevec for giving me the opportunity to gain a great insight into the work of an American research institute. I want to thank my two supervisors, Tim Hawkins and Arne Schmitz, for taking time to give me helpful criticism and advices for my thesis. Special thanks to Pieter Peers for giving me all the feedback for my optical flow algorithm, as well as Charles-Félix Chabert and Brian Miller, who made the shoot in the Light Stage 6 possible. I also want to thank all other staff members of the Graphics Lab of the Institute for Creative Technologies for their ideas and the sharing of their knowledge. Furthermore, I want to thank Helge Weiss, Lilian Walk, Matthias Tandler and Vera Klautke for their extensive proof reading.

I especially like to thank Horst and Lucia Liebl for helping me to settle down in Los Angeles and to handle the new circumstances I had to face in that metropolis. Last but not least, I thank my family for their emotional encouragement and my girlfriend, Vera, for her love and her endurance to strengthen me selflessly, even when I was thousands of miles away.

Contents

Chapter 1

Introduction

In 1979 James J. Gibson (1904-1979), an American psychologist for visual perception, introduced the so-called "Ecological Approach to Visual Perception" [Gib79]. Unlike other theorists, he argued that the visual perception of a human-being or an animal is the result of an *active exploration* of the environment. He focused on the stimuli which subjects perceive instead of the explanation of the mechanism which performs the visual cognition. Gibson investigated how fighter pilots in the Second World War were able to successfully perform landing operations while estimating the correct distances between the aircraft and the runways. In this context he introduced the term *optical flow*: if a human-being moves through the world, the stimulating patterns on the retina in the eyes are permanently moving.

These movements yield a field of so-called *flow vectors*, the *optical flow field*, which gives clear clues about the depth, the orientation and the locomotion of objects in the environment. When pilots are landing an aircraft they probably focus on a specific point on the runway. This point is always projected onto the same position on the retina. However, the surrounding points change their projected areas in the eye while the aircraft approaches the airport. The speed at which these points change their positions on the retina yields precise information about the distances to the airstrip. Gibson strongly influenced the comprehension of visual perception and proved that optical flow is an important element of it.

The pioneer work of Berthold K. P. Horn and Brian G. Schunck in 1981 [HS81] brought optical flow to Computer Science, and since then the estimation of optical flow has become more and more important in the field of Computer Vision. Informally spoken, given a sequence of two-dimensional images, e.g. a video, the optical flow describes where the pixels move from one frame to the next. The knowledge of this information provides a vast application range like video compression, motion segmentation, feature tracking, matting, extraction of geometry, etc.

In the field of Computer Vision the term optical flow is derived from the *motion field*, which has been defined by Horn & Schunck [HS94]. The motion field describes the movements of objects in a three-dimensional scene projected onto the image plane (see figure 1.1). Given two frames of an image sequence, the motion field maps each pixel in the first frame to its new position in the second frame by adding a *displacement vector*. Therefore the motion field is a strictly geometric concept.

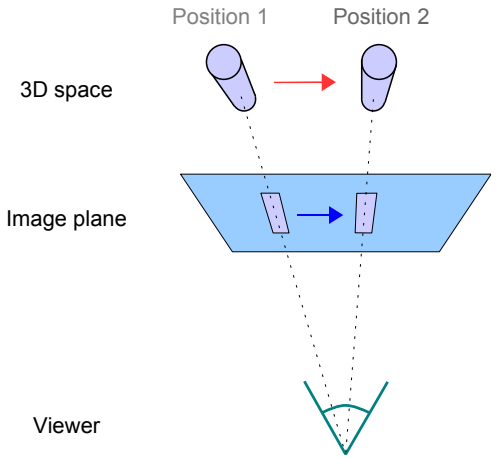


Figure 1.1: The motion field represents the movement of objects in a 3D scene (red arrow) projected onto the image plane (blue arrow).

According to the psychology of visual perception the *optical flow* denotes the projection of the *apparent motion* onto the image plane. Usually, the optical flow is desired to match the motion field but the fields are not necessarily the same. This can be illustrated with the example of a Barber's pole (see figure 1.2). The motion field is unique but there is an infinite number of optical flow fields describing the *apparent* motion.

1.1 Assumptions

All optical flow algorithms that have been developed since the publication of Horn & Schunck's first optical flow paper introduce model assumptions, which must hold to find the displacement of a pixel from one frame to the next. The three most important assumptions are:

Data conservation constraint All optical flow algorithms presume the *data conservation*: "Image measurements (e.g. image intensity) corresponding to a small image region remain the same, although the location of the region may change over time." [Bla92] Given a sequence of images $\mathbf{I} : \mathbb{R}^3 \rightarrow \mathbb{R}$, where

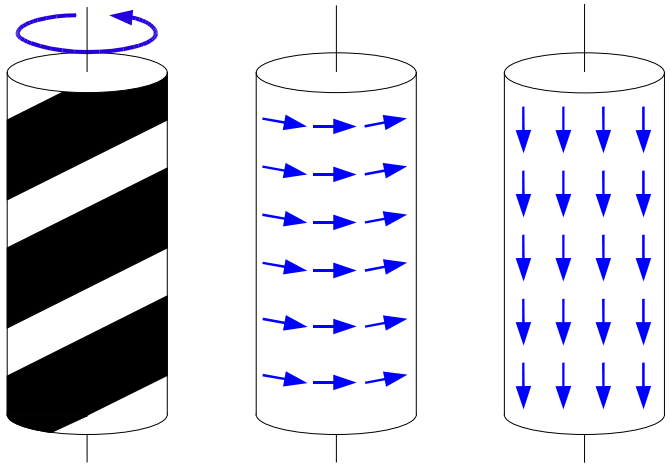


Figure 1.2: Ambiguous optical flow. Left: Rotating Barber's pole. Middle: Motion field. Right: One alternative optical flow field.

$\mathbf{I}(x, y, t)$ denotes the image intensity of a pixel (x, y) at time t, the prevalent formulation of this constraint is

$$\begin{aligned} \mathbf{I}(x, y, t) &= \mathbf{I}(x + \delta x, y + \delta y, t + \delta t) \\ &= \mathbf{I}(x + u\delta t, y + v\delta t, t + \delta t) \end{aligned} \tag{1.1}$$

$\mathbf{u} = (u, v)$ designates the displacement, the *flow vector*, where u and v are the velocity of the pixel in horizontal and vertical direction respectively. δt is a small time-difference. Therefore, the intensity of a pixel does not change over time due to motion, which is normally referred to as *brightness constancy assumption*.

Usually, optical flow is computed between two subsequent frames in a sequence. Therefore the time parameter t commonly corresponds to the t-th frame in the sequence and $\delta t = 1$ denotes a single step from one frame to the next.

Spatial coherence constraint Since the incorporation of a single pixel is usually not sufficient to determine its flow vector, most algorithms state that flow vectors are *spatially coherent*: "Neighboring points in the scene typically belong to the same surface and hence have similar velocities. Since neighboring points in the scene project to neighboring points in the image plane, we expect optical flow to vary smoothly." [Bla92] This is also known as the *smoothness constraint*.

Temporal coherence constraint Finally, a lot of algorithms assume that a flow field does not change much over time, i.e. the flow field between the frames t and $t + 1$ in a sequence of images is similar to the field between the frames $t - 1$ and t. This constraint is often called the *temporal smoothness constraint*.

1.2 Requirements

Finding the optical flow between two frames is a non-trivial task. Until today no algorithm has been developed that can handle all potential problems. The most common issues optical flow algorithms face are:

Complex material properties Most of the existing optical flow algorithms presume that all materials in the scene reflect light in a diffuse way. But specularities on object surfaces can cause ambiguous flow fields. Imagine a scene containing a rotating sphere with a diffuse texture and a specular highlight. If only a simple data conservation constraint is assumed, the rotation of the sphere will not be detected within the area of the specular highlight, because despite the rotation its position remains static in the image plane. The situation becomes even more complicated if translucency is involved. If two translucent surfaces move in opposed directions, there are two apparent motions interfering with each other.

Occlusion and appearance A flow field between two frames of a sequence should be *dense*. It should contain a displacement for each pixel in the first frame in order to map it to the second one. However, it is common in natural images that parts of objects disappear, because they are occluded by other objects, move out of the image rectangle, or are just transformed in a way that surface points which were visible before face back from the viewer. Let us look at an even worse situation where a rectangle simply appears in the second frame that has not been visible in the first one. The flow vectors should be zero in that area, even though the data conservation constraint is strongly violated. Analogously, an approach should be able to cope with pixels that disappear in the second frame.
A robust optical flow algorithm must be capable to deal with the following question: How to assign a flow vector to a pixel if there is no visible correspondence between both frames? It should be able to identify these pixels and treat them in a special way.

Motion discontinuities The two *coherence constraints* are often violated in common image sequences as well. For instance, if two objects move next to each other in opposite directions, the spatial coherence constraint is violated at the boundaries between them because the optical flow field does not vary smoothly at these places. While preserving coherences in areas that move rigidly, optical flow algorithms should allow discontinuities in the flow field at motion boundaries.

Brightness changes When objects move in natural or even synthetic images the apparent brightness of their surface points usually changes from frame to frame. The reasons are manifold: changing illumination, interreflections, non-diffuse materials, etc. The data conservation constraint should consider these brightness changes.

Noise Optical flow is primarily intended to be used for natural images since it can be calculated for synthetic ones in an analytical way (see section 5.1.2). The acquisition of pictures with photo or video cameras naturally adds noise to the resulting images. This becomes even worse if the images are compressed as a postprocessing step. The detection of the optical flow vectors should be robust against noise and compression artifacts.

Complex motion Finally, an optical flow algorithm should be capable to deal with different kinds of motion (translation, rotation, sheering, perspective foreshortening etc.) in the same scene while avoiding the interference of these.

1.3 Contribution

The general difficulty that optical flow algorithms face is the lack of knowledge about the scene geometry. Most of today's optical flow approaches rely on two-dimensional information, which is provided by the input images. This tends to be scanty if the model assumptions are violated. We believe that the optical flow estimation can be strongly improved by the introduction of additional geometric information about the scene, in which optical flow fields are computed. Thus, we try to approach the optical flow nearer to the motion field.

We implement an existing highly accurate optical flow algorithm and incorporate two types of geometric information:

- **Depth discontinuities** Motion discontinuities usually appear at boundaries between objects. We follow the idea of *discontinuity preserving stereo* by Feris et al. [FRC+05] and assume that motion discontinuities match discontinuities in the depth map of a frame. With the knowledge of these discontinuities, our basic optical flow algorithm can be modified to specifically regard these locations. Furthermore, we extend the approach of Feris et al. to dynamic scenes.
- **Surface normal maps** The knowledge about the *surface normals* of objects in the scene might provide better clues for the detection of correspondences than just the color information in the image sequences. This is especially the case if objects contain repetitive texture patterns which make it harder to find a unique displacement. We will extend our algorithm in order to regard an extended *feature vector* for a more accurate flow estimation: the texture information *and* the normal information.

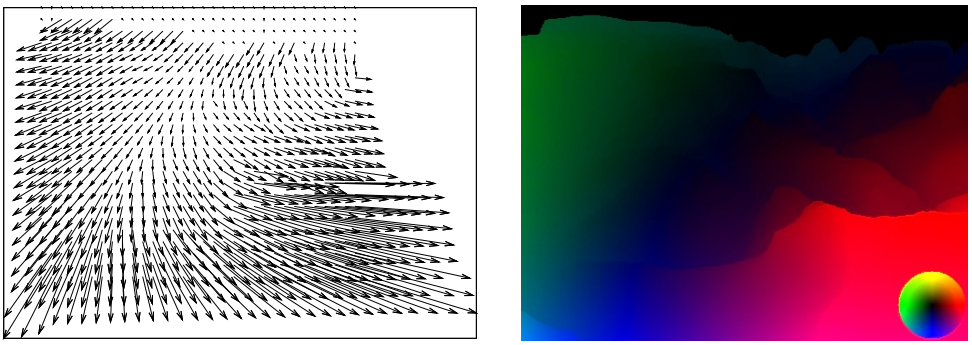
We detect these two information channels without the use of a 3D model but by capturing the scene multiple times under different lighting conditions. Even though this approach can only be generalized to a limited extent, since an additional capturing process is required, we assume that the optical flow estimation is strongly improved and the effort is worthwhile. We focus our work on the acquisition of human performances and intend to use it in the context of Image Based Relighting.

1.4 Notations

In this section we briefly explain the notations in this thesis. We consider a rectangular image as a function $I : \Omega \rightarrow \mathbb{R}^i$ which maps a pixel position $(x, y) = \mathbf{x} \in \Omega$ to an intensity ($i = 1$) or RGB color value ($i = 3$). Each pixel is mapped to a *real* scalar or a *real-valued* vector, since we process high-dynamic range (HDR) images. If a low dynamic range image is loaded, the discrete range of $[0, 255]$ is mapped to $[0, 1]$. We assume that images are *continuous* functions. The values of intermediate pixels are computed by means of bilinear interpolation of the four nearest neighbors.

A sequence of images is defined by $\mathbf{I} : \Theta \rightarrow \mathbb{R}^i$ which maps a *temporal* pixel position $(x, y, t) \in \Theta$ to an intensity or color value. The parameter t refers to the integer frame index in the sequence as mentioned above.

A flow field is a two-dimensional function $I : \Omega \rightarrow \mathbb{R}^2$ which maps pixel positions $\mathbf{x}$ to flow vectors $\mathbf{u}$. The flow vector describes the displacement, i.e. the offset (u, v), which is *added* to a pixel position in one frame to retrieve the corresponding position in the second frame. Most of the authors in the literature represent an optical flow field by a subsampled vector field. Figure 1.3a shows the flow field of the camera movement through a canyon (see section 5.3.1.2) in that representation. We work with images in high resolutions and want to examine the entire flow field, especially at the boundaries between objects. Therefore, we follow Xiao et al. [XCS+06] and visualize the flow fields by a color circle, in which complementary colors are located opposite from each other (see circle in figure 1.3b).



(a) (b)

Figure 1.3: Encoding flow fields. (a) Vector field representation. (b) Color encoded representation.

The color in the circle encodes the direction of the flow vector, where the brightness refers to the magnitude. Figure 1.3b illustrates the flow field of the same scene using our color encoding. Apparently, this representation is more appropriate for our purposes.

1.5 Structure

This diploma thesis is structured as follows. In the next chapter we give a short overview over the most common techniques to determine optical flow. In chapter 3 we explain our basic optical flow algorithm in detail, which we enrich with depth discontinuities and surface normals. We introduce the detection of these two information channels in chapter 4. In chapter 5 we give a detailed evaluation of our approach according to synthetic and natural images. Furthermore, we present how our extended optical flow estimation can be incorporated into the field of Image Based Relighting. The thesis closes with a summary and an outlook to future work.

Chapter 2

Optical Flow

In the last 25 years lots of algorithms have been developed which compute optical flow fields in different ways and intend to cope with the issues explained before. In this chapter we present established techniques to determine the optical flow.

2.1 Regression Approaches

First, we introduce the *regression approaches* which determine the flow vector for a particular pixel by fitting a parametric motion model to a small spatial neighborhood.

Since this usually yields a linear system, which can easily be solved, it is common to approximate equation 1.1 by the first order Taylor expansion [HS81]:

$$\mathbf{I}(x,y,t) \quad = \quad \mathbf{I}(x,y,t) + \mathbf{I}_x \delta x + \mathbf{I}_y \delta y + \mathbf{I}_t \delta t + \varepsilon \tag{2.1}$$

Ignoring the high-order terms ε, subtracting $\mathbf{I}(x,y,t)$ on both sides and dividing by δt yield the famous *optical flow constraint*:

$$\begin{aligned} 0 &= \mathbf{I}_x \cdot u + \mathbf{I}_y \cdot v + \mathbf{I}_t \\ \Leftrightarrow \quad 0 &= \nabla \mathbf{I}^T \mathbf{u} + \mathbf{I}_t \end{aligned} \tag{2.2}$$

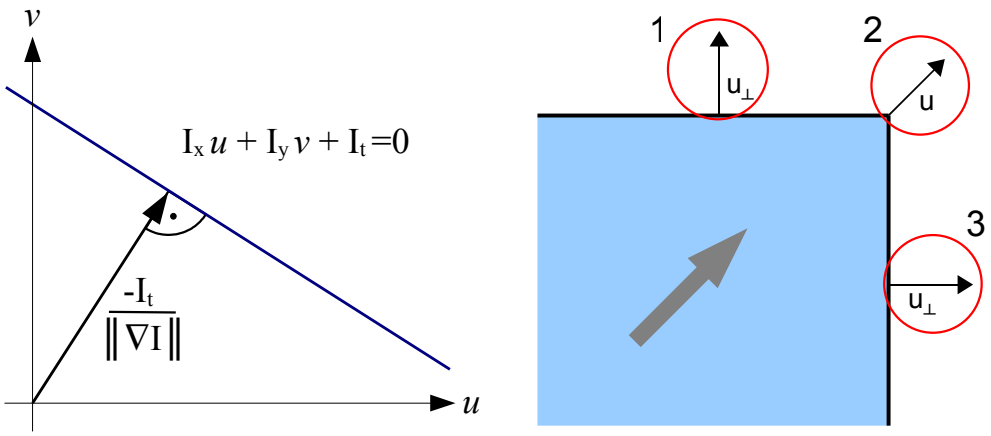
where $\mathbf{u} = (u \;\; v)^T$. The image derivatives can be computed using a standard discretization

$$\begin{aligned} \nabla \mathbf{I}(x,y,t) &:= \begin{bmatrix} \mathbf{I}(x+1,y,t) - \mathbf{I}(x,y,t) \\ \mathbf{I}(x,y+1,t) - \mathbf{I}(x,y,t) \end{bmatrix} \\ \mathbf{I}_t(x,y,t) &:= \mathbf{I}(x,y,t+1) - \mathbf{I}(x,y,t) \end{aligned}$$

The problem is ill-posed at this point, because equation 2.2 has two unknowns, which constrain the vector to lie on a line. As illustrated in figure 2.1a, only the component of the flow vector in the direction of the image gradient can be determined:

$$\mathbf{u}_\perp \quad = \quad \frac{-\mathbf{I}_t}{\|\nabla \mathbf{I}\|} \cdot \nabla \mathbf{I} \tag{2.3}$$

However, the flow vector component along the image contours, i.e. perpendicular to the gradient, can not be computed. This results in the *aperture problem* [Hor86] as shown in figure 2.1b. The issue becomes even worse if the intensity gradient vanishes somewhere. In this case no flow vector can be determined. Therefore, more constraints are necessary to retrieve a unique solution.



(a) (b)

Figure 2.1: (a) The linearized brightness constancy assumption constrains the flow vector to lie on a line perpendicular to the image gradient. (b) The aperture problem: A rectangle (blue) is moving to the top right. Using the linearized brightness constraint, the flow vectors can only be determined in the direction of the image gradient (1 and 3) as viewed through small apertures (red circles). Only in situation 2 the flow vector can fully be determined, since its direction matches the image gradient.

The approach by Lucas & Kanade [LK81] regards more surrounding pixels by assuming that the flow vectors are constant within a local neighborhood. The authors retrieve the flow vector for a pixel by minimizing

$$E(u,v) \quad = \quad \sum_{\mathbf{x}\in R} W(\mathbf{x}) \cdot \rho\left(\nabla \mathbf{I}(\mathbf{x})^T \cdot \mathbf{u} + \mathbf{I}_t(\mathbf{x})\right) \tag{2.4}$$

where R denotes a spatial neighborhood around the pixel and $W(\mathbf{x})$ is a window function that weights the influence of each pixel. Usually one applies a Gaussian to give central constraints a stronger influence. $\rho(s) := s^2$ is a least-squares estimator, i.e. deviations from the brightness constancy model are squared. We generalize our problem to a more complex motion model:

$$\mathbf{u}(x,y) \quad = \quad \mathbf{u}(x,y;\mathbf{a})$$

where finding the solution corresponds to determining the motion parameters $\mathbf{a}$ for a given pixel.

The generalized version of equation 2.4 then reads:

$$E(u,v) = \sum_{\mathbf{x}\in R} W(\mathbf{x}) \cdot \rho\left(\nabla \mathbf{I}^T(\mathbf{x}) \cdot \mathbf{u}(\mathbf{x};\mathbf{a}) + \mathbf{I}_t(\mathbf{x})\right) \tag{2.5}$$

If we assume a constant motion

$$\mathbf{u}(x,y;\mathbf{a}) := \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

then the minimum of equation 2.5 is found by setting the derivatives with respect to u and v to zero:

$$\begin{aligned} \frac{\delta E(u,v)}{\delta u} &= \sum_{\mathbf{x}\in R} W(\mathbf{x}) \cdot (u\mathbf{I}_x^2 + v\mathbf{I}_x\mathbf{I}_y + \mathbf{I}_x\mathbf{I}_t) = 0 \\ \frac{\delta E(u,v)}{\delta v} &= \sum_{\mathbf{x}\in R} W(\mathbf{x}) \cdot (v\mathbf{I}_y^2 + u\mathbf{I}_x\mathbf{I}_y + \mathbf{I}_y\mathbf{I}_t) = 0 \end{aligned}$$

These equations can furthermore be written as a linear map:

$$M\mathbf{u} = b$$

where

$$M = \begin{bmatrix} \sum\limits_{\mathbf{x}\in R} W(\mathbf{x})\mathbf{I}_x^2 & \sum\limits_{\mathbf{x}\in R} W(\mathbf{x})\mathbf{I}_x\mathbf{I}_y \\ \sum\limits_{\mathbf{x}\in R} W(\mathbf{x})\mathbf{I}_x\mathbf{I}_y & \sum\limits_{\mathbf{x}\in R} W(\mathbf{x})\mathbf{I}_y^2 \end{bmatrix} \tag{2.6}$$

$$b = -\begin{pmatrix} \sum\limits_{\mathbf{x}\in R} W(\mathbf{x})\mathbf{I}_x\mathbf{I}_t \\ \sum\limits_{\mathbf{x}\in R} W(\mathbf{x})\mathbf{I}_y\mathbf{I}_t \end{pmatrix}$$

If M is invertible (has rank 2), the flow vector is obtained by $\mathbf{u} = M^{-1}b$. Otherwise, the flow vector can not be uniquely determined, which is just another occurrence of the aperture problem.

An advantage of this approach is that it provides a *confidence measurement*: The smallest eigenvalue λ_1 of M determines the quality of the flow vector estimation. If λ_1 is small, then the local structure within the search window is insufficient to determine the 2D flow vector (aperture problem). A high eigenvalue identifies a more reliable flow vector. Confidence measurements are very helpful for *refinement techniques*, e.g. if a flow field is used as an initial estimate for a subsequent flow field.

More complex motion models like the affine model

$$\mathbf{u}(x,y;\mathbf{a}) := \begin{pmatrix} a_1 + a_2 \cdot x + a_3 \cdot y \\ a_4 + a_5 \cdot x + a_6 \cdot y \end{pmatrix}$$

induce more unknowns which need to be determined and, therefore, a bigger spatial neighborhood is required to constrain the solution sufficiently.

Other authors added the second order derivatives to extend equation 1.1 in order to retrieve a clearly constrained solution. Uras et al. [UGVT89] and Nagel [Nag83b] introduced the constraint:

$$0 = (\nabla\nabla\mathbf{I})^T\mathbf{u} + \nabla\mathbf{I}_t \tag{2.7}$$

Nagel pointed out that the aperture problem completely disappears at image corners if second-order derivatives are used. A drawback of high-order derivatives is that they are usually very sensitive to noise.

We refer to equations like 2.4 as *energy functions*: the stronger a flow vector deviates from a model assumption (in this case the brightness constancy assumption), the more *energy* it adds. One normally says that one *penalizes* deviations from the model constraints with a higher energy. Therefore, the best solution is the minimum of the energy function.

As all approaches, which incorporate a local neighborhood R, do, the regression approaches suffer from the same problem: if R is too small, the correct solution is under-constrained, more sensitive to noise, or the aperture problem occurs. But the bigger the included neighborhood is, the more likely the model assumptions are violated. This is especially the case if the image sequence contains multiple opposed motions. If we try to find the flow vector at a motion discontinuity, i.e. at the boundary between two differently moving objects, using a large spatial neighborhood of pixels produces wrong results if these discontinuities are not explicitly taken into account.

Schunck [Sch89] developed a method called **Constraint Line Clustering**, which intends to allow motion discontinuities. As mentioned earlier equation 2.2 constrains the solution to lie on a line. For a given pixel $\mathbf{x}$, Schunck calculates all constraint lines within a local neighborhood R. These lines are intersected with the constraint line at the center $\mathbf{x}$. The authors then statistically determine the smallest cluster which contains at least half of the intersection points. This cluster represents the dominant motion. The approach is illustrated in figure 2.2.

Similar to our idea, Woodham [Woo92] published an optical flow algorithm, which detects the optical flow field by the use of multiple lighting conditions. He captures the scene lit with a light source from (at least) three directions, and then obtains an overconstraint system of equations containing a linearized optical flow constraint for each lighting condition:

$$\begin{aligned} 0 &= \mathbf{I}_x^1 \cdot u + \mathbf{I}_y^1 \cdot v + \mathbf{I}_t^1 \\ 0 &= \mathbf{I}_x^2 \cdot u + \mathbf{I}_y^2 \cdot v + \mathbf{I}_t^2 \\ 0 &= \mathbf{I}_x^3 \cdot u + \mathbf{I}_y^3 \cdot v + \mathbf{I}_t^3 \end{aligned}$$

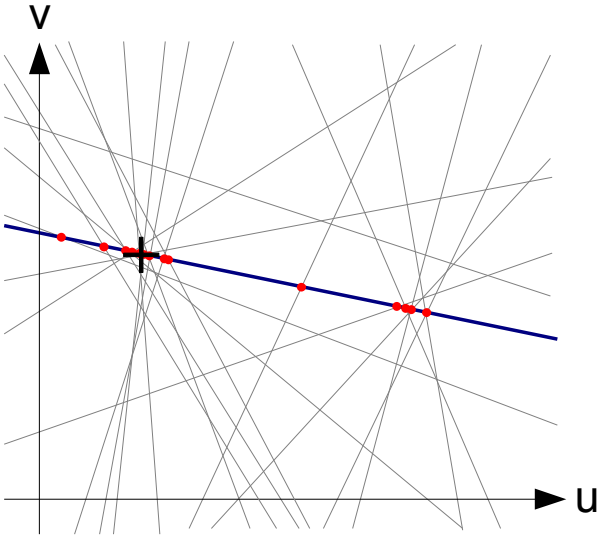


Figure 2.2: Constraint line clustering. Firstly all constraint lines in a local neighborhood are calculated (gray lines). These lines are then intersected with the constraint line in the center of the neighborhood (blue). The intersections are marked as red dots. The dominant motion (cross) is then found by a clustering strategy.

where $\mathbf{I}_*^i$ refers to the image sequence taken under the i-th lighting condition. The system of equations is solved with the standard least-squares technique. Even though the approach provides a convenient analytic solution for each pixel, it is not reliable enough for practical use: if a pixel lies on a surface discontinuity or a surface marking, all lighting conditions create a similar pixel value, which "is dominated by the scene features largely independent of the direction of illumination" [Woo92].

2.2 Global Smoothness Approaches

The approaches, which are presented in this section, explicitly demand an explicit smoothness constraint as explained in the introduction. In their pioneer paper Horn & Schunck [HS81] solved the aperture problem by including a *global smoothness constraint*, a so-called *regularizer*, which demands that the flow field varies *smoothly*. The energy which has to be minimized to obtain the final flow field reads:

$$E(u,v) = \int_{\mathbf{x}\in\Omega} \underbrace{\rho\left(\nabla\mathbf{I}^T\mathbf{u}+\mathbf{I}_t\right)}_{\text{data}} + \alpha\cdot\underbrace{\rho\left(|\nabla u|^2+|\nabla v|^2\right)}_{\text{smoothness}} d\mathbf{x} \tag{2.8}$$

The left term, the *data energy*, is known from above and represents the linearized brightness constancy assumption. The right term, the global smoothness constraint, penalizes the gradient of the flow field and represents the *smoothness*

energy: the greater the flow gradient at a particular pixel, the higher the resulting energy. The *smoothness weight* $\alpha > 0$ determines the importance of the smoothness constraint. A higher smoothness weight yields a smoother flow field. The closer this value is to zero, the less important becomes the smoothness. The *statistic* $\rho(s) := s^2$ squares all deviations from the model assumption. Therefore, again we search for the least-squares solution of the problem.

An equation like 2.8 is called an *energy functional*, since it is a function (energy) over a function (flow field which maps pixel positions to flow vectors). Rather than independently finding a flow vector for each pixel, we *optimize* this equation in order to obtain the entire flow field *function*. In general, solving an energy functional is non-trivial, since neighbored flow vectors depend on each other and the functional may contain non-linear terms. Due to the *calculus of variations* [Arf85], a functional as in equation 2.8 in the form of $E(\mathbf{u}) = \int F(\mathbf{x}, \mathbf{u}, \nabla\mathbf{u})d\mathbf{x}$ is minimized by solving the Euler-Lagrange equations:

$$\frac{\delta F}{\delta \mathbf{u}} - \frac{d}{d\mathbf{x}} \cdot \frac{\delta F}{\delta \nabla \mathbf{u}} = 0 \tag{2.9}$$

Therefore, optical flow approaches which are based on minimizing an energy functional are classified as *variational approaches*. In the last years these approaches became very popular for two reasons. First, they allow a transparent modelling of the flow constraints by construction: all assumptions are incorporated in the energy functional and no intermediate or post processing, which could bias the consistency of the whole approach, is required. Second, they yield a *dense* flow field: even if insufficient local structure is available around a particular pixel, the global smoothness allows the estimation of a displacement by propagating the flow vectors from nearby pixels.

Due to the linearization of the data term and the choice of $\rho(s) := s^2$, Horn & Schunck reduce the minimization process to a linear system, which can be solved by the Gauss-Seidl or SOR algorithm (see appendix A). The approach provides a compromise between the matching of intensity values (data) and a spatial coherent flow field (smoothness). Hence it solves the aperture problem, because flow vectors are propagated to their neighbors. Moreover, the smoothing reduces the impact of noise.

A major drawback of the approach by [HS81] is that it generally ignores motion discontinuities in the image sequence. The *homogeneous* smoothness regularizer is applied everywhere regardless of the local geometric structure. Therefore, the flow field is *oversmoothed* at motion boundaries resulting in noticeable errors in these areas. Apart from this issue, violations of the data constraint like occlusions, (dis)appearances, etc. are not taken into account. The original approach just expects *one* dominant motion and does *not* allow any violations of the model assumptions.

There are basically two ways to deal with model violations. Either the deviations are explicitly modeled, or the existing model is modified in a way that outliers,

e.g. two adjacent discontinuous flow vectors at a motion boundary, are tolerated. There are many approaches which face the problems of violations. Some of the most important ones are presented in the following.

2.2.1 Image-Driven Regularizers

The image-driven regularizers extend the global smoothness term of the energy functional by the consideration of the local image structure: the smoothness is lowered at intensity edges. Two different regularizers are explained in this section. Figure 2.3 illustrates these approaches.

The **isotropic image-driven regularizer** adapts the smoothness term to the intensity images. The higher the intensity gradient is at a specific position, the lower the smoothness is demanded. Figure 2.3c visualizes this idea. A common regularizer of this type is

$$\int_{\mathbf{x}\in\Omega} \alpha\left(\rho(\|\nabla \mathbf{I}\|)\right) \cdot \rho\left(|\nabla u|^2 + |\nabla v|^2\right) d\mathbf{x}$$

where $\alpha(x)$ denotes a decreasing, strictly positive function that weights the smoothness term according to the magnitude of the image gradient $\|\nabla \mathbf{I}\|$. The simplest weighting function is $\alpha(x) := \frac{1}{x}$. Sand et al. [ST06] introduced the concept of *local smoothness*, which is explained in section 3.4 and follows the same idea.

Nagel [Nag83a] introduced the first **anisotropic image-driven regularizer**. Rather than reducing the weight of smoothness at intensity edges they apply an *orientated* smoothness regularizer, where non-smoothness is penalized *alongside*, but *not over* edges (see figure 2.3d). Therefore the smoothness constraint is *aligned* to the intensity edge: the smaller the angle between the image gradient and a flow vector, the higher the corresponding smoothness energy. The regularizer by Nagel reads:

$$\int_{\mathbf{x}\in\Omega} \rho\left((\nabla \mathbf{u})^T D(\nabla \mathbf{I}) \nabla \mathbf{u}\right) d\mathbf{x}$$

where $\nabla \mathbf{u}$ is the gradient of the flow vector and D is the *anisotropic diffusion tensor* defined as:

$$D(\nabla \mathbf{I}) \quad = \quad \frac{1}{\|\nabla \mathbf{I}\|^2 + 2\upsilon^2} \left(\nabla \mathbf{I}^{\perp} \nabla \mathbf{I}^{\perp T} + \upsilon^2 \mathbf{1}\right)$$

$\nabla \mathbf{I}^{\perp}$ is the vector perpendicular to the image gradient, $\mathbf{1}$ denotes the unity matrix and υ controls the degree of isotropy.

The basic issue of image-driven regularizers is the assumption that intensity edges correspond to object boundaries, which is hardly the case in general. This is explained in section 3.4 in more detail.

