Intermediate View Interpolation of Stereoscopic Images for 3D-Display

Examensarbete utfört i Reglerteknik vid Tekniska högskolan i Linköping av

Oskar Thulin

LITH-ISY-EX--06/3866--SE

Linköping 2006
Intermediate View Interpolation of Stereoscopic Images for 3D-Display

Examensarbete utfört i Reglerteknik
vid Tekniska högskolan i Linköping
av

Oskar Thulin

LITH-ISY-EX--06/3866--SE

Handledare: Jeroen Hol
ISY, Linköpings universitet

Thomas Ericsson
Setred AB

Examinator: Fredrik Gustafsson
ISY, Linköpings universitet

Linköping, 31 October, 2006
This thesis investigates how disparity estimation may be used to visualize an object on a 3D-screen. The first part looks into different methods of disparity estimation, and the second part examines different ways to visualize an object from one or several stereo pairs and a disparity map. Input to the system is one or several stereo pairs, and output is a sequence of images of the input scene but from more angles. This sequence of images can be shown on Setred AB’s 3D-screen. The system has high real time demands and the goal is to do the disparity estimation and visualization in real time.

In the first part of the thesis, three different ways to calculate disparity maps are implemented and compared. The three methods are correlation-based, local structure-based and phase-based techniques. The correlation-based methods cannot satisfy the real-time demands due to the large number of 2D-convolutions required per pixel. The local structure-based methods have too much noise and cannot satisfy the quality requirements. Therefore, the best method by far is the phase-based method. This method has been implemented in Matlab and C and comparisons between the different implementations are presented.

The quality of the disparity maps is satisfying, but the real-time demands cannot yet be fulfilled. The future work is therefore to optimize the C code and move some functions to a GPU, because a GPU can perform calculations in parallel with the CPU. Another reason is that many of the calculations are related to resizing and warping, which are well-suited to implementation on a GPU.
Abstract

This thesis investigates how disparity estimation may be used to visualize an object on a 3D-screen. The first part looks into different methods of disparity estimation, and the second part examines different ways to visualize an object from one or several stereo pairs and a disparity map. Input to the system is one or several stereo pairs, and output is a sequence of images of the input scene but from more angles. This sequence of images can be shown on Setred AB’s 3D-screen. The system has high real-time demands and the goal is to do the disparity estimation and visualization in real time.

In the first part of the thesis, three different ways to calculate disparity maps are implemented and compared. The three methods are correlation-based, local structure-based and phase-based techniques. The correlation-based methods cannot satisfy the real-time demands due to the large number of 2D-convolutions required per pixel. The local structure-based methods have too much noise and cannot satisfy the quality requirements. Therefore, the best method by far is the phase-based method. This method has been implemented in Matlab and C and comparisons between the different implementations are presented.

The quality of the disparity maps is satisfying, but the real-time demands cannot yet be fulfilled. The future work is therefore to optimize the C code and move some functions to a GPU, because a GPU can perform calculations in parallel with the CPU. Another reason is that many of the calculations are related to resizing and warping, which are well-suited to implementation on a GPU.

Sammanfattning


I den första delen av examensarbetet undersöcktes framför allt tre sätt att beräkna disparitet: fasbaserade, korrelationsbaserade och lokal-strukturbaserade metoder. Med prestandsakraven kan man snabbt utsluta de korrelationsbaserade metoderna då dessa ofta kräver 2D-faltningar per pixel. Den lokal-strukturbaserade
metoden kallas för polynomexpansion och har inte visat sig kunna ge disparitet-
skartor av tillräckligt hög kvalitet. Metoden som framstår som den klart främsta
är den fasbaserade. Denna har i detta arbete, förutom att implementeras i Matlab,
even implementerats i C för att försöka tillåtesgå prestandakraven.

Den fasbaserade metoden bygger på att beräkna den lokala signalens lokala
fas och frekvens, där signalen ses som en horisontell linje i bilderna. Det första
steget är att beräkna komplexa filtersvar till hela bilderna (höger och vänster),
med hjälp av kvadraturlfilter. Nästa steg är att beräkna fas och frekvens, för att
sedan kunna beräkna dispariteten. För att förbättra beräkningarna görs även en
beräkning av säkerhetsmått, för att uppskatta hur mycket man ska lita på vår-
dena. Alla beräkningar görs i flera skalor för att kunna hitta signalernas olika
frekvenskomponenter. Man kan få samma resultat av att beräkna kvadraturlfiltr-
tren i olika skalor, men detta kräver fyra gånger fler beräkningar i varje skala.
Fasmetoden har gett tillfredsställande resultat men inte nått tillräcklig fart. Fort-
sättningen på arbetet blir således att optimera C-koden och byta ut vissa delar
till GPU-beräkningar för att nå högre hastighet. Koden lämpar sig väl för GPU,
eftersom stora delar av beräkningarna är upp/nedsampling, warping och liknande
grafikoperationer.
Acknowledgments

I would like to thank Setred AB for the opportunity to write this thesis, especially Doug Patterson with C programming and proof reading, and Joel de Vahl with C and OpenGL programming. They have both been of invaluable help throughout the project. I would also like to thank my examiner, professor Fredrik Gustafsson, my supervisor Jeroen Hol and my opponent Stefan Wedell for their contribution to my work. Christina Gratorp has been of great help with proof reading.

Reference literature that has been of great help throughout the work is Gustafsson et al.[9] for signaling, Granlund and Knutsson[8] for computer vision, Oetiker et al.[15] for LaTeX and Owens[17] for GPU programming.
## Contents

1 Introduction 1  
   1.1 Setred 2  
   1.2 Background 2  

2 Stereo Vision Basics 5  
   2.1 Camera setup 5  
   2.2 Stereo Pairs 6  

3 Methods for Disparity Estimation 9  
   3.1 Correlation-based Techniques 10  
   3.2 Feature-based Techniques 13  
   3.3 Local Polynomial Expansion 13  
   3.4 Phase-Based Techniques 15  
      3.4.1 Methods for enhancing disparity maps 20  

4 Methods for Visualization 23  
   4.1 Creating a 3D-object 23  
   4.2 Image sequencing 23  
   4.3 PSNR 26  
   4.4 Slicing 27  

5 Real Time Implementation 29  
   5.1 C Implementation 29  
      5.1.1 Performance 30  
      5.1.2 Optimization 32  
   5.2 GPU Functions 32  
   5.3 Hardware Implementation 33  

6 Conclusions 35  
   6.1 Disparity Estimation 35  
   6.2 Visualization 36  
   6.3 Future work 36  
   6.4 Applications 36  

Bibliography 39
Chapter 1

Introduction

Disparity estimation is a fundamental problem in image processing, and means determining the distance to objects from a stereoscopic image pair. This can be used in motion detection and 3D-representation. The great advantage over other distance determining methods such as radar is that no advanced equipment is required. The only requirements are cameras (two or more) with the same image plane.

