

Calculating Dense Disparity Maps from Color Stereo Images, an Efficient Implementation

Karsten Mühlmann, Dennis Maier, Jürgen Hesser, Reinhard Männer
{muehlmann;maier;hesser;maenner}@ti.uni-mannheim.de

Lehrstuhl für Informatik V, Universität Mannheim, B6, 23-29, C, D-68131 Mannheim, Germany

Abstract—This paper presents an efficient implementation for correlation based stereo. Research in this area can roughly be divided in two classes: improving accuracy regardless of computing time and scene reconstruction in real-time. Algorithms achieving video frame rates must have strong limitations in image size and disparity search range, whereas high quality results often need several minutes per image pair. This paper tries to fill the gap, it provides instructions how to implement correlation based disparity calculation with high speed and reasonable quality that can be used in a wide range of applications or to provide an initial solution for more sophisticated methods. Left-right consistency checking and uniqueness validation are used to eliminate false matches. Optionally, a fast median filter can be applied to the results to further remove outliers. Source code will be made publicly available as contribution to the Open Source Computer Vision Library, further acceleration with SIMD instructions is planned for the near future.

Keywords—Stereo, Disparity, Correlation

I. INTRODUCTION

STEREO matching is used in many applications, like 3D-reconstruction, autonomous vehicles and augmented reality.

The use of color images becomes more and more common, particularly because color cameras are available at all prices and they are superseding the monochrome ones. In many cases, color information brings a substantial gain.

We try to deliver a disparity map whose quality is sufficient for a lot of applications "out of the box". Alternatively, the techniques described here can be used as starting point for implementing improved algorithms. Many of them aiming at high quality need an initial guess for the disparity map [1],[6] or disparity space [15]. The algorithm described in this paper can as well be used as a fast first stage for those algorithms.

The implementation is biased towards speed, however we do not want any restrictions that real-time algorithms have [11]. For an image size of 160×120 pixels, 20fps can be achieved, but images of several megapixels can be matched as well.

Although the details are important here, we do not descend into the lowest level, interested readers are en-

couraged to examine the source code that will be available with the Open Source Computer Vision Library.

II. IMPLEMENTATION

Our algorithm requires pairs of rectified images, which means that corresponding epipolar lines are horizontal and on the same height. Then, interpolation has to be done only once per image and the search of the corresponding pixel takes place in horizontal direction only, resulting in a significant speedup compared to searching along the epipolar lines of the unrectified images. For simplicity, it is assumed that the images are of the same size.

The disparity map is calculated for the left image only. Since only matches are kept for which the left-to-right consistency check is positive, the disparity map for the right image does not provide additional information.

A. Disparity Space Volume

For every pixel in the left image our goal is to find the corresponding pixel in the right image (a match). Matching single pixels is nearly impossible, therefore every pixel is represented by a small region containing it, called correlation window. We chose the window to be centered around the pixel and of constant size. This choice has some drawbacks, a discussion how to overcome them is beyond the scope of this paper, see for example [6],[15].

Difference values are stored in memory in a cuboid (figure 1, [9],[13]) whose dimensions are given by the width w and height h of the images and the disparity search range $d_{min} - d_{max}$.

The difference measure between the window around (x, y) in the left image and the window $(x + d, y)$ in the right image is stored at position (x, y, d) in the volume.

B. Correlation Measure

If the correlation measure is separable in x and y and all pixels within the correlation window are equally weighted, an efficient implementation is possible. We

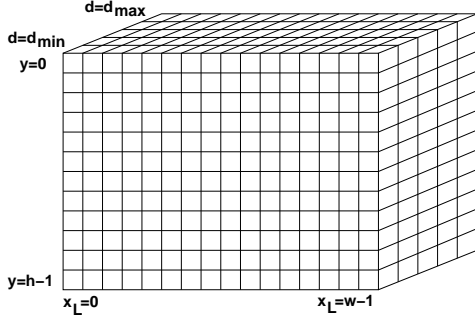


Fig. 1. Volume defined by the dimensions of the left image and the disparity search range.

chose the sum of absolute RGB differences (SAD), because it performs better than the sum of squared differences in the presence of outliers and it has a smaller computational complexity than normalized correlation measures. According to [10], using color information instead of gray values improves the signal to noise ratio by 20%–25%. Particularly in medical images the use of color gives substantial improvements.

$$\text{SAD}(x, y, d) = \sum_{i=-\frac{1}{2}(\text{win}_x-1)}^{\frac{1}{2}(\text{win}_x-1)} \sum_{j=-\frac{1}{2}(\text{win}_y-1)}^{\frac{1}{2}(\text{win}_y-1)} [|R_L(x+i, y+j) - R_R(x+i+d, y+j)| + |G_L(x+i, y+j) - G_R(x+i+d, y+j)| + |B_L(x+i, y+j) - B_R(x+i+d, y+j)|]$$

Where R , G and B are the color components of the respective pixel, index L stands for the left image, index R for the right one. The dimensions of the correlation window win_x and win_y have to be uneven numbers, otherwise there would be an offset of half a pixel between the disparity map and the left image [12].