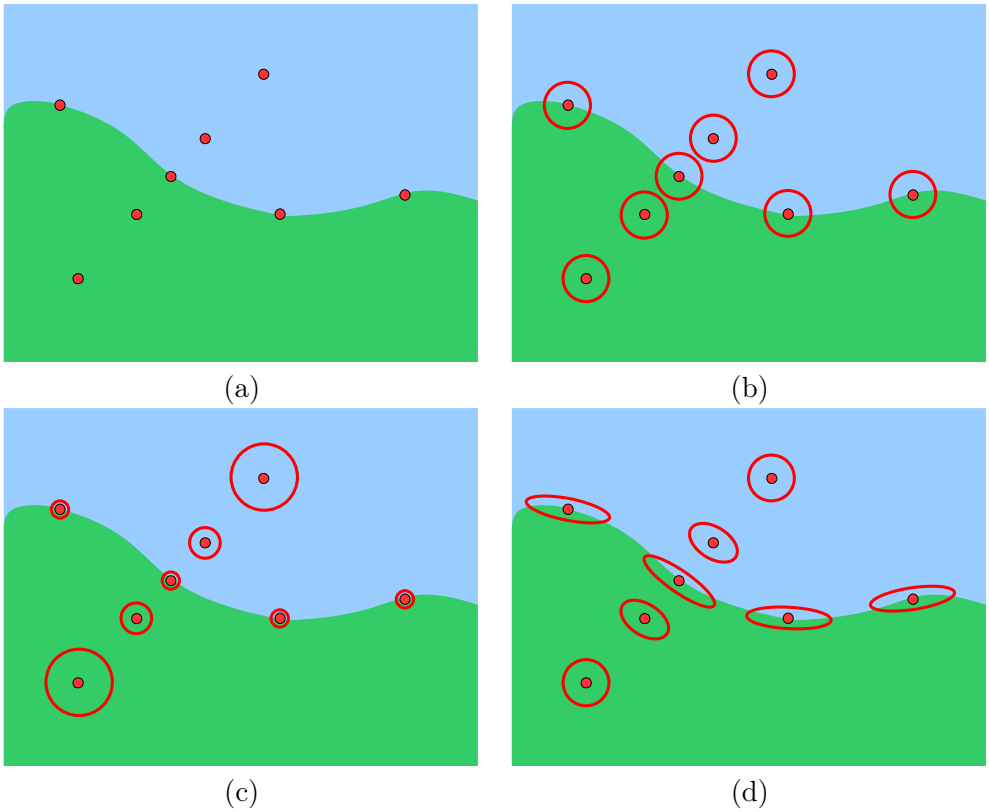


Figure 2.3: Image driven regularizers. (a) Given an image with two distinct areas we illustrate the filter kernel at particular points. (b) The approach by Horn & Schunck applies the same homogeneous smoothing for all points and therefore oversmooths boundaries. (c) An isotropic image-driven regularizer decreases the smoothing at image gradients but still oversmooths edges. (d) The anisotropic image-driven regularizer aligns the smoothing kernel to the image gradient. At intensity boundaries smoothing is large alongside the edges but small over the edges.

2.2.2 Robust Statistics

Most optical flow algorithms apply the quadratic statistic $\rho(s) := s^2$ to their model deviations in order to find a least-squares solution for the flow errors. This is convenient since $\rho(s)$ is convex and the minimization of the energy functionals can usually be reduced to a linear system of equations, which can easily be solved by standard techniques. The downside of the least-squares approach is the requirement that all deviations from the model assumption are Gaussian distributed. This is nearly *never* the case. The quadratic *penalizer* gives outliers being far away from the solution a strong impact. Thus, the solution is adapted to be closer to these extreme deviations. Least-squares approaches simply do not *accept* that the model assumptions may be violated by outliers [Bro05].

Unfortunately, outliers are very common in video sequences. Motion discontinuities are one example: flow vectors at motion boundaries of two oppositely moving objects have a high gradient and violate all smoothness constraints, as explained above. In these cases it would be helpful to identify and exclude these outliers from the minimization process. The problem can be compared to fitting a line to a set of points (see figure 2.4). If the set contains two distinct clusters, the least-squares fit produces a line that finds a compromise between the dominant cluster and the smaller cluster of outliers. In our example the line models none of the two point aggregations. The right solution would ignore the outliers (top left) and only take the dominate bulk into account.

Black & Anandan [BA93] replace the quadratic penalizer with a *robust statistic*. One of the possible functions is shown in figure 2.5b. Where the influence of outliers increases linearly and without bound with the size of the deviation using a least-squares estimator (figure 2.5a), *robust estimators* decrease the weight of outliers beyond a given threshold. A common robust estimator is the *Lorentzian* function: deviations from the model are squared up to a certain point. Beyond that, their impact is reduced (figure 2.5b).

The robust statistic penalizers allow outliers in the data term like occlusions, disappearances and brightness changes as well as outliers in the smoothness assumptions like motion discontinuities. Although the introduction of robust statistics to the energy functionals highly improves the flow estimation, minimizing the Euler-Lagrange equations becomes much harder due to the nonlinearity of these functions.

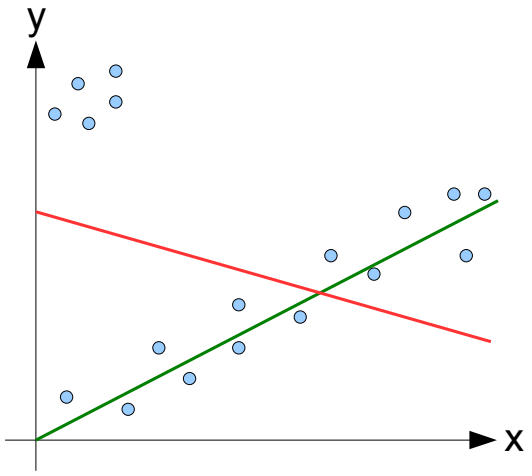


Figure 2.4: Fitting a line through a set of points. The least-squares fit gives outliers a strong influence. The resulting line (red) does not model the bulk of data. The solution should fit the dominant cluster of points and ignore the outliers (green line).

Global smoothness constraints using a robust statistic are often referred to as **isotropic flow-driven regularizers**. However, robust statistics are not limited to global smoothness approaches. They can be applied to all techniques which penalize deviations from the model assumptions.

2.2.3 Explicit Discontinuities

Line processes explicitly model discontinuities in the flow field and consider images as dual $n \times n$-lattices using the following notation. Let Ω be a the set of pixels, then $G := \{(s,t)|s,t \in \Omega\}$ denotes all pairs of adjacent pixels and $R(s) := \{t|(s,t) \in G\}$ contains all neighbors (adjacent pixels) of s. The *line process* $l := \{l_{s,t}|(s,t) \in G\ \}$ with $l_{s,t} \in \{0,1\}$ describes whether there is a motion discontinuity between pixel s and t ($l_{s,t} = 1$) or not ($l_{s,t} = 0$) [GG88]. See figure 2.6 for an illustration.

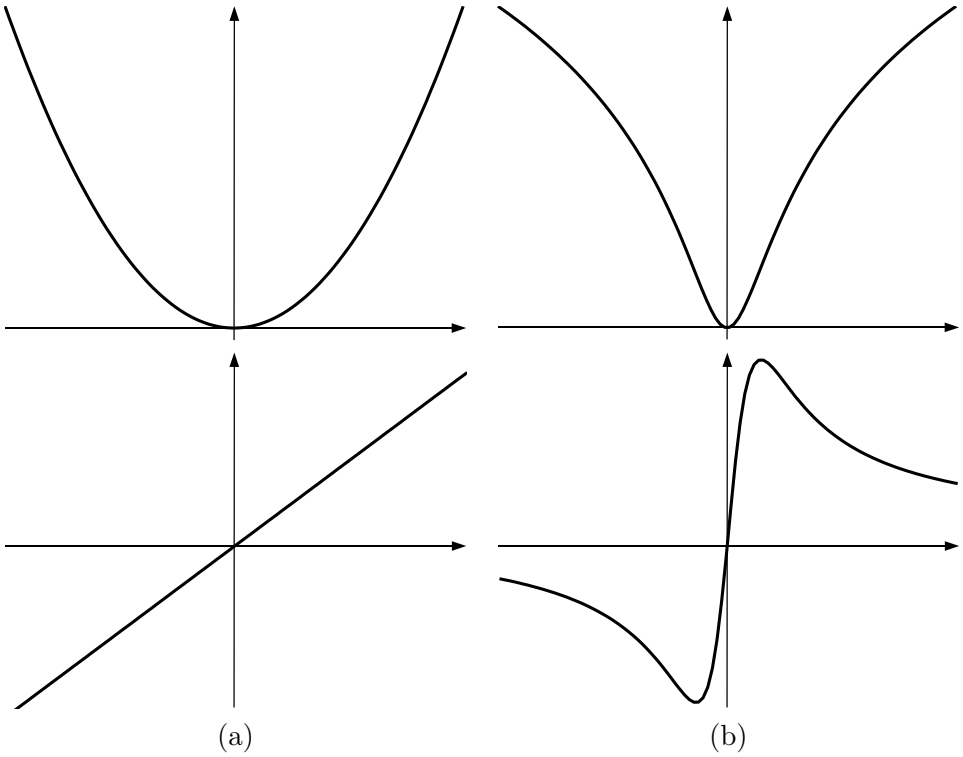


Figure 2.5: Comparison of least-squares with robust Lorentzian estimator. Top: Estimator function. Bottom: Derivation. (a) The least-squares estimator $\rho(x) := x^2$. Errors are squared (top) and the influence of outliers increases linearly with the deviation from the model assumption (bottom). (b) The robust Lorentzian estimator $\rho(x) := \log\left(1 + \frac{1}{2}\left(\frac{x}{\sigma}\right)^2\right)$ (top). The influence of model deviations increases up to a certain threshold. Beyond this threshold deviations are considered as outliers and their contribution to the error is reduced (bottom).

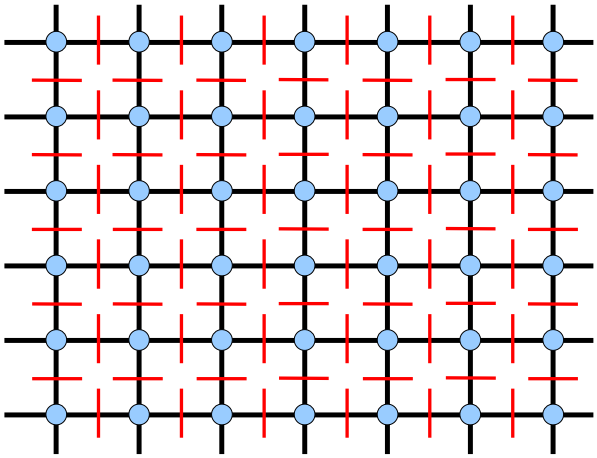


Figure 2.6: Line processes. The image is considered as lattice of pixels, where the edges correspond to direct neighborhoods. The line processes (red lines) determine if there is a discontinuity between two adjacent pixels.

According to German & German [GG88] the energy that is to be minimized reads:

$$E(\mathbf{u}, l) = \sum_{s \in \Omega} (\underbrace{E_D(\mathbf{u}_s)}_{\text{data}} + \underbrace{\sum_{t \in R(s)} \left[(1 - l_{s,t}) \|\mathbf{u}_s - \mathbf{u}_t\|^2 + \alpha \cdot l_{s,t} \right]}_{\text{smoothness}}) \tag{2.10}$$

The energy is summed over all pixels ($s \in \Omega$). For each pixel, the data and the smoothness energy is accumulated. Again $E_D(\mathbf{u}_s)$ represents the data conservation constraint, i.e. the less the intensity of pixels match, the higher the energy is. The smoothness term is more complex. For each neighbor t of the current pixel s, the authors compute the corresponding energy as follows. If there is no discontinuity between s and t (i.e. $1 - l_{s,t} = 1$), the magnitude of the flow gradient is penalized, as explained before. Otherwise, the flow gradient at the potential discontinuity between the two pixels is ignored ($1 - l_{s,t} = 0$). The last expression in the smoothness term penalizes the occurrence of a discontinuity because discontinuities are considered rare. Note that equation 2.10 is unknown in the field *and* in the line processes (the discontinuities), which complicates the minimization process.

Xiao et al. [XCS+06] incorporate an explicit **occlusion term** into the data energy and apply a novel **adaptive bilateral regularizer**. The first version of their energy functional reads:

$$E(\mathbf{u}) = \sum_{\mathbf{x} \in \Omega} (E_d(\mathbf{u}) + E_s(\nabla \mathbf{u})) \cdot O(\mathbf{u}) + E_{oc} \cdot (1 - O(\mathbf{u}))\, d\mathbf{x} \tag{2.11}$$

$E_d(\mathbf{u}) = (\nabla \mathbf{I}^T + \mathbf{I}_t)^2$ is the common optical flow constraint, i.e. the linearized version of the brightness constancy assumption, and $E_s(\mathbf{u}) = \left((\nabla \mathbf{u})^T D(\nabla \mathbf{I}) \nabla \mathbf{u}\right)$ refers to the anisotropic image-driven regularizer by Nagel explained above. $O(\mathbf{u})$ determines whether a particular pixel $\mathbf{x}$ is occluded or not:

$$O(\mathbf{u}) \quad := \quad \begin{cases} 0 & \text{if } (I_1(\mathbf{x}) - I_2(\mathbf{x}+\mathbf{u}))^2 > \varepsilon_O \\ 1 & \text{otherwise} \end{cases}$$

A pixel in the first image is declared as "occluded" if the intensity difference to the corresponding pixel in the second image exceeds a given threshold ε_O. Equation 2.11 splits the energy in occluded and non-occluded energy. If a pixel is not occluded ($O(\mathbf{u}) = 1$), its flow vector is penalized according to the mentioned data and smoothness constraints. Otherwise, just a constant value E_{oc} is added to penalize the occurrence of an occlusion.

Unlike other authors, Xiao et al. split up the minimization process into a strict two-stage loop: first, the data energy is minimized, then the optical flow field is smoothed. Furthermore, they replace the smoothness term of Nagel with a bilateral filter, which preserves intensity and motion discontinuities and excludes occluded pixels from the smoothing operation. Each flow vector is a weighted average of its neighbors, where the weights are determined by the convolution of several Gaussians. The filtered flow vector $\mathbf{u}'$ of $\mathbf{u}$ at pixel $\mathbf{x}$ is computed by:

$$\mathbf{u}'(\mathbf{x}) \quad := \quad \frac{1}{\omega} \int\limits_{\mathbf{x}' \in \Omega} \mathbf{u}(\mathbf{x}') \cdot \mathcal{N}_s(\mathbf{x} - \mathbf{x}'; \sigma_s) \cdot \mathcal{N}_I(I_1(\mathbf{x}) - I_2(\mathbf{x}'); \sigma_I) \cdot \mathcal{N}_u(\mathbf{u}(\mathbf{x}) - \mathbf{u}(\mathbf{x}'); \sigma_u) \cdot O(\mathbf{x}')$$

$\mathcal{N}_s$ is the Gaussian that weights nearer flow vectors stronger than farther ones. The support of $\mathcal{N}_s$ is adaptive to make sure that always non-occluded pixels are taken into account. $\mathcal{N}_I$ reduces the influence of intensity-dissimilar pixels. $\mathcal{N}_u$ preserve motion discontinuities, i.e. it reduces the influence of an adjacent flow vector if it strongly differs from the central one. Finally O excepts all neighbors which are occluded. σ_s, $\sigma_\mathbf{I}$, σ_u refer to the variances of the Gaussians, which need to be chosen carefully. Figure 2.7 illustrates the idea of this adaptive filter, which is both image and flow-driven.

Even though Xiao et al. seem to respect most of the issues of optical flow, we consider the strict separation of data and smoothness minimization as a major issue: if the data minimization creates a wrong initial guess of the flow vectors, these errors might be kept or even amplified by the bilateral filter due to the intensity and motion discontinuity preserving filter. As an example, we assume two adjacent flow vectors on a rigid, but high-frequency textured model. The data minimization could find a "match" of both pixels in opposite directions, although they should move in the same direction. The bilateral filter will not effectively smooth out this mistake since it preserves motion discontinuities. Unfortunately, the authors of the paper do not point out how they weight between the data and the smoothness constraint. Sand et al. [ST06] use this bilateral

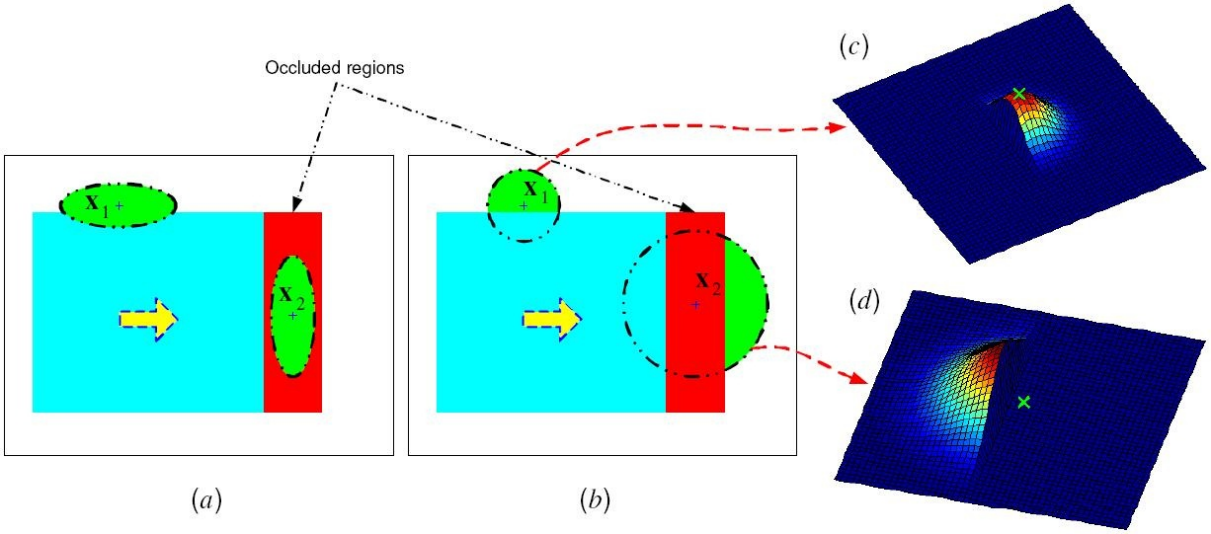


Figure 2.7: Bilateral adaptive filter by Xiao et al. [XCS+06]. A rectangle (cyan) moves from the left to the right and occludes a strip of pixels (red). The green areas illustrate the pixels which are taken into account for smoothing. (a) Two typical situations where anisotropic image driven filters fail. The shear boundary is oversmoothed at x_1. Moreover, the filter does only take occluded pixels into account at x_2 and therefore produces a wrong regularization. (b) The adaptive filter excludes all pixels at x_1 from the filter kernel, which intensities are dissimilar from the center pixel. The kernel of x_2 is increased in order to include enough non-occluded pixels for a correct estimated. Again intensity dissimilar pixels are omitted. (c)-(d) 3D plot of the filter kernels. The Gaussian shape results from the spatial filter.

filter as a post-process but limit its application to flow boundaries, which they identify by the magnitude of the flow gradients.

2.2.4 Multi-Scale Approaches

An implicit assumption of the *optical flow constraint* (equation 2.2) is that the displacements are small, strictly speaking, less than one pixel. Energy functionals, as explained in this chapter, are *multi-modal functional*, i.e. they have more than one local minimum. Common iterative solvers, like the gradient descent method or fixed point iterations, run the risk of being trapped into a local minimum rather than finding the global one (see figure 2.8a). Therefore, these solvers would not be able to cope with large displacements.

A common technique to avoid that problem is the use of *continuation* or *graduated non-convexity (GNC)* methods [BZ87]: Instead of searching the minimum for the original functional, it is smoothed several times in order to create a series of functionals, in which each functional is smoother than its predecessor.

High-frequent content is removed from the signal and after several iterations a version is reached, which only contains one unique minimum. In this version the minimum is easily detected by an iterative solver. The approach continues with the prior, less smoothed functional and takes the previously found minimum as an initial estimate. This process is continued until the global minimum is found in the original functional. The figures 2.8b-c illustrate this technique. These methods usually work quite well, even though there are cases where they fail (see figure 2.8d).

In case of finding the global minimum of an energy functional, applying the *GNC method* implies that smoother versions of the intensity function, i.e. of the input images, need to be created. One way to do that is to apply Gaussian filters for each smoothing step. Thereafter we would start the flow determination between the most smoothed images and take the resulting flow field as an initial guess for the next less smoothed versions.

Since filtering the original images is expensive concerning run-time and memory, it is common to use a downsampling strategy by creating a Laplacian pyramid [BA87]. Rather than filtering the intensity images, they are scaled down multiple times. First, an *image pyramid* is created for all input images containing n

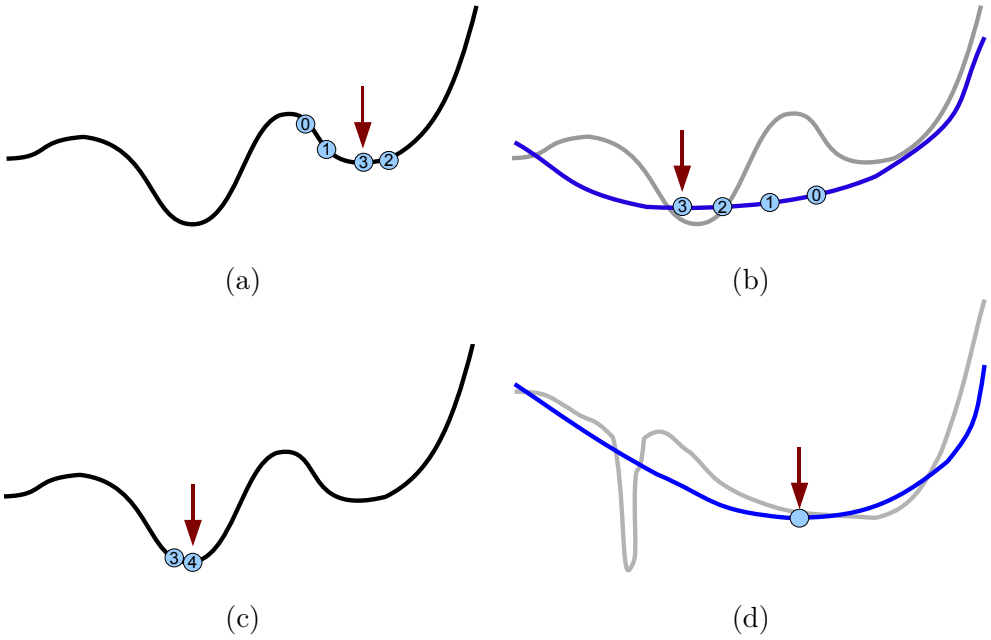


Figure 2.8: Idea of graduated non-convexity (GNC). (a) A simple iterative solver can easily trap into a local minimum when starting with an inappropriate initial estimate (0). (b) GNC firstly searches for the minimum for a smoothed version (blue) of the original function (gray). (c) Accordingly, the minimum of the smoothed function is used as an initial estimate (3) for the finer version. The global minimum (4) is found. (d) Bad case where GNC fails.

levels, where each level represents a downsampled version of the previous one. Then the flow field at the *coarsest* level 0 is detected, i.e. the energy functional between the images of the lowest resolution is minimized. After this the resulting flow field is scaled up to the next finer resolution, where it gives the initial estimate for the flow field on the next level. This process is repeated until the flow field for the finest level $n-1$ is obtained.

Multi-scale approaches are very common in todays optical flow algorithms and effectively help to cope with large displacements in image sequences.

2.2.5 Minimization via Fixed Point Iterations

In 2004 Brox et al. [BBPW04] developed an optical flow algorithm which combined an energy functional like [HS81] with a robust statistic and a multi-scale approach. The actual contribution of their approach is a novel minimization process, which completely refrains from a linearization of the brightness constraint as in equation 2.2, but minimizes the energy functional using two nested fixed point iterations. The approach currently represents one of the most accurate optical flow algorithms known in literature. We explain this technique in chapter 3 in detail.

Slesareva et al. [SBW05] extended the paper to stereo matching by adding an epipolar constraint to the energy functional and Bruhn et al. [BW05] ported the algorithm to the GPU in order to yield real-time performance. Amiaz et al. [AK06] combined the data and the smoothness constraint in the energy functional with a level-set approach, which segments the optical flow into two distinct fields while minimizing the data and smoothness energy. Later, Brox at al. [BBW06] extended that approach by providing an arbitrary number of segments.

2.3 Region-Based Matching

Another class of local optical flow algorithms are named *region-based matching approaches*. They are similar to the regression-based approaches. However, rather than setting up a system of equations they *match search windows* in order to find the correct flow vector for a particular pixel. This avoids the imprecise calculation of image derivatives.

Let I_1, I_2 be two frames between which the flow field is to be determined. For a given pixel $\mathbf{x}$ Anandan [Ana89] finds the integer displacement $\mathbf{d} = (\mathbf{d}_x, \mathbf{d}_y)$ by minimizing

$$\begin{aligned} SSD(\mathbf{x};\mathbf{d}) &:= \sum_{j=-n}^{n}\sum_{i=-n}^{n} W(i,j)\cdot\rho\left[I_1(\mathbf{x}+(i,j)) - I_2(\mathbf{x}+\mathbf{d}+(i,j))\right] \\ &= W(\mathbf{x}) * \rho\left[I_1(\mathbf{x}) - I_2(\mathbf{x}+\mathbf{d})\right] \end{aligned} \tag{2.12}$$

Again W is a weighting function for a window of size n, which usually weights central points stronger. Evaluating $SSD(\mathbf{x};\mathbf{d})$ in the local neighborhood results in a so-called *sum-of-squared differences (SSD) surface* which maps displacements to deviation errors (see figure 2.9). The optimal solution is found at the minimum of the surface.

Anandan utilizes a multi-scale approach (see section 2.2.4). He first starts to search for the flow vectors at a coarse scale of the input images, where displacements are assumed to cover one pixel or less. The results are used as initial estimates for the next finer scale. This is repeated until the finest image version is reached. In order to allow sub-pixel displacements as well, the SSD surface around the resulting integer displacement $\mathbf{d}$ is approximated by a quadratic function. Then the final flow vector $\mathbf{u}_0$ is obtained by the minimum of this approximation.

The approach derives confidence measurements from the principal curvatures c_{min} and c_{max} of the SSD surface, i.e. a high curvature corresponds to a high confidence. Thereafter these are used to perform a post-smoothing of the entire flow field. The smoothed flow field is obtained by minimizing the energy functional in order to find the final displacements $\mathbf{u}$:

$$\begin{aligned} E(u,v) \quad = \quad & \int_{\mathbf{x}\in\Omega} \rho\left(|\nabla u|^2 + |\nabla v|^2\right) \\ & + \; c_{max} \cdot \rho\left(\mathbf{u}\cdot c_{max} - \mathbf{u}_0 \cdot c_{max}\right) \\ & + \; c_{min} \cdot \rho\left(\mathbf{u}\cdot c_{min} - \mathbf{u}_0 \cdot c_{min}\right) d\mathbf{x} \end{aligned}$$

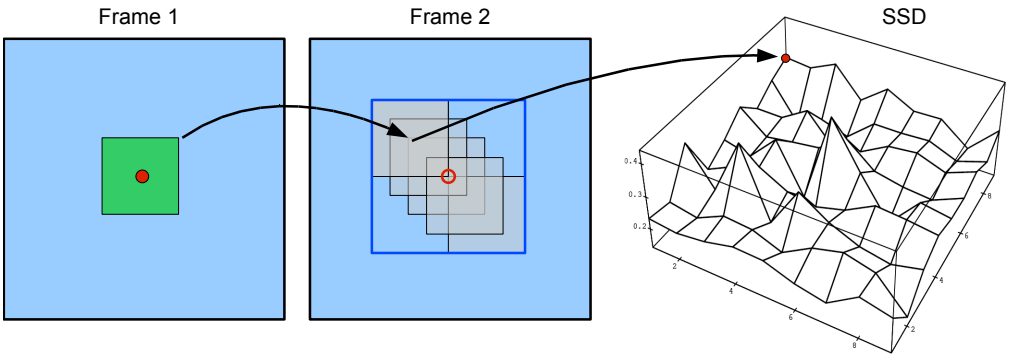


Figure 2.9: For a given pixel (red dot) in the first image a SSD surface is obtained by finding a match for a *source area* around the pixel (green rectangle). In the second frame the distances between the source area and destination areas within a certain search window (blue frame) are computed. For each center position of the destination areas a value is obtained characterizing the error of the match. This yields a two-dimensional image which maps displacements to error distances (right plot). The best displacement is the minimum of that map.

Anandan apparently combines the homogeneous smoothness term of Horn & Schunck [HS81] (equation 2.8) with a term that penalizes deviations from the principal curvatures of the SSD surface. Note that there is no user-defined smoothness factor in these equations. The weighting of the terms directly depends on the confidence measurements c_{min} and c_{max}.

Singh and Allen [SA92] proposed a two-stage technique. In the first step, given three adjacent intensity images I_{-1}, I_0, I_{+1} a SSD distribution SSD_0 for pixel $\mathbf{x}$ is formed by:

$$SSD_0(\mathbf{x}, \mathbf{d}) \quad := \quad SSD_{0,1}(\mathbf{x}, \mathbf{d}) + SSD_{0,-1}(\mathbf{x}, \mathbf{d})$$

where $SSD_{i,j}(\mathbf{x}, \mathbf{d})$ denotes the SSD surface at position $\mathbf{x}$, when searching for the flow vector from frame i to frame j. The authors retrieve a probability distribution from SSD_0:

$$P_c(\mathbf{d}) \quad := \quad e^{-k \cdot SSD_0(\mathbf{x}, \mathbf{d})}$$

where k is a normalization constant. The sub-pixel flow estimates $\mathbf{u}_c = (u_c, v_c)$ are then obtained as mean of this distribution with respect to $\mathbf{d}_x$ and $\mathbf{d}_y$:

$$\begin{aligned} u_c &= \frac{\sum_{\mathbf{d}} P_c(\mathbf{d}) \mathbf{d}_x}{\sum_{\mathbf{d}} P_c(\mathbf{d})} \\ v_c &= \frac{\sum_{\mathbf{d}} P_c(\mathbf{d}) \mathbf{d}_y}{\sum_{\mathbf{d}} P_c(\mathbf{d})} \end{aligned}$$

Using these estimates a covariance matrix S is created:

$$S_c = \frac{1}{\sum_{\mathbf{d}} P_c(\mathbf{d})} \begin{pmatrix} \sum_{\mathbf{d}} P_c(\mathbf{d})(\mathbf{d}_x - u_c)^2 & \sum_{\mathbf{d}} P_c(\mathbf{d})(\mathbf{d}_x - u_c)(\mathbf{d}_y - v_c) \\ \sum_{\mathbf{d}} P_c(\mathbf{d})(\mathbf{d}_x - u_c)(\mathbf{d}_y - v_c) & \sum_{\mathbf{d}} P_c(\mathbf{d})(\mathbf{d}_y - v_c)^2 \end{pmatrix}$$

Singh and Allen use the eigenvalues of $(S_c)^{-1}$ as confidence measures. After obtaining all flow vectors $\mathbf{u}_c = (u_c, v_c)$ they are propagated to their neighbors in order to smooth the flow field.

Given a flow field $\mathbf{u}_i = (u_i, v_i)$ the *average flow vector* $\mathbf{u}_n = (u_n, v_n)$ is computed again as a mean of the probability distribution in the local neighborhood:

$$\begin{aligned} u_n &= \frac{\sum_{\mathbf{d}} P_c(\mathbf{u}_i(\mathbf{d})) \cdot \mathbf{u}_i(\mathbf{d})_u}{\sum_{\mathbf{d}} P_c(\mathbf{u}_i(\mathbf{d}))} \\ v_n &= \frac{\sum_{\mathbf{d}} P_c(\mathbf{u}_i(\mathbf{d})) \cdot \mathbf{u}_i(\mathbf{d})_v}{\sum_{\mathbf{d}} P_c(\mathbf{u}_i(\mathbf{d}))} \end{aligned}$$

where $\mathbf{u}_i(\mathbf{d})$ denotes the precomputed flow vector at the displacement $\mathbf{d}$ and the subscripts u and v refer to the horizontal and vertical component respectively. In the next step a covariance matrix S_n is calculated for $\mathbf{u}_n = (u_n, v_n)$ in the same manner as above. The final flow field is then obtained by minimizing the smoothness functional:

$$\int_{\mathbf{x} \in \Omega} (\mathbf{u} - \mathbf{u}_n)(S_n)^{-1}(\mathbf{u} - \mathbf{u}_n) + (\mathbf{u} - \mathbf{u}_c)(S_c)^{-1}(\mathbf{u} - \mathbf{u}_c) d\mathbf{x}$$

The final flow vector is a weighted average between the precomputed displacement $\mathbf{u}_c$ and the average flow vector $\mathbf{u}_n$. The weighting is determined by the respective confidences measures. Singh and Allen use a multi-scale approach similar to Anandan to enable the computation of large displacements.

Region-matching algorithms suffer from the same problems as correlation-based approaches. An increased window size normally improves the estimates, but it also runs the risk of overlapping motion discontinuities, which leads to false results. Okutomi et al. [OK92] propose an algorithm using an adaptive search window, whose size is dynamically adjusted to allow motion discontinuities. However, since their approach restricts the window to a rectangular area, motion discontinuities and non-rectangular occlusions avoid a correct motion detection.

2.4 Frequency-Based Techniques

Frequency-based techniques compute the optical flow between subsequent frames in the frequency domain. The main idea behind these techniques is that velocities can be retrieved by analyzing the response of filters in frequency domain, as edges in images can be detected by applying edge detection filters. Usually several filters are applied to an input signal (i.e. an image sequence), which decomposes that signal according to scale, speed and orientation.

Firstly, an image is hierarchically transformed into the frequency domain. The 3D Fourier transformation of an image $\mathbf{I}(x, y, t)$ is:

$$\hat{I}(f_x, f_y, f_t) = \int\int\int \mathbf{I}(x, y, t) e^{-2\pi i(f_x x + f_y y + f_t t)} dx dy dt$$

where f_x, f_y denote the spatial and f_t the temporal frequency. By using the derivative property of Fourier transforms

$$\int \frac{\delta \mathbf{I}(v)}{\delta v} e^{-2\pi i f v} dv = -2\pi i f \int \mathbf{I}(v) e^{-2\pi i f v} dv$$

equation 2.2 can be written as:

$$-2\pi i(u \cdot f_x + v \cdot f_y + f_t)\hat{I}(f_x, f_y, f_t) = 0$$

This yields the optical flow constraint in the frequency domain:

$$u \cdot f_x + v \cdot f_y + f_t = 0 \tag{2.13}$$

The equation implies that the flow vector is a function of the spatio-temporal frequency, which forms a plane in frequency space.

Energy-based matching Heeger [Hee88] apply twelve so-called *Gabor filters* to the input signal at different spatial scales, tuned for different spatial orientation and temporal frequencies. The final flow vector for a local patch is determined by finding a plane that best fits the filter response for that patch in frequency domain.

