ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume I-3, 2012
XXII ISPRS Congress, 25 August – 01 September 2012, Melbourne, Australia

MEMORY EFFICIENT SEMI-GLOBAL MATCHING

Heiko Hirschmüller and Maximilian Buder and Ines Ernst

Commission ICWG III/VII

KEY WORDS: Stereoscopic, Matching, Real-time, Robotics, Hardware

ABSTRACT:

Semi-Global Matching (SGM) is a robust stereo method that has proven its usefulness in various applications ranging from aerial image matching to driver assistance systems. It supports pixelwise matching for maintaining sharp object boundaries and fine structures and can be implemented efficiently on different computation hardware. Furthermore, the method is not sensitive to the choice of parameters. The structure of the matching algorithm is well suited to be processed by highly paralleling hardware e.g. FPGAs and GPUs. The drawback of SGM is the temporary memory requirement that depends on the number of pixels and the disparity range. On the one hand this results in long idle times due to the bandwidth limitations of the external memory and on the other hand the capacity bounds are quickly reached. A full HD image with a size of 1920 $\times$ 1080 pixels and a disparity range of 512 pixels requires already 1 billion elements, which is at least several GB of RAM, depending on the element size, wich are not available at standard FPGA- and GPU-boards. The novel memory efficient (eSGM) method is an advancement in which the amount of temporary memory only depends on the number of pixels and not on the disparity range. This permits matching of huge images in one piece and reduces the requirements of the memory bandwidth for real-time mobile robotics. The feature comes at the cost of 50% more compute operations as compared to SGM. This overhead is compensated by the previously idle compute logic within the FPGA and the GPU and therefore results in an overall performance increase. We show that eSGM produces the same high quality disparity images as SGM and demonstrate its performance both on an aerial image pair with 142 MPixel and within a real-time mobile robotic application. We have implemented the new method on the CPU, GPU and FPGA. We conclude that eSGM is advantageous for a GPU implementation and essential for an implementation on our FPGA.

1 MOTIVATION

The justification for SGM arises from the limitations of other global approaches for the real-world applications.

Graph Cuts (Kolmogorov and Zabih, 2001, Szeliski et al., 2008) is a global stereo method that translates the matching problem into a graph and seeks a partition, which effectively approximates the minimization of a global energy. The method is not only slow, but also quite memory intensive. Belief Propagation (Felzenszwalb and Huttenlocher, 2004, Szeliski et al., 2008) is another approach for minimizing a global cost function. It passes messages iteratively in a three dimensional grid structure for approximating the global minimum. The size of the temporary memory is linear to the number of pixels and the disparity range, as explained above. A Dynamic Programming stereo matching method (Birchfield and Tomasi, 1998) can be implemented with much less memory requirements, since the method optimizes the global function along each scanline separately. Unfortunately, this leads to streaking artifacts.

The Semi-Global Matching (SGM) method (Hirschmüller, 2008) minimizes a global cost function only in one dimension like Dynamic Programming, but the direction is not oriented along scanlines. Instead, it is performed symmetrically from eight directions towards all pixels in the image. SGM does not suffer from streaking artifacts like Dynamic Programming and does not require iterations like Belief Propagation. The rather simple and regular integer operations of the SGM algorithm make it suitable for implementations on the GPU (Rosenberg et al., 2006, Gibson and Marques, 2008, Ernst and Hirschmüller, 2008) and FPGA (Gehrig et al., 2009, Banz et al., 2010), which permits real-time performance.

In practice, SGM appears rather robust and insensitive to the choice of parameters in contrast to other methods like Graph Cuts. This makes SGM suitable for real world applications like (tilewise) aerial image matching (Hirschmüller, 2008, Gehrke et al., 2010) and automotive applications (Steingrube et al., 2009).

The drawback of SGM is its memory consumption that depends, as for all global methods, on the number of pixels and the disparity range. This limitation may be overcome in principle with tilling the input data. While this approach is usually sufficient for CPUs it does not take advantage of the high degree of parallelism commonly found in FPGAs and GPUs.

