Depth estimation for stereo image pairs

Dion Boesten

October 15, 2009
Abstract

Philips has developed a multi-view autostereoscopic 3D television which is capable of displaying video in real 3D without the use of external glasses or any other devices. Creating 3D content for these screens is difficult because they require not only the video itself but also the depth map as input. Depth maps are computed from a multi-view recording of the video sequence. To extract the depth map we have to solve the so called stereo correspondence problem. We developed a model, which with the help of some additional human input, can solve it reasonably well. We started by casting the original problem into the form of an energy minimization problem. The energy of a depth map consists of a weighted combination of two terms, a data term which indicates how well the depth maps corresponds with the input images, and an interaction term which is a measure of smoothness. The problem is then converted into the maximization of the probability of a Markov Random Field (MRF), a much used concept in the field of Computer Vision. An iterative procedure to obtain an approximate solution is derived from a Variational Mean Field approximation of the MRF. Several extensions are made to the resulting model in order to make it more suitable for the difficulties that arise in practical situations. These include a multi-scale approach, a gradient based automatic boundary detection algorithm, the possibility to incorporate (manually selected) occluded and boundary pixels in the computation of the depth map and the option of using input from three camera views instead of just two. Our implementation of the model allows for several ways of giving manual input in order to obtain improved depth maps, of which the most important is the manual correction of the automatically computed boundaries.
Contents

1 Introduction .................................................. 4
   1.1 History of 3D films ...................................... 4
   1.2 Depth perception ........................................ 5
   1.3 3D televisions .......................................... 6
       1.3.1 Stereoscopic displays ............................. 6
       1.3.2 Auto-stereoscopic displays ..................... 7
   1.4 Depth maps ............................................. 7

2 Problem description and assumptions ........................ 9
   2.1 Problem overview ...................................... 9
   2.2 Scene projection ....................................... 9
   2.3 Matching pixels ....................................... 10

3 Model overview ........................................... 12
   3.1 Current methods and starting point of our model .... 12
   3.2 General approach ..................................... 13
   3.3 Model components ................................... 13
   3.4 Graph representation .................................. 14

4 Finding the optimum disparity map .......................... 16
   4.1 Solvability of the model ............................... 16
   4.2 Markov Random Field .................................. 16
   4.3 Variational Mean Field equations .................... 17
   4.4 Iterative procedure ................................... 19

5 Model extensions .......................................... 21
   5.1 Multi-scale approach .................................. 21
   5.2 Depth discontinuities .................................. 22
   5.3 Occluded pixels ....................................... 23
   5.4 Neighborhood extension ............................... 24
   5.5 Multiple cameras ..................................... 24
   5.6 Window size ........................................... 24

6 Boundary detection ........................................ 26
   6.1 Automatic versus manual .............................. 26
   6.2 Finding the correct object boundaries using disparity information ........................... 26
   6.3 Automatic edge detection ............................. 27
       6.3.1 Sobel operator .................................... 28
       6.3.2 Boundary extraction ............................ 29
   6.4 Manual input .......................................... 30
7 Results

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Influence of the parameter $\beta$</td>
<td>31</td>
</tr>
<tr>
<td>7.2</td>
<td>Disparity based boundary estimation</td>
<td>32</td>
</tr>
<tr>
<td>7.3</td>
<td>Using three camera views</td>
<td>33</td>
</tr>
<tr>
<td>7.4</td>
<td>Poorly textured regions</td>
<td>33</td>
</tr>
<tr>
<td>7.5</td>
<td>Manually corrected boundaries</td>
<td>34</td>
</tr>
</tbody>
</table>

8 Conclusions and further work 36
List of Figures

1.1 Geometrical example of how our brain is capable of determining the distance of an object. .................................................. 5
1.2 Overview of anaglyph stereo. ................................................ 6
1.3 Overview of an auto-stereoscopic display. ............................... 7
1.4 Example of the Philips 2.5D format. ....................................... 8

2.1 Projection of a 3D point onto two camera planes. .................... 10
3.1 The neighborhood $\mathcal{N}_{ij}$ of the pixel $(i,j)$. ..................... 14
3.2 Graph representation of the model. ....................................... 15
4.1 Illustration of the method to compute a fixed point $x^*$ of the function $f(x) = \sqrt{x}$. 19

5.1 At iteration $k = 2$ the red pixel is influenced only by the blue pixels. .... 21
5.2 The red blocks each consist of 4 pixels but are treated as single pixels. .... 22
5.3 The red pixels are considered boundary pixels. All the edges which touch this boundary are removed. These are colored blue. ............. 23
5.4 A square window of size $w = 2$ is centered around the original pixel and matched onto the right image. ......................... 25

6.1 A simple cube with different colored sides. ............................ 26
6.2 Matching pixels near a left boundary of an object. ..................... 27
6.3 The Sobel operator applied to a color image. The result is shown as a gray-scale image, where brighter pixels have a higher gradient value. ............... 29
6.4 Visual representation of the procedure to determine the final boundary pixels. .. 29

7.1 Computed depth maps for increasing values of $\beta$. ....................... 31
7.2 Results from running the algorithm in both directions, matching the left image onto the right and vice versa. ......................... 32
7.3 The end result of the combination of both disparity maps. ............... 32
7.4 Example showing the added benefit of having a third view available. ........ 33
7.5 Example of a scene that has large areas with poor texture. ............... 33
7.6 Example scene to illustrate the necessity of boundary detection. ........... 34
7.7 Depth map results using automatically computed object boundaries. .......... 35
7.8 Depth map result using manually corrected boundaries. .................. 35
Chapter 1

Introduction

This report describes the work performed by the first author during his internship at Philips Research in Eindhoven. The internship was done as a final project for the Master program Discrete Mathematics & Applications (DMA) at the Technical University Eindhoven (TU/e). Supervision duties were carried out by Patrick Vandewalle (Philips Research), Cor Hurkens and Rudi Pendavingh (TU/e, Department of Mathematics and Computing Science).

The internship focused on the development of a good depth estimation algorithm that can be used for the content creation for the Philips branch of autostereoscopic 3D televisions. These televisions are capable of displaying images (and video) in 3D but they require so called depth maps in order to function. These depth maps are a means of indicating the depth value for each pixel in the image. While it is possible to compute a depth map for a single image, things become much easier when the same image has been shot with multiple cameras from slightly different viewpoints. We call them stereo images in the case of 2 cameras, and multi-view stereo when there are more than 2. The algorithm that we developed does precisely this, i.e. compute a depth map for a set of stereo (or multi-view stereo) images. The main objective was to create accurate depth maps which are smooth (contain little noise) and have well defined object boundaries (places where the depth changes discontinuously) that match the scene.

The outline of this report is as follows. In the rest of this chapter we will give a short history on 3D television and show how humans perceive depth. We will also describe how the Philips 3D televisions work and why the depth maps are so important. In chapter 2 a more detailed description of the exact problem that our algorithm tries to solve is given. A first direction on how the problem could be solved is also sketched by analyzing how 3D points in a real-world scene are projected onto two or more cameras. We present the approach that our model takes in chapter 3. Chapter 4 deals with how to solve this model. Because it is not solvable in an exact way we have to resort to an approximation scheme. In chapter 5 we discuss several types of extensions that can be made to improve the model and obtain better results. The object boundary detection part of our model is described in chapter 6. Here also the possible ways are summarized in which manual input can be given in our implementation of the model. We end with our conclusions and delineate paths for further research related with our work.

