Accurate Real-Time Multi-Camera Stereo-Matching on the GPU for 3D Reconstruction

Klaus Denker
HTWG Konstanz, Germany
kdenker@htwg-konstanz.de

Georg Umlauf
HTWG Konstanz, Germany
umlauf@htwg-konstanz.de

ABSTRACT

Using multi-camera matching techniques for 3d reconstruction there is usually the trade-off between the quality of the computed depth map and the speed of the computations. Whereas high quality matching methods take several seconds to several minutes to compute a depth map for one set of images, real-time methods achieve only low quality results. In this paper we present a multi-camera matching method that runs in real-time and yields high resolution depth maps.
Our method is based on a novel multi-level combination of normalized cross correlation, deformed matching windows based on the multi-level depth map information, and sub-pixel precise disparity maps. The whole process is implemented completely on the GPU. With this approach we can process four 0.7 megapixel images in 129 milliseconds to a full resolution 3d depth map. Our technique is tailored for the recognition of non-technical shapes, because our target application is face recognition.

Keywords
Stereo-matching, multi-camera, real-time, gpu, computer vision.

1 INTRODUCTION

Stereo matching is a technique to compute depth information of a captured object or environment from two or more 2d camera images. Many applications ranging from remote sensing to robotics, archeology, cultural heritage, reverse engineering, and 3d face recognition [15, 17, 10, 26] use stereo matching. It is the only passive method to generate depth information. This means there is no artificial interaction with the object that might do any harm and only natural light is used for the data acquisition.

The main challenge of stereo matching is the trade-off between the quality of the depth map and the computation time to compute the depth map. For some applications a real-time computation is not important. So many stereo- and multi-view-matching methods focus on high quality results instead of fast computation times. These high quality methods need at least several seconds to compute a single depth map from one set of images [9]. However, for robotics faster computation times are more important than the quality of the depth map. This led to the development of GPU based real-time matching methods [28, 27].

Our target application is 3d face recognition. For face recognition the requirements are somewhere between these fields. A trade-off between a high depth map quality and an acceptable speed must be found. The whole reconstruction and recognition needs to be done in less than half a second. A longer delay is not acceptable for the captured person. Nevertheless, the quality of the reconstructed surface needs to be high enough for a reliable recognition of the person.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1.1 Overview and contribution

In order to classify our approach for the subsequent related work section we give here a brief layout of our system. It is based on weighted normalized cross-correlation for all matching windows of a reference image to a set of additional images from different perspectives. This cross-correlation yields a score for every matching window position and the maximal score indicates the best matching position. This best matching position corresponds to a disparity of the matching windows and thus to the depth information. These steps will be described in Sections 3 - 4. Our contribution in this process is the GPU optimized use of weighted normalized cross-correlations, the combination of multiple cameras to a total score for simultaneously moved matching window, a projection-free depth-map-based deformation of the matching windows, and a sub-pixel precise disparity estimation. These techniques account for the quality of the generated depth maps. To compute the depth maps in real-time our process is implemented on the GPU. This is described in Section 5 and has not be done in such a consequent form before.

2 RELATED WORK

Our method may be classified between two very different classes of stereo matching methods. On the one hand, the high quality methods with long computation time to achieve excellent results. On the other hand,

the fast GPU methods using much simpler algorithms. Therefore, we will contrast our approach to both classes of stereo matching methods.

2.1 High quality methods

High quality stereo matching methods have been developed based on various techniques. The quality of such methods is compared at [19, 21, 25]. Newer benchmark results are available on the associated websites [20, 22, 24].

One of the earliest methods in this class is the adaptive least squares correlation of [6]. In this approach local affine transformations are estimated using a least squares approximation. Although, this method theoretically converges to an optimal solution, the convergence is too slow and the computation too costly due to the size of the linear systems.

Today, best reconstruction quality is achieved by region growing algorithms, e.g. [5, 9]. These methods are typical for high quality matching algorithms, where a set of good matches is generated using a sparse set of interesting features. Then, these good matches are extended with a growing strategy. The growing operations are iterated in combination with filter operations to control the quality of the matches. Because the growing process is based on an optimization of complex objective functions, these methods do not allow a fast GPU implementation.