2 MEMORY EFFICIENT SEMI-GLOBAL MATCHING

The SGM method (Hirschmüller, 2008) aims to minimize the global cost function

$$E(D) = \sum_{\mathbf{p}} (C(\mathbf{p}, D(\mathbf{p})) + \sum_{\mathbf{q} \in N_\mathbf{p}} P_1 \, \mathrm{T}[|D(\mathbf{p}) - D(\mathbf{q})| = 1] + \sum_{\mathbf{q} \in N_\mathbf{p}} P_2 \, \mathrm{T}[|D(\mathbf{p}) - D(\mathbf{q})| > 1]). \quad (1)$$

The function consists of one term that sums the matching costs C over all pixels $\mathbf{p}$, according to given disparities $D_\mathbf{p}$. For rectified images, the absolute difference would be calculated by

$$C(\mathbf{p}, d) = |I_L(\mathbf{p}) - I_R(\mathbf{p} - d)|. \quad (2)$$

This just serves as an example. In practice, more robust matching costs should be used (Hirschmüller and Scharstein, 2009). The second term of (1) sums small penalties P_1 for all pixels, where the disparity difference to the neighbor is at most one pixel. The last term sums a higher penalty P_2 for all pixels with a higher

disparity difference to neighboring pixels. The goal is to find the disparity image D that minimizes this function.

2.1 Review of the SGM Method

In SGM (Hirschmüller, 2008), the minimization of (1) is done by going along one dimensional paths L in eight directions $\mathbf{r}$ through the image. Thus, at each pixel $\mathbf{p}$, paths from eight directions meet as shown on the left in Figure 1.

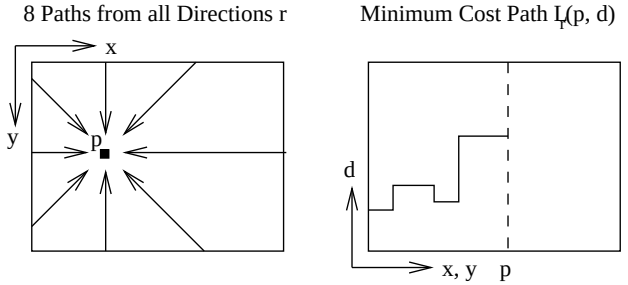

Figure 1: Aggregation of costs in disparity space.

Along each path $L_{\mathbf{r}}$, the minimum cost to reach all disparities of a pixel $\mathbf{p}$ on the path is computed according to the global cost function (1). The minimum cost along a path is visualized on the right in Figure 1. Mathematically, the cost computation is done by recursively computing

$$\begin{aligned} L_{\mathbf{r}}(\mathbf{p},d) = & C(\mathbf{p},d) + \min(L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d), \\ & L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d-1)+P_1, \\ & L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d+1)+P_1, \\ & \min_i L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},i)+P_2) - \min_k L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},k). \end{aligned} \tag{3}$$

Thus, for each pixel $\mathbf{p}$ and disparity d, the cost is computed by the sum of the matching cost and the minimum path cost of the previous pixel $\mathbf{p}-\mathbf{r}$, considering the penalties P_1 and P_2. The latter is done by computing the minimum over four values. The first value is the path cost at the previous pixel at the same disparity. This value is taken without any penalty. The second and third value is the path cost at the previous pixel with the next lower and higher disparity. Here, the small penalty P_1 is added. The last value is the minimum cost at the previous pixel over all disparities with the additional higher penalty P_2.

The last term of equation (3) subtracts the minimum cost at the previous pixel from all costs of the current pixel. This is done for keeping the values L low for using a small data type. In fact, an arbitrary value could be chosen, as long as it is constant for all disparities. The minimum of the previous pixel is used, because it is already available and subtraction will never make the whole term negative.

Since disparity changes are usually indicated by intensity changes, the penalty P_2 is adapted to the intensity gradient between the current and the previous pixel, according to

$$P_2 = \frac{P_2'}{|I_L(\mathbf{p}) - I_L(\mathbf{p}-\mathbf{r})|}. \tag{4}$$

The information from all paths is fused for all pixels and disparities by

$$S(\mathbf{p},d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p},d). \tag{5}$$

The disparity for each pixel corresponds to the minimum cost, i.e.

$$D_L(\mathbf{p}) = \underset{d}{\operatorname{argmin}}\, S(\mathbf{p},d). \tag{6}$$

Subpixel interpolation can be performed by either fitting a parabola through the neighboring cost values or by using an equiangular line fit (Shimizu and Okutomi, 2005). The choice depends on the matching cost.