Phase-based matching Fleet & Jepson [FJ90] assume that a flow vector can be derived from the motion of level phase contours in the response of band-pass filters. Firstly, the input signal is decomposed using bandpass-filters tuned for scale, speed and orientation. Then the filter responses are separated into amplitude and phase (see figure 2.10 for an 1D-example). Phase-based matching approaches assume that the phase is conserved under motion. For each of the filter responses the *component velocity* for a given pixel is determined by analysing the motion of the respective phase contour. The final velocity is eventually determined by a least-squares fit of the component velocities to a parametric motion model. The assumption of phase conservation is motivated by the fact that phase is independent from amplitude. Therefore, phase-based matching approaches are less sensitive to small brightness changes between frames.

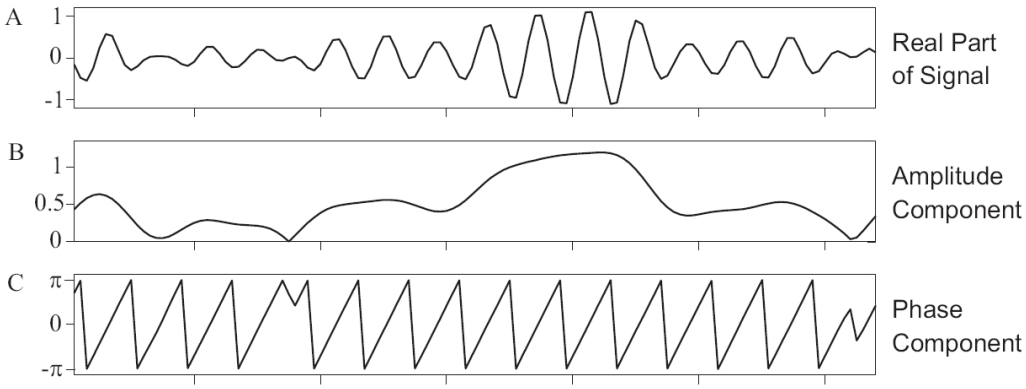


Figure 2.10: Composition of a 1D-signal into phase and amplitude.

Frequency-based techniques allow the determination of flow vectors in situations where all previous approaches tend to fail. For instance, if two random dot patterns move in opposed directions, the responses of the velocity-tuned filters might detect these discontinuous motions, where other approaches fail only incorporating spatial neighborhoods.

2.5 Conclusions

In this chapter we described several optical flow algorithms. We decided to implement the approach by Brox et al. [BBPW04] as our basic optical flow algorithm, since it produces very accurate flow fields and already allows violations of the model assumptions due to the robust statistic. Moreover, it provides an easy way to incorporate our additional information channels: the surface normals and the depth discontinuities.

Chapter 3

Basic Optical Flow Algorithm

This chapter deals with our basic algorithm, the variational optical flow algorithm by Brox et al. [BBPW04]. The approach combines the following ideas:

- A variational model that includes a gradient constancy constraint and a robust statistic in order to allow violations of the model assumptions.
- An elaborated minimization of the energy functional using two nested fixed point iterations instead of linearizing the data conservation constraint.
- A multi-scale approach that prevents the minimization process to trap into a local minimum.

In the following the algorithm and its underlying theory is explained in detail.

3.1 Variational Model

The variational model of Brox et al. includes three assumptions. Let $\mathbf{I} : \mathbb{R}^3 \to \mathbb{R}$ be a sequence of intensity images.

Gray value constancy assumption As already introduced by the first optical flow algorithm [HS81], the approach assumes that the intensity of a pixel does not change due to its displacement from one frame to the next:

$$\mathbf{I}(x, y, t) \quad = \quad \mathbf{I}(x + u, y + v, t + 1) \tag{3.1}$$

Many algorithms linearize this constraint by the first order Taylor expansion (see equation 2.2). This is consciously avoided by Brox et al. since a linearization requires that the image intensity changes linearly along the displacement which is usually not the case. Hence, approaches using a linearized intensity constraint cannot deal well with large displacements.

Gradient constancy assumption Natural image sequences usually contain slight changes in illumination, which violate the gray value constancy assumption. Therefore, another constraint is added which assumes that the *first derivative* of the intensity is constant:

$$\nabla\mathbf{I}(x, y, t) \quad = \quad \nabla\mathbf{I}(x + u, y + v, t + 1) \tag{3.2}$$

where ∇ denominates the spatial gradient. The constraint is invariant under the gray value constancy assumption and tolerates intensity changes. However, it is only helpful if the motion of a pixel is translatory. If an object in the scene is rotated or perspectively foreshortened, the assumption is violated.

Smoothness assumption As stated in the previous chapter, the last two constraints are insufficient for a flow estimation which is supposed to be robust against noise and reliably solves the aperture problem. Therefore, as a third constraint we assume that the flow field varies smoothly, i.e. that flow vectors of adjacent pixels are similar. Since a sequence can also contain opposing movements, e.g. if adjacent objects move in different directions, we demand a *piecewise smooth* flow field. The smoothness assumption can refer to the spatial domain only, when the flow between two frames is computed, or can be extended to a *spatio-temporal smoothness assumption*, which postulates that the flow field also varies smoothly *over time*, i.e. over the entire image sequence.

Brox et al. compose the three assumptions in an energy functional that penalizes all deviations from the model assumptions. Let $\mathbf{x} := (x, y, t)^T$ be the spatio-temporal position in the image sequence. Furthermore, $\mathbf{w} := (u, v, 1)^T$ denotes the searched flow vector which maps a pixel from frame t in the sequence to its successor at index $t + 1$. Since the gray value constancy and the gradient constancy assumption regard the actual content of the images, we refer to these as *data constraints*. The global deviations from these assumptions are measured by the energy functional

$$E_{\text{Data}}(u, v) := \int_{\Omega} \Psi \left(|\mathbf{I}(\mathbf{x} + \mathbf{w}) - \mathbf{I}(\mathbf{x})|^2 + \gamma |\nabla \mathbf{I}(\mathbf{x} + \mathbf{w}) - \nabla \mathbf{I}(\mathbf{x})|^2 \right) d\mathbf{x} \quad (3.3)$$

where Ψ is a robust statistic which is defined below. γ denotes the weight between the gray value constancy and the gradient constancy constraint. Similar to Horn & Schunck [HS81] our smoothness assumption assigns a high energy to all flow vectors which strongly differ from the displacements of their neighbors. This yields the energy functional

$$E_{\text{Smooth}}(u, v) := \alpha \cdot \int_{\Omega} \Psi \left(|\nabla_3 u|^2 + |\nabla_3 v|^2 \right) d\mathbf{x} \quad (3.4)$$

where $\nabla_3 := (\delta_x \ \delta_y \ \delta_t)^T$ is the spatial-temporal gradient operator, i.e. the derivatives in spatial and temporal domain. If the flow is only calculated between two frames, it is replaced with the spatial gradient $\nabla := (\delta_x \ \delta_y)^T$.

The final energy of the flow field from a frame t to its successor $t+1$, which has to be minimized, reads

$$E(u, v) = E_{\text{Data}}(u, v) + E_{\text{Smooth}}(u, v) \quad (3.5)$$

where α in equation 3.4 weights the impact of the smoothness term. The higher the value of α is, the smoother is the resulting flow field. If $\alpha = 0$, smoothness is not demanded and the flow computation is based on the data constraints only.

Robust statistic As introduced in section 2.2.2, Brox et al. apply a robust statistic to the energy deviations in order to reduce the influence of outliers. They apply a modified L^1 norm $\Psi(s^2) := \sqrt{s^2 + \varepsilon^2} \approx |s|$ as a penalizer. ε is a small constant that ensures that the function is differentiable. We used $\varepsilon = 0.0001$ in our implementation. The fact that $\Psi(s^2)$ is a convex function is helpful for the minimization process.

3.2 Minimization

This section treats the minimization of the energy functional $E(u, v)$.

3.2.1 Multi-Scale Approach

Brox et al. apply a multi-scale approach as explained in section 2.2.4 in order to allow large displacements by avoiding the minimization process to trap into a local minimum. Our implementation uses a bilinear interpolation strategy to scale images down. Let $J^k : \mathbb{R}^2 \to \mathbb{R}$ be an intensity image of size $cx \times cy$ at a certain level $k > 0$. Furthermore, let η be the scale factor with $0 < \eta < 1$. Then the next coarser image $J^{k-1} : \mathbb{R}^2 \to \mathbb{R}$ with size $(\eta \cdot cx) \times (\eta \cdot cy)$ is computed as:

$$J^{k-1}(x, y) \quad := \quad J^k \left(\frac{1}{\eta} \cdot x, \frac{1}{\eta} \cdot y \right)$$

As mentioned in section 1.4 we retrieve intermediate image values by means of bilinear interpolation.

When a flow field has been computed for a particular level, it is scaled up to the next finer scale, where it acts as initial estimate. The upscaling operation works similarly but with the inverse scale factor η^{-1}, of course. Additionally, the flow vectors need to be scaled by η^{-1} as well, since they span a greater displacement on the new level.

Although $\eta = 0.5$ is a commonly used scale factor in recent optical flow algorithms, Brox et al. propose to use a higher value in order to prevent the initial estimate of a particular level to trap directly into a local minimum. Our implementation allows an arbitrary scale factor. In section 3.6 we demonstrate that the accuracy of the final flow field strongly depends on this value.

3.2.2 Euler-Lagrange Equation

According to the calculus of variations the minimum of the energy functional 3.5 must fulfill the Euler-Lagrange equations. Similar to Brox et al. [BBPW04] we use the following notation:

$$\begin{aligned} \mathbf{I}_x &:= \delta_x \mathbf{I}(\mathbf{x} + \mathbf{w}) \\ \mathbf{I}_y &:= \delta_y \mathbf{I}(\mathbf{x} + \mathbf{w}) \\ \mathbf{I}_z &:= \mathbf{I}(\mathbf{x} + \mathbf{w}) - \mathbf{I}(\mathbf{x}) \end{aligned}$$

$$\begin{aligned}
\mathbf{I}_{xx} &:= \delta_{xx}\mathbf{I}(\mathbf{x}+\mathbf{w}) \\
\mathbf{I}_{xy} &:= \delta_{xy}\mathbf{I}(\mathbf{x}+\mathbf{w}) \\
\mathbf{I}_{yy} &:= \delta_{yy}\mathbf{I}(\mathbf{x}+\mathbf{w}) \\
\mathbf{I}_{xz} &:= \delta_{x}\mathbf{I}(\mathbf{x}+\mathbf{w}) - \delta_{x}\mathbf{I}(\mathbf{x}) \\
\mathbf{I}_{yz} &:= \delta_{y}\mathbf{I}(\mathbf{x}+\mathbf{w}) - \delta_{y}\mathbf{I}(\mathbf{x})
\end{aligned} \tag{3.6}$$

With this notation the Euler-Lagrange equations can be written as:

$$\begin{aligned}
\Psi'(\mathbf{I}_z^2 + \gamma(\mathbf{I}_{xz}^2 + \mathbf{I}_{yz}^2)) \cdot (\mathbf{I}_x\mathbf{I}_z + \gamma(\mathbf{I}_{xx}\mathbf{I}_{xz} + \mathbf{I}_{xy}\mathbf{I}_{yz}) & \\
-\alpha \cdot \mathrm{div}(\Psi'|\nabla_3 u|^2 + |\nabla_3 v|^2)\nabla_3 u) &= 0 \\
\Psi'(\mathbf{I}_z^2 + \gamma(\mathbf{I}_{xz}^2 + \mathbf{I}_{yz}^2)) \cdot (\mathbf{I}_y\mathbf{I}_z + \gamma(\mathbf{I}_{yy}\mathbf{I}_{yz} + \mathbf{I}_{xy}\mathbf{I}_{xz}) & \\
-\alpha \cdot \mathrm{div}(\Psi'|\nabla_3 u|^2 + |\nabla_3 v|^2)\nabla_3 v) &= 0
\end{aligned} \tag{3.7}$$

with reflecting Neumann boundary conditions, i.e. the derivatives at the image boundaries are zero. $\Psi'(s^2) = \frac{1}{2\sqrt{s^2+\varepsilon^2}}$ is the derivative of the robust statistic $\Psi(s^2)$ with respect to its argument s^2.

3.2.3 Numerical Approximation

Solving the Euler-Lagrange equations is a hard task since it is highly non-linear in the terms u and v. Rather than linearizing equations 3.7, Brox et al. find the minimum by two nested fixed point iterations. They transform the solution into a series of linear systems.

3.2.3.1 Outer Fixed Point Iteration

Brox et al. combine the multi-scale approach with a fixed point iteration. The flow vectors at the coarsest level are initialized with $\mathbf{w} = (0, 0, 1)$. As soon as a fixed point is found for one level in the image pyramid, the flow field is scaled up and taken as an initial guess to the next level. Let $\mathbf{w}^k := (u^k, v^k, 1)$ be the fixed point of level k. Then the fixed point $\mathbf{w}^{k+1}$ of the next finer level is the solution of:

$$\begin{aligned}
\Psi'((\mathbf{I}_z^{k+1})^2 + \gamma((\mathbf{I}_{xz}^{k+1})^2 + (\mathbf{I}_{yz}^{k+1})^2)) \cdot (\mathbf{I}_x^k\mathbf{I}_z^{k+1} + \gamma(\mathbf{I}_{xx}^k\mathbf{I}_{xz}^{k+1} + \mathbf{I}_{xy}^k\mathbf{I}_{yz}^{k+1}) & \\
-\alpha \cdot \mathrm{div}(\Psi'|\nabla_3 u^{k+1}|^2 + |\nabla_3 v^{k+1}|^2)\nabla_3 u^{k+1}) &= 0 \\
\Psi'((\mathbf{I}_z^{k+1})^2 + \gamma((\mathbf{I}_{xz}^{k+1})^2 + (\mathbf{I}_{yz}^{k+1})^2)) \cdot (\mathbf{I}_y^k\mathbf{I}_z^{k+1} + \gamma(\mathbf{I}_{yy}^k\mathbf{I}_{yz}^{k+1} + \mathbf{I}_{xy}^k\mathbf{I}_{xz}^{k+1}) & \\
-\alpha \cdot \mathrm{div}(\Psi'|\nabla_3 u^{k+1}|^2 + |\nabla_3 v^{k+1}|^2)\nabla_3 v^{k+1}) &= 0
\end{aligned} \tag{3.8}$$

where I_*^k correspond to the notations in equations 3.6, in which $\mathbf{w}$ has been replaced by $\mathbf{w}^k$. The equations 3.8 use a fully implicit scheme for the smoothness term and a semi-implicit scheme for the data term. Implicit schemes usually yield higher stability and faster convergence [BBPW04]. The system is still non-linear due to the non-linear penalizer function Ψ' and the symbols I_*^{k+1}.

As a next step towards a linear system the terms I_{*z}^{k+1} are linearized using the first order Taylor expansion:

$$\begin{aligned}
\mathbf{I}_z^{k+1} &\approx \mathbf{I}_z^k + \mathbf{I}_x^k du^k + \mathbf{I}_y^k dv^k \\
\mathbf{I}_{xz}^{k+1} &\approx \mathbf{I}_{xz}^k + \mathbf{I}_{xx}^k du^k + \mathbf{I}_{xy}^k dv^k \\
\mathbf{I}_{yz}^{k+1} &\approx \mathbf{I}_{yz}^k + \mathbf{I}_{xy}^k du^k + \mathbf{I}_{yy}^k dv^k
\end{aligned}$$

where $u^{k+1} = u^k + du^k$ and $v^{k+1} = v^k + dv^k$. Therefore, the system of equations does not depend on the current flow vectors u^{k+1} and v^{k+1} anymore, but on the previous solutions u^k and v^k and unknown increments du^k and dv^k. Replacing I_{*z}^{k+1} with their linear approximations results in:

$$\begin{aligned}
0 &= (\Psi')^k_{\text{Data}} \cdot \left(\mathbf{I}_x^k(\mathbf{I}_z^k + \mathbf{I}_x^k du^k + \mathbf{I}_y^k dv^k)\right) \\
&+ \gamma(\Psi')^k_{\text{Data}} \cdot \left(\mathbf{I}_{xx}^k(\mathbf{I}_{xz}^k + \mathbf{I}_{xx}^k du^k + \mathbf{I}_{xy}^k dv^k) + \mathbf{I}_{xy}^k(\mathbf{I}_{yz}^k + \mathbf{I}_{xy}^k du^k + \mathbf{I}_{yy}^k dv^k)\right) \\
&- \alpha \cdot \operatorname{div}\left((\Psi')^k_{\text{Smooth}} \nabla_3(u^k + du^k)\right) \\
0 &= (\Psi')^k_{\text{Data}} \cdot \left(\mathbf{I}_y^k(\mathbf{I}_z^k + \mathbf{I}_x^k du^k + \mathbf{I}_y^k dv^k)\right) \\
&+ \gamma(\Psi')^k_{\text{Data}} \cdot \left(\mathbf{I}_{yy}^k(\mathbf{I}_{yz}^k + \mathbf{I}_{xy}^k du^k + \mathbf{I}_{yy}^k dv^k) + \mathbf{I}_{xy}^k(\mathbf{I}_{xz}^k + \mathbf{I}_{xx}^k du^k + \mathbf{I}_{xy}^k dv^k)\right) \\
&- \alpha \cdot \operatorname{div}\left((\Psi')^k_{\text{Smooth}} \nabla_3(v^k + dv^k)\right)
\end{aligned}$$

where

$$\begin{aligned}
(\Psi')^k_{\text{Data}} &:= \Psi'(\ (\mathbf{I}_z^k + \mathbf{I}_x^k du^k + \mathbf{I}_y^k dv^k)^2 \\
&\quad + \gamma(\mathbf{I}_{xz}^k + \mathbf{I}_{xx}^k du^k + \mathbf{I}_{xy}^k dv^k)^2 + \gamma(\mathbf{I}_{yz}^k + \mathbf{I}_{xy}^k du^k + \mathbf{I}_{yy}^k dv^k)^2\ ) \\
(\Psi')^k_{\text{Smooth}} &:= \Psi'\left(|\nabla_3(u^k + du^k)|^2 + |\nabla_3(v^k + dv^k)|^2\right) \qquad (3.9)
\end{aligned}$$

3.2.3.2 Inner Fixed Point Iteration

Due to the non-linear function Ψ, the system is still non-linear in the unknown increments du^k and dv^k. Since Ψ is a convex function the remaining optimization problem has a unique minimum. The leftover non-linearity in Ψ' is removed by applying another *inner* fixed point iteration. Let $du^{k,0} = 0$ and $dv^{k,0} = 0$ be the initial increment for a given level k. Moreover, $(\Psi')^{k,l}_{\text{Data}}$ and $(\Psi')^{k,l}_{\text{Smooth}}$ equal equations 3.9 for iteration step l. Then the final linear system for $du^{k,l+1}$ and $dv^{k,l+1}$ is:

$$\begin{aligned}
0 &= (\Psi')^{k,l}_{\text{Data}} \cdot \left(\mathbf{I}_x^k(\mathbf{I}_z^k + \mathbf{I}_x^k du^{k,l+1} + \mathbf{I}_y^k dv^{k,l+1})\right) \\
&+ \gamma(\Psi')^{k,l}_{\text{Data}} \cdot \left(\mathbf{I}_{xx}^k(\mathbf{I}_{xz}^k + \mathbf{I}_{xx}^k du^{k,l+1} + \mathbf{I}_{xy}^k dv^{k,l+1}) + \mathbf{I}_{xy}^k(\mathbf{I}_{yz}^k + \mathbf{I}_{xy}^k du^{k,l+1} + \mathbf{I}_{yy}^k dv^{k,l+1})\right) \\
&- \alpha \cdot \operatorname{div}\left((\Psi')^{k,l}_{\text{Smooth}} \nabla_3(u^k + du^{k,l+1})\right) \\
0 &= (\Psi')^{k,l}_{\text{Data}} \cdot \left(\mathbf{I}_y^k(\mathbf{I}_z^k + \mathbf{I}_x^k du^{k,l+1} + \mathbf{I}_y^k dv^{k,l+1})\right) \\
&+ \gamma(\Psi')^{k,l}_{\text{Data}} \cdot \left(\mathbf{I}_{yy}^k(\mathbf{I}_{yz}^k + \mathbf{I}_{xy}^k du^{k,l+1} + \mathbf{I}_{yy}^k dv^{k,l+1}) + \mathbf{I}_{xy}^k(\mathbf{I}_{xz}^k + \mathbf{I}_{xx}^k du^{k,l+1} + \mathbf{I}_{xy}^k dv^{k,l+1})\right) \\
&- \alpha \cdot \operatorname{div}\left((\Psi')^{k,l}_{\text{Smooth}} \nabla_3(v^k + dv^{k,l+1})\right)
\end{aligned}$$

After discretizing the gradient and the divergence operators this sparse linear system can be solved using the Gauss-Seidl or the Successive Overrelaxation (SOR) method.

It is important to emphasize that only the single fixed point iterations are linearized, *not* the energy functional itself. The entire numerical process minimizes the *non-linear* energy functional using a *set* of linear systems.

3.3 Multi-Channel Flow

We follow the idea of Sand et al. [ST06] and allow our algorithm to run on multiple images at once. The intention is to force the optical flow algorithm to match the pixels of multiple channels not just the intensity values. The new version of the data energy functional 3.3 reads:

$$E_{\text{Data}}(u,v) \quad := \quad \frac{1}{\gamma_\Sigma} \sum_i \gamma_i \int_\Omega \Psi\left(|\mathbf{I}_i(\mathbf{x}+\mathbf{w}) - \mathbf{I}_i(\mathbf{x})|^2\right) d\mathbf{x} \tag{3.10}$$

$\mathbf{I}_i$ denotes the i-th one-dimensional input channel, which is weighted by γ_i. We normalize the channels by dividing by $\gamma_\Sigma = \sum_i \gamma_i$ for usability reasons: when users decide to exclude one or more channels, they are not required to adapt the smoothing weight α in equation 3.4. The better all image channels match, the lower is the data energy. Brox et al. analytically enforce the gradient constancy constraint. We just take the gradient of the intensity image as another input channel.

3.4 Local Smoothness

Sand et al. [ST06] introduce an isotropic image-driven regularizer by adding a *local smoothness* term into the energy functional in order to "discourage flow discontinuities at locations with small image gradients". Moreover, their approach avoids smoothing over intensity discontinuities. If a bright object moves in front of a darker one, smoothing should be lowered at the object boundaries, which can easily be detected by the intensity gradient in this case. Furthermore, the smoothness should be high inside solid regions of the image.

Rather than using one global smoothness weight α as in equation 3.4, the authors replace this constant with a local smoothness *function*, which computes the smoothness factor depending on the image position $\mathbf{x} \in \Omega$. Thus, we obtain a new energy functional for the smoothness constraint:

$$E_{\text{Smooth}}(u,v) \quad := \quad \int_\Omega \alpha(\mathbf{x}) \cdot \Psi\left(|\nabla_3 u|^2 + |\nabla_3 v|^2\right) d\mathbf{x} \tag{3.11}$$

Let I be the intensity images of frame t. Then the local smoothness function for the flow computation between t and $t+1$ is calculated as follows:

$$\alpha(\mathbf{x}) \quad := \quad \alpha_{Gl} + \alpha_L \cdot \mathcal{N}\left(\sqrt{(255 \cdot I_x(\mathbf{x}))^2 + (255 \cdot I_y(\mathbf{x}))^2}; \sigma_l\right) \tag{3.12}$$

$\mathcal{N}$ is the zero-mean non-normalized Gaussian $\mathcal{N}(x;\sigma_l) = e^{-\frac{1}{2}\frac{x^2}{\sigma_l^2}}$ with $\sigma_l = 2.0$. The factor 255 results from the conversion from low to high dynamic range images.

We precompute this term for the entire image and name the result the *local smoothness map*. An example is shown in figure 3.1. α_{Gl} is the *global smoothness factor* as in equation 3.4. The scalar α_L weights the influence of the local smoothness map. If the intensity gradient is high, e.g. on a boundary between two differently bright objects, the resulting smoothness value is low. Conversely a low gradient yields a high smoothness, e.g. inside solid regions. Therefore, the algorithm allows flow discontinuities by reducing the smoothness energy on gradient borders. Local smoothness is disabled for $\alpha_L = 0$.

(a) (b)

Figure 3.1: Local smoothness. (a) Intensity image of a mannequin. (b) Corresponding local smoothness map. The map successfully reduces the smoothness at the boundary between the shirt and the background (black lines), but the image also exposes some issues. Due to the stripes in the shirt the smoothness is lowered in rigid areas as well. Moreover, the critical discontinuity between the right arm and the chest is not detected at all. Therefore, this boundary would be oversmoothed. Intensity differences are apparently not the right way to ensure a piecewise smooth flow field.

The approach works quite well if the motion discontinuities and the high intensity gradients match. However, since this technique only regards intensity values it fails if an object has a high-frequency texture. This is often the case in natural images, especially, when the clothes of human-beings are involved. Figure 3.1 also illustrates this issue.

3.5 Implementation

This section explains our implementation of the optical flow algorithm in detail. Given a sequence of image, we describe how the flow field from a frame at time t to the next frame at time $t+1$ is computed. We implemented both, a spatial and a spatio-temporal regularizer. For reasons of simplicity, we present the minimization process using spatial smoothness term only. The extension to temporal smoothing is briefly explained below (section 3.5.3).

3.5.1 Discretization

We store the flow vectors u, v and the increments du, dv in rectangular arrays. Moreover, we precompute the terms $(\Psi')^{k,l}_{\text{Data}}$ and $(\Psi')^{k,l}_{\text{Smooth}}$ for all pixels and store them in rectangular arrays, too. In the following $f(x,y)$ denotes the content of such an array f at the coordinate (x,y). Based on the modifications of the energy functional in sections 3.3 and 3.4 we obtain two equations for a particular pixel (x,y):

$$\begin{aligned} 0 &= \frac{1}{\gamma_\Sigma}\sum_i (\Psi')^{k,l}_{i,\text{Data}}(x,y)\cdot\Big(\mathbf{I}^k_{i,x}(\mathbf{I}^k_{i,z}+\mathbf{I}^k_{i,x}du^{k,l+1}(x,y)+\mathbf{I}^k_{i,y}dv^{k,l+1}(x,y))\Big) \\ &- \alpha(\mathbf{x})\cdot\operatorname{div}\Big((\Psi')^{k,l}_{\text{Smooth}}(x,y)\nabla(u^k(x,y)+du^{k,l+1}(x,y))\Big) \qquad (3.13) \\ 0 &= \frac{1}{\gamma_\Sigma}\sum_i (\Psi')^{k,l}_{i,\text{Data}}(x,y)\cdot\Big(\mathbf{I}^k_{i,y}(\mathbf{I}^k_{i,z}+\mathbf{I}^k_{i,x}du^{k,l+1}(x,y)+\mathbf{I}^k_{i,y}dv^{k,l+1}(x,y))\Big) \\ &- \alpha(\mathbf{x})\cdot\operatorname{div}\Big((\Psi')^{k,l}_{\text{Smooth}}(x,y)\nabla(v^k(x,y)+dv^{k,l+1}(x,y))\Big) \qquad (3.14) \end{aligned}$$

We omitted the subscripts (x,y,t) of the expressions $\mathbf{I}^k_{i,*}$ in the data term for reasons of readability. These expressions correspond to the notations in equation 3.6 according to the i-th color channel. We discretize them by

$$\begin{aligned} \mathbf{I}^k_{i,x}(x,y,t) &:= \mathbf{I}_i(x+u^k(x,y)+1, y+v^k(x,y), t+1) - \mathbf{I}_i(x+u^k(x,y)-1, y+v^k(x,y), t+1) \\ \mathbf{I}^k_{i,y}(x,y,t) &:= \mathbf{I}_i(x+u^k(x,y), y+v^k(x,y)+1, t+1) - \mathbf{I}_i(x+u^k(x,y), y+v^k(x,y)-1, t+1) \\ \mathbf{I}^k_{i,z}(x,y,t) &:= \mathbf{I}_i(x+u^k(x,y), y+v^k(x,y), t+1) - \mathbf{I}_i(x,y,t). \end{aligned}$$

Furthermore, we need to discretize the gradient and the divergence operator of the smoothness term for both equations. We demonstrate this according to the first of the two Euler-Lagrange equations (3.13):

$$
\begin{aligned}
& \alpha(\mathbf{x}) \cdot \operatorname{div}\left((\Psi')^{k,l}_{\text{Smooth}}(x,y)\nabla(u^k(x,y)+du^{k,l+1}(x,y))\right) \\
= \;& \alpha(\mathbf{x}) \cdot \operatorname{div}\left((\Psi')^{k,l}_{\text{Smooth}}(x,y)\begin{bmatrix} u^k(x,y)-u^k(x-1,y)+du^{k,l+1}(x,y)-du^{k,l+1}(x-1,y) \\ u^k(x,y)-u^k(x,y-1)+du^{k,l+1}(x,y)-du^{k,l+1}(x,y-1)\end{bmatrix}\right) \\
= \;& \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x+1,y) \cdot \left(u^k(x+1,y)-u^k(x,y)+du^{k,l+1}(x+1,y)-du^{k,l+1}(x,y)\right) - \\
& \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x,y) \cdot \left(u^k(x,y)-u^k(x-1,y)+du^{k,l+1}(x,y)-du^{k,l+1}(x-1,y)\right) + \\
& \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x,y+1) \cdot \left(u^k(x,y+1)-u^k(x,y)+du^{k,l+1}(x,y+1)-du^{k,l+1}(x,y)\right) - \\
& \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x,y) \cdot \left(u^k(x,y)-u^k(x,y-1)+du^{k,l+1}(x,y)-du^{k,l+1}(x,y-1)\right)
\end{aligned}
$$

As Sand et al. [ST06], we use the pattern $[-1\ 1\ 0]$ for the derivatives in the gradient, but the pattern $[0\ -1\ 1]$ for the discretization of the divergence operator in order to center the second derivative $[1\ -2\ 1]$ at the position (x, y). The discretization of the smoothness term in the second equation (3.14) works analogously:

$$
\begin{aligned}
& \alpha(\mathbf{x}) \cdot \operatorname{div}\left((\Psi')^{k,l}_{\text{Smooth}}(x,y)\nabla(v^k(x,y)+dv^{k,l+1}(x,y))\right) \\
= \;& \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x+1,y) \cdot \left(v^k(x+1,y)-v^k(x,y)+dv^{k,l+1}(x+1,y)-dv^{k,l+1}(x,y)\right) - \\
& \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x,y) \cdot \left(v^k(x,y)-v^k(x-1,y)+dv^{k,l+1}(x,y)-dv^{k,l+1}(x-1,y)\right) + \\
& \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x,y+1) \cdot \left(v^k(x,y+1)-v^k(x,y)+dv^{k,l+1}(x,y+1)-dv^{k,l+1}(x,y)\right) - \\
& \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x,y) \cdot \left(v^k(x,y)-v^k(x,y-1)+dv^{k,l+1}(x,y)-dv^{k,l+1}(x,y-1)\right)
\end{aligned}
$$

Due to our discretizations, we obtain two equations for each pixel with 10 unknowns $du(x,y)$, $dv(x,y)$, $du(x+1,y)$, $dv(x+1,y)$, $du(x-1,y)$, $dv(x-1,y)$, $du(x,y+1)$, $dv(x,y+1)$, $du(x,y-1)$ and $dv(x,y-1)$.

3.5.2 Algorithm

Initialization Before the start of the flow computation, our algorithm creates the image pyramids for all input channels and the local smoothness map if specified. After that the flow field in horizontal and vertical direction u, v is initialized with 0 at the coarsest-level.

Outer fixed point iteration (coarse-to-fine) At the beginning of each outer iteration we reset the incrementals du, dv to 0. After du, dv have been calculated within the inner fixed point iterations, the flow field is updated by $u' = u+du, v' = v+dv$ and scaled up to the next level. The iteration is repeated until the finest level is reached. For each level we precompute the values $I^k_{i,*}$ for all image channels, since they remain invariant for the whole inner loop.

Inner fixed point iteration At the beginning of each inner fixed point iteration we precompute the values $(\Psi')^{k,l}_{\text{Data}}(x,y)$ and $(\Psi')^{k,l}_{\text{Smooth}}(x,y)$. We use the same discretizations for the gradients in the smoothness value as described in section 3.5.1. In order to reduce the number of operations we precompute

$$(\bar{\Psi}')^{k,l}_{\text{Smooth}}(x,y) \;:=\; \alpha(\mathbf{x}) \cdot (\Psi')^{k,l}_{\text{Smooth}}(x,y)$$

Linear system Then the inner linear system is solved. We repeat the *Successive Overrelaxation (SOR)* method for a user-defined number of times. The method is an extension of the Gauss-Seidel method and speeds up its convergence by extrapolating the solution (see appendix A for more information). In each step we obtain new estimates $du(x,y), dv(x,y)$ for all pixels. Since the SOR solver always uses the most recent results to compute the next solution, no further arrays are needed to store intermediate results.

