Master's Thesis – Winter Semester 2009/2010

Tracking and Structure from Motion

Author:
Andreas Weishaupt

Supervisor:
Prof. Pierre Vandergheynst

Assistants:
Luigi Bagnato
Emmanuel D'Angelo

Signal Processing Laboratory – Section of Electrical Engineering – School of Engineering
Swiss Federal Institute of Technology. Lausanne, Switzerland.

January 14, 2010

“Wagen wir, die Dinge zu sehen, wie sie sind.”

Acknowledgement

I would like to thank everybody who has helped me making this work possible. Special thanks go to Prof. Pierre Vandergheynst for offering me a place in his laboratory to carry out my Master's project and for his supervision. My thanks go to the very friendly and helpful team and in particular to Luigi Bagnato and Emmanuel D'Angelo without whom this work would have been impossible. I would like to thank my family, Corazon, Jakob and Sarah, for their endless love and support during my studies. Finally, I would like to thank all my friends for their love, for their inspiration and respect for what I am doing.

Contents

Chapter 1

Introduction

Knowing the geometric structure of the world is a necessity for successful navigation. While the human visual system is capable of extracting meaningful geometric information from light efficiently and in a very transparent way, the research on computer vision has been attempting for decades to conceive a system that provides us with the same information in an equally transparent way. Such a system would be of invaluable benefit for a big variety of fields: autonomous navigation, robotics, architecture, film industry, computer game industry, room acoustics etc.

Extracting geometric structure means essentially to estimate distances from the observer to objects in the scene. Once the distances are known, the three-dimensional structure is known from the point of view of the observer. Traditionnally there have been two approaches to estimate distances to nearby objects: computer stereo vision and structure from motion. Computer stereo vision imitates the eyes of the human visiual system by using two fixed cameras pointing at the same scene. In structure from motion a camera moving in space is employed. Both methods provide us with a parallax that allows estimating the distances to the captured objects. In this Master's thesis, only structure from motion is treated: it is the more interesting subject with respect to applicability in many related projects. Although computer stereo vision has been studied extensively for a wide variety of applications it doesn't seem being used in practice in everyday life and thus it is not of interest in this work.

There are more recent methods for inference of the three-dimensional structure which are not based on image processing directly. These methods don't rely on cameras but rather on some sophisticated devices based on laser scanning or infra-red emmision and sensing. Those methods are not of interest in this work neither.

1.1 Traditional structure from motion

In traditional structure from motion, there are two fundamental problems:

1. Corresponding pairs of points must be found for two successive images. This is also known as the **correspondence problem**.
2. Using **epipolar geometry** and knowing the camera's rigid movement between the acquisition of the two images, the distance of the camera to the true point can be calculated.

While the second point is rather trivial once the camera movement is determined, the first point is a difficult and time-consuming procedure and is thus the bottleneck in traditional structure from motion methods. Consequently, usual structure from motion has been limited to determination of the true geometry only for very few pairs of points. It is obvious that the so called sparse structure from motion does not correspond to a true reconstruction of the world's geometry. Extending the sparse structure such that it becomes dense or extracting the structure without solving the correspondence problem has extensively been studied with not very convincing results. This is why structure from motion has often been considered as being a "difficult" topic in signal processing and computer vision and has not been touched by many researchers.

1.2 Tracking

In order to obtain the structure from motion from a video sequence, the correspondence problem does not have to be solved for every pair of successive frames: points given as the solution of the correspondence problem (i.e. feature points) can be tracked through successive frames. This is clearly very efficient but opens a whole new box of problems: what is the best method for tracking? How can it be implemented in real-time? What can be done if tracking fails? Various approaches have been proposed and studied. Some of them have been implemented and tested in robot systems. But the problem remains the same: sparse 3D reconstruction is not meaningful for many applications.

1.3 Motion estimation

In computer stereo vision the camera setup geometry is known in advance and this knowledge can be included in epipolar geometry evaluation for depth calculation without any major difficulties. In structure from motion the camera motion parameters have to be known in order to solve the epipolar geometry problem. Usually, when mounted on robot systems, the motion parameters are obtained from external trajectory estimation systems composed by

inertial measurement units, GPS or other positioning system and a Kalman filter or some other error reduction techniques.

Advances in video processing, video coding and associated motion estimation in and from video sequence can be used to extract the camera motion from the video sequence it captures. This is known as **ego-motion estimation**.

1.4 Dense structure from motion

Recent advances in computer science and video processing have lead to new solutions for the dense structure from motion problem. The traditional correspondence approach can be omitted and replaced by optical flow estimation. Bagnato et al. showed in [2] that by posing a good optimization problem optical flow and depth estimation can be combined in one nice mathematical framework.

Additionnaly, Pock showed in [22] how to robustly solve the optical flow problem in real-time by making use of the enormous calculation power given by recent graphics card processors. Implementing computational concepts for processing on graphics card is known as **GPGPU: general purpose computation on graphics processing unit**. Being a recent topic it will have a big impact on applied mathematics and related domains, especially video processing, as it opens very new possibilities: what has been far from it so far can now be made ready for real-time user-computer interaction.

1.5 Dense SfM for planar images in real-time

In this Master's thesis it is shown how to extend the structure from motion framework proposed by Bagnato et al. to planar image sequences, i.e. for images obtained by regular planar image sensors. Those sensors are found in every digital camcorder, digital camera and handy-cams. Thus, extending structure from motion to planar images is an important step towards true application. Finally it is shown how to make use of GPGPU in order to implement the framework for real-time processing.

1.6 Organization of the work

The document will be structured as follows: In Chapter 2, advanced and robust image processing methods based on optimization, variational calculus and partial differential equations are presented briefly in order to cover the preliminary theory work made prior to this thesis and needed to understand it thoroughly. In Chapter 3, the real-time TV-L^1 optical flow by Pock will be presented in more detail since TV-L^1 regularized optimization is very similar in the structure from motion framework. An application of optical flow to image distortion

correction known as *rolling shutter correction* is shown (D'Angelo [8]). In a real application, rolling shutter correction may be implemented advantageously together with the structure from motion algorithm. In Chapter 4, the structure from motion framework and its adaptation to the planar imaging geometry will be derived. In Chapter 5, it is quickly presented how to possibly stabilize the structure from motion algorithm for general stochastic movement. And finally, in Chapter 6, GPGPU is presented and it is shown how to perform video processing such as TV-L^1 optical flow and structure from motion computation on the GPU.

Chapter 2

Variational Methods in Computer Vision

2.1 Inverse problems

In computer vision and image processing we often want to solve inverse problems: find the model m from measurement data d. Some examples are:

- Find the 3D view of the scene from multiple images.
- Find the true image from a blurred, noisy image.
- Find the movement of the camera from successive images.
- Find the correct segmentation of an image.
- Find the true contours of objects on an image.

Inverse problems are not restricted to the world of computer vision. E.g. in acoustics, locating sources in space from a (mixed) audio-track or determining the signal to apply for active noise cancellation would be some other examples of inverse problems.

When dealing with images, we intrinsically solve inverse problems because an image acquired by a camera is always a projection of the true world. The direct problems (in the sense of "not inverse") of the examples above are well known and they can often be solved using special software:

- From a 3D model of a scene, generate an image. (Ray-tracer, 3D modelling software)
- Given a noise-free image, generate a blurred and noisy one. (MATLAB, Photoshop)
- Given camera movement and a scene, generate a video. (Ray-tracer, video software)

Typically inverse problems are ill-posed: one or more of the three necessary conditions for well-posed problems—existance, uniqueness and stability of the solution—are violated. While existance is normally not of big concern, uniqueness is almost never guaranteed and stability is strongly violated: for most inverse problems there exists an infinity of solutions, and a small error in the measurement yields a big error in the solution.

Often we have prior knowledge of the solution such that we can restrict the search space of possible solutions. This process is called *regularization*. It can be shown that many regularized inverse problems can be recovered from Bayesian inference.

2.2 Bayesian inference

Let us demonstrate the last statement using the classical example of image denoising. The inverse problem can be formulated as follows: *Given an intensity image $f = f(x, y)$ which is obtained by perturbing the noise-free image $u = u(x, y)$ with additive noise following a gaussian distribution $N(0, \sigma)$, restore the original image u.* Verification of the ill-posedness of the problem is left to the reader.

Any image can be perceived as a set of independent random variables with given probability distribution. Therefore we can formulate the solution of the inverse problem in a probabilistic approach as selecting the image u which has the highest probability given an image f:

$$u^* = \arg\max_u \, p(u|f) \tag{2.1}$$

where $p(u|f)$ denotes the **a posteriori probability**: the belief in the correctness of u for given f. This optimization process is also known as maximum a posteriori estimation (MAP). $p(u|f)$ is given by Bayes' rule:

$$p(u|f) = \frac{p(f|u)\, p(u)}{p(f)} \tag{2.2}$$

where $p(u)$ is the a priori probability, i.e. the belief in correctness of u without knowing f. $p(f|u)$ is the conditional probability and is simple to evaluate. $p(f)$ is just a normalization factor and can be left out for optimization since it does not depend on u. This means that once we can formulate the prior $p(u)$ we are able to solve the optimization and thus the inverse problem in a probabilistic sense. Unfortunately, in general it is not easy to accurately model $p(u)$.

Let's face again the denoising problem! Statistics of natural images have been studied exhaustively and it has been found that the stastics of two adjacent pixels (i.e. in the sense of the gradient of an intensity image) can be well described by the generalized Laplace distribution:

$$f(x) = \frac{1}{Z} e^{-|\frac{x}{\sigma}|^{\alpha}} \tag{2.3}$$

Where x is the response of a derivative filter (i.e. $x = \|\nabla I\|$), σ is the variance of the distribution, Z a normalization factor and α is related to the shape of the distribution. For $\alpha = 0.55$ the distribution fits best the statistics of the studied natural images. Unfortunately, modeling $p(u)$ using $\alpha = 0.55$ would end in a non-convex functional which is very difficult to optimize. Therefore, imposing $\alpha = 1$ in the prior seems to be the best compromise between formulating a solvable (but not strictly convex) optimization problem and having the most accurate image model possible. The prior $p(u)$ can now be formulated using pixel-wise independant probability:

$$p(u) = \prod_{(x,y)\in D} \frac{1}{2} e^{-|\nabla u(x,y)|} \tag{2.4}$$

where (x, y) denote the coordinates of the pixel and the pixels lie in the domain $D = W \times H$ where W is the width and H the height of the image. The conditional probability $p(f|u)$ simply results from the model assumption of additive gaussian distributed noise with zero mean and variance σ:

$$p(f|u) = \prod_{(x,y)\in D} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(f(x,y)-u(x,y))^2}{\sigma^2}} \tag{2.5}$$

Using (2.4) and (2.5) in (2.2) and (2.1) results in the following maximum a posteriori optimization:

$$u^* = \arg\max_u \prod_{(x,y)\in D} \frac{1}{2\sqrt{2\pi}\sigma} e^{-\left(|\nabla u(x,y)| + \frac{(f(x,y)-u(x,y))^2}{\sigma^2}\right)} \tag{2.6}$$

2.3 Energy functional

Using the Gibb's formula from statistical mechanics, it can be shown that maximizing the a posteriori probability is equal to minimizing the following energy:

$$u^* = \arg\min_u E(u) \tag{2.7}$$

and

$$E(u) = \sum_{(x,y)\in D} |\nabla u(x,y)| + \lambda^2 \left(f(x,y) - u(x,y)\right)^2 \tag{2.8}$$

Extending the energy function from the discrete pixel domain to a continuous domain is achieved simply by substituting the sum by an integral which results in the following energy functional minimization problem:

$$u^* = \arg\min_u \int_\Omega |\nabla u(x,y)| + \frac{1}{2\lambda}(f(x,y) - u(x,y))^2 \, d\Omega \tag{2.9}$$

Where λ can be related to the noise variance by $\lambda = \frac{\sigma^2}{2}$. The first and second terms in (2.9) are usually referred to as the **regularization term** and the **data term** respectively. The above minimization problem is well known as **total variation (TV) image denoising** since the first term is the **TV-norm** of u:

$$TV(u) = \int_\Omega |\nabla u(x,y)| \; d\Omega \tag{2.10}$$

The model is also known as the unconstrained (convex) version of a denoising model that has been originally formulated by Rudin, Osher and Fatemi in 1980s and is therefor sometimes referred to as the ROF model. Since the L^1 norm of the gradient is minimized, edges in the image are well preserved: outliers in the derivative are not strictly disfavored (in contrast to denoising using the L^2 norm). The L^1 norm regularization imposes sparsity in the solution which is well adapted to images since images are sparse by nature.

The term energy is not uncommon in signal processing and there is the related notion of **power spectral density (PSD)** to solve problems in the frequency domain. For example, the optimal solution of the denoising problem for normal distributed noise has been solved by Wiener in the 1940s(!) and is well known as the Wiener filter in theory on adaptive filtering.

2.4 Variational calculus

Since in signal processing the energy of an image is proportional to the sum of all pixels, many of the inverse problems in computer vision can be transformed into an energy functional minimization. We need to know how to solve the functional minimization problem: *Given* $F(u) = \int_\Omega f(u, \nabla u, \ldots) d\,\Omega$, *find* $u^* = \arg\min_u F(u)$.

This can be achieved by finding u^* such that:

$$\left[\frac{\partial F}{\partial u}\right]_{u=u*} = 0 \qquad \text{and} \qquad \left[\frac{\partial^2 F}{\partial u^2}\right]_{u=u*} > 0. \tag{2.11}$$

Fortunately mathematics is always some decades, centuries or more in advance to engineering and differentiating functions with respect to other functions is well studied in variational calculus which is a generalization of the differential calculus. The given derivative in (2.11) is called the Gâteaux derivative or 1st variation. Its solution satisfies the following partial differential equation (PDE) called the **Euler-Lagrange equation**:

$$\begin{cases} \dfrac{\partial F}{\partial u} = \left(\dfrac{\partial}{\partial u} - \nabla \cdot \dfrac{\partial}{\partial(\nabla u)}\right) f(u, \nabla) = 0 \\ \\ \text{boundary conditions} \end{cases} \tag{2.12}$$

In image processing the boundary conditions are usually equal to $\nabla u = 0$. Finally u is obtained by solving the Euler-Lagrange equation. For simple functionals, a closed-form solution can be obtained whereas for more complicated functionals, the partial differential equation can be solved numerically, e.g. by using the following iterative scheme which corresponds to the gradient descent method:

$$u^{k+1} = u^k - \Delta t \cdot \frac{\partial F}{\partial u} \tag{2.13}$$

and $u^0 = f$. Δt is the optimization time step and is problem dependent: $\Delta t \leq c|\nabla u|$. There are the known disadvantages of the gradient descent method. Depending on the PDE, these problems might become the major drawbacks of the method. Nevertheless, gradient descent or some modified version is often used to solve inverse problems formulated as variational minimization problems. For example, TV denoising can be achieved by minimizing (2.9), which results in solving the following PDE:

$$\frac{1}{\lambda}(u - f) - \nabla \cdot \frac{\nabla u}{|\nabla u|} = 0 \tag{2.14}$$

Introducting in the iterative scheme and setting the initial condition to $u = f$ we obtain:

$$u^{k+1} = u^k + \Delta t \cdot \lambda \nabla \cdot \frac{\nabla u}{|\nabla u|} \tag{2.15}$$

Finally, if the time-step Δt becomes infinitely small, (2.15) can be reformulated as another PDE, known as **anisotropic diffusion**:

$$\frac{\partial u}{\partial t} = \lambda \nabla \cdot \frac{\nabla u}{|\nabla u|} \tag{2.16}$$

Imposing a smoothness instead of a sparseness constraint results in a PDE which is well known from thermodynamics as **heat diffusion**. This illustrates well the link to physics when dealing with energy minimization approaches in image processing. The smoothness constraint is given by the L^2 or square norm of the gradient in the regularization term. Together with the usual L^2 norm data term we obtain the so called Tikhonov denoising model:

$$u* = \arg\min_u \int_\Omega \frac{1}{2} |\nabla u|^2 + \frac{1}{2\lambda} (f - u)^2 \, d\Omega \tag{2.17}$$

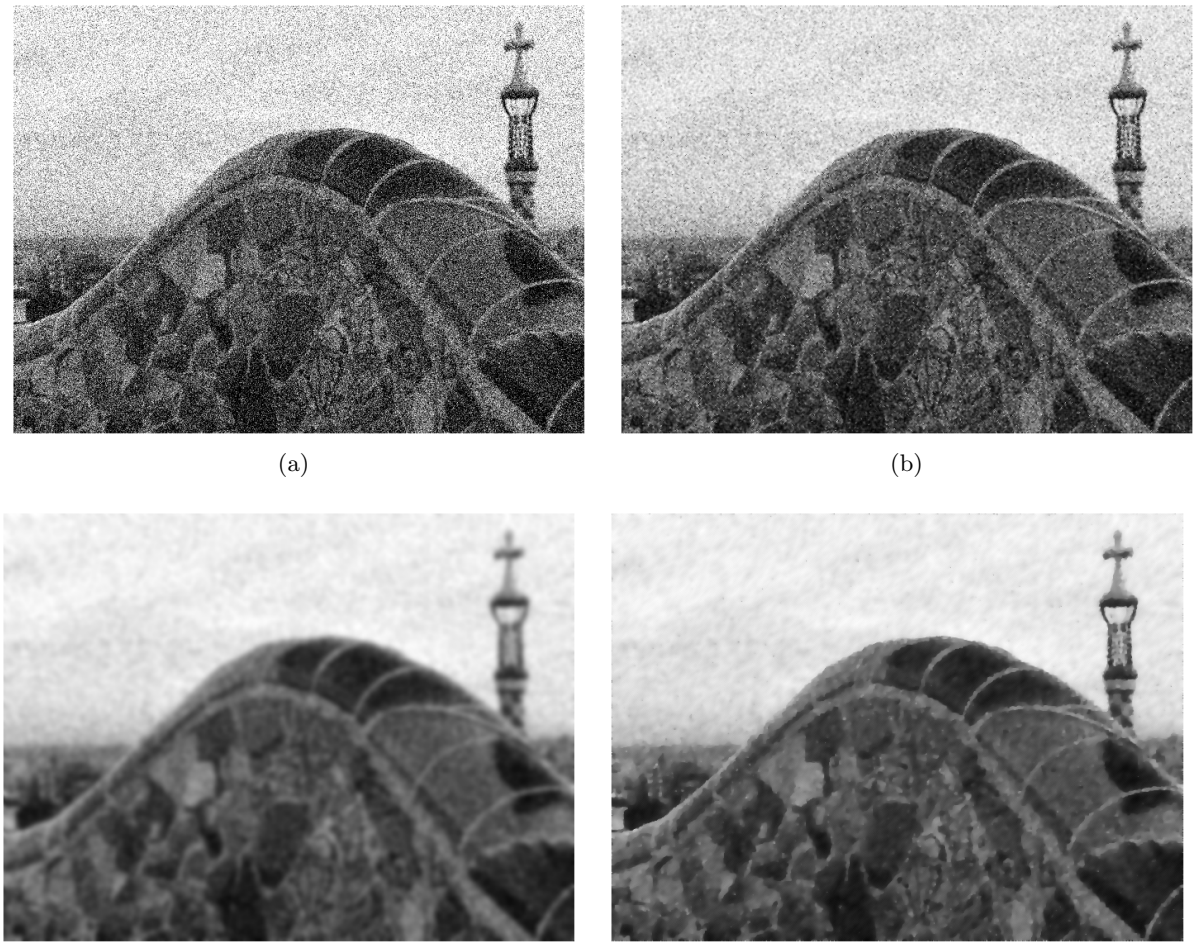
Its Euler-Lagrange equation is given by:

$$\frac{1}{\lambda} (f - u) - \nabla^2 u = 0 \tag{2.18}$$

Gradient descent with $\Delta t \to 0$ and $u = f$ leads to the heat diffusion equation:

$$\frac{\partial u}{\partial t} = \lambda \nabla^2 u \tag{2.19}$$

On Figure 2.1 we show a noisy image with 10% additive noise.



(a) (b)

(c) (d)

Figure 2.1: (a) an artistic picture of the Parc Güell perturbed with 10% additive white noise, (b) denoising using Wiener filter with a 3×3 neighborhood for mean and variance estimation (c) denoising using the Tikhonov model, 40 iterations with GD (d) denoising using the ROF model, 40 iterations with GD

Chapter 3

TV-L^1 Optical Flow

3.1 Introduction

With the preliminary work of Gibson in 1959 on "Motion parallax as a determinant of perceived depth" [10] where the terms "optical change of the visual field", "optical motion" and "optical velocity" first appeared, Horn and Schunck, in 1981 [14], define the "optical flow" as *"the apparent motion of brightness patterns observed when a camera is moving relative to the objects being imaged"*. Their work has had a big influence on advanced image processing and motion estimation methods. Nowadays, the optical flow is used as a basic building block in many methods and applications in image and video processing, computer and machine vision and in related fields such as biomedical imaging, imaging for nagivation etc. For example, it is used in many video compression algorithms as a basic motion estimator. In biomedical imaging, it serves e.g. as building block for automatic computer tomograph image segmentation or for highly nonlinear image restoration for MRI images when the patient was moving during acquisition.

