Runtime Analysis of GPU-Based Stereo Matching

Christian Zentner
School of Software Engineering
Tongji University
Shanghai, China

Yan Liu
School of Software Engineering
Tongji University
Shanghai, China

***Abstract*—This paper elaborates on the possibility to leverage the highly parallel nature of GPUs to implement more efficient stereo matching algorithms. Different algorithms have been implemented and compared on the CPU and the GPU in order to show the speedup gained by moving the computation to the graphics card. The results were evaluated for accuracy using the test available on the Middlebury website for stereo vision. An assessment of the runtime performance was done by a script which examined the runtime behaviour of the individual steps of the stereo matching algorithm.**

***Keywords*—stereo matching; GPU computing; runtime analysis; computer vision; image processing**

I. Introduction

Stereo matching is a technique used to generate a depth map from at least two simple images of the same scene. The resulting depth map is an image which represents the distances from the camera to the objects shown in the original image for every pixel. The information provided by the resulting depth maps can then be used to analyse the scene and support other methods like object tracking algorithms. In our case, the maps will be used in combination with a mobile robot, to support its visual navigation within a room.

The stereo matching process works similar to how human eyes would perceive the scene. Due to the difference in the location of the two eyes, we can see the scene from two different view points. Objects closer to the eye will appear shifted by a larger degree than objects within a longer distance. Based on this shift, the brain can calculate the depth of the objects we see and produce a three dimensional image. For the purpose of the stereo matching algorithm, we only calculate the depth information and skip the three dimensional image.

An integral part of the matching procedure is the correct setup and calibration of the cameras. Analogous to the example with the human eyes, the camera setup needs to consist of at least two cameras. Setups with more than two cameras are possible as well and can increase the precision of the stereo matching algorithm by providing more than one pair of sample images. The cameras have to be positioned in a certain way, so they both record the same scene but from a slightly different position. The distance between the cameras is crucial — if they are too close, the objects are not shifted far enough to precisely calculate the depth, if they are too far from each other, the shift is so large that some objects are not contained in both images. The orientation of the cameras needs to be adjusted as well, so that they provide the same view of the scene.

A. GPU Computing

Graphics hardware has become an integral part of modern computers. It is mainly used to accelerate multidimensional graphical computations, which are usually needed to display complex computer games or for graphical animations of the operating system. A common characteristic of this kind of computations is their huge amount of smaller, often independent calculations, which have to be performed on a regular basis and within a short time interval, in order to create a smooth visual presentation.

The reason why a graphics card can handle such a large amount of computations better than a single CPU is its highly parallel nature, which is perfectly suited for this problem. Instead of a single computing unit, a graphics card consists of several thousands of computing units which can perform many computations in parallel. Since graphical computations are often performed independently on a per-pixel or per-vertex basis, the graphics card can perform a lot of these at the same time, whereas the CPU would have to calculate them sequentially, one at a time.

Graphical computations, however, are not the only kind of computations that could benefit from having such a large array of computing units as offered by the graphics card. In fact, graphics cards have more recently become popular for performing general purpose computations, which is often referred to as GPU computing. An important area of application of GPU computing is image processing, where independent calculations are performed in parallel on every pixel.

For the navigation of our mobile robot, it is crucial to get a most recent snapshot of the current environment. Otherwise, decisions, based on the outdated spatial information, will not take the current position of the objects around the robot into account and might result in fatal collisions. Since stereo matching is an image processing technique, an implementation on the GPU should improve the runtime of the process and allow for a more accurate navigation.

A drawback worth mentioning is the relatively high initialisation cost of GPU programs. The data needed for the following calculations has to be transferred to the memory of the graphics card before the actual computation can begin. Depending on the size of the data in relation to the total amount of calculations, this comparatively slow data transfer can ruin the runtime benefit of the parallel computing units. However, the actual impact of the data transfer is hard to estimate, and the best way to investigate the runtime behaviour of GPU-based applications is to run and compare them with their CPU-based counterparts.

II. Matching Process

A. Preprocessing

Once the cameras are set up, they are ready to take a pair of images for further processing. Before these can be used by the matching algorithm, they need to be calibrated and rectified. The calibration is needed to remove distortions that are specific to a camera due to minimal irregularities in the build process. These distortions can be removed by applying a transformation on the image taken by that camera. The parameters for the transformation need to be determined by analysing the cameras distortion. This can be done using the Camera Calibration Toolbox for Matlab [2].