1.1 History of 3D films

Even though it still sounds somewhat esoteric, 3D imaging — i.e., images that seem 3-dimensional when you look at them — has been around for almost as long as common 2D images — the first patent related to 3D appears already in the 19th century. In fact, four major waves (or surges) of interest in 3D films have occurred.
In the beginning of the 20th century, a few projections were made using anaglyph glasses with one red and one blue lens. The Great Depression, first, and the Second World War, later, did not allow the necessary technology to fully develop and interest in 3D waned.

The 1950s saw interest in 3D resurge. Pressed by the competition of television, movie studios turned their hopes on new technologies to bring people back to cinemas. Several high-profile studios, such as Columbia Pictures, Warner Bros. and Walt Disney, came out with 3D versions of some of their films. They even casted well-known figures, like The Three Stooges, to appear in their 3D films. Sadly, there were several technical and cost problems that affected 3D projection and after a few years studios turned to other ways to add additional value to films with better returns.

After some of the earlier technical problems had been solved, there was a new wave of excitement about 3D films in the early 80s. Again, the studios jumped on the bandwagon and produced some high-profile titles, notably Jaws 3D. Again, as well, the interest rapidly faded.

The fourth wave of interest is today. Arguably pressed by the threat of digital downloads, film studios have seen again in 3D the path to their economical future. This time though, film directors seem to have picked an interest on 3D as way to improve the artistic merits of films. James Cameron, director of the publicly acclaimed film “Titanic”, has become one of the loudest voices in defense of a future in 3D.

Thus, everything seems now set to make 3D the standard and not the exception in the film industry. High-quality video content enhanced with 3D, is on its way. For Philips the aim is to bring that experience from the cinema also to the living-room.

1.2 Depth perception

One of the most important ways in which humans are capable of seeing in 3D is by stereopsis (also called retinal disparity). It is a simple process that is best illustrated by the 2D figure below.

![Figure 1.1: Geometrical example of how our brain is capable of determining the distance of an object.](image)
to connecting the point $P$ with the two lens centers $C_1$ and $C_2$ by straight lines. The intersection with the two retinas gives us the two projected points $P_1$ and $P_2$. As can be seen from the drawing the relative locations of these points with respect to the origins of their retina differ. If we define $x_1 := P_1 - O_1$ and $x_2 := P_2 - O_2$ as the coordinates of the two projected points we call $x_2 - x_1$ the retinal disparity. With some simple geometry we can deduce that the perpendicular distance of the point $P$, i.e. what we could call the depth of $P$, is inversely proportional to the retinal disparity. As $P$ moves to infinite distance the retinal disparity becomes zero, while for points close to us the disparity becomes very large.

It is precisely this property that our brain uses in determining the depth of objects and thus giving us 3D perception. It is also the reason why it is very hard to decide how far away things are when we are looking with one of our eyes closed. When we look at a scene our brain is constantly matching objects seen by our two eyes and calculating their disparity in order to give us a sense of depth. All this happens in real-time, quite an achievement!

## 1.3 3D televisions

All 3D televisions make use of the same principle described in the previous section by providing our two eyes with two slightly different views of the same scene. However, the ways in which they achieve this, differ quite a lot. We can make a rough distinction into two categories. The stereoscopic displays require the viewer to wear some sort of glasses, while the auto-stereoscopic displays do not. The Philips line of 3D TV’s is of the auto-stereoscopic kind.

### 1.3.1 Stereoscopic displays

All stereoscopic displays function in the same way. They come with a pair of glasses that in some way or the other block part of the light from the screen depending on whether it should go to the left or right eye. There are several mechanisms to achieve this.

The first and oldest is by using the color property of light. Anaglyph glasses are glasses which allow only light of a certain color to pass through. In practical applications this is usually red and blue.

The two images which are meant for the left and right eye are encoded in one of two colors and put together in a single color image. When looking at such an image through the anaglyph glasses each eye sees only the colored light from the image meant for that eye. This gives the desired 3D effect. An example image of anaglyph stereo is given in figure 1.2.
Shutter based glasses work by alternating between blocking light from the left and right eye. The screen switches quickly between the left and right image. The glasses are synchronized with the screen in such a way that at each moment either the left eye sees nothing and the right eye sees the right image or vice versa. The result is that each eye sees the correct image about half the time and the other half sees darkness.

Polarized glasses depend on the polarization property of light. Only light with the correct polarization is able to pass the lens. The screen modulates the polarization of each pixel such that light from a pixel of the left image is able to pass through the left lens of the glasses but is blocked by the right lens, and the other way around for pixels of the right image. The result is again the correct separation of the two images, which give the 3D effect.

1.3.2 Auto-stereoscopic displays

Auto-stereoscopic displays dispense with the glasses altogether and provide the 3D experience in a different way. While there are several mechanisms by which this can be done we will only describe the most relevant one. The Philips 3D TV’s make use of a sheet of lenticular lenses that are placed in front of the display (see figure 1.3a) [1].

Each pixel on the screen consists of 9 sub-pixels of which light is redirected by the lenses in different angles. This means every direction receives light from a certain subset of all sub-pixels on the screen. Clever manipulation of which sub-pixels have to be combined in order to form an image make it possible to create 9 different views at slightly different angles. When we look at the screen our two eyes see two different views and the 3D effect is achieved. Because light from each pixel is also able to pass through neighboring lenses the 9 views are repeated throughout space and we obtain a distribution of views like in figure 1.3b. This provides higher flexibility in the position of the viewer. Now the 3D effect is visible from almost every angle instead of just a small area right in front of the screen.

1.4 Depth maps

The multi-view autostereoscopic display has many advantages over the stereoscopic displays but it has one major disadvantage. In order to function properly and render all the 9 views to produce the 3D effect it requires the depth map of the scene. To create 3D content for stereoscopic displays we can “simply” record a movie with two separate cameras at positions where our eyes would normally be and then later use these two views to create the 3D effect. For an auto-stereoscopic display this doesn’t work as easily. We can not simply make the 9 required views out of the 2 views without first determining the 3D information of the entire scene. We need to know the depth value of each pixel to be able to compute alternate viewpoints.
Depth maps of a scene are usually represented by a gray-scale image. The intensity of a pixel indicates its depth. The further away a pixel is, the lower the intensity. Pixels that are very close are drawn very bright, while pixels at infinity are completely black. Philips uses a special format in which an image and its corresponding depth map are combined into a single new image. This is called the 2.5D format of which an example is shown in figure 1.4.

![Figure 1.4: Example of the Philips 2.5D format.](image)

The Philips 3D TV takes this format as input while producing the 9 views as output on the screen such that a viewer sees the image in 3D from any angle. The details of this process are outside the scope of this report. Our task lies in the computation of the depth maps from the two recorded views of a scene. In the next chapter we will start with a more detailed description of the problem.
Chapter 2

Problem description and assumptions

2.1 Problem overview

The basic problem that we want to solve is as follows. We are given a set of two images of equal size, one representing the left view of a certain scene and the other the right view (the cameras are next to each other). The left image is denoted by \( I_L \) and is a function of the 2-dimensional grid \( G \) of pixels to the color space \( C \). The right image is a similar function and denoted by \( I_R \). The size of the grid is \( m \times n \), so both images are \( m \) pixels wide and \( n \) pixels high. The color space is the familiar 3 dimensional RGB space, where each color is represented by its red, green and blue values. In other words we can define:

\[
G = [1, \ldots, m] \times [1, \ldots, n] \quad (2.1)
\]
\[
C = [0, \ldots, 255]^3 \quad (2.2)
\]

Our task is to find a correct estimate of the depth value of each pixel in the left image. To explain what we mean with depth we will start by looking at how a point in the scene ends up on both images. We model the two cameras by pinhole cameras, which means that the images are the projections of the scene onto two camera planes. We make the important assumption that the cameras are identical, oriented in the same way and are separated by a small horizontal distance. It is not entirely realistic to assume this, but it is a good approximation and it will make things a lot easier. Note that there are algorithms which can correct the images in case the cameras do not fully satisfy our assumptions. This process is called image rectification but we will not consider these issues here.