Using normalized correlation measures requires an additional processing stage that calculates and stores sums of the R , G and B values for every correlation window. Building these sums can be accelerated in the same way as the sum of the differences, see section II-D. In our applications we control exposure and white balance of the cameras to minimize the difference of brightness and contrast between the images. So far it has been sufficient to use the unnormalized form.

C. Memory Organization

There are several possibilities how to store the volume in memory. Three promising approaches will be discussed in the following paragraphs. They differ in how the volume is divided into slices and which variable runs fastest inside a slice.

C.1 x fastest, y , slices in d

Very straightforward implementation, complete slices for constant d are computed. However, for every d -slice both images must be read completely and the complete disparity space volume has to be stored, which can be prohibitive for very large images and search ranges. Two runs through the volume are needed, because the search cannot start until the SAD for all disparities has been calculated. Searching the minimum along the disparity direction requires large jumps in memory between the slices.

C.2 x fastest, d , slices in y

More compact layout, since every y slice is used only $\text{win}_y + 2$ times (figure 4) and can be discarded afterwards. This allows to implement a ring buffer in y , reducing the memory consumption significantly. Even if x runs fastest in memory, it is advisable to use d for the innermost loop and compute all disparities for a fixed pixel in the left image. The RGB value of this pixel is read only once and kept in a dedicated register, thus every pixel in the left image is accessed only once during the whole computation.

C.3 d fastest, x , slices in y

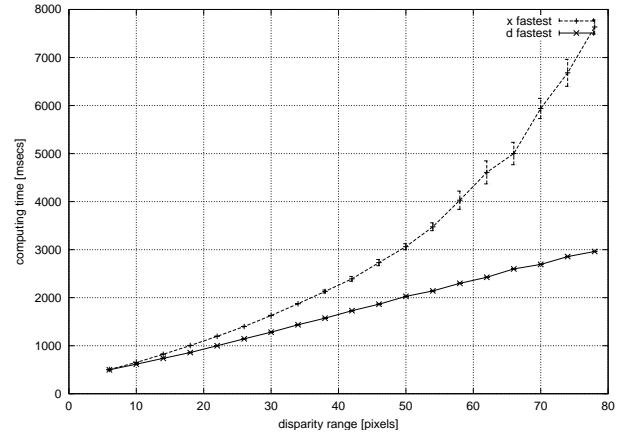


Fig. 2. Comparison of running times for two different memory layouts, varying only the disparity search range for an image size of 752×582 pixels. For every disparity range value, mean and standard deviation of 10 runs are shown.

Performs better than the previous variant. Loading a cache line into the Pentium III processor involves 32 bytes [8]. Using one value in the disparity space volume brings seven other values with it into the L1-cache (using 32bit unsigned integers for the SAD). In case x runs fastest, seven other values in x direction are loaded, but we don't need them before the search at the current x position has been completed. It is

very likely, that these values will not survive in the cache until they are used in a calculation. In contrast, the seven following values in d -direction *will* be needed during the search.

Figure 2 shows a comparison of the last two variants. If x runs fastest, the probability of a cache miss increases with growing disparity search range. There is also a bigger variation of computation time for longer runs, most likely due to other tasks being scheduled and polluting the cache. Additionally, we're using an SMP system and sometimes during longer runs the calculating thread loses its processor affinity. For small search ranges and in the case d runs fastest, the variation of running time is negligible.

D. Summing up the Windows

It can easily be seen, that the RGB difference of one pixel pair is required in $win_x \times win_y$ windows that contain the two pixels. By using a sliding window technique to calculate the window sums ([12], [4]), this difference does not have to be recomputed for every window.

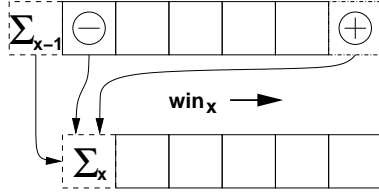


Fig. 3. Sliding window sum in x -direction.

Figure 3 shows how the differences between pixels are accumulated along the x -direction to build a row sum. We only need the sum of the first window as bootstrap. Every following sum is calculated from the previous one with one addition and one subtraction and then stored in x -direction immediately behind the new window (if d -first order is used to store the volume, section II-C.3, the stride between successive x -positions is $d_{max} - d_{min} + 1$).

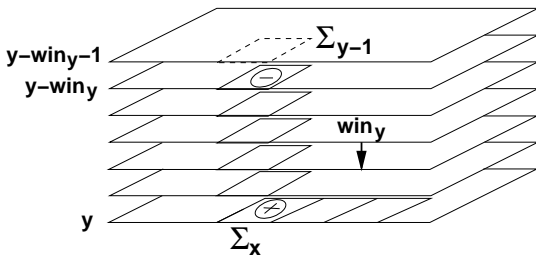


Fig. 4. Sliding window sum in y -direction.