The next step is the rectification of the images. The matching algorithm will include a step where a certain element of one image will be searched for in the other image. To simplify this search, the images are transformed in a way that all objects that are on a plane, which is spanned by a point in the room and the coordinates of the two cameras, occur on the same line in both images taken by the cameras. This step is crucial for the following algorithms as it simplifies the search for a given object dramatically. Instead of searching across the whole image, the given object will now be within the same line as in the original image.

B. Algorithms

The matching algorithms work by analysing the so-called disparity, the shift of objects depicted in the images. Images closer to the cameras will show a larger shift when the two images are compared, whereas objects with a longer distance to the cameras will only show little deviation. To designate a certain disparity to a given pixel, the matching algorithm has to find the same pixel in the second image. However, the information contained in a single pixel is very limited, which makes it difficult to accurately determine the disparity. More information can be extracted from neighbouring pixels or even by taking the image as a whole into account. Stereo matching algorithms are hence divided into local and global methods.

The more information the algorithm uses, the more accurate are the results. Thus, global methods tend to produce much better depth maps than their local counterparts. This, however, comes at the cost of runtime, because global methods have to compute a much larger amount of information. For the purpose of providing up-to-date depth maps, this paper will focus on local methods, due to their significantly faster execution.

Local stereo matching algorithms consist of three basic steps and can include further enhancements to increase the quality of the generated depth maps. For the assessment of the runtime behaviour of different algorithms, a simple and a more sophisticated version of each of the three basic steps have been implemented on the GPU and also on the CPU. Additional enhancements have been neglected. A complete stereo matching algorithm is obtained by combining algorithms for each step. This modular approach allows for different setups which can be used to evaluate the impact of certain parts of the algorithm.

1) Likeliness: The first step is to calculate the likeliness of pixels at a given disparity in the second image compared to each pixel in the first image. To find a pixel with a certain disparity, we simply need to shift the x coordinate by the value of the disparity. This works because of the rectification step which was part of the preprocessing. It will assure that a point in the scene will be mapped to the same line in both images, regardless of the disparity. The actual likeliness is defined by a measure which differs depending on the algorithm used. This has to be performed for every disparity up to a certain maximum.

The simple version, that has been implemented for this step, makes use of a sum of squared differences as the measurement for likeliness. As a little enhancement, the maximum difference is limited by a constant, which keeps the influence of outliers within sane limits [6]. Given a pair of images I and J, the likeliness (or also called cost) for a given pixel $\mathbf{p}$, a disparity of $\mathbf{d} = \begin{pmatrix} d & 0 \end{pmatrix}^T$ and a limit of c_{max} can be calculated as follows:

$$c_{ssd}(\mathbf{p}, \mathbf{d}) = \min \left(\sum_{i=R,G,B} (I_i(\mathbf{p}) - J_i(\mathbf{p} - \mathbf{d}))^2, c_{max} \right)$$

For structured surfaces, the results produced by this simple method are quite good, but it falls short for plain areas with little colour differences.

A more advanced version of this step actually combines two different algorithms to exploit their benefits and cancel out each others drawbacks. One value is determined by using the absolute differences of the colour channels of the pixels, which works similar to the simple version:

$$c_{ad}(\mathbf{p}, \mathbf{d}) = \frac{1}{3} \sum_{i=R,G,B} |I_i(\mathbf{p}) - J_i(\mathbf{p} - \mathbf{d})|$$

Then, another value is calculated using the Census transform, which compares the intensity of a pixel with its neighbours. The results of the comparison for each image are stored in a bit string, where each bit encodes whether the intensity of the neighbour was greater or less the intensity of the pixel of interest. The final value c_{cen} is then calculated as the Hamming distance between the two bit strings, which is the amount of bits that are different. Eventually, these two values are normalised using a function ρ, which takes the value for the likeliness c and a parameter λ to adjust the influence of outliers. The normalised values are then accumulated to yield the final result:

$$\rho(c, \lambda) = 1 - \exp\left(-\frac{c}{\lambda}\right)$$

$$c_{adc}(\mathbf{p}, \mathbf{d}) = \rho(c_{ad}(\mathbf{p}, \mathbf{d}), \lambda_{ad}) + \rho(c_{cen}(\mathbf{p}, \mathbf{d}), \lambda_{cen})$$

By combining two different measures, this version performs well under most conditions.

