Large scale Semi-Global Matching on the CPU

Robert Spangenberg, Tobias Langner, Sven Adfeldt and Raúl Rojas

Original Publication:

Spangenberg, R.; Langner, T.; Adfeldt, S. & Rojas, R. Large scale Semi-Global Matching on the CPU Intelligent Vehicles Symposium Proceedings, 2014 IEEE, 2014, 195-201

The final publication is available at

http://dx.doi.org/10.1109/IVS.2014.6856419

Large Scale Semi-Global Matching on the CPU

Robert Spangenberg[1], Tobias Langner[1], Sven Adfeldt[1] and Raúl Rojas[1]

***Abstract*— Semi-Global Matching (SGM) is widely used for real-time stereo vision in the automotive context. Despite its popularity, only implementations using reconfigurable hardware (FPGA) or graphics hardware (GPU) achieve high enough frame rates for intelligent vehicles. Existing real-time implementations for general purpose PCs use image and disparity sub-sampling at the expense of matching quality. We study methods to improve the efficiency of SGM on general purpose PCs, through fine grained parallelization and usage of multiple cores. The different approaches are evaluated on the KITTI benchmark, which provides real imagery with LIDAR ground truth. The system is able to compute disparity maps of VGA image pairs with a disparity range of 128 values at more than 16 Hz. The approach is scalable to the number of available cores and portable to embedded processors.**

I. INTRODUCTION

Stereo vision systems provide 3D perception by calculating the disparities of two input images. Due to their low cost, they can compete with active technologies as RADAR or LIDAR and are widely used in robotics and automotive systems. They are employed in commercial vehicles to support functions such as object, or pedestrian detection and automated collision avoidance. The classical benchmark to compare stereo algorithms is the Middlebury Stereo Dataset [1]. It provides indoor scenes in a controlled setting. The KITTI Vision benchmark suite [2] provides real-world test images from the automotive context. Semi-global matching (SGM) [3] achieves good performance on both benchmarks and has been selected due to its relative efficiency and algorithmic simplicity.

The SGM method approximates a global two-dimensional Markov Random Field (MRF) regularized cost function by following one dimensional paths in several directions through the image. It is sufficient to use 8 or 16 paths to cover the structure of the image. Along each path, the minimum cost is calculated by means of dynamic programming. The minimized energy consists of the data-term for photo-consistency and two smoothness terms for slanted surfaces and depth discontinuities. For automotive applications, the data term is often census-based.

FPGA and GPU implementations of SGM for real-time performance exist. While GPUs with sufficient performance consume much energy, FPGAs are hard to integrate in general purpose PCs or laptops. Furthermore, the development process for FPGAs is quite complicated and time consuming, even for small changes. Therefore, the goal is to achieve frame-rates above 10 Hz for robotics and driver assistance applications on a general purpose multi-core CPU platform.

Due to the high memory requirements ($\mathcal{O}(h \cdot w \cdot d)$) of SGM, Hirschmüller proposed to process large images in tiles and interpolate the resulting cost cubes at the tile borders [3]. Here h denotes the image height, w the image width and d the number of disparities. In [4], the image is divided into several horizontal stripes for multi-core processing. The authors report a degradation of roughly 1% less correctly matched pixels on the Middlebury Dataset. We apply this idea, but spend an additional overlap-area for the stripes, in order to reduce the performance degradation.

This paper is organized as follows. Related work is presented in section II. We then shortly explain the SGM algorithm and detail the optimization approaches (section III). The conducted experiments and their results are shown in section V, followed by the conclusions in section VI.

II. RELATED WORK

The FPGA implementation in [5] achieves 33 Hz for VGA images at a disparity resolution of 64 pixels using low-cost to mid-range hardware. Another approach processes 680×400 px images at 25 Hz with a disparity range of 128 and a power consumption of 3W [6]. Due to memory bandwidth restrictions, two SGM engines work on 340×200 px image pairs with 64 disparities, whose results are combined.

GPU implementations reach 11.7 fps for VGA resolution and 64 disparities on a GeForce GTX 480 with a CUDA implementation [7]. An earlier implementation gets 13 fps on QVGA images, and a 64 pixel disparity range with a Cg implementation on a GeForce 8800 Ultra [8]. The power consumption of these GPUs is clearly above 150 W.

The CPU implementation in [9] uses image and depth sub-sampling to achieve a frame rate of 14 Hz for a resolution of 640×320 pixels. The depth sub-sampling guarantees a depth uncertainty below 1 m. SIMD instructions (single instruction, multiple data) are employed to compute 16 disparities in one step during the path accumulation step. It uses OpenMP to calculate each independent path in parallel. To calculate the similarity criterion, a 5×5 Census transform is computed, parallelized line-wise with OpenMP.