2.2 Scene projection

Assume we have a scene point \( z \in \mathbb{R}^3 \) with coordinates \((z_1, z_2, z_3)\). The two camera centers are positioned at locations \( c_1 = (-a, 0, 0) \) and \( c_2 = (a, 0, 0) \) and their projection planes are situated at a distance \( f \) like in figure 2.1. The points \( x, y \in \mathbb{R}^2 \) are the projections of the point \( z \) onto the two camera planes.

This means we can write the point \( z \) in two different ways:

\[
z = c_1 + \lambda(x_1, x_2, f)
\]
\[
z = c_2 + \mu(y_1, y_2, f)
\]
Figure 2.1: Projection of a 3D point onto two camera planes.

where \( \lambda, \mu \in \mathbb{R} \).

From these equations we can deduce:

\[
\begin{align*}
    z_3 &= \lambda f = \mu f \quad \rightarrow \quad \lambda = \mu = \frac{z_3}{f} \\
    x_2 &= y_2 = \frac{z_2}{\lambda} \\
    x_1 &= \frac{z_1 + a}{\lambda} \\
    y_1 &= \frac{z_1 - a}{\lambda}
\end{align*}
\]

From the last two lines we can derive that \( x_1 - y_1 = \frac{2a}{\lambda} \) and hence it follows that:

\[
z_3 = \frac{2af}{x_1 - y_1} \tag{2.3}
\]

We see that the third coordinate \( z_3 \) of the point \( z \), which we will call the depth, is inversely proportional to the difference in the horizontal coordinates of the projections \( x_1 - y_1 \). This last quantity is called the disparity and it is the key to determining the correct depth values of pixels. Once we know the disparity of a pixel the depth value is easily computed through equation (2.3). Our goal is then to assign the correct disparity value to each pixel in the left image.

2.3 Matching pixels

The procedure for a single pixel \( p \) in the left image with coordinates \((i, j) \in G\) is simple. We need to find the matching pixel (the one representing the same scene point) in the right image and determine how much it has shifted. The above derivation shows us that the vertical coordinates \( x_2 \) and \( y_2 \) of the two projections are equal. This means that we can restrict the search space for the right image pixel to the horizontal line \((i', j)\) with \(0 \leq i' \leq i\). Here necessarily \( i' \leq i \) because by going from the left view to the right view a scene point will always shift to the left. Once we find the corresponding pixel we assign the disparity \( i - i' \) to the pixel \( p \).
Continuing in this way for each pixel in the left image we end up with a function $d : G \rightarrow N$ which is called the disparity map. A particular value of the disparity map at an image location $(i, j)$ is denoted by $d(i, j)$ or the shorter notation $d_{ij}$.

The main difficulty, and the way in which nearly all depth estimation algorithms work, is to determine what exactly are the matching pixels in the two images.
Chapter 3

Model overview

3.1 Current methods and starting point of our model

The problem that we are trying to solve is called the stereo correspondence problem. It is a well known problem that has received much attention over the years. An overview of the algorithms that have been developed for it is given in [7, 11]. In the first article the authors compare the workings and results of several different algorithms. They also keep track of the results of new algorithms on their website [6].

Most of the algorithms that try to solve the stereo problem function in a very similar way. They start with computing a matching cost function for each pixel in the image. These functions are then used to obtain a final matching cost for each pixel by some kind of aggregation procedure. This can be summing or averaging over a region of neighboring pixels for instance. The disparity map is computed from these matching costs either by a local or global method. Local methods apply a winner-takes-all (WTA) approach, simply choose at each pixel the disparity associated with the minimum cost value. Most of the work of local methods is in computing good matching functions and the aggregation procedure. The optimization part is relatively simple. In contrast, global methods perform almost all of their work in the optimization part and often skip the aggregation phase. They are often formulated in an energy-minimization framework. The objective is to compute the disparity map that minimizes the energy:

\[ E(d) = E_{\text{data}}(d) + \beta E_{\text{smooth}}(d) \] (3.1)

The data term \( E_{\text{data}}(d) \) measure how well the disparity map corresponds with the input images and usually consists of a weighted sum of the cost functions of all pixels. The smoothness term encodes the smoothness assumptions made by the algorithm. Very often, to simplify the computations, the term \( E_{\text{smooth}}(d) \) is chosen to be a sum of contributions \( \rho(d_{ij} - d_{kl}) \) from all sets of neighboring pixels \((i,j)\) and \((k,l)\). The function \( \rho \) is a monotonically increasing function of the disparity difference, such that neighboring pixels are promoted to have similar disparity values. The parameter \( \beta \) determines the relative importance of both components.

Our model will be of the global kind and is based on ideas from the article [4]. In this article the authors propose a cooperative method that tries to estimate both disparity and object boundaries at the same time. We started our research by implementing their model. After getting some initial results we were not quite happy with the object boundary estimation part of the algorithm so we decided to replace it. This was also done in consideration of the fact that our algorithm is allowed to use a small amount of manual input and the boundary detection seemed to us the most suitable candidate for this.
3.2 General approach

The general idea of our model is to cast the problem of finding the best disparity map into a minimization problem over $N^{m \times n}$ (the space of all possible disparity assignments). We use a function $E(d)$ that gives a score to every possible disparity map $d$. This score, which we will call the total energy of the disparity map, is an element of $\mathbb{R}$. It tells us how good the map is in an overall sense. The analogy with an energy minimization problem from physics is commonly used as it is a good aid in the visualization and understanding of the problem.

The energy function $E(d)$ is a weighted sum of two separate energies. One component, which we will call the data energy, is based on the information from the two images. It ensures that the depth map corresponds correctly to the scene. The other component, which we will call the interaction energy, does not depend on the images at all. It is a measure of smoothness, roughly indicating how much noise the disparity map contains. In summary we can write:

$$E(d) = E_{\text{data}}(d) + \beta \cdot E_{\text{interaction}}(d)$$

(3.2)

where $\beta$ is the parameter that determines the relative weight of each component. The details will be explained in the rest of this chapter.

The disparity map that our algorithm outputs will be the one which minimizes the total energy $E(d)$. How we do this minimization is explained in chapter 4.

3.3 Model components

The first thing that we do is to restrict the possible values of the disparity at each point. This is done to lower the computation time needed by our model. It is not a severe restriction since it is very uncommon to have large disparity values (objects would have to be very close to the camera). So we assume $d$ to take values in the set $D = [0, \ldots, d_{\text{max}}]$.

We introduce an energy function $U_{ij}(d_{ij})$ for each element $(i,j)$ of the grid $G$. This function maps the possible disparity values $d_{ij} \in D$ at a location $(i,j)$ in the left image to positive real numbers. It should be interpreted as a score (a lower energy is better) that tells us how well the pixel $p = (i,j)$ in the left image corresponds to the shifted pixel $q = (i - d_{ij}, j)$ in the right image.