A novel alternative is the phase only correlation of [23]. Here, the disparity of matching windows is estimated by the phase difference of the image signal along epipolar lines. This requires the computation of a Fourier transformation, which is difficult to implement on the GPU [14]. This is particularly problematic if the Fourier transform must be evaluated for every pixel of the captured image.

Global optimization of a Markov Random Field (MRF) is used in [1]. For each pixel multiple depth hypotheses are stored and the best is picked by the MRF optimization. The solution of this NP-hard problem is approximated using a sequential tree re-weighted message passing algorithm [11]. Although the GPU is used to solve several steps of the algorithm, the global optimization makes it much slower than typical GPU methods.

A particle cloud optimization is used by [8] to generate depth representations for each camera image. The particles are aware of depth discontinuous silhouettes and use a special volumetric view space parametrization instead of the usual image-based parametrization of matching windows. Then, these depth representations are combined and rendered in real-time using the GPU.

Approaches based on dynamic programming, e.g. [12, 18], are relatively similar to our approach. For these methods fields of matching scores are computed for every epipolar line. Within these fields an optimal path is computed using dynamic programming. The computations of the optimal path can either be done on the CPU or on the GPU requiring significant amount of memory.

Our approach is also based on matching scores along epipolar lines, but the computations are local and simple to allow an implementation on the GPU.

2.2 GPU methods

Much faster methods implement the stereo-matching algorithm on the GPU using hardware features of the graphics card like mip-mapping.

A typical example for this class of methods is described in [27]. This approach consists of a set of individual steps of the overall stereo-matching process implemented on the GPU. For the matching score either the sum of squared differences or the sum of absolute differences are used. These matching scores are easily implemented on the GPU, but yield only low quality disparity maps. To exploit the capabilities of the mip-map a pyramidal matching kernel is used, which does not allow for an independent movement of the individual levels in the pyramid. In both aspects our approach improves this method. Some other aspects of [27], like cross-checking and feature aligned matching windows, could easily be integrated to our system.

A different approach of the same first author is [28]. Here five calibrated cameras are matched at once. Using the same technique with a reconfigurable array of 48 cameras is described in [30]. For this technique the matching window covers only one pixel to simplify the computations on the GPUs. This local approach is not stable but very fast and avoids all disadvantages of large matching windows.

Another technique for a large number of images is [29]. It is not as fast as the other GPU methods, but includes a volumetric reconstruction of the objects. A plane sweep method is used for depth estimation on non-rectified images.

The method from [2] uses the pyramidal matching kernel and mip-mapping from [27] and adds a foreground/background separation on the GPU. This additional step avoids typical artifacts of the pyramidal kernel like wrong depth estimates for regions with low texture details usually found in the background. Our improved multi-level approach does not show such problems.

3 THE CAMERA SYSTEMS

We built a system of four USB Logitech® QuickCam® Pro 9000 cameras, see Figure 1(a). Each camera is used at a resolution of 960×720 at five frames per second. The cameras could yield a much higher resolution, but the bandwidth of the USB 2.0 controllers is limited.

To improve the quality for later face recognition, we built a second camera system of four Point Grey Flea®2 FireWire 800 cameras, see Figure 1(b). These cameras synchronously capture images at a resolution of 1392×1032 at 15 fps. For synchronization we use all four cameras on a single FireWire 800 Bus. Thus, in RGB mode only a frame rate of 3.75 fps is possible. This can be improved by de-mosaicking on the GPU and transferring the data in eight bit raw mode. This allows for 11.25 fps.

Our experiments showed that a Y-constellation of four cameras as shown in Figure 1 gives the best results. The image of the central camera is used as reference image for matching and texturing. Each possible image pair has a different angle. Otherwise preferred directions of the camera constellation could deteriorate the detection of features along these directions, e.g. an image containing horizontal stripes causes problems for horizontal camera arrangements.