The Central-symmetric census transform is proposed as a sparse census transform in [10] to allow bigger patch sizes while preserving the computational speed of smaller patch sizes. Another way to sparsify the Census transform was proposed in [11], consisting of leaving out every second pixel of a patch.

[1]all authors are with Institut für Informatik at Freie Universität Berlin, Arnimallee 7, 14195 Berlin, Germany robert.spangenberg@fu-berlin.de

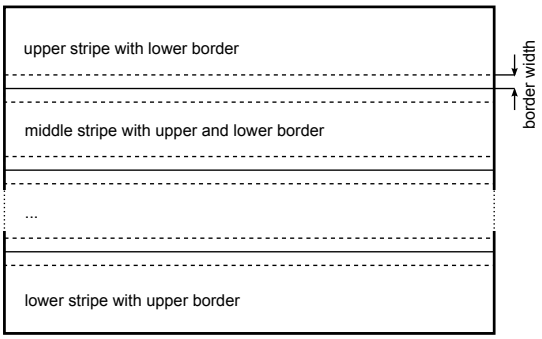


Fig. 1: Image stripes overlap

III. SEMI-GLOBAL MATCHING

A. Algorithmic structure

The SGM algorithm can be structured as follows. At first the similarity criterion is computed, which in our case is a Hamming Distance of a 5×5 or 9×7 Census window (used by [4] as well). Compared to the original mutual information criterion in [3], Census has been found to perform better in outdoor scenes [12] and with slight decalibrations [13]. Both conditions apply to our own scenarios. After that, path accumulation is done over 8 paths. In our application, the variant using 16 paths only leads to minor improvements, which do not justify the greater computational work. The path costs are summed up into one disparity cost cube. From this cube, the disparities connected to the minimum costs are obtained for both viewpoints. This is done by a simple winner-takes-all (WTA) strategy. Both images are median filtered and occluded regions are invalidated with the left-right-consistency check. In conclusion, we get the following computational parts:

- Census mask computation
- Data cost calculation
- Path accumulation
- WTA left and WTA right
- Sub-pixel interpolation
- Median filtering
- Left-right-consistency check.

B. Parallelization Concept

The first step of optimization is to employ SIMD instructions for the most expensive parts of the algorithm, i.e. the first four parts. The question remains of how to parallelize on thread-level. Except for path accumulation, image lines can be processed in parallel with no additional synchronization cost. Therefore a parallelization based on horizontal image stripes seems favorable, assigning one strip to each thread. Path accumulation suffers, as it cannot access the respective stripes above or below the current one. Paths across stripes are cut and information cannot be propagated over them. We fix the performance degradation by adding an additional upper and lower border to the processed image stripe during the path accumulation (Fig. 1). As we will see in the results section, a small additional border is sufficient.

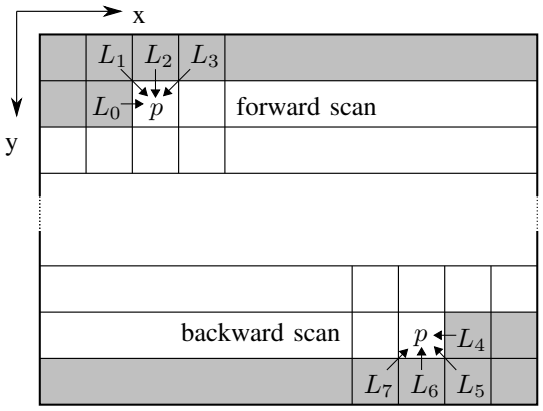


Fig. 2: Independent paths and scans - grey areas are calculated before the new pixel p.

Another option would be to calculate independent paths in parallel during accumulation (similar to [9], see Fig. 2). This involves an additional synchronization point at the summation of the path costs. Our tests with OpenMP did not produce any speedup compared to the scalar version. We think that this parallelization was too fine grained for OpenMP.

C. Disparity Space Compression

Ideally, the SGM algorithm should be executed over the whole disparity range. Depending on the application, it might be sufficient to sample greater disparities with a sparser grid. As detailed in [9], we can reduce the number of computed disparities, while guaranteeing a certain depth uncertainty. We can sample every value at the disparity range 0 to 63, and every second or fourth in the range of 64 to 127.

As all parts of the algorithm except the data cost calculation are agnostic to the actual costs minimized, we can partially sub-sample the disparity space in the data cost calculation. We compress all data costs into one continuous disparity volume and then apply the rest of the algorithm on this reduced volume (Fig. 3). The resulting disparity maps are uncompressed by a simple remapping operation. This maps all sub-sampled disparities to their true values. Since the input images are not sub-sampled, the data cost have to be tolerant to small offsets induced by sub-sampling. If a pattern belongs to an image patch of disparity d_1 and this disparity is not hit by the sub-sampling, the data cost should give a minimum at either the left or right side of d_1. Furthermore, we indirectly modify the SGM smoothness term, such that pixels in the sub-sampled range are allowed to have larger disparity changes.

