Efficient dense stereo matching using CUDA

Ke Zhu [1], Matthias Butenuth [1] & Pablo d'Angelo [2]

Technische Universität München

Remote Sensing Technology

1) Technische Universität München, Remote Sensing Technology
{ke.zhu} | {matthias.butenuth}@bv.tum.de

2) The Remote Sensing Technology Institute, German Aerospace Center
pablo.angelo@dlr.de

Dense Stereo Processing using Semi-Global Matching

The proposed work demonstrates the general strategy for parallelization of dense matching methods on GPUs, shows the potential capability of common graphics cards for general computation, and compares the implementations between local and global methods with the example of Sum of Absolute difference (SAD) and Semi-Global Matching (SGM).

The Semi-Global Matching method approximates a global, 2D smoothness constraint by combining many 1D constraints from different aggregation directions for pixelwise matching. The global energy for the disparity image is defined as *E(D)*:

$$E(D) = \sum_p (C(p, D_p) + \sum_{q \in N_p} P_1 T\left[|D_p - D_q| = 1\right] + \sum_{q \in N_p} P_2 T\left[|D_p - D_q| > 1\right]$$

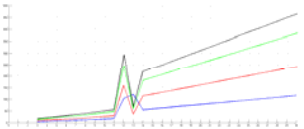
The first term sums the costs of all pixels in the image with their particular disparities D_p. The next two terms penalize the discontinuities with penalty factors P_1 and P_2, which differ in small or large disparity difference within a neighborhood q of the pixel *p*. This minimization approximation is realized by aggregating *S(p,d)* of path wise costs into a cost volume:

$$S(p, d) = \sum_r L_r(p, d)$$

In this study, the Semi-Global Matching (SGM) method is used as the stereo algorithm for evaluating different matching costs because of its robustness, speed and accuracy.

Comparison with local method (SAD)

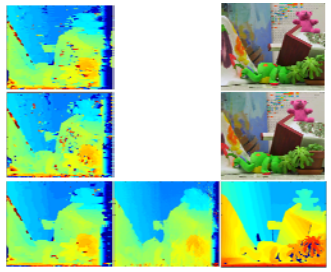
The comparison shows by the implementation of dense matching method on GPU, the global methods keep their accuracy advantage and the cost/performance ratio of local matching methods is not beneficial for a fast processing on GPUs. Thus, the semi-global methods like the SGM perform a better and more efficient result.



The run-time analysis of the SAD method with different fixed window sizes in *x* direction: 3 x in blue , 5x in red, 7x in green and 9 x in black.

	3x	5x	7x	9x	11x	13x	15x	17x	19x	21x	23x	25x	27x	29x
3x	4.9	8.4	10.0	14.3	18.6	121.0	59.5	67.1	74.7	82.2	89.9	97.3	104.9	113.2
5x	9.5	13.8	18.1	24.6	30.1	37.3	122.5	138.4	154.3	170.2	186.6	201.9	217.8	234.6
7x	14.6	22.7	29.3	38.3	48.1	59.5	194.6	220.0	245.2	270.6	295.9	322.0	346.6	371.9
9x	17.3	26.2	35.9	45.6	56.9	69.7	235.4	266.3	296.9	327.7	358.4	389.9	419.9	451.4

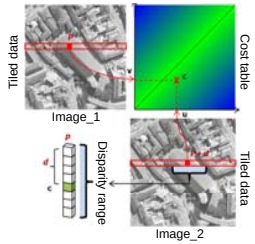
Run-time increasing with different window sizes in x-direction and changed sizes in y-direction.



The results comparisons between the SAD method with different window-sizes and the SGM result.