In order to illustrate what optical flow is we will use the fly's eye: In recent years, much attention has been paid to the research of the fly and especially the fly's eye and navigation system in order to obtain good inspiration for autonomous navigation systems for (air) vehicles [16]. Since the fly has omnidirectional view, any kind of ego-motion, fixed and moving obstacle has a *unique optical flow pattern* as illustrated in Figure 3.1 (reprinted from [16]; refer to Chapter 4 for a more complete overview of flow patterns). It has been found out, that the fly's brain is capable of using the percieved optical flow as a primary source of information for successful navigation in space. For a final illustration, an example of optical flow for two frames of motion picture (video) is shown in 3.2.

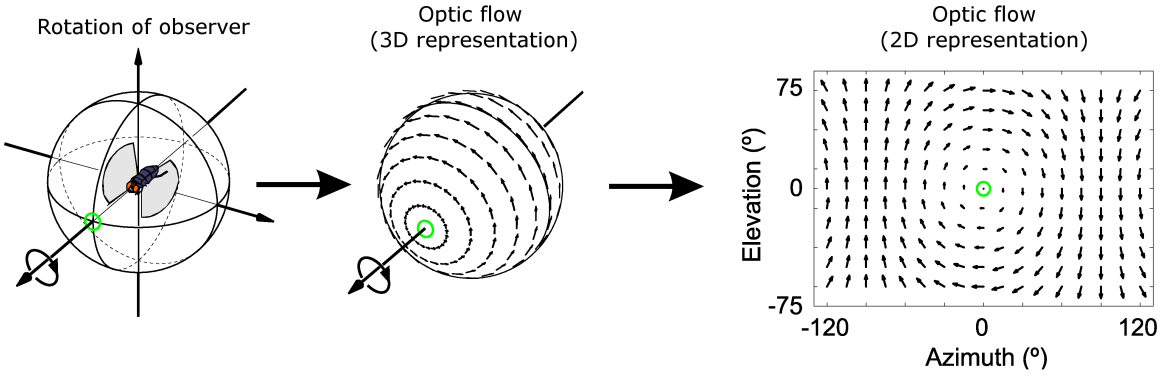


Figure 3.1: Rotation of the fly induces characteristic optical flow pattern on the fly's eye.

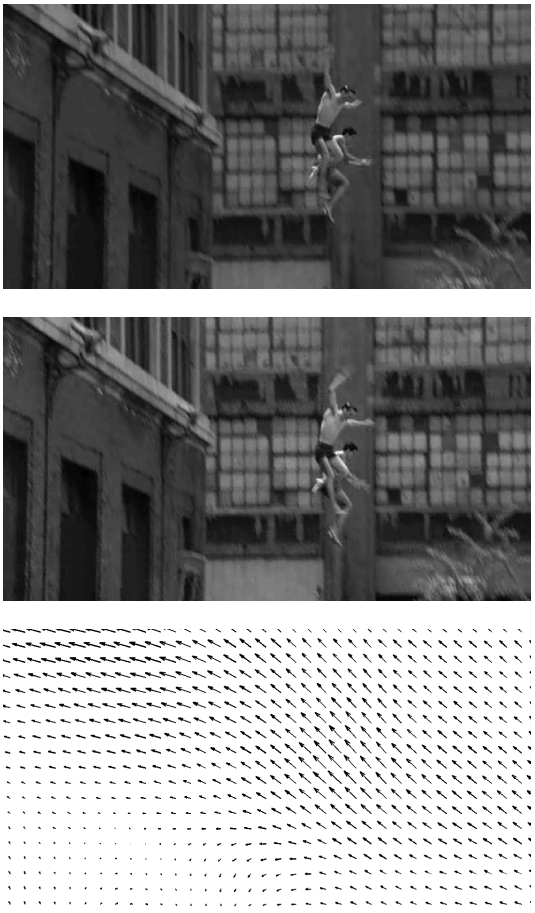


Figure 3.2: Two successive frames of motion picture and the associated optical flow field.

3.2 The model of Horn and Schunck

Horn and Schunck showed that the optical flow does not have to correspond to the motion field (see Figure 3.3 for illustration): in contrast to the "motion field" which is a purely geometrical concept relating the movement of imaged objects (resp. of the camera) to velocity vectors on the image plane by projection equations, the optical flow is the true vector field $\mathbf{u}$ that relates two successively captured intensity images I_0 and I_1 by a general coordinate transformation:

$$I_1(\mathbf{x} + \mathbf{u}) - I_0(\mathbf{x}) = 0 \tag{3.1}$$

In order to calculate the optical flow, Horn and Schunck derived the nowadays still used *optical flow constraint equation* or now preferably referred to as the **brightness constancy equation**, which is given by a first order Taylor series development of (3.1):

$$\langle \nabla I, \mathbf{u} \rangle + \frac{\partial I}{\partial t} = 0 \tag{3.2}$$

However, the above equation is not sufficient in order to solve for $\mathbf{u}$ and another equation had to be introduced. Horn and Schunck formulated the assumption that for smooth (camera) motion the optical flow should vary smoothely over the image. As a measure of departure of smoothess they use the square norm of the spatial derivatives of the optical flow. The optical flow calculation is then still a highly ill-posed inverse problem, i.e. a solution is guaranteed but not its uniqueness and finally, a small error in the brightness image such as noise will be strongly amplified by the spatial derivatives. Therefore the following energy functional minimization problem had been proposed:

$$\mathbf{u}^* = \arg\min_u \int_\Omega \|\nabla \mathbf{u}\|^2 \, d\Omega + \lambda \int_\Omega \left(\langle \nabla I, \mathbf{u} \rangle + \frac{\partial I}{\partial t} \right)^2 d\Omega \tag{3.3}$$

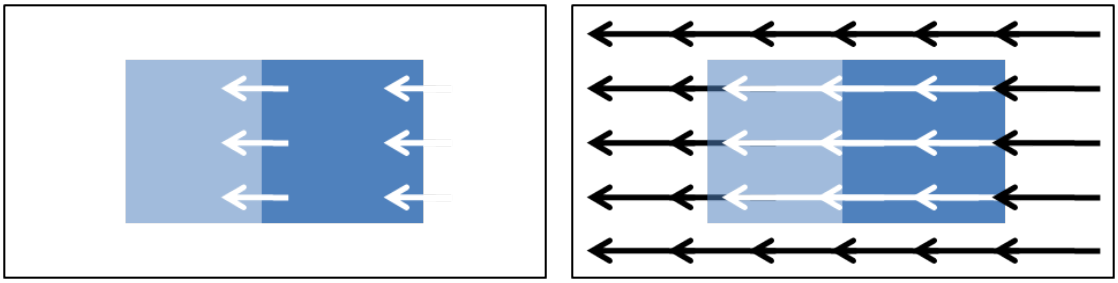


Figure 3.3: Left: the optical flow corresponds to the relative change of image brightness. Right: the motion field is the projection of motion on the image.

Where λ is an optimization parameter and controls the influence of error in the brightness constancy assumption with respect to the influence of the smoothness constraint. $\nabla \mathbf{u}$ is the gradient of $\mathbf{u}$ (and not the divergence as one could suppose from the sloppy notation) such that $\|\nabla \mathbf{u}\|^2$ evaluates to:

$$\|\nabla \mathbf{u}\|^2 = \left(\frac{\partial u_1}{\partial x}\right)^2 + \left(\frac{\partial u_1}{\partial y}\right)^2 + \left(\frac{\partial u_2}{\partial x}\right)^2 + \left(\frac{\partial u_2}{\partial y}\right)^2 \tag{3.4}$$

Evaluating the Euler-Lagrange equation yields the two following partial differential equations which can be solved using gradient descent as shown in Chapter 2:

$$\nabla^2 \mathbf{u} = \lambda \left(\langle \nabla I, \mathbf{u} \rangle + \frac{\partial I}{\partial t}\right) \nabla I \tag{3.5}$$

3.3 Problems of the Horn and Schunck model

There are three main problems in the optical flow model of Horn and Schunck:

1. the smoothness term does not allow for discontinuities in the flow field
2. the data term disfavors errors in the brightness constancy such as edges
3. solving a minimization problem for $2 \times W \times H$ parameters where W and H correspond to the width and height of the digital image was not ready for a dense recovery or even real-time application.

Those problems have driven the research on optical flow methods and many have come up since the original publication: phase correlation, block-based methods, motion of edges etc. Up to the Lucas-Kanade coarse-to-fine block-wise optical flow determination using local neighborhood assumptions on optical flow, there was not much successful innovation. Even today, the original optical flow as computed from the Horn and Schunck model yields comparable results. Good innovation came during the passing decade: in 2002, Alvarez et al. [1] proposed a formulation for symmetrical optical flow with occlusion detection, then in 2004 Brox et al. [6] proposed symmetrical modified TV-L^1 optical flow. Finally, in 2007 Zach et al. [25] proposed the TV-L^1 optical flow using a duality based approach for the solution of the non-convex optimization problem without loss of generality (in contrast to Brox et al. which use the degenerated TV-L^1 model). Furthermore, he implemented it in real-time using the innovative concept and possibility of GPGPU (General Purpose Computation on Graphics Processing Units) which had its first appearance at a conference on high-performance computing in 2004.

3.4 TV-L^1 optical flow

Motivated by the results by Chambolle [7] in 2004, who showed how to solve L^1-ROF model for image denoising in a duality based approach, Zach et al. [25] showed in Zach et al. [25] how to use Chambolle's technique to minimize the TV-L^1 energy functional for optical flow. The model proposed by Zach et al. is the following:

$$\mathbf{u}^* = \arg\min_u \int_\Omega \lambda\Phi(I_0, I_1, \mathbf{u}, ...) + \Psi(\mathbf{u}, \nabla\mathbf{u}, ...)d\Omega \tag{3.6}$$

$\Phi(I_0, I_1, \mathbf{u}, ...)$ and $\Psi(\mathbf{u}, \nabla\mathbf{u}, ...)$ are the data and regularization terms respectively and λ is a weighting factor of those terms. When choosing $\Phi = \Phi(\mathbf{x}) = \|\mathbf{x}\|^2$, $\Psi = \Psi(\nabla\mathbf{u}) = \|\nabla\mathbf{u}\|^2$ and doing a 1st order Taylor development of $I_1(\mathbf{x} + \mathbf{u})$, we find exactly the Horn-Schunck model as presented previously. To bypass the shortcomings of the Horn-Schunck model, different terms are chosen: a robust L^1 data term $\Phi = \Phi(\mathbf{x}) = \|\mathbf{x}\|$ and a total variation regularization term $\Psi = \Psi(\nabla\mathbf{u}) = \|\nabla\mathbf{u}\|$. This results in the TV-L^1 optical flow model:

$$\mathbf{u}^* = \arg\min_u \int_\Omega \|\nabla\mathbf{u}\| + \lambda \left|I_1(\mathbf{x} + \mathbf{u_0}) + \langle\nabla I_1(\mathbf{x} + \mathbf{u_0}), \mathbf{u}\rangle - I_0(\mathbf{x})\right| \, d\Omega \tag{3.7}$$

What seems to be a simple change of the used norm at first, reveales as a difficult optimization problem. In fact (3.7 represents a not stricly convex optimization which is shown clearly by its Euler-Lagrange equation:

$$\lambda \, \frac{\rho(I_0, I_1, \mathbf{u})}{|\rho(I_0, I_1, \mathbf{u})|} - \nabla \cdot \frac{\nabla\mathbf{u}}{\|\nabla\mathbf{u}\|} = 0 \tag{3.8}$$

ρ denotes the image residual:

$$\rho = \rho(\mathbf{u}) = \rho(I_0, I_1, \mathbf{u}) = \langle I_1(\mathbf{x} + \mathbf{u_0}), \mathbf{u}\rangle - I_0(\mathbf{x}) \tag{3.9}$$

An approximation, a **convex relaxation**, has been proposed which results in an optimization in two variables, $\mathbf{u}$ and $\mathbf{v}$, where a coupling term has been introduced such that $\mathbf{u}$ and $\mathbf{v}$ are close approximations of each other:

$$\min_{u,v} \int_\Omega \|\nabla\mathbf{u}\| + \frac{1}{2\theta}(\mathbf{u} - \mathbf{v})^2 + \lambda \left|\rho(\mathbf{v})\right| \, d\Omega \tag{3.10}$$

The above minimization problem can be split in to two distinct optimization problems which can be solved in alternative iteration steps:

$$\mathbf{u}^* = \arg\min_u \int_\Omega \|\nabla \mathbf{u}\| + \frac{1}{2\theta}(\mathbf{u}-\mathbf{v})^2 \, d\Omega \tag{3.11}$$

and

$$\mathbf{v}^* = \arg\min_v \int_\Omega \frac{1}{2\theta}(\mathbf{u}-\mathbf{v})^2 + \lambda |\rho(\mathbf{v})| \, d\Omega \tag{3.12}$$

While solving (3.12) results in a fixed-point thresholding scheme as will be shown later, (3.11) corresponds to the originally formulated ROF model for denoising. But in contrast to denoising, we cannot use simple gradient descent to solve the associated PDE since it won't reach a stable solution for $\|\mathbf{u}\| \to 0$. Fortunately, in Chambolle proposed a **dual formulation** of the ROF model and an elegant way we to solve it. The dual formulation of the ROF model is obtained using the **dual formulation of the TV norm**:

$$\|\nabla \mathbf{u}\| = TV(\mathbf{u}) = \max\{\mathbf{p}\cdot\nabla\mathbf{u} : \|\mathbf{p}\| \leq 1\} \tag{3.13}$$

Inserting in the ROF model yields its dual formulation:

$$\min_{u,\|\mathbf{p}\|\leq 1} \left\{ -\int_\Omega \mathbf{p}\cdot\nabla\mathbf{u}\, d\Omega + \frac{1}{2\lambda}\int_\Omega (\mathbf{u}-\mathbf{v})^2 \, d\Omega \right\} \tag{3.14}$$

The Euler-Lagrange equation shows the relation between $\mathbf{u}$ and $\mathbf{p}$:

$$\mathbf{u} = \mathbf{v} + \lambda\nabla\cdot\mathbf{p} \tag{3.15}$$

Furthermore $\mathbf{p}$ is obtained by solving the very remarkable constrained minimization problem found by Chambolle:

$$\mathbf{p}^* = \arg\min_{\|\mathbf{p}\|\leq 1} \frac{1}{2}\int_\Omega (\mathbf{v}+\lambda\nabla\cdot\mathbf{p})^2 \, d\Omega \tag{3.16}$$

And finally, $\mathbf{p}$ can be calculated iteratively using the iterative **Chambolle algorithm**:

$$\mathbf{p}^{n+1} = \frac{\mathbf{p}^n + \frac{\tau}{\theta}(\nabla(\mathbf{v}+\lambda\nabla\cdot\mathbf{p}^n))}{1+\frac{\tau}{\theta}(\nabla(\mathbf{v}+\lambda\nabla\cdot\mathbf{p}^n))} \tag{3.17}$$

Optimizing for the approximation variable $\mathbf{v}$ is given by formulating and solving the Euler-Lagrange equation for (3.12) which is easy since (3.12) does not depend on $\nabla\mathbf{v}$:

$$\mathbf{v} = \mathbf{u} + \lambda\theta \frac{\rho(\mathbf{v})}{|\rho(\mathbf{v})|} \nabla I_1 \tag{3.18}$$

Alternatively this can be percieved as a soft-thresholding step with respect to $\rho(\mathbf{u})$:

$$\mathbf{v} = \mathbf{u} + \begin{cases} \lambda\theta\nabla I_1 & \text{if} \quad \rho(\mathbf{u}) < -\lambda\theta\|\nabla I_1\|^2 \\ -\lambda\theta\nabla I_1 & \text{if} \quad \rho(\mathbf{u}) > \lambda\theta\|\nabla I_1\|^2 \\ \lambda\theta\frac{\nabla I_1}{\|\nabla I_1\|^2} & \text{if} \quad \rho(\mathbf{u}) \leq \lambda\theta\|\nabla I_1\|^2 \end{cases} \tag{3.19}$$

3.5 Algorithm and parameters

Originally Horn and Schunck locally linearized the optical flow by a 1st order Taylor development. To overcome this disadvantage, a **multi-scale/coarse-to-fine** method can be employed such that the equations described in the previous section are solved at different levels of resolution, i.e. using downsampled versions of I_0 and I_1. Sometimes such an approach is referred to as pyramid method. This will in turn result in flow fields ${}^k\mathbf{u}$ and their approximation respectively alternate projections ${}^k\mathbf{v}$ and ${}^k\mathbf{p}$ with limited resolution where the superscript k denotes the level. For convention, the coarsest level is given by $k = L - 1$ with L the total number of levels. The initial flow field at the coarsest level, i.e. ${}^{L-1}\mathbf{u}$ will be initialized to zero which is well justified. The complete algorithm algorithm for the solution of the TV-L^1 optical flow based on a convex relaxation and solved using a duality based approach for the dual ROF model is illustrated in Figure 3.4. For numerical concerns and its implementation using GPGPU refer to Chapter 6.

It is shown that the algorithm converges for a time step $\tau \leq \frac{1}{2d}$ where d is the dimension of the problem (of $\mathbf{u}$), i.e. here it converges for $\tau \leq \frac{1}{4}$. Furthermore, the parameters λ and θ should be chosen such that $\lambda\theta \approx 5 \ldots 20\%$ of the numerical gray-level range of the input images. I.e. for normalized images we propose $\lambda = 1.0$ and $\theta = 0.1$. Using $\lambda > 1.0$ or $\lambda < 1.0$ does not yield good results in general. Decreasing θ leads to a more accurate TV-L^1 model but requires much more iterations for the algorithm to give good results. Increasing θ results in much more fast convergence but the obtained flow will be less accurate and more smooth. Finally, the number of levels in the coarse-to-fine approach should be set such that the coarsest resolution is somewhere between 20×20 and 40×40. Using less levels does not lead to good results at all while using more levels is not needed. We use a constant resolution scaling factor of 2 between the different levels.

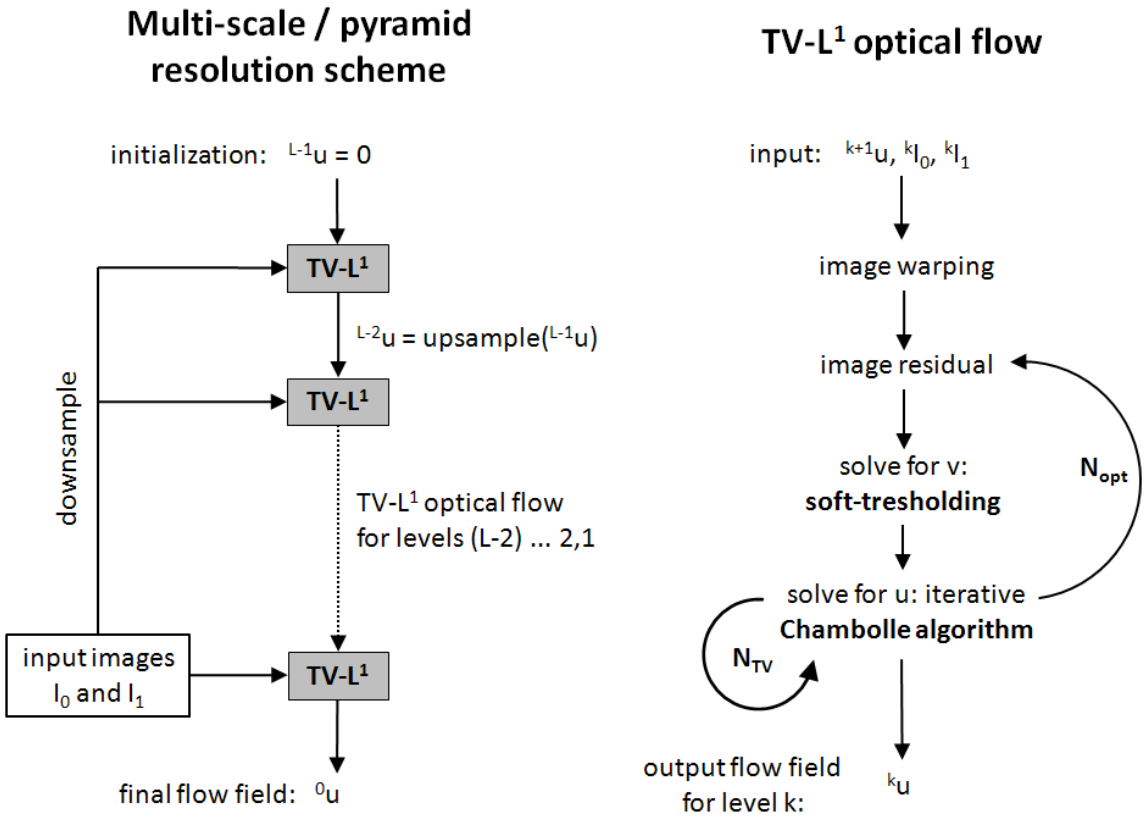


Figure 3.4: The TV-L^1 optical flow algorithm together with the pyramid method scheme.