Unlike Dynamic Programming solutions, the pathwise cost computation does not contain any handling for occlusions. This is not possible because the paths are not oriented in the direction of epipolar lines. Therefore, the disparity of the right image D_R is computed by either a diagonal search in S (Hirschmüller, 2008), or by switching the roles of the left and right image and computing from scratch. A consistency check is performed by comparing D_L with D_R and differing disparities are set to invalid or interpolated if needed (Hirschmüller, 2008).

Aggregation can be performed in two passes. The first pass goes from the top left pixel line wise through the image and computes the path from left, diagonally from top-left, from top and diagonally from top-right. For each pixel, the four paths are continued from the previous pixels to the current pixel according to (3) as shown on the left in Figure 2. This requires storing the path costs L_r for the previous pixels for all disparities for each direction separately. Consequently, the required memory has a size of $3 \times w \times d_{max} + d_{max}$ elements, with w as width of the image and d_{max} as disparity range. After computing the four pathwise costs for a pixel, the result is added according to (5) to the array S, which has the size $w \times h \times d_{max}$ elements.

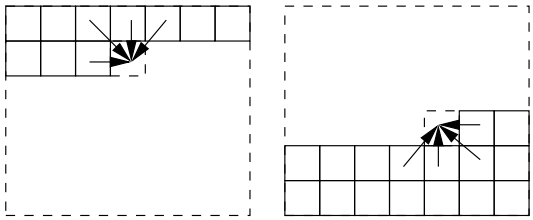

Figure 2: Calculation of the eight path directions in a top-down pass (left) and a bottom-up pass (right).

Thereafter, a second pass is required that starts from the bottom, right pixel and goes upwards for computing the remaining four paths. This completes the computation of S. The disparity image is derived from S immediately according to equation (6). Thus, the total temporary memory requirement of SGM is

$$M_{sgm} = w \times h \times d_{max} + 3 \times w \times d_{max} + d_{max}. \tag{7}$$

The main problem is the memory size of S as it depends on the width, height and disparity range. In contrast, the memory for pathwise costs only depends on the width and disparity range. Since a disparity range that is larger than the total image size does not make sense, the required memory will be at most in the order of the size of the input image.

2.2 The Novel eSGM Method

In the SGM algorithm, the disparity is determined as the index that corresponds to minimum cost of all disparities of a pixel (6). However, each of the eight paths that contribute to the cost for a pixel, carries its own preference for the location of the minimum. Figure 3 shows the costs of the eight paths from different directions at pixel $\mathbf{p}$ for all disparities. Ideally, the location of the

minimum of the eight paths would be the same disparity. However, we have to anticipate that paths from some directions may be disturbed, e.g. near depth discontinuities. Nevertheless, at least one path should predict the disparity correctly. The idea is to focus only at the locations of S, where the eight paths have their minima. These are at most eight distinct places. For all other disparities, the costs need not to be stored. Thus, S becomes a sparse S', whose size does not depend on the number of disparities any more.

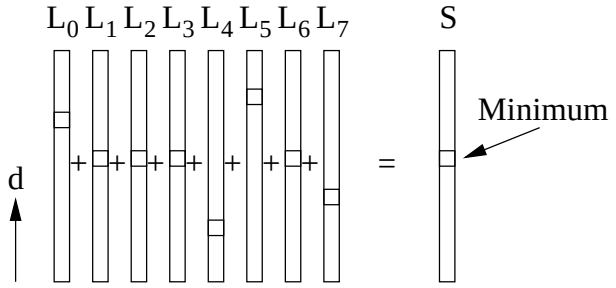

Figure 3: Costs for all disparities of a certain pixel from paths of eight different directions that are summed to S. The minimum of S may differ from the minima of the pathwise costs.

The immediate question is if we do not reduce the number of possible disparities too early? What if the minimum of S is at a different location than the minimum of any of the eight paths? The answer is that it is very unlikely that *true* disparity is not detected by any of the paths *and* at the same time appears as minimum in S. We claim that if the minimum of S is at a different place than any of the eight pathwise minima, then it can only be the correct disparity by pure chance. Thus, the new eSGM method should have qualitatively the same output as the SGM method.