Independently of the used hardware system, our method can be adapted to other camera constellations. This adaption is much easier for camera systems where all cameras are mounted on a plane perpendicular to the viewing direction. The individual camera images are rectified using a lens correction similar to [4].

4 MATCHING

The overall matching process consists of several nested loops shown in Figure 3. We describe this process from the inner to the outer loop.

4.1 Stereo matching

The aim of stereo matching is to find corresponding points in two images. Usually two square regions, called *matching windows* are compared. These windows are moved over the images to find the best matching position. To identify the best position, a score is computed, that rates the similarity of two matching windows. Similar to [13] we use a *weighted normalized cross-correlation* on RGB color values. First the weighted average color $\overline{f}_i$ of the matching window W_i in the i-th image is computed

$$\overline{f}_i = \sum_{(x,y)\in W_i} w(x,y) f(x,y). \tag{1}$$

Here $w(x,y) = \cos^2(\pi x/a) \cdot \cos^2(\pi y/a)$ is a weight function that smooths the result to emphasize pixels at the center of the matching window over pixels at the border, and a denotes the matching window size in pixels. Then the weighted auto-correlation α_i of each matching window with itself is computed as

$$\alpha_i = \sum_{x,y} w(x,y) \left[f_i(x,y) - \overline{f}_i \right]^2. \tag{2}$$

To evaluate the similarity of two matching windows W_i and W_j the weighted cross-correlation $\beta_{i,j}$ is computed

$$\beta_{i,j} = \sum_{x,y} w(x,y) \left[f_i(x,y) - \overline{f}_i \right] \cdot \left[f_j(x,y) - \overline{f}_j \right]. \tag{3}$$

The weighted normalized score $\gamma_{i,j}$ is computed as the weighted cross-correlation normalized by the geometric mean of the respective weighted auto-correlations

$$\gamma_{i,j} = \beta_{i,j} / \sqrt{\alpha_i \cdot \alpha_j}. \tag{4}$$

4.2 Multi-camera matching

Stereo matching evaluates the similarity of two matching windows. We extend this score to a set of n cameras and matching windows by summing up the weighted normalized scores of all possible image pairs. Thus, we need $n(n-1)/2$ stereo matching operations. To compute a total score we compute a camera score

$$\gamma_i = \sum_{j \neq i} \gamma_{ij} \tag{5}$$

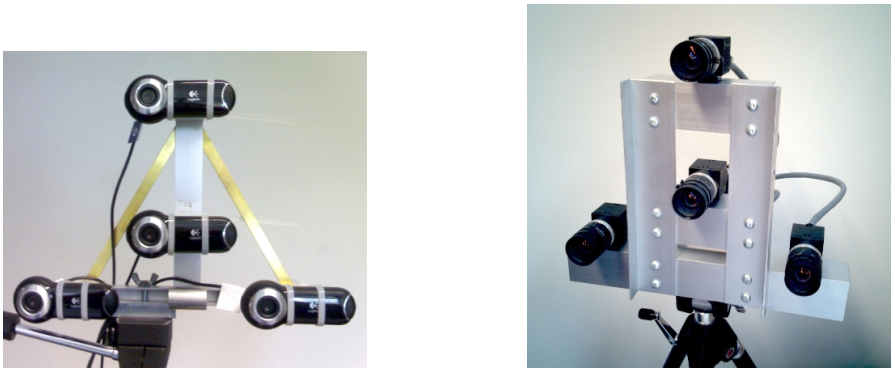
and a total score

$$\gamma = \sum_i \gamma_i - 2 \min_i \gamma_i. \tag{6}$$

This eliminates all scores from the worst matching camera to improve robustness to occlusion on one of the cameras. The total score is used to evaluate the similarity of matching windows of multiple cameras simultaneously.