2) Grouping: For the second basic step, more information about the neighbourhood of the given pixel is being incorporated. Here, the algorithm defines a certain area around the pixel and uses the average likeliness of this area as the value for the pixel. The strategy for deciding which pixels should be included into this area is very important and has a great influence on the detail of the depth map.

The naive approach does not pay any attention to the objects in the image. It simply defines a window of size $b_x \times b_y$

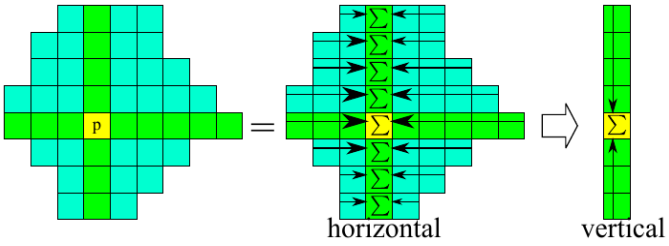

Fig. 1: Calculation of the average likeliness using cross-based aggregation (Mei *et al.*, 2011 [5]).

over which the average likeliness a of the included pixels with likeliness $c(\mathbf{p}, \mathbf{d})$ is calculated:

$$a_{box}(\mathbf{p}, \mathbf{d}) = \frac{1}{2b_x + 2b_y + 2} \sum_{x=p_x-b_x}^{p_x+b_x} \sum_{y=p_y-b_y}^{p_y+b_y} c\left(\begin{pmatrix} x \\ y \end{pmatrix}, \mathbf{d}\right)$$

Since this algorithm does not recognise objects within the image, it might average over object boarders and include elements with a large visual difference. Because of this, details will be lost and the result is not very accurate.

An improved version of this step is called cross-based aggregation, as described by Zhang *et al.* [15]. This approach defines the area, used for the aggregation, by extending into all directions from the pixel, for which the value is being aggregated. This creates arms of pixels to be included in the area, as depicted in figure 1. The length of these arms is limited by a maximum length and a threshold for the intensity difference. A further enhancement can be introduced by defining a second length and threshold. If an arm passes this second length, the second and more strict threshold should be used instead. The values chosen for the maximum length of the arms during the test are 17 and 34 pixels, combined with a threshold of 20 and 6 respectively. This algorithm, particularly with the mentioned enhancement, produces much cleaner results than the naive version, because it takes object boarders into account and tries to preserve details in the image.

3) Evaluation: Eventually, the previously computed values need to be evaluated, in order to estimate the correct disparity for the given pixel. The possibly correct disparity is the one, where the two pixels of both images have the least difference, i.e. the smallest aggregated likeliness value. This step can be used to apply further optimisations, the simplest approach, however, does not apply any optimisation and only selects the disparity which yields the smallest value for the previous step. This version is called winner-takes-all.

An approach, which performs a further optimisation, is called scan-line optimisation. This algorithm generates four new values for each pixel and each disparity and uses their average as the actual value at the given disparity. These four values are calculated based on the values of the pixels along all four directions originating from the given pixel. When all the averages for all disparities for one pixel are determined, the final result is again determined using the winner-takes-all algorithm.

III. Implementation

All of the mentioned algorithms have been implemented on a regular CPU in C++, as well as on the GPU using the OpenCL framework. The CPU versions follow a straight forward implementation, processing the pixels sequentially on a single core. To make full use of the parallel programming paradigm of the graphics hardware, a more complicated approach than for the CPU version is required

Luckily, except for the scan-line optimisation, all employed algorithms can mostly be executed independently and in parallel on a per-pixel basis. For the scan-line optimisation, each scan-line has to be calculated sequentially, which does not allow for a per-pixel division. However, for this algorithm, the disparities for a pixel can be calculated independently.

Another important aspect of GPU programming is the memory access. The graphics card features memory regions with different access speeds. The faster local memory regions are, however, only accessible by computing units of a certain group. That makes the actual division of computing units an important factor for the total runtime of the program, because accessing the main memory region is significantly slower.