For a particular pixel (x,y) we obtain $du(x,y)$ by:

$$\begin{aligned}
A &:= 2 \cdot (\bar{\Psi}')^{k,l}_{\text{Smooth}}(x,y) + (\bar{\Psi}')^{k,l}_{\text{Smooth}}(x+1,y) + (\bar{\Psi}')^{k,l}_{\text{Smooth}}(x,y+1) \\
&\quad + \sum_i (\bar{\Psi}')^{k,l}_{i,\text{Data}}(x,y) \cdot (\mathbf{I}^k_{i,x}(x,y,t))^2 \\
B &:= \sum_i (\bar{\Psi}')^{k,l}_{i,\text{Data}}(x,y) \cdot \mathbf{I}^k_{i,y}(x,y,t) \cdot \mathbf{I}^k_{i,x}(x,y,t) \\
C &:= -(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x+1,y) \\
D &:= -(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x,y) \\
E &:= -(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x,y+1) \\
\text{Const} &:= -(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x,y) \cdot (2 \cdot u(x,y) - u(x-1,y) - u(x,y-1)) \\
&\quad + (\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x+1,y) \cdot (u(x+1,y) - u(x,y)) \\
&\quad + (\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x,y+1) \cdot (u(x,y+1) - u(x,y)) \\
&\quad - \sum_i (\bar{\Psi}')^{k,l}_{i,\text{Data}}(x,y) \cdot \mathbf{I}^k_{i,x}(x,y,t) \cdot \mathbf{I}^k_{i,z}(x,y,t) \\
du(x,y) &= \omega \cdot (\text{Const} - B \cdot dv(x,y) - C \cdot du(x+1,y) - D \cdot du(x-1,y) \\
&\qquad - E \cdot du(x,y+1) - D \cdot du(x,y-1))/A \cdot \\
&\quad (1-\omega) \cdot du(x,y)
\end{aligned}$$

where $\omega \in [1,2]$ is the extrapolation factor of the SOR method, which we usually set to 1.9 for our optimizations. $dv(x,y)$ is retrieved analogously:

$$\begin{aligned}
A &:= \sum_i (\bar{\Psi}')^{k,l}_{i,\text{Data}}(x,y) \cdot \mathbf{I}^k_{i,x}(x,y,t) \cdot \mathbf{I}^k_{i,y}(x,y,t) \\
B &:= 2 \cdot (\bar{\Psi}')^{k,l}_{\text{Smooth}}(x,y) + (\bar{\Psi}')^{k,l}_{\text{Smooth}}(x+1,y) + (\bar{\Psi}')^{k,l}_{\text{Smooth}}(x,y+1) \\
&\quad + \sum_i (\bar{\Psi}')^{k,l}_{i,\text{Data}}(x,y) \cdot (\mathbf{I}^k_{i,y}(x,y,t))^2 \\
C &:= -(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x+1,y)
\end{aligned}$$

$$
\begin{aligned}
D &:= -(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x,y) \\
E &:= -(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x,y+1) \\
\text{Const} &:= -(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x,y)\cdot(2\cdot v(x,y)-v(x-1,y)-v(x,y-1)) \\
&\quad +(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x+1,y)\cdot(v(x+1,y)-v(x,y)) \\
&\quad +(\bar{\Psi}')^{k,l}_{i,\text{Smooth}}(x,y+1)\cdot(v(x,y+1)-v(x,y)) \\
&\quad -\sum_i(\bar{\Psi}')^{k,l}_{i,\text{Data}}(x,y)\cdot\mathbf{I}^k_{i,y}(x,y,t)\cdot\mathbf{I}^k_{i,z}(x,y,t) \\
dv(x,y) &= \omega\cdot(\text{Const}-A\cdot du(x,y)-C\cdot dv(x+1,y)-D\cdot dv(x-1,y) \\
&\qquad -E\cdot dv(x,y+1)-D\cdot dv(x,y-1))/B\cdot \\
&\quad (1-\omega)\cdot dv(x,y)
\end{aligned}
$$

3.5.3 Spatio-Temporal Smoothing

The extension to a spatio-temporal regularizer is straight forward: instead of the 2D gradient a 3D gradient is used in the smoothness term, which includes the spatial *and* temporal neighbors of each flow vector. Again, reflected boundary conditions are applied, i.e. the temporal derivatives in the first frame and the last frame are zero. A drawback of the spatio-temporal smoothness regularizer is that it requires to solve the linear systems for all flow fields in a sequence in parallel. This might result in a memory issue when big images and/or long sequences are processed.

3.6 Parameter Dependence

The optical flow algorithm provides several tuning parameters that influence the quality of the flow field and the runtime of the computation. These are the scale factor of the image pyramid (η), the number of coarse-to-fine levels, the number of inner fixed point iterations and the number of SOR iterations to solve the inner linear system. In this section we illustrate the relation of these parameters to the error of the results and the runtime of the algorithm with several experiments. We tested the algorithm for the two frames of the scene in section 5.3.1.1: a motionless synthetic human model captured under a changing viewpoint. We used the three color channels of the 336×477 pixel images as input channels and applied a uniform weight ($\gamma_i = 1$). In this experiment we did not take local smoothness into account. Moreover, we limited the minimum extension of an image in the image pyramid to 12.

After each calculation we compared the *quality* of the obtained flow field with the ground truth flow field of the scene. In our case the term "quality" denotes the average L^2-error, which is introduced in section 5.2.1. We refer to section 5.1.2 for a detailed description of ground truth optical flow. All flow fields have been computed on a notebook with an Intel T2400 dual-core processor at 1.83GHz and 1GB RAM.

The quality of the flow field strongly depends on the chosen **global smoothness** value α_{Gl}. Figure 3.2 relates the global smoothness value to the L^2 error of the flow field. On the one hand, if α_{Gl} is too high the flow field is oversmoothed and discontinuous motions are not detected. On the other hand, a very low smoothness value yields a computation that is more sensitive to noise and creates non-rigid flow fields. Moreover, the function seems to be roughly convex. We used a binary search algorithm to find the approximatively best smoothness value for a given scene. In this case $\alpha_{Gl} = 0.011758$ turned out to be the best smoothness value.

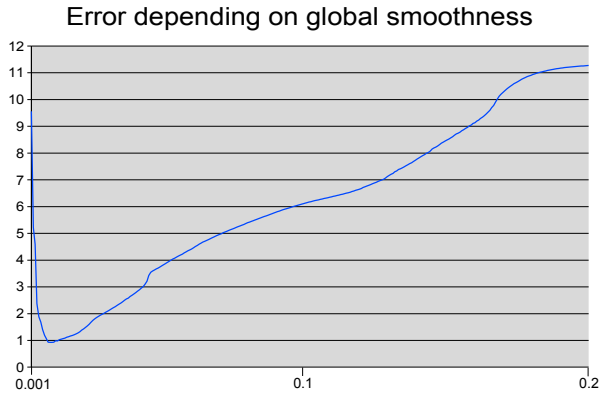


Figure 3.2: Performance of our experiment depending on the global smoothness factor.

In the following we discuss the parameters for the numerical approximation. Brox points out that solving the linear systems until convergence induces high computational costs [Bro05]. Moreover, he proposes to iterate the SOR several times and then continue with the next inner fixed point iteration. Brox – and most of the subsequent papers using the same fixed point iteration scheme – fix the number of iterations without checking for convergence. In general, more iterations give rise to a more precise result, but since each iteration increase the runtime of the algorithm, the parameters should be chosen carefully in order to ensure the efficiency of the computation.

As stated before, the **scale factor** for the creation of the image pyramid is crucial. Table 3.1 relates the used scale factor to the L^2-error and to the runtime of the algorithm. The data are plotted in the top row of figure 3.3. The scale factor directly corresponds to the number of outer fixed point iterations, which equal the number of levels in the image pyramid. The quality of the flow computation highly depends on the scale factor and we investigated that a scale factor below 0.9 is inappropriate.

We compared our implementation with an implementation of the older Black & Anandan's algorithm [BA93], which assumes similar constraints as Brox et al.: a *linearized* brightness constancy assumption, global regularization combined

with a robust statistic and a multi-scale approach. We modified the latter to allow arbitrary scale factors.
Our modified Brox-algorithm outperforms the implementation of Black & Anandan's technique firstly at the scale factor 0.7. More interestingly a higher scale factor does not seem to help the technique of Black & Anandan significantly. Due to the minimization via fixed point iterations the algorithm by Brox et al. requires a longer runtime than its older pendant, which is justified by a higher accuracy in the flow estimation.
The scale factor also has a strong impact on the runtime. Let us assume an input image of size $cx \times cy$. If the extensions of the smallest image in the image pyramids are limited to S_{min} and the scale factor is set to η, then the number of outer fixed point iterations equals $n = \log_{\frac{1}{\eta}} \frac{S_{min}}{\min\{cx,cy\}}$. Therefore, the number of these iterations grows with $\mathcal{O}(\frac{1}{\log \eta})$. The curve of the runtime (see top right of figure 3.3) depicts this thought.

Scale factor	Levels	L^2-Error Our	Runtime [ms] Our	L^2-Error B & A	Runtime [ms] B & A
0.50	5	7.655	9.667	3.456	6.922
0.55	6	5.473	10.294	3.386	7.406
0.60	7	5.034	11.228	3.048	8.063
0.65	8	4.873	12.738	3.069	8.953
0.70	10	2.553	14.132	2.969	10.016
0.75	12	2.486	16.440	2.947	11.640
0.80	15	1.852	19.795	2.927	14.000
0.85	20	1.171	25.560	2.919	18.047
0.90	29	0.924	37.179	2.916	25.906
0.95	55	0.792	70.836	2.905	49.906

Table 3.1: Performance of our algorithm depending on scale factor (number of outer iterations) using five inner fixed point and 100 SOR iterations, compared to the performance of the implementation of Black & Anandan (B & A).

Table 3.2 illustrates the contribution of the **number of inner fixed point iteration** and the **number of SOR iterations** to the quality of the flow fields in our example. We investigated that five inner fixed point iterations and 100 SOR iterations result in an appropriate compromise between quality and runtime. Both number of iterations raise the runtime linearly as shown in the middle and bottom row of figure 3.3.

3.7 Discussion

The algorithm of Brox et al. produces highly accurate results at the expense of runtime. On the one hand, the runtime of the algorithm is higher than that of most previous algorithms, mostly due to the fixed point iteration scheme and the high scale factor. On the other hand, the results are accurate and the robust statistic already allows motion discontinuities in the flow field. However, motion

boundaries are still noticeably oversmoothed. Moreover, it does not find correct correspondences if the textures of objects in the scene contain high-frequency patterns. These issues are visualized in chapter 5 and motivate the extensions, which we present in the next chapter.

Inner iterations	L^2-Error	Runtime [ms]
1	1.314	9.765
2	1.131	15.657
3	1.020	22.610
4	0.952	29.156
5	0.924	36.047
6	0.921	42.593
7	0.907	50.597
8	0.904	58.211
9	0.898	65.412
10	0.898	72.352
11	0.889	78.082
12	0.896	88.834
13	0.889	94.230
14	0.893	102.572
15	0.889	110.275

SOR iterations	L^2-Error	Runtime [ms]
25	1.253	12.891
50	1.000	21.373
75	0.945	29.642
100	0.924	38.028
125	0.901	45.574
150	0.876	53.495
175	0.857	62.228
200	0.843	70.495
225	0.836	78.556
250	0.830	85.529
275	0.827	93.168
300	0.825	101.525
325	0.824	110.399
350	0.823	117.041

Table 3.2: Left: Performance of our algorithm depending on the number of inner fixed point iterations using 100 SOR iterations. Right: Performance of the algorithm depending on the number of SOR iterations using five inner fixed point iterations. We applied a scale factor of 0.9 in both cases.

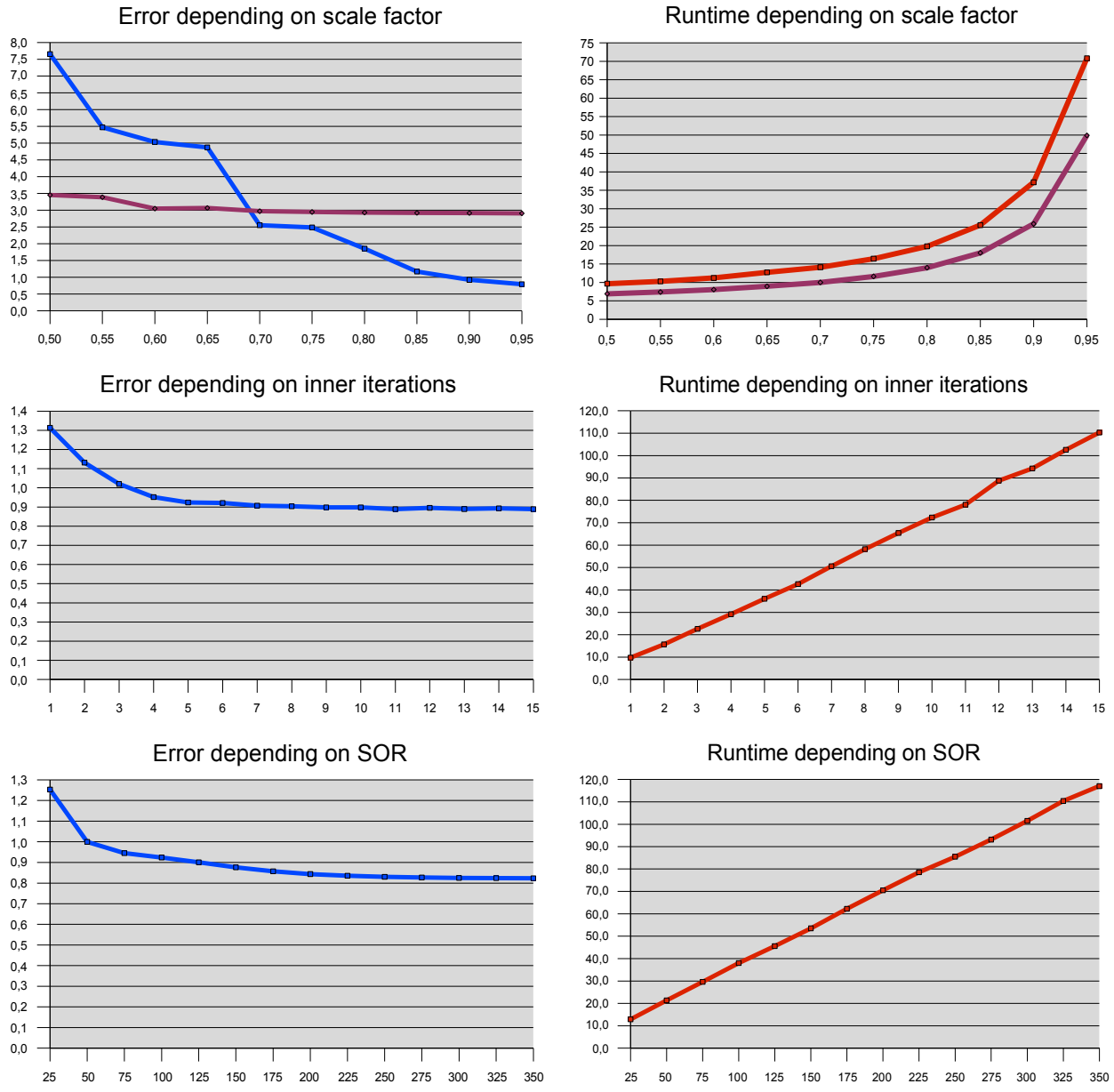


Figure 3.3: Performance of our implementation depending on parameters for numerical minimization. Blue lines: L^2-error produced by our algorithm, red lines: runtime in milliseconds, violet line: L^2-error/runtime produced by Black & Anandan's algorithm.

Chapter 4

Extensions

This chapter treats the acquisition of the two information channels, with which we enrich our optical flow estimation: the surface normals and the depth discontinuities. After the introduction of each technique we show how it is incorporated into the optical flow algorithm.

4.1 Surface Normals

The surface normal at a particular point p on an object is the vector, which is perpendicular to the tangent plane to the surface at p. This section explains how this information can be extracted from a set of two-dimensional images which are illuminated under different lighting conditions.

4.1.1 Photometric Stereo

The technique *Photometric Stereo* was published by Woodham [Woo89]. If an object is frequently captured with a (perfectly) Lambertian material from a static viewpoint, but illuminated from different lighting directions, the surface normal can be analytically computed.

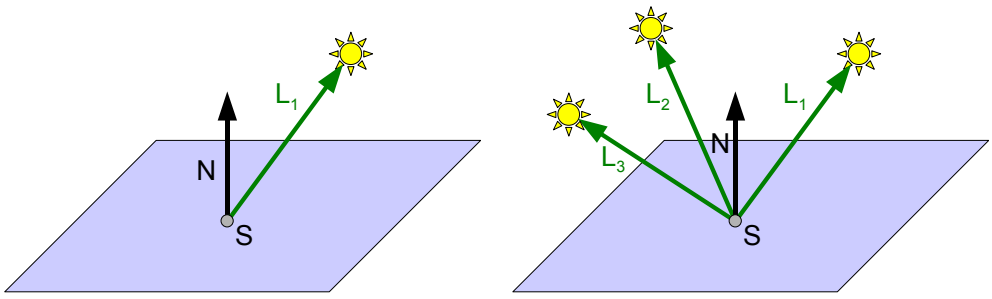
Let S be a point on a surface, N is its normal (see figure 4.1a). If L^1 is the direction of the only point light source in the scene, then, under the assumption of Lambertian reflection, the intensity i_1 is calculated as follows:

$$i_1 \quad = \quad \rho \cdot L^1 \cdot N$$

where ρ denotes the albedo of the surface at point S. If we capture the scene under three lighting conditions, using a point light source from a different direction each time (L^j for $j \in \{1, 2, 3\}$, see figure 4.1b), we can write the transformation as a linear map:

$$\begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} \quad = \quad \rho \cdot \begin{pmatrix} L_x^1 & L_y^1 & L_z^1 \\ L_x^2 & L_y^2 & L_z^2 \\ L_x^3 & L_y^3 & L_z^3 \end{pmatrix} \cdot \begin{pmatrix} N_x \\ N_y \\ N_z \end{pmatrix}$$

$$\Leftrightarrow \quad I \quad = \quad \rho \cdot L \cdot N$$



(a) (b)

Figure 4.1: (a) The intensity of point S on a surface with Lambertian reflection is determined by the albedo at S and the dot product of the lighting direction L_1 with the surface normal N. (b) If we add two more light sources, the normal N and the albedo can be easily computed by the three lighting directions and the resulting intensity values at S.

where $\begin{pmatrix} L_x^i & L_y^i & L_z^i \end{pmatrix}^T$ represents the lighting direction that corresponds to intensity i_i. If the intensity values and the lighting directions are known, we retrieve the surface normal and the albedo by solving the linear system:

$$M = L^{-1} \cdot I = \rho \cdot N \tag{4.1}$$

$$\Rightarrow \quad N = \frac{M}{\|M\|} \quad \wedge \quad \rho = \|M\| \tag{4.2}$$

L is invertible if the three lighting directions are linearly independent. If we have captured three images of a scene from a static viewpoint, but illuminated from different lighting directions, we can compute a normal map by applying the above formulas to each pixel, i.e. we setup the vector I for the surface point under a particular pixel by extracting the intensity values from the pixel values (e.g. by taking the average of all color components). The lighting direction matrix is the same for all pixels and needs to be initialized once only. In the following we call a pair of an intensity value and its respective lighting direction a *photometric sample*.

The above formulas require that all light sources irradiate with the same intensity. If this is not the case, the intensity values of each pixel must be normalized by dividing through the intensity of the light source.

Real images Real images usually introduce noise, self-shadowing, interreflections and specular reflections. Hence, the normal maps usually contain errors or at least noise if only three images are taken into account. Therefore, more photometric samples per pixel are needed to retrieve a good approximation of the normal map. If we acquire n images for $n > 3$, I becomes a n-dimensional vector and L a non-quadratic $n \times 3$ matrix, i.e. we process n intensity values per pixel from n different lighting conditions.

$$\begin{pmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{pmatrix} = \rho \cdot \begin{pmatrix} L_x^1 & L_y^1 & L_z^1 \\ L_x^2 & L_y^2 & L_z^2 \\ \vdots & \vdots & \vdots \\ L_x^n & L_y^n & L_z^n \end{pmatrix} \cdot \begin{pmatrix} N_x \\ N_y \\ N_z \end{pmatrix}$$
$$\Leftrightarrow \quad I = \rho \cdot L \cdot N$$

Now the linear system is over-constrained and cannot be solved directly. We apply a least-squares approach by computing the pseudo-inverse of L. Therefore, we perform a Singular Value Decomposition (SVD) on L:

$$L = U \cdot \Sigma \cdot V^*$$

where U is a $n \times n$ matrix, Σ is a $n \times 3$ matrix, whose diagonal elements are non-zero only, and V^* is a 3×3 matrix. This representation is used to calculate the *pseudo-inverse* L^+:

$$L^+ = V \cdot \Sigma^+ \cdot U^*$$

where Σ^+ denotes the transpose of Σ, whose diagonal elements are replaced by their reciprocal, and U^* is the conjugate transpose of U. Since U is a real-valued matrix, $U^* = U^T$. Now M, the albedo and the normal can be calculated as in equations 4.1 and 4.2 by replacing L^{-1} with L^+.

In our implementation we usually drop all samples, whose intensity value falls under a certain threshold in order to remove noisy pixels. Additionally, we drop samples which exceed a specific threshold in order to remove all samples which could be the part of specular highlights.

Example In this example we calculate the surface normals of a plant that has been captured from multiple lighting directions[1] (see figure 4.2a). All input images have been normalized using the measured intensities of the light sources. The scene contains a lot of self-shadowing and specular reflections. Moreover, the camera introduced noise that is noticeable in dark regions. We encode the resulting normal map as an RGB image, where red represents the x-component, green the y-component and blue the z-component of the normal for each pixel. For a better visualization we store the normal maps in *offset gray mode*, i.e. we map all color components from $[-1, 1]$ to $[0, 1]$ in order to avoid negative values. As shown in figure 4.2b, it is not sufficient to regard only three lighting directions in order to compute the normal map. 253 lighting directions have been taken into account for figure 4.2c, where we have dropped all samples with an intensity below 0.1 in order to remove noise. Apparently, this result is more accurate.

[1]This dataset including the directions and intensities of the light sources is available at http://gl.ict.usc.edu/Data/LightStage/

(a) (b) (c)

Figure 4.2: (a) The captured plant. The red frame indicates the area of the image that is shown in the other two figures. (b) Normal map using three distinct lighting directions. The results are imprecise and lack of details in regions, where at least one of the lighting conditions casts a shadow. (c) Normal map using 263 lighting directions. The normal map is more accurate and also contains normals in regions that are shadowed under some lighting conditions.

4.1.2 Gradient Patterns

A major drawback of the method just mentioned is the demand for lots of lighting conditions in order to obtain a correct result. This is especially a problem if a moving subject is captured. Ma et al. [MHP+07] propose a technique which only uses four different lighting conditions. The approach firstly illuminates the scene by the environment maps of the first two bands of the Spherical Harmonic (SH) basis. Therefore, for each lighting condition we need to sample the incident illumination from an entire surrounding sphere.

Let n be the number of light sources which we have been uniformly distributed on a (infinitely far away) sphere around the object. L_j is the direction of the j-th light source that points to the center of the scene. $\tilde{i}_j^x$ is the intensity of that light source for lighting condition x. For the only SH basis vector in the first band the scene is uniformly illuminated from all directions. We call this the *uniform lighting condition* ($\tilde{i}_j^{\text{uniform}} := 1$). The intensities for the second band of the SH basis result from the dot product of the unit vectors with the lighting direction:

$$\begin{aligned}
\tilde{i}_j^{\text{left}} &:= L_i \cdot \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T \\
\tilde{i}_j^{\text{top}} &:= L_i \cdot \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T \\
\tilde{i}_j^{\text{front}} &:= L_i \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T
\end{aligned}$$

Since we cannot radiate negative light, we map all intensities from $[-1, 1]$ to $[0, 1]$:

$$i_j^x := 0.5 \cdot (\tilde{i}_j^x + 1)$$

In figures 4.3a-h you can see these four lighting patterns and the plant of the above example (figure 4.2a) illuminated under these conditions. If we have captured a scene using these so-called *gradient patterns*, we calculate the ratio

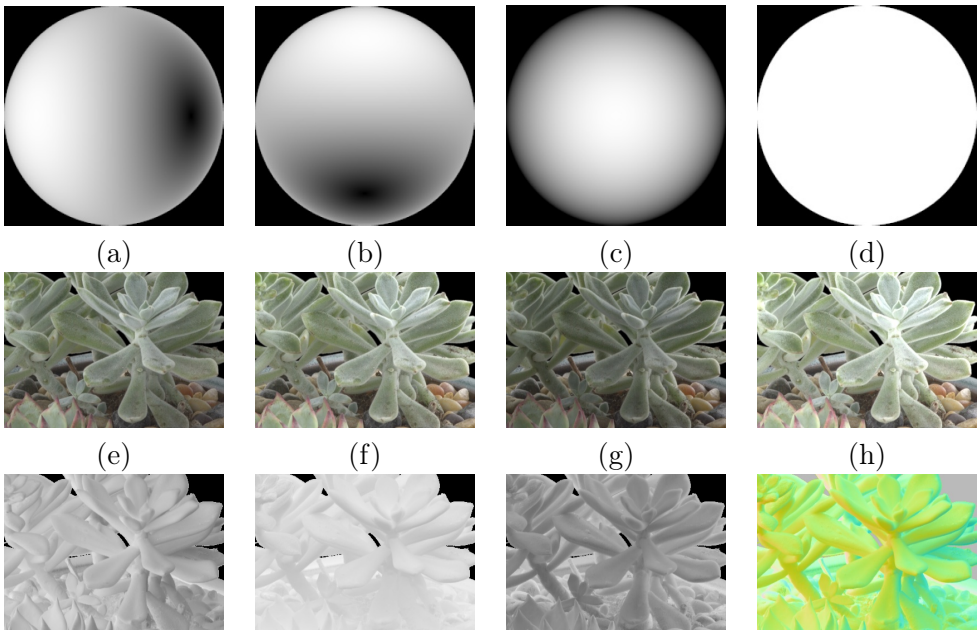


Figure 4.3: Spherical Harmonic patterns viewed through a mirrored ball. (a)-(c) Left, top and front gradient pattern (second SH band). (d) Full lit pattern (first SH band). (e)-(h) Plant captured under the four patterns. (i)-(k) Ratio images for left, top and front pattern. (l) Final normal map.

between the second band SH lighting conditions (left, top, front) and the uniform condition. Let $I_{\text{left}}, I_{\text{top}}, I_{\text{front}}, I_{\text{uniform}}$ be the intensity images that have been taken under the respective lighting conditions, then the *ratio images* are calculated as follows.

$$R_{\text{left}}(\mathbf{x}) := I_{\text{left}}(\mathbf{x})/I_{\text{uniform}}(\mathbf{x}) \tag{4.3}$$

$$R_{\text{top}}(\mathbf{x}) := I_{\text{top}}(\mathbf{x})/I_{\text{uniform}}(\mathbf{x}) \tag{4.4}$$

$$R_{\text{front}}(\mathbf{x}) := I_{\text{front}}(\mathbf{x})/I_{\text{uniform}}(\mathbf{x}) \tag{4.5}$$

Ma et al. show that the intensity values of the ratio images lie between $\frac{1}{6}$ and $\frac{5}{6}$ and that they are linearly related to the normal components. For a given pixel $\mathbf{x} \in \Omega$ we compute the normal $N'(\mathbf{x})$ by mapping the ratio values to $[-1, 1]$:

$$N'(\mathbf{x}) := \begin{bmatrix} (R_{\text{left}}(\mathbf{x}) - \frac{1}{2}) \cdot 3 \\ (R_{\text{top}}(\mathbf{x}) - \frac{1}{2}) \cdot 3 \\ (R_{\text{front}}(\mathbf{x}) - \frac{1}{2}) \cdot 3 \end{bmatrix} \tag{4.6}$$

Normalization ($N(\mathbf{x}) := N'(\mathbf{x})/\|N'(\mathbf{x})\|$) yields the final normal vector. The final normal map of our example is shown in figure 4.3l.

A big advantage of this approach is that it only needs four images to produce good normal maps, which do not suffer from noise that much. Moreover, this

technique works equally well for an arbitrary number of viewpoints. The normals are estimated in world space (since we used unit vectors for the light intensities), but, due to the usage of a Spherical Harmonic basis, the four images can be transformed into any coordinate system by a rotation matrix [RH02]. However, sampling the incident light of an environment map is not a simple task. The light source should be sufficiently far away from the center of the scene, since we assume directional lighting. In addition, the sampling of light sources must be dense enough to obtain a good approximation of incident environmental lighting. One device that might fullfil this requirement is explained in appendix C.

4.1.3 Incorporation into the Optical Flow Framework

As described above, we encode normal maps as RGB color images. The normal maps are simply incorporated into the existing optical flow framework by interpreting them as additional input channels. We just add the red, green and blue data channel to the multi-channel data functional in equation 3.10 (section 3.3).

4.2 Depth Discontinuities

A depth discontinuity is a C_0 discontinuity in the depth map of a scene. We assume that the knowledge of these discontinuities helps to provide a piecewise smooth flow field. Let $I : \mathbb{N}^2 \rightarrow \mathbb{R}^3$ be a rectangular RGB color image and $d : \mathbb{N}^2 \rightarrow \mathbb{R}$ the corresponding depth function, that maps each pixel position to a depth value between 0 and 1. In our implementation we encode the *depth discontinuity map* $\mathbf{D} : \mathbb{N}^2 \rightarrow \{0,1\}^3$ as an RGB image in order to distinguish between horizontal and vertical depth discontinuities:

$$\mathbf{D}(x,y)_{\mathrm{r}} := \begin{cases} 1 & |d(x,y) - d(x+1,y)| > \varepsilon_D \\ 0 & otherwise \end{cases}$$

$$\mathbf{D}(x,y)_{\mathrm{g}} := \begin{cases} 1 & |d(x,y) - d(x,y+1)| > \varepsilon_D \\ 0 & otherwise \end{cases}$$

The red channel $\mathbf{D}(x,y)_{\mathrm{r}}$ corresponds to horizontal and the green channel $\mathbf{D}(x,y)_{\mathrm{g}}$ to vertical depth discontinuities. The blue channel $\mathbf{D}(x,y)_{\mathrm{b}}$ is always set to zero. ε_D denotes the threshold that differentiates between normal transitions in the depth map and discontinuities, which separate objects from each other.

4.2.1 Based on Normal Maps

The normal maps, which have been presented in section 4.1, build a potential source to derive discontinuities from. As a first simple approach, we consider all pixels in a normal map which strongly differ from their neighbors as part of depth discontinuity borders. Let $\mathbf{N} : \mathbb{N}^2 \rightarrow \mathbb{R}^3$ be a normal map as shown in

figure 4.4a, where normals are encoded as RGB colors. Based on this map we compute a threshold map T as follows:

$$\mathbf{D}(x,y)_{\mathrm{r}} := \begin{cases} 1 & \text{if } \|\mathbf{N}(x+1,y) - \mathbf{N}(x,y)\| > \varepsilon_N \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

$$\mathbf{D}(x,y)_{\mathrm{g}} := \begin{cases} 1 & \text{if } \|\mathbf{N}(x,y+1) - \mathbf{N}(x,y)\| > \varepsilon_N \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where $\|\cdot\|$ denotes the L^2 norm in color space. $\varepsilon_N > 0$ is a threshold, which distinguishes between regular surface normal transitions and normal discontinuities caused by presumable object boundaries.

We found out that the surface normal map is not a reliable basis for the detection of depth discontinuities as shown in figure 4.4b. Crucial depth discontinuities are often not detected because the normals between two pixels are too similar, although they lie on a discontinuity border. There are two reason for this issue: Firstly, two *parallel* surfaces can never be detected as discontinuous (the shirt and the trousers are nearly parallel at the discontinuity border), even if they are very far from each other. Secondly, if the normal map is not perfect (e.g. due to noise or self-shadowing), the derivative of the normals can be strongly biased. Moreover, we found out that depth discontinuities based on normal maps often contain many false positives, e.g. on surfaces with a rough material.

4.2.2 Based on Shadow Images

Raskar et al. [RTF+05] developed a robust algorithm to find depth discontinuities. The paper bases on the idea that a single light source being placed next to the camera in a scene casts shadows at depth discontinuities. The wider a

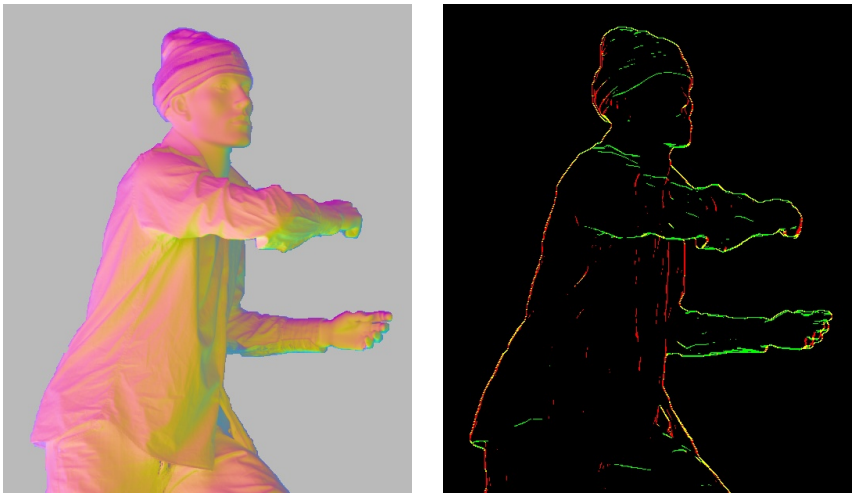
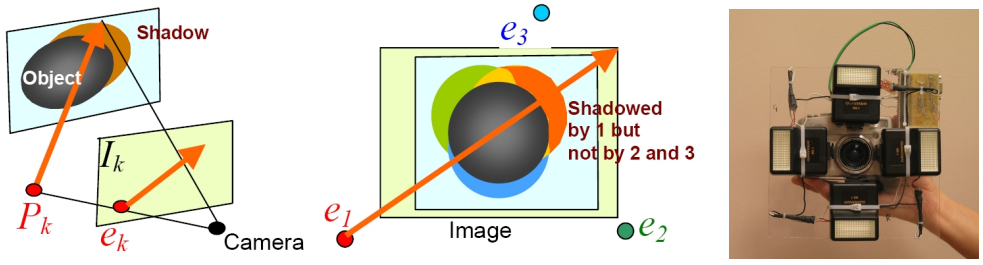


Figure 4.4: Discontinuities by normals. (a) Normal map. (b) Discontinuity map created by thresholding the normals.

shadow is, the deeper are the depth discontinuities. Formally spoken, let P_k be a point light source next to the camera and e_k the image of this light source in the image plane, the so-called *light epipole*. The pencil rays originating at P_k are called *epipolar rays* (see figure 4.5a). The author made three observations:

- The shadow of a depth edge pixel lies along the epipolar ray.
- The shadow at a depth edge always lies on the opposite side of the light epipole along the epipolar rays.
- If two epipoles lie on opposite sides of an edge in different images, a shadow will be observed in one image but not in the other one (see figure 4.5a).



(a) (b)

Figure 4.5: (a) Idea of the algorithm: Shadows are cast along epipolar rays. Light sources are chosen that shadows which appear in one image are lit at least in one different image. (b) Camera that produces images under four different lighting conditions. Source of images: [RTF+05]

In order to obtain a depth discontinuity map, the image needs to be traversed along the epipolar rays to detect the shadows. At least four images with four different light source positions are required to find all depth discontinuities. The authors simplify this process by putting the light sources directly left, top, right and bottom to the camera respectively. Thus, the epipolar rays are orientated corresponding to the image-space. The algorithm is explained in detail in the next section.