The runtime of the core algorithm depends roughly linearly on the number of evaluated disparities and the additional mapping function is fast. Therefore, disparity subsampling offers a great speed improvement, if the loss of disparity resolution is acceptable for the application. In contrast to image sub-sampling methods, all image information is used and the disparity image itself is not downsized.

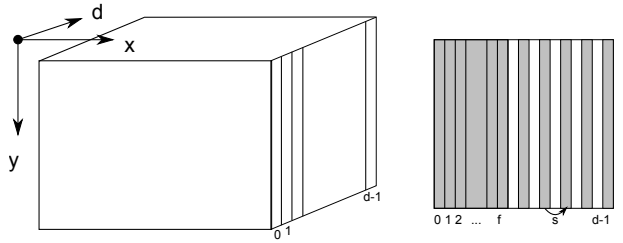


Fig. 3: Disparity space compression: The full cost cube has one slice per disparity value. For the compression, one starts to sub-sample at a disparity f with a step width s until the last disparity $d-1$. Only the gray parts form the compressed slice.

IV. IMPLEMENTATION

A. Data Cost Calculation

We use a 5×5 or 9×7 Census mask as input for the Hamming cost calculation. The census images are calculated with SIMD instructions. For 8 bit inputs we process 16 pixels at once, as those fit completely into the 128 bit SSE registers. For 12 bit images, we process 8 pixels at once. Thirty-two bit integers suffice to store the result of the transformation for the 5×5 mask, 64 bit for the 9×7 mask.

The cost computation step is performed $h \cdot w \cdot d$ times per frame, making it time-critical. Therefore, we parallelize the calculation on image stripes. The Hamming distance between two census integers is calculated through an XOR-operation followed by a population count, the summation of set bits. The critical part is the calculation of the population count. Lookup-tables (LUTs) are the fastest option without any special instruction set available, but they are not vectorizable and cost a significant amount of memory bandwidth. If hardware population counts are available as part of the instruction set (e.g. POPCNT), they are faster than LUTs, but work only on single values. Parallel logic can be used for the computation as well [4]. The SSSE3 instruction PSHUFB offers the possibility to implement a parallel 4 bit-LUT in logic [14], which can be used in connection with horizontal adds to get population counts of 32 or 64 bit values. This is similar to the LUT version, but without memory bandwidth usage. The main advantage to the hardware solution is that a parallel calculation of population counts over the full SIMD register width is possible. This fits better to the parallel load and store operation of the SIMD instructions and is faster than parallel logic.

B. Path Accumulation

Path accumulation is performed using the following equation:

$$\begin{aligned} L_{\mathbf{r}}(\mathbf{p}, d) = & C(\mathbf{p}, d) + \min[L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d) \\ & , L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d-1) + P_1, L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d+1) + P_1 \\ & , \min_i L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, i) + P_2]. \end{aligned} \tag{1}$$

This equation for a path $\mathbf{r}$ must be evaluated $8 \cdot h \cdot w \cdot d$ times per frame, when using 8 paths. As we limit the aggregated costs to a 16 bit value, we can process 8 disparities in one step. Since we can use hardware instructions for saturated add, minimum and maximum, the speedup is higher than 8, which could be expected from the data parallelization.

The information from all paths is summed for all pixels and disparities giving the accumulated costs

$$S(\mathbf{p}, d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p}, d). \tag{2}$$

We combine the individual paths to the final aggregated sum by using two scans (Fig. 2). The first scan goes from the top left to the bottom right, and combines the first 4 paths, storing an intermediate result for each pixel and disparity. The second runs back from the bottom right to the top left adding the results of the remaining 4 paths to the intermediate, giving the accumulated costs.

C. Winner Takes All

We use a classic search for the disparity of minimum cost. The second minimum is calculated as well and is used to invalidate the disparity, if the cost is not distinctive enough. This has to be done for $h \cdot w$ pixels of the left and right image. We use the SSE4 instruction PHMINPOSUW to calculate the minimum and position of 8 cost values at once. For the WTA of the right image, we use diagonal search in the cost volume. For this part, the memory layout prevents direct SIMD processing. We copy all costs of disparities belonging to a pixel location to a temporary storage and then perform minimum search as for the left image.

D. Remaining steps

We use the equiangular interpolation as suggested by [15] to compute sub-pixel disparities, since we minimize linear costs. After that, a 3×3 median filtering is done in parallel using SIMD instructions and a sorting network using parallel minimum and maximum instructions. The left-right check is applied at last.