The disparity is defined as the horizontal pixel separation between the left and right image, and the distance is calculated from the disparity as

\[ Z = \frac{2hf}{d} \]  

where \( Z \) is distance to the object, \( h \) is the camera separation, \( f \) is the focal length of the cameras and \( d \) is the pixel separation (disparity). There are many ways to determine the pixel separation, but the central problem in all methods is the correspondance problem. The correspondance problem asks which point in one image matches a certain point in the other image. This is investigated in section 2.2.

The object of the thesis is to write an implementation for visualizing an object on a 3D-display, where the input data is one or several stereo pairs. The first part of the project is to evaluate different methods for calculating the disparity or depth maps between one or several stereo pairs. Different methods to determine the disparity will be implemented and compared in Matlab. The second part examines different ways to generate image sequences from the depth maps. This will be referred to as the visualization part. This part also contains an evaluation of different methods and implementations. The 3D-representation will be done on Setred’s 3D-screen and will not be investigated in this thesis, more than that the images need cropping and sorting before they can be sent to the screen.

The image sequence can be thought of as a series of snapshots taken by a camera moving between the pairs. This sequence can also be stretched outside the pairs with good accuracy, although not as good as in between, as the quality of the interpolated images decreases with larger angles. It is of interest to generate these
sequences from several stereo pairs in order to look at an object from many angles with better accuracy. The reason for choosing Matlab is that it has good support for signal and image processing. When a satisfactory method for calculating the disparities and visualizing the images was found, it was also implemented in a low-level language, C. This is due to the fact that the final implementation will have high real-time demands which will not be accomplished in Matlab. Once the C implementation was operational, there was an evaluation of performance followed by optimization of the code. One example of further optimization options is to move certain functions to a GPU (Graphics Processing Unit). The language that will be used is the OpenGL Shader Language which is compatible with the C program. The input data to this GPU function are the two original images and the disparity map. The output data is the sliced image sequence that will be presented on the screen. This means that the GPU part is well differentiated from the other C code. Since the GPU does the last part of the algorithm, the CPU can start computing the next disparity map. This means that the runtimes can be reduced by the time it takes to make the visualizations, see section 5.2.

1.1 Setred

Setred aims to become the leading provider of high-end 3D display solutions. The company’s display technology resulted from joint research between Massachusetts Institute of Technology and Cambridge University.

Setred aims to be the leader in enabling more intuitive and realistic interpretation of three-dimensional information. The company’s first product is a 3D display that acts as a digital hologram. It works with any software application that uses the computer’s 3D graphics card, e.g. CAD applications and games.

The display has a combination of properties that break the current compromises.

- True 3D, allowing you to look around objects by moving your head sideways.
- No restriction in head movement or number of viewers.
- Full colour and full resolution.
- Possible to make flat panel.
- No headgear or head tracking equipment.

There is currently a 20 inch greyscale prototype of the display in addition to a 20 inch colour lab prototype. This greyscale version will be sold to pilot customers, see Figure 1.1. Commercial 50-inch and 20-inch 3D displays will be launched during 2007.

1.2 Background

The reason why Setred want this to be investigated is to make it possible to show images on their 3D-display in real time. Setred has lots of software making it
possible to intercept graphic calls sent to a 2D-display, and to show them on the 3D-display. This thesis is a natural step in order to expand the possible usage for the display. A cheap real time implementation from stereoscopic images will enable Setred to find customers in new areas, such as ROV (Remotely Operated Vehicle) constructors. In this area, it is interesting to get good depth perception and accurate images in real-time 3D.
Introduction
Chapter 2

Stereo Vision Basics

2.1 Camera setup

Stereo vision is about extracting depth information from images. This can be done in different ways depending on the camera setup. In Makoto[14] a method to interpolate intermediate views by calculating projection geometry is investigated. This group of methods determines distances to objects by taking photographs from many angles, but with the cameras facing the same point, see Figure 2.1(a). An obvious disadvantage with this camera setup is that the cameras must at least partially surround the object. On the other hand, this camera setup covers the scene from a wide range of angles.

The other possible camera setup is to have the cameras next to each other facing the same direction, see Figure 2.1(b). This enables good image depth, but does not cover the object from as many angles as the surrounding camera setup. In this thesis we use the same image plane setup due to the expected applications, see section 6.4. The images captured by the cameras are called stereoscopic images, or stereo pairs.

Figure 2.1. Camera setup.
2.2 Stereo Pairs

In this thesis, the depth information is recovered from two or several images. The images are taken of the same scene, with the cameras next to each other, see basic principle in Figure 2.2. There are some constraints on the stereo images. The most important one is that the camera image planes must be the same, see Figure 2.3. The mathematics that extracts the depth information is based on this assumption. Another constraint is that the cameras must have the same focal length, $f$. In order to simplify the calculations, we also add a third constraint, the cameras must be vertically aligned.

When two images are taken of a scene from different view points, the 3D point projection on the images will be in different places. The (horizontal) separation between these points is called the disparity, $d$. To find this pixel separation, we need to calculate the correspondence between pixels in the two images. This is called the stereo matching problem or correspondence problem. This answers the question of which point in the right image matches a certain point in the left image? which is a central problem in stereo matching. To solve this problem, pick one pixel in one image, then search a 2D area around the same pixel in the other image to find the corresponding point, or, the point that best corresponds to the point in the first image. It can be shown that a 1D-search is sufficient, because of the so called epipolar constraint. This assures that if $P_l$ is a projection point of a scene object, $P$, then the projection point in the other image, $P_r$ will be on the epipolar line. The epipolar line is the intersection of the image planes, and the plane that contains $P_l$ and the two centers of projection, $C_l$ and $C_r$. If the cameras are vertically aligned, then the epipolar line will coincide with a horizontal image line, see Figure 2.4. Since we only have a horizontal disparity this will simplify the calculations. A certain disparity is often expected, for example between 0 and 50 pixels and this can make the search area even smaller.

When we know the correspondence, we can extract the disparity by simply calculating the horizontal shift (in pixels) between the two points. To summarize,

- The cameras have the same image plane and focal lengths,
2.2 Stereo Pairs

The cameras are vertically aligned,

The disparity is defined as the horizontal pixel separation between the same point in the two images.