4.2.2.1 Algorithm

This paragraph explains the algorithm that finds depth discontinuities using the approach by [RTF+05]. In order to illustrate this approach, we apply all steps to a sample scene using our implementation. The algorithm is structured as follows:

1. Image acquisition
   (a) Take ambient images without any additional lighting.
   (b) Take images where light sources are placed at left, top, right and bottom of the camera respectively (so-called *Raskar images*).

2. Convert input images to gray scale and normalize them.
3. Remove ambient light from Raskar images.
4. Compute maximum intensity image.
5. Create ratio images.
6. Intensity step detection
   (a) Apply edge detection filter on ratio images.
   (b) Perform hysteresis thresholding on all ratio images.
   (c) Merge all threshold maps to create final depth discontinuity image.

First of all, we need four *Raskar images* in which a light source is place at the left, top, right and bottom side of the camera respectively. Raskar et al. [RTF+05] use a modified camera with four attached flashes at the respective positions (see figure 4.5b). In our example we use a spotlight, which is manually placed about 10 cm next to the camera in all directions in order to simulate these flashes. You find the resulting images in figure 4.6. Moreover, the original approach needs an *ambient image*, which captures the lighting of the scene without any flashes. We assume that we do not have any ambient light. Thus, our ambient image contains black pixels only.

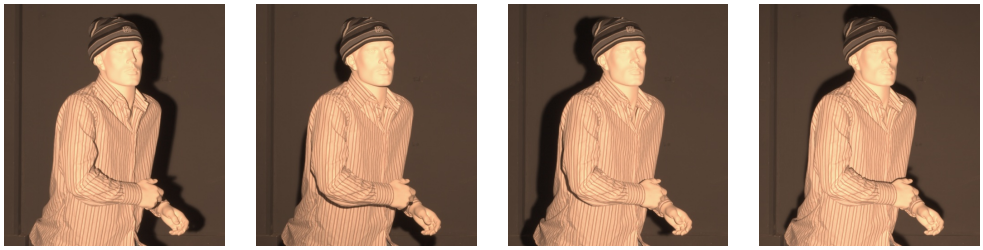


Figure 4.6: Pictures of mannequin where light source is placed left, top, right and bottom from camera respectively.

Intensity images First, all RGB input pictures are converted to *intensity images* by taking the average value of all color channels (figure 4.7). In the following procedure we remove the ambient light from all Raskar images by subtracting the ambient image (which does not make any difference in our example). Finally, the intensity images are normalized: each image is scaled by the average intensity value of all images divided by its own mean intensity value.

Maximum intensity images Then the *maximum intensity image* M is created which takes the maximum intensity value of all Raskar images (figure 4.8). Let $I_{\text{left}}, I_{\text{top}}, I_{\text{right}}, I_{\text{bottom}} : \mathbb{N}^2 \to \mathbb{R}$ be the Raskar intensity images where a

Figure 4.7: Original images after conversion to gray scale and normalization.

Figure 4.8: Maximum intensity image.

light source is located left, top, right or bottom to the camera respectively, then $M : \mathbb{N}^2 \to \mathbb{R}$ is defined as follows:

$$M(\mathbf{x}) \quad := \quad \max(I_{\text{left}}(\mathbf{x}), I_{\text{top}}(\mathbf{x}), I_{\text{right}}(\mathbf{x}), I_{\text{bottom}}(\mathbf{x})) \tag{4.9}$$

This image approximates a light source that is placed at the center of projection. It does not contain any shadows, because if one flash causes a shadow at a particular pixel position, at least the image of the opposite flash position normally produces a lit pixel (figure 4.5a).

Ratio images Now *ratio images* can be calculated by dividing each of the Raskar intensity images by the maximum image M. These ratio images R_D for $D \in \{\text{left}, \text{top}, \text{right}, \text{bottom}\}$ are calculated as follows:

$$R_D(\mathbf{x}) \quad := \quad \begin{cases} \frac{I_D(\mathbf{x})}{M(\mathbf{x})} & \text{for } M(\mathbf{x}) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

We calculate the ratio image because we need to distinguish between texture discontinuities and real depth discontinuities caused by shadows. If we assume a Lambertian material the ratio image can also be expressed as:

$$R_D(\mathbf{x}) \quad = \quad \frac{\mu_D \cdot \rho_D \cdot (L_D \cdot N(\mathbf{x}))}{\max_E\{\mu_E \cdot \rho_E \cdot (L_E \cdot N(\mathbf{x}))\}} \tag{4.11}$$

where μ_D denotes the magnitude of the light intensity of image D at pixel $\mathbf{x}$, ρ_D is the respective albedo, $N(\mathbf{x})$ is the surface normal at pixel $\mathbf{x}$ and L_D is

Figure 4.9: Ratio images.

the direction of the light source of image D. Since the albedo is the same for all images, ρ_D cancels out due to the division. Furthermore, since the light flashes are close to the camera ($L_D \cdot N(\mathbf{x}) \approx L_E \cdot N(\mathbf{x})$ for $D \neq E$) the ratio image can be approximated by $R_D(\mathbf{x}) = \mu_D / \max_E\{\mu_E\}$. Therefore, our ratio images are *independent from texture* under the assumption of diffuse reflection.

Shadowed pixels in Raskar images correspond to values near 0 in the ratio images, where pixels in non-shadowed areas are mapped to higher values. Detecting the depth discontinuities in the acquired images is now reduced to finding large intensity steps in the ratio images. You find the ratio images for our example in figure 4.9.

Edge detection We detect intensity steps by applying simple edge detection filters to the ratio images. If the light source is placed on the left or the right side of the camera, shadows are cast in horizontal directions. Consequently, we apply a horizontal Sobel filter to R_{left} and R_{right}:

$$S_{\text{Hor}} := \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \tag{4.12}$$

Since we are only interested in positive intensity steps for the left image (from non-shadowed to shadowed areas), we clamp all negative values to zero. Analogously, we clamp the positive values for the right image to zero. Taking the absolute value for all values results in the *confidence images*.

The top and bottom Raskar images cast shadows in the vertical direction. Thus, we apply the vertical Sobel filter to R_{top} and R_{bottom}:

$$S_{\text{Ver}} := \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{4.13}$$

and clamp negative values for the top image and positive values for the bottom image to zero. Again we receive confidence images by taking the absolute value. You can find the confidence images for our example in figure 4.10.

Figure 4.10: Filtered images. Depth discontinuities appear as highlighted lines.

(a) (b) (c)

Figure 4.11: Threshold maps. (a) The *high threshold map* contains the major discontinuities, but a couple of contours are incomplete or dashed. (b) The *low threshold map* contains the complete discontinuities, however also introduces noise ("false positives"). (c) The result of the hysteresis thresholding.

Hysteresis thresholding Finally, we obtain binary discontinuity maps from each of the four confidence images. The easiest way to do that is a thresholding approach. Let $C : \mathbb{N}^2 \to \mathbb{R}$ be a confidence image. The *high* threshold map $T_{\text{High}} : \mathbb{N}^2 \to \{0, 1\}$ is defined as:

$$T_{\text{High}}(\mathbf{x}) \quad := \quad \begin{cases} 1 & \text{if } C(\mathbf{x}) > \varepsilon_{\text{High}} \\ 0 & \text{otherwise} \end{cases} \tag{4.14}$$

where $\varepsilon_{\text{High}}$ denominates the user-defined threshold which distinguishes between discontinuities and regular intensity steps. You can see the threshold map for the right Raskar image of our example in figure 4.11a. As a major drawback this approach either produces interleaved lines, when $\varepsilon_{\text{High}}$ is too high and values alongside a discontinuity border jump below and beyond the threshold, or creates noisy regions if the threshold is too low (see figure 4.11b). Hence, we apply a technique called *hysteresis thresholding*.

We create an additional *low* threshold map $T_{\text{Low}} : \mathbb{N}^2 \to \{0, 1\}$:

$$T_{\text{Low}}(\mathbf{x}) \quad := \quad \begin{cases} 1 & \text{if } C(\mathbf{x}) > \varepsilon_{\text{Low}} \\ 0 & \text{otherwise} \end{cases} \tag{4.15}$$

Figure 4.12: Threshold maps of all four views.

where $\varepsilon_{\text{Low}} < \varepsilon_{\text{High}}$. The *final* threshold map $T_{\text{Final}} : \mathbb{N}^2 \to \{0, 1\}$ is then calculated as follows.

1. For all values with $C(\mathbf{x}) \geq \varepsilon_{\text{High}}$ set $T_{\text{Final}}(\mathbf{x}) = 1$.
2. For all values with $C(\mathbf{x}) < \varepsilon_{\text{Low}}$ set $T_{\text{Final}}(\mathbf{x}) = 0$.
3. For all values $\varepsilon_{\text{Low}} \leq C(\mathbf{x}) < \varepsilon_{\text{High}}$:
   - $T_{\text{Final}}(\mathbf{x}) = 1$ if $C(\mathbf{x})$ can be connected to any pixel $\bar{\mathbf{x}} \in \Omega$ with $C(\bar{\mathbf{x}}) > \varepsilon_{\text{High}}$ through a chain where all values are $> \varepsilon_{\text{Low}}$.
   - $T_{\text{Final}}(\mathbf{x}) = 0$ otherwise

Informally spoken, T_{Final} contains all connected lines of the low threshold map that includes at least one pixel, which also appears in the high threshold map. The pseudocode of the algorithm can be found in appendix B. The result of the hysteresis thresholding of the right sample image is shown in figure 4.11c. The thresholds should be chosen carefully. We set $\varepsilon_{\text{High}} = 1.5$ and $\varepsilon_{\text{Low}} = 0.6$ in our example.

Final discontinuity map Let T_{Left}, T_{Top}, T_{Right} and T_{Bottom} be the threshold maps that have been created from the four confidence images (see figure 4.12). In order to obtain the final discontinuity map $\mathbf{D}$ we apply the binary OR-operator between the horizontal and vertical threshold maps respectively:

$$\mathbf{D}(\mathbf{x})_r := \begin{cases} 1 & \text{if } T_{\text{Left}}(\mathbf{x}) = 1 \text{ or } T_{\text{Right}}(\mathbf{x}) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

$$\mathbf{D}(\mathbf{x})_g := \begin{cases} 1 & \text{if } T_{\text{Top}}(\mathbf{x}) = 1 \text{ or } T_{\text{Bottom}}(\mathbf{x}) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

You find the final depth discontinuity map for our example in figure 4.13.

4.2.3 Incorporation into the Optical Flow Framework

The *local smoothness* approach in section 3.4 already represents a first way to deal with discontinuities. We replace the local smoothness function (see equation 3.12), that only regards intensity values, with a new function that takes

Figure 4.13: Final depth discontinuity map.

the discrete discontinuities into account, which we obtained by the approach of Raskar et al.:

$$\alpha(\mathbf{x}) \quad := \quad \begin{cases} \alpha_B & \text{if } \mathbf{x} \text{ lies on a discontinuity border} \\ \alpha_G & \text{if } \mathbf{x} \text{ lies inside a rigid region} \end{cases}$$

where $\alpha_B \ll \alpha_G$. We apply a low *boundary smoothness value* α_B for pixels on depth discontinuities, because we want to allow the flow to be discontinuous at these places. Conversely, we use a high *general smoothness values* α_G for the regions between discontinuities, because we assume that the flow is rigid in those areas. The function $B : \mathbb{N}^2 \to \{0, 1\}$ describes for each pixel in the image whether it is located on a depth discontinuity border or not in terms of the depth discontinuity map $\mathbf{D}$:

$$B(\mathbf{x}) \quad := \quad \begin{cases} 1 & \text{if } \mathbf{D}(x, y) = 1 \text{ or } \mathbf{D}(x-1, y) = 1 \text{ or } \mathbf{D}(x, y-1) = 1 \\ 0 & \text{otherwise} \end{cases}$$

Coarse-to-fine Since the depth discontinuity maps are component-wisely binary maps, they cannot be scaled down for the coarse-to-fine iterations using bilinear interpolation as the other input channels. We scale these maps down as follows: Let $\mathbf{D}$ be the original depth discontinuity map. $\mathbf{D}^*$ will be the smaller version of it.

- Set all pixels in $\mathbf{D}^*$ to black (no discontinuities).
- For all pixel positions $\mathbf{x} = (x, y)$ in $\mathbf{D}$ with $\mathbf{D}(\mathbf{x})_r = 1$ (horizontal discontinuity):
  1. We assume that the discontinuity lies exactly in the middle of (x, y) and $(x + 1, y)$, i.e. on $(x + 0.5, y)$.
  2. Map $(x + 0.5, y)$ to the position in the downscaled image (x^*, y).
  3. Set $\mathbf{D}^*(\lfloor x^* \rfloor, y)_r = 1$.
- Proceed analogously for vertical discontinuities.

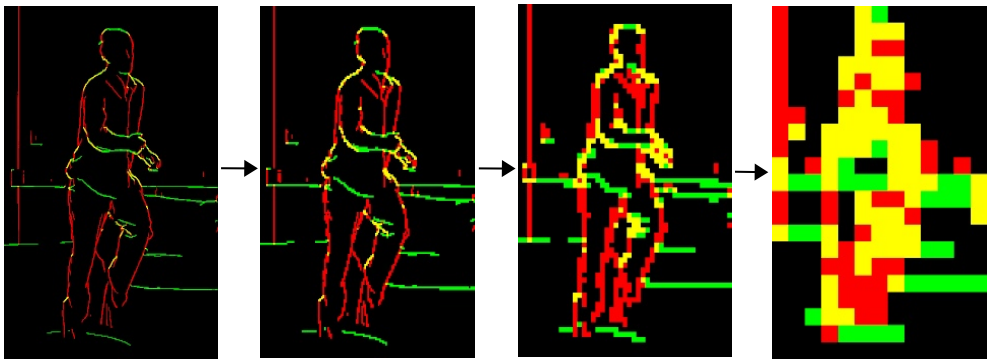


Figure 4.14: Downscaling discontinuity maps. The mannequin at the coarsest level mainly consists of discontinuities.

We map all discontinuity labels of a certain level to new positions on the coarser level. If we scaled the discontinuity maps down by performing a nearest neighbor search instead, it would be very likely that closed discontinuity borders would become interleaved. But the method presented also introduces a problem: If the original normal-sized discontinuity map contains a lot of discontinuity labels, the downscaled versions of this maps may only consist of discontinuities at the coarsest levels (see figure 4.14). Therefore, a too low smoothness value might be applied in the first outer fixed point iterations *to most pixels*. This could seed large errors in the first estimations of the flow field and impair the final result. We circumvent this issue by applying the discontinuity preserving flow estimation at first when the image has reached a certain size. If an image of the current level in the image pyramid is smaller than a certain percentage of the original size, the general smoothness value α_G is used for all pixels.

4.3 Conclusions

In this chapter we explained two techniques, which retrieve surface normals from captured images under different illumination patterns. Both photometric stereo and the gradient patterns produce accurate normal maps, whereas the latter approach requires only four lighting conditions but a special lighting apparatus. Moreover, we introduced the algorithm of Raskar et al. which extracts depth discontinuities from shadow map images. Both information channels were easily incorporated into our basic optical flow framework.

If the objects in a scene do not move, the computation of surface normals and depth discontinuity maps is rather simple. In the context of dynamic human performances the acquisition becomes an issue because the differently lit images can not be captured simultaneously. This and other results are discussed in the following chapter.

Chapter 5

Evaluation

This chapter judges our hypothesis by comparing the flow fields created with our extensions to the flow fields computed without the additional information channels. The evaluation of an optical flow algorithm is not a trivial task. The majority of the papers in recent literature evaluate the quality of their algorithms by applying it to well-known synthetic test cases, whose ground truth optical flows (see section 5.1.2) are known. This allows a convenient *quantitative* analysis of the algorithms but these synthetic cases are usually too artificial. Most of them are created in a way that many of the model assumptions (brightness constancy assumption, etc.) hold, that displacements are small, or that objects only underlie simple motion models. Since the application of optical flow usually aims at processing *real* images the test cases should be as realistic as possible. We created our own synthetic test cases and tried to focus on more real situations containing a virtual subject.

Real images are more interesting for the evaluation since they introduce a lot of problems. Beside the violations of the standard optical flow constraints (due to specular materials, transparency, shadows, interreflections, etc.), they introduce camera noise and/or compression artifacts. Unfortunately, one is usually restricted to a *qualitative* analysis of the flow fields since no ground truth is available and since there is no established metric to judge a flow field without a ground truth. This makes it difficult or even impossible to compare algorithms based on real images, which is just another reason why synthetic cases should be close to reality.

The next section describes the algorithms which we use to render synthetic test cases and to create a ground truth optical flow estimation. Section 5.2 explains the error metrics for the quantitative analysis between the computed and the ground truth flow fields. We evaluate our extensions on synthetic data in section 5.3. Section 5.4 judges our hypothesis on real images. Furthermore, we show how our extended flow estimation can be incorporated into the context of Image Based Relighting. A conclusion follows in section 5.5.

As in section 3.6, we use a notebook with an Intel T2400 dual-core processor at 1.83GHz and 1GB working memory for our experiments.

5.1 Synthetic data

We developed a Win32 GUI application called GroundTruthRenderer, which provides the rendering of two subsequent frames of a sequence and the respective ground truth optical flow between them. The user can load 3D models, which may contain diffuse, specular and textured materials, and set them up in a scene in two different poses and/or from two different camera viewpoints. After arranging both scenes the user triggers the creation of the synthetic data which is explained in the following.

5.1.1 Renderings

This section describes the types of renderings which GroundTruthRenderer produces.

5.1.1.1 OpenGL standard and normal view

First of all GroundTruthRenderer outputs a standard OpenGL rendering of both textured scenes in a user-selected resolution, where no lighting is used. Moreover, it creates the normal map of both views. Since the output of accurate normal maps requires high precision the program renders all frames to a floating-point texture, which is read out by the OpenGL function glReadPixels. Some older graphic card drivers clamp negative values. Therefore, the components of the normals are firstly rendered to the range of $[0, 1]$ and then mapped to $[-1, 1]$ after extracting the respective color values.

5.1.1.2 Matte

GroundTruthRenderer also creates a matte for both frames, which separates the foreground from the background. It is simply drawn by clearing the background with black color and rendering all polygons of the model in white color without lighting and textures.

5.1.1.3 OpenGL depth discontinuities

Our implementation provides the output of a precise depth discontinuity map. Again OpenGL is used for that purpose. The scenes are normally rendered, however this time the z-buffer is extracted. This is used to create a discontinuity map as described in section 4.2: A pixel is marked as border of a discontinuity if the difference of the z-buffer value between it and its adjacent neighbor is beyond a certain threshold, which can be specified by the user.

5.1.1.4 Noise

The user may optionally add Gaussian distributed noise to the output images in order to simulate camera noise. Let $I : \mathbb{N}^2 \to \mathbb{R}^3$ be an RGB image. The noisy image $I' : \mathbb{N}^2 \to \mathbb{R}^3$ is created by adding a Gaussian distributed random variable to the color components:

$$\begin{aligned} I'_r(\mathbf{x}) &= I_r(\mathbf{x}) + \text{GaussianRandom}(\sigma_R) \\ I'_g(\mathbf{x}) &= I_g(\mathbf{x}) + \text{GaussianRandom}(\sigma_R) \\ I'_b(\mathbf{x}) &= I_b(\mathbf{x}) + \text{GaussianRandom}(\sigma_R) \end{aligned}$$

where the subscripts r, g, b refer to the red, green and blue color channel respectively. GaussianRandom(σ_R) creates a random value depending on σ_R: the larger σ_R the higher the amplitude of the noise. The function works as follows, where n denotes the current number of calls to the function:

1. Generate two random numbers $r_1, r_2 \in [0, 1)$ using the C function rand().
2. Compute the result:

$$\text{GaussianRandom}(\sigma_R) := \sigma_R \cdot \begin{cases} \sqrt{-2 \cdot \ln(1 - r_2)} \cdot \cos(2\pi \cdot r_1) & \text{if } n \text{ is even} \\ \sqrt{-2 \cdot \ln(1 - r_2)} \cdot \sin(2\pi \cdot r_1) & \text{if } n \text{ is odd} \end{cases}$$

5.1.1.5 PBRT

Beside OpenGL we support the external renderer from the book "Physically-Based Rendering: From Theory to Application (PBRT)" [PH04] in order to obtain more realistic renderings including global illumination. If desired by the user, both frames are exported as so-called *scene files* and then sent to PBRT, which creates the output images.

This feature is intended to render human models, which are placed in a virtual lighting environment that samples the incident lighting of an environment map. The program currently supports the simulation of Light Stage 6, which is explained in appendix C, and Light Stage 5 (see [WGT+05] for further information). We simulate the lighting apparatus by substituting each of the real light sources with a virtual spotlight, which points to the center of the stage.

The user is requested to define the size and vertical offset of the human model. Furthermore, one may choose between three different rendering techniques which PBRT provides: Photon Mapping, Path Tracing and Irradiance Caching. We preferred Irradiance Caching since it rendered faster than the other approaches and created less noisy images. Our software allows to run multiple rendering processes in parallel (in the case the used computer supports multi-processing). PBRT outputs all images in a high-dynamic range (HDR) format.

GroundTruthRenderer supports the output of three types of PBRT renderings, which are briefly explained in the following.

Gradient normal maps The program supports the computation of normal maps using the gradient pattern approach, which was described in section 4.1.2. It produces the fully lit and the three *gradient images*, whereas the intensity of the spotlights is adapted to the respective lighting pattern of the second Spherical Harmonic band. Afterwards the output images are used to compute the normal maps as described before.

Raskar depth discontinuities Finally GroundTruthRenderer provides a "less synthetic" way to obtain depth discontinuities. It supports the creation of depth discontinuity maps as elucidated in section 4.2.2. For this purpose the virtual lighting environment is deactivated. Instead spotlights are set left, above, right and below the camera in order to create the four "Raskar images". After the rendering the algorithm by Raskar et al. [RTF+05] is executed as explained before. The user is asked to specify the low and the high threshold for the hysteresis thresholding in advance.

5.1.2 Ground Truth Calculation

The term *ground truth optical flow*, which is often used in literature, denominates the *motion field* of a scene, i.e. the projection of scene motion onto the image plane. The expression is contradictory since there might be multiple valid optical flow fields in a scene, which we called *apparent motion* in the introduction. As stated before, optical flow is naturally ambiguous, whereas the motion field is a strict geometric concept.

Nonetheless we need a way to quantify the accuracy of the flow algorithm. In this section we present how GroundTruthRenderer generates a motion field from the two synthetic scenes. We reduce the ambiguousness of our resulting ground truth flow fields by using simple diffuse reflection in our scenes only. Additionally, we label all ambiguous flow vectors as *undefined* and exclude them from the evaluation. There are three situations in which the flow vector from the first to the second frame is not defined at a particular position:

- The pixel is part of the solid *background*. We render our scenes on a black background. The movement of a background pixel should be zero but all other displacements leading to another background pixel are correct as well.
- The pixel is *occluded* in the second frame.
- The pixel moves *outside* the image rectangle.

We want to point out again that in all cases it is possible to calculate the motion field vector, however the optical flow vector is ambiguous and should not be taken into account for the evaluation.

5.1.3 The Idea

We follow a similar idea for the motion field computation as published by Galvin et al. [GMN+98], whereas our approach additionally allows the deformation of the objects in the scene. For each pixel P in the first frame the displacement to the corresponding pixel in the second frame is calculated as follows:

1. Shoot a ray through pixel P in the scene and find the nearest intersection P' with the surface of the model.
2. Find respective surface point Q' in the second frame.
3. Project Q' back onto the pixel coordinate Q in terms of the second viewpoint.
4. The flow vector pixel at P is: $\mathbf{u} = Q - P$
5. If pixel P is part of the (solid) background, occluded or moving out of the frame, label it as undefined.

This principle is illustrated in figure 5.1. The object in the second frame may be arbitrarily moved and deformed as long as the correspondences of the faces are ensured, i.e. if the scene contains a mesh the *topology* of this mesh must be the same in both frames.

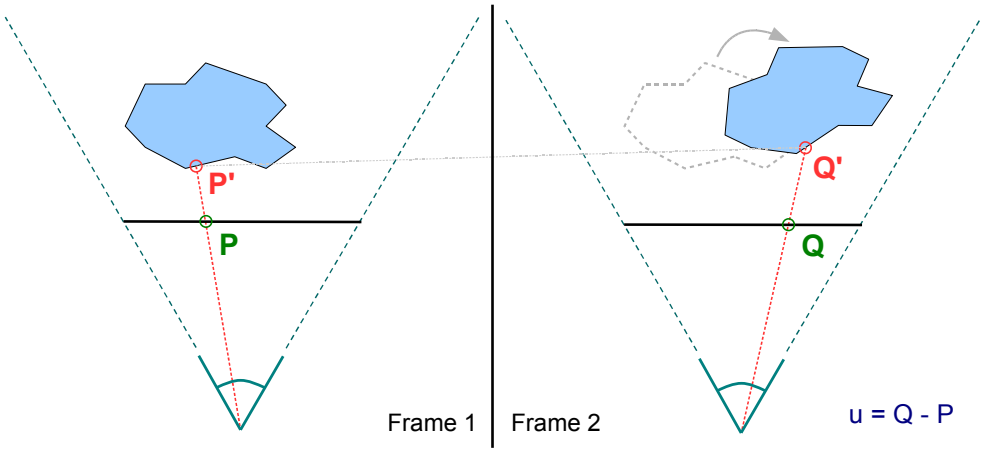


Figure 5.1: Principle of our ground truth estimation.

5.1.4 Implementation

Our algorithm follows the principle explained above but restricts the approach to triangulated Wavefront OBJ models. As mentioned before, both models need to have the same topology, which means that the model in the second view is constrained to be a copy or *deformed* version of the model in the first one. We retrieve the flow vector for a given pixel $P = (x, y)$ as follows:

Find viewing ray through P First the ray from the viewpoint through the pixel P in the first scene is determined. The function gluUnProject of the *OpenGL Utility library (GLU)*[1] provides an easy way to calculate the unprojected 3D point A on the near and B the far clip plane. The viewing ray is then determined by $x = A + \lambda(B - A)$.

Find surface intersection P' After that the nearest intersection of this ray with the geometry is detected. In order to diminish the number of triangle-ray intersection tests, an octree is used in our implementation. This reduces the complexity of the search from $\mathcal{O}(n)$ to $\mathcal{O}(\log n)$. P' is the intersection that is closest to the viewer, i.e. its relative position λ on the viewing ray is smaller (but larger than 0) than that of all other intersections.
The algorithm saves the index of the triangle n and the barycentric coordinates α and β of the intersection P'. If no intersection is found, this pixel is labeled as *undefined* background pixel.

Find displacement Since the model in the second scene contains the same topology as in the first one, the corresponding 3D position Q' in the second view can be easily determined by the triangle and the barycentric coordinates of the intersection. Let T_A, T_B and T_C be the points of n-th triangle of the second model. Then $Q' = \alpha \cdot T_A + \beta \cdot T_B + (1 - \alpha - \beta) \cdot T_C$. Using the view transformation matrices of the second view and the GLU function gluProject Q' is transformed to the pixel coordinates Q. Now the displacement reads $\mathbf{u} = Q - P$.

Check validity Finally, the algorithm checks if the flow vector at P is really *defined*. First, it tests if Q is inside the image extension. If not, the pixel is only visible in the first frame and, therefore, *undefined*.
Furthermore, our approach tests if the pixel is occluded. This is achieved by shooting another ray through the pixel Q using the second view transformation. Let R' be the 3D intersection of this ray with the geometry. If the distance $\|R' - Q'\|$ is beyond a certain threshold, Q is probably occluded and only visible in the first frame. In this case it is also marked as *undefined*.

5.1.4.1 Speed optimization using face maps

The algorithm runs with a complexity of $\mathcal{O}(\#\text{ of pixels} \cdot \log(\#\text{ of triangles}))$. We support another way of computing ground truth, which lowers the complexity to $\mathcal{O}(\#\text{ of pixels} + \#\text{ of triangles})$ by modifying the ray-intersection tests. Instead of incorporating an octree we use OpenGL to render a *face map* for each frame: each face is drawn with a unique color which encodes the face index. Therefore, the face index n at a given pixel position P can easily be detected. If a ray-intersection test is required for a given pixel P, we look up the face index using the respective face map and obtain the barycentric coordinates of the intersection by *one* ray-intersection test.

[1] See http://www.opengl.org/documentation/specs/ for more information.

The use of face maps yields a significantly shorter runtime. But a major drawback of this approach is the limitation to a discrete pixel grid: we are only able to look up faces at discrete pixel positions. When we shoot another ray through pixel position Q, to determine whether the pixel is occluded or not, we are limited to use the rounded pixel position $\bar{Q}$. Even though Q' is visible in the second frame, it is possible that the intersection of the ray through $\bar{Q}$ with the object is so far away from Q' that it is considered as occluded. We investigated that this causes aliasing effects: pixels at boundaries between faces or on very small faces are incorrectly declared as occluded. In order to overcome this problem we added an *oversampling* approach: The map of undefined pixels is computed for a higher image resolution (usually about three times larger) and then downsampled using a median filter. The median filter removes the incorrect occlusion lines near boundaries, which are mostly one pixel in width.

The usage of face maps significantly increases the performance but the variant which applies an octree is more precise and does not introduce aliasing artifacts into the estimation of occlusions. Therefore, we use the octree technique in the following chapters.

5.2 Error Metrics

This section explains quantitative and qualitative metrics, which are used for optical flow evaluation. The former metrics are applied when a ground truth flow field is available (synthetic images). The latter ones normally matter for natural images.

5.2.1 Based on Ground Truth Flow

In case of synthetic data the ground truth flow is a good estimator for the quality of the computed flow field. As described in section 5.1.2 the displacement of some pixels is undefined due to occlusion, background or movement outside the image area. We store this information in the *void-map* $V : \mathbb{N}^2 \to \{0, 1\}$ which is defined as follows:

$$V(\mathbf{x}) \quad := \quad \begin{cases} 1 & \text{flow vector at position } \mathbf{x} \text{ is undefined} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Let $G : \mathbb{N}^2 \to \mathbb{R}^2$ be the ground truth flow field and $F : \mathbb{N}^2 \to \mathbb{R}^2$ the flow field that the optical flow algorithm generated from two synthetic renderings. The ground truth error $E_{GT}(G, F)$ is computed by first summing up the error of all flow vectors between G and F. Afterwards the result is averaged by the number of flow vectors that have been taken into account.

Thus, the ground truth error $E_{GT}(G, F)$ is defined as

$$E_{GT}(G, F) \quad := \quad \frac{1}{N} \sum_{\mathbf{x} \in \Omega} (1 - V(\mathbf{x})) \cdot \Theta(G(\mathbf{x}), F(\mathbf{x})) \tag{5.2}$$

where $\Theta(\mathbf{g}, \mathbf{u})$ detects the error between two vectors $\mathbf{g}$ and $\mathbf{u}$. $N = \sum_{\mathbf{x}\in\Omega}(1 - V(\mathbf{x}))$ represents the number of respected flow vectors. Additionally to the ground truth error the standard deviation is obtained by

$$\tilde{E}_{GT}(G, F) := \sqrt{\frac{1}{N-1}\sum_{\mathbf{x}\in\Omega}(1 - V(\mathbf{x})) \cdot (E_{GT}(G, F) - \Theta(G(\mathbf{x}), F(\mathbf{x})))^2} \quad (5.3)$$

In the following we present three ways to evaluate the error between two flow vectors. Each metric assumes a third component $t = 1$ in the flow vectors, which projects the vector one time step further.

5.2.1.1 Average Angular Error (AAE)

The function

$$\Theta_{AE}(\mathbf{g}, \mathbf{u}) := cos^{-1}\frac{\mathbf{g} \cdot \mathbf{u} + 1}{\sqrt{((\mathbf{g}_u)^2 + (\mathbf{g}_v)^2 + 1) \cdot ((\mathbf{u}_u)^2 + (\mathbf{u}_v)^2 + 1)}} \quad (5.4)$$

calculates the angular error between two flow vectors. This is the most common way to compare a calculated result with ground truth optical flow. As a major drawback this measurement does not respect the magnitude of the flow vectors. Even though this measurement is mostly applied in literature, the lack of magnitude can create a false perception of quality, especially, if the ground truth optical flow contains large displacements.

5.2.1.2 Average Magnitude Error (AME)

Analogous to the angular error

$$\Theta_{ME}(\mathbf{g}, \mathbf{u}) := \left|\sqrt{(\mathbf{g}_u)^2 + (\mathbf{g}_v)^2 + 1} - \sqrt{(\mathbf{u}_u)^2 + (\mathbf{u}_v)^2 + 1}\right|_1 \quad (5.5)$$

determines the magnitude error between two flow vectors. Since the angle between the vectors is not respected, this measurement is apparently insufficient as well.

5.2.1.3 Average L^2 Error (AL2E)

The L^2 norm between two vectors

$$\Theta_{L^2}(\mathbf{g}, \mathbf{u}) := \|u_G - u_F\| \quad (5.6)$$

incorporates the angle *and* the magnitude of two vectors into the error estimation. We will use this metric as primary estimator for evaluation based on ground truth.

5.2.1.4 Conclusion

Ground truth evaluation is the best estimator for synthetic data. However, one should keep in mind that ground truth is not the only correct flow in general. If a surface is weakly textured, there are usually multiple solutions possible. The L^2 norm matches our needs the best way since it includes the angular and magnitude information of the flow vectors.