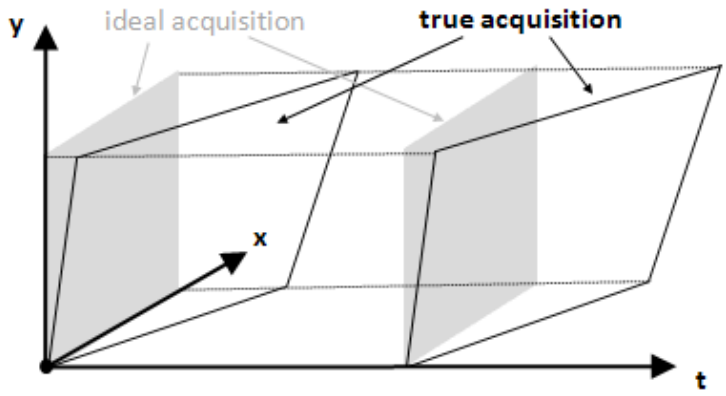


Figure 3.5: True CMOS image acquisition has linear distortion as a function of time.

3.6 Optical flow to remove CMOS rolling shutter distortion

As a small but very nice application, the implemented optical flow algorithm has been used in order to correct image sequences acquired by CMOS imaging sensors. Details can be found in D'Angelo [8]. In brief, the problem is the following: for CMOS sensors, the acquisition of an image, i.e. of the pixels constituting the image, is not achieved in an infinetly small time δt, but there is a non-negligeable quasi-constant delay in the acquisition for each successive pixel resulting in geometrical distortion if the camera is moved fast with respect to the delay during acquisition. The problem is illustrated in Figure 3.5.

Robust dense optical flow can be used to find every pair of corresponding pixels for two successive images. Knowing the relationship between the delay and the introduced geometrical distortion, a complete image sequence can be corrected. The geometrical distortion is given by:

$$\mathbf{r}_\tau = \frac{1}{t' - t}\left((\mathbf{x}t' - t\mathbf{x}') + \tau(\mathbf{x} - \mathbf{x}')\right) \tag{3.20}$$

τ corresponds to the instant τ where the distortion shall be calculated. t and t' can be expressed by the following linear geometrical relationship and belong to the (relative) time instants of acquisition of the pixels at coordinates $\mathbf{x} = (x,\ y)^T$ and $\mathbf{x}' = (x',\ y')^T$ for the successive frames $I = I_0$ and $I' = I_1$ respectively:

$$t = \frac{x + W\,y}{WH} - 1 \tag{3.21}$$

and

$$t' = \frac{x' + W\,y'}{WH} \tag{3.22}$$

The begin of each acquistion is quantized to integer instants in time such that the acquisitions of I and I' begin at instants $t = -1$ and $t = 0$ respectively. Therefore τ can be set to any value in the interval $[-1, 1]$ although the applied correction will be very sensitive to its choice and is also dependant on the used correction scheme, i.e. if correction is applied between two uncorrected or between a corrected and an uncorrected frame. And finally, $\mathbf{x}'$ is simply obtained by optical flow $\mathbf{u}$:

$$\mathbf{x}' = \mathbf{x} + \mathbf{u} \tag{3.23}$$

3.7 Results and discussion

Optical flow can by calculated for any pair of successive (gray-level) images, be they synthetic, obtained from video sequences or even from distinct images obtained by capturing the same scene under the same lighting conditions but with a parallax. Since the research on optical flow determination methods is quite broad, there exist many conventions on how to assess accuracy and how to optimally represent optical flow. Various test sequences with and without ground truth (i.e. the true underlying optical flow) can be found when examining literature. For this Master's thesis, the Yosemite sequence and some of the Middlebury datasets are chosen in order to benchmark our imlementation of the TV-L^1 optical flow. And finally, optical flow and rolling shutter correction of a true video sequence is shown.

3.7.1 Representing optical flow

Optical flow can be represented either conventionnaly by drawing arrows on the image where the length and direction are given by the module and angle of the field respectively:

$$\|\mathbf{u}\| = \sqrt{u_1^2 + u_2^2} \qquad \text{and} \qquad \arg(\mathbf{u}) = \arctan \frac{u_2}{u_1} \tag{3.24}$$

This is a very intuitive and appelling form of representation but it has several drawbacks:

- The optical flow is given for every pixel. Important sub-sampling of the field must be applied in order to correctly perceive the visualization. Normally, a constant sub-sampling in both x- and y-direction is applied with distances reaching from 5 up to 30 pixels.
- In general, the optical flow module is very small for methods using sparseness constraints such as TV-L^1 and in areas with low gradient it even tends to zero, if it is not initialized with a previous solution in order to guarantee large-module-flow. Therefore, the "arrow" cannot be seen and the flow seems to be negligeable. It has to be

prolongated artificially. A good choice is around 1/3 to 1 times the flow sub-sample distance.

Using false color coding of the (normalized) flow module and angle and showing them directly as distinct image(s) is good for assessing accuracy and comparision but is absolutely not an intuitive representation. There is another representation of the optical flow as described in Baker et al. [3]. This coding, let's call it the *HSV coding of optical flow*, has the advantage that it is good for comparison and gives still a more or less intuitve view. Furthermore, we don't have to manually play around with or sub-sample the flow in order to visualize it conveniently. Since it has been presented together with the publication of the Middlebury dataset for optical flow evaluation, it has become the quasi de facto standard for (comparable) optical flow representaion in research publications. It is given by coding an image in the HSV color space with $(H,\ S,\ V)^T \subset [0,1] \times [0,1] \times [0,1]$ where H is hue (color), S saturation and V value (darkness) as given by the optical flow as follows:

$$\begin{aligned} H &= \frac{\arg(\mathbf{u})}{2\pi} \\ S &= \frac{\|\mathbf{u}\|}{\max \|\mathbf{u}\|} \\ V &= 1 \end{aligned} \tag{3.25}$$

3.7.2 Assessing accuracy

Finally, accuracy can be assessed by using similar measures. Given a reference optical flow $\mathbf{u}_{\text{ref}}$ which e.g. can be ground truth flow or a flow obtained using another method, the following two measures are the ones most encountered in literature:

$$\begin{aligned} AE &= \arccos\left(\frac{\mathbf{u'}^T \mathbf{u'}_{\text{ref}}}{\|\mathbf{u'}\|\|\mathbf{u}\|}\right) \\ EE &= \|\mathbf{u'} - \mathbf{u'}_{\text{ref}}\| \end{aligned} \tag{3.26}$$

AE means angular error and EE endpoint error respectively. $\mathbf{u'}$ and u'_{ref} are the the flow vectors in projective geometry, i.e. $\mathbf{u'} = (u_1,\ u_2,\ 1)^T$ for example. Sometimes also the (per-pixel) averages of the above measures are given:

$$\begin{aligned} AAE &= \frac{\|AE\|}{WH} \\ AEE &= \frac{\|AEE\|}{WH} \end{aligned} \tag{3.27}$$

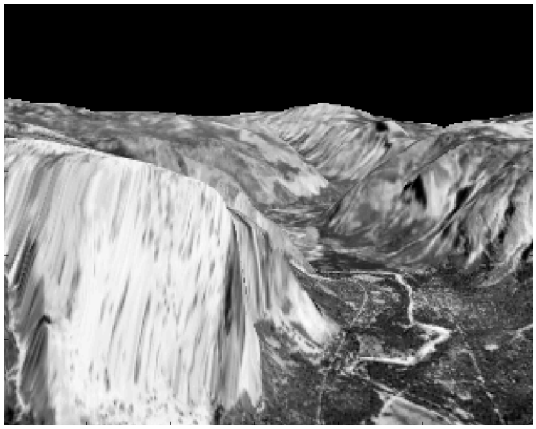
Finally, when no ground truth data is given, the only way to verify the correctness of the optical flow is to evaluate the warping error image $WE = I_1(\mathbf{x} + \mathbf{u}) - I_0$. Ideally it should be zero everywhere, but in practice we find pixel values going up to $0.1 \ldots 0.2$ such that sometimes it becomes difficult to say if the optical flow can be declared as "accurate enough".

3.7.3 Results

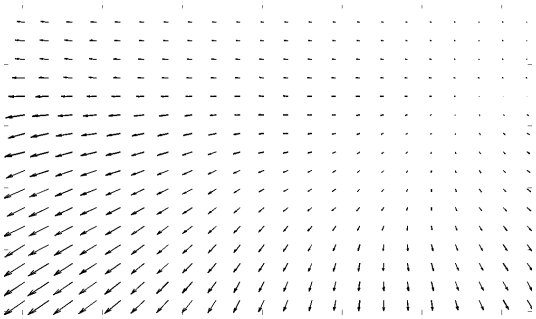
In Figures 3.6, 3.7 and 3.8 results are shown for the Yosmite, the Middlebury datasets and the rolling shutter correction respectively. Parameters for optical flow, errors and other details are given either in Table 3.1 or together with the figures.

Sequence	**Levels**	$n_{\mathbf{iter}}$	$n_{\mathbf{tv}}$	λ	θ	**AEE**	**AAE** [°]
Real sequences:							
Dimetrodon	6	33	3	1.0	0.99	0.52	10.2
Hydrangea	6	33	3	1.0	0.99	0.89	12.5
Rubberwhale	6	33	3	1.0	0.99	0.62	21.3
Synthetic sequences:							
Grove2	6	100	3	1.0	0.99	0.59	9.25
Urban2	6	100	3	1.0	0.99	7.04	35.2
Venus	6	100	3	1.0	0.99	2.08	32.2

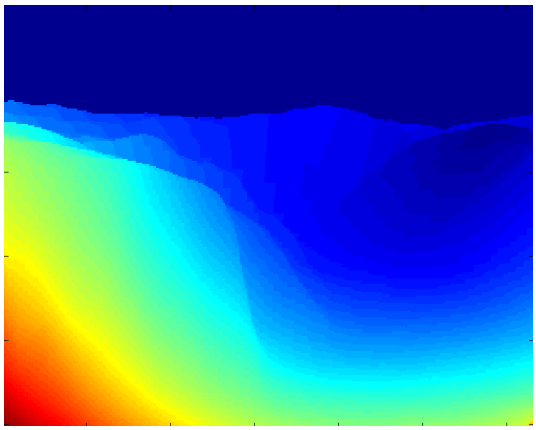
Table 3.1: A summery of parameters and errors with respect to ground truth for the Middlebury dataset. AEE: average end-point error, AEE: average angular error.



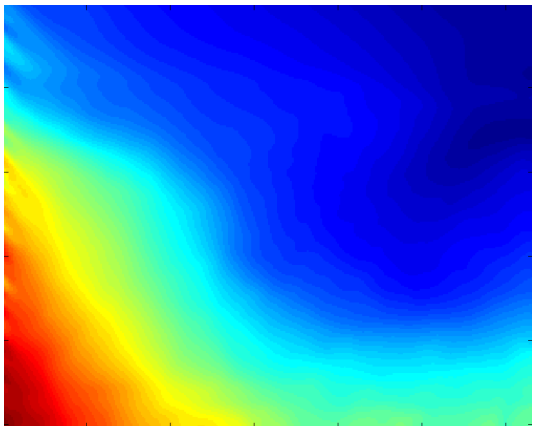
(a) Input image (only one frame shown)



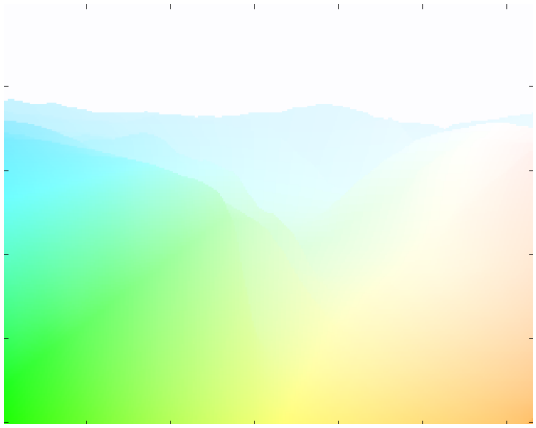
(b) Optical flow field by TV-L^1



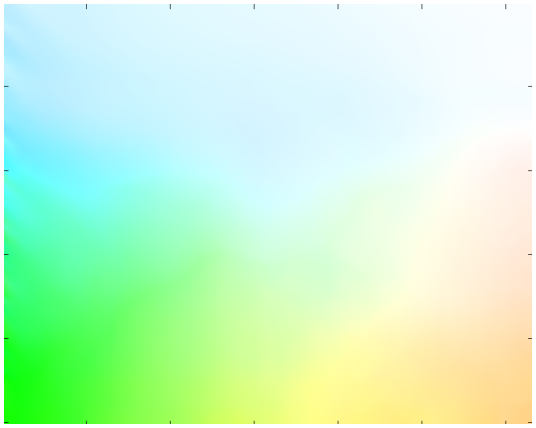
(c) Ground truth (GT) module



(d) TV-L^1 module

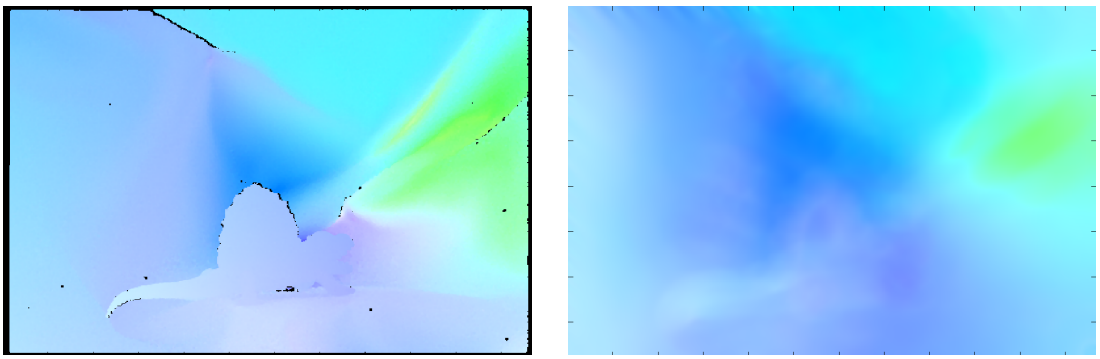


(e) GT HSV coding



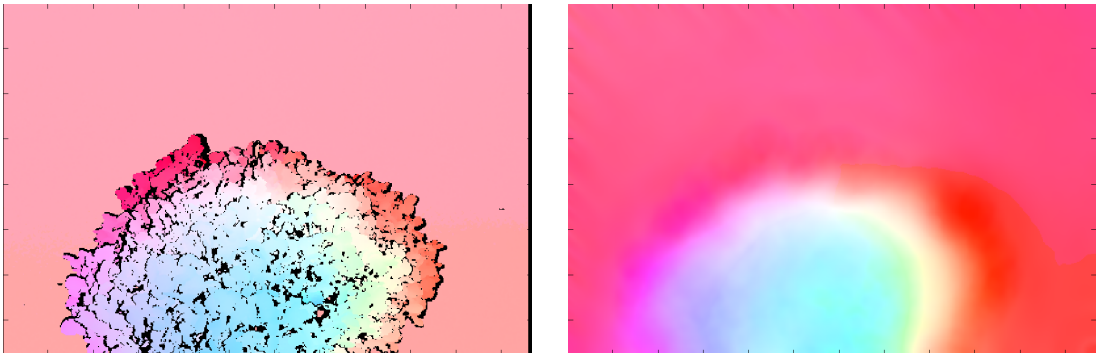
(f) TV-L^1 HSV

Figure 3.6: Optical flow for frames 1 and 2 of the synthetic, cloudless Yosemite dataset. Left column: ground truth. Right column: recovered by TV-L^1. TV-L^1 parameters: 5 levels, 33 global iterations, 3 Chambolle iterations, $\lambda = 1.0$, $\theta = 1.0$. Average end-point error: 0.28. Average angular error: 9.9°.



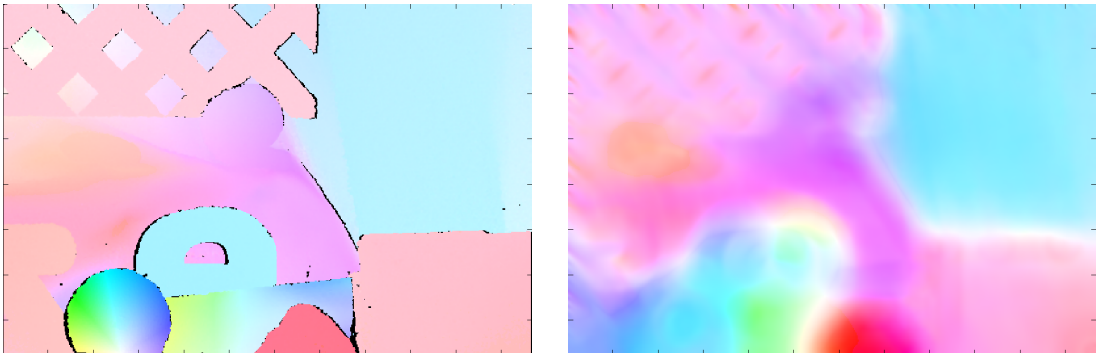
(a) Dimetrodon ground truth optical flow

(b) Dimetrodon TV-L^1 flow



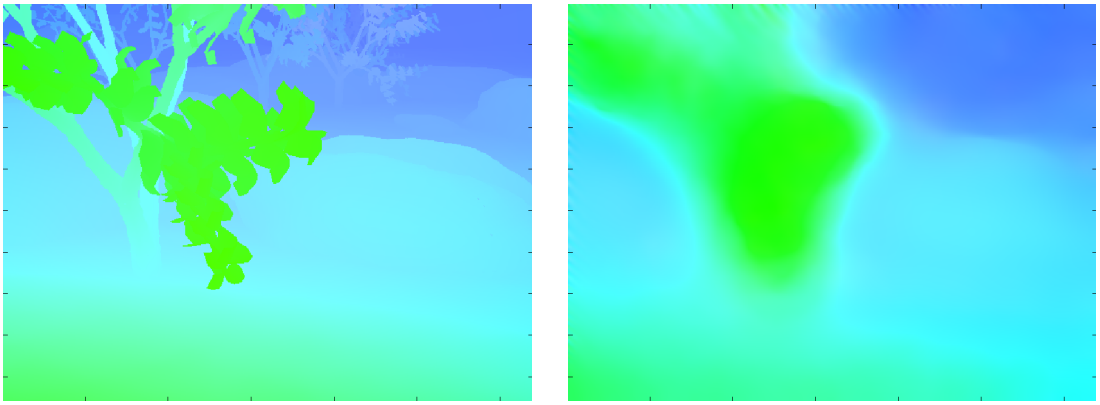
(c) Hydrangea GT

(d) Hydrangea TV-L^1



(e) Rubberwhale GT

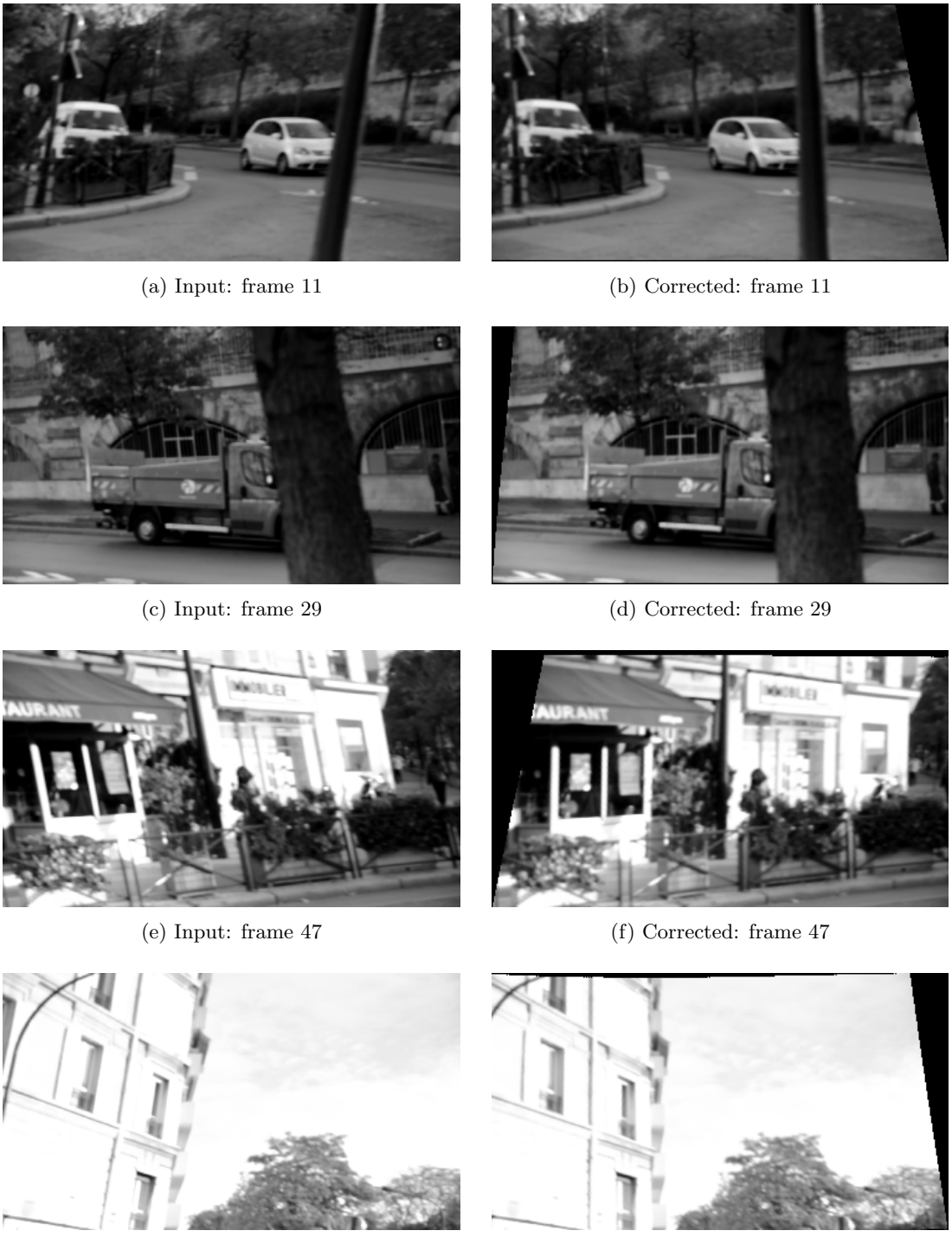
(f) Rubberwhale TV-L^1



(g) Grove2 GT

(h) Grove2 TV-L^1

Figure 3.7: Optical flow for frames 10 and 11 of the Middlebury dataset. Left column: ground truth. Right column: recovered by TV-L^1.