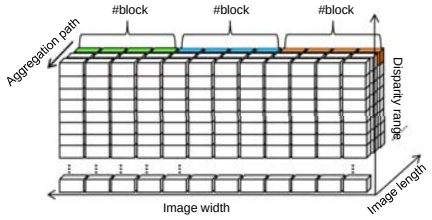
Implementation of SGM using CUDA

Matching cost calculation:
The values in the image are partitioned line-by-line and tiled into the shared memory to reduce the memory accesses on global memory and increase the data utilization rate, because each pixel from the right image can be used (disparity range - 1) times. Each thread in a block answers to a pixel in the image line.



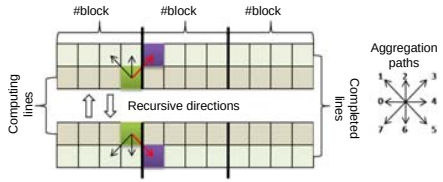
Cost aggregation: The cost optimization for each pixel in one direction requires the storage of both, the computed cost values and the aggregated costs from the previously visited pixel. A pixel in the image contains disparity range data elements in the cost cube, in which each concerned element of them maps to a thread. The block size is depended on the disparity range and the number of pixels inside each block.

The optimized results backwards along a path are used for the actual optimization. The results must be rewritten into the global memory for the aggregation with other paths.



Block tiling for the cost aggregation

The fast aggregation achieves the cost optimization in six directions with two passes through the images. Aggregation can be extended for more directions, if the sweepings start from the other sides of the image.



Fast aggregation processing within one sweeping

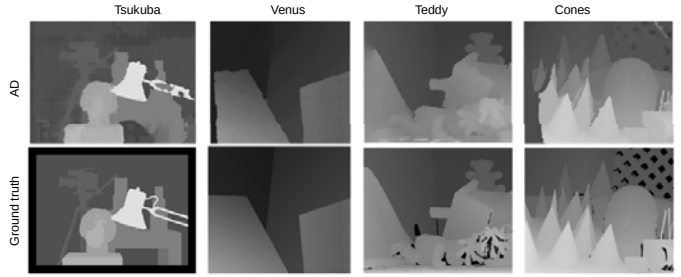
References

Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision 47 (2002) 7-42

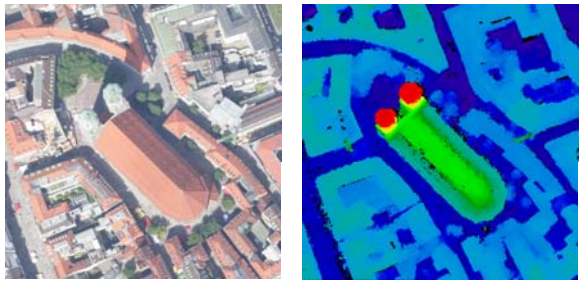
Hirschmller, H.: Stereo processing by semi-global matching and mutual information. IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (2008) 328-341

GPU Performance

The experimental results are computed on a NVIDIA GeForce GTX 295 graphics card. The GPU implementations use the Middlebury Stereo datasets as well as aerial photos.

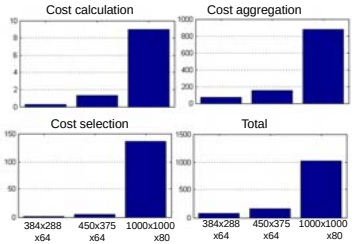


Results on the Middlebury Datasets for SGM with AD



Result on the remote sensing data

The compared CPU implementation runs on an Intel Core2 Q9450 CPU with 6 MB L2 Cache. The CPU implementation needs about 5200 ms to finish the stereo processing including rectification. In contrast, the CUDA improvement requires 722ms for six aggregation directions and 1120 ms for eight aggregations totally. The run-time using CUDA on small images with 384x288 pixels and a disparity range of 64 reaches 13 fps.



The SGM GPU run-times on different image sizes

	CPU Implementation	GPU Implementation
Rectification	432 ms	96 ms
Cost calculation	200 ms	9 ms
Cost aggregation	4215 ms	481 ms
Disparity selection	362 ms	136 ms
Total	5209 ms	722 ms

Run-time comparison between CPU and GPU SGM implementation