We define the functions $U_{ij} : D \rightarrow \mathbb{R}^+$ as:

$$U_{ij}(d_{ij}) = \min\{\lambda_1 |I_L(i,j) - I_R(i - d_{ij}, j)|, T_1\}$$

(3.3)

where the norm function $| \cdot | : C \rightarrow \mathbb{R}$ can be chosen as desired. We took it to be the standard Euclidean norm. The parameters $\lambda_1, T_1$ can be varied but are the same for each $U_{ij}$. The global data energy $E_{\text{data}}(d)$ is now simply the sum of these separate pixel energies:

$$E_{\text{data}}(d) = \sum_{(i,j) \in G} U_{ij}(d_{ij})$$

(3.4)

If we minimize $E_{\text{data}}(d)$, which we can do separately for each pixel since the functions $U_{ij}(d_{ij})$ are independent, we get a disparity map that already shows the basic features of the image but still has a lot of noise. We would like the disparity map to be smooth. It shouldn’t vary a lot over a small region, unless we are at the boundary of an object.
To incorporate this desired behavior into the model we introduce an interaction energy \( V : D^2 \rightarrow \mathbb{R}^+ \) for each set of neighboring pixels. A pixel has 8 neighbors unless it is somewhere at the border of the image. The set of neighbors of pixel \((i,j)\) is denoted by \( N_{ij} \) and displayed in figure 3.1 below.

![Figure 3.1: The neighborhood \( N_{ij} \) of the pixel \((i,j)\).](image)

The function \( V \) is defined as:

\[
V(d_1, d_2) = \min\{\lambda_2|d_1 - d_2|, T_2\}
\]  

where \( \lambda_2, T_2 \) are parameters that can be varied but are the same for each set of neighboring pixels. It is clear that by including \( V \) we promote the disparity values of neighboring pixels to be similar in the form of a lower interaction energy. Note that \( V \) is a symmetric function. The global interaction energy \( E_{\text{interaction}}(d) \) is the sum of all the individual interaction energies between each two neighboring pixels:

\[
E_{\text{interaction}}(d) = \sum_{(i,j) \in G} \sum_{(k,l) \in N_{ij}} V(d_{ij}, d_{kl})
\]  

The optimum disparity map \( d^* \) is computed as the one which minimizes the total energy \( E(d) \) from equation (3.2):

\[
E(d) = \sum_{(i,j) \in G} \left\{ U_{ij}(d_{ij}) + \beta \sum_{(k,l) \in N_{ij}} V(d_{ij}, d_{kl}) \right\}
\]  

### 3.4 Graph representation

A good way to represent the model that we have described so far is by a graph \( G_{\text{disparity}} = (VG, EG) \) where \( VG \) is the set of vertices and \( EG \) the set of edges of the graph. If we take \( VG \) to be the individual pixels we can draw an edge between each two neighboring pixels that contribute to the total interaction energy. The graph will look like this:

We can associate a label with each vertex, namely the disparity value for the corresponding pixel. It is then possible to write the total energy from equation 3.2 as:

\[
E(d) = \sum_{v \in VG} U_v(d_v) + \beta \sum_{uv \in EG} V(d_u, d_v)
\]  

where \( d_v \) indicates the disparity value for the vertex \( v \), and \( U_v \) is the data energy function for the pixel represented by \( v \).
The problem can then be reformulated as finding an optimum labeling $d^*$ of the vertices such that the total energy is minimized. This way of looking at the problem will prove helpful when discussing extensions to the model.
Chapter 4

Finding the optimum disparity map

4.1 Solvability of the model

Solving the minimization problem of the energy function $E(d)$ in an exact way is infeasible due to a number of reasons. The amount of variables involved is very large ($m \times n$ is in the order of 1 million pixels) and they are linked together through the interaction energies. Furthermore the data energy functions for each pixel are very irregular and can not be expressed in simple formulas. We have to resort to some kind of approximation scheme to find a solution to the minimization of $E(d)$. There are several of these to be found in the literature [7], but in our case we choose to represent the model by a Markov Random Field (MRF) [9, 8]. Then we use a so called Variational Mean Field (VMF) approximation to obtain a solution to the MRF, which is also a solution to our original minimization problem. The details will be explained next.

4.2 Markov Random Field

The key step is to translate the energy value $E(d)$ for every possible assignment of $d = (d_{11}, d_{12}, \ldots, d_{mn})$ into a probability value. This is done as follows:

$$P[d] = \frac{1}{Z} e^{-E(d)} \quad (4.1)$$

where $Z$ is the normalizing factor which turns it into a probability measure:

$$Z = \sum_d e^{-E(d)}$$

Here the sum over $d$ represents the $m \times n$ dimensional sum $\sum_{d_{11}} \sum_{d_{12}} \cdots \sum_{d_{mn}}$ where each $d_{ij}$ takes values in the set $D$. The minimization of the energy $E$ now corresponds exactly to the maximization of the probability $P$. We have translated the original problem into a different problem. The model that we obtain is commonly called a Markov Random Field (MRF), a much used construct in the field of Computer Vision [9].

Markov Random Fields can be characterized by their clique factorization (if it exists). In our case the factorization is quite simple. The cliques are the single pixels $(i, j) \in G$ and all combinations of neighboring pixels $(i, j), (k, l) \in G$ where $(k, l) \in N_{ij}$. The associated clique potentials are (up to a normalizing factor) respectively $e^{-U_{ij}(d_{ij})}$ and $e^{-\beta V(d_{ij}, d_{kl})}$. In this way we can write:
\[ P[d] = \frac{1}{Z} \prod_{(i,j) \in G} e^{-U_{ij}(d_{ij})} \prod_{(i,j),(k,l) \in G} e^{-\beta V(d_{ij},d_{kl})} \] (4.2)

for the probability density of the disparity map.

### 4.3 Variational Mean Field equations

The new problem is to find the disparity map \( d^* \) which has maximum probability. This is done by using a Variational Mean Field (VMF) approach [3]. The resulting disparity map will not be the exact solution to the maximization problem, but only an approximation. Nonetheless this approximation will prove to give satisfying results. The essence of the approach is to find a suitable approximation \( Q \) to the complicated probability density \( P \). Then we do the maximization process using the simpler density \( Q \).

The key point of VMF is that the approximating distribution \( Q \) is taken to be a fully factorized distribution:

\[ Q[d] = \prod_{(i,j) \in G} Q_{ij}(d_{ij}) \] (4.3)

Next we need some sort of distance measure that indicates how well \( Q \) approximates \( P \). For this the standard Kullback-Leibler (KL) divergence (also called the relative entropy) is used:

\[ \text{KL}(Q\|P) = \sum_d Q[d] \ln \left( \frac{Q[d]}{P[d]} \right) \] (4.4)

Using the factorization of \( Q \) we can express KL as:

\[
\text{KL}(Q\|P) = \sum_d \prod_{(i,j) \in G} Q_{ij}(d_{ij}) \ln \left( \prod_{(i,j) \in G} Q_{ij}(d_{ij}) \right) - \sum_d \prod_{(i,j) \in G} Q_{ij}(d_{ij}) \ln (P[d]) \\
= \sum_{d_{11}} Q_{11}(d_{11}) \sum_{d_{12}} Q_{12}(d_{12}) \cdots \sum_{d_{mn}} Q_{mn}(d_{mn}) \sum_{(r,s) \in G} \ln (Q_{rs}(d_{rs})) - \cdots \\
= \sum_{(i,j) \in G} \sum_{d_{ij}} Q_{ij}(d_{ij}) \ln (Q_{ij}(d_{ij})) - \cdots \\
= - \sum_{(i,j) \in G} H(Q_{ij}) - \cdots \\
= - \sum_{(i,j) \in G} H(Q_{ij}) - \sum_d \prod_{(i,j) \in G} Q_{ij}(d_{ij}) \cdot (-E(d) - \ln Z) \\
= - \sum_{(i,j) \in G} H(Q_{ij}) + \ln Z + \sum_d \prod_{(i,j) \in G} Q_{ij}(d_{ij}) E(d) \] (4.5)

where \( H \) denotes the Shannon entropy function, i.e. \( H(Q_{ij}) = \sum_{d_{ij}} Q_{ij}(d_{ij}) \ln (Q_{ij}(d_{ij})) \).