4.3 Moving the matching windows

Between the images a disparity estimation is computed to get the depth information. Therefore, the matching windows are moved simultaneously over all images. A total score of each position and the best matching window position with the highest total score are computed. Since the evaluation of all possible positions is too expensive, the movement of the matching window is limited to the epipolar lines projected by the center point of the matching window of the reference image. The image of the central camera is used as reference image, i.e. the matching window on the central image is fixed. Figure 2 shows the simultaneous movement of the matching windows in the other images along the epipolar lines. These movements along the epipolar line have a step size of one pixel for our camera configuration. For other camera configurations the step size depends linearly on the distance to the central camera. We test $3 \leq k \leq 35$ different positions for each matching window, see Section 4.5. Note that the color values for the score computations are bi-linearly interpolated to allow an exact movement along the epipolar line. The best similarity of the matching windows is marked by the matching window position with the highest score



(a) USB camera system. (b) FireWire camera system.

Figure 1: For our experiments we use two systems of four cameras arranged in an upside down Y-constellation.

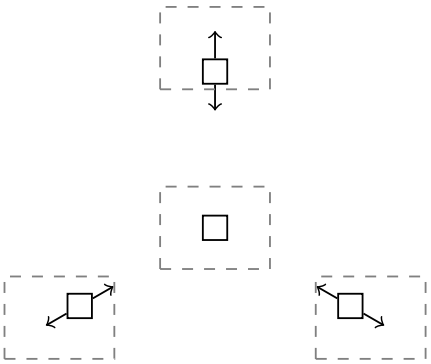


Figure 2: Moving the matching windows (solid squares) in all images (dashed rectangles) along epipolar lines (arrows) simultaneously.

s_{best}. From the position on the epipolar line, the disparity d_{best} of the best match is estimated. The real depth can be computed by reverse projection using the position of the reference camera, the distances to the other cameras, and the disparity.

4.4 Sub-pixel matching

To achieve sub-pixel precision for the disparity map we use a method similar to sub-pixel accurate edge detection of [3]. The best disparity is achieved at a local maximum of the total score, i.e. both neighboring scores s_{left} and s_{right} are smaller or at most one of them is equal to s_{best}

$$s_{\text{left}} \leq s_{\text{best}} > s_{\text{right}} \quad \text{or} \quad s_{\text{left}} < s_{\text{best}} \geq s_{\text{right}}. \qquad (7)$$

Interpolating these three total scores with a quadratic polynomial yields a best sub-pixel score at the global maximum of the quadratic polynomial. This maximum is achieved within half the distance to the neighbor positions. The position of this maximum is the interpolated sub-pixel disparity d_{sub}.

4.5 Multi-level matching

Our method generates disparity data for one image at a fixed resolution. To allow large disparities, many possible matching window positions must be evaluated. Because this is computationally expensive, we use a real multi-level approach that can reduce the effort for large disparities. A similar approach in [27] uses a matching pyramid. In contrast to our method, the windows on different detail levels cannot be moved independently.

Independent levels allow us to re-use high level information to get a much faster low level disparity computation. The graphics card stores the lens corrected image in a mip-map at eight different resolutions. Each level has half the horizontal and vertical resolution of the one below. All matching windows have a fixed size of 7×7 pixels. A smaller window size increases the noise while a larger size blurs sharp features. Starting on the coarsest resolution level $l = 7$, the disparities of all pixels in the reference images are computed at the same coarse resolution. The matching windows are evaluated at $k = 35$ different positions. Then the image resolution is doubled and the same process starts again, while $k = 1 + 2\lfloor 1.5 + l^2/3 \rfloor$ is reduced quadratically. As starting position for the matching windows on lower levels, the bi-linearly interpolated disparities of the next coarser level are used. Thus, the matching window moves k pixels around the best position of the previous level.

4.6 Deformed matching windows

Square matching windows can only yield good results, if the captured object surface is parallel to the image plane. Every surface not parallel to the image plane generates imprecision. To avoid this the matching windows are deformed to fit the perspective deformation of the object surface. The idea is based on [7], but we use the multi-level depth information and a projection free computation.