(a) Input: frame 11

(b) Corrected: frame 11

(c) Input: frame 29

(d) Corrected: frame 29

(e) Input: frame 47

(f) Corrected: frame 47

(g) Input: frame 71

(h) Corrected: frame 71

Figure 3.8: Left: input from a true sequence by a fast moving hand-held CMOS camcorder. Right: corrected images. TV-L^1 parameters: 5 levels, 20 global iterations, 5 Chambolle iterations, $\lambda = 1.0, \theta = 0.5, \Delta t = 0.125$. Out-of-image warps are displayed in black. Performance: being an old implementation combining OpenCV and GPGPU, approx 1-2 fps.

3.7.4 Discussion

From a first glance at the results, the implemented TV-L^1 optical flow algorithm seems to perform well for the Yosemite sequence. However, when analyzing in detail we see that the recovered flow is smoother than the ground truth. The reason is that θ has been set to 1.0 and thus the TV-L^1 model is not well approximated. Unfortunately, when reducing θ we obtain recovery errors which are higher than the reached $AEE = 0.28$ and $AAE = 9.9°$. In order to obtain good results for reduced θ we would have to heavily increase the number of iterations, depending on the choice of of θ, to 20'000–100'000 iterations.

The same comments hold for the true training Middlebury test sequences (Dimetrodon, Hydrangea, Rubberwhale). But there is a big difference: the underlying optical flows are in no way trivial; they contain a lot of discontinuities and have every possible numerical value from approximately -15 to +15 pixels. Given those difficulties, it is obvious that the recovery errors are 2-3 times higher than for the Yosemite sequence.

The Middlebury synthetic sequences are still more challenging, since they contain large areas with a very smoothly and slightly varying optical flow and suddenly a big discontinuity. Of course such sequences are far from reality and thus it is not surprising that methods destined for optical flow estimation for true images can not recover accurately such syntethic flow patterns.

A comparison on `vision.middlebury.edu` reveals that for the Yosemite sequences our TV-L^1 optical flow is approximately in the middle ranks with respect to the AEE, for the Grove sequence it might be even in the better ranks. But for the Urban sequence it is performing very poorly. The AAE are in generally quite well correlated to the AEE and thus, a comparison with respect to AAE leads to the same conclusions.

Given these not too bad pseudo-ranking results we can conclude that the implemented and used optical flow in this Master's project works quite well. However, the currently best performing implementations yield errors which are 2–3 times lower, e.g. [23] or [24]. Therefore it would be advantageous to update our implementation of the optical flow soon. We could take inspiration from the lastest convincing results and develop an even improved optical flow algorithm by using FISTA [4] and eventually something like a wavelet based approach in order to better handle discontinuities in the flow field.

Finally, we see that the rolling shutter correction works nicely for the shown sequence which is a clear sign for a well implemented optical flow. Unfortunately the correction does not work equally well for other tested sequences which in general contain much less camera motion. This is very probably due to bad settings of the optical flow. Concretely, it would be much better to use functional evaluation instead of a fixed number of iterations. Furthermore, we would have to assess efficieny of the rolling shutter correction when setting $\theta \to 1$ which, as we've seen, decreases the optical flow error if only few iterations are used.

Chapter 4

Structure from Motion

4.1 Introduction

The efficient three-dimensional recovery of a scene from images has been a long-term aim in computer vision. Successful methods would have a big impact on a broad range of fields such as autonomous navigation, biomedical imaging, virtual and augmented reality, architecture, modern photography and film industry. Since the 1980s, the geometry that links the 3D structure of a scene and its projection on images has been studied thoroughly in **"Multi-view geometry"** [12] or "The geometry of multiple images" [9]. In "Robot Vision" [15] all approaches found until 1985 for the recovery of the 3D structure of the scene are well summarized for the first time. At that time there was one main approach: **stereo photography** of the scene and its reconstruction using **epipolar geometry**.

4.1.1 Multi-view stereo

Stereo photography means capturing a scene from two clearly distinct points of view, i.e. with a big distance between the cameras such that a big parallax is obtained. As time evolved, methods were found to include more than two cameras and to reconstruct the scene even if cameras are not properly aligned or if their calibration is not fully known. Those methods are generally referred to as **multi-view stereo reconstruction** (see Figure 4.1).

Reconstructing the scene means usually to calculate the distance to the objects of the scene, i.e. the percieved depth. So the usual aim in scene recovery is the recovery of the **depth map** which contains the apparent depth to objects in the scene (up to some constant scale factor) or its inverse. In order to reconstruct the scene correctly, the epipolar geometry is evaluated for matching pairs of the multi-view images. The challenging problem in this approach is to obtain those matching pairs of points which in general is called the **"correspondence problem"** and methods for its solution are usually referred to as "image registration". Nowadays there exists a big variety of registration methods, e.g. methods

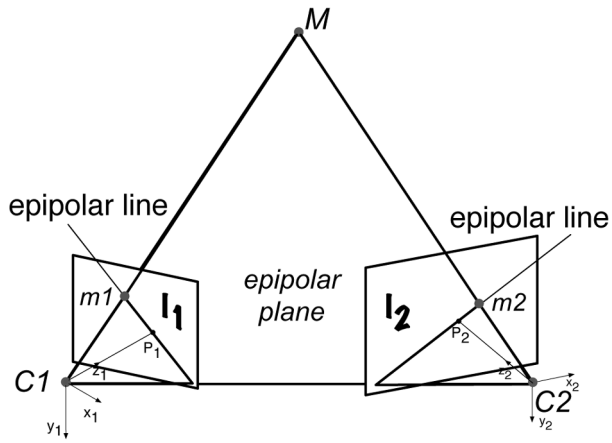


Figure 4.1: Schematic stereo camera view of a scene and its associated epipolar geometry. Using the epipolar constraint equation $(\mathbf{R}^{\mathbf{T}}\mathbf{x_m})^{\mathbf{T}}\mathbf{t} \times \mathbf{x'_m} = \mathbf{0}$, the distance to point M to either camera can be found, once enough points $\mathbf{x_m}$ and $\mathbf{x'_m}$ are matched such that the equation system is well defined. $\mathbf{x_m}$ and $\mathbf{x'_m}$ are vectors pointing to M from the camera centers C_1 and C_2. $\mathbf{R}$ and $\mathbf{t}$ are rigid rotation and translation respectively for $C_1 \rightarrow C_2$.

based on RANSAC (Random Sample Consensus) have become popular since they are very robust and accurate. Since the computational cost of the matching process is rather high, usually only the minimum number up to some dozens of points are matched in order to solve the epipolar geometry problem. Clearly such a recovery results in what is called a "sparse depth map", i.e. after reconstruction we only know depth at very few points in the image.

Naturally sparse depth recovery is somehow very unsatisfying when we want to know the 3D scene in its integrality and much research has been carried out in order to fuse multiple (sparse) depth maps to a obtain coherent dense depth maps or even 3D models.

4.1.2 Structure from motion

Another way to recover the 3D structure of the scene is to use successive images, i.e. a video sequence, acquired by only one camera which is mounted on a mobile robot or, more recently, by a hand-held camera or camcorder. This is type of recovery named **"structure from motion"**. The geometry is the same as in multi-view stereo, and thus traditionnaly the methods involving the correspondence problem and epipolar geometry have been applied. The major difference to multi-view stereo is that the camera motion has to be known in order to evaluate the epipolar geometry. Various methods have been proposed in literature to estimate this motion from the picture the camera acquires, generally known as **ego-motion estimation**—all with their benefits and drawbacks. In general, when compared to multi-view stereo reconstruction, it is much more difficult to obtain an accurate depth estimation in structure from motion due to following issues:

- The epipolar geometry problem cannot be solved precisely and/or uniquely since in general there is very small parallax for successive images and a numerical solution is very instable and error prone.
- It cannot be solved properly because their is much error in the ego-motion estimation due to the drawbacks of the applied methods (noise and numerical problems normally).
- It cannot be solved correctly since with many methods the correspondence problem is hard to solve for very similar images such as found in usual video sequence.

Moreover, either to reduce computational cost or to add more robustness in the ego-motion estimation, tracking of feature points is applied usually in order to omit the correspondence problem once it is solved at an initial step. Since tracking of image features generally results in a highly nonlinear and thus suboptimal tracking scheme (e.g. extended Kalman or particle filtering), additional errors in this approach can reduce the accuracy of the scene recovery.

4.1.3 Fast structure from motion

Very recently, Bagnato et al. [2] proposed a variational method to solve the ego-motion, the correspondence problem and depth estimation in a nice dyadic iterative scheme which results in fast convergence. In this chapter, it is shown how to adapt his framework, originally formulated for omnidirectional cameras, to planar images.

4.2 Image formation

4.2.1 Central projection

For successful 3D reconstruction of a scene it is important to know how a scene is projected on an image for planar image sensors. Here we don't account for the physical part concerning light integration etc. but only for the geometric description of the used projection model. Given a pinhole camera model which is the usual model in computer vision, a point (x, y) on the image, in a simplistic description, corresponds to the central projection of a point (X, Y, Z) in the scene given the equivalent focal length f of the complete lens and camera focal system as illustrated in Figure 4.2. Thus we have:

$$x = f\frac{X}{Z} \qquad \text{and} \qquad y = f\frac{X}{Z} \tag{4.1}$$

All measures f, x, y and X, Y, Z are in arbitrary units. Furthermore, in this model the optical axis is in the center of the image with x and y going from $-W/2$ to $+W/2$ and $-H/2$ and $H/2$ respectively where W and H are arbitrary. In reality, the image domain is usually defined for positive values only, thus it would involve a coordinate shift of about (x_0, y_0)

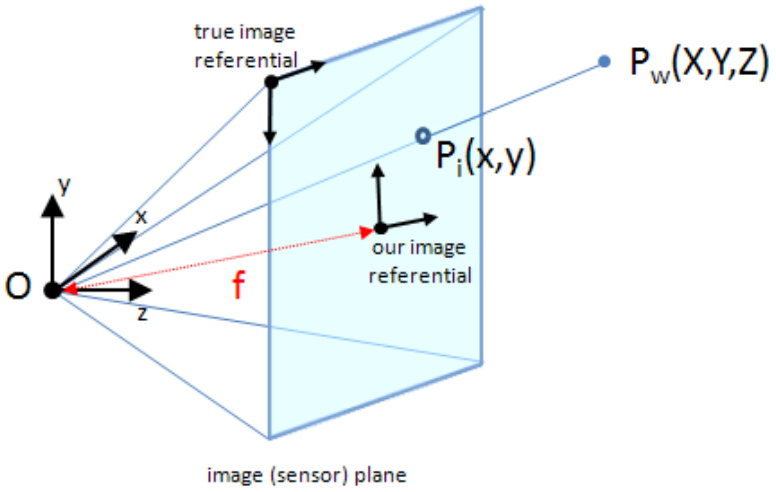


Figure 4.2: The geometry of image formation for an uncalibrated camera.

which defines the intersection of the image plane with the optical axis. The true relationship describing the mapping of X, Y, Z in a metric system to and x, y in pixels is not established here but can be achieved by using information on the used sensor and performing **camera calibration**. For the validity of the depth map estimation, a calibrated camera is not needed explicitely and therefore this subject is not handled in this thesis.

The relationship between (x, y) and (X, Y, Z) is nonlinear in the depth Z, i.e. the distance of a point in the scene to the focal point of the camera lens system. We further denote depth by d and its inverse, the depth map by Z, such that (4.1) becomes linear in depth map:

$$x = f\frac{X}{d} = fXZ \qquad \text{and} \qquad y = f\frac{X}{d} = fXZ \tag{4.2}$$

4.2.2 Projection of camera movement

It can be shown that camera movement and movement of the scene are interchangable as long as the scene follows a rigid model, i.e. does not contain deformation. We can extend the projection model given above for a moving camera. This is shown in 4.3. We model the camera movement using general 3D translation $t = (t_x, t_y, t_z)^T$ and rotation $\Omega = (\Omega_x, \Omega_y, \Omega_z)^T$. When the camera moves linearly by t and rotates by Ω, a point $\mathbf{p} = (X, Y, Z)^T$ in the scene is then transformed to $\mathbf{p}'$:

$$\mathbf{p}' = \mathbf{p} - (\mathbf{t} + \boldsymbol{\Omega} \times \mathbf{p}) \tag{4.3}$$

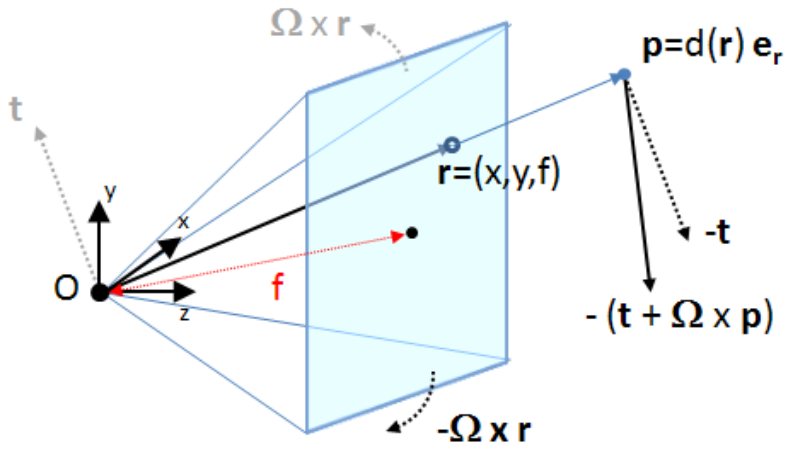


Figure 4.3: Intechangeability of camera and scene movement for a ridig scene.

Alternatively we can write $\mathbf{p}$ with respect to the image plane:

$$\mathbf{p} = d(\mathbf{r})\ \mathbf{e_r} \tag{4.4}$$

where $d(\mathbf{r}) = d(x, y, f)$ is the depth of the point $\mathbf{p}$ in the direction of the point $(x, y, f)^T$ on the image plane. $\mathbf{e_r}$ is the unit vector pointing at a point in the image plane (hint: it only lies on the image plane for $(x, y) = (0, 0)$ whereas it is $\mathbf{r} = (x, y, f)$ that points to the image exactly), given by:

$$\mathbf{e_r} = \frac{\mathbf{r}}{\|\mathbf{r}\|} = \frac{1}{\sqrt{x^2 + y^2 + f^2}} (x,\ y,\ f)^T \tag{4.5}$$

4.2.3 Relating projected camera movement and motion field

Translational movement

The projection of the relative motion $\Delta\mathbf{p} = \mathbf{p}' - \mathbf{p}$ on the image plane is illustrated in Figure 4.4. It can be described as a projection in two steps:

1. Projection $\mathbf{t_p}$ at $\mathbf{p}$ on the plane parallel to the image plane.
2. Projection $\mathbf{v_t}$ of $\mathbf{t_p}$ on the image plane.

For simplicity, let's consider only translational movement at first. Thus, the first projection can be calulated exploiting the similarities between triangles. We denote $(\mathbf{p})_{x,y,z}$ the projection of a vector $\mathbf{p}$ on cartesian coordinates (x, y, z). We have the following similarity situation

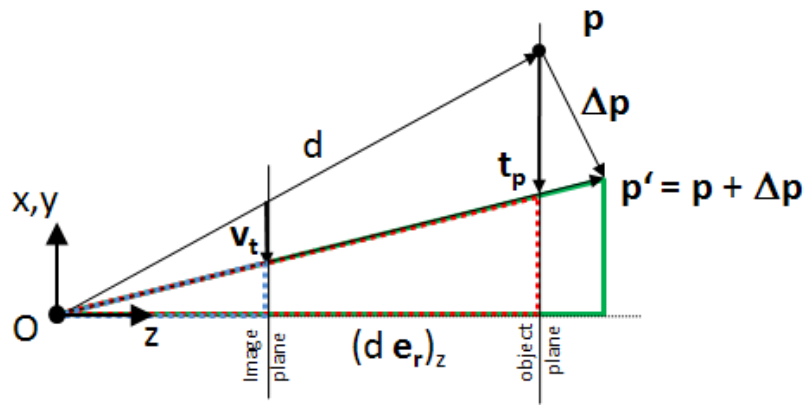


Figure 4.4: Projection of camera movement in two steps: first on the object plane, then on the image plane. We can exploit the similarity between the triangles shown in green, red and blue.

$$\frac{(\mathbf{p}+\mathbf{t})_{x,y}}{(\mathbf{p}+\mathbf{t})_z} = \frac{(\mathbf{p}+\mathbf{t_p})_{x,y}}{(\mathbf{p}+\mathbf{t_p})_z} \tag{4.6}$$

This results in:

$$\mathbf{t_p} = (\mathbf{t_p})_{x,y} = (\mathbf{p})_z \frac{(\mathbf{p}+\mathbf{t}_{x,y})}{(\mathbf{p}+\mathbf{t})_z} - (\mathbf{p})_{x,y} \tag{4.7}$$

Using $\mathbf{p} = d\frac{\mathbf{r}}{\|\mathbf{r}\|}$ leads to:

$$\mathbf{t_p} = d\frac{r_z}{\|\mathbf{r}\|} \cdot \frac{d\frac{(\mathbf{r})_{x,y}}{\|\mathbf{r}\|} + (\mathbf{t})_{x,y}}{d\frac{r_z}{\|\mathbf{r}\|} + t_z} - d\frac{(\mathbf{r})_{x,y}}{\|\mathbf{r}\|} \tag{4.8}$$

Cleaning up (4.8) and substituting depth d by (inverse) depth map Z leads to the **parallel projection model for planar images** for linear camera translation:

$$\mathbf{t_p} = \frac{r_z}{\|\mathbf{r}\|Z} \cdot \frac{(\mathbf{r})_{x,y} + \|\mathbf{r}\|Z\,(\mathbf{t})_{x,y}}{(\mathbf{r})_z + \|\mathbf{r}\|Z\,(\mathbf{t})_z} - \frac{(\mathbf{r})_{x,y}}{\|\mathbf{r}\|Z} \tag{4.9}$$

The projection $\mathbf{v}$ on the image plane can be obtained by using another triangle similarity:

$$\frac{(\mathbf{p}+\mathbf{t})_{x,y}}{(\mathbf{p}+\mathbf{t})_z} = \frac{(\mathbf{r}+\mathbf{v_t})_{x,y}}{(\mathbf{r}+\mathbf{v_t})_z} \tag{4.10}$$

Solving for $\mathbf{v_t} = (\mathbf{v_t})_{x,y}$ yields:

$$\mathbf{v_t} = \mathbf{r_z} \cdot \frac{(\mathbf{r})_{x,y} + \|\mathbf{r}\|Z\,(\mathbf{t})_{x,y}}{(\mathbf{r})_z + \|\mathbf{r}\|Z\,(\mathbf{t})_z} - (\mathbf{r})_{x,y} \tag{4.11}$$

As explained by Horn and Schunck, this geometrical projection of the camera movement on the image plane is called the **motion field**. In (4.11) it is given in explicit form as a function of the camera motion. When comparing to (4.9) we can derive an expression which relates the motion field and the parallel projection:

$$\mathbf{v_t} = \|\mathbf{r}\| Z \mathbf{t_p} \tag{4.12}$$

Horn and Schunck show that in general the motion field is not equal to the optical flow: the motion field has a geometrical meaning while the optical flow is related to movement of the brightness pattern! But still, the optical flow for areas with much texture is approximately equal to the motion field such that latter can be used as an approximation of the optical flow. We will see that this is well justified since we don't use the motion field directly, but only to constrain the optimization problems for ego-motion and depth estimation.