The idea is to minimize \( \text{KL}(Q\|P) \) with respect to \( Q \). This is done by deriving the set of equations for the minimization with respect to a distribution \( Q_{rs} \) for a specific pixel \((r,s)\). We don’t
make any assumptions about \(Q_{rs}\) so to completely describe the distribution we need to list all the possible probabilities \(\{Q_{rs}(0), Q_{rs}(1), \ldots, Q_{rs}(d_{\text{max}})\}\) and treat them as separate variables.

We first separate \(\text{KL}(Q\|P)\) into the part that depends on \(Q_{rs}\) and the part that doesn’t:

\[
\text{KL}(Q\|P) = -H(Q_{rs}) - \sum_{(i,j) \in G, (i,j) \neq (r,s)} H(Q_{ij}) + \ln Z + \sum_{d_{rs}} Q_{rs}(d_{rs}) \sum_{d \setminus d_{rs}} \left( \prod_{(i,j) \in G, (i,j) \neq (r,s)} Q_{ij}(d_{ij}) \right) \sum_{(i,j) \in G} \left\{ U_{ij}(d_{ij}) + \beta \sum_{(k,l) \in N_{ij}} V(d_{ij}, d_{kl}) \right\}
\]

(4.6)

where we replaced \(E(d)\) by the expression from equation (3.7). In the last line the sum \(\sum_{d}\) was split into the sum \(\sum_{d_{rs}}\) over \(d_{rs}\) and the \(\sum_{d \setminus d_{rs}}\) over all the remaining variables.

If we now take the partial derivative of \(\text{KL}(Q\|P)\) with respect to a particular variable \(Q_{rs}(t)\) we get:

\[
\frac{\partial \text{KL}(Q\|P)}{\partial Q_{rs}(t)} = 1 + \ln(Q_{rs}(t)) + \sum_{d_{rs} = t} \left( \prod_{(i,j) \in G, (i,j) \neq (r,s)} Q_{ij}(d_{ij}) \right) \sum_{(i,j) \in G} \left\{ U_{ij}(d_{ij}) + \beta \sum_{(k,l) \in N_{ij}} V(d_{ij}, d_{kl}) \right\}
\]

\[
= 1 + \ln(Q_{rs}(t)) + U_{rs}(t) + 2\beta \sum_{d_{N_{rs}}} \left( \prod_{(i,j) \in N_{rs}} Q_{ij}(d_{ij}) \right) \sum_{(i,j) \in N_{rs}} V(t, d_{ij})
\]

\[
= 1 + \ln(Q_{rs}(t)) + U_{rs}(t) + 2\beta \sum_{(i,j) \in N_{rs}} W_{ij}(t) + C_0
\]

(4.7)

where \(d_{N_{rs}}\) indicates the 8 (or less) dimensional sum over the possible disparity values of the neighbors of the pixel \((r, s)\). We introduced the shorter notation:

\[
W_{ij}(t) = \sum_{d_{ij}} Q_{ij}(d_{ij})V(d_{ij}, t)
\]

(4.8)

for the expected value of \(V(\cdot, t)\) under the distribution \(Q_{ij}\). The constant \(C_0\) is the part from the sum that we left out and which does not depend on \(t\). It will become clear in a bit why we don’t need to compute it.

Since \(Q_{rs}(d_{rs})\) is a probability distribution it satisfies the constraint:

\[
\sum_{d_{rs}} Q_{rs}(d_{rs}) = 1
\]

To minimize \(\text{KL}(Q\|P)\) with respect to \(Q_{rs}\) we introduce the Lagrangian function

\[
L(Q_{rs}) = \text{KL}(Q\|P) - \lambda \left( \sum_{d_{rs}} Q_{rs}(d_{rs}) - 1 \right)
\]

(4.9)
Taking the derivative with respect to $Q_{rs}(t)$ and equating it to 0 we get:

$$\frac{\partial L(Q_{rs}(t))}{\partial Q_{rs}(t)} = 1 + \ln(Q_{rs}(t)) + U_{rs}(t) + 2\beta \sum_{(i,j) \in \mathcal{N}_{rs}} W_{ij}(t) + C_0 - \lambda = 0$$

$$\ln Q^*_rs(t) = -U_{rs}(t) - 2\beta \sum_{(i,j) \in \mathcal{N}_{rs}} W_{ij}(t) + \lambda - C_0 - 1 \quad (4.10)$$

as the set of equations for the optimal distribution $Q^*_rs(t)$ for each $t \in D$. The part $\lambda-C_0-1 \equiv \ln \frac{1}{K}$ does not depend on $t$ and we can easily determine this constant from the normalization constraint.

We then end up with:

$$Q^*_rs(t) = e^{-U_{rs}(t) - 2\beta \sum_{(i,j) \in \mathcal{N}_{rs}} W_{ij}(t)} K \quad (4.11)$$

where the normalizing factor $K$ is equal to:

$$K = \sum_t e^{-U_{rs}(t) - 2\beta \sum_{(i,j) \in \mathcal{N}_{rs}} W_{ij}(t)}$$

The equations represented by (4.11) are called the Variational Mean Field equations. They are the cornerstone of our model and it is our aim to solve them. How this is done is explained next.

### 4.4 Iterative procedure

Finding the optimum approximating distribution $Q^*$ is done by computing a fixed point of the VMF equations through an iterative procedure. An example is shown in figure 4.1 below.

![Figure 4.1](image_url)

**Figure 4.1:** Illustration of the method to compute a fixed point $x^*$ of the function $f(x) = \sqrt{x}$.

We start with the fully uniform distribution $Q^{(0)}$, i.e. $Q^{(0)}_{rs}(d_{rs}) = \frac{1}{1+\delta_{\text{max}}}$ for each $(r,s) \in G$. Then we iteratively compute $Q^{(k)}$ for $k = 1, 2, \ldots$ through equation (4.11) until we are satisfied with the approximation $Q^{(k)}$. In more detail:

$$Q^{(k+1)}_{rs}(t) \sim e^{-U_{rs}(t) - 2\beta \sum_{(i,j) \in \mathcal{N}_{rs}} W^{(k)}_{ij}(t)} \quad (4.12)$$

where $W^{(k)}_{ij}(t) = \sum_{d_{ij}} Q^{(k)}_{ij}(d_{ij}) V(d_{ij}, t)$ uses the distribution $Q^{(k)}$ from the previous iteration step.

After computing these values $Q^{(k+1)}_{rs}(t)$ for each $t \in D$ we do a normalization such that $Q^{(k+1)}_{rs}$
becomes a probability distribution again, i.e. sums up to 1. Note that in each iteration \( k \) a distribution \( Q_{rs}^{(k)} \) is updated based on the distributions from the neighborhood \( N_{rs} \) only. This means that not many computations are needed for each pixel, and the model can be implemented quite efficiently on a normal pc.

Finally, selecting the best disparity map \( d^* \) is a straightforward procedure. For each pixel \((r, s)\) we take the value \( d_{rs}^* \) which maximizes the probability, i.e. \( d_{rs}^* = \arg \max_{d_{rs}} Q_{rs}^{(k)}(d_{rs}). \)
Chapter 5

Model extensions

In this chapter we will discuss some of the extensions that can be made to the basic model which we have presented so far. The objective of these extensions is either to increase the computation speed of the algorithm or to improve the resulting disparity map. Real world examples of stereo images have a lot of difficulties in them and the objective of several of the extensions is to overcome these. Think of areas in the image with very little texture, very poorly visible object boundaries or objects which are present in only one of the two images because they disappear behind another object in the other image. This last kind is called an occluded object and it is one of the major difficulties in creating a good disparity map.