From Figure 2.3 we can extract the following equations describing the horizontal locations of the points in the images, as

\[ x_l = f \frac{h + X}{Z} \quad \text{and} \quad x_r = f \frac{X - h}{Z}. \]  

(2.1)

The \( y \)-coordinate is the same for the two pictures, as it should be according to the epipolar constraint, see Figure 2.4. This means that there is no vertical disparity.

\[ y_l = f \frac{Y}{Z} \quad \text{and} \quad y_r = f \frac{Y}{Z}. \]  

(2.2)

The (horizontal) disparity is defined as the horizontal pixel separation between the two images,

\[ d = x_l - x_r. \]  

(2.3)

From equation (2.1) and (2.3) we can rewrite the disparity, \( d \) and solve for \( Z \) to get the expression for the distance to the objects in the images,

\[ d = \frac{2hf}{Z} \quad \Rightarrow \quad Z = \frac{2hf}{d}. \]  

(2.4)

where \( Z \) is distance, \( h \) is the camera separation and \( f \) is the focal length of the cameras.

Figure 2.3. Stereo vision geometry.
We have now defined the disparity and determined the constraints on the stereoscopic images. The theory behind stereo vision is quite simple and can be summarized in equations (2.1)-(2.4). On the other hand, to solve the correspondence problem is a non-trivial problem which has kept image processing engineers busy for the last three decades. Solving this problem (to find $x_l$ and $x_r$) will be investigated in the following chapter.
Chapter 3

Methods for Disparity Estimation

Determining the disparity of a stereo pair is a fundamental problem in computer vision. The disparity can be estimated using various methods, which can be organized into the following three categories.

- Correlation- or block-based
- Feature-based
- Phase-based

The correlation methods use 2D-matrix correlation calculations between the two images to determine which pixel in one image corresponds to a certain pixel in the other image. The size of the kernels, or convolution blocks, are usually 10-20 pixels high and wide. The convolutions can be more effective if searching is done in certain regions, where the corresponding pixel is expected to be. Read more about this in section 3.1 and see Figure 3.2 and 3.3.

The feature detecting methods are a wide range of methods, based on finding distinct features in the images, such as lines, angles, corners etc. A list of features is stored for each image and then the lists are compared to determine correspondence. The method which is evaluated in this thesis, however, approximates the characteristics around the pixels as a polynomial and then compares the parameters of these polynomials. This has been given its own section in the thesis, since there are some differences between this method and the standard feature detecting methods.

In the phase-based group of methods, the local phase and frequency of the horizontal 1D-signals are extracted. The signals are extracted by computing complex quadrature filter responses of the entire images. The phase and frequency can be calculated from these filter responses. The disparity can then be calculated from the phase and frequency. More on this can be found in section 3.4.
Matlab implementations have been made for all methods. The methods have been tested on several different images with different characteristics. Figure 3.1 shows some of the pictures which are used in the thesis: an aerial view of the Pentagon, a tree and a synthetic image of a corridor.

3.1 Correlation-based Techniques

The objective with this group of methods is to find the corresponding pixels in the left and right images by calculating block correlations, see Figure 3.2. These methods have high computational complexity since the techniques use 2D-convolutions with large kernels for each pixel. This technique does not get better resolution than one pixel, since the disparity is calculated separately for each pixel. Resolution means here the minimum separation between two different levels of disparity. A resolution of one pixel means that the disparity can only be integers. This results in depth maps looking like separate layers on top of each other. On the plus side, it has very little noise and a very straightforward implementation in Matlab.

The Normalized Cross Correlation is one way to determine the similarity between different regions, and is defined as (3.1).

\[ r(m) = \frac{1}{\sigma_{x_L}^2 \cdot \sigma_{x_R}^2} \sum_{n=0}^{N-1} (x_L(n) - \bar{x}_L) \cdot (x_R(n + m) - \bar{x}_R) \]  

The method is slow, but can be made significantly faster by only searching horizontally for the correlated sub-images, see Figure 3.3.

Since we know that the stereo pair does not have a vertical disparity, we can rule out all such sub-images. If we know roughly how big the disparity is, we can make the search area even smaller. A problem here, though, is to choose the window size, which is a non-trivial problem itself. A large window means more calculations, since the convolutions per pixel increase. A small window might not include enough information to compare the surroundings properly. Choosing the window size can be done adaptively or through multiple scale algorithms. Multiple scale algorithms down-sample the original left and right images and calculate a disparity map for each scale in order to find both small and large disparities, and to get more information on the surroundings with a smaller window.

Another disadvantage with this method is that it cannot distinguish a step up from a step down, see Figure 3.4, since equation (3.1) only calculates differences. A step up or down, means an increase or decrease in pixel intensity. This results in correlation matrices looking like Figure 3.5 where we have two distinct peaks. These Figures typically occur when we have vertical lines in the stereoscopic images, for example the vertical branches in tree images. The problem here is to choose the correct one, i.e. to choose the “first” peak, and not the mirrored one. The correct peak might not have the largest value.

In Figure 3.6 we can see disparity maps for two sets of stereo pairs. The images on the right has been modified with a changed DC-level and filtering. We can clearly see the “layered” effect in the disparity maps. In the disparity maps
3.1 Correlation-based Techniques

Figure 3.1. Original stereo pairs.
Figure 3.2. Correlation searching

Figure 3.3. Correlation searching, horizontal

Figure 3.4. Step up and down.
3.2 Feature-based Techniques

In this group of methods, the image intensities are converted into a set of features which are assumed to be more reliable than the original intensities. Such features are corners, lines, local min/max etc. and are stored in a list of features for both images. The next step is to determine which feature in one picture corresponds to a certain feature in the other, the correspondence problem. Solving this includes adding constraints to the final solution. This method uses less calculations than correlation-based methods, but does not have unique correspondences between left and right images, due to the projection onto a subspace. This means that the technique only calculates the disparity between some of the pixels in each image. These disparity calculations are usually very accurate, however. The fact that we only get estimation for certain regions is not a good feature for our purposes, since we want to create different angles through interpolation. We need as many points as possible for these calculations.

It might be interesting to use this method in combination with another method, perhaps together with a phase-based method, to calculate the disparities along edges and corners for an enhanced disparity map. This method has not been implemented, but is an interesting topic for future enhancements of the disparity map calculations. Feature-based methods can be studied in Konrad and Lan[12] and Kovesi[13].

3.3 Local Polynomial Expansion

This method analyzes the local signal around every pixel in both pictures, and calculates parameters to describe the structure. These parameters approximate throughout the thesis, bright values represent objects close to the cameras, and dark values represent objects far away.

There are many ways to enhance the block-based correlation methods, to reduce computational complexity or to achieve sub-pixel accuracy. Read more about this in Falkenhagen[3].