The cross-based aggregation, for example, was grouped in a way, that one group is responsible for calculating one arm. The data needed for this arm can be loaded into the local memory, which reduces the access to the main memory to zero. By grouping the computing units this way, two effects could be witnessed. As the first improvement, due to the more organised way of grouping, the amount of lines of source code could be reduce to a third of the earlier, naive implementation. Furthermore, the runtime was reduced by ten times. This shows how important the memory access pattern for computations that need to read a lot of data is.

IV. Results

A. Accuracy

Before the assessment of the runtime performance, the quality of the depth maps, produced by the algorithms, will be tested and compared to other common stereo matching algorithms. This should guarantee, that the results for the runtime tests reflect the behaviour of algorithms that can be used by actual applications, which are not solely designed and optimised for their speed of execution.

The accuracy is tested using the Middlebury website [8], which provides an online test for stereo matching algorithms. A set of sample stereo images, that were used for previous publications [6] and [7], are provided, and the resulting depth maps will automatically be compared with laser scans of the actual environment. Figure 2 shows some of the images used for the test and their results. For each input image pair, one of the input images is shown, followed by the laser-scanned depth map of the scene and the resulting images for the combination of simple algorithms and for the advanced versions.

The evaluation shows the average error and a ranking position in a table containing other implementations. The error is calculated according to the paper on the evaluation of stereo correspondance algorithms [6] and is basically the root-mean-square error of the calculated depth compared to the laser-scanned depth for each pixel. Even the combination of the

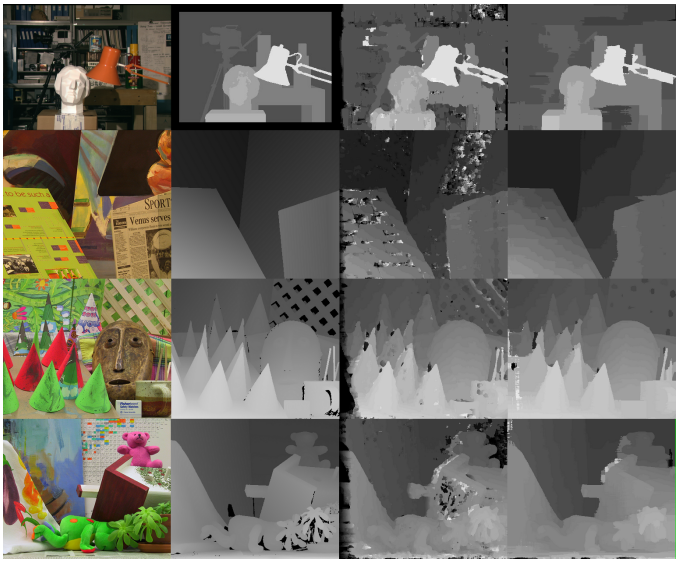

Fig. 2: Test images from the Middlebury website and the resulting depth maps. From left to right: One of the input images (from top to bottom: Tsukuba, Venus, Cones and Teddy), laser-scanned depth map, result for the combination of simple algorithms (sum of squared differences, box-based aggregation and winner takes all), result for the combination of advanced algorithms (absolute differences & census transform, cross-based aggregation and scan-line optimisation).

Image	SSD [ms]	BOX [ms]	WTA [ms]	Σ [ms]
Tsukuba	5.97	111.35	2.73	120.05
Venus	15.97	304.36	6.03	326.37
Cones	31.53	617.59	10.86	659.98
Teddy	31.44	617.21	10.85	659.50
Image	ADC [ms]	CBA [ms]	SLO [ms]	Σ [ms]
Tsukuba	686.11	255.25	225.60	1166.96
Venus	1905.53	507.04	875.40	3287.97
Cones	3735.04	567.88	2871.59	7174.50
Teddy	3736.18	764.99	2814.76	7315.93

TABLE I: Runtime on an Intel Core i5-2500K CPU for the combinations SSD (sum of squared differences), BOX (box-based aggregation) and WTA (winner takes all) as well as ADC (absolute differences & census transform), CBA (cross-based aggregation), and SLO (scan-line optimisation).

most simple algorithms is able to compete against some of the methods with low ranking. However, the quality is far from what the high ranked methods can produce, at a relatively high average error of 18.2%. The combination of the more advanced algorithms shows much better results, reducing the average error to 9.3%. Thus, the algorithm gains a place in the middle field of the table. Though these results are much better, and the resulting depth maps mostly provide clear and useful information, there is still a huge gap compared to the best ranked methods.