E. Disparity Space Compression

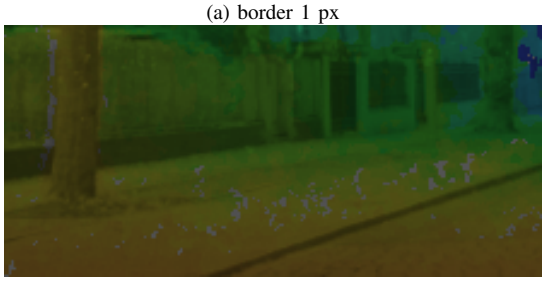
For each disparity space compression variant, a specific function has to be designed to implement the sub-sampling. The remapping function is realized without branches and uses SIMD instructions.

V. EXPERIMENTAL RESULTS

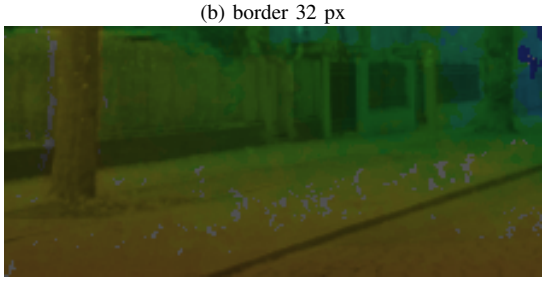
A. Setup

We used an Intel Core i7 i7-4960HQ notebook processor with four cores and 2.6 GHz base clock rate. We compiled the program in 64 bit mode using MSVC 2010. Parameters for the SGM method are $P_1 = 7$, $P_2 = 100$. Fig. 5 shows a part of an own test sequence with 12 Bit intensity values and PAL resolution.

(a) border 1 px



(b) border 32 px



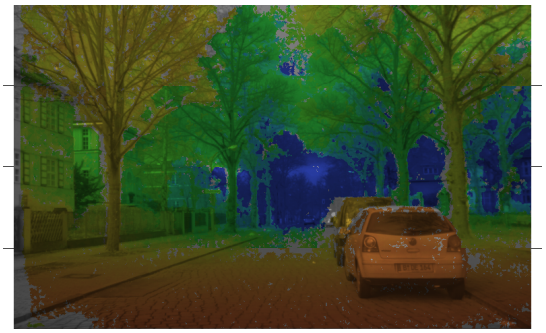
(c) full SGM

Fig. 4: 4-striped SGM - pavement detail

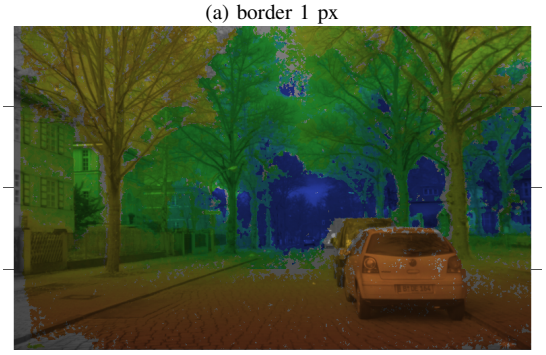
B. *Quality*

The KITTI stereo data set [2] is used for quantitative evaluation, providing ground-truth obtained by a laser-scanner. The scenes are rather complex, with large regions of poor contrast, lighting differences among stereo pairs, motion blur and a large disparity range. The data set is separated into a training and testing data set of around 200 images each. Ground truth is provided freely accessible for the training data set only, results for the testing data set are obtained by an on-line service.

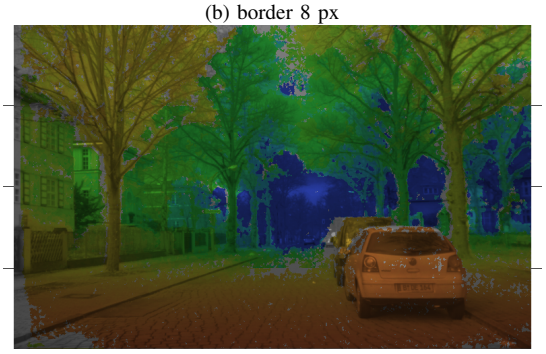
At first we tested the quality degradation caused by the separation into stripes. With only a small additional overlap, the border regions between the stripes can be clearly seen on the road, the sidewalk and in the trees (Fig. 5, upper image, extract at Fig. 4). Especially textureless areas tend to get filled with the disparity values of their surroundings, where texture is present. When the stripes separate less textured areas from textured ones, jumps in the disparities become apparent. With increasing overlap area, the disparity discontinuities disappear. However, in scenes with very large untextured areas, they might persist, as information is lacking from the adjacent stripe. In this case, the classical SGM does provide a propagation of the full image neighborhood dominated by the regularization penalties. Depending on the structure of the scene and the regularization parameters, each of them might be closer to the true values.