Figure 3.5. Cross-Correlation matrices for pixels in tree images.
Figure 3.6. Disparity maps using correlation.
3.4 Phase-Based Techniques

polynomials of appropriate degree and determine its coefficients. For example, a Gaussian window can be applied around a pixel to get weighted values from the nearby pixels. In practice, at least a 9x9 pixel window is required for a good result. After the coefficients have been determined, the disparity can be calculated by solving a least-squares problem.

This method usually has high detail, but lots of noise. The greatest advantage of this method is that it is very fast. The method is explained in Farnebäck[4]. Assuming that the local signal can be approximated by a polynomial, one way to determine the coefficients of this polynomial is to do a MacLaurin expansion around every pixel. A second order MacLaurin series is given by

\[
I(x, y) = I(0, 0) + \frac{\partial I}{\partial x}(0, 0) \cdot x + \frac{\partial I}{\partial y}(0, 0) \cdot y + \frac{\partial^2 I}{\partial x^2}(0, 0) \cdot x^2 + \frac{\partial^2 I}{\partial y^2}(0, 0) \cdot y^2 + \frac{\partial^2 I}{\partial x \partial y}(0, 0) \cdot xy + O((x^2 + y^2)^{\frac{3}{2}}).
\] (3.2)

We assume that the local signal can be approximated as

\[
p(x, y) = r_1 + r_2 x + r_3 y + r_4 x^2 + r_5 y^2 + r_6 xy.
\] (3.3)

Then determine the coefficients \( r_k \) to minimize

\[
\sum_{x,y} [w(x, y)(I(x, y) - p(x, y))]^2.
\] (3.4)

Disparity maps for Pentagon and tree images can be seen in Figure 3.7. As we can see, the results are very noisy, even after smoothing.

3.4 Phase-Based Techniques

These methods derive Fourier-phase images from the intensity data. It can be considered as a local contrast equalization reducing the effect of many intensity variations between the two images. Fleet and Jepson[7], [6] studied the behavior of the local phase and its derivative, instantaneous frequency, extensively. This can also be studied in Hansen et al.[10]. These methods are usually a bit harder to implement, due to their more complex mathematics, but require less calculations than correlation-based methods. They also have sub-pixel accuracy. These methods are most commonly used in disparity estimation today. A problem with these methods is the fact that edges get distorted or “smeared out”. These effects can be reduced by using some edge enhancing methods after the disparity has been calculated, or by preserving the edges while calculating the disparity. It is difficult to give an intuitive explanation of how the phase-based methods extract disparity, but it can be seen as a comparison of local frequency in the 1D-signals (lines) in the images. Where the phase and frequency match, we have found the corresponding pixels between the images. A phase-based method for disparity estimation combined with an energy-based method is explained in Huang and Dubois[11].
Figure 3.7. Disparity maps using local polynomial expansion.
3.4 Phase-Based Techniques

The local phase has a number of interesting invariance and equivariance properties:

- Local phase estimates are invariant to signal energy.
- Local phase estimates and spatial position are equivariant.
- The spatial derivative of local phase estimates is equivariant with spatial frequency. The phase derivative is called local or instantaneous frequency, $\omega$.

The local signals can be written as

$$f_l(x) = DC + A \cos(x + \phi_l)$$
$$f_r(x) = DC + A \cos(x + \phi_r)$$

We assume that the local signal around the same pixel in left and right images only differs by a separation of the vector $d$.

$$f_l(x) = f_r(x + d)$$

According to the shift theorem of the Fourier transform, a shift in the spatial domain transforms into a modulation in the frequency domain.

$$f(x + d) \longleftrightarrow F(i\omega)e^{i\omega d}$$

The disparity can then be calculated from the phase difference and the local frequency in $x$-direction. The frequency in $y$-direction is not interesting due to the camera setup.

$$d(x) = \frac{\phi_l(x) - \phi_r(x)}{\omega} = \frac{\Delta \phi}{\omega}$$

The phase can be calculated with a 2D quadrature filter. This is a filter that satisfies

$$H_{edge}(u) = \begin{cases} -iH_{line}(u), & \text{if } u < 0 \\ iH_{line}(u), & \text{if } u \geq 0 \end{cases}$$

The filter response in the left and right images, respectively, is given by

$$-q_l = A_l e^{i\phi_l},$$
$$-q_r = A_r e^{i\phi_r}.$$
Methods for Disparity Estimation

Figure 3.8. Disparity and certainty maps for Pentagon images in five scale, one iteration, generated in 1.9 seconds in Visual Studio 2005 on two thread AMD Athlon.

The phase difference can be estimated in four ways, as combinations of left and right derivative in the left and right images,

$$\omega_x = \arg[q(x + 1, y)q(x, y)].$$  \hspace{1cm} (3.15)

The weighted mean of these four phase differences is a better estimate of the local frequency. The mean is calculated from $q_l q_r$ and not from $\arg(q_l q_r)$.

It is interesting to know how good our estimate really is. When we know this we can calculate the weighted means of the accumulated disparity, i.e. how much we are going to trust our disparity estimates. The parameter $c$ (certainty) can be weighted together with the accumulated $c$ over the different scales, as with

$$c_1 = |q_l q_r|^{0.5} \left[ \frac{2|q_l q_r|}{|q_l|^2 + |q_r|^2} \right]^4$$  \hspace{1cm} (3.16)

$$c_2 = \cos\left(\frac{\Delta \phi}{2}\right)^2$$  \hspace{1cm} (3.17)

$$c_3 = \left| \frac{1}{4} \cdot \left( e^{i\arg(f_l^+)} + e^{i\arg(f_l^-)} + e^{i\arg(f_r^+)} + e^{i\arg(f_r^-)} \right) \right|^4$$  \hspace{1cm} (3.18)

$$c_4 = c_3 \geq 0$$  \hspace{1cm} (3.19)

$$c = c_1 \cdot c_2 \cdot c_3 \cdot c_4.$$  \hspace{1cm} (3.20)

The different scales mean that the original stereo pair is down sampled with a scale factor into several new, smaller, images. The scale factor is always a power of two. The algorithm starts in the coarsest scale and runs through to the finest scale. This entire sequence can be iterated several times in order to achieve a better estimate, which is called number of iterations, see Algorithm 1.

After each cycle in the algorithm, it is necessary to smooth the disparity map. This can be done with a Gaussian filter, symmetrical or asymmetrical. The best
3.4 Phase-Based Techniques

results come from an asymmetrical Gaussian filter with more smoothing in the vertical direction. This gives the best result because we do not want to smooth out edges too much in the horizontal direction. The Gaussian filter is calculated either with a symmetrical filter,

\[ h_{x,y} = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}} \]  \hspace{1cm} (3.21)

or with separate filters for horizontal

\[ h_x = \frac{1}{\sqrt{2\pi\sigma_x}} e^{-\frac{x^2}{2\sigma_x^2}} \]  \hspace{1cm} (3.22)

and vertical

\[ h_y = \frac{1}{\sqrt{2\pi\sigma_y}} e^{-\frac{y^2}{2\sigma_y^2}}. \]  \hspace{1cm} (3.23)

directions, which enables us to use different sizes and smoothing factors, \( \sigma \), in the different orientations.