B. Runtime

Finally, the runtime performance of the CPU implementation is compared to its GPU counterpart. For this purpose, a given combination of algorithms were run for 20 times and their performance, shown in tables I and II, was analysed

Image	SSD [ms]	BOX [ms]	WTA [ms]	Σ [ms]
Tsukuba	3.53	11.32	1.64	16.50
Venus	9.48	36.02	3.32	48.82
Cones	18.87	75.72	5.69	100.28
Teddy	18.87	75.67	5.69	100.23
Image	ADC [ms]	CBA [ms]	SLO [ms]	Σ [ms]
Tsukuba	9.82	17.15	13.75	40.72
Venus	24.79	62.91	29.46	117.16
Cones	74.42	137.92	54.56	266.91
Teddy	74.41	139.52	54.53	268.46

TABLE II: Runtime on a NVIDIA GeForce GTX 570 GPU for the combinations SSD (sum of squared differences), BOX (box-based aggregation) and WTA (winner takes all) as well as ADC (absolute differences & census transform), CBA (cross-based aggregation), and SLO (scan-line optimisation).

and averaged by a script. The chosen configurations feature the simple sub-algorithms, SSD (sum of squared differences), BOX (box-based aggregation) and WTA (winner takes all), as well as the more advanced versions, ADC (absolute differences & census transform), CBA (cross-based aggregation) and SLO (scan-line optimisation).

The script read the timings, produced by the algorithms for each step of the stereo matching process, and computed for each of them, and for the total run of the combination, the average result and the standard deviation. This allows to compare the runtime of every part of the algorithm and give a more detailed analysis of the benefit of implementing these on the GPU. All tests were performed on a system with an Intel Core i5-2500K @ 3.30 GHz processor and an NVIDIA GeForce GTX 570 graphics card.

The results show that the algorithms responded differently on the implementation choice. While the simpler algorithms, like SSD and WTA, could only gain a speedup by a factor of around 2, other algorithms, like the combination of the absolute differences and the census transform, could profit from the parallelism a lot, showing a speedup by up to a factor of 50 to 75 compared to the CPU version. This also has an impact on the accumulated runtime of the whole stereo matching methods.

Figure 3 gives an overview of the total runtime of the stereo matching for different images from the Middlebury website. The chosen algorithms are the same combinations as in the tables, using the simpler version SBW (SSD, BOX & WTA) and the more advanced version ACS (ADC, CBA & SLO). Note that the axis for the runtime is logarithmic. As for the comparison of the individual parts, the total runtime of the advanced version shows a larger speedup than the less sophisticated implementation. This might be related to the higher initialisation cost of GPU-based programs.

V. Conclusion

This paper provides a rough introduction to stereo matching on the graphics hardware and compares the speedup gained in comparison to a similar CPU implementation. The evaluation can be used as a basic guidance for other developers to decide, whether GPU computing is a suitable option for heavy computations within their projects, and as a starting point for further investigations. Especially, time-critical and more complex operations could benefit a lot from an implementation

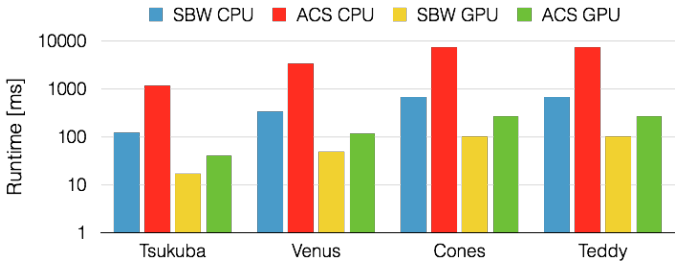

Fig. 3: Comparison of the total runtime of the combinations SBW (sum of squared differences, box-based aggregation and winnter takes all) and ACS (absolute differences & census transform, cross-based aggregation and scan-line optimisation).

on the GPU, given the underlying algorithm can be parallelised in some way.