Rotational movement

Similar derivations can be made for rotation of the camera. As we know from multi-view geometry, depth can not be calculated from rotation since rotation does not produce any parallax. Or in other words: the motion field due to camera rotation does not depend on the depth. Therefore it makes no sense to perform a two-step projection as for the translational movement. We will only show the effect of rotation on the motion field. We exploit the following triangle similarity:

$$\frac{(\Omega \times \mathbf{r})_{x,y}}{(\Omega \times \mathbf{r})_z} = \frac{(\mathbf{r}')_{x,y}}{(\mathbf{r}')_z} \tag{4.13}$$

The rotational part of the motion field is then given by:

$$\mathbf{v_r} = \big(\mathbf{r}' - \mathbf{r}\big)_{x,y} = r_z \frac{(\Omega \times \mathbf{r})_{x,y}}{(\Omega \times \mathbf{r})_z} - (\mathbf{r})_{x,y} \tag{4.14}$$

Complete motion field

Finally the complete motion field for rigid 3D camera movement is simply given by:

$$\mathbf{v} = \mathbf{v_t} + \mathbf{v_r} \tag{4.15}$$

4.3 Ego-motion estimation

Ego-motion estimation means finding the camera motion parameters, translation $\mathbf{t} = (t_x, t_y, t_z)^T$ and rotation $\mathbf{\Omega} = (\Omega_x, \Omega_y, \Omega_z)^T$, from two sucessive images and eventually optical flow, depth map or both. Traditionally there have been the three following approaches to solve this problem [15]:

1. Discrete approach: a sparse optical flow is directly related to the six motion parameters such that they can be calculated directly.
2. Differential approach: first and second spatial derivatives of the optical flow are calculated and directly related to motion parameters.
3. Least-squares approach: the complete optical flow field is used and an over-determined system is solved.

While the approaches 1 and 2 are in no way robust to errors in the images or in the optical flow, the last approach seems to yield sufficient results. As Bagnato et al. proposed a linear least-squares method to solve the ego-motion problem for omnidirectional images, we will use it as a starting point for the ego-motion estimation for planar images. He exploits the brightness constancy assumption which in the case of omnidirectional images gives a linear constraint in the motion parameters because the optical flow/motion field and depth map are related by $\mathbf{u} = Z(\mathbf{r})\mathbf{t}_\perp$:

$$I_1 + \nabla I_1^T \left(Z(\mathbf{r})\mathbf{t}_\perp\right) - I_0 = 0 \tag{4.16}$$

where $\mathbf{t}_\perp$ is the projection of the camera movement on the object tangent plane. From the conceptual point of view, this projection is similar to the parallel projection $\mathbf{t_p}$ for planar images.

4.3.1 Solving over-determined linear systems

When a discrete gray-level image with $W \times H$ pixels is used to determine six motion parameters, the equation system will be over-determined. While $\mathbf{A} = \mathbf{MB}$ can be solved easily when it describes a linear system with N unknowns and N equations by inverting the system with $\mathbf{M} = \mathbf{AB^{-1}}$, the over-determined problem can solved in a least-squares sense resulting in:

$$\mathbf{M} = \mathbf{A}\,\mathbf{B^T}\left(\mathbf{B}\,\mathbf{B^T}\right)^{-1} \tag{4.17}$$

$\mathbf{B^T}\left(\mathbf{B}\,\mathbf{B^T}\right)^{-1}$ is called the *pseudoinverse* of $\mathbf{B}$. Unfortunately it is not easy to derive a system $\mathbf{A} = \mathbf{MB}$ relating images and motion parameters for planar images given the nonlinearities in the projection model. We will adress the problem manually where it can be solved by linear optimization and show how nonlinear optimization can be performed in the general case.

4.3.2 Linear least-squares

When looking at our projection model, we see that it is linear in one **special case: for** $t_z = 0$ **and zero rotation**, i.e. $\mathbf{\Omega} = (0,0,0)^T$. In that case we have:

$$\mathbf{t_p} = \mathbf{t} \qquad \text{and} \qquad \mathbf{v} = \|\mathbf{r}\|\, Z\, \mathbf{t_p} \tag{4.18}$$

Thus this results in a similar optimization problem as solved by Bagnato et al.. Unfortunately it is found that formulating the pseudo-inverse is not straightforward given the spatial dependancy of $\mathbf{r}, Z$ and the images I_0 and I_1 resp. its derivative ∇I_1. The linear least-squares problem is solved manually as follows. For convenience, we will use the following notation:

$$\frac{\partial I_k}{\partial t_k} = I_1 - I_0 \tag{4.19}$$

The error function to be minimized with respect to $\mathbf{t}$ is given by:

$$E_{\text{lin}}(\mathbf{t}) = \sum_{(x,y)\in D} \left[\frac{\partial I_k}{\partial t_k} + \nabla I_1^T \left(\|\mathbf{r}\| Z \mathbf{t}\right)\right]^2 \tag{4.20}$$

Minimizing with respect to $\mathbf{t}$ gives:

$$\frac{d\, E_{\text{lin}}(\mathbf{t})}{d\, \mathbf{t}} = \sum_{(x,y)\in D} \left(\frac{\partial I_k}{\partial t_k} + \|\mathbf{r}\| Z \nabla I_1^T \mathbf{t}\right) \|\mathbf{r}\| Z \nabla I_1 = 0 \tag{4.21}$$

This results in the following system of equations:

$$\begin{cases} \displaystyle\sum_{(x,y)\in D} \|\mathbf{r}\| Z \frac{\partial I_k}{\partial t_k}\frac{\partial I_1}{\partial x} + \|\mathbf{r}\|^2 Z^2 \left(\frac{\partial I_1}{\partial x} t_x + \frac{\partial I_1}{\partial y} t_2\right)\frac{\partial I_1}{\partial x} &= 0 \\ \displaystyle\sum_{(x,y)\in D} \|\mathbf{r}\| Z \frac{\partial I_k}{\partial t_k}\frac{\partial I_1}{\partial y} + \|\mathbf{r}\|^2 Z^2 \left(\frac{\partial I_1}{\partial x} t_x + \frac{\partial I_1}{\partial y} t_2\right)\frac{\partial I_1}{\partial y} &= 0 \end{cases} \tag{4.22}$$

(4.22) can be written in matrix notation as follows:

$$\mathbf{A}(\mathbf{x})\mathbf{b} = \mathbf{c}(\mathbf{x}) \tag{4.23}$$

With $\mathbf{b} = (t_x, t_y)^T$ and

$$\begin{aligned} \mathbf{A}(\mathbf{x}) &= \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \\ &= \begin{pmatrix} \sum_{(x,y)\in D} \|\mathbf{r}\|^2 Z^2 \left(\frac{\partial I_1}{\partial x}\right)^2 & \sum_{(x,y)\in D} \|\mathbf{r}\|^2 Z^2 \frac{\partial I_1}{\partial x}\frac{\partial I_1}{\partial y} \\ \sum_{(x,y)\in D} \|\mathbf{r}\|^2 Z^2 \frac{\partial I_1}{\partial x}\frac{\partial I_1}{\partial y} & \sum_{(x,y)\in D} \|\mathbf{r}\|^2 Z^2 \left(\frac{\partial I_1}{\partial y}\right)^2 \end{pmatrix} \end{aligned} \tag{4.24}$$

and

$$\mathbf{c}(\mathbf{x}) = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} -\sum_{(x,y)\in D} \|\mathbf{r}\| Z \frac{\partial I_t}{\partial t_k}\frac{\partial I_1}{\partial x} \\ -\sum_{(x,y)\in D} \|\mathbf{r}\| Z \frac{\partial I_t}{\partial t_k}\frac{\partial I_1}{\partial y} \end{pmatrix} \tag{4.25}$$

Finally, t_x and t_y can be obtained by:

$$\begin{aligned} t_x &= \frac{a_{22}c_1 - a_{12}c_2}{a_{11}a_{22} - a_{12}a_{21}} \\ t_y &= \frac{a_{11}c_1 - a_{21}c_2}{a_{11}a_{22} - a_{12}a_{21}} \end{aligned} \tag{4.26}$$

4.3.3 Nonlinear least-squares

In the case of general camera motion, the constraining equations become nonlinear with respect to the motion parameters and thus the system has to be solved iteratively, for example using gradient descent or the popular Levenberg-Marquardt algorithm. Similar steps have to be carried for the nonlinear case in order to obtain the system which will be submit to the nonlinear optimization. Here we only show the final expressions for the gradient descent method. The iterative solution is given by the gradient descent method:

$$\mathbf{y}^{n+1} = \mathbf{y}^n + \gamma \nabla E(\mathbf{y}^n), \quad n \geq 0 \tag{4.27}$$

Where γ is the optimization time step and $\mathbf{y} = (t_x, t_y, t_z, \Omega_x, \Omega_y, \Omega_z)^T$ contains the six motion parameters. The derivatives with respect to E are given by:

$$\frac{dE}{dy_i} = 2 \sum_{(x,y)\in D} \left(\frac{\partial I_k}{\partial t_k} + \nabla I_1^T \mathbf{u}\right) \nabla I_1^T \frac{d\mathbf{u}}{dy_i},\ i = 1 \ldots 6 \tag{4.28}$$

The derivatives of $\mathbf{u}$ with respect to $\mathbf{y}$ are given by the Jacobi matrix:

$$\mathbf{J_u}^T = \begin{pmatrix} \frac{du_1}{dy_1} & \cdots & \frac{du_1}{dy_6} \\ & & \\ \frac{du_2}{dy_1} & \cdots & \frac{du_2}{dy_6} \end{pmatrix}^T \tag{4.29}$$

which evaluates to:

$$\mathbf{J_u}^T = \begin{pmatrix} \frac{r_z\|\mathbf{r}\|Z}{r_z+\|\mathbf{r}\|Zt_z} & 0 \\ 0 & \frac{r_z\|\mathbf{r}\|Z}{r_z+\|\mathbf{r}\|Zt_z} \\ -r_z\|\mathbf{r}\|Z\frac{r_x+\|\mathbf{r}\|Zt_x}{(r_z+\|\mathbf{r}\|Zt_z)^2} & -r_z\|\mathbf{r}\|Z\frac{r_y+\|\mathbf{r}\|Zt_y}{(r_z+\|\mathbf{r}\|Zt_z)^2} \\ -r_yr_z\frac{\Omega_xr_z-\Omega_zr_y}{(\Omega_xr_y-\Omega_yr_x)^2} & -r_z\frac{r_y(\Omega_zr_x-\Omega_xr_z)+r_z(\Omega_xr_y-\Omega_yr_x)}{(\Omega_xr_y-\Omega_yr_x)^2} \\ r_z\frac{r_x(\Omega_yr_z-\Omega_zr_y)+r_z(\Omega_xr_y-\Omega_yr_x)}{(\Omega_xr_y-\Omega_yr_x)^2} & r_xr_z\frac{\Omega_zr_x-\Omega_xr_z}{(\Omega_xr_y-\Omega_yr_x)^2} \\ -r_yr_z & r_xr_z \end{pmatrix} \tag{4.30}$$

4.4 TV-L^1 depth from motion estimation

In the previous sections we have seen how camera movement is related, by projection on the image plane, to the depth of the scene and the motion field $\mathbf{v}$. Now let's formulate the inverse problem for depth estimation: *Given the camera motion parameters* $\mathbf{t}$ *and* $\mathbf{\Omega}$ *and two successive images* I_0 *and* I_1*, calculate the (inverse) depth of the scene.*

Knowing that the motion field $\mathbf{v}$ is an estimator of the optical flow $\mathbf{u}$ we can exploit the image brightness constancy equation in order to relate inverse depth, images and motion parameters linearly:

$$I_1(\mathbf{x}+\mathbf{u_0}) + \nabla I_1(\mathbf{x}+\mathbf{u_0})^T \left(\|\mathbf{r}\| Z(\mathbf{x})\mathbf{t_p} + \mathbf{v_r} - \mathbf{u_0}\right) - I_0(\mathbf{x}) = 0 \tag{4.31}$$

Unfortunately, the above system is ill-posed, i.e. small errors in I_1, its spatial deriviate or in the motion parameters are heavily amplified in the depth map Z. We have to formulate an optimization problem which can be solved robustly for Z. We can define the left side of (4.31) as the image redisual $\rho = \rho(Z) = \rho(I_0, I_1, Z)$ and use its square norm as a measure of error in the least-squares sense. Due to ill-posedness we should add an optimization constraint. Since the nature of images should be well-preserved for the depth map Z, the TV norm of the depth Z can be used in order to impose sparseness while preserving edges. This would result in the classical ROF model for depth.

The ROF model would account for errors in the residual in a quadratic way. But as the used motion field $\mathbf{v}$ as an estimator of the optical flow $\mathbf{u}$ it will be advantageous to account less for errors in the residual. Similar to TV-L^1 optical flow, Bagnato et al. has formulated an error functional involving an L^1 data term and a TV-norm sparseness constraint as follows:

$$Z^* = \arg\min_Z \int_\Omega \Psi(Z, \nabla Z, ...) + \lambda \Phi(\rho)\, d\Omega \tag{4.32}$$

Where ρ is the image residual:

$$\rho(Z) = I_1(\mathbf{x}+\mathbf{u_0}) + \nabla I_1(\mathbf{x}+\mathbf{u_0})^T \left(\|\mathbf{r}\| Z(\mathbf{x})\mathbf{t_p} + \mathbf{v_r} - \mathbf{u_0}\right) - I_0(\mathbf{x}) \tag{4.33}$$

We set $\Psi(Z, \nabla Z, ...) = |\nabla Z|$ such that it corresponds to the TV norm of Z. Similarly, we use $\Phi(\rho) = |\rho|$ to have the L^1 norm of the image residual. The same problems are encountered as in the TV-L^1 optical flow estimation: the error functional is not strictly convex and thus, its associated Euler-Lagrange equation becomes degenerated for $|\rho| \to 0$. Thus we propose the following convex relaxation:

$$Z^* = \arg\min_Z \int_\Omega |\nabla Z| + \frac{1}{2\theta}(V - Z)^2 + \lambda |\rho(V)|\, d\Omega \tag{4.34}$$

V is a close approximation of Z and for $\theta \to 0$ we have $V \to Z$ and thus we obtain the true TV-L^1 model. From (4.34) we have functional splitting, i.e. we must solve for the variables V and Z seperately which can be achieved using an alternative two-step iteration scheme:

1. For fixed Z, solve for V:

$$V^* = \arg\min_V \int_\Omega \frac{1}{2\theta}(V - Z)^2 + \lambda|\rho(V)|\, d\Omega \tag{4.35}$$

2. For fixed V, solve for Z:

$$Z^* = \arg\min_Z \int_\Omega |\nabla Z| + \frac{1}{2\theta}(V - Z)^2\, d\Omega \tag{4.36}$$

By evaluating the Euler-Lagrange equation for (3.19) it is found that minimizing for V results in the following equation, which is not a partial differential equation:

$$\frac{1}{\theta}(V - Z) + \lambda \frac{\rho(V)}{|\rho(V)|}\|\mathbf{r}\|\nabla I_1^t \mathbf{t_p} = 0 \tag{4.37}$$

This leads to:

$$V = Z + \theta\lambda \frac{\rho(V)}{|\rho(V)|}\|\mathbf{r}\|\nabla I_1^t \mathbf{t_p} \tag{4.38}$$

where $\rho(V)$ is given by:

$$\rho(V) = \rho(Z) - (Z - V)\|\mathbf{r}\|\nabla I_1^T \mathbf{t_p} \tag{4.39}$$

Alternatively (4.38) can be interpreted as soft-thresholding with respect to $\rho(Z)$:

$$V = Z + \begin{cases} \theta\lambda\|\mathbf{r}\|\nabla I_1^T \mathbf{t_p} & \text{if} \quad \rho(Z) < -\theta\lambda(\|\mathbf{r}\|\nabla I_1^T \mathbf{t_p})^2 \\ -\theta\lambda\|\mathbf{r}\|\nabla I_1^T \mathbf{t_p} & \text{if} \quad \rho(Z) > \theta\lambda(\|\mathbf{r}\|\nabla I_1^T \mathbf{t_p})^2 \\ \dfrac{\rho(Z)}{\|\mathbf{r}\|\nabla I_1^T \mathbf{t_p}} & \text{if} \quad |\rho(Z)| \leq \theta\lambda(\|\mathbf{r}\|\nabla I_1^T \mathbf{t_p})^2 \end{cases} \tag{4.40}$$

The second optimization is exactly the same as TV-denoising using the original ROF model. For $|\nabla Z| \to 0$ the model is degenerated such that solving the Euler-Lagrange equation by gradient descent is not appropriate. It has already been shown in Chapter 2 that the dual

formulation of the TV-norm resulting in the dual variable $\mathbf{p}$ can be exploited in order to formulate the dual ROF model. The Euler-Lagrange equation of the dual model yields:

$$Z = V - \theta \nabla \cdot \mathbf{p} \tag{4.41}$$

Finally, Chambolle proposed an efficient iterative scheme to solve the dual ROF model:

$$\mathbf{p}^{n+1} = \frac{\mathbf{p}^n + \tau \nabla (\nabla \cdot \mathbf{p}^n - V/\theta)}{1 + \tau \|\nabla (\nabla \cdot \mathbf{p}^n - V/\theta)\|} \tag{4.42}$$

It is shown that this scheme has stable convergence for $\tau < 1/4$. Finally, the same type of multi-scale resolution scheme as for optical flow shall be employed. The difference with respect to optical flow multi-scale is that the sampled version of $\mathbf{r}, \mathbf{t}$ and Z shall not be scaled but only the estimated optical flow $\mathbf{u}$ respectively motion field $\mathbf{v}$.

4.5 Joint depth and ego-motion estimation

In the previous sections it is shown how to first estimate camera ego-motion given the depth map Z and two successive images I_0 and I_1 and in a second step, given the camera motion parameters $\mathbf{t}$ and $\mathbf{\Omega}$ and I_0 and I_1, how compute the depth map Z. It is obvious that the two steps should be somehow combined in order to estimate the depth map Z from the input images only. Based on the proposition of Bagnato et al., we can combine the two steps in the multi-scale iterative scheme. Thus, for the coarsest resolution we start with zero-initial camera motion parameters and we compute an initial rough estimate of the depth map by the robust TV-L^1 algorithm. Then from this depth map, the least-squares method can be used to recompute the camera motion parameters. Finally, the camera motion parameters and the depth map are propagated to the finer resolution where TV-L^1 depth and least-squares ego-motion can be repeated and so on up to full resolution. The complete TV-L^1 joint depth and ego-motion estimation algorithm is shown schematically in Figure 4.5.

4.6 Multiple depth maps from video sequence

If we want to compute multiple depth maps from video sequence, we could use the joint depth and ego-motion estimation algorithm described in the previous section for every pair of successive frames. Fortunatately successive depth maps are very similar, i.e. they contain much redundancy. We can make use uf this redundancy as follows:

- Since the depth map for frame k only changes little with respect to frame $k-1$, we can reduce the number of iterations as well as for the complete algorithm as also for the Chambolle algorithm.