An example of disparity and certainty maps for the Pentagon images is shown in Figure 3.8. The bright values in the certainty map means high certainty. It is clear that the best areas are the areas with vertical edges and defined structures. The horizontal edges are much harder to find, because the correspondence search is done horizontally.

**Data:** Stereoscopic images

**Result:** Disparity map initialization;

for All scales do
  Down sample left and right images;
  Calculate complex filter responses;
end loop;

for All scales do
  for Iterations do
    Warp filter responses, \( q \), according to accumulated disparity;
    Calculate \( \Delta \phi \) and \( \omega_x \), (local phase and frequency);
    Calculate \( c \) and \( d \), (certainty and disparity matrices);
    Filter (convolve) \( d \) and \( c \);
    Add \( d \) and \( c \) to accumulated \( d \) and \( c \), respectively;
  end
  Interpolate accumulated \( d \) and \( c \) to fit next scale level;
end

**Algorithm 1:** Phase-based disparity estimation algorithm.

When the algorithm has reached the final state, one disparity map and one certainty map have been calculated. These maps can be considered to be placed between the original stereo images. This is due to the fact that the left and right images are warped half way towards each other when the disparity and certainty
are calculated. This means that when we calculate the interpolated images (in the visualization stage, chapter 4) we have to create two separate disparity maps, see Figure 4.2.

To see examples of disparity maps calculated with this phase-based technique see Figure 3.9. In the bottom part of the corridor image we can see a repetitive pattern which is due to the fact that the checked floor has large areas with the same intensity. In these areas there is hard to find any disparity.

3.4.1 Methods for enhancing disparity maps

Feature Detection

This is a very interesting topic for enhancing the disparity maps from the phase-based method. The reason is that there often is poor disparity estimation certainty in areas where there are horizontal lines. One example is the upper part of Pentagon, where the certainty is very low, see Figure 3.8(b). Here, we could use a feature detecting disparity estimation method to find these horizontal lines and calculate the disparity. This disparity can then be weighted together with the original phase-based method in the same areas.

Magnitude Information

The first phase-based algorithms used Gabor filters where the disparity was only evaluated for one frequency. As an enhancement, a FFT-based algorithm was introduced, where the phase differences between a pixel in two images was calculated for several frequencies. Performing a FFT also yields magnitude information, which can be used in combination with the phase information to further improve the disparity estimation. This is explained in Ahlvers and Zölzer[1]. Other methods for accuracy improvements can be found in Ouali et al.[16].

Monogenic Phase

This method replaces the classical quadrature filters with spherical quadrature filters. The resulting estimation formula is more flexible than the one based on horizontal quadrature filters. The disparity estimation is less noise sensitive, and edges in the disparity maps are preserved as well as for more complex methods, such as the canonical correlation method. This method is described in Felsberg[5].

 Colour Information

An interesting topic in enhancing disparity maps is to perform separate calculations for the three different colour components, Red, Green and Blue. Since different image depths often (but not always) have different colours this could be a very simple way to extract further depth information from the stereoscopic images. The way the calculations are performed now is to create the mean from the RGB components which means the calculations are always performed on a black and white image. The obvious disadvantage of performing separate calculations
Figure 3.9. Phase-based disparity estimation
for each colour component is that the disparity calculations will take three times as long.

Figure 3.10. 3D-plot of Pentagon disparity map.
Chapter 4

Methods for Visualization

This chapter considers methods for visualizing images on the 3D-screen from the depth maps and the original stereoscopic images. We can see two different ways to do this, making an entire 3D-object from the depth maps, or creating an image sequence with the object observed from several angles. It is important to keep in mind that the visualization aims to show a scene from certain predetermined angles, and is not to be interacted with.

4.1 Creating a 3D-object

The intuitive way of visualizing an object is to create a true 3D-object from the disparity maps. This however, requires a very good disparity map in order to make a decent 3D-object. Furthermore, the 3D-object must be translated into images to be presented on the screen. This means lots of computing and a more difficult implementation without much gain. Therefore, creating a 3D-object has not been implemented or investigated further.

4.2 Image sequencing

The image sequencing is not that hard to implement using interpolation from the depth maps. It is also easy to change the number of angles that are to be covered, and exactly which angles. See Figure 4.1. This is very useful for presentation on the 3D-display. This has been implemented in a straightforward way, where the images are calculated from warping the stereo pair with the disparity map. First however, the disparity map is warped according to itself in order to create left and right disparity maps. The reason for this is intuitive, but is also explained in Figure 4.2. After this, the left and right images are warped according to their disparity maps to create two images at the same position in between (or outside) the original pair. The interpolated image is then formed as a weighted sum of these two images. The image at position \( p \in [0, 1] \) between the stereo pair is calculated as
Methods for Visualization

Figure 4.1. Interpolation scheme, dashed - interpolated images

\[ I_p = \frac{I_{\text{left}_p} \cdot p + I_{\text{right}_p} \cdot (1-p)}{2}, \tag{4.1} \]

where

\[ I_{\text{left}_p} = \text{warp}(I_{\text{left}}, p_d_{\text{left}}) \tag{4.2} \]
\[ I_{\text{right}_p} = \text{warp}(I_{\text{left}}, (1-p)d_{\text{right}}). \tag{4.3} \]

**Data:** Stereoscopic images, disparity map(s), number of images

**Result:** Image sequence

initialization;
Warp disparity map to match left and right images;
loop;
for Number of images between pair do
  Warp left and right images according to left and right disparity maps, respectively;
  Weigh together contributions from left and right images to create an angle;
end
for Number of images outside pair do
  Warp left and right images according to left and right disparity maps, respectively;
  Weigh together contributions from left and right images to create an angle;
end

**Algorithm 2:** Visualization algorithm.

Example of visualization of real images, taken without a stereoscopic camera rig, can be seen in Figure 4.3(b). This shows that we can get good results without sophisticated camera equipment.

More reading on intermediate view interpolation can be found in Zitnik et al.[18].
4.2 Image sequencing

**Figure 4.2.** Warped disparity maps.

(a) Interpolated middle image.  
(b) Interpolated middle image.