5.2.2 Based on Backwarped Image

If optical flow is computed on real images, ground truth optical flow is usually not available. Since in these cases the flow calculation mostly intends to create a perceptually correct flow rather than a mathematically correct one, the *backwarped image* is a potential way to determine the quality of a flow. The backwarped image is created by warping the second frame onto the first one by adding the computed displacements. Let $I_1 : \mathbb{R}^2 \to \mathbb{R}^2$ be the first and $I_2 : \mathbb{R}^2 \to \mathbb{R}^2$ the second image. Furthermore, let $F : \mathbb{N}^2 \to \mathbb{R}^2$ be the calculated flow field. Then the backwarped image $I_B : \mathbb{N}^2 \to \mathbb{R}^2$ is defined as:

$$I_B(\mathbf{x}) \quad := \quad I_2(\mathbf{x} + F(\mathbf{x})) \tag{5.7}$$

The difference between the first image I_1 and the backwarped image I_B determines the size of the error based on the L^2 norm. Similar to the ground truth evaluation the backwarped image error E_{BW} is computed by averaging the errors of all pixels in the image:

$$E_{BW}(I_1, I_B) \quad := \quad \frac{1}{N} \sum_{\mathbf{x} \in \Omega} \|I_1(\mathbf{x}) - I_B(\mathbf{x})\| \tag{5.8}$$

where N denotes the number of pixels. Furthermore, the standard deviation is calculated:

$$\tilde{E}_{BW}(I_1, I_B) \quad := \quad \sqrt{\frac{1}{N-1} \sum_{\mathbf{x} \in \Omega} \left(E_{BW}(I_1, I_B) - \|I_1(\mathbf{x}) - I_B(\mathbf{x})\|\right)^2} \tag{5.9}$$

The backwarped image is an easy way to estimate the visual correctness of a flow computation. Nevertheless, this estimator causes two issues which should be borne in mind. Firstly, changes in illumination, occlusions, appearances and disappearances between both frames increase the error value. Thus, even if the flow field is perfectly correct, the error will hardly be zero. Secondly, a small error value does not necessarily imply that the flow field is correct. The error metric only states if the pixel values of both frames match. This is easily obtained by applying a very small smoothness parameter to the flow calculation. Obviously such a parameter would cause points to move more or less independently from each other. Therefore, the user should make sure that the flow field is adequately smooth before analysing this error value.

5.2.3 Based on Morph Sequence

In many cases optical flow is used to create *intermediate* images between two frames, e.g. for motion compensation. Therefore, it is reasonable to judge the quality of a flow field by the perceptual correctness of these intermediate images. This chapter illustrates three different ways to compute them.

Let I_0 be the first and I_1 the second frame. The following algorithms create intermediate frames $\Im_t$, where $t \in [0, 1]$ determines the interpolation factor. Let $F_{0\to1}$ be the flow field from the first to the second and $F_{1\to0}$ the flow field from the second to the first frame.

5.2.3.1 Splatting

Splatting is the most intuitive way to render the transition between two frames using optical flow. The pixels in the first frame are moved to their positions in the second frame using their respective flow vectors. The intermediate position of a pixel $\mathbf{x}$ in the original image is easily computed in terms of linear interpolation:

$$p(\mathbf{x}, t) \quad := \quad \mathbf{x} + t \cdot F_{0\to 1}(\mathbf{x}) \tag{5.10}$$

Moreover, the approach linearly blends the color of the pixel between the first and the second frame:

$$c(\mathbf{x}, t) \quad := \quad (1-t) \cdot I_0(\mathbf{x}) + t \cdot I_1(\mathbf{x} + F_{0\to 1}(\mathbf{x})) \tag{5.11}$$

The moving pixels are rendered as so-called *splats* in order to avoid holes in the intermediate frames. The following function determines the final color at pixel position $\mathbf{x}$ in the intermediate image.

$$\Im_t(\mathbf{x}) \quad := \quad \omega \cdot \sum_{\mathbf{x}_I \in \Omega} \rho(\|\mathbf{x} - p(\mathbf{x}_I, t)\|^2) \cdot c(\mathbf{x}_I, t) \tag{5.12}$$

where $\omega = 1/(\sum_{\mathbf{x}_I \in \Omega} \rho(\|\mathbf{x} - p(\mathbf{x}_I, t)\|^2))$ is a normalization factor.

In our implementation we use the quadratic filter kernel $\rho(d) := \left(\frac{1}{0.2+d}\right)^2$. In order to increase the performance, we limit the kernel window to a small quadratic area around the pixel. Additionally rather than searching for all pixels in the neighborhood of $\mathbf{x}$ we go the other way round: First, we clear the intermediate image $\Im$ with black. Afterwards all pixels of the original images are translated to their positions at time t and add their weighted color to the pixels around that position within the kernel window. After normalization the image represents the intermediate frame.

5.2.3.2 Bidirectional Splatting

If parts of objects appear in the second image, which were not visible before, the splatting approach will probably produce holes since there is no pixel correspondence between the two frames for these areas. Though, an increased kernel window avoids the holes to be *black*, they are filled with wrong color values of distant pixels. Bidirectional splatting fills holes due to appearance by incorporating the flow field in the opposite direction. Pixels which appear from the first to the second frame *disappear* from the second to the first one. Accordingly, the necessary color information is available.

Bidirectional splatting applies the simple splatting approach in both directions and interpolates between them. Therefore, the flow field $F_{1\to 0}$ is also required. We generalize the position and the color interpolation function to arbitrary flow fields from frame i to j:

$$p(\mathbf{x}, t)_{i\to j} \quad := \quad \mathbf{x} + t \cdot F_{i\to j}(\mathbf{x}) \tag{5.13}$$

$$c(\mathbf{x}, t)_{i\to j} \quad := \quad (1-t) \cdot I_i(\mathbf{x}) + t \cdot I_j(\mathbf{x} + t \cdot F_{i\to j}(\mathbf{x})) \tag{5.14}$$

The intermediate image is then calculated by:

$$\begin{aligned}\Im_t(\mathbf{x}) := \quad \omega_{BD} \cdot \sum_{\mathbf{x}_I \in \Omega} (\quad & \rho(\|\mathbf{x} - p(\mathbf{x}_I, t)_{0\to1}\|^2) \cdot c(\mathbf{x}_I, t)_{0\to1} + \\ & \rho(\|\mathbf{x} - p(\mathbf{x}_I, 1-t)_{1\to0}\|^2) \cdot c(\mathbf{x}_I, 1-t)_{1\to0} \quad)\end{aligned}$$

where

$$\omega_{BD} \quad = \quad 1/\left(\sum_{\mathbf{x}_I \in \Omega} \rho\left(\|\mathbf{x} - p(\mathbf{x}_I, t)_{0\to1}\|^2\right) + \sum_{\mathbf{x}_I \in \Omega} \rho\left(\|\mathbf{x} - p(\mathbf{x}_I, 1-t)_{1\to0}\|^2\right) \right)$$

again normalizes the color accumulation.

This approach avoids holes due to appearance. However, it only works satisfyingly if the flows in both directions are consistent. If the correspondences are not correct, the respective bidirectional splats interfere with each other and cause noisy intermediate images.

5.2.3.3 Sampling

Both of the previous approaches can still produce holes in the intermediate image since splatting does not map the pixels in a bijective matter: multiple pixels can be mapped to one position. The *sampling* approach produces intermediate images which do not contain any holes by the following function:

$$\Im_t(\mathbf{x}) \quad := \quad (1-t) \cdot I_0(\mathbf{x} - t \cdot F_{0\to1}(\mathbf{x})) + t \cdot I_1(\mathbf{x} - (1-t) \cdot F_{1\to0}(\mathbf{x}))$$

This approach assumes that the flow field is rigid. Strictly speaking, given a flow field F the equation $F(\mathbf{x}) = F(\mathbf{x} - F(\mathbf{x}))$ must hold. Even though its assumption is never correct in practice, this approach produces visually good results for small displacements.

The simple splatting approach is the best way to visually evaluate the correctness of a flow field. The two other approaches produce visually better images, which is particularly useful to align frames in a sequence (see section 5.4.2.1). A drawback of all methods which are based on morphing is the lack of a mathematical model to evaluate the intermediate images. Thus, it requires the viewer to have a good conception of the correct flow.

5.3 Synthetic Data

In this section we judge the quality of our enriched optical flow algorithm based on synthetic renderings and calculated ground truth optical flow. For each frame of an experiment we produce three kinds of renderings: the *standard views*, which exhibit the scene using texture maps and illumination, the *normal maps* and the *depth discontinuity maps*.

In each experiment we firstly compare three different input configurations.

1. **Color**: The three color channels of the standard views (red, green, blue)
2. **Normal**: The three channels of the normal map (x, y, z)
3. **Color + Normal**: The combination of the standard views and the normal maps which results in six input channels.

Furthermore, we test the influence of our binary depth discontinuity maps to the flow quality. We balance the input channels as a preprocessing step in order to apply the same data weight for all channels ($\gamma_i = 1$, see equation 3.10 in section 3.3): The components of the normal maps are mapped to $[0, 1]$ (*offset gray*). Moreover, since PBRT creates HDR images, we *normalize* the standard views: we divide each color channel by its maximum value in both views in order to scale the high-dynamic range images down to $[0, 1]$. We found the approximately "best" smoothness value for each input setting by binary search in the L^2 error function as the relation between smoothness and L^2 error is roughly a convex function (see section 3.6). All flow fields were computed by using the scale factor $\eta = 0.9$, five inner fixed point iteration and 100 SOR iterations.

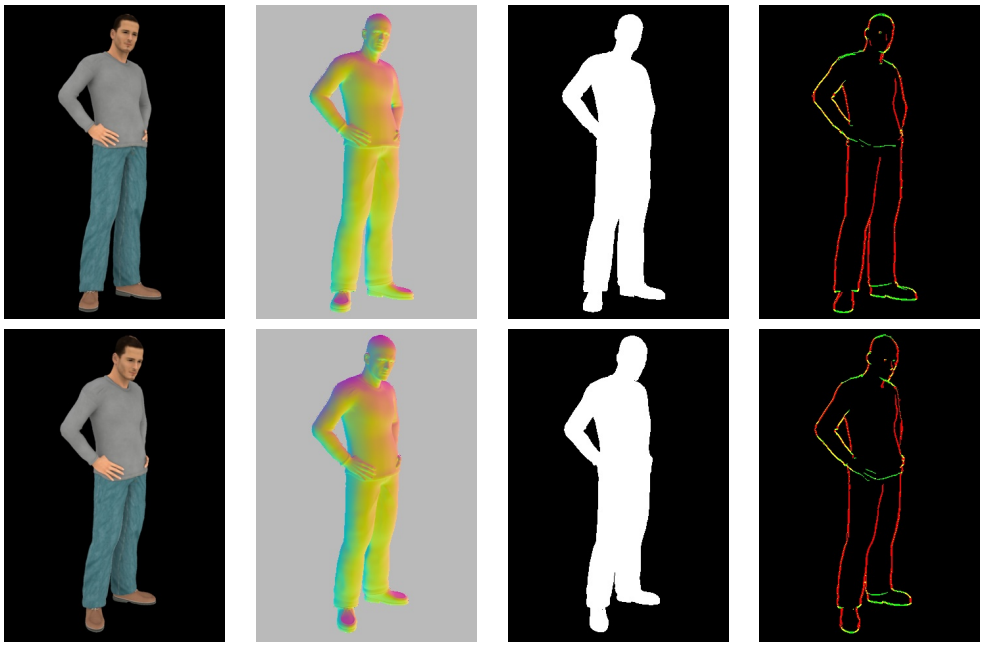
5.3.1 Static Scene

This subsection deals with synthetic static scenes where only the viewpoint changes. The problem of finding the displacements in such a scene is known as *stereo matching*.

5.3.1.1 Poser Model

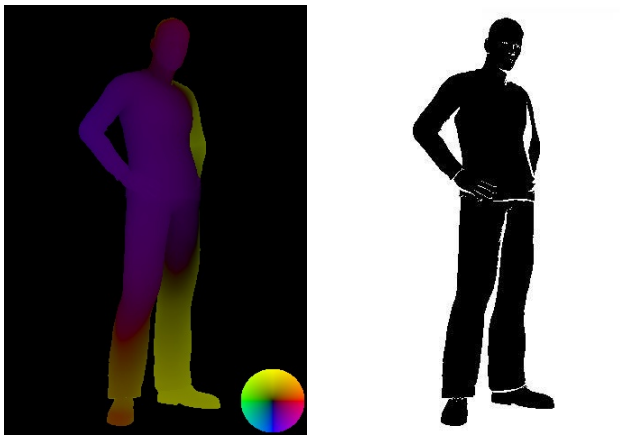
First of all, we rendered a synthetic Poser Model in 336×477 pixels and slightly rotated the camera from the first to the second frame. We used PBRT to create two fully lit standard views and two normal maps using the gradient approach as explained in sections 4.1.2 and 5.1.1.5. Next the color range of the standard views and the normal maps are balanced as explained above. You can see the resulting renderings in figures 5.2a-b.

Moreover, we obtain the depth discontinuities by the Raskar shadow images as explained in section 5.1.1.5 with a low threshold of $\varepsilon_{\text{Low}} = 1$ and a high threshold of $\varepsilon_{\text{High}} = 1.5$. Since the background is black and does not exhibit any shadows, we enrich the depth discontinuity maps with the discontinuities in the background mattes in order to obtain the *external discontinuities* (the silhouette) as well. As mentioned before, we can easily render the mattes using OpenGL (see figure 5.2c). We add a depth discontinuity wherever the discrete derivative of the matte is non-zero. The final depth discontinuity maps are shown in figure 5.2d. The ground truth estimation based on the ray-tracing technique and the corresponding void-map, which excludes all undefined pixels, can be viewed in figure 5.3.



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

Figure 5.2: Static scene with synthetic Poser model and changing viewpoint. Top: First frame. Bottom: Second frame. (a) PBRT rendering after normalization. (b) PBRT gradient normal maps in grayscale. (c) Mattes. (d) Depth discontinuities created with Raskar's shadow approach and the silhouette of the matte.



(a) (b)

Figure 5.3: (a) Ground truth optical flow. (b) Void-map where white pixels correspond to undefined pixels.

Table 5.1 compares the average angular errors in degree (AAE), the average magnitude errors (AME) and the average L^2 norm errors (AL2E) of the com-

puted flow fields using the different input configurations. Apparently, running the flow algorithm on the normal maps produces a better result than using the standard views only. The normal map gives a better clue about the surface of the model than the weak texture on the shirts and the trousers. Consequently, the correspondence between the frames can be determined more precisely. The combination of color and normal results in an even better flow. The six input channels seem to complement each other: if the structure of the texture is weak, the normal map might help. Vice versa, if the normal map information is insufficient to find a reliable displacement, the texture information supports the estimation.

Input Channel	AAE	AME	AL2E	Smoothness	RT
Color	5.065 (0.280)	0.652 (1.432)	**0.924** (2.036)	0.011758	36.1s
Normal	3.263 (0.206)	0.410 (0.935)	**0.581** (1.373)	0.018124	36.3s
Color + Normal	3.195 (0.222)	0.337 (0.915)	**0.514** (1.536)	0.013799	39.6s

Table 5.1: Results of the flow computation using different input channels at static Poser scene. The brackets refer to the standard deviations.

Interestingly, the additional three image channels in the "Color + Normal" mode just slightly increase the run-time (RT) by about 3.3 seconds. The creation of the additional image pyramids for the extra channels takes about 0.8 seconds, where the remaining 2.5 seconds are consumed by the minimization process.

Finally, we repeated the flow estimation using the depth discontinuity maps, where we kept the global smoothness value as general smoothness ($\alpha_G = \alpha_{Gl}$). We used a lower boundary smoothness weight α_B, which is at least half of the global smoothness value. As in all following experiments we started to apply the discontinuity maps up from 20% of the image resolution. Table 5.2 displays the results.

Input Channel	AAE	AME	AL2E	α_B	RT
Color	4.527 (0.267)	0.486 (1.165)	**0.747** (1.780)	0.002879	36.0s
Normal	3.209 (0.214)	0.373 (1.011)	**0.552** (1.621)	0.009062	36.3s
Color + Normal	2.899 (0.201)	0.309 (0.874)	**0.471** (1.381)	0.006899	39.5s

Table 5.2: Results of the flow computation using different input channels at static Poser scene with depth discontinuities.

In all cases the knowledge about the depth discontinuity maps significantly reduces the L^2 error, especially at the right arm of the model. Figure 5.4 illustrates the impact of these maps. The error of the flow estimation is effectively reduced at the boundaries but the runtime does not noticeably change due to the depth discontinuity maps.

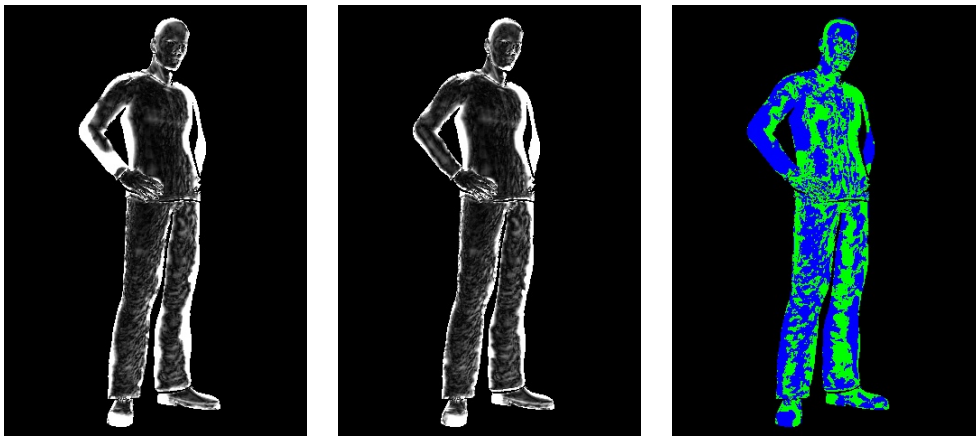


Figure 5.4: Comparison between flow with and without discontinuities. (a) L^2 error without discontinuities. (b) L^2 error using discontinuities. (c) Binary quality comparison map. Green: Flow without discontinuities yields lower L^2 error. Blue: Flow with applied discontinuities produces lower error.

5.3.1.2 Grand Canyon Scene

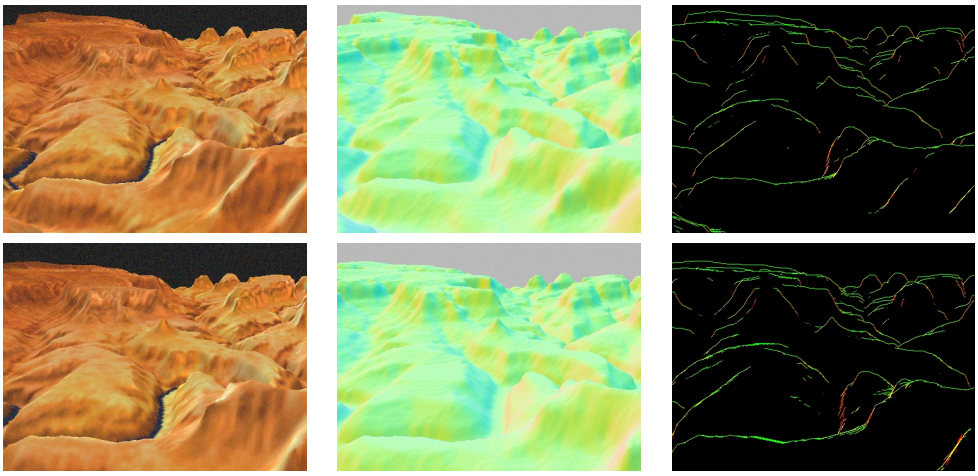
We compared the influence of the input channels on a second static scene similar to the popular "Yosemite sequence" [2]. We refrained from testing that sequence, which is commonly used in literature, because no normal maps were available. In addition, the published ground truth field seemed to be erroneous [BBW06].

We created a 3D model of the Grand Canyon national park by using the high map and texture map published in Hoppe [Hop98] [3]. We rendered the canyon and its normal maps from two viewpoints in 480×360 pixels using OpenGL. The camera in the second view is moved forwards and slightly turned. We added Gaussian noise with $\sigma_R = 0.02$ (see section 5.1.1.4) to the standard views and the normal maps. We obtained the depth discontinuity map by evaluating the z-buffer as described in section 5.1.1.3 using the depth discontinuity threshold $\varepsilon_D = 6 \cdot 10^{-5}$. You can see all renderings in figure 5.5.

The quality measurement over the whole image is dominated by the large errors in the bottom right part. Since we want to obtain a good impression of the quality difference induced by the different input channels, we restrict the calculation of the flow errors to the top part of the image (see red rectangle in figure 5.6). Table 5.3 shows the results of the quality estimation depending on the different image channels. Figure 5.7 exhibits the corresponding error images, which map each flow vector to its L^2 error value. Even though running the optical flow algorithm on the normal channels yields a worse average L^2 error than on the standard views, it finds better correspondences in the top left and the bottom right part of the image (figures 5.7a-b). Figure 5.7c illustrates this difference:

[2] The Yosemite sequence can be downloaded at http://www.cs.brown.edu/ black/.

[3] You can find the source maps at ftp://ftp.research.microsoft.com/users/hhoppe/data/gcanyon/.



(a) (b) (c)

Figure 5.5: Static scene with Grand Canyon model and changing viewpoint. (a) Standard views. (b) Offset gray normal maps. (c) Depth discontinuities.

In all green areas the optical flow based on the color maps yields a better result. Vice versa, in the blue areas the normal maps produce better results. Again we obtain the best result by combining the standard views and normal maps. Apparently, both channels complement each-other as shown in figures 5.7c-d.

Input Channel	AAE	AME	AL2E	Smoothness	RT
Color	1.309 (0.045)	0.551 (1.617)	**0.769** (1.907)	0.025536	39.2s
Normal	1.367 (0.091)	0.657 (2.890)	**0.873** (3.374)	0.029663	39.2s
Color + Normal	0.960 (0.063)	0.393 (1.495)	**0.548** (1.806)	0.042576	42.9s

Table 5.3: Results of the flow computation using different input channels at static Grand Canyon scene.

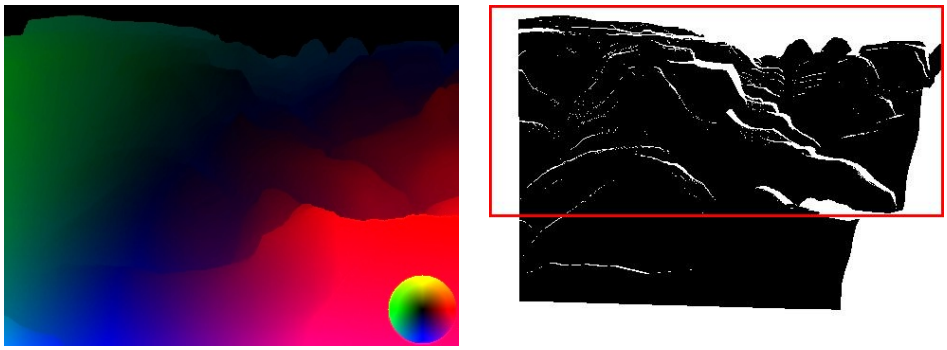


Figure 5.6: Ground truth of Grand Canyon scene. Left: Ground truth optical flow. Right: Void map. Our region of interest is marked by the red rectangle.

Input Channel	AAE	AME	AL2E	α_B	RT
Color	1.281 (0.046)	0.532 (1.618)	**0.746** (1.909)	0.012768	39.6s
Normal	1.276 (0.082)	0.617 (2.838)	**0.819** (3.223)	0.021288	39.3s
Color + Normal	0.942 (0.063)	0.353 (1.276)	**0.509** (1.619)	0.014831	42.9s

Table 5.4: Results of the flow computation at static Grand Canyon scene with depth discontinuities.

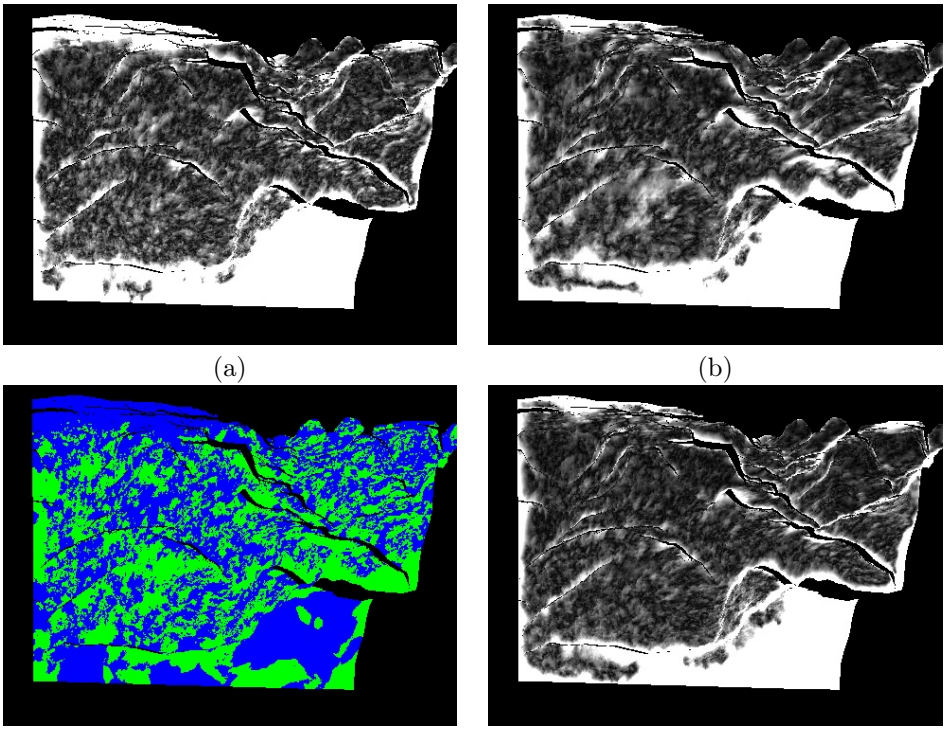


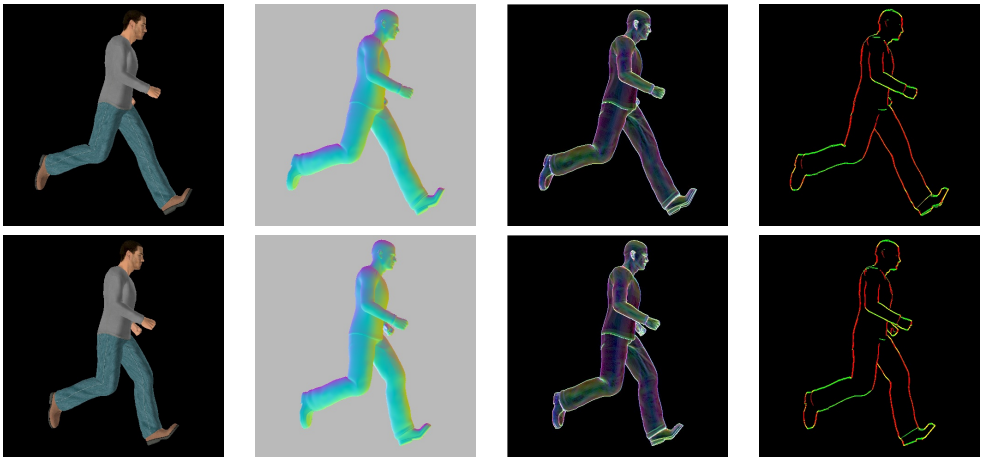
Figure 5.7: L^2 error maps of flow fields for Grand Canyon scene. Used image channels: (a) Colors only. (b) Normals only. (c) Binary comparison map. Green: L^2 error using *colors* is lower. Blue: L^2 error using *normals* is lower. (d) Color and normals.

Again we applied the depth discontinuity maps in a second pass. As stated in table 5.4, these maps slightly improve the flow estimation. The improvement is not as strong as before: The discontinuity boundaries in flow fields are sharpened but the robust statistic in our basis optical algorithm already preserves these discontinuities quite well.

5.3.2 Dynamic Subject

We executed the same experiment as in section 5.3.1.1 using a Poser model but this time we kept the viewpoint constant while deforming the model. We rendered two intermediate frames of a running animation. Figure 5.8 exhibits the

renderings. The corresponding ground truth optical flow and the void-map are shown in figure 5.9.



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

Figure 5.8: Dynamic scene with moving Poser model and static viewpoint. Top: First frame. Bottom: Second frame. (a) PBRT rendering after normalization. (b) PBRT normal maps in grayscale. (c) Gradients of normal maps. (d) Depth discontinuities created with Raskar's shadow approach and the silhouette of the matte.

Table 5.5 compares the quality of the optical flow using different input channels. The flow calculation on the normal maps produces significantly worse results than the computation on the standard views. This is not surprising since the deformation of the model implies that the *surface normal constancy assumption* is violated: the surface normals change from one frame to the next. Consequently, the algorithm cannot find the right correspondences. A combination of the color and the normal images produces slightly better results but probably because the color channels partly compensate the errors induced by the normal map.

Input Channel	AAE	AME	AL2E	Smoothness	RT
Color	8.188 (0.302)	0.643 (1.276)	**1.082** (1.684)	0.014824	52.991s
Normal	13.414 (0.298)	1.117 (1.311)	**2.056** (1.963)	0.036777	53.045s
Color + Normal	10.486 (0.315)	0.847 (1.140)	**1.450** (1.760)	0.022383	58.291s
∇ Normal	7.399 (0.200)	0.662 (1.242)	**1.374** (2.033)	0.004516	54.391s
Color + ∇ Normal	8.434 (0.315)	0.591 (1.154)	**1.055** (1.634)	0.022383	58.259s

Table 5.5: Results of the flow computation using different input channels at dynamic Poser scene.

Running optical flow just on the normal maps does not help if the subject is deformed. However, a new constraint, which we call the *surface gradient*

(a)

(b)

Figure 5.9: Dynamic Poser model. (a) Ground truth optical flow. (b) Void-map.

constancy assumption, might hold: the derivative of the normals should be invariant at parts of the object, which rigidly move. We tested this idea by running the optical flow on the *gradient images* of the normals instead of on the normals themselves. The gradient image encodes the magnitude of the image gradients per color channel. The results are listed in table 5.5 as well.
The normal gradient maps yield better results than the normal maps but they do not outperform the flow fields running on the standard views only (see figure 5.10). The flow field on the gradient images contains lower L^2 errors at the hip and the head but the normal derivatives strongly change on the legs and the arm, which gives rise to high errors in these areas. Combining the color and the normal gradient channels yield an slightly better average L^2 error because the displacements at the head are more precise (figure 5.10d).

There are some situations where the gradient of the surface normals can never support the flow estimation. If a surface is deformed at some place the respective normals and their derivatives change. The normal gradient may only improve the result at rigidly moving parts. Moreover, the gradient of the normal is not invariant under perspective foreshortening, e.g. when an object is turned away from the user.

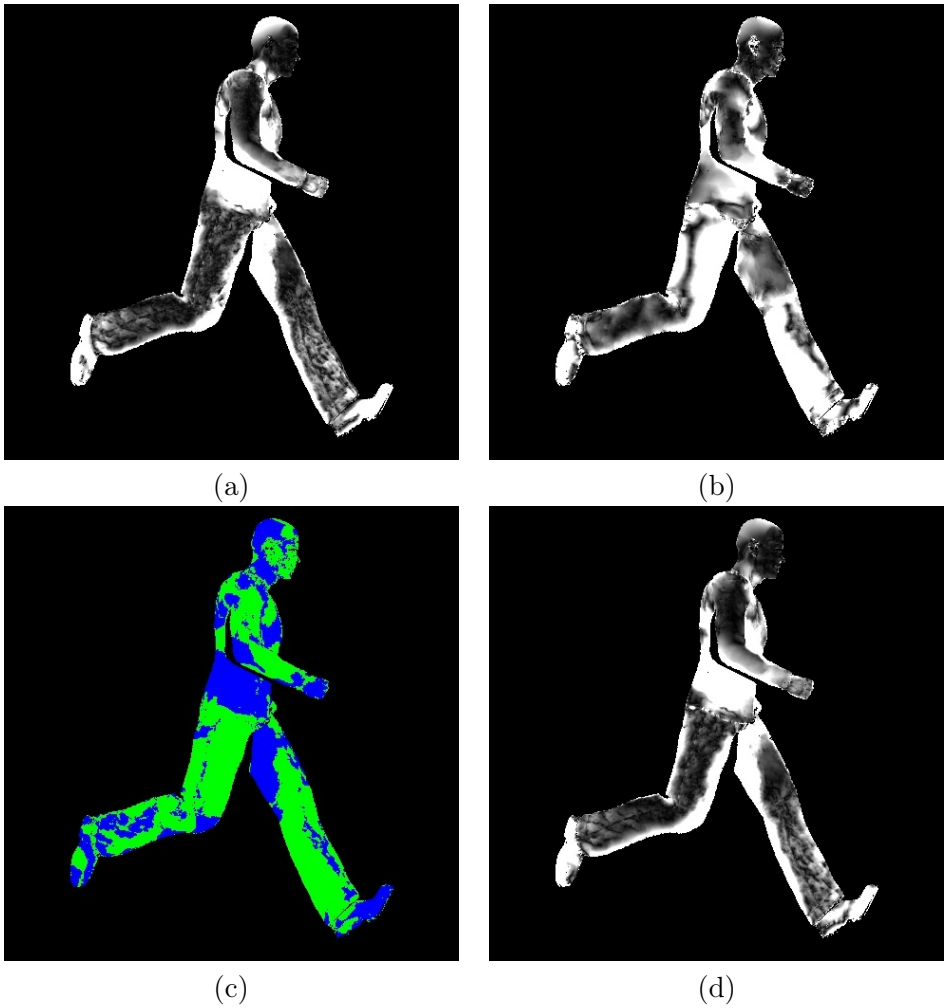
Input Channel	AAE	AME	AL2E	α_B	RT
Color	8.097 (0.302)	0.599 (1.201)	**1.028** (1.613)	0.007412	52.9s
∇ Normal	7.435 (0.205)	0.651 (1.218)	**1.350** (2.036)	0.001258	53.8s
Color + ∇ Normal	8.430 (0.316)	0.566 (1.111)	**1.034** (1.609)	0.006201	57.7s

Table 5.6: Results of the flow computation using different input channels at dynamic Poser scene with depth discontinuities.

We finally applied the depth discontinuity maps. The results are shown in table 5.6. Although the average L^2 error is just slightly lowered, the flow fields are noticeably advanced on the right arm of the model of the running subject as shown in figure 5.11. The flow field becomes piecewise smooth at the arm which visibly improves the flow estimation.

5.4 Real Data

Synthetic test cases are usually very idealized and fulfill assumptions that are commonly violated in natural images. This section deals with optical flow esti-



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

Figure 5.10: L^2 error maps of flow fields for dynamic Poser model scene. Used image channels: (a) Colors only. (b) ∇ Normals only. (c) Binary comparison map. Green: L^2 error using *colors* is lower. Blue: L^2 error using *normal gradients* is lower. (d) Color + ∇ Normals.