To estimate the deformation we use the disparity map of the previous multi-level step. First nine disparity values at the corners, the edge midpoints, and the center of the matching window are interpolated. This gives a disparity estimate for every pixel in the actual matching

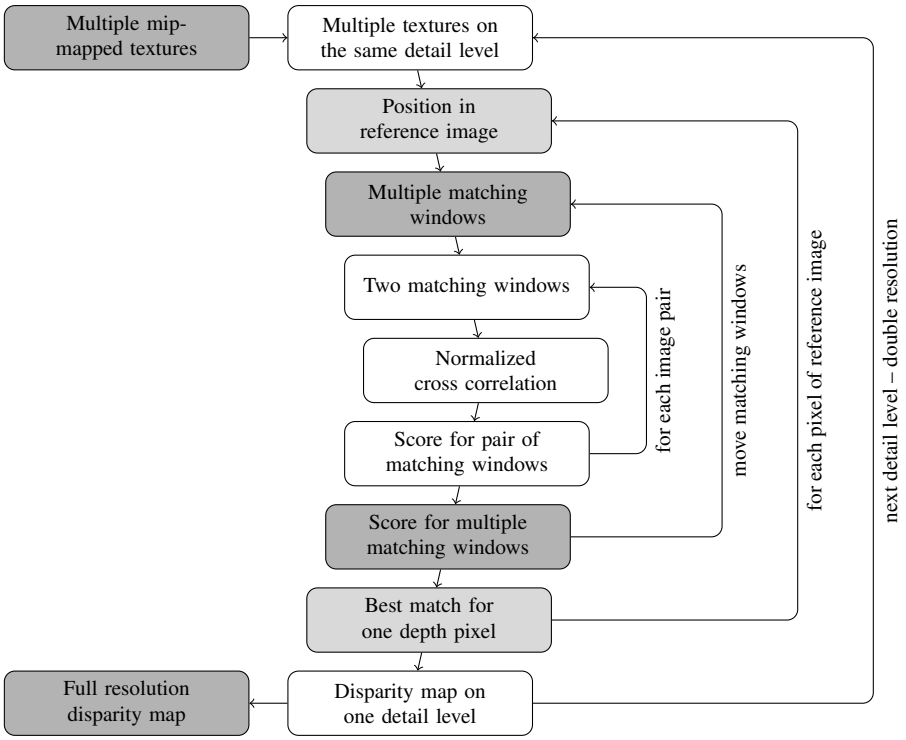


Figure 3: Overview of our matching process.

window. Subtracting the disparity at the center of the matching window yields a local displacement for every pixel. This displacement is added to the pixel coordinates before the color values are read. This results in a matching window adapted to the perspective of the previous level without computing any perspective projections. Note, that for planar object surfaces this approach is almost equivalent to the projections used by [7]. The difference is that it is based on disparity instead of depth.

4.7 Measuring the matching quality

For each resolution level a complete disparity map is computed. So, for each pixel of this map the best total score computed is also stored. Averaging these total scores over multiple resolution levels gives a quality measure for each pixel of the full resolution depth map, see Figure 4(b). Pixels with low quality measures can be masked for rendering or subsequent computations of the user application.

The quality measure is also used to improve the performance of the multi-level matching. A low quality measure on a coarse matching step usually causes the finer level matches in this region to fail too. Matching calculations are skipped if the quality measure on the next coarser level is too low.

5 IMPLEMENTATION ON THE GPU

The method described so far uses images and generates a depth image as result. Therefore, we use GLSL fragment shaders for the GPU implementation. A fragment shader is a program that runs in parallel on the GPU and processes one or multiple texture images into one result image. For our shader operations we need GPUs which support at least shader model 4.0. The required amount of computations in a single shader run is not feasible on older GPUs.

5.1 GPU lens correction

Our input data are multiple raw camera images. Each raw image is corrected by a shader implementing a lens correction similar to [4]. The resulting corrected images are rendered into separate textures. Each of these textures is then transformed into a mip-map. These mip-maps of the corrected images are used by all subsequent shaders of our system.

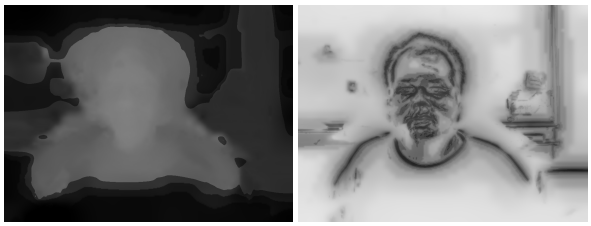
5.2 GPU optimized matching

A single pixel shader run usually computes the color values for one result image, each pixel separately. More complex computations require the combination of multiple shader runs. Three fragment shader programs are used for each step of our multi-level matching.