**Figure 4.3.** Interpolated middle images for cars images and hands images.
4.3 PSNR

A method for making quality measurements for comparison between different methods is calculating the PSNR, or Peak Signal to Noise Ratio. The PSNR can be calculated when we have at least three images in a sequence. Then we can make a disparity map between images one and three, calculate an interpolated middle image, and compare this to the original middle image. PSNR is a measurement which should be interpreted with care, since the results may vary within an image, and gives little information of how good a method will look on the 3D-screen. It is difficult to find a really good quality measurement, since the most accurate calculations might not give the best results and vice versa. However, the PSNR is a good measurement to compare images with different parameter settings, together with an optical inspection.

PSNR is calculated as follows, where $\max_{\text{pixel}}$ is the maximum pixel value in the associated colour map, normally 255. Typical PSNR values range between 20 and 30 decibel. The PSNR has been calculated for Flower Garden images, see Table 4.1, using the three different methods for disparity calculations.

\[
MSE = \frac{1}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} [f(i,j) - F(i,j)]^2 \quad (4.5)
\]

\[
PSNR = 20 \cdot \log_{10} \left( \frac{\max_{\text{pixel}}}{\sqrt{MSE}} \right) \quad (4.6)
\]
4.4 Slicing

Slicing means in this thesis that the interpolated images are cut and put together in a certain order to fit the 3D-screen. This means that we don’t need all of the interpolated images, only between half and a fifth of each image depending on the 3D-setup. This is due to the fact that the 3D-screen only shows certain views of the images at each time. In a real time implementation this is very important since we can reduce the computational cost of the visualization dramatically. The slicing part is closely related to Setred’s other software and hardware and will not be investigated further here.
Chapter 5

Real Time Implementation

Since the Matlab implementation is too slow for any real-time demands, the project also includes a comparison of different low-level implementations. The final implementation should be able to handle high real-time demands. Ideally, this means to calculate disparity maps for several image pairs (2-5), and create about 80 interpolated images, slice them into about 20 images in 1024x768 resolution, at 20-30Hz. Whether this is possible or not will be investigated. In Figure 5.1 we can see the different runtimes for the different parts of the calculations. The most time demanding parts are warping and calculating the filter responses. Almost all of the time spent in visualization is warping. After this, we also need to improve the different stages of the disparity calculations, to improve calculations times for the finest scale. This leads us to believe that the most important thing to speed up is warping. This is easily done on a GPU which is perfect for our purposes.

5.1 C Implementation

The entire Matlab algorithm for calculating the phase-based disparities and visualizations has been implemented in C. This was a large portion of the work in this thesis. The reason for this is to later achieve real-time performance. A block diagram of the different parts of disparity calculations and a plan of C functions can be seen in Figure 5.2. Programming math and signaling in C is very different from programming Matlab. In C, all the mathematical functions was written from the beginning. Examples of such functions and structures are complex operators, Cartesian separated Gaussian filters and convolutions. More reading in Appendix A. These functions could have been found in a C math library, but for control and compatibility reasons, they were rewritten.

The code is currently using FFTW (Fastest Fourier Transform in the West) for its Fourier transforms, but we may switch to Kiss FFT due to licensing reasons.
5.1.1 Performance

The performance of the C program is currently running considerably faster than the Matlab implementation. For optimized C code, we expect it to be several times faster than the unoptimized C code, and with warping moved to a GPU probably another 2-3 times faster. Today, we can calculate a disparity map from images of 512x512 pixels, and visualize them in 18 images in under four seconds. For optimized C code and warping in a GPU, we will most likely be able to do this in well under a second. In table 5.1 we can see the run times for calculating the disparity map and visualizing it in 20 images for Pentagon images, 512x512 pixels, in five scales with two iterations per scale. For higher demands, we can run only one iteration per scale which will reduce the computational cost by another third. This will reduce the quality of the disparity maps, but not very much. This can be seen in Figure 5.3. To get a good quality visualization on the 3D-screen, we need to create an image sequence of more than 18 images, probably around 80, which will increase the visualization time by a factor of four.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Celeron 450MHz</td>
<td>107.9</td>
<td>62.7</td>
<td>45.3</td>
<td>10.6</td>
</tr>
<tr>
<td>AMD 3500+ VS2003</td>
<td>-</td>
<td>-</td>
<td>9.3</td>
<td>1.7</td>
</tr>
<tr>
<td>AMD 3500+ VS2005</td>
<td>-</td>
<td>-</td>
<td>6.6</td>
<td>1.2</td>
</tr>
<tr>
<td>AMD 4600+ VS2003</td>
<td>-</td>
<td>-</td>
<td>6.6</td>
<td>1.8</td>
</tr>
<tr>
<td>AMD 4600+ VS2005</td>
<td>-</td>
<td>-</td>
<td>5.2</td>
<td>1.1</td>
</tr>
<tr>
<td>AMD 4600+X2 VS2005</td>
<td>-</td>
<td>-</td>
<td>3.2</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 5.1. Run times (in seconds) for Pentagon images in five scales and two iterations with optimized Visual Studio. Visualizing in 20 angles.
5.1 C Implementation

![Block Diagram of C functions for phase-based disparity calculations.](image)

**Figure 5.2.** Block Diagram of C functions for phase-based disparity calculations.
5.1.2 Optimization

The C code can be optimized for real-time applications by using different techniques. The first step may be to rewrite a lot of the code into a “pipeline” setup. Right now all complex filters are initialized before the disparity loop starts. This means allocating and reallocating lots of memory along the algorithm before it is used. In a pipeline setup, we can allocate a lot of memory at the start, and then reuse it every iteration. This will save time since we don’t need to allocate and release memory as many times. The application can also be optimized by using multiple-thread CPUs. A first look at such optimizations has been done on a AMD Athlon 4600+ X2 in Visual Studio 2005. By using the command 

#pragma omp parallel for

before for-loops, see Appendix A.1, the loops can be calculated on two threads at the same time, speeding up these calculations by (optimally) two times. Run times can be studied in table 5.1.

5.2 GPU Functions

After the C implementation is done, some of the functions will be moved to a GPU. This because a GPU will be able to calculate warping and sampling/interpolation faster than a C program. Another advantage is that the warping can be done on the GPU at the same time as the next disparity map is calculated on the CPU. This means that we can reduce the disparity calculations by about a third.

There is good knowledge at Setred about this as well as in C programming. Due to time limitations it will not be done within this thesis work. The functions to be written here will be determined after an evaluation of the C program. The first step however is to write a function that warps a matrix (image) according to a disparity map. The reason why this function is interesting is that a GPU is specialized for these kinds of calculations. The programming here will be done in OpenGL and OpenGL Shader Language.