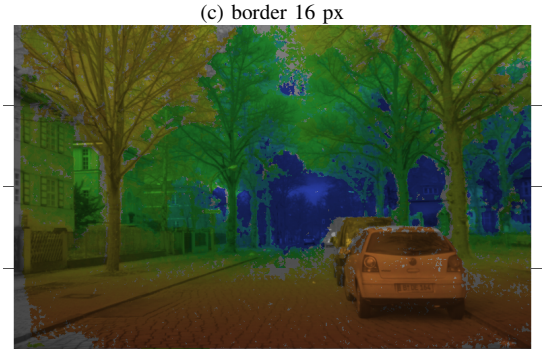
(a) border 1 px



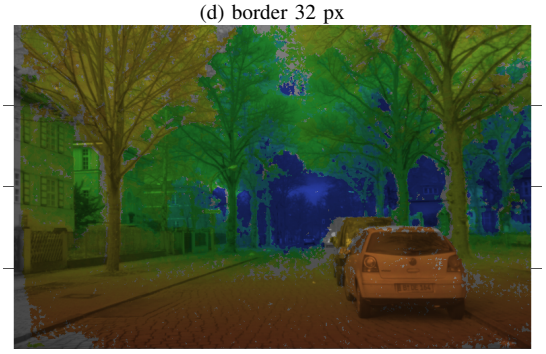
(b) border 8 px



(c) border 16 px



(d) border 32 px



(e) full SGM

Fig. 5: 4-striped SGM - lines at the left/right show stripes.

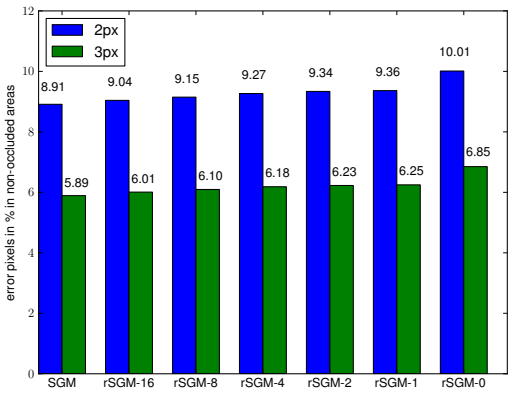


Fig. 6: Border width and quality for 4-striped SGM on the KITTI training data set

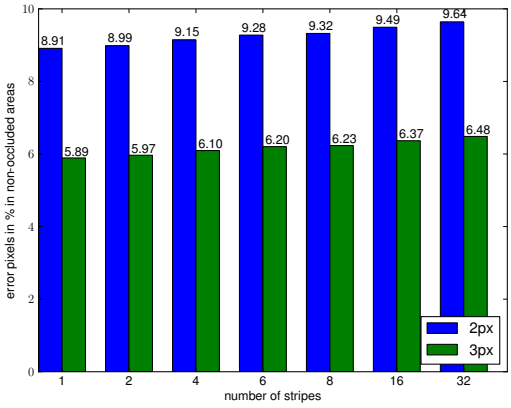
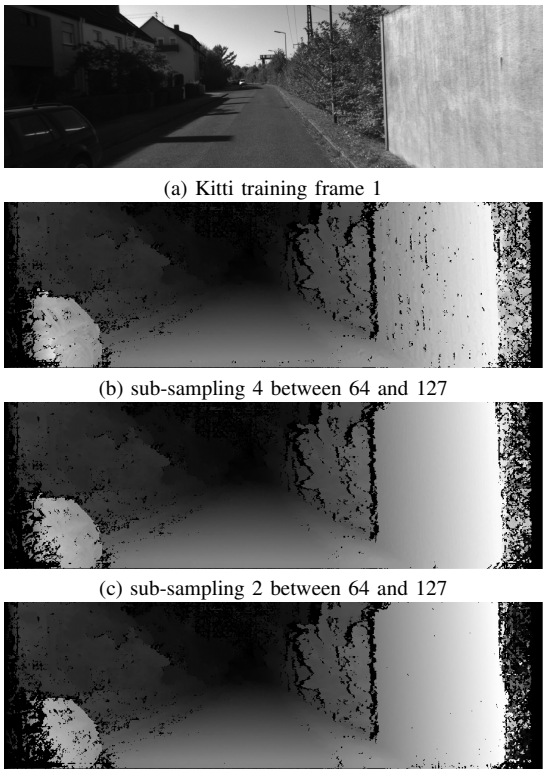


Fig. 7: Border width and quality for striped SGM with a border of 8 px on the KITTI training data set

At a width of 32 px the difference to the non-striped version is barely noticeable. The quantitative results for the KITTI training data set confirm this observation. Without an additional border, the number of erroneous pixels increases by more than 1% (Fig. 6) compared to the standard SGM method. An additional border of 16 pixels reduces this increase to 0.12% for the 2 px error threshold and 0.13% for the 3 px error threshold.