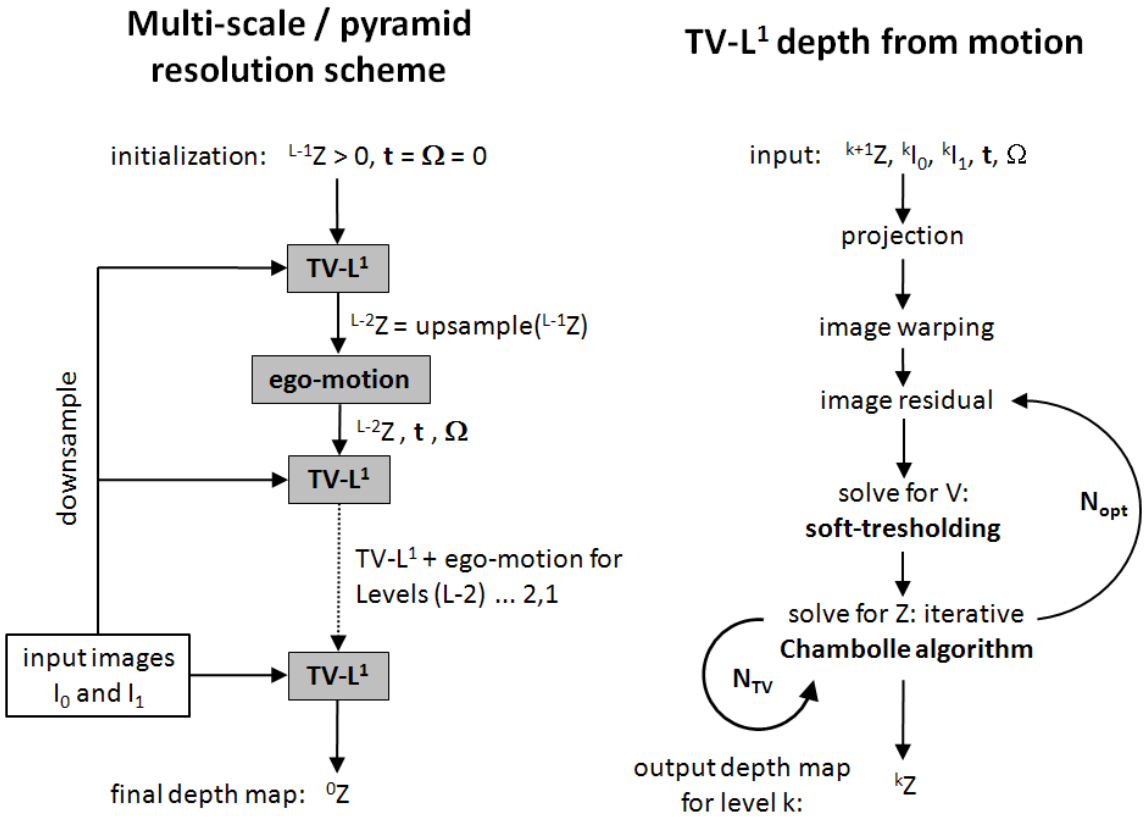


Figure 4.5: The TV-L^1 joint depth and ego-motion estimation algorithm.

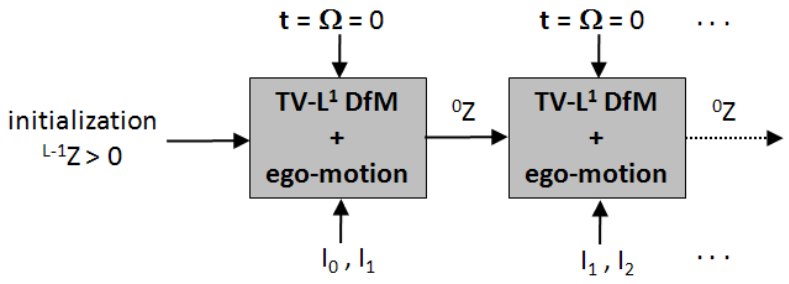


Figure 4.6: The solution is propagated if recovering depth maps from video.

- Since the input images change only little, we don't need the multi-scale approach and we can perform the $TV - L^1$ depth estimation only for the highest resolution.
- Since the calculated depth map is quite accurate at the highest resolution, the ego-motion estimation returns accurate camera parameters.
- Since a big part of the depth map at frame k will be the same as for frame $k-1$, starting the $TV - L^1$ depth estimation with the previous depth map will result in an even more accurate depth map for the recovering parts.

The complete algorithm is depicted in Figure 4.6.

4.7 Results and discussion

In order to have full control in assessing efficiency, accuracy and testing and since we do not account for camera calibration in this project, synthetic images are generated together with ground truth depth maps. We use (a hacked version of) YafRay 0.9.0 as a ray-tracing tool to render long image sequences with different camera trajectories. The XML files to control the ray-tracer camera settings including its position, looking direction and up-vector are generated by a MATLAB script. The scene is the same as used by Bagnato in Ref.X. It depicts a cozy living room with chairs and tables, couches, a pink toy hog, Rembrandt wall pictures etc. The following trajectories respectively sequences are generated (an identifier is associated with it as an abreviation for easy reference):

- X10, Y10, Z10: 3×10 frames each for movement in x,y and respectively z-direction.
- XZ10, YZ10: 2×10 frames each for movement in x+z and y+z direction.
- XYZ10: 10 frames for movement in x+y+z direction.
- XLINEAR160: 160 frames for movement in x direction.
- ZLINEAR100: 100 frames for movement in z direction.

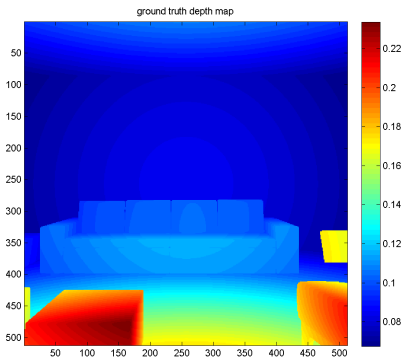
The generated images are of size 512×512. The used camera focal is varied and is shown together with the results for ego-motion estimation in Tables 4.1 and 4.2. Finally, results are shown in Figures 4.7, 4.8 and 4.9 for seperate and for joint depth and ego-motion estimation respectively. For convenience and due to depth estimation "noise" causing very small values in the depth map, we do not show the true depth (the inverse of the depth map). TV-L^1 parameters are maintained the same for all sequences: 5 pyramid levels, 50 global iterations, 4 Chambolle iterations, $\lambda = 0.5$, $\theta = 0.05$.

Sequence	**true t**	**mean(t)**	**std(t)**	**median(t)**
X10	(-1.0, 0.0)	(-1.00, 0.04)	(0.00, 0.01)	(-1.00, 0.04)
Y10	(0.0,-1.0)	(0.00, -1.00)	(0.02, 0.00)	(0.01, -1.00)
X160	(-1.0, 0.0)	(-1.00, 0.04)	(0.00, 0.08)	(-1.00, 0.02)

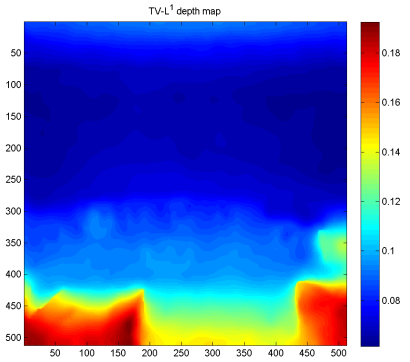
Table 4.1: Ego-motion estimation using **linear-least squares**. Focals are $f = 1.09$ for X10 and Y10 except X160 where it is $f = 0.8$. Unfortunately, with linear-least squares, we can only estimate camera movement in x- and y-direction. We see that the estimation works quite well.

Sequence	**true t**	**mean(t)**	**std(t)**	**median(t)**
X10	(-1.0, 0.0, 0.0)	(-1.00, 0.03, -0.06)	(0.00, 0.04, 0.17)	(-1.00, 0.03, -0.13)
Y10	(0.0,-1.0, 0.0)	(0.01,-1.00, -0.33)	(0.01, 0.00, 0.21)	(0.02, -1.00, -0.40)
Z10	(0.0, 0.0,-1.0)	(0.05, -0.05, -1.00)	(0.01,-0.01, 0.00)	(0.05, -0.05, -1.00)
XZ10	(-1.0, 0.0,-1.0)	(-0.94, -0.24, 0.99)	(0.05, 0.03, 0.03)	(-0.95, -0.23, 1.00)
YZ10	(0.0,-1.0,-1.0)	(0.08, -1.00, 0.11)	(0.07, 0.00, 0.14)	(0.10, -1.00, 0.09)
XYZ10	(-1.0,-1.0,-1.0)	(-0.34, -1.00, -0.29)	(0.05, 0.00, 0.27)	(-0.36, -1.00, -0.45)
Z100	(0.0, 0.0,-1.0)	(0.07, -0.02, -1.00)	(0.05, 0.08, 0.00)	(0.09, -0.03, -1.00)

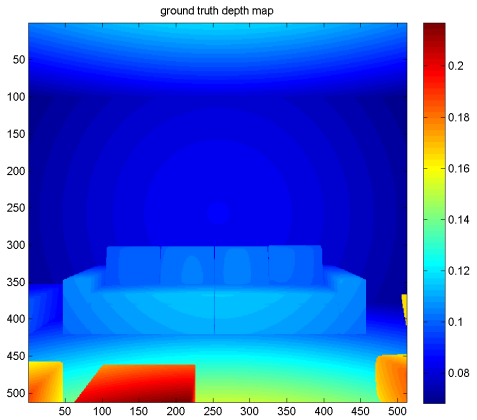
Table 4.2: Ego-motion estimation using **nonlinear-least squares**. Focals are $f = 1.09$ for all sequences except Z100 where it is $f = 0.8$. We use MATLAB's Levenberg-Marquardt solver with standard parameters and initial search point $t^0 = (0, 0, 0)$. There seem to be some outliers for movement in y-direction; concerned are Y10, YZ10, XYZ10.



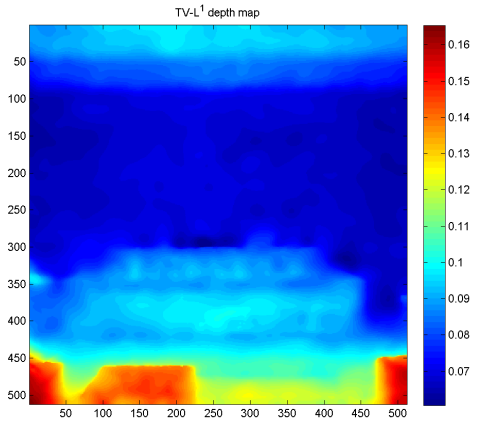
(a) X10: frame 6 — ground truth depth map



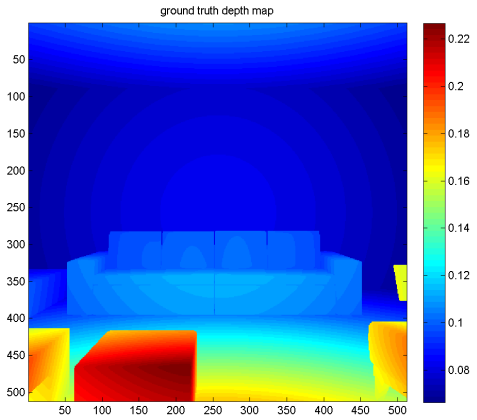
(b) X10: frame 6 — TV-L^1 depth map



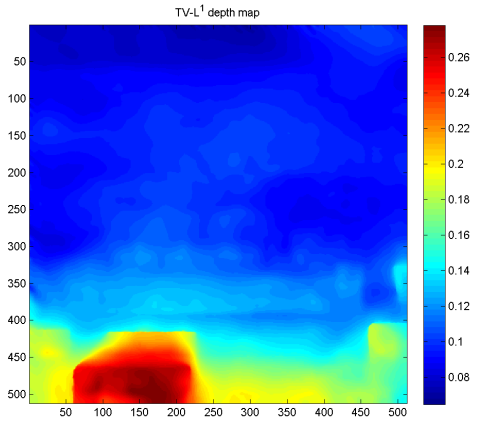
(c) Y10: frame 5 — ground truth depth map



(d) Y10: frame 5 — TV-L^1 depth map

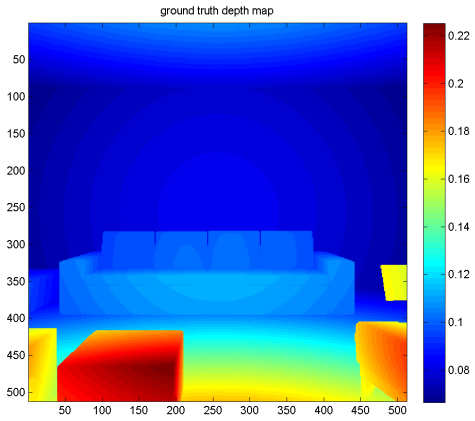


(e) Z10: frame 8 — ground truth depth map

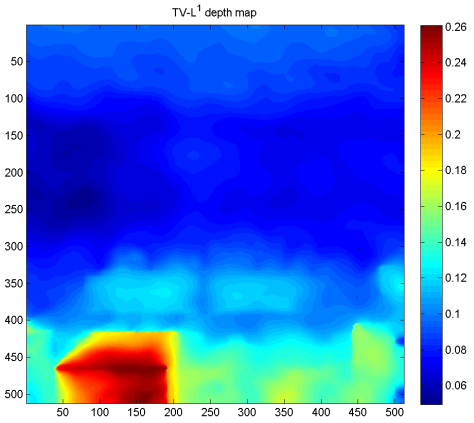


(f) Z10: frame 8 — TV-L^1 depth map

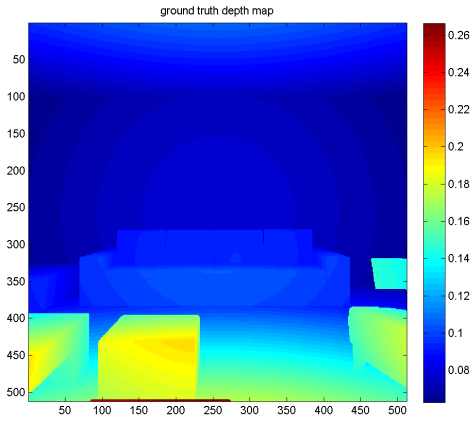
Figure 4.7: Selected frames of the "living room sequence" (without sunshine). Left: ground truth depth maps. Right: recovered depth maps by TV-L^1 depth from motion estimation. Due to the limited number of iterations, the recovered depth maps appear rather smoothley and lack some detail. However, the numerical range of the solution covers well the one of the ground truth data. Given the complexity and large scale of the inverse problem, the recovery works very well—even for movement in z-direction!



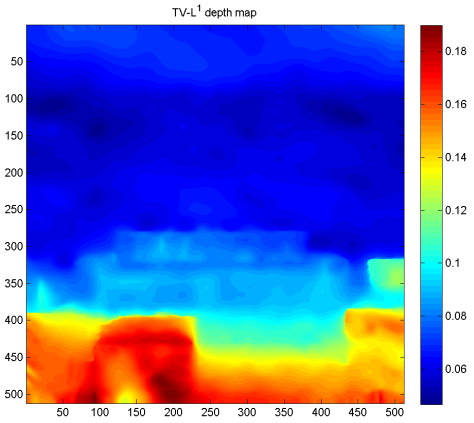
(a) XZ10: frame 8 — ground truth depth map



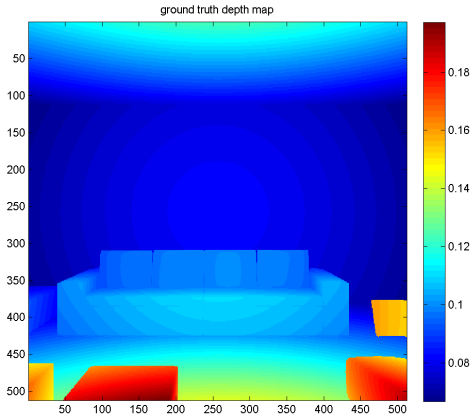
(b) XZ10: frame 8 — TV-L^1 depth map



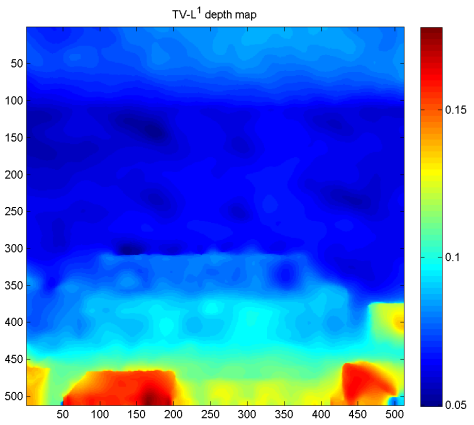
(c) YZ10: frame 1 — ground truth depth map



(d) YZ10: frame 1 — TV-L^1 depth map

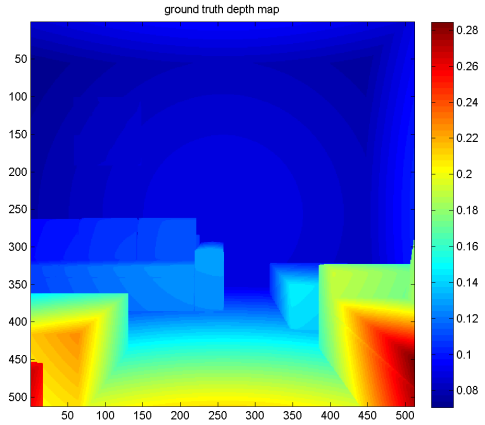


(e) XYZ10: frame 7 — ground truth depth map

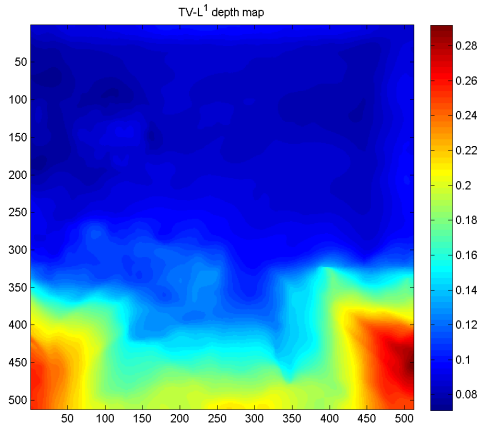


(f) XYZ10: frame 7 — TV-L^1 depth map

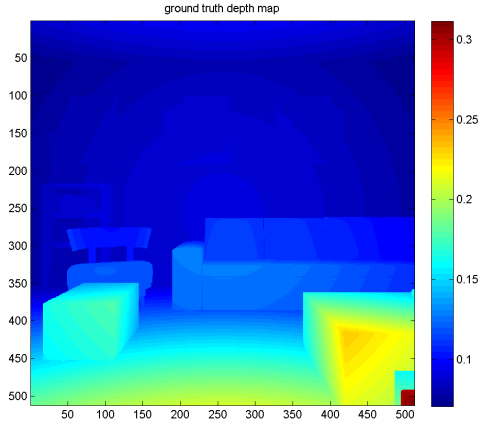
Figure 4.8: Selected frames of the "living room sequence" (without sunshine) and for linear combinations of linear camera movement. Left: ground truth depth maps. Right: recovered depth maps by TV-L^1 depth from motion estimation.



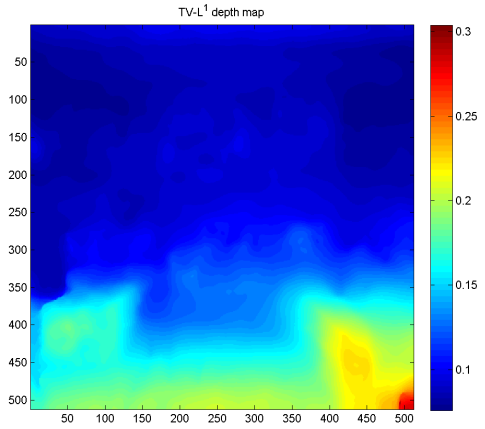
(a) X160: frame 13 — ground truth depth map



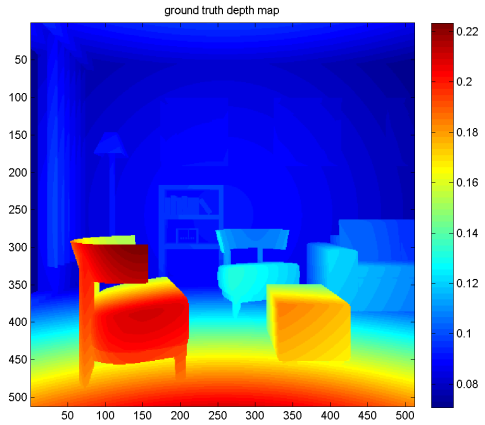
(b) X160: frame 13 — TV-L^1 depth map



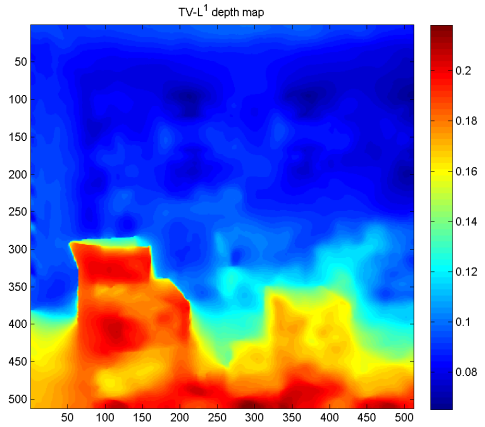
(c) X160: frame 84 — ground truth depth map



(d) X160: frame 84 — TV-L^1 depth map



(e) X160: frame 123 — ground truth depth map



(f) X160: frame 123 — TV-L^1 depth map

Figure 4.9: Selected frames of the "living room sequence" (*with* sunshine). Left: ground truth depth maps. Right: recovered depth maps by TV-L^1 **joint ego-motion and depth estimation**. In the ego-motion we normalize the obtained motion vector and scale it to a meaningful value.