The first shader takes the corrected image mip-map and computes the weighted average color of the pixels of a matching window at the actual resolution level. These averages are rendered to separate average textures. This shader is invoked once for every image.

The second shader takes the corrected image mip-map and the average texture and computes the weighted auto-correlation for the same matching window. Again

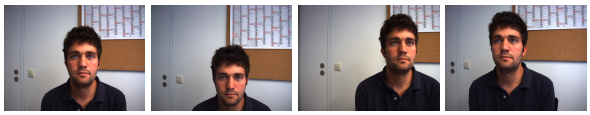
(a) The four captured sample images.



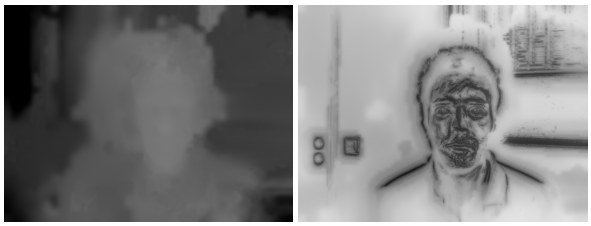
(b) Result textures. Disparity map (left) and quality measure (right).

(c) Reconstructed 3d model.

Figure 4: Example from our USB camera system.



(a) The four captured sample images.



(b) Result textures. Disparity map (left) and quality measure (right).

(c) Reconstructed 3d model.

Figure 5: Example from our FireWire camera system.

the result is rendered to a separate auto-correlation texture and the shader is invoked once for every image.

The third shader takes the average and auto-correlation textures and performs all matching operations, i.e. it moves the deformed matching windows, computes the total score, and finds the best sub-pixel score. The result is rendered as the disparity map, the best total score of the finest resolution and the quality measure to the three color channels of a separate texture. These three shaders are invoked once per resolution level.

Most important strategies used to improve the GPU performance are the pre-calculation of weighted average and weighted auto-correlation just described and the multi-level matching described in Section 4.5.

6 RESULTS

Our target application is face recognition. We present our results in that area. For easier comparison with other algorithms we also applied our algorithm to a well known benchmark for stereo matching.

6.1 Face reconstruction

We took some example images with our USB camera system shown in Figure 4(a). The disparities between these images are very large. The result texture of the fragment shaders holds the disparity map, the best total score of the finest resolution level, and the quality measure encoded in the color channels, see Figure 4(b).

After transformation of the disparities to depth values, the data can be rendered as 3d model, see Figure 4(c). The low quality regions are masked and ignored in this rendering.

A typical problem of stereo matching can be seen at the highlights on the forehead generating small dents, because the reflection is further away from the cameras than the forehead. More diffuse lighting could avoid this problem. The computation for the example images takes an average processing time of 129 ms on an NVidia GeForce GTX 285 GPU. This allows real-time frame rates of 7.5 fps.

A higher resolution of 1392×1032 is achieved by the FireWire camera system. An example image set is shown in Figure 5(a). Figure 5(b) shows the result textures and Figure 5(c) a 3d model of the resulting depth

map. The higher camera resolution yields a better shape quality at the most important regions of the face. Especially the reconstruction of the eye and mouth regions is much more precise.

For this example an average processing time of 263 ms is needed on the same GPU. For images of 30 different persons we get an average processing time of 254 ms. In most of these images the face region is smaller than in the displayed examples, so the computations are a bit faster. In comparison to the first example, the computation time grows almost linearly with the number of pixels p. This conforms to a runtime of $O(p \log p)$ for our multi-level algorithm: The matching window size, the stretch of the window movement, and the count of image pairs are constant. So the worst case costs for the computations in each depth map pixel is constant. The pixels of the resulting depth map, or smaller versions of it, are computed once for each of the $\log_2(\text{width}) \in O(\log p)$ multi-level steps. Hence the overall count of pixel calculations and the complexity of the algorithm is within $O(p \log p)$.