Varying the number of stripes with a constant border shows the expected degradation with increasing number of stripes (Fig. 7). As the stripes get too small, probability of cutting a less textured area increases.

The results for the testing data set (Table II) show that the method is comparable to other methods of the SGM family. Its runtime is the fastest of the top ranking methods. The runtime reported in Table II is higher than expected, since we use the Center-Symmetric Census measure as data cost, which is not optimized and due to the additional post-processing. We do a speckle-filtering and apply a bilateral filter approximation to smooth out small errors.



(a) Kitti training frame 1

(b) sub-sampling 4 between 64 and 127

(c) sub-sampling 2 between 64 and 127

(d) no sub-sampling

Fig. 8: Influence of disparity compression - In the upper three images the wall at the right side has disparities from 60 to 127.

TABLE I: Influence of disparity compression on the Kitti training data set - Out-Noc is the percentage of non-occluded pixels with a disparity error bigger than the threshold. SGM-DC2 uses a disparity compression of 2 for the upper 64 values. SGM-DC4 uses a disparity compression of 4.

method	Out-Noc 2 px	Out-Noc 3 px
SGM	8.91%	5.89%
SGM-DC2	9.24%	6.09%
SGM-DC4	10.25%	6.71%

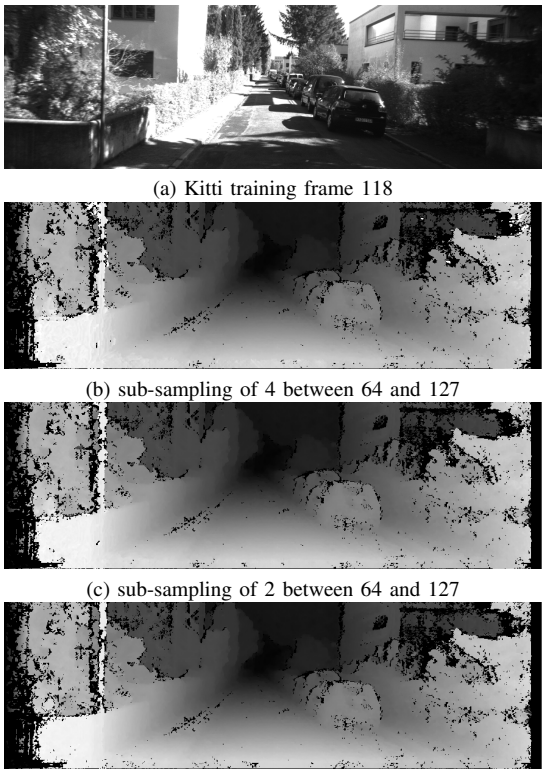
The effect of disparity compression is clearly visible in Fig. 8. Disparities on slanted surfaces are less smooth and at a sub-sampling of four the number of matched pixels decreases. In addition, the effectiveness of the Left-right-check is reduced (Fig. 9). The pole in the scene fattens in comparison to the not sub-sampled version. This seems to be an effect of image sub-sampling during the calculation of the compressed cost cube. The quality on the Kitti training data set is reduced by 0.33% for a sub-sampling of 2 and 1.34% for a sub-sampling of 4 for the 2 px error threshold (Table I).

C. Speed

Our input images for the speed evaluation have 12 bit intensity depth. Eight bit images are possible as well. The input image bit depth does not have a noticeable effect on

TABLE II: Evaluation on KITTI test set with error threshold 3 px. It shows the top 17 ranking methods at middle of January 2014 and the OpenCV Implementation of SGM with rank 27. SGM based methods are highlighted in bold: *Weighted Semi-Global Matching* wSGM [10], our *rapid SGM* with 4 stripes and a border of 16 rSGM, *Iterative Semi-Global Matching* iSGM [16], OCV-SGBM2 (anonymous submission), *Semi-Global Matching* SGM [3] and *OpenCV Semi-Global Block Matching* OCV-SGBM.