5.1 Multi-scale approach

The optimization process to find the disparity map can take quite a lot of iterations before something good comes out. The main problem is that information only travels at a speed of 1 pixel per iteration. What we mean by this is that at iteration \( k \) the optimal disparity at a certain pixel \((i,j)\) is entirely determined by the pixels which are at a distance of \( k \) or less away (see figure 5.1). All the other pixels do not influence the value that we obtain for the original pixel.

In the end though we want a disparity map that is close to a global optimum, so one where information has traveled over the entire image and not just over a small region.

A very good way to incorporate this global information flow and at the same time cut down on the number of iterations needed is to look at the image on multiple scales [2, 5]. The idea is to first group together blocks of pixels of a certain size (for instance \( 2 \times 2 \)) and treat them as a single pixel. This is depicted in figure 5.2 below.

![Figure 5.1: At iteration \( k = 2 \) the red pixel is influenced only by the blue pixels.](image-url)
In terms of the graph model of section 3.4 this can be described as shrinking each set of 4 vertices that belong to a block into a single vertex. The new graph looks exactly the same as the old one, except it has 4 times less vertices. The edges, which represent interaction energies, now are between the blocks which are assigned a single disparity value as a whole. This presents one problem though. We have to come up with a good cost function $U_B(d_B)$ for the disparity $d_B$ of a block $B$. A very simple but effective solution is to just take the average of the individual pixel cost functions:

$$U_B(d_B) = \frac{1}{|B|} \sum_{(i,j) \in B} U_{ij}(d_B)$$  \hspace{1cm} (5.1)

where $|B|$ indicates the number of pixels in the block $B$. The interaction energy between the blocks is still represented by the function $V$ from equation (3.5).

We now simply run the algorithm with the altered data cost functions $U_B$ to obtain a disparity distribution $Q_B(d_B)$ for each block $B$. Then we decrease the size of the blocks by a factor of 2 and use these block distributions as the initial distributions for the 4 sub-blocks that made up the original. In graph language this means that we undo the shrinking of the vertices. We keep repeating this process, each time running the algorithm with a fixed number of iterations on a decreasing scale until we arrive at pixel level. In each step, the information speed is limited by 1 block per iteration, but because blocks can be as large as for instance $8 \times 8$ pixels we see that it is a major improvement over the original algorithm. In our computations we generally started on a scale in which blocks are $8 \times 8$ pixels.

5.2 Depth discontinuities

The disparity map is not required to be smooth everywhere. In the scene itself there are places where the depth changes discontinuously, for instance at the boundary of an object that stands in front of a background. This should be reflected in our disparity map, i.e. there should be locations where the disparity field makes a large jump. We will call these locations boundary locations even though they do not always occur exclusively on object boundaries (an object might also have large depth variation internally). How can we incorporate the presence of these boundaries into our model? The answer is quite simple. We just remove all edges in the graph which cross or intersect a boundary (see figure 5.3).
All pixels which are connected by a blue edge will no longer interact with each other, i.e. they do not contribute an energy term like equation (3.5) to the total interaction energy (3.5). In terms of the algorithm this means that in the computation of equation (4.12) the neighborhood $\mathcal{N}_r$ becomes smaller for all pixels which are incident with a blue edge. The corresponding energy term $W_{ij}(t)$ from equation (4.8) is removed from the sum. This effectively means that each two pixels for which we removed an edge are not stimulated to have similar disparity values anymore, precisely what we want to achieve. A final note is that boundary pixels themselves still interact with all their neighbors. The reason for this is that it we want them to take on a disparity value similar to one of the two regions that they border. Which of the two they belong to is something that we leave up to the algorithm, so we do not keep track of such information for each individual boundary pixel. The graph picture above is not entirely accurate. A better way would be to replace the undirected graph with a directed graph and remove all the arcs which cross or point away from the boundary, but the general idea of how discontinuities are handled should be clear.

In our model boundary locations are always on pixels themselves. We do not keep track of boundaries which go between pixels. This is not really a severe restriction because in practical applications the input images are usually not accurate enough to determine the object boundaries to such a precise level. How we do determine meaningful boundaries to use in the model is described in chapter 6.

5.3 Occluded pixels

There can be pixels in the left image that are not present in the right image because they have disappeared behind another object. This presents some difficulties because those pixels can not be accurately matched and assigned a correct disparity value. Assuming that we can determine in some way which pixels are occluded, the natural way to handle them in our algorithm is to simply ignore the data energy functions $U_{ij}$ from equation (3.3) for all occluded locations $(i, j)$. Those functions have no meaning anymore. Any preference for a certain disparity value in the form of a lower energy would be purely coincidental.

What will happen is that, as we do more and more iterations of the algorithm, information from pixels in their neighborhood which can reliably be matched will reach the occluded pixel locations and influence them. Eventually those pixels will take on disparity values similar to their surroundings. Since we have no reliable information for the depth of occluded pixels at all this is about the best that we can do. How we determine which pixels are occluded in a meaningful way is described in chapter 6.
5.4 Neighborhood extension

To improve the smoothness of the disparity map we can extend the neighborhood $N_{ij}$ of a pixel $(i, j)$ to also include neighbors which are at a distance of 2 or even further away. Here the distance $\text{dist}((i, j), (k, l))$ between two pixels $(i, j)$ and $(k, l)$ is defined to be the maximum of their absolute coordinate differences:

$$\text{dist}((i, j), (k, l)) = \max\{|i - k|, |j - l|\} \quad (5.2)$$

Intuitively we could say that the higher the distance between two pixels the less they should interact. This is reflected in the weight $\frac{1}{a}$ that we include with each neighbor at a distance $a$. The equation that we have to modify in the algorithm is once again equation (4.12). The sum $\sum_{(i,j) \in N_{rs}} W^{(k)}_{ij}(t)$ is replaced with:

$$\sum_{a=1}^{a_{\text{max}}} \frac{1}{a} \sum_{(i,j) \in G} W^{(k)}_{ij}(t) \quad (5.3)$$

where $a_{\text{max}}$ is the maximum distance which we take into account.

5.5 Multiple cameras

In the case that we have a third view of the scene available we can improve the algorithm by modifying the data energy functions $U_{ij}$ for each pixel $(i, j)$. We assume that the three cameras are oriented the same way and are stationed in a line with the distance between the left and middle camera being equal to that between the middle and right. Under these assumptions a scene point that appears on the middle image at coordinates $(i, j)$ will be visible on the left and right images at the locations $(i + d_{ij}, j)$ and $(i - d_{ij}, j)$ for some value of $d_{ij}$ unless that pixel is occluded on either image. This means that we have two ways of matching a pixel if we decide to compute a disparity map for the middle image. If a pixel is occluded on the left image because it disappeared behind another object we have a good chance to find it on the right image. The only possibility where a pixel would be occluded on both images is the case where two different objects are very close to each other but with a small gap between them. This is a rare occasion however.

If we denote the left, middle and right images by $I_L$, $I_M$ and $I_R$ respectively we can modify equation 3.3 by:

$$U_{ij}(d_{ij}) = \min\{\lambda_1 |I_M(i, j) - I_L(i + d_{ij}, j)|, \lambda_1 |I_M(i, j) - I_R(i - d_{ij}, j)|, T_1\} \quad (5.4)$$

This approach solves a lot of the occlusion issues but it requires three aligned camera views, something which might not always be available in practical applications.

5.6 Window size

A further improvement that we can make to the data energy functions $U_{ij}$ is to include also neighboring pixels of the pixels $(i, j)$ in the computation. Instead of matching just single pixels we can start with a square window of pixels of size $w$ centered on the original pixel $(i, j)$ in the left image and try to match the entire window in the right image (see figure 5.4).