6.2 Stereo vision benchmarks

Several benchmarks can be used to compare the quality of stereo matching algorithms [19, 21, 25]. Our algorithm is tailored to face reconstruction and contains simplifications that require a planar camera configuration. Thus, it is not comparable to the benchmark [25]. Furthermore, our algorithm is also tailored to large disparities between the images and achieves a much better reconstruction quality using more than two cameras. So, only a comparison with the results of the extended datasets of the Middlebury stereo benchmark [20] is relatively fair. However, this benchmark does not provide an official score.

Compared to the algorithms providing results and timings for these benchmark our algorithm works much faster. At the same time the quality of our result is comparable to the quality of these algorithms. However, for this comparison we have to adapt our algorithm.

For the Middlebury stereo evaluation [20], we integrated a modified local version of Multi Hypothesis Matching [1] to improve the sharpness of edges in our algorithm. The movement range of the matching windows is extended to the depth extrema of the local neighborhood on the last detail level. Instead of evaluating only the best matching score, the eight best matching scores are stored. A post-processing step re-weights these scores based on the values and depth distances to the best scores in the direct pixel neighborhood. The re-weighting is repeated two times without any global optimization as in [1]. This multi hypothesis matching is implemented as an post-processing fragment shader on the GPU. The additional shader and the increased range for the matching windows cause a large performance loss. Processing our example images at a resolution of 960×720 pixels takes 900 ms. This is still faster than the other algorithms in [20], but not fast enough for our target application.

Figure 6 shows our algorithm applied to the extended *Tsukuba* dataset from [20, 16]. The two images in Figure 6(b) show the results from all five input images without and with the additional edge improvement.

7 CONCLUSION AND FUTURE WORK

The quality of the resulting surface model is sufficient and the processing times are more than sufficient for our target application 3d face recognition. Additional methods like cross-checking that can be implemented on the GPU could further improve our results. Furthermore, for an application of our method in a face recognition system, a simple method to guide persons to the optimal distance from the camera system is required.

For the future we plan to record synchronous video sequences with the FireWire camera system. Similar to multi-level matching, the matching information of an earlier video frame could be used to improve the performance.

ACKNOWLEDGMENTS

This work was supported by DFG GK 1131 and AiF ZIM Project KF 2372101SS9. We thank Jens Hensler for his help on creating a collection of face pictures.

REFERENCES

[1] Neill D. Campbell, George Vogiatzis, Carlos Hernández, and Roberto Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. In *Proc. of the 10th European Conf. on Computer Vision*, pages 766–779, 2008.

[2] Jia-Ching Cheng and Shin-Jang Feng. A real-time multiresolutional stereo matching algorithm. In *ICIP (3)*, pages 373–376, 2005.

[3] F. Devernay. A non-maxima suppression method for edge detection with sub-pixel accuracy. Technical Report RR-2724, INRIA, 1995.

[4] F. Devernay and O. Faugeras. Straight lines have to be straight: automatic calibration and removal of distortion from scenes of structured enviroments. *Mach. Vision Appl.*, 13(1):14–24, 2001.

[5] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. In *CVPR*, pages 1–8, 2007.

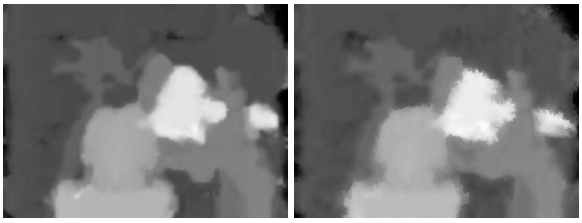
[6] A W Gruen. Adaptive least squares correlation: A powerful image matching technique. *South African Journal of Photogrammetry, Remote Sensing and Cartography*, 14:175–187, 1985.

[7] Hiroshi Hattori and Atsuto Maki. Stereo matching with direct surface orientation recovery. In *In Ninth British Machine Vision Conference*, pages 356–366, 1998.