Rank	Method	Setting	Out-Noc	Out-All	Avg-Noc	Avg-All	Runtime	Environment
1	SceneFlow	fl ms	2.98 %	3.86 %	0.8 px	1.0 px	6 min	4 cores @ 3.0 Ghz (Matlab + C/C++)
2	PCBP-SS		3.40 %	4.72 %	0.8 px	1.0 px	5 min	4 cores @ 2.5 Ghz (Matlab + C/C++)
3	gtRF-SS		3.83 %	4.59 %	0.9 px	1.0 px	1 min	1 core @ 2.5 Ghz (Matlab + C/C++)
4	StereoSLIC		3.92 %	5.11 %	0.9 px	1.0 px	2.3 s	1 core @ 3.0 Ghz (C/C++)
5	PR-Sf+E	fl	4.02 %	4.87 %	0.9 px	1.0 px	200 s	4 cores @ 3.0 Ghz (Matlab + C/C++)
6	PCBP		4.04 %	5.37 %	0.9 px	1.1 px	5 min	4 cores @ 2.5 Ghz (Matlab + C/C++)
7	PR-Sceneflow	fl	4.36 %	5.22 %	0.9 px	1.1 px	150 sec	4 core @ 3.0 Ghz (Matlab - C/C++)
8	**wSGM**		4.97 %	6.18 %	1.3 px	1.6 px	6s	1 core @ 3.5 Ghz (C/C++)
9	ATGV		5.02 %	6.88 %	1.0 px	1.6 px	6 min	>8 cores @ 3.0 Ghz (Matlab + C/C++)
10	**rSGM**		5.03 %	6.60 %	1.1 px	1.5 px	0.3 s	4 cores @ 2.6 Ghz (C/C++)
11	**iSGM**		5.11 %	7.15 %	1.2 px	2.1 px	8 s	2 cores @ 2.5 Ghz (C/C++)
12	AARBM		5.14 %	6.20 %	1.1 px	1.2 px	0.4 s	1 core @ 3.0 Ghz (C/C++)
13	ALTGV		5.36 %	6.49 %	1.1 px	1.2 px	20 s	GPU @ 2.5 Ghz (C/C++)
14	**OCV-SGBM2**		5.38 %	6.50 %	1.0 px	1.2 px	2 s	1 core @ 2.5 Ghz (C/C++)
15	AABM		5.42 %	6.52 %	1.1 px	1.3 px	0.43 s	1 core @ 3.0 Ghz (C/C++)
16	RBM		5.50 %	6.48 %	1.2 px	1.3 px	0.2 s	1 core @ 3.0 Ghz (C/C++)
17	**SGM**		5.76 %	7.00 %	1.2 px	1.3 px	3.7 s	1 core @ 3.0 Ghz (C/C++)
27	**OCV-SGBM**		7.64 %	9.13 %	1.8 px	2.0 px	1.1 s	1 core @ 2.5 Ghz (C/C++)



(a) Kitti training frame 118

(b) sub-sampling of 4 between 64 and 127

(c) sub-sampling of 2 between 64 and 127

(d) no sub-sampling

Fig. 9: Influence of disparity compression - The pole at the left shows less distinct borders with increased sub-sampling.

the algorithm run-times, since only the Census transform is affected. We obtain execution-time differences below 1 ms.

The timings for the classical SGM formulation show, that our solution (referred as SGM in Table III) clearly outperforms the CPU implementation in [9], using only two instead of four cores. This is not only a result of the newer processor platform, but of the parallelization choices made. The solution is faster than the older GPU implementation of [8] and slightly slower than [7], but uses only 47 W at maximum. For larger disparity ranges, the solution gets even near to the FPGA implementation of [6], if we use the number of evaluated disparities per second as a metric. The detailed timings in Table IV show an almost direct dependency of data cost calculation and WTA from the number of disparities to evaluate. The path accumulation step is slightly below a factor of one and is the most time consuming.

The solution using image stripes with borders is able to reach shorter cycle times, as all 4 cores can be used during the whole algorithm. We call this method rapid SGM (rSGM). We obtain a frame rate of 19.6 Hz for VGA resolution and 64 disparity values, and 11.9 Hz for 128 disparity values. With disparity compression, a frame rate of 16 Hz for 128 disparities is reached. This type of sub-sampling induces little overhead, keeping the number of evaluated disparities high and providing significant speedup compared with the full SGM version.