The new data energy functions $\bar{U}_{ij}$ are computed as:
Figure 5.4: A square window of size $w = 2$ is centered around the original pixel and matched onto the right image.

$$
\bar{U}_{ij}(d_{ij}) = \sum_{a=0}^{w} \frac{1}{a + 1} \sum_{(k,l) \in G, \text{dist}((i,j),(k,l)) = a} U_{kl}(d_{ij})
$$

(5.5)

where we included a weight $\frac{1}{1+a}$ such that the further away a window pixel is from the center the less it contributes to the overall energy.

In most of our runs of the algorithm we kept the window size relatively small, for instance $w = 1$ or $w = 0$ in which case we have windows of single pixels again. This seemed to work best since part of the window idea is already captured by the block approach of section 5.1. On higher scales we are already matching blocks of pixels so it seems redundant to make the window size $w$ too large.
Chapter 6

Boundary detection

6.1 Automatic versus manual

In an ideal world we would be able to find all the object boundaries with a fully automatic edge detecting algorithm. Unfortunately in practical applications this never works and at best we can hope that such an algorithm correctly detects some of the right edges. That is why we opt for a semi-automatic approach. The idea is to first apply a good edge detecting algorithm to the left image, and then let a user manually adjust the resulting boundary map. Our aim is that the more input a user applies to this boundary map the better the result of the computed depth map will be. The model should still work to a reasonable extent even with no user input at all. A trade off will then have to be made between how much input the user is willing to give and if they are satisfied with the quality of the resulting depth map.

6.2 Finding the correct object boundaries using disparity information

A first approach to find correct object boundaries is by using the information from the disparity map obtained by our algorithm. Most of the boundaries which result from the algorithm so far are reasonably good. There really is only one major thing that can cause incorrect boundaries, namely occlusions. Occlusions are pixels which can be seen in one image, but not in the other because they disappear behind an object. The algorithm doesn’t detect them and the depth values assigned to them can be completely wrong.

For instance if we look at the very simple scene in figure 6.1 we notice that on the right image the yellow side of the cube is no longer visible. This presents a problem to our algorithm because it tries to match each pixel in the left image to one in the right image. This goes wrong for the yellow pixels, because they can’t be matched. The result is that the left boundary of

\begin{figure}[h]
\centering
\subfloat[Left image.]{
\includegraphics[width=0.4\textwidth]{left_image.png}}\quad
\subfloat[Right image.]{
\includegraphics[width=0.4\textwidth]{right_image.png}}
\caption{A simple cube with different colored sides.}
\end{figure}
the cube becomes distorted as some of the yellow pixels are matched onto green pixels. Notice
that a similar problem does not happen on the right boundary of the cube because there the blue
pixels aren’t present in the left image. The right boundary will appear correctly in the depth image.

This gives us an idea. If we do the reverse and try to match the pixels in the right image onto
the left image we get exactly the opposite effect. Now the left boundaries will look nicely and
the right boundaries will be more vague. We could combine these new left boundaries with the
original right boundaries to obtain a depth map with good boundaries on both sides. The exact
way in which this is done is described next.

First we match the right image onto the left one and we extract a map of discontinuity locations
by looking for pixels \((i, j)\) which satisfy:

\[
d(i, j) - d(i - 1, j) > \frac{T_2}{\lambda_2}
\]

or in other words the interaction energy from equation (3.5) between the pixels is equal to \(T_2\).
Note that we only get the left boundaries since we are comparing pixels with their left neighbor.
We determine the occluded pixels that result from this boundary pixel by comparing \(d(i, j)\) with
\(d(i - 1, j)\). Since pixel \(p_2 = (i, j)\) in the right image is matched to pixel \(p_1 = (i + d(i, j), j)\) in
the left, and similarly \(q_2 = (i - 1, j)\) to \(q_1 = (i - 1 + d(i - 1, j), j)\), we mark the set of pixels
\((q_1 \ldots p_1)\), the yellow pixels, as being occluded in the left image. The rightmost pixel \(p_1\) is marked
as a boundary pixel. The whole process is illustrated in figure 6.2 below.

![Figure 6.2: Matching pixels near a left boundary of an object.](image)

This approach was one of the first that we tried. While it does give quite nice boundaries and even
reliable occlusion information, it is not very robust. It requires a reasonably good disparity map in
the first place, especially near the left or right object boundaries. In a lot of practical applications
even un-occluded pixels can sometimes not be accurately matched and the boundaries will end up
vague and distorted. This approach then fails. It also doesn’t lend itself very well to allow a user
to improve the boundaries.

### 6.3 Automatic edge detection

There are many types of automatic edge detection algorithms [10]. For our model we decided to
use one of the simplest but which is still very effective. This is the Sobel edge detector. It is based
on the discrete Sobel gradient operator which tries to compute an approximation to the gradient
of the image intensity function. The gradient is a measure of how abruptly the image changes at
a certain location and is a good indication of where an edge might be. Our left image is given by
the color intensity function \(I_L(i, j)\), which is a vector valued function. Because the gradient can
only be applied to a single valued function we have to treat each of the three color components
separately.
6.3.1 Sobel operator

The Sobel operator uses the following two $3 \times 3$ matrices $G_x$ and $G_y$ to compute the gradient of a function $f(i,j)$ sampled at discrete locations $(i,j) \in G$:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

and

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Here $G_x$ is used for the gradient component in the $x$ direction, and $G_y$ for the $y$ direction. The gradient values $g_x$ and $g_y$ are computed by centering these two matrices on top of the pixel $(i,j)$ and summing over the $3 \times 3$ window of pixels around it:

$$g_x = \sum_{p=-1}^{1} \sum_{q=-1}^{1} f(i+p,j+q) G_x(p,q)$$

$$g_y = \sum_{p=-1}^{1} \sum_{q=-1}^{1} f(i+p,j+q) G_y(p,q)$$

where $G_x(p,q)$ indicates the entry $pq$ of the matrix $G_x$ and likewise for $G_y$. For simplicity we let the matrix indices run from $-1, \ldots, 1$.

The length of the gradient $g$ is computed as:

$$g = \sqrt{g_x^2 + g_y^2}$$

In our case, because we are working with color images, we have three components $g_{\text{red}}, g_{\text{blue}}, g_{\text{green}}$ such that the total gradient $g_{\text{total}}$ can be written as:

$$g_{\text{total}} = \sqrt{g_{\text{red}}^2 + g_{\text{blue}}^2 + g_{\text{green}}^2}$$
6.3.2 Boundary extraction

An example of applying the Sobel gradient operator to a real image is shown in figure 6.3.

![Sobel operator applied to a color image. The result is shown as a gray-scale image, where brighter pixels have a higher gradient value.](image)

As can be seen the areas with high gradient value correspond quite well with object boundaries. A first approach to extract the boundaries would be to simply apply a threshold $T_g$. We select all pixels $(i, j) \in G$ with a gradient value $g_{\text{total}}(i, j) \geq T_g$ as boundary pixels. This gives quite nice results but some boundaries are too thick. We would like to have sharply defined boundaries so we do a further refinement of the boundary map by selecting only those pixels which are a local maximum. The procedure is illustrated in figure 6.4 below.

![Visual representation of the procedure to determine the final boundary pixels.](image)

Only the pixels which satisfy the following two criteria are retained as boundary pixels:

- The gradient value at the pixel is above the threshold $T_g$.
- In at least 1 of the 4 directions depicted in the figure above the pixel has maximum gradient among the three pixels on that line.