[8] Alexander Hornung and Leif Kobbelt. Interactive pixel-accurate free viewpoint rendering from images with silhouette aware sampling. *Comp. Graph. Forum*, 28(8):2090–2103, 2009.

[9] Andreas Klaus, Mario Sormann, and Konrad Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *Proc. of the 18th Int. Conf. on Pattern Recognition*, pages 15–18, 2006.

(a) The extended *Tsukuba* dataset pictures.



(b) Result disparity map of our algorithm without (left) and with edge enhancement (right).

(c) Ground truth disparity map (left) and 3d rendering of the result with edge enhancement (right).

Figure 6: Results of the extended *Tsukuba* dataset from the Middlebury stereo benchmark [20].

[10] Reinhard Koch, Marc Pollefeys, and Luc Van Gool. Realistic 3-d scene modeling from uncalibrated image sequences. In *ICIP'99, Kobe: Japan*, pages 500–504, 1999.

[11] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1568–1583, 2006.

[12] Cheng Lei, Jason Selzer, and Yee-Hong Yang. Region-tree based stereo using dynamic programming optimization. In *Proc. of the 2006 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 2378–2385, 2006.

[13] J. P. Lewis. Fast template matching. In *Vision Interface*, pages 120–123, 1995.

[14] Kenneth Moreland and Edward Angel. The FFT on a GPU. In *Proc. of the ACM Conf. on Graphics Hardware*, pages 112–119, 2003.

[15] Don Murray and Jim Little. Using real-time stereo vision for mobile robot navigation. In *Autonomous Robots*, pages 161–171, 2000.

[16] Yuichi Nakamura, Tomohiko Matsuura, Kiyohide Satoh, and Yuichi Ohta. Occlusion detectable stereo – occlusion patterns in camera matrix. In *CVPR*, pages 371–378, 1996.

[17] D.T. Pham and L.C. Hieu. Reverse engineering - hardware and software. In V. Raja and K.J. Fernandes, editors, *Reverse Engineering - An Industrial Perspective*, pages 33–30. Springer, 2008.

[18] H. Sadeghi, P. Moallem, and S. A. Monadjemi. Feature based dense stereo matching using dynamic programming and color. *International Journal of Computational Intelligence*, 4(3):179–186, 2008.

[19] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.

[20] D. Scharstein and R. Szeliski. Middlebury stereo vision page. `http://vision.middlebury.edu/stereo/`, 2007.

[21] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. of the 2006 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 519–528, 2006.

[22] Steve Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. Multi-view stereo. `http://vision.middlebury.edu/mview/`, 2009.

[23] T. Shibahara, T. Aoki, H. Nakajima, and K. Kobayashi. A sub-pixel stereo correspondence technique based on 1d phase-only correlation. In *ICIP07*, pages 221–224, 2007.

[24] Christoph Strecha. Multi-view stereo test images. `http://cvlab.epfl.ch/~strecha/multiview/`, 2008.

[25] Christoph Strecha, Wolfgang von Hansen, Luc J. Van Gool, Pascal Fua, and Ulrich Thoennessen. On benchmarking camera calibration and multi-view stereo for high resolution imagery. In *CVPR*, pages 1–8. IEEE Computer Society, 2008.

[26] Francesca Voltolini, Sabry El-Hakim, Fabio Remondino, and Lorenzo Gonzo. Effective high resolution 3d geometric reconstruction of heritage and archaeological sites from images. In *Proc. of the 35th CAA Conference*, pages 43–50, 2007.

[27] Ruigang Yang and Marc Pollefeys. A versatile stereo implementation on commodity graphics hardware. *Real-Time Imaging*, 11(1):7–18, 2005.

[28] Ruigang Yang, Greg Welch, and Gary Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. In *Proc. of the 10th Pacific Conf. on Computer Graphics and Applications*, pages 225–235, 2002.

[29] Christopher Zach, Mario Sormann, and Konrad F. Karner. High-performance multi-view reconstruction. In *3DPVT*, pages 113–120, 2006.

[30] Cha Zhang and Tsuhan Chen. A self-reconfigurable camera array. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 151, 2004.