VI. SUMMARY

We have created a system performing large scale semi-global matching on a CPU[1]. With this system, we can compute disparity maps suitable for intelligent vehicles and

[1]C++ source code on-line at http://userpage.fu-berlin.de/spangenb/

TABLE III: Timings - rSGM uses 4 stripes and a border of 16 pixels. rSGM-DC2 additionally uses a disparity compression of 2 for the upper 64 values. rSGM-DC4 uses a disparity compression of 4 for the upper 64 values

Method **Unit**	**Cores**	**image size** **[px]**	**disparities** **[px]**	**cycle** **[ms]**	**disp/s** $[10^6/s]$
GPU [8]		320×240	64	76	64
GPU [7]		640×480	64	85	230
FPGA [6]		2·340×200	$2 \cdot 64$	40	218
FPGA [5]		640×480	64	30	648
CPU [9]	4	640×320	128	224	117
SGM	2	640×480	64	105	187
SGM	2	640×480	128	184	214
rSGM	4	640×480	64	51	390
rSGM	4	640×480	128	84	468
CPU [9]	4	268800	128	69	65
rSGM-DC2	4	640×480	128	71	415
rSGM-DC4	4	640×480	128	62	396

TABLE IV: Detailed timings for the full SGM variant

Algorithmic step **#disparities** **time**	**640×480** **64** **[ms]**	**640×480** **128** **[ms]**	**1248×384** **128** **[ms]**
5×5 Census	1	1	1
Data cost calculation	7	14	21
Path accumulation	76	137	217
WTA left & right	16	30	46
Median filtering	1	1	2
Left-right consistency check	1	1	2
Sub-pixel interpolation	2	2	5

autonomous driving without special additional hardware. Even images with higher resolution than VGA can be processed in around 0.3 s. The induced load is low enough to enable subsequent processing steps such as free-space calculation or vehicle detection. The key is an algorithm that enables low- and high-level parallelism without the need for explicit synchronization. Furthermore, the method is scalable to the number of available processing cores. We look forward to an implementation using AVX2 instructions to further increase the SIMD parallelism.

REFERENCES

[1] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *INTERNATIONAL JOURNAL OF COMPUTER VISION*, vol. 47, pp. 7–42, 2001.

[2] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *CVPR*. IEEE, 2012, pp. 3354–3361.

[3] H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 2, pp. 328–341, 2008.

[4] M. Humenberger, T. Engelke, and W. Kubinger, "A census-based stereo vision algorithm using modified semi-global matching and plane fitting to improve matching quality," 2010, pp. 77–84.

[5] M. Buder, "Dense real-time stereo matching using memory efficient semi-global-matching variant based on fpgas," pp. 843 709–843 709–9, 2012.

[6] S. K. Gehrig, F. Eberli, and T. Meyer, "A real-time low-power stereo vision engine using semi-global matching," in *ICVS*, ser. Lecture Notes in Computer Science, M. Fritz, B. Schiele, and J. H. Piater, Eds., vol. 5815. Springer, 2009, pp. 134–143.

[7] M. Michael, J. Salmen, J. Stallkamp, and M. Schlipsing, "Real-time stereo vision: Optimizing semi-global matching," in *Intelligent Vehicles Symposium (IV), 2013 IEEE*, 2013, pp. 1197–1202.

[8] I. Ernst and H. Hirschmüller, "Mutual information based semi-global stereo matching on the gpu," in *Proceedings of the 4th International Symposium on Advances in Visual Computing*, ser. ISVC '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 228–239.

[9] S. K. Gehrig and C. Rabe, "Real-Time Semi-Global Matching on the CPU," in *Proceedings of the IEEE Computer Vision and Pattern Recognition Workshops*, San Francisco, CA, USA, June 2010, pp. 85–92.

[10] R. Spangenberg, T. Langner, and R. Rojas, "Weighted semi-global matching and center-symmetric census transform for robust driver assistance," in *Computer Analysis of Images and Patterns*, ser. Lecture Notes in Computer Science, R. Wilson, E. Hancock, A. Bors, and W. Smith, Eds. Springer Berlin Heidelberg, 2013, vol. 8048, pp. 34–41.

[11] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, "A fast stereo matching algorithm suitable for embedded real-time systems," *Comput. Vis. Image Underst.*, vol. 114, no. 11, pp. 1180–1202, Nov. 2010.

[12] H. Hirschmüller and D. Scharstein, "Evaluation of stereo matching costs on images with radiometric differences," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 9, pp. 1582–1599, 2009.

[13] H. Hirschmüller and S. K. Gehrig, "Stereo matching in the presence of sub-pixel calibration errors," in *CVPR*. IEEE, 2009, pp. 437–444.

[14] W. Mua, "Ssse3: Fast popcount (accessed 4.1.2014)," 2010. [Online]. Available: http://wm.ite.pl/articles/sse-popcount.html

[15] M. Shimizu and M. Okutomi, "Sub-pixel estimation error cancellation on area-based matching," *Int. J. Comput. Vision*, vol. 63, no. 3, pp. 207–224, July 2005. [Online]. Available: dx.doi.org/10.1007/s11263-005-6878-5

[16] S. Hermann and R. Klette, "Iterative semi-global matching for robust driver assistance systems," in *ACCV (3)*, ser. Lecture Notes in Computer Science, K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, Eds., vol. 7726. Springer, 2012, pp. 465–478.