The effect of this simple refinement procedure is that in areas of the image where the color changes a lot over a small range of pixels we still obtain a sharp boundary which is 1 or 2 pixels thick. In such a situation multiple neighboring pixels could have a gradient value above the threshold $T_g$ and we would normally have obtained a boundary which is 3 or more pixels thick.
6.4 Manual input

Our implementation of the model allows for several ways of giving manual input in order to obtain an improved depth map. The user can alter the automatically computed boundary map by deleting faulty edges or adding missing ones. If edges are drawn such that it results in a completely closed region in the image, the user can assign a specific search range for the disparity values of the pixels inside the region. Instead of the initial assumption that the values should lie in the set $[0, \ldots, d_{\text{max}}]$ the user can specify a new range $[d_{\text{lower}}, d_{\text{upper}}]$ for that region. This search space restriction can give very good results if there are large areas in the image which have relatively uniform disparity. For instance the background of most images usually displays very little depth variation and would be a good candidate for this. This method can also be used to assign a global disparity range for the entire image. A further advantage of this approach is that it improves the running time of the algorithm by a lot since in all the computations disparity values outside the restricted range are now ignored resulting in much fewer computations overall. The downside is that it requires the user to draw a complete segmentation of the area which might in some cases simply be too much work. A final way in which the user can give input are the various parameters which have been introduced in the model.
Chapter 7

Results

In this chapter we will present some results obtained from our model.

7.1 Influence of the parameter $\beta$

We start with some results that show the influence of the parameter $\beta$.

![Image of depth maps for increasing values of $\beta$.](image)

**Figure 7.1:** Computed depth maps for increasing values of $\beta$.

We can see that for $\beta = 0$ the general features of the scene are present in the depth map but there is a lot of noise. Increasing $\beta$ removes this noise but has the downside of also removing details. For $\beta = 2$ the boundaries are already starting to become seriously distorted. Increasing $\beta$ further would result in a depth map which is even smoother but corresponds very poorly with the scene.
7.2 Disparity based boundary estimation

We present some results of the method to obtain object boundaries using disparity information from both images, which was described in section 6.2.

![Disparity maps](image)

**Figure 7.2:** Results from running the algorithm in both directions, matching the left image onto the right and vice versa.

We can see that in the disparity map of the left image all the right boundaries look quite nicely. The opposite effect is visible in the right disparity map. There the left boundaries are of much higher quality. Using these two disparity maps we can combine both boundaries, extract a map of occluded pixels and use this information to obtain a final improved disparity map. The results from this process are shown in figure 7.3.

![Combined boundaries](image)

**Figure 7.3:** The end result of the combination of both disparity maps.
7.3 Using three camera views

Using a third view of the scene greatly improves the resulting depth map as problems with a lot of occluded pixels disappear. We present some results that demonstrate the ideas from section 5.5.

![The middle view](image1.png) ![Resulting disparity map](image2.png)

Figure 7.4: Example showing the added benefit of having a third view available.

We computed the disparity map for the middle view out of three views. If we compare figure 7.4 with the results from figures 7.2 and 7.3 we can see that three view approach greatly reduces occlusion difficulties. The boundaries are sharp and well positioned on both sides and the top left part of the plant corresponds much better to the image than in the two-view case where it was completely distorted because of occlusions.

7.4 Poorly textured regions

Poorly texture regions, i.e. regions of very homogenous color, are one of the hardest things to handle in depth estimation. The reason is that matching pixels from this region between two images is very difficult. There are a lot of possible candidate pixels which are all very much alike. Even a human can have difficulty sometimes in determining the depth of a poorly textured object. Think for instance of a completely black ball in front of a very dark background. The only way to extract reliable depth information is from the boundary of the ball. We show some results for a scene with a very poorly textured background.

![The original image](image3.png) ![Automatically computed depth map](image4.png) ![Background segmented depth map](image5.png)

Figure 7.5: Example of a scene that has large areas with poor texture.

The depth map in figure 7.5(b) is computed by our regular algorithm. We see that the objects in the scene are assigned good depth values but that the background is almost exclusively dominated
by noise. This is expected since there are very few points in the background that can give us a cue about its depth. In 7.5(c) we have made a manual segmentation of the background and restricted the depth of all pixels in it to a very small interval.

### 7.5 Manually corrected boundaries

Finally we show some results from depth maps using manual correction of object boundaries. We start first with some entirely automated depth map results.

![Example scene to illustrate the necessity of boundary detection.](image)

No manual input has been given yet in the computations of these depth maps except setting the value of $\beta$. Increasing $\beta$ further is necessary to remove the noise but doing so will distort object boundaries. In figure 7.8 we show the results when automatically computed boundaries are incorporated. We changed the neighbor inclusion distance from section 5.4 from 1 to 3 such that values for $\beta$ can be set a little lower than usual.
The noise in the foreground objects is further removed but the automatically detected boundaries are still not sufficient. With some manual correction of the boundaries and setting a global disparity interval for all pixels in the image we can increase $\beta$ even further, without worrying about distorting the boundaries, and we finally obtain the following result.
Chapter 8

Conclusions and further work

In this report we have described a model for computing a depth map from a set of stereo images. We started with a mathematically sound model by casting the problem in the form of an energy minimization problem. This minimization problem is not exactly solvable and an iterative procedure was developed in order to obtain an approximate solution. While this solution method gives reasonable results, we made several extensions to the model in order to overcome problems that present themselves in real world example images. An important guideline in adding these extensions was that we wanted our model to accept user input in order to produce better depth maps. Manually corrected object boundaries and the marking of occluded pixels are two examples in which a user can add input. This results in much better depth maps but then the question arises of how much manual input is acceptable? This is a question that is difficult to answer and which we have not really investigated in this report but one that will be very important for practical applications.

Our model is very flexible in the sense that there are a lot of simple changes that could be made to it, which could potentially give much better results. We have briefly investigated some of these changes. We left them out of the report because they did not give very satisfying results. Still we think they are a very good candidate for improvements to the model. A few examples are:

- **Variable data weights:** In the current model each data energy function $U_{ij}$ has the same weight. This could be changed to give a separate weight to each pixel depending on how reliably we think that pixel can be matched. For instance pixels which are in regions with high texture can more easily be matched in the other image and thus have more reliable disparity values. A possible approach would be to give these pixels more weight so neighboring pixels which are harder to match receive more influence from these reliable pixels.

- **Variable interaction weights:** In the same way we can also give a separate weight to each interaction energy function $V(d_{ij}, d_{kl})$ for two neighboring pixels $(i, j)$ and $(k, l)$. An idea would be to give nearby regions of very similar color a higher weight such that these regions tend to take on similar disparity values since it’s very likely that they are part of the same surface.

- **Energy functions:** The form of the energy functions $U$ and $V$ are a good choice for experimentation. We think that the simple formulas which we use now capture the basic ideas, but that more advanced functions might give better results.

- **Poorly textured regions:** One of the biggest difficulties for the model is regions of poor texture. These areas often occur in the background parts of an image. They are hard to match or assign a good disparity value to them, especially when they consist of a large amount of pixels. We are not happy with the way how these regions are handled in the model currently. An approach that could be taken is to do some kind of color clustering and
treat regions of very homogenous color as single blocks in the model. This way all pixels in a certain region would automatically get assigned a similar disparity value since they are likely to be part of the same surface and thus have roughly similar depth. In this way we do not have to increase the $\beta$ parameter too much to remove noise in the background and we can keep the important details in the foreground objects.

One of the biggest improvements that can be made is the extension to video sequences. The model now only works for a single frame. While a video consists of a large number of frames and we could just run the algorithm separately for each frame this simply takes too much time. To obtain good results at least some amount of user input is needed but it is infeasible to do this for every single frame. Some kind of interpolation scheme has to be thought of, where some manual input is given only at certain frames in the sequence. In addition, extra information can be extracted from comparing subsequent frames in a movie sequence which could help in creating better depth maps. Exactly what information is valuable and how it should be used is an area of further research.
Bibliography