The new method can be implemented with three passes. The first pass works like in the original method, except for storing the result to S. Instead of storing the costs for all disparities, for each pixel the four minima of the four paths that meet in this pixel are determined and the costs are only stored for these four disparities. For subpixel interpolation, the costs of the next lower and higher disparity is stored as well. Figure 4 shows the values that need to be stored for one pixel. The second pass computes the remaining four paths from the bottom up. The costs are added at the places that were identified by the first pass. These four places are complete, which allows the computation of an intermediate result by choosing the lowest cost among the four and calculating the sub-pixel disparity position. This frees the memory of the four minima of the first pass. The memory is used for storing the costs at the four minima of the paths from the second pass, which are in general different to the minima of the first pass. The aggregation is completed by a third pass that computes the same paths as in the first pass, but adds the costs at the minima that where identified in the second pass. The final minima of each pixel can then be selected among these four minima and the intermediate result of the second pass.

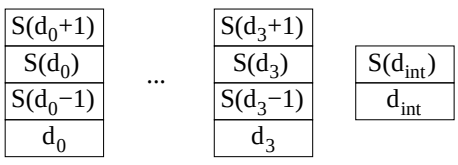

Figure 4: Definition of 18 data elements that need to be stored for each pixel in the eSGM method.

Thus, for this algorithm, the amount of temporary memory is just

$$M_{esgm} = w \times h \times 18 + 3 \times w \times d_{max} + d_{max}. \quad (8)$$

Obviously, the computational effort of eSGM is increased by 50% in comparison to SGM due to the necessary third pass.

The new eSGM method does not allow the direct derivation of the right disparity image from the cost array S by a diagonal search (Hirschmüller, 2008). Therefore, we suggest another kind of fast consistency check that projects D_L directly into D_R by considering the costs that are available at the disparities of D_L. For rectified images it can be written as $D_R(x - D_L(x,y), y) := D_L(x,y)$. Double mappings are resolved by storing the disparity with the lower associated cost into D_R.

The reasoning behind the new method permits the derivation of matching confidence as the number of paths that have the minimum at the same disparity as the final disparity. Obviously, if paths from all directions have the same minimum, then the trust in the found disparity is very high. In contrast, if other path directions suggest a different disparity, then the confidence in the correctness of the match is lower. Computing this confidence number is computationally very cheap, because the eSGM algorithm already computes the individual minima of all paths.

3 IMPLEMENTATIONS

We have implemented the new method on different hardware platforms for different applications. As matching cost, we have focused on Census, since this matching cost appears to have the highest radiometric robustness (Hirschmüller and Scharstein, 2009).

3.1 CPU

We have implemented the SGM and eSGM method on the CPU as explained in Section 2 using unsigned integer values with 16 bit as basic elements. The pathwise cost calculation loop is computed using SSE2 vector commands. In case of SGM, the Census matching cost was computed only once and the pairwise matching costs stored for each pixel and disparity, which doubles the memory requirement, but offers fastest processing. Thus, the total temporary memory requirement in bytes of our SGM implementation is four times equation (7).

Our eSGM implementation uses unsigned 16 bit integer values as well and computes the Hamming distance of Census transformed images for each of the three passes, using *xor* and *popcnt* commands, that are part of the SSE 4.2 or SSE 4a specification. Thus, the total amount temporary memory in bytes is two times equation (8).

3.2 GPU

The GPU implementation of eSGM uses conventional OpenGL/Cg (Segal and Akeley, 2009, NVI, 2009) programming techniques (Rosenberg et al., 2006, Ernst and Hirschmüller, 2008). Several render buffers of an OpenGL frame buffer object keep the data in a 16 bit float data format while the arithmetic operations are usually done in 32 bit float precision. All the work is carried out through the execution of OpenGL rendering commands with specialized fragment and vertex shaders. With respect to the memory bandwidth, necessary for direct matching cost calculation, we have chosen a Census cost function with a 5×5 window and alternatively Mutual Information as matching cost (Hirschmüller and Scharstein, 2009). Mutual Information requires a hierarchical eSGM algorithm.

Due to the two-step approach, the computing time for the path cost values is theoretically doubled as compared to SGM. It can be reduced if the minimal path cost values are saved during the