In Figure 5.4 we study different processing schemes for presenting a stereo pair...
5.3 Hardware Implementation

An FPGA implementation of the disparity estimation and visualization will take a long time to develop and is not a part of the thesis. However, this is a very interesting topic to investigate in future work. Hardware implementations can be
designed to perform calculations on large matrices, such as images and disparity maps very quickly. A master thesis calculating disparity maps using FPGAs is explained in Darabiha[2]. In future work, this is to be investigated after GPU functions for visualizing have been implemented. With an FPGA implementation for calculating disparity maps and visualizing it is very likely that we will accomplish the real time demands.

**Figure 5.5.** Diagram of computing sequences when Slicing is done on GPU.
Chapter 6

Conclusions

In this thesis I have investigated methods to visualize stereoscopic images on a 3D-display in real time. With the first block-based methods, it seemed highly unlikely that real time demands could be achieved, but later in the project, this early scepticism changed. We can now conclude that it is possible to accomplish the real time demands. With the unoptimized C functions used in this thesis we have reached promising calculations speed, and with the changes proposed in chapter 5.1 real time performance can probably be reached with acceptable quality. In this chapter follows conclusions on the different parts of the thesis.

6.1 Disparity Estimation

The conclusion regarding which method to use in estimating dense disparity maps is clear, the phase-based methods have proven significantly better than the other methods. The local stucture-based methods are very fast, but suffer from poor quality in the disparity maps. The block-based methods tested here are too slow.

The phase-based method results in disparity maps with slightly blurry edges, see Figures in section 3.4. These can probably be enhanced with feature detecting disparity estimation methods. A combination of these methods may be the best way to create dense disparity maps. One reason for why this has not been implemented is that it is not very important to improve the quality of disparity maps. The current maps give good visualization results. The most important thing is that the disparity maps are robust, and that there are no large textured areas where the disparity is wrong.

- Phase-based disparity estimation is the best algorithm for our purposes.
- The disparity maps can be enhanced by a feature detecting algorithm, but they do not have to be improved very much.
6.2 Visualization

The visualization is very easy to perform with the image sequence method. There really is not much more to it than equations (4.1) and (4.2). This simplicity together with the fact that this can be computed quickly with a GPU means that there is no need to create a 3D-object. This method also has an error correcting characteristic since large areas with the same texture can come out very well in the resulting interpolated images. When the stereo pair lack structure in large areas of the images, such as a large blue sky, the disparity maps can become very poor. In these places we can estimate a large or small disparity and this is hard to control. This however, is quite fine with the image sequence warping method, since the images have the same colours in these areas! This means that even though we might warp the images too much or too little, it will not show in the resulting images.

- Visualization will be done by warping images directly into an image sequence.

6.3 Future work

The future work is about optimizing and speeding up the calculations, in order to achieve real time performance. There is also some work involved in capturing images in real time from webcams and sending the results to the 3D-display, however this should not pose any significant problems.

The optimization should start with rewriting the code in a pipeline setup, where we allocate as little memory as possible at all times. This will also result in smaller, more specified function calls. The next thing to do is to move warping and resizing functions to a GPU and to make the GPU calculate in parallel to the CPU. This is where a lot of time will be saved, at least the time required to make the visualizations. After this, it’s necessary do a complete profiling to determine where the remaining time is spent, and make optimizations where possible.

- Rewrite the C code into a pipeline setup.
- Move the warping and slicing to the GPU.
- Calculate on the CPU and GPU in parallel.
- Profile and optimize the code.

6.4 Applications

There are several areas where stereo visualization can be useful together with a 3D-screen. Some are where we want 3D in real time, such as remotely operated vehicles where it is necessary to have a good perception of the surroundings and there are high demands on interactivity.

Other areas where the results of this thesis are interesting is in stereo webcams. Here, the 3D-display itself is not necessary, but the ability to generate an interpolated center image is. The reason is that when web cameras are used in video calls
people are not able to look directly at each other. This is due to the fact that the camera is located beside the screen for obvious reasons. With two cameras and interpolation software, we could place one camera on each side of the screen and send the interpolated middle images (where the person is looking straight ahead) to the recipient.

- Remotely Operated Vehicles (ROVs).
- Stereo webcams.
Bibliography


Appendix A

C code examples

A.1 Complex numbers

The code uses lots of complex data which are handled in the Complex structure. The complex information, together with magnitude and angle information is stored in the struct Data.

```c
struct Data {
    float im;
    float re;
    float ab;
    float an;
};
struct Complex {
    int width;
    int height;
    Data* pixels;
};
```

This structure is operated on by many functions, one of which is the Angle function. This calculates the angle and magnitude of a complex array and stores the results within the struct.

```c
void Angle(Complex* cmp)
{
    #pragma omp parallel for
    for(int y = 0; y < cmp->height; y++){
        for(int x = 0; x < cmp->width; x++){
            int i = x + y * cmp->width;
            cmp->pixels[i].an =
                (float)atan2f(cmp->pixels[i].im,cmp->pixels[i].re);
        }
    }
}
```
cmp->pixels[i].ab = (float)sqrt(pow(cmp->pixels[i].im,2.0f) + pow(cmp->pixels[i].re,2.0f));
}
}

A.2 Gauss filters

The C code also uses some Gauss filters which are handled by the Gauss struct. This struct can handle asymmetrical, cartesian separated Gauss filters to speed up the convolutions.

```
struct Gauss {
    int width;
    int height;
    float* kernel;
};
```

The function for calculating the separated Gauss filter kernels. The vertical and horizontal kernels are stored after each other in one array.

```
void CreateGaussSeparated(Gauss* filt, int Nx, int Ny, int sigmax, int sigmay) {
    filt->height = Ny;
    filt->width = Nx;
    filt->kernel = (float*)malloc((filt->width + filt->height) * sizeof(float));
    int nx=(filt->width - 1)/2;
    int ny=(filt->height - 1)/2;
    for(int x = -nx; x < nx + 1; x++){
        int k = (x + nx);
        float i = (float)(-(x * x)) / (2.0f * sigmax * sigmax);
        float v = (float)exp(i) / (2.0f * (float)PI * sigmax);
        filt->kernel[k]=v;
    }
    for(int y = -ny; y < ny + 1; y++){
        int k = (y + ny) + filt->width;
        float i = (float)(-(y * y)) / (2.0f * sigmay * sigmay);
        float v = (float)exp(i) / (2.0f * (float)PI * sigmay);
        filt->kernel[k] = v;
    }
}
```
A.3 Bitmaps and Convolutions

The most important struct is the `Bitmap` struct, which is used for storing images throughout the code. The struct stores the colour information in a separate struct, `Color`, see below.

```c
struct Color {
    unsigned char b;
    unsigned char g;
    unsigned char r;
};
struct Bitmap {
    int width;
    int height;
    Color* pixels;
};
```