The test results for the accuracy of the implemented algorithms show that there is still a large space for improvements. However, the produced depth maps of the advanced algorithms are by all means sufficient for most applications, and should suffice for the purpose of this paper. The runtime comparison shows that some of the algorithms can benefit significantly from the highly parallel nature of the graphics hardware. Even algorithms like the scan-line optimisation, which are hard to implement efficiently in a parallel manner, given their partly serial computations, seem to gain a huge performance boost on the GPU.

These observations, in relation with the relatively low cost and high availability of graphics hardware and the increasing interest and popularity of GPU computing, which also pushes the development of frameworks and development tools, certainly justify the use of graphics cards for stereo matching algorithms. Indeed, many recently developed stereo matching methods make use of GPU computing and show that it is possible to obtain high quality depth maps within a short runtime using this technology [5].

For our mobile robot, the presented GPU implementation provides for a much more accurate navigation based on very recent depth maps. Another benefit of the short runtime of this algorithm is that other parts of the whole process, that is responsible for the robot's navigation, could be allowed to consume more time in order to improve their accuracy. This allows for a more flexible fine-tuning of the whole navigation process, to find the perfect balance between using most up-to-date data and using more accurate algorithms.

The only drawback is the higher level of complexity needed to implement efficient algorithms on the GPU. For more sophisticated methods, a GPU implemetation, that exploits the offered level of parallelism to a sane extend, needs to pay attention to many pitfalls that do not pose any obstacles for a CPU version. Especially, finding a performant way of parallelising an algorithm and optimising memory accesses are an important and sometimes difficult task. However, implementing an efficient CPU-based algorithm also requires for complex mechanisms. Eventually, the huge runtime speedup on the graphics hardware makes GPU computing an alternative that is worth to be explored.

References

[1] M. Bleyer and M. Gelautz, "A layered stereo matching algorithm using image segmentation and global visibility constraints," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 59, no. 3, pp. 128–150, 2005.

[2] J.-Y. Bouguet, *Camera calibration toolbox for matlab*. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib%5C_doc/index.html (visited on Nov. 25, 2012).

[3] S. Chambon and A. Crouzil, "Similarity measures for image matching despite occlusions in stereo vision," *Pattern Recognition*, vol. 44, no. 9, pp. 2063–2075, 2011.

[4] N. Lazaros, G. C. Sirakoulis, and A. Gasteratos, "Review of stereo vision algorithms: From software to hardware," *International Journal of Optomechatronics*, vol. 2, no. 4, pp. 435–462, 2008.

[5] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On building an accurate stereo matching system on graphics hardware," *GPUCV: ICCV Workshop on GPU in Computer Vision Applications*, 2011.

[6] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[7] ——, "High-accuracy stereo depth maps using structured light," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, vol. 1, pp. 195–202, 2003.

[8] ——, *Middlebury stereo vision page*. [Online]. Available: http://vision.middlebury.edu/stereo/ (visited on Nov. 25, 2012).

[9] T. Schöpping, "Entwicklung eines stereo-vision-verfahrens für einen mobilen roboter," Bachelor thesis, Technische Fakultät der Universität Bielefeld, Bielefeld, 2011.

[10] O. Schreer, *Stereoanalyse und Bildsynthese*. Berlin: Springer, 2005, ISBN: 3-540-23439-X.

[11] L. D. Stefano, M. Marchionni, and S. Mattoccia, "A fast area-based stereo matching algorithm," *Image and Vision Computing*, vol. 22, no. 12, pp. 983–1005, 2004.

[12] O. Veksler, "Reducing search space for stereo correspondence with graph cuts," *Proceedings of British Machine Vision Conference*, vol. 2, pp. 709–718, 2006.

[13] K.-J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 650–656, 2006.

[14] C. Zentner, "Gpu-basiertes stereo-matching auf einem mobilen roboter," Bachelor thesis, Technische Fakultät der Universität Bielefeld, Bielefeld, 2012.

[15] K. Zhang, J. Lu, and G. Lafruit, "Cross-based local stereo matching using orthogonal integral images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 7, pp. 1073–1079, 2009.