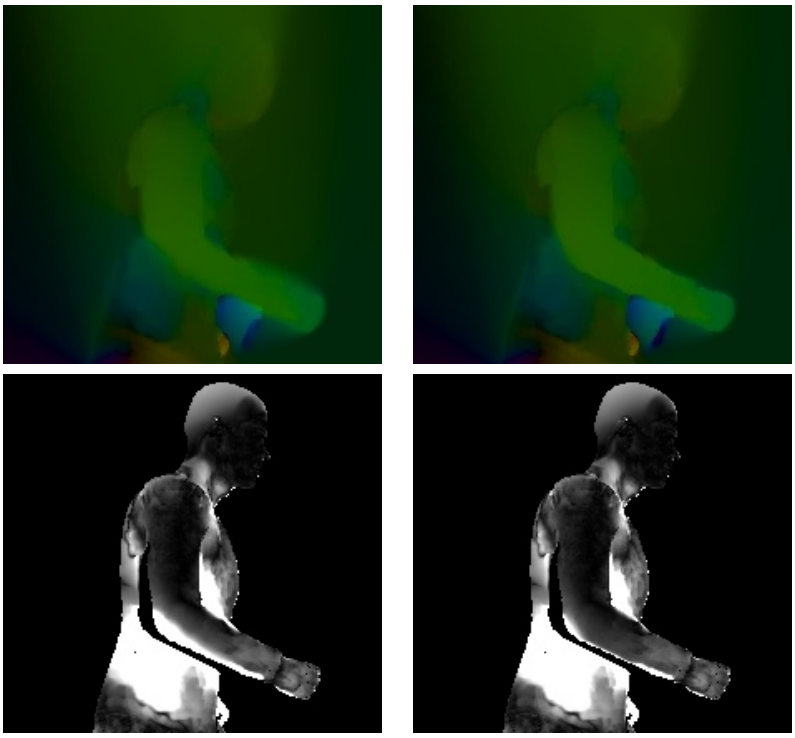


Figure 5.11: Influence of discontinuity maps to the flow estimation based on standard views (cropped area). Top: Flow fields. Bottom: L^2 error maps. Left: Without discontinuity maps. Right: With discontinuity maps.

mation on natural images. Again, we distinguish between static scenes (stereo matching) and dynamic scenes.

5.4.1 Static Scene

We captured a static mannequin wearing street clothes from different viewpoints using the Light Stage 6 apparatus (see appendix C) under different lighting conditions. From the first to the second frame the camera slightly rotates counter-clockwise around the y-axis of the stage. We create the standard views by activating all lights in the stage. The normal maps are created with the gradient approach described in section 4.1.2. We cropped the resulting images to a resolution of 603×1201. Moreover, we manually matted out the background because it varies strongly from one viewpoint to the next and could bias the flow estimation. You find all images in figure 5.12. Since no ground truth is available, the global smoothness value cannot be found using binary search. We empirically determined an appropriate smoothness value ($\alpha_{Gl} = 0.05$) by evaluation of the resulting splatting sequences (see section 5.2.3.1). Furthermore, we chose a high scale factor of $\eta = 0.95$ and computed the flow fields with five inner fixed point iteration and 200 SOR iterations.

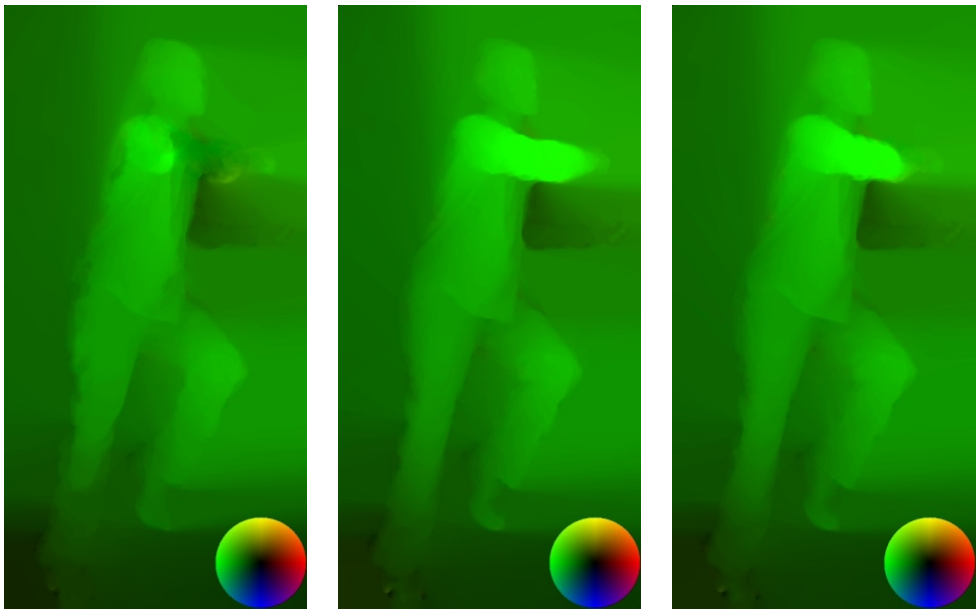
(a) (b) (c) (d)

Figure 5.12: (a)-(b) Standard views of both frames. (c)-(d) Normal maps of both frames.

The flow computation on the standard views or the normal maps took about ten minutes (606 seconds), where the estimation on the combination of all input channels needed nearly 11.5 minutes (683 seconds).

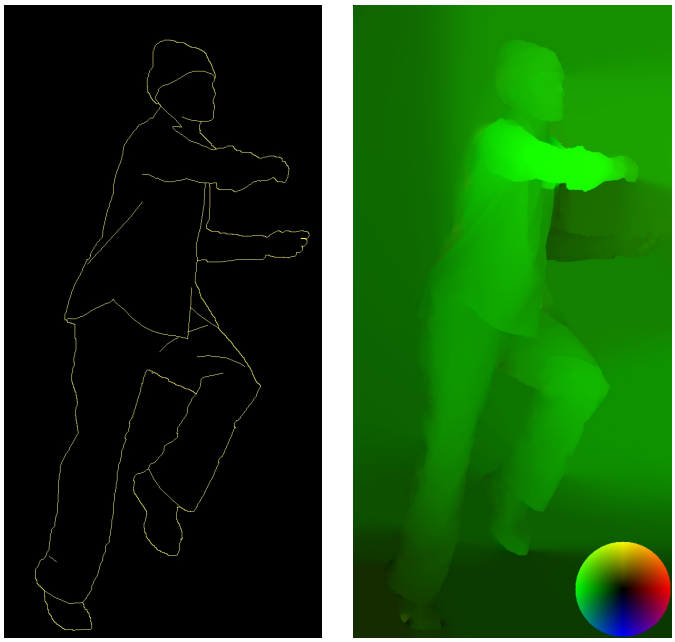
The resulting flow fields are shown in figure 5.13. Both, the color and the normal channel, find roughly the correct displacements to the left but the flows based on the color information produces serious errors on the right arm of the mannequin. Due to the repetitive texture, the optical flow algorithm is not capable to find the right correspondences on that limb (figure 5.13a). However, the normal maps provide enough unique structural information to determine the right displacement as shown in figure 5.13b: the bright green area indicates the large flow vectors of the arm, which is nearer to the camera than the other body parts and, therefore, moves faster due to the rotation. Furthermore, the normal maps create a more rigid flow field on the back and at the legs of the mannequin. The combination of both, color and normal maps, gives rise to a rigid flow field in most body parts but the right arm still presents noticeable issues due to the errors, which are introduced by the color map (figure 5.13c).

This experiment is an extreme case in which the repetitive texture structure – the stripes on the shirt of the mannequin – irritates the optical flow estimation, even though a high scale factor is chosen, which ensures a smooth transition between the different scales of the coarse-to-fine approach. In this case the normal maps contain distinct clues for the displacements of the surface points and should be preferred to the standard views. Of course, the opposite situation is imaginable: a repetitive normal map structure in combination with a nonrecurrent texture pattern. Accordingly, the importance of the standard views and the normal maps should be weighted depending on the objects in the scene.



(a) (b) (c)

Figure 5.13: Flow fields depending on input channels. (a) Colors. (b) Normals. (c) Colors + Normals.



(a) (b)

Figure 5.14: Application of depth discontinuities. (a) Depth discontinuity map. (b) Flow using colors, normals and depth discontinuity map.

We manually created a simple depth discontinuity map for this dataset which is shown in figure 5.14a. We then repeated the experiment using color and normal information but applied this discontinuity map using the boundary smoothness value $\alpha_B = 0.001$. The calculation needed 676 seconds, i.e. again there is no significant run-time difference to the computation without depth discontinuities. Figure 5.14b exhibits the result. The discontinuity maps yield a piecewise smooth flow field and correct the errors at the arm introduced by the wrong color matching. They avoid the area around rigidly moving objects to bias the flow estimation.

5.4.2 Dynamic Subject

For a final evaluation, we treat natural images of moving subjects. This case is non-trivial since the normal and discontinuity maps cannot be retrieved simultaneously. If the subject moves, the acquired standard views, the normal maps and the depth discontinuity maps are shifted in time, i.e. we are working on *time demultiplexed data*. In this section we firstly introduce a field of application where optical flow crucially contributes to its success. Secondly, we explain our test data, a shoot of a human performance captured with a high speed camera. Afterwards we present an optical flow framework which processes this data and, finally, we discuss the results.

5.4.2.1 Image Based Relighting and Optical Flow

Image Based Relighting (IBR) is a research field of Computer Graphics which generates photo realistic images while refraining from the creation of 3D models. Usually, a subject is illuminated and captured under a set of light conditions, the so-called *lighting basis*. This basis samples the incident lighting from each direction of a (hemi)sphere. Due to the linear property of light the light conditions can be weighted and added up in order to obtain an *arbitrary* illumination of the subject. This technique is explained fundamentally in [DHT$^+$00], [HWT$^+$04], and [WGT$^+$05].

IBR can easily be applied to static scenes, which do not contain any motion at all. The scene is captured under the whole lighting basis and a linear combination of the acquired images yields the relit result. This is illustrated in figure 5.15.

However, as soon as a subject moves the problem becomes complicated: The subject changes the position from frame to frame, which means that the lighting conditions are *not aligned*. If the images are simply added up, the resulting images look considerably blurred, even if the subject is shot at a high frame rate.

Optical flow helps to solve the problem of dynamic scenes as demonstrated in figure 5.16. So-called *tracking frames* are added to the lighting conditions in a certain interval. Tracking frames are images in which the subject is uniformly

lit from all directions, and hardly contain any shadows. The latter is important since brightness changes between tracking frames are undesirable for the next step: The optical flow fields between the tracking frames are computed.

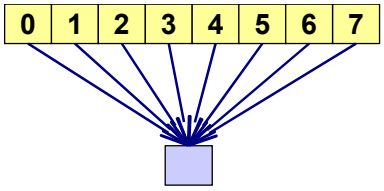


Figure 5.15: Principle of Image Based Relighting. A scene is captured under different light conditions. The resulting images (yellow boxes) are then linearly combined to obtain an arbitrary relit version of the scene (light blue box).

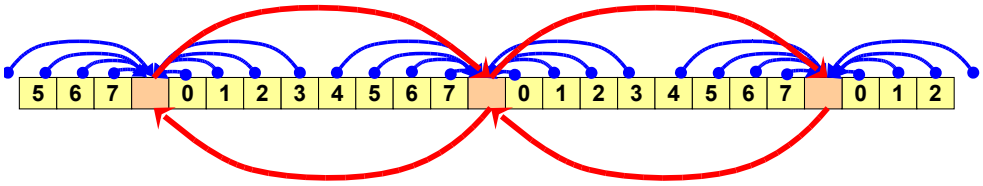


Figure 5.16: Tracking frames (red boxes) are added between the lighting basis. The flows (red arrows) between these tracking frames are used to temporarily align (blue arrows) all frames of the lighting basis (yellow boxes).

In the following procedure these flow fields are used to temporarily *align* the neighbored frames to the tracking frame. Let I_T^i and I_T^{i+s} be two subsequent tracking frames where $s-1$ is the number of lighting conditions between these frames. Furthermore, let I^{i+k} be a regular frame ($0 < k < s$). We linearly interpolate the optical flow between the two tracking frames, i.e. we assume that the flow from I^{i+k} to I_T^{i+s} equals $\frac{k}{s}$-th of the flow from the tracking frame I_T^i to tracking frame I_T^{i+s}. This applies analogously to the flow in the other direction. The idea is illustrated in figure 5.17. Apparently, the linear interpolation is only valid if the motion between the tracking frames is linear. Thus, the tracking frames should not be placed too far from each other. Using the interpolated flow frame I^{i+k} can be aligned to the next tracking frame, e.g. using the backwarp approach in section 5.2.2 in order to *multiplex* the frames to one point in time. In the easiest case the tracking frames are placed in the beginning of each lighting basis. If the basis contains a lot of lighting conditions, it is sensible to add additional tracking frames within the basis vectors. Subsequently, a two-stage alignment approach is applied as visualized in figure 5.18. Firstly, all frames are aligned to their nearest tracking frames as just described. Then the flow fields between the tracking frames and a particular *output frame*, which exists once per lighting basis, are computed. These fields are finally used to warp the prealigned frames to the output image.

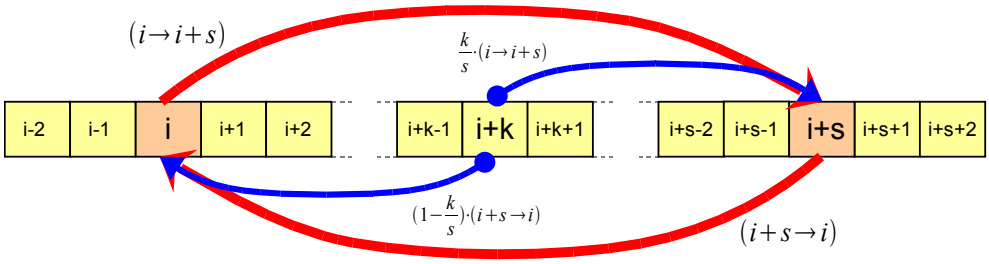


Figure 5.17: A frame in the sequence, I^{i+k}, is aligned to the adjacent tracking frames, I_T^i and I_T^{i+s}, using the linear interpolation (blue arrows) of the tracking frame flows (red arrows).

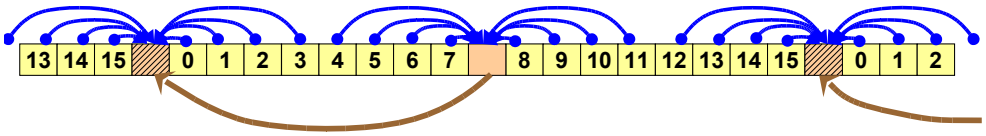


Figure 5.18: Again, tracking frames (red boxes) are added within the lighting basis. First the frames are aligned (blue arrows) to their next tracking frames. Afterwards these prealigned frames are warped (brown arrows) to the final output frames (hatched boxes).

5.4.2.2 The Idea

In this experiment we compare the tracking frame technique as explained above with *extended tracking frame alignment*: we add *auxiliary frames* next to the full-lit tracking frames which provide normal and depth discontinuity maps. These frames are aligned to the tracking frame and used to improve the optical flow estimation (see figure 5.19). Consequently, we compare the regular flows between the tracking frames with the flows of the *enriched* tracking frames.

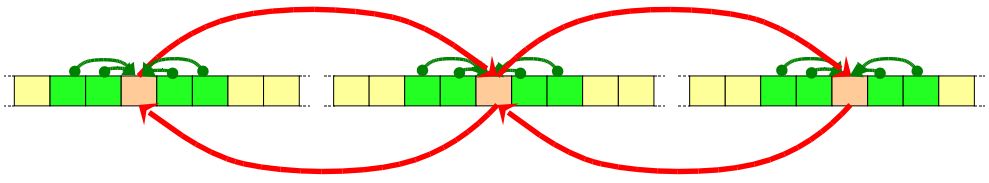


Figure 5.19: Extended tracking frame alignment. Additional frames (green) are added next to the tracking frames to enrich the flow estimation.

5.4.2.3 Data acquisition

We captured a moving subject in the Light Stage 6 (see appendix C) using a Vision Research Phantom V10 high-speed camera. The subject is running on a treadmill while being illuminated under different lighting conditions. Our *lighting basis* consists of nine different illumination patterns as illustrated in figure 5.20. Each lighting condition requires the activation of at least 15 light sources to provide a sufficient exposure. The lighting basis contains:

- Four patterns in order to produce the **Raskar discontinuity maps**. The patterns contain light sources directly left, above, right, and below the camera respectively. Since one single light source is not bright enough, we approximate the flash light sources next to the camera by strips of 15 activated light sources in each direction.
- One full-lit pattern, the **tracking frame**.
- Five **photometric stereo patterns**. Again, we need multiple light sources to provide a sufficient brightness. We enable five distinct circles of 15 light sources to simulate directional lights.

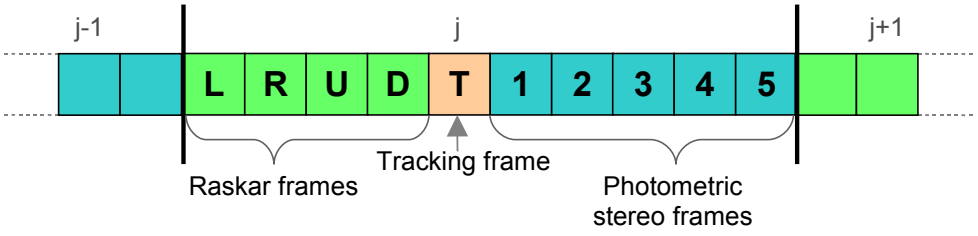


Figure 5.20: Lighting basis consisting of nine lighting conditions: four Raskar frames, one full-lit frames and five photometric stereo frames.

Figure 5.21 visualizes which light sources are activated per lighting condition. Furthermore, figure 5.22 exhibits one sample sequence taken under all lighting conditions. We shoot the performance using a 28mm lens at the aperture of $f/2$. The images are acquired at 300 frames per seconds. Each frame is exposed for $3100\mu s$, which results in an unused interval of $233\mu s$ ($3100\mu s + 233\mu s = \frac{1s}{300}$).

Before the subject begins to perform, we shoot a **clean-plate** by capturing the scene without the subject. The clean-plate is used to obtain a matte, which distinguishes our subject from the background. Commonly, the matte is used to remove the background when the subject is relit in the context of Image Based Relighting. In our case we retrieve the external discontinuities from it in order to avoid the background noise to distract the flow estimation.

The treadmill starts about one minute before the camera begins the data acquisition. The subject may use this period to adapt himself to the running

performance. Then the person is signaled by a special lighting pattern five seconds before the recording.

5.4.2.4 Color correction

Beside the clean-plate we shot a *session reference* in advance. The lens is covered by its cap and the resulting *black level* is recorded which the camera subtracts from the output images in order to reduce noise effects and to map black pixels to the origin of the RGB color space.

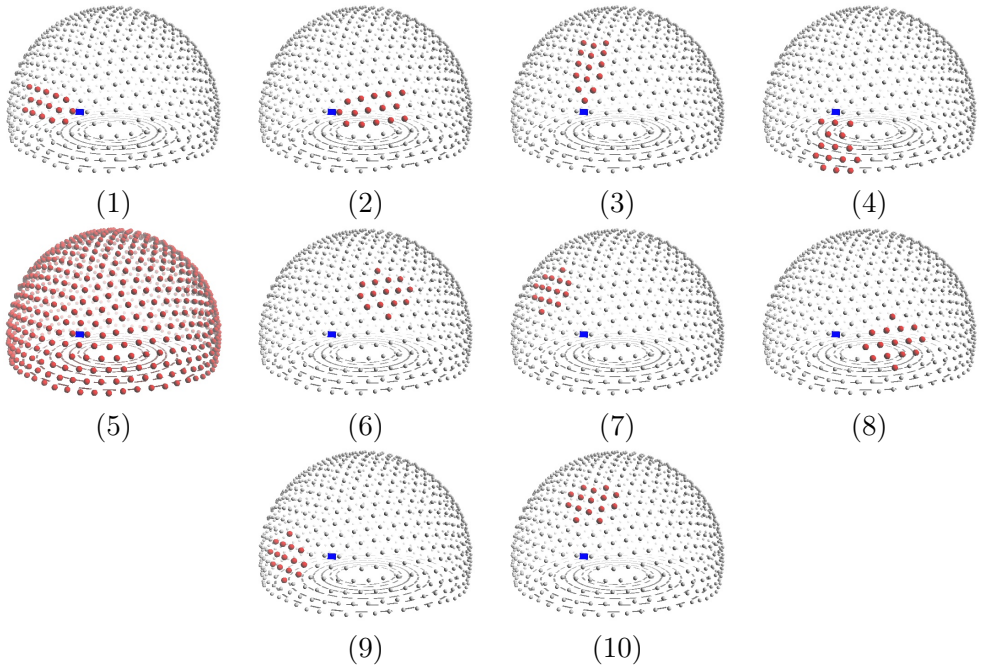


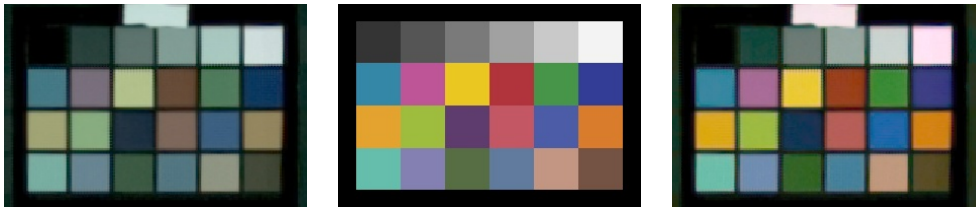
Figure 5.21: Lighting basis. Each representation exhibits which light sources are activated in each frame (red spheres) and which are not (gray spheres). The lighting conditions are arranged around the camera (blue rectangle). (1)-(4) Raskar pattern, (5) Tracking frame, (6)-(10) Photometric stereo frames.

Figure 5.22: Sample sequence captured under the entire lighting basis

In addition, we perform a color correction using a MacBeth color chart, a palette of colors whose *ground truth color values* are known. Figure 5.23b displays the ground truth MacBeth color charts and figure 5.23a shows the chart as we acquired it with a high speed camera. We compute a linear map between the acquired and the ground truth color values by finding the least-squares match. This results in a 3×3-matrix M_C:

$$M_C = \begin{bmatrix} 2.918764 & -0.829588 & -0.154388 \\ -0.372775 & 1.441435 & -0.055011 \\ -0.010895 & -1.125107 & 2.425955 \end{bmatrix}$$

All pixels in the image sequence are transformed with this matrix. Figures 5.23c and 5.24 exhibit the difference between the raw and the color-corrected images.



(a) (b) (c)

Figure 5.23: MacBeth color charts. (a) Captured MacBeth chart. (b) Ground truth. (c) Color-corrected version of acquired MacBeth chart.

Figure 5.24: Color correction. Left: Raw image. Right: Corrected image using color matrix M_C.

5.4.2.5 Algorithm

This subsection explains the algorithm which we use to evaluate the flows on extended tracking frames. The procedure is sketched in figure 5.25. We computed all flow fields using the scale factor $\eta = 0.9$, five inner fixed point iterations and 100 SOR iterations.

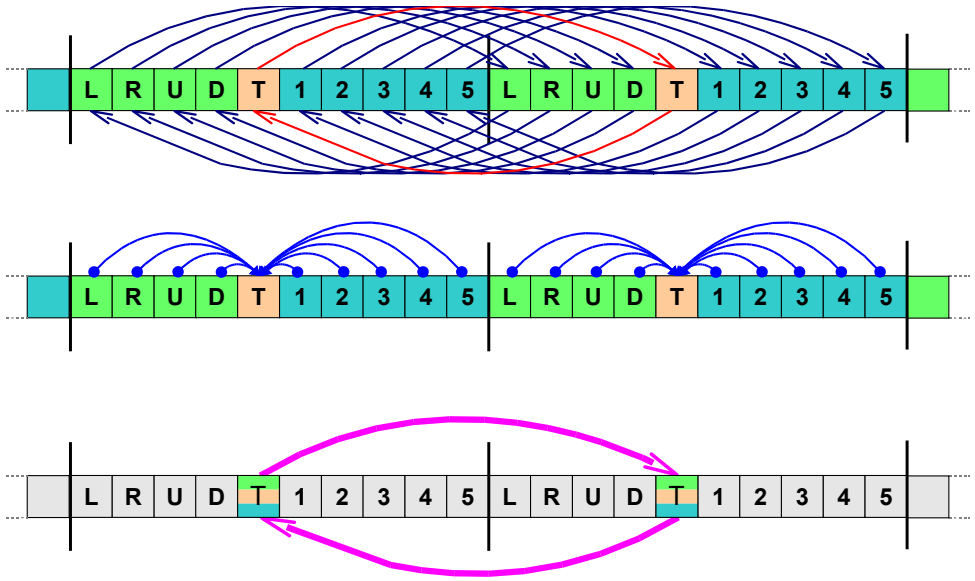


Figure 5.25: Extended flow estimation. Top: Basic flow estimation. Middle: All frames are aligned to the tracking frame. Bottom: Flow is recomputed using enriched tracking frame.

Basic flow estimation Firstly, all flows between subsequent frames of the same type are computed in both directions using a global smoothness value of $\alpha_{Gl} = 0.02$. Formally spoken, for a given frame I_i we compute the flow fields $F_{i\to i+10}$, $F_{i+10\to i}$, $F_{i-10\to i}$, and $F_{i\to i-10}$, where $F_{i\to j}$ denotes the flow field from frame i to frame j.

Tracking frame alignment Consequently, all frames are warped to the tracking frame. Let I_i be the i-th frame in the sequence, and w.l.o.g. $i \mod 10 \geq 5$. I_i is aligned to the tracking frame $i_T = i + 10 - (i \mod 10)$ using the flow fields $F_{i\to i+10}$ and $F_{i+10\to i}$. In order to warp the frame i to the i_T, we apply the interpolation factor $t = \frac{(10-(i \mod 10)) \mod 10}{10}$ to the bidirectional splatting algorithm (section 5.2.3.2). Since the displacements in the sequence are rather large, bidirectional splatting produces perspicuously better intermediate images than the sampling approach (section 5.2.3.3).

Background matte If a clean-plate C is available, the computation of a matte from a given reference frame I is rather simple in theory: the color differences between the pixels of C and I are computed, and all differences which exceed a predefined magnitude correspond to foreground pixels. In practice we face several issues: the first is sensor noise, which slightly changes the pixel value in each frame. Secondly, the subject reflects light to the background, which makes a simple pixel comparison inappropriate. Finally, we have a heuristic problem according to the feet: the treadmill is slightly pushed down when a subject stands on it. Moreover, its belt is permanently moving which makes it impossible to detect a matte by thresholding color differences in these areas. Our matting algorithm is illustrated in figure 5.26 and described in the following. White pixels in the matte correspond to the foreground, black pixels to the background.

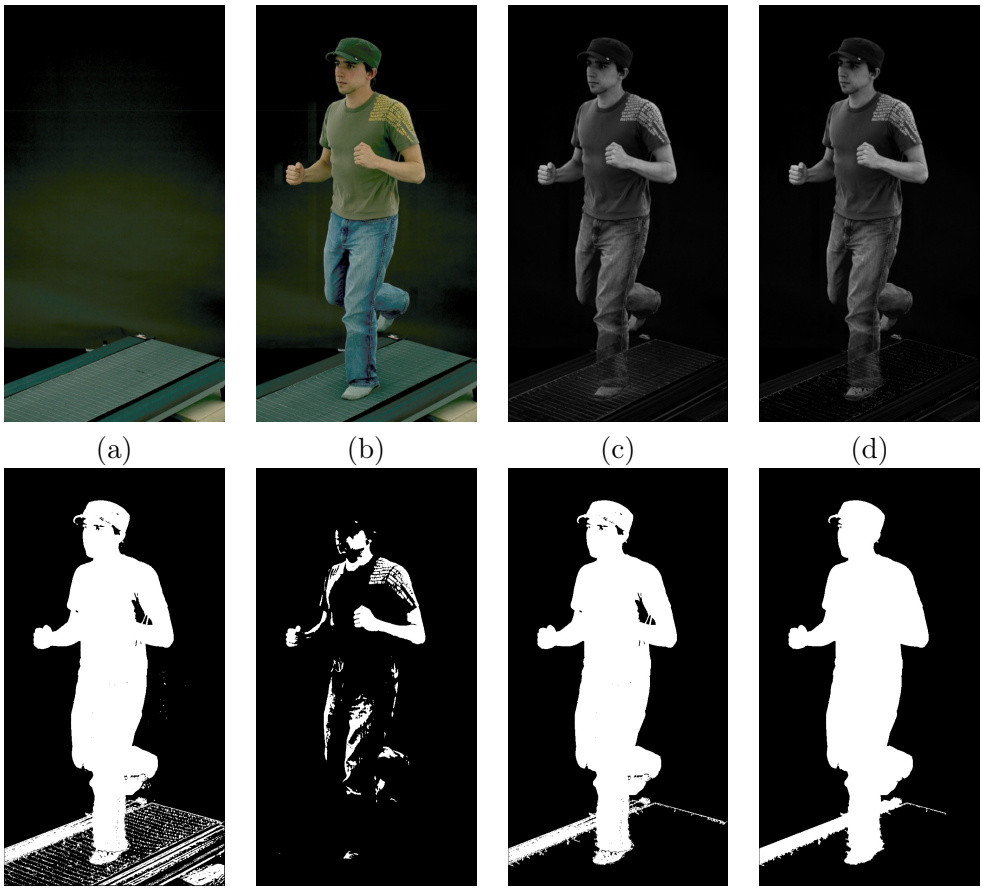


Figure 5.26: Our matting algorithm, (a) clean-plate and (b) reference image are given. (c) Confidence map based on 3×3 window comparison. (d) Confidence map after removing pixels in hue of treadmill. (e) Low threshold map. (f) High threshold map. (g) Matte after hysteresis thresholding. (h) Final result after removing white spots and filling gaps.

1. **Confidence map** At first we compute a *confidence map*, which represents the magnitude of the color differences between the clean-plate and the reference frame. We use the L^2 distance between the corresponding RGB color vectors as error metric. In order to reduce the impact of noise, we compare the average error on a 3×3 window rather than one single pixel. You find the confidence map in figure 5.26c.

2. **Treadmill belt** We furthermore remove large parts of the belt by a simple color analysis: we map the RGB color values to the HSV-format (hue, saturation and value) and label all pixels as part of the background with a hue value between 147.6° and 160.2°. The confidence of these values is set to 0 (see figure 5.26d).

3. **Hysteresis threshold** We apply a hysteresis thresholding (see section 4.2.2.1) approach to the confidence image in order to remove false positives due to noise. We use a low threshold of 0.0005 and a high threshold of 0.1. Figures 5.26e-g exhibit the result of the hysteresis thresholding.

4. **Remove small spots** The foreground should consist of one large single area, the subject itself. Therefore, we remove all remaining four-connected *white* spots containing less than 10^5 pixels.

5. **Fill gaps** Finally, we fill small gaps inside the subject, which we consider as false rejections. We apply the last step in the inverse way: we remove all four-connected *black* spots which cover less than than 10^4 pixels. The final matte is shown in figure 5.26h.

Even though the resulting mattes are not perfect they are still precise enough to avoid most of the distraction from the noisy background.

Depth discontinuity map In the following the aligned Raskar frames are used to create a discontinuity map. We investigated that the original Raskar approach does not work really well for our purpose. On the one hand, it detects all significant wrinkles in the clothing. On the other hand, it misses the important discontinuities between the hand and the body (see figure 5.27a). The reason is that the shadows are slightly blurred due to the bilateral splatting in the alignment process. The Sobel filter does not recognize the resulting smooth intensity steps as discontinuities. Furthermore, the original approach detects a lot of wrong discontinuities on the script pattern on the left shoulder.
We modify the algorithm by Raskar as follows: we use two rather small thresholds for the depth discontinuity detection ($\varepsilon_{\text{Low}} = 0.14$, $\varepsilon_{\text{High}} = 0.22$) in order to obtain the discontinuities at smoothed limb boundaries. Moreover, we *reject* all depth discontinuities at shadows which are not dark enough. Thus, only depth discontinuities at (nearly) black shadows are detected. This excludes shadows caused by wrinkles, which are not of interest, but includes crucial discontinuities between rigidly moving parts.

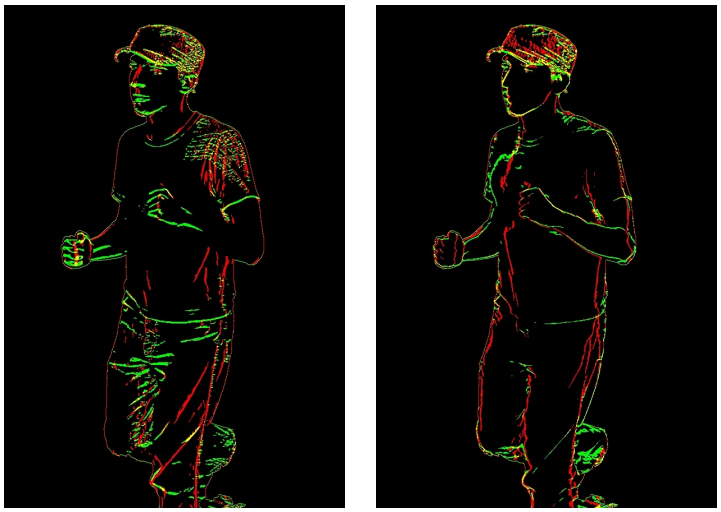


Figure 5.27: Modification of Raskar algorithm. (a) Depth discontinuities detected by original approach. (b) Discontinuity map after application of our modifications.

We post-process the discontinuity map by incorporating the matte: all discontinuities on background pixels are removed and discontinuity labels are added alongside the silhouette. The result of our modified approach is shown in figure 5.27b.

Normal map We furthermore use the aligned five photometric stereo images to obtain a normal map. We retrieve the approximated lighting directions for each frame by taking the center of gravity (COG) of the positions of the active light sources. The direction equals the difference of the COG with the center of the stage.

Enriched flow estimation Next, we calculate the optical flow fields between the tracking frames, however, this time enriched with the normal maps and the depth discontinuities. Again, we map the normal maps to offset gray in order to weight all input channels uniformly ($\gamma_i = 1$). According to the discontinuity maps, we adopt the global smoothness weight as the general one ($\alpha_G = \alpha_{Gl} = 0.02$) and determine a boundary smoothness of $\alpha_B = 0.005$. We refrain from using a spatio-temporal regularizer since we investigated that a simple spatial smoothness constraint yield better results.