The function for performing the separated convolutions on a bitmap, using the separated Gauss filter kernels from function `CreateGaussSeparated`.

```c
void ProcessBitmapSeparated(Bitmap* bmp, Bitmap* bmpInter,
                             Bitmap* bmpOut, Gauss* filt)
{
    if(!bmp->pixels)
        return;
    int kx, ky, x, y, max;
    float d1, d2, sumg, sumr, sumb;
    max = 255; //Max pixel value
    d1 = 0;
    d2 = 0;
    for(int n = 0; n < filt->width; n++)
        d1 += filt->kernel[n];
    for(int n = filt->width; n < filt->width + filt->height; n++)
        d2 += filt->kernel[n];
    int nx = (filt->width - 1) / 2;
    int ny = (filt->height - 1) / 2;
    //Horizontal convolution
    for(y = ny; y < bmp->height - ny; y++){
        for(x = nx; x < bmp->width - nx; x++){
```
sumb = 0;
sumg = 0;
sumr = 0;
int i = x + y * bmp->width;
for(kx = 0; kx < filt->width; kx++){
    int ki = kx;
    int j = i + (kx-nx);
    sumb = sumb + bmp->pixels[j].b *
        filt->kernel[ki];
    sumg = sumg + bmp->pixels[j].g *
        filt->kernel[ki];
    sumr = sumr + bmp->pixels[j].r *
        filt->kernel[ki];
}
sumb = sumb/d1;
if(sumb < 0) sumb = 0;
if(sumb > max) sumb = (float)max;
sumg = sumg/d1;
if(sumg < 0) sumg = 0;
if(sumg > max) sumg = (float)max;
sumr = sumr/d1;
if(sumr < 0) sumr = 0;
if(sumr > max) sumr = (float)max;
bmpInter->pixels[i].b = (unsigned char)sumb;
bmpInter->pixels[i].g = (unsigned char)sumg;
bmpInter->pixels[i].r = (unsigned char)sumr;
}

//Vertical convolution
for(y = ny; y < bmp->height - ny; y++){
    for(x = nx; x < bmp->width - nx; x++){
        sumb = 0;
        sumg = 0;
        sumr = 0;
        int i = x + y * bmp->width;
        for(ky = 0; ky < filt->height; ky++){
            int ki = filt->width + ky;
            int j = i + (ky - ny) * bmp->width;
            sumb = sumb + bmpInter->pixels[j].b *
                filt->kernel[ki];
            sumg = sumg + bmpInter->pixels[j].g *
                filt->kernel[ki];
            sumr = sumr + bmpInter->pixels[j].r *
                filt->kernel[ki];
        }
        sumb = sumb/d2;
A.4 FFT-shift function

Function ShifterCmp does the same thing as fftshift does in Matlab, which is to place the zero-frequency component in the center of the spectrum. In this case, for images, ShifterCmp switches quadrant one and three, and quadrant two and four. The same function also exist for struct Bitmap.

The function is called before and after the Fourier transformation, which takes place in the filter initializing stage of the algorithm.

```c
void ShifterCmp(fftwf_complex* cmp, int rows, int cols)
{
    fftwf_complex *cmpTemp;
    cmpTemp = (fftwf_complex*) fftwf_malloc(
        sizeof(fftwf_complex) * rows * cols);
    int center = cols * (rows / 2);
    for(int rowStart = 0; rowStart < center; rowStart += cols){
        //First/third quadrant
        memcpy(&cmpTemp[center + cols / 2 + rowStart],
               &cmp[rowStart], (cols / 2) * sizeof(fftwf_complex));
        memcpy(&cmpTemp[center + rowStart],
               &cmp[cols / 2 + rowStart], (cols / 2) * sizeof(fftwf_complex));
        //Second/fourth quadrant
        memcpy(&cmpTemp[rowStart + cols / 2],
               &cmp[center + rowStart], (cols / 2) * sizeof(fftwf_complex));
        memcpy(&cmpTemp[rowStart],
               &cmp[rowStart], (cols / 2) * sizeof(fftwf_complex));
    }
}
```
&cmp[center + cols / 2 + rowStart], (cols / 2) * sizeof(fftwf_complex));
}

// Copy to original memory
memcpy(cmp, cmpTemp, rows * cols * sizeof(fftwf_complex));
fftwf_free(cmpTemp);
}
Appendix B

Matlab code examples

B.1 PSNR function

This function calculates the PSNR between two images, \texttt{image1} and \texttt{image2} as described in section 4.3.

\begin{verbatim}
function [psnr error] = PSNR(image1,image2,maxValue)
    size1 = size(image1); % size(image1) = size(image2)
    MSE = sum((image1(:) - image2(:)) .^ 2)
        / (size1(1) * size1(2));
    psnr = 20 * log10(maxValue / sqrt(MSE));
    error = 2 .* (image1 - image2) + 128;
\end{verbatim}

B.2 Visualization

This function visualizes several disparity maps and stereoscopic images by warping them according to each other.

\begin{verbatim}
function [imageComplete] = visualize(imagesIn,disparity,N);
    % Determine number of disparity maps
    s1 = size(disparity);
    disparityN = s1(3);
    % Allocate memory
    image = zeros([s2(1:2) N * disparityN - disparityN + 1]);
    imageLeft = zeros([s2(1:2) N]);
    imageRight = zeros([s2(1:2) N]);
    % Images between the stereo pairs
    for disp = 1:disparityN
\end{verbatim}
for k = 1:N
    p = (k - 1) / (N - 1);
    image(:,(disp - 1) * (N - 1) + k) = 
        (1 - p) * warp(imagesIn(:,disp),
        p * disparity(:,disp)) +
    p * warp(imagesIn(:,disp + 1),
        -disparity(:,disp) * (1 - p));
end
end
% Images outside of all the pairs (runs only once)
for k = 1:N
    q = k / (N - 1);
    imageLeft(:,k) = 0.8 * warp(imagesIn(:,end),
        disparity(:,end) * q)+
    0.2 * warp(imagesIn(:,end - 1),
        disparity(:,end) * (1 + q));
    imageRight(:,k) = 0.8 * warp(imagesIn(:,1),
        -disparity(:,1) * q) +
    0.2 * warp(imagesIn(:,2),
        -disparity(:,1) * (1 + q));
end
% Flip imageRight to left-right
imageRight = flipdim(imageRight,3);
% Dimension of imageComplete
dimsComp = (N - 1) * disparityN + 1 + 2 * (ceil(N / 2) - 1);
% Allocate
imageComplete=zeros([s2(1:2) dimsComp]);
% Concatenate image sequences
imageComplete=cat(3,imageRight,image, imageLeft);
Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera åktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Oskar Thulin