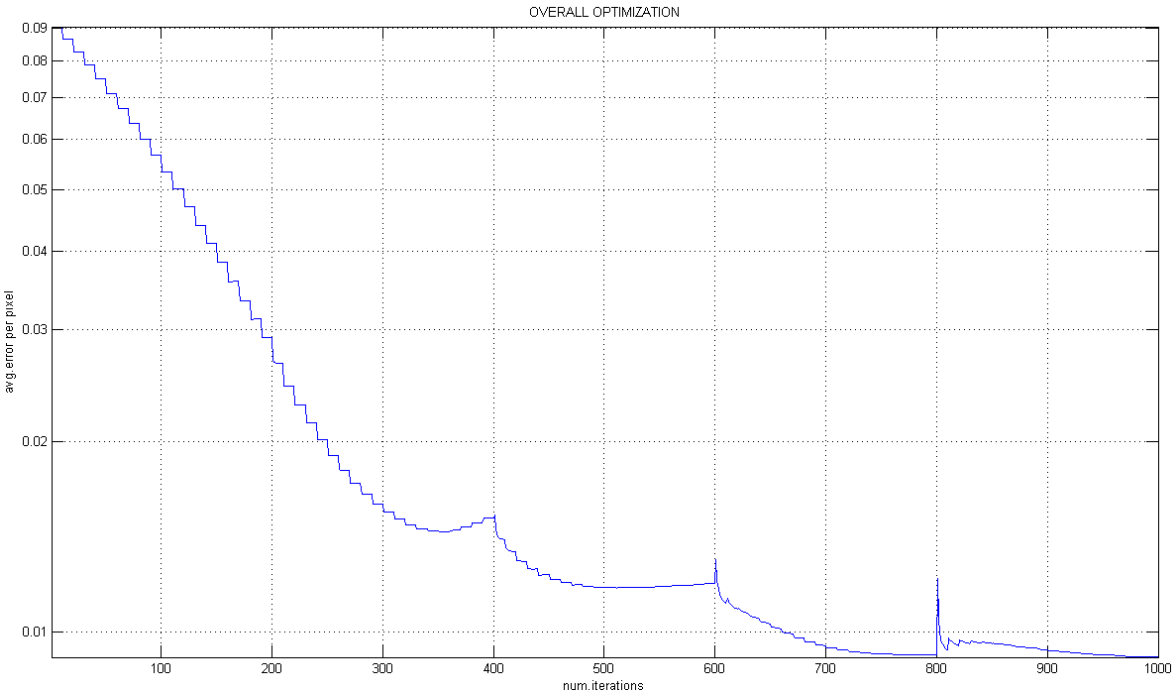


Figure 4.10: The overall distance to the ground truth depth map for X10 (frame 1) shows that the TV-L^1 algorithm is well converging to the solution. Of course, in reality we do not have access to the ground truth data and functional evaluation would be necessary in order to see if the algorithm converges properly. We see further that the convergence becomes very slow as we are approaching the true ground truth depth map. We conclude that, in order to recover the depth map as accurately as possible we would need to decrease θ to 0.1 (or less) and to run about $> 10^5$ iterations.

4.7.1 Discussion

From the results for ego-motion estimation we see that the explicit linear least-sqares estimation works quite well for camera movement in x- and y-direction. Unfortunately the nonlinear least-squares estimation does not work equally well. The outliers for movement in y-directions could come from a little implementation bug (a sign error for example). But stabilizing the motion parameters as briefly presented in Chapter 5 would be welcome anyway. Further we were not able to correctly implement and test ego-motion estimation for camera rotation. This is due to the vector product parametrization of the rotation which is not well adapted to the nonlinearity in the projection model. Euler or unit quaternion rotation will certainly prove to be more appropriate.

The depth from motion recovery works very well for movement in x-, y- and z-direction alone. We see that in general near objects are better recovered than those who are far. This is nothing surprising since it has already been well explained in [15]. In order to solve this

"naturally given" problem, some unconventional methods would have to be employed which e.g. use information of the input images in order to refine the acquired depth maps. Zhang et al. [26] seems to be one of the only which have succuessfully implemented and tested such an approach.

For combined movement, e.g. in x-z-, y-z- and x-y-z-direction there are some errors in the recovery. The structure of the depth map seems to be well found, but the numerical range of the recovery proves to be wrong. This comes from the fact that the projection model has been implemented as a superposition of the movement in x- and y- direction which is linear in depth map and of movement in z-direction which is nonlinear in depth map. In fact, we cannot decompose those movements in the projection model. This will be corrected quickly given the formulas derived in this thesis.

Given the above results, it is clear that the complete joint depth and ego-motion estimation works quite well for the shown sequence and the given movement in x-direction. Of course the algorithm would have to be tested for other camera motion and sequences to better assess efficiency.

Concerning the convergence of the TV-L^1 the same comments as for optical flow have to be made (see also Figure 4.10). Firstly, in order to recover depth maps with more detail, many more iterations would be necessary. Secondly, it might be advantageous to perform functional evaluation instead of fixed number iterations. And finally, employing an alternative optimization scheme such as FISTA might be of great benefit since it proves to have much faster convergence.

Chapter 5

Camera Self-Tracking

5.1 Introduction

As we see from the results in the chapter on structure from motion, depth maps recovered by the joint depth and ego-motion estimation are quite accurate. It is therefore not justified in this thesis to include a part on tracking to stabilize any motion parameters. However for successful application of the algorithm for real video sequences, two more considerations must be made:

1. True camera motion is a stochastic and not a deterministic process.
2. The least-squares ego-motion estimation cannot always yield very accurate values which is in general due to a) small errors in the depth map which are strongly amplified, b) the numerical approximation of the spatial derivatives and c) images or regions of images with low gradient.

Fortunately, TV-L^1 optimization is very robust with respect to errors in the motion parameters. But as we use propagation of the solution in order to reduce computation cost, a very fast motion can result in two very different successive images and thus, ego-motion and depth estimation will fail.

In order to guarantee the successful application of the algorithm in real applications, motion parameters be recovered as accurately as possible for each frame. We should therefore include an algorithm that can provide such accuracy. Eventually this algorithm can also take in to account camera trajectory information, i.e. information from the past—in contrast to what is done in the least-squares ego-motion estimation where only current motion is estimated. We could use some of the following (Brown and Hwang [5]):

1. When noise in the camera trajectory is additive white noise, we can use filtering (or prediction) of the trajectory based on mean- or median filtering resulting in a local minimum least-squares estimation.

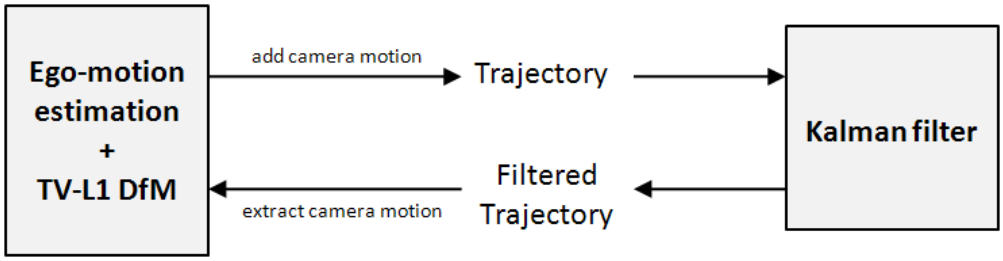


Figure 5.1: Kalman filtering can be used to obtain more accurate camera motion parameters.

2. When noise in the camera trajectory is additive (but not Gaussian explicitely), we can use Wiener filtering (or prediction) on the trajectory if we can afford to keep track of the whole (or an important part of the) trajectory.

3. When noise in the camera trajectory is additive, we can use Kalman filtering (or prediction) on the trajectory resulting in a mimimum mean least-squares estimation.

4. When we want to combine camera motion information from different sources (e.g. trajectory from the ego-motion estimation and from accelerometers or some other IMU), we can use Kalman filtering (or prediction).

5. When we want to obtain an additional camera motion estimation from images only, we can perform feature detection and tracking, generally carried out by an extended Kalman or particle filter and then fuse with or replace the ego-motion estimation.

While the two last propositions seem rather to over-complicate things, proposition 1 and 3 can be implemented and tested quickly. Proposition 2 is not really adapted for real-time processing.

5.2 Tracking and structure from motion

In order to take full benefit of such a tracking or filtering, we must include it appropriately in the structure from motion algorithm. We will initialize the motion parameters used for the TV-L^1 algorithm from the parameters coming from the filter/tracker/predictor; then TV-L^1 is performed as usual followed by regular least-squares ego-motion estimation; finally, the values from the ego-motion estimation will be fed back to the filter/tracker/predictor. The general structure from motion algorithm can then be extended as illustrated in Figure 5.1.

Unfortunately time is short in a Master's project and at this moment nothing has been implemented and tested so far. Hopefully, at least the first and even better also the third proposition will be implemented and tested for the presentation of this Master's thesis.

Chapter 6

GPGPU Implementation for Real-Time Processing

6.1 Introduction

Moore's Law dating back to 1965—Gordon Moore is a co-founder of Intel Corp.—states that the transistor count on integrated circuits doubles approximately every two years [19]. See Figure 6.1 for a concrete illustration. The law has had and is still having important consequences on computer industry and all its dependancies: general purpose computation circuits, **CPUs**, are today containing billions of transistors allowing a vast range of computational demands in extraordinary high-speed. At the same time, the size of transistors is halfed approximately every 18 months—today Intel fabricates transistors with a gate size of 45nm! But there is a physical limit due to quantum physical effects at this size of transistors forcing Intel to re-think the general purpose computation concept. The trend of **parallel computation** in order to process huge amounts of data, especially multimedia such as image and video, has come up such that Intel and AMD have begun to fabricate CPUs with multiple computation cores.

NVIDIA realized in 2000 that their graphics card processors, so-called **GPUs**, are in principle very well adapted to this new trend. However, those application specific processors were in no way programmable and were only destined for one operation: **rendering**. Rendering is the process of calculating two-dimensional images from two- or three-dimensional geometrical description possibly together with textures and lighting information. So NVIDIA began to fabricate processors which allow programming the rendering pipeline via **shading language**.

At the SIGGRAPH 2004 conference it was first introduced how to use this programmable rendering pipeline in order to perform general purpose computation in parallel on the graphics processor. This is now well known as **GPGPU**. Due to conceptual limitations as a result

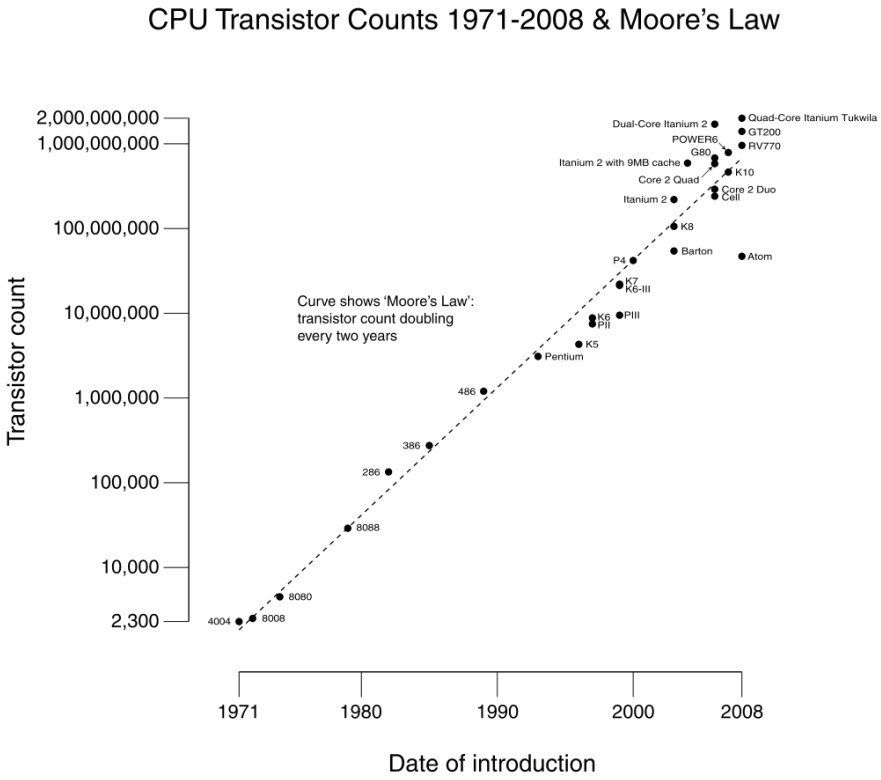


Figure 6.1: Transistor count for some of the most popular processors.

of an application specific hardware implementation of the graphics processor much research has been spent on how to fully exploit the GPU as a mathematical co-processor. [21] gives a good insight on those computational concepts as well as showing the main bottlenecks and limitations of GPGPU computation. NVIDIA realized that there is a big demand in GPGPU and they have begun to fabricate processors which allow for much more flexibility in general-purpose computation by the means of **CUDA**: Compute Unified Device Architecture. Together with an extension for C programming and appropriate software tools, CUDA is a very powerful and inexpensive way to achieve high-performance computing on personnal computers (and it is maybe to come up also for mobile devices).

Before CUDA has been invented, GPGPU has in usually been implemented employing OpenGL and a shading language. The Khronos group which administrates the OpenGL standard, has very recently created another standard: **OpenCL** (Open Compute Language). This standard unifies general purpose (parallel) computation for several platforms including CPU, GPU and DSPs and is strongly based on CUDA. With its first appearance on Mac OS X 10.6 (Snow Leopard) this standard will become very popular and especially important in the domain of high-performance and real-time computing.

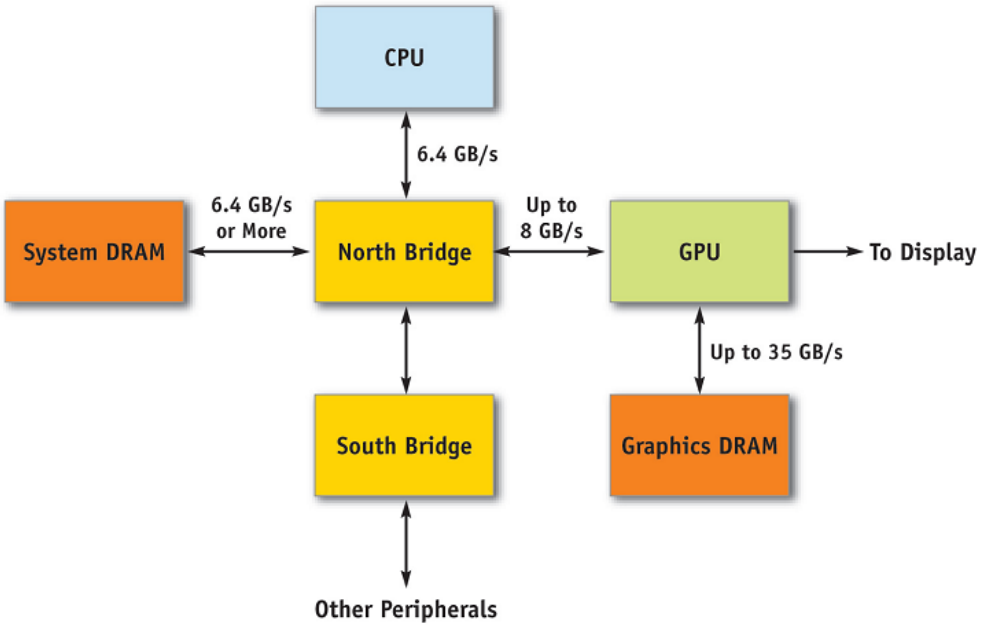


Figure 6.2: Data transfers from CPU to GPU are slow. For comparison: CPU caching reaches up to 300 GByte/s and in for the GPU it is about 100 GByte/s.

Motivated by the GPGPU implementation of the TV-L^1 optical flow by Zach and Pock, in this Master's thesis, the GPGPU TV-L^1 optical flow is reproduced in order to become familiar with the GPGPU concepts and to become an insight on GPU programming. As a final stage of this Master's project, the TV-L^1 structure from motion algorithm shall be implemented on graphics processor. GPGPU based on OpenGL is used. Porting to CUDA would be straightforward and would certainly have an important performance gain on CUDA-enabled devices.

6.2 Porting computational concepts to GPU

6.2.1 Architecture

In order to perform computations on the GPU, we must have a basic understanding of its architecture and of the associated rendering pipeline. In Figure 6.2 the involved part of computer architecture is shown schematically. The figure shows essentially, that when we want to perform computations on the GPU, we must transfer data from CPU or system memory to GPU memory and/or graphics card memory. Due to limited memory transfer rates this step is considered as the main performance bottleneck in GPGPU.

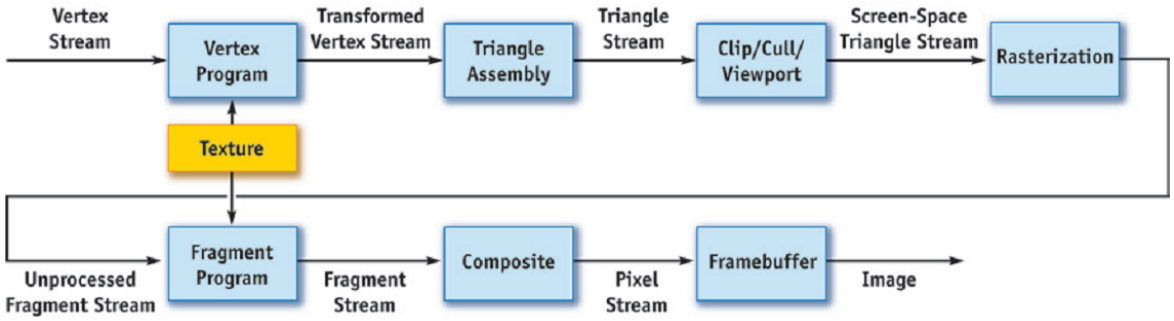


Figure 6.3: The rendering pipeline schematically shown as a stream processor architecture allowing for parallel computation with SIMD or even MIMD.

Now let us look at a simplified schematic of the rendering pipeline which can be seen more generally as a finite state machine. Using a shading language, we can program the vertex and the fragment processing operation. The two most popular shading languages are Cg and GL shading language (GLSL). For our implementation we will use GLSL [17]. The steps shown in Figure 6.3 are briefly summarized as follows:

1. **Vertex and raster stage**: Geometrical primitives in the form of vertices—e.g. four points for a rectangle—are taken as input. Model, camera and viewport transformations are applied such as to obtain the 2D coordinates of those vertices. Then multiple vertex processors calculate the 2D coordinates for per pixel, usually involving interpolation, such that the output of this stage are all coordinates forming the primitive (e.g. the triangle) in device coordinates. This set of points is called the fragment.

2. **Fragment stage**: In the fragment stage, multiple fragment processors calculate the color components per-pixel in the fragment. This color information can be taken from data in the memory, in general referred to as **texture**. In general the fragment processors are able to perform any kind of arithmetic operation as well as some advanced functions such as square-root, trigonometric, exponential functions etc.

3. **Output stage**: Blending and anti-aliasing are performed for smooth output. Finally, the calculated image can be either output to the screen or to texture memory when using the **frame-buffer**. The frame-buffer allows **multiple render targets (MRTs)**, i.e. the computed data can be written to different textures (up to 4 normally).

6.2.2 Exploiting the rendering pipeline for GPGPU

For GPGPU we can exploit the rendering pipeline as follows ([11], [21], [17], [20]):

- **Scattering and MIMD**: in the vertex stage we can have multiple input in the form of vertices. Since the output adress will be calculated in this stage, scattering can be implemented. Scattering means to write the result of any operation to any adress, which in the usual C-array-notation is the following:

  $$\mathrm{T[a[i]]} = func()$$

  If additionally we use some of the vertices not in the sense of geometrical primitives but as lists of 2D or 3D data, we can perform MIMD in the vertex stage—multiple instruction stream, multiple data stream in the form of:

  $$\begin{aligned} \mathrm{T[a[i]]} &= func_1(\mathrm{a}[i+1], \mathrm{a}[i+2]) \\ \mathrm{T[b[i]]} &= func_2(\mathrm{b}[i+1], \mathrm{b}[i+2]) \\ &\dots \end{aligned}$$

- **Gathering and SIMD**: in the fragment stage we have SIMD—single instruction, multiple data, i.e. the same fragment processing operation is applied to every pixel of the fragment. Since the output adress of the pixel is fixed prior to the fragment stage, we can (only) implement gathering, i.e.

  $$\text{pixel} = func(\text{ constants, texture data })$$

- **Multiple output, limited MIMD**: when rendering to the frame-buffer we can usually write to four textures simultaneously such that we are able to implement very limited MIMD, but no scattering(!), in the sense of:

  $$\begin{aligned} \text{pixel}_1 &= func_1(\text{ constants, texture data }) \\ &\dots \\ \text{pixel}_4 &= func_4(\text{ constants, texture data }) \end{aligned}$$

6.2.3 Easy image processing on the GPU

The vertex stage is very powerful for general purpose computation and fully exploiting it together with fragment processing will yield the full computation performance of graphics processors which is up to a factor 1000 of the one of CPUs.