Finally, we compare the quality of the basic flow fields between the tracking frames with the enriched flow field. The results are presented in the following section.

5.4.2.6 Results

Figure 5.28 shows the aligned frames between index 1000 and 1060 of our shoot sequence.

Our algorithm detects most of the important depth discontinuities, especially the ones between the rigidly moving limbs. However, some minor problems oc-

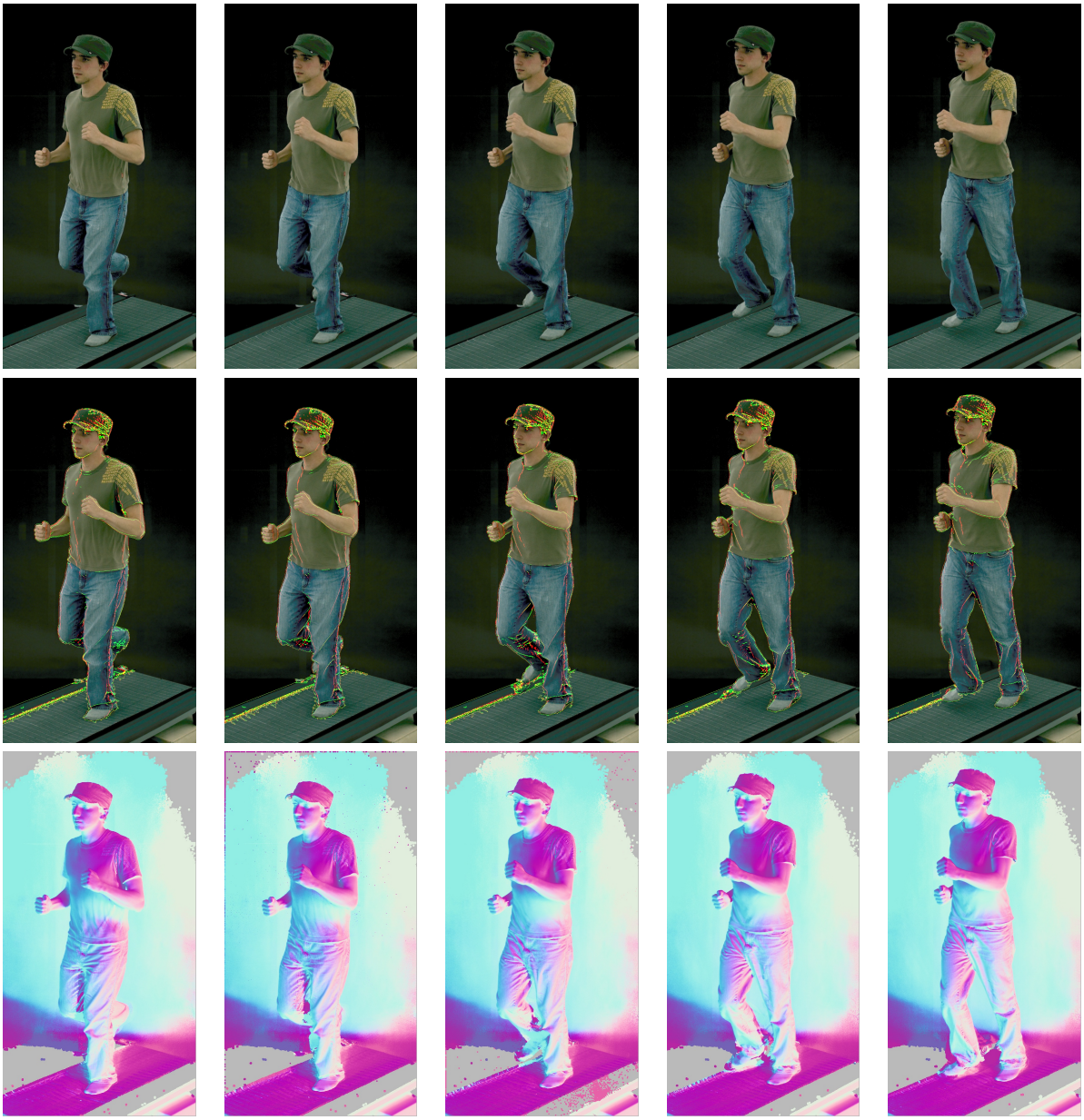


Figure 5.28: Aligned frames 1000 to 1060 of Light Stage shoot. Top: Standard views. Middle: Depth discontinuities lain over standard views. Bottom: Normal maps in offset-gray mode.

cur as well: The discontinuities which separated the arms from the body seem to be shifted (see figure 5.29a). The reason is that the strips of light sources for the Raskar approach are not near enough to the camera – in some cases the arm simply casts a shadow on itself instead of on the upper body part. Moreover, in some frames (e.g. frame 1044) the shadow of the arm is not directly attached and a non-shadowed gap between the limb and the chest appears. The depth discontinuity detection also suffers slightly from sensor noise.

The normal maps appear rather correct in most parts of the subject as illustrated in figure 5.29b-d. However, as usual for photometric stereo the results are biased by self-shadowing effects and noise: in all places where the subject casts a shadow on itself the resulting normals are wrong. We computed the gradient of the normals maps (as in section 5.3.2) in order to obtain better flow fields. As explained in section 4.1.1, better and less noisy normals could be retrieved by using additional samples, i.e. more lighting conditions. Unfortunately, additional lighting conditions imply more motion within the particular photometric stereo frames, which would lead to even more incorrect results. The imprint on the shirt slightly appears in the normal maps, probably due to misalignment of the basic flow.

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

Figure 5.29: Close-up on upper part of the body in frame 1024. (a) Depth discontinuity map lain on standard view. (b)-(d) Absolute x-, y- and z-component of normal maps.

Figure 5.31 exhibits all flow fields and the influence of our additional information channels. First of all, the optical flow algorithm by T. Brox produces already rather good results and allows discontinuous flow vectors which are blurred at the motion discontinuity boundaries though (top row). Major errors only appear at the right leg of the subject, where the algorithm incorrectly assumes a discontinuous flow field.

Even though the normal maps showed up some significant issues, they successfully support the flow estimation. Apparently, the basic optical flow algorithm

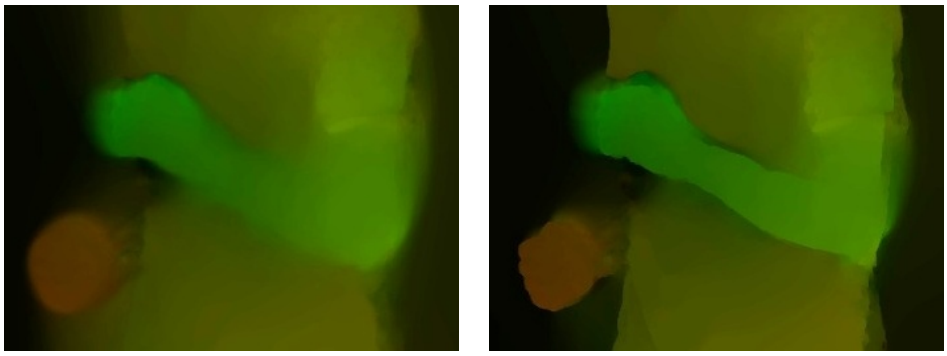


Figure 5.30: Close-up on flow 1034 → 1044 using color and normal gradient channels. Left: Standard flow. Right: Flow using depth discontinuity map.

by T. Brox is rather insensitive to noise. The flow fields based on the standard views are more accurate in most cases, as we investigated by evaluation of the bidirectional splatting sequence. But in some frames the flow estimation based on the standard views introduce errors, which the computation on the normal gradient maps do not produce. See for example the flow field between 1034 and 1044 (figure 5.31). The flows on the standard views produce an invalid discontinuity on the left leg (red spot), which does not exist in the normal gradient flows. The combined flow estimation based on standard views *and* normal maps mostly produces the best flow fields, which contain more rigid areas.

The incorporation of our binary depth discontinuity maps sharpens the flow fields at the detected boundaries and the flow field becomes piecewise smooth (see figure 5.30). In some frames our extension corrects errors which were caused by oversmoothing effects. The flow on the left arm is rigid due to the discontinuity map but it is important to point out that wrong discontinuities might worsen the result. For instance, since the discontinuity borders on the arm are slightly shifted in some frames, the flow above the borders is erroneous. It is crucial that the depth discontinuity maps are as accurate as possible. Due to the added external discontinuities, the flow is not smoothed over the silhouette of the subject. Figure 5.32 shows the flow fields using depth discontinuity maps.

5.5 Conclusions

This chapter discussed synthetic and natural test cases, primarily focused on the motion of human subjects. In all cases the basic optical flow algorithm turned out to be rather accurate and motion discontinuity preserving but it still smoothes over the discontinuity borders. The incorporation of depth discontinuities sharpens the boundaries between rigidly moving objects and yields a piecewise smooth flow field. The knowledge about these discontinuities avoid independently moving objects to distract the flow estimation of each other. It

has also been shown that finding depth discontinuities is not an easy task and requires a good setup and careful parameter tuning.

The additional geometry input channel, the normal maps, can highly improve the flow computation. It is especially helpful in the static case (stereo) where it helps to find the correct displacement vectors where flow estimation based on (repetitive) texture information fails. In the dynamic cases we may add the *gradient* of the normal maps as auxiliary input channel but we did not improve our results on synthetic images with this additional information. However, they supported our shoot sequence.

The evaluation of the shoot sequence showed that the combination of depth discontinuities and a normal map can be helpful for improving the flow estimation in realistic, dynamic cases, even though there is potential to improve both input channels if time-multiplexed illumination is used. This is explained in the following chapter.

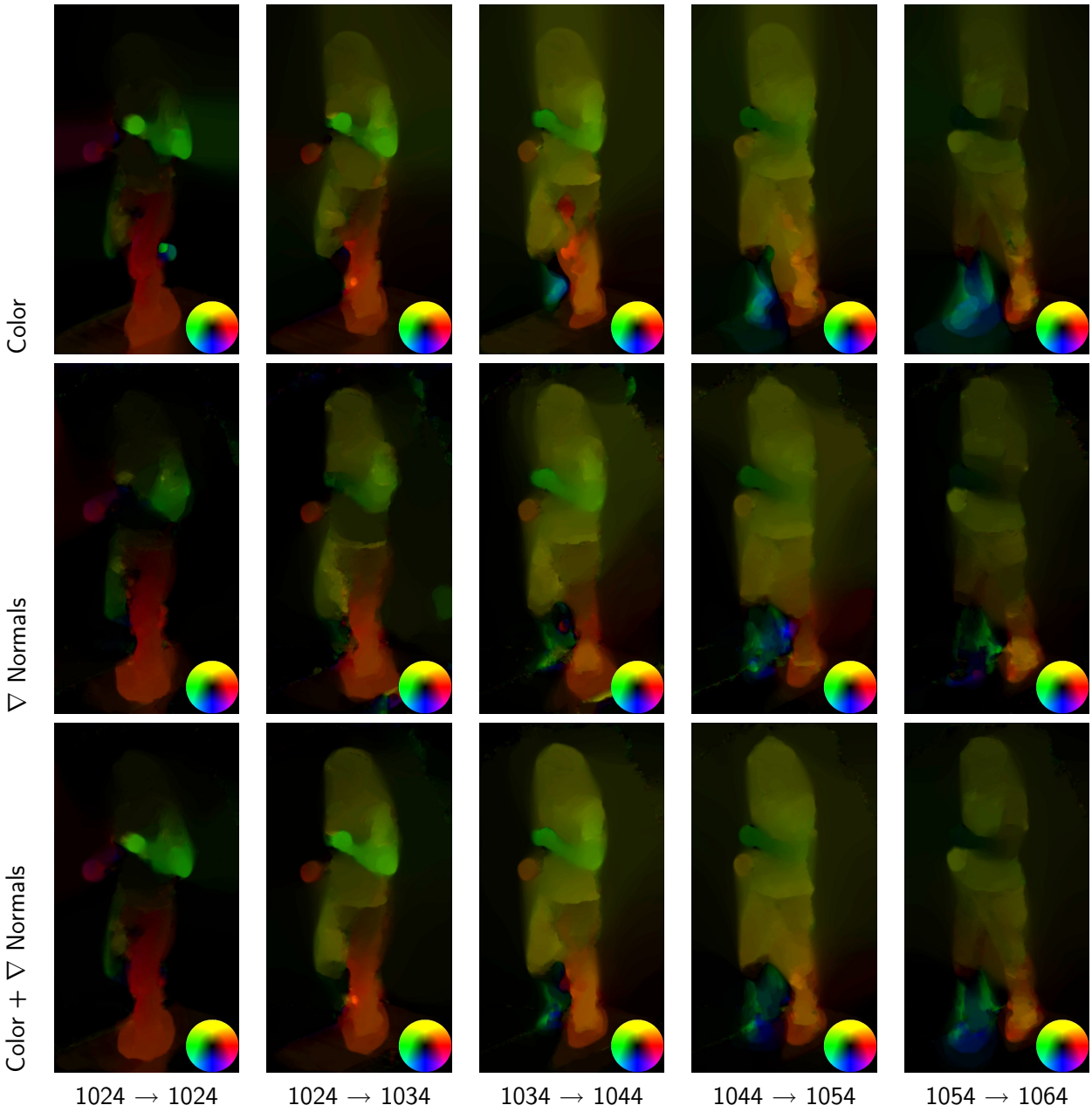


Figure 5.31: Flow fields between subsequent tracking frames from 1014 to 1064 *without* depth discontinuity maps. Top: Based on standard views. Middle: Based on normal gradients. Bottom: Based on standard views and normal gradients.

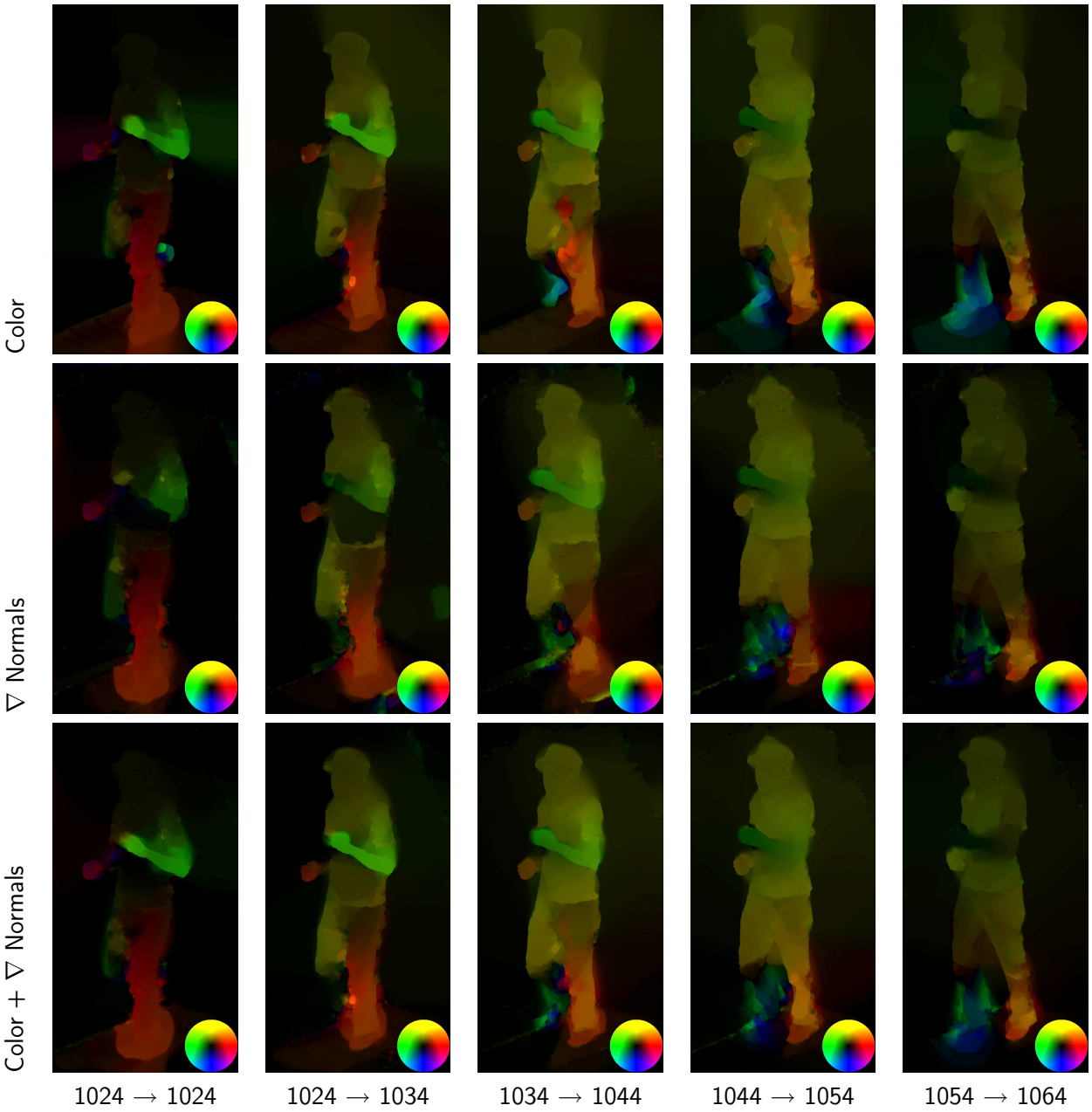


Figure 5.32: Flow fields between subsequent tracking frames from 1014 to 1064 *with* depth discontinuity maps. Top: Based on standard views. Middle: Based on normal gradients. Bottom: Based on standard views and normal gradients.

Chapter 6

Summary and Future Work

The tests of our algorithm on synthetic and realistic images show that the surface normals support the estimation of optical flow especially in the static case (stereo matching). Furthermore, they potentially yield a more accurate optical flow when the subject is moving. Depth discontinuities help in all cases, as they yield a piecewise smooth flow field: discontinuities between rigidly moving objects were preserved by sharp boundaries.

The algorithm also produces promising results in the dynamic context of Image-Based Relighting. Nonetheless, there is still potential to retrieve more precise flow fields by tuning the lighting conditions. In future work the depth discontinuity maps could be highly improved by the usage of bright flash lights, which are supposed to be as close to the camera as possible, rather than groups of light sources as in the case of the Light Stage 6 apparatus. The current setup using groups of lights induced two drawbacks: on the one hand, the light sources are further away from the camera and cause the objects in a scene to cast shadows on theirselves (shifted depth discontinuities), and on the other hand, the shadows appear more blurred because multiple lights in a spatial extension are used. Blurred shadows make it harder to detect discontinuities because their intensity gradient is smaller.

Accurate flow fields between the tracking frames yield a better frame alignment and thereby a better visual composition of the light conditions. Contrariwise, the *auxiliary frames* can usually not be employed for relighting, which means that we are *loosing* potential lighting conditions. This might impair the quality of the final linear combination. The producer must find a compromise between the number of lighting conditions, which he sacrifices for the flow improvement, and the number of frames, which represent lighting vectors for the relighting process.

Rather than lighting conditions for photometric stereo normals, the gradient patterns as explained in section 4.1.2 could be incorporated into the lighting basis. We would take advantage from this patterns in two ways: firstly, the normals would be more robust against noise and self-shadowing and secondly,

the Spherical Harmonic patterns could be used for relighting purposes. In one potential scenario the whole lighting basis could consist of the first n bands of Spherical Harmonic patterns. The first two bands would be used for normal estimation, but *all* frames could be used for the relighting process.

If the scene is static and only the viewpoint changes, the illumination under additive lighting conditions is not critical and technically simple: the Raskar shadow maps can be created with a simple spotlight as well as the photometric stereo normal maps, whose number of samples per pixel is not limited as in dynamic scenes. We showed that the incorporation of these input channels only slightly increases the runtime.

In a dynamic scene we need a special lighting apparatus and the number of lighting conditions in a lighting basis is limited. Even if the normal maps are retrieved by only four gradient patterns, these additional frames should only be added to the tracking frames if a precise flow field is necessary. The approach by Brox et al. [BBPW04] already allows the creation of intermediate frames which are "visually plausible" in most cases, although the flow fields are not completely correct. However, if an accurate flow is required, our algorithm offers a powerful tool to obtain this goal.

Appendix A

Successive Overrelaxation

The **Gauss-Seidl method** iteratively solves the n equations of linear systems in the form $Ax = b$, where A is a strictly diagonally dominant or symmetric positive definite matrix. The solution for x is found component by component, where previous solutions are incorporated as soon as they are available. The iterative formula for the k-th component in the solution vector x depending on the $k-1$ previously computed values reads:

$$x_i^k = \frac{b_i - \sum_{j<i} a_{ij} x_j^k - \sum_{j>i} a_{ij} x_j^{k-1}}{a_{ii}}$$

The technique is repeated for the entire vector x several times until convergence. Since the solution is found in a serial manner, the *order* in which the components are calculated affects the progression of the solution.

The **Successive Overrelaxation (SOR) method** is an extension of this technique. It linearly extrapolates the solution of the Gauss-Seidl method. The new k-th component of x then reads:

$$x_i^k = \omega \cdot \bar{x}_i^k + (1 - \omega) \cdot x_i^{k-1}$$

where $\bar{x}_i^k$ denotes the solution of the Gauss-Seidl method and $\omega \in (0, 2)$ determines the extrapolation factor. $\omega = 1$ reduces the equation to the simple Gauss-Seidl method. It has been shown by Kahan [Kah58] that the technique diverges for $\omega \notin (0, 2)$. The SOR method usually provides a faster convergence, but there is no heuristic to determine the optimal extrapolation factor.

Appendix B

Hysteresis Thresholding

This appendix contains the pseudocode for the hysteresis thresholding as introduced in section 4.2.2.1. Our function expects a confidence image as well as a low and a high threshold. Consequently, the algorithm returns the produced binary map.

```
FUNCTION HysteresisThreshold(imgConfidence : IMAGE of REAL,
                             fLowThreshold : REAL, fHighThreshold : REAL)

VAR cx, cy          : INTEGER;
    imgResult       : IMAGE of BOOL;
    listCoordinates : STACK;

BEGIN

    (cx,cy) = GetImageSize(imgConfidence, cx, cy);    // Retrieve image extensions

    // Create low and high threshold map

    imgLow  = CreateBooleanImage(cx, cy);
    imgHigh = CreateBooleanImage(cx, cy);

    forall (x,y) with 0<=x<cx and 0<=y<cy
    {
        imgHigh(x,y) = (imgConfidence(x,y) > fHighThreshold) ? true : false;
        imgLow(x,y)  = (imgConfidence(x,y) > fLowThreshold)  ? true : false;
    }

    imgResult = imgHigh;          // Result contains high threshold map

    empty( listCoordinates );     // Empty stack, which contains coordinates
```

```
    // Process the entire image
    forall (x,y) with 0<=x<cx and 0<=y<cy
    {
        // If a pixel is part of the high map add all pixels,
        // which can be connected via the low threshold map
        if( imgHigh(x,y) == true )
        {
            // Put the pixel on the stack and mark it as processed
            push( listCoordinates, (x,y) );
            imgLow(x,y) = false;

            // Process coordinate stack until it is emtpy
            while( !isEmpty(listCoordinates) )
            {
                // Pop top coordinate from stack
                (coordx,coordy) = pop( listCoordinates );

                imgResult(coordx,coordy) = true;    // Store this pixel in result map
                imgHigh(coordx,coordy)   = false;   // Do not process it again

                // Add the pixels in the neighborhood to the process list
                // unless they are already processed
                for(dy = -1; dy <= 1; dy++)
                {
                    for(dx = -1; dx <= 1; dx++)
                    {
                        if( coordx+dx >= 0 and coordx+dx < cx and
                            coordy+dy >= 0 and coordy+dy < cy and
                            imgLow(coordx+dx, coordy+dy) = true )
                        {
                            // Add this pixel to the process list and
                            // avoid a repetitive processing
                            push( listCoordinates, (coordx+dx, coordy+dy) );
                            imgLow(coordx+dx, coordy+dy) = false;
                        }
                    }
                }
            }
        }
    }

    return imgResult;
END
```

Appendix C

Light Stage 6

In the last years the Institute of Creative Technologies developed several lighting apparatus which allowed to simulate arbitrary distant illumination [DHT+00, HWT+04, WGT+05, CEJ+06]. Since this thesis intends to investigate the potential improvement of optical flow on human performances, the so-called *Light Stage 6* is the most appropriate apparatus for our purpose.

The Light Stage 6 dome was created in order to capture full-body performances under arbitrary illumination. The design is illustrated in figure C.1. It is the top two-thirds of an 8th-frequency geodesic sphere with a diameter of eight meters. Thus, the equator of the dome is 1.5 meters above the ground. 901 uniformly distributed light sources cover the dome which contain six LEDs in an 18cm-diameter hexagon each. Additionally, the device contains 140 floor lights at the height of 85cm below the equator of the dome, which also consist of six LEDs and simulate a Lambertian reflecting ground plate. All lights are controlled by microcontroller boards, which synchronize the lights and the camera. The subject is placed on a virtual floor 65cm above the ground. The setup provides a turnable treadmill, which the subjects can run on. The cameras which capture the performance can basically be placed everywhere, i.e. from an arbitrary 3D position, either mounted on tripods or directly on the stage. For detailed information we refer to the original paper [CEJ+06].

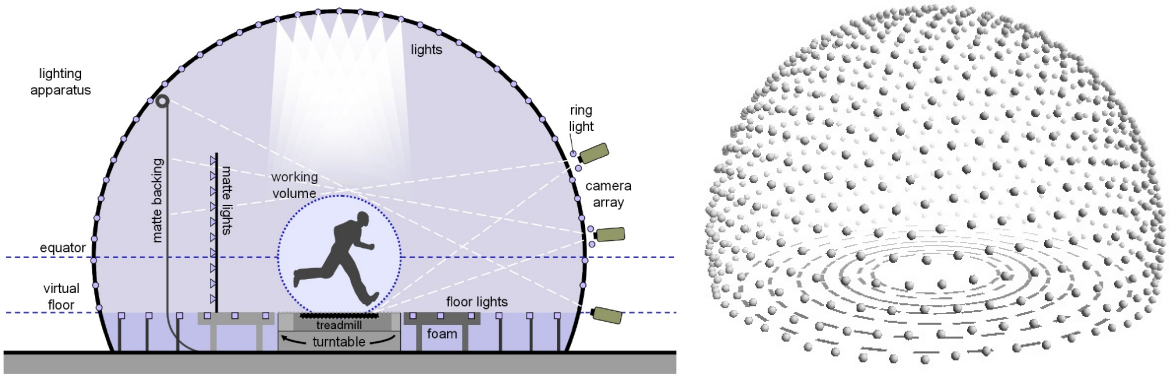


Figure C.1: Light Stage 6. Left: Technical drawing as shown in [CEJ+06]. Right: 3D model.

Bibliography

[AK06] T. Amiaz and N. Kiryati. Piecewise-Smooth Dense Optical Flow via Level Sets. *Int. J. Comput. Vision*, 68(2):111–124, 2006.

[Ana89] P. Anandan. A Computational Framework and an Algorithm for the Measurement of Visual Motion. *IJCV*, 2(3):283–310, January 1989.

[Arf85] G. Arfken. *Mathematical Methods for Physicists*. Academic Press, San Diego, CA, 3^{rd} edition, 1985.

[BA87] Peter J. Burt and Edward H. Adelson. The Laplacian Pyramid as a Compact Image Code. pages 671–679, 1987.

[BA93] M.J. Black and P. Anandan. A Framework for the Robust Estimation of Optical Flow. In *ICCV93*, pages 231–236, 1993.

[BBPW04] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High Accuracy Optical Flow Estimation Based on a Theory for Warping. In T. Pajdla and J. Matas, editors, *European Conference on Computer Vision (ECCV)*, volume 3024 of *LNCS*, pages 25–36, Prague, Czech Republic, May 2004. Springer.

[BBW06] T. Brox, A. Bruhn, and J. Weickert. Variational Motion Segmentation with Level Sets. In A. Leonardis, H. Bischof, and A. Pinz, editors, *European Conference on Computer Vision (ECCV)*, volume 3951 of *LNCS*, pages 471–483, Graz, Austria, May 2006. Springer.

[Bla92] Michael Julian Black. *Robust Incremental Optical Flow*. PhD thesis, New Haven, CT, USA, 1992.

[Bro05] T. Brox. *From Pixels to Regions: Partial Differential Equations in Image Analysis*. PhD thesis, Faculty of Mathematics and Computer Science, Saarland University, Germany, April 2005.

[BW05] Andres Bruhn and Joachim Weickert. Towards Ultimate Motion Estimation: Combining Highest Accuracy with Real-Time Performance. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, pages 749–755, Washington, DC, USA, 2005. IEEE Computer Society.

[BZ87] Andrew Blake and Andrew Zisserman. *Visual Reconstruction*. MIT Press, Cambridge, MA, USA, 1987.

[CEJ+06] Charles-Felix Chabert, Per Einarsson, Andrew Jones, Bruce Lamond, Wan-Chun Ma, Sebastian Sylwan, Tim Hawkins, and Paul Debevec. Relighting Human Locomotion with Flowed Reflectance Fields. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 76, New York, NY, USA, 2006. ACM Press.

[DHT+00] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. Acquiring the Reflectance Field of a Human Face. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 145–156. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[FJ90] David J. Fleet and A. D. Jepson. Computation of Component Image Velocity from Local Phase Information. *Int. J. Comput. Vision*, 5(1):77–104, 1990.

[FRC+05] Rogerio Feris, Ramesh Raskar, Longbin Chen, Kar-Han Tan, and Matthew Turk. Discontinuity Preserving Stereo with Small Baseline Multi-Flash Illumination. In *ICCV*, pages 412–419, 2005.

[GG88] Stuart German and Donald German. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. pages 611–634, 1988.

[Gib79] James J. Gibson. *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, 1979.

[GMN+98] Ben Galvin, Brendan McCane, Kevin Novins, David Mason, and Steven Mills. Recovering Motion Fields: An Evaluation of Eight Optical Flow Algorithms. In *BMVC*, 1998.

[Hee88] D.J. Heeger. Optical Flow Using Spatiotemporal Filters. *International Journal for Computer Vision*, 1:279–302, 1988.

[Hop98] Hugues Hoppe. Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 35–42, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[Hor86] B. K. P. Horn. *Robot Vision*. The MIT Press, Cambridge, MA, USA, 1986.

[HS81] B.K.P. Horn and B.G. Schunck. Determining Optical Flow. 17(1-3):185–203, August 1981.

[HS94] Berthold K. P. Horn and B. G. Schunck. Determining Optical Flow: a Retrospective. pages 81–87, 1994.

[HWT+04] Tim Hawkins, Andreas Wenger, Chris Tchou, Andrew Gardner, Fredrik Goransson, and Paul Debevec. Animatable Facial Reflectance Fields. In A. Keller and H. W. Jensen, editors, *2004 Eurographics Symposium on Rendering*, pages 309–319, 2004.

[Kah58] William Morton Kahan. *Gauss-Seidel methods of solving large systems of linear equations*. Ph.D. thesis, University of Toronto, Toronto, ON, Canada, 1958.

[LK81] B.D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *IJCAI81*, pages 674–679, 1981.

[MHP+07] Wan-Chun Ma, Tim Hawkins, Pieter Peers, Charles-Felix Chabert, Malte Weiss, and Paul Debevec. Rapid acquisition of specular and diffuse normal maps from polarized spherical gradient illumination. In *2007 Eurographics Symposium on Rendering (to appear)*, June 2007.

[Nag83a] H. H. Nagel. Constraints for the Estimation of Displacement Vector Fields From Image Sequences. In *Proc. of the 8th IJCAI*, pages 945–951, Karlsruhe, Germany, 1983.

[Nag83b] H.H. Nagel. Displacement Vectors Derived from Second-Order Intensity Variations in Image Sequences. *CVGIP*, 21(1):85–117, January 1983.

[OK92] Masatoshi Okutomi and Takeo Kanade. A Locally Adaptive Window for Signal Matching. *Int. J. Comput. Vision*, 7(2):143–162, 1992.

[PH04] P. H. A. R. R. Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (The Interactive 3d Technology Series)*. Morgan Kaufmann, August 2004.

[RH02] Ravi Ramamoorthi and Pat Hanrahan. Frequency Space Environment Map Rendering. *ACM Trans. Graph.*, 21(3):517–526, 2002.

[RTF+05] Ramesh Raskar, Kar-Han Tan, Rogerio Feris, Jingyi Yu, and Matthew Turk. Non-Photorealistic Camera: Depth Edge Detection and Stylized Rendering Using Multi-Flash Imaging. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 2, New York, NY, USA, 2005. ACM Press.

[SA92] Ajit Singh and Peter Allen. Image-Flow Computation: an Estimation-Theoretic Framework and a Unified Perspective. *CVGIP: Image Underst.*, 56(2):152–177, 1992.

[SBW05] Natalia Slesareva, Andrés Bruhn, and Joachim Weickert. Optic Flow Goes Stereo: A Variational Method for Estimating Discontinuity-Preserving Dense Disparity Maps. In *DAGM-Symposium*, pages 33–40, 2005.

[Sch89] B. G. Schunck. Image Flow Segmentation and Estimation by Constraint Line Clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(10):1010–1027, 1989.

[ST06] Peter Sand and Seth Teller. Particle Video: Long-Range Motion Estimation Using Point Trajectories. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2195–2202, Washington, DC, USA, 2006. IEEE Computer Society.

[UGVT89] S. Uras, F. Girosi, A. Verri, and V. Torre. A Computational Approach to Motion Perception. 60:79–87, 1989.

[WGT+05] Andreas Wenger, Andrew Gardner, Chris Tchou, Jonas Unger, Tim Hawkins, and Paul Debevec. Performance Relighting and Reflectance Transformation with Time-Multiplexed Illumination. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 756–764, New York, NY, USA, 2005. ACM Press.

[Woo89] Robert J. Woodham. Photometric Method for Determining Surface Orientation from Multiple Images. pages 513–531, 1989.

[Woo92] Robert J. Woodham. Multiple Light Source Optical Flow. pages 417–421, 1992.

[XCS+06] J. Xiao, H. Cheng, H.S. Sawhney, C. Rao, and M. Isnardi. Bilateral Filtering-Based Optical Flow Estimation with Occlusion Detection. In *ECCV06*, pages I: 211–224, 2006.

List of Figures