Summation in y direction is is done in an analogous

manner (figure 4). As soon as a row sum is computed, it is added to the window sum stored $y - win_y - 1$ slices above (belonging to the window one pixel above the current one). The row sum $y - win_y$ slices above is subtracted and the final sum is stored in its place. Using the row sum as soon as it is available prevents a reload from memory and takes advantage of the cache.

The sum of a window is stored $(win_x + 1)/2$ positions left of and $(win_y + 1)/2$ positions above its center pixel. This has to be compensated by a constant offset when storing the disparity map.

To avoid special treatment at the image borders, we add a small overscan area of dimension $(win_x + 1) \times (d_{max} - d_{min} + 1)$ ahead of every slice that allows negative x coordinates. In the beginning it is filled with zeros, this way the summation of the first win_x elements can be done with the sliding window as well. Clearing the slices of the ring buffer prior to the calculation ensures that the window sum in y is added up correctly for top win_y image rows.

The summation stops with the last valid pixel position in a row or column, we do not have an overscan area at the end. As a consequence, the border of the disparity map ($x < (win_x - 1)/2$, $x > w - (win_x + 1)/2$, $y < (win_y - 1)/2$ or $y > h - (win_y + 1)/2$) is considered invalid.

E. Searching Minima

As already said in section II-C.3, the outer loops run over y and x in the left image. While the inner loop runs over d , the minimum SAD value and its position are memorized. If the minimum is not unique (section II-H) or its position is d_{min} or d_{max} , the match is discarded, otherwise the position of the minimum is stored as left-right disparity.

Some care has to be taken at the borders to avoid pixel positions outside the images. We divide the d - x slice of the disparity space volume in up to three parts for which the whole disparity range or only part of it has to be considered (outlined in bold in figure 5 and 6). This is done by three separate loops over x , for which the first and last d can be precalculated.

The small squares in the middle of the left parts in figure 5 and 6 illustrate the search region for a sample pixel at (x, y) in the left image.

After the left-right calculation has finished for a slice, the right-left disparities are determined. The computed SADs can be reused, only the search has to be performed.

Which positions in a slice have to be searched for a fixed window position (x, y) in the *right* image? To compare it with the window (x, y) in the left image, we use the SAD value at $(x, y, 0)$. For the comparison

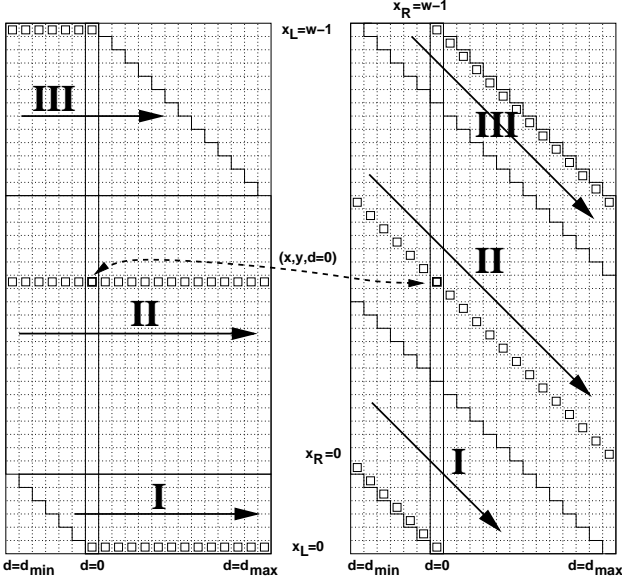


Fig. 5. Search directions in the case $d_{min} < 0 < d_{max}$. Left figure: from left to right. Right figure: right to left. The search range is restricted to the area where both pixels lie inside the images.

with the left window at $(x + 1, y)$, the SAD we are looking for is located at $(x + 1, y, -1)$, and so forth. The resulting pattern is a diagonal that contains the point $(x, y, 0)$, shown as small squares in the right parts of figure 5 and 6.

We want to validate the left-right disparity map with the check described in the next paragraph, a small temporal storage of w elements is used to store the right-left disparities.

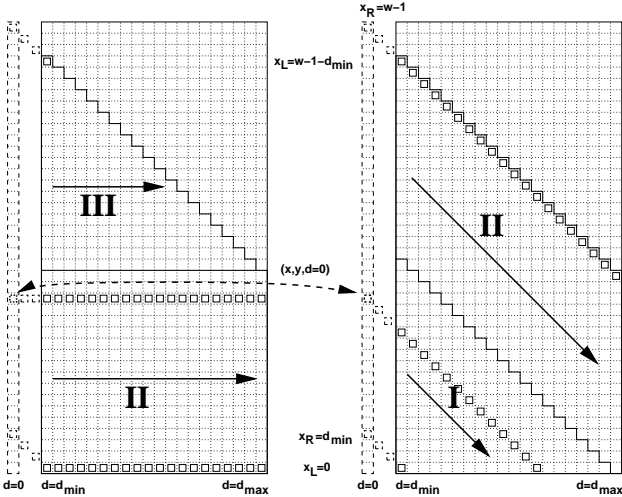