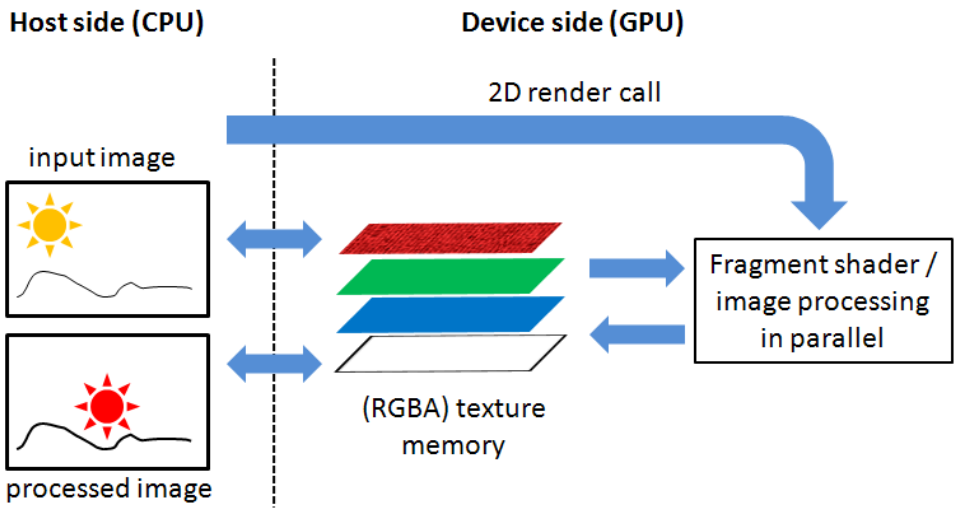


Figure 6.4: How to achieve image processing in an easy way on the GPU.

Unfortunately, the vertex stage is rather complicated to exploit for GPGPU and is not well adapted for image processing. Therefore we will use the fragment processing stage only for our implementations of optical flow and structure from motion. In general, the procedure for image processing using the fragment stage is the following (see Figure 6.4):

1. **Transfer** image data from CPU to GPU texture memory.
2. **Render** a rectangle covering the whole image in an orthogonal projection (no 3D rendering) such that the fragment corresponds to image. Thus, for every pixel of the image the fragment processor will be executed.
3. **Computation** of the output pixel(s) by the fragment processor. The computation to be performed is specified in the **kernel** respectively **fragment shader** which can be specified by the programmer.
4. **Save** the result of the computation in textures.
5. **Repeat** steps 2 bis 5 possibly involving other kernels and/or data in order to perform complex computations which involve intermediate results...
6. **Back transfer** the resulting texture(s) from GPU to CPU or **display** the result directly on the screen.

The kernel programs are the key to GPGPU. They specify the mathematical operation to be performed. Usually, the mathematical operation instruction in a fragment shader is usually very short. Conceiving a mathematical operation for implementation in the fragment processor is sometimes referred to as **SIMD math**. In general, it is achieved by thinking

about what has to be done with a single pixel in function of the other pixels. This is very different to programming on the CPU, where we generally think about how the complete image has to be processed (in rows, in columnes etc.) and then by formulating the iteration and finally the mathetmatical operation. For example, the SIMD math instruction for a horizontal 3-tap mean filter would be the following, where B is the output and A the input texture respectively, b is the output pixel and i, j horizontal respectively vertical texture indices :

$$b = 1/3 * (A[i-1, j] + A[i, j] + A[i+1, j])$$

We must pay attention to the following points:

- Even if we want to process a single picture I with the above fragment processor, we will have to define two(!) textures since the GPU is not able to simultaneously read and write texture memory. We usually initialize $A := I$.
- The above instruction is written for one pixel and must be valable for every pixel in the image. Since the output adress of the result is specified prior to the fragment stage, we *cannot and do not need* specify it. I.e. there is no $B[i, j]$, but only b!
- The access indices $i+k, j+m$ are generally not known inside the kernel for $k \neq 0, m \neq 0$ (only i, j is known). They must be provided to the fragment shader as an argument.

6.2.4 Performance considerations

GPUs are very high-performance computing floating point number processors. The nominal number of floating-point operations per second (FLOPS) that can be performed today is up to 1000 times higher than that of a CPU (see Figure 6.5). But there are several limitations which have the consequence, that implementations on the GPU might only be speeded up a factor 2 to 100. These limitations are basically due to:

- the transfer bottleneck problem as mentioned in Section 6.2.1
- not very careful implementations
- non-local algorithms
- algorithms/problems that are very difficult to parallelize

This leads to the fact that for example general matrix multiplications (GEMM, in constrast to element-wise matrix multiplications) gain only up to a factor of 2-3 when compared to CPU implementation since they are not local, their implementation is difficult with respect to caching, and they are difficult to parallelize. Other classical problems in computer science

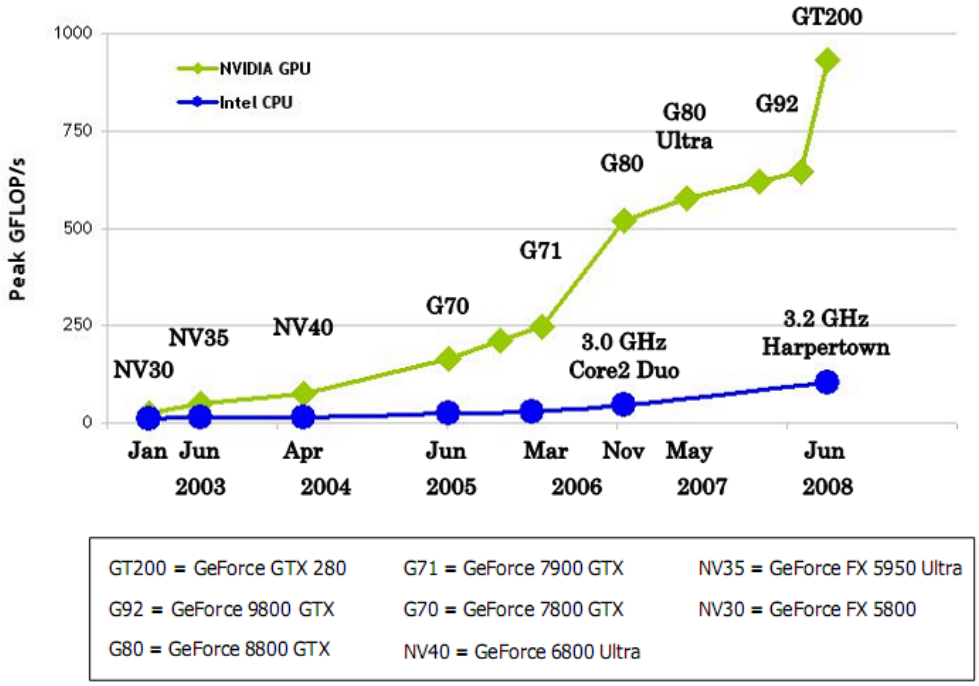


Figure 6.5: The popular NVIDIA GTX280 has almost 500 times higher peak performance than an average Intel CPU.

like sorting are difficult to parallelize as well. There exists a huge palette of research papers on implementations of common algorithms and mathematical operations on the GPU. The main performance gain is optained when using local algorithms and to write code such that the **arithmetic intensity** is the highest possible—arithmetic intensity means the number of floating point operations with respect to (texture) memory accesses.

6.3 Numerical issues

Another common problem for numerical methods are the numerical issues. Natively, GPUs work with single-precision floating point arithmetics, i.e. with 32-bit. In [13] its consequences are well explained. What is important for implementation is to:

- use multiplication instead of divsion where possible
- reduce the number of operations which are not addition or subtraction where possible
- to use correct rounding instead of truncation by *ceil* or $floor$

Furthermore there are the classical numerical issues which do not depend on the used arithmetic precision, but rather on the discretization of a continuous operation. In the TV-L^1

optical flow and structure from motion implementation, these will be the following operations which have a great influence on convergence and accuracy:

- **Sampling and anti-aliasing.** Since we work with a multi-scale solution scheme, images must be down- and upsampled. Here most information gets lost, if not special care is taken. That's why, in the final implementation we use **bi-cubic interpolation**.
- **General image warping.** Here an image shall be warped with floating-point sub-pixel accuracy. Therefore interpolation is employed as well. A bi-linear interpolation is well parallizable and yields results that are accurate enough for a good convergence of the algorithm.
- **Numerical derivation.** Simple forward and backward differences are employed for gradient respectively divergence calculation. Continuous approximation derivation methods are complicated and not suited for (real-time) image processing. But still, more precise derivative approximations could be employed such as central differences, or even k-point stencil central differences with $k = 5, 7, 9$. The difference approximation of the derivative has the advantages that it acts like a non-ideal high-pass filter, i.e. discontinuities do not result in infinite output of the derivative operation. In the GPU implementation we can choose between forward/backward and the usual central differences.

6.4 Optical Flow and Structure from Motion on the GPU

The TV-L^1 Optical Flow and Structure from Motion algorithms can be implemented on GPU the same way as already shown in Chapter 3 in 4. For GPU implementation, the algorithms must be split into several fragment shaders. Unfortunately, such fragment shaders cannot be found readily on the web—every mathematical operation must be implemented manually! The interested reader is referred to the source code and the acompanying README file.

6.4.1 Results and Discussion

6.6 and 6.7 show the optical flow running using GLUT (the OpenGL Utility Toolkit). Structure from Motion is not yet functionnally implemented on the GPU, but of course will be for the presentation. Results can only be given in the form of performance / execution time measurements, refer to Table 6.1

The implemented optical flow algorithm is fast. But it is not as fast as the GPGPU implementations by Zach et al. and it is not as fast as to provide true real-time performance. There are several reasons. First, the fragment shaders are written naivly and limit the performance on the GPU side. The most important bottleneck is the bi-cubic interpolation

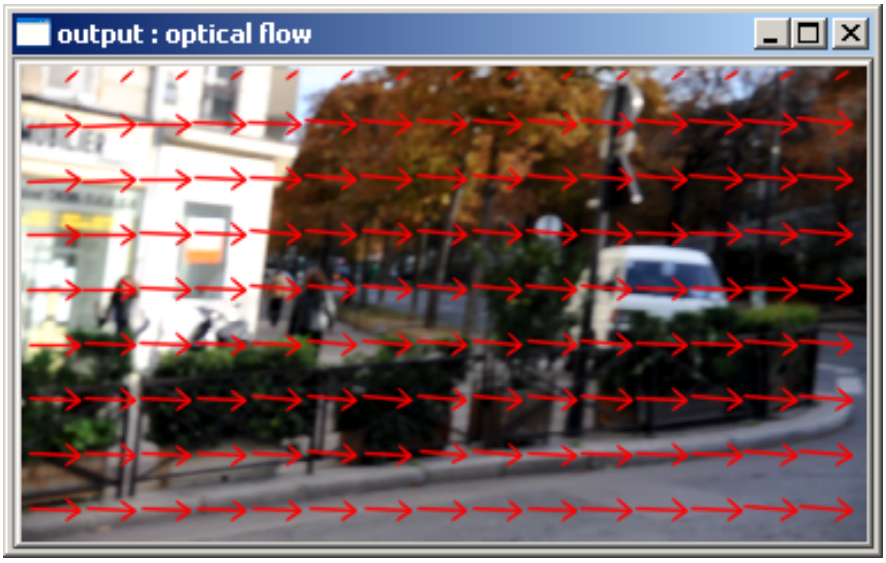


Figure 6.6: Optical flow shown in the form of arrows directly on the input frame..

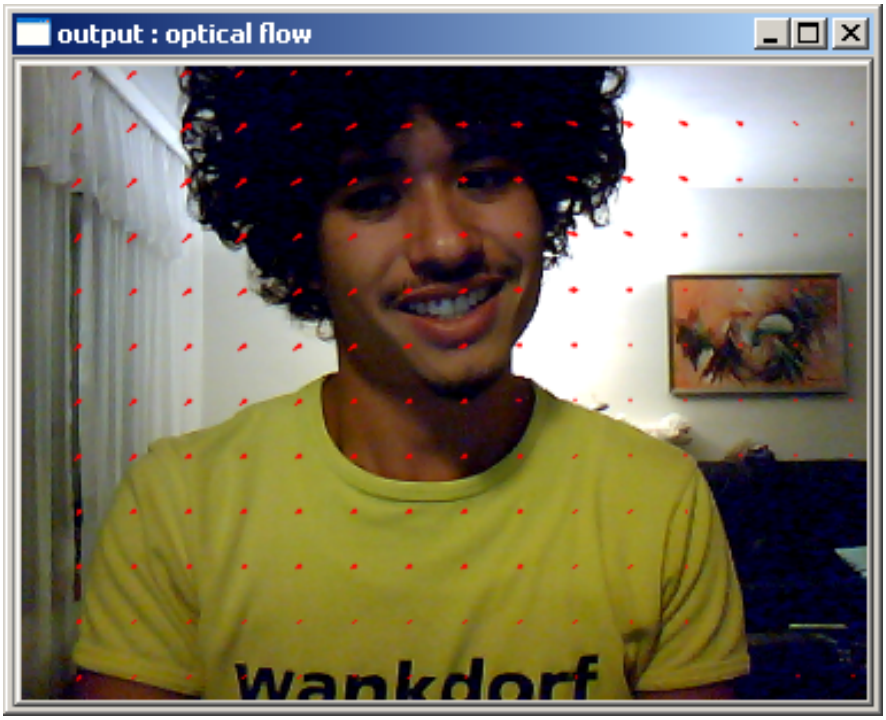


Figure 6.7: The optical flow for a bending head—input taken on the fly from webcam.

Platform	Implementation	Calculation speed (ms)	Total speed (fps)
Windows XP SP3 Intel Core2 Duo T9400 ATI mobility Radeon HD3650	OpenCV+OpenGL	291	1-2
Windows XP SP3 Intel Core2 Duo T9400 ATI mobility Radeon HD3650	OpenGL only	162	3-4
Mac OS X 10.5 What Intel? What ATI?	OpenGL only	187	2-3
Mac OS X 10.5 What Intel? NVIDIA GTX285	OpenGL only	103	5-6

Table 6.1: Performance measurements of the GPGPU optical flow implementation. TV-L^1 parameters are: 5 levels, 10 global iterations, 10 Chambolle iterations.

used for down- and upsampling. Rewriting the fragment shaders carefully might result in a speed up of a factor 2. Second, the texture accesses are not reduced to the possible minimum. This is basically due to a suboptimal repartitioning of the fragment shader tasks. We can reduce the number of used fragment shaders which might give another speed up factor of maybe 20–30%. Third, we do not exploit the full parallel processing capabilities of the GPU since we use one RGBA texture per (gray-scale) image. We can use all of the four RGBA components when downsampling the input images and for calculating its derivatives. Furthermore, we can use at least two of the RGBA components for upsampling the obtained flow images. This might result in another speed gain of 150–250%. Fourth, we do not exploit the full CPU–GPU transfer speed as it is not obvious how to exploit this in conventional GPGPU using OpenGL. However, in [11] it is well explained how to achieve this. Fifth and finally, there is some important latency due to hard disk accesses when calculating the optical flow from image or video sequences (about 200 ms) and a smaller but non-negligeable latency when running the optical flow on webcam input (about 100 ms). When all these points are well respected, a performance of 20–35 fps might be achieved. Consequently, I am looking forward to obtain a much better performance for the presentation.

Chapter 7

Future Prospects and Conclusion

The essential contributions of this Master's thesis are an adaptation of the robust TV-L^1 Structure from Motion for omnidirectional image sequences, as proposed by Bagnato et al., to planar images as well as a portation of this framework for implementation on GPU for real-time processing. To the best of our knowledge, this thesis shows the first time appearance of a real-time dense structure from motion algorithm based on a variational framework. However, due to the nonlinear dependencies on depth and on the camera motion in the projection model, a complete, fully functional Structure from Motion algorithm for planar image sequences could not be implemented and tested—essentially due to a lack of time. Nevertheless, the obtained results are already very promising. Since 99% of images and videos in everyday life are obtained by planar image sensors, the contributions of this thesis are certainly of quite an importance in the fields of image processing and computer vision. This is very encouraging for future completion, correction, extension and improvement.

A nonlinear least-squares ego-motion estimation framework has been proposed and presented with detailled equations. It has to be implemented, tested and corrected. Tracking or other methods have to be derived in order to well handle general stochastic camera motion in the joint depth from motion and ego-motion estimation framework. Such a framework has been briefly described. Given the robustness of the TV-L^1 optimization, tracking is probably straightforward and simple to implement. Camera calibration has to be included in the the Structure from Motion algorithm in order to apply it to images captured by any true camera device. Finally, a complete robust Structure from Motion framework can be implemented and tested on devices such as laptop computers or even on the iPhone.

Together with above mentioned future prospects we should reconsider the optimization equations and better acount for the nonlinearities in the projection model which, in the TV-L^1 optimization, have been linearized so far. As an alternative to TV-L^1 we can try to adapt the depth estimation problem to FISTA—a fast iterative-shrinkage thresholding algorithm to solve linear inverse problems—which is proven to be several magnitudes faster than common

large-scale iterative-shrinkage algorithms. Moreover, a more efficient implementation can be obtained be rewriting the GPU portation with more care or eventually by employing CUDA or OpenCL respectively.

All preliminary work to this thesis, the results, considerations and propositions shown in this thesis and the future prospects lead to one final conclusion: we are close to true dense three-dimensional scene recovery by structure from motion. But how close? Only the future knows.

Bibliography

[1] L. Alvarez, R. Deriche, T. Papadopoulo, and J. Sanchez. Symmetrical dense optical flow estimation with occlusion detection. In *In ECCV*, pages 721–735. Springer, 2002.

[2] L. Bagnato, P. Vandergheynst, and P. Frossard. A Variational Framework for Structure from Motion in Omnidirectional Image Sequences. *IEEE Transactions on Image Processing*, 2009.

[3] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. Technical report, Microsoft Research. Microsoft Corporation., 2009.

[4] T. A. Beck A. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaing Sciences*, 2(1):183–202, 2009.

[5] R. G. Brown and P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering.* Wiley, 1996.

[6] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *8th European Conference on Computer Vision*, volume 4, pages 25–36. Springer-Verlag Berlin, 2004.

[7] A. Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20:89–97, 2004.

[8] E. D'Angelo. Estimation et correction des artefacts géometriques induits par les caméras cmos "rolling shutter". Technical report, Signal Processing Laboratory, Institute of Electrical Engineering. Ecole polytechnique fédérale de Lausanne., 2009.

[9] L. Q.-T. Faugeras O. *The Geometry of Multiple Images. The Laws That Govern the Formation of Multiple Images of a Scene and Some of Their Applications.* The MIT Press, 2001.

[10] E. J. Gibson. Motion parallax as a determinant of perceived depth. *Journal of Experimental Psychology*, 58:40–51, 1959.

[11] D. Goeddeke. Gpgpu tutorials, 2007. URL `http://www.mathematik.tu-dortmund.de`.

[12] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[13] K. E. Hillesland. Gpu floating-point paranoia. In *Proceedings of GP2*, 2004.

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

[15] B. K. P. Horn. *Robot Vision*. The MIT Press, 1986.

[16] S. J. Huston and H. G. Krapp. Visuomotor transformation in the fly gaze stabilization system. *PLoS Biol*, 6(7):e173, 07 2008.

[17] J. Kessenich. *The OpenGL Shading Language*. The Khronos Group, September 2006.

[18] R. G. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-29:1153–1160, 1981.

[19] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8): ?, April 1965.

[20] *CUDA Technical Training. Volume I: Introduction to CUDA programming.* NVIDIA Corporation, 2008.

[21] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005. ISBN 0321335597.

[22] T. Pock. *Fast Total Variation for Computer Vision*. PhD thesis, Graz University of Technology, 2008.

[23] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers. An improved algorithm for tv-l1 optical flow. pages 23–45, 2009.

[24] M. Werlberger, W. Trobin, T. Pock, A. Wedel, D. Cremers, and H. Bischof. Anisotropic huber-l1 optical flow. 2009.

[25] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l1 optical flow. In *DAGM-Symposium*, pages 214–223, 2007.

[26] G. Zhang, J. Jia, T. Wong, and H. Bao. Consistent depth maps recovery from a video sequence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:974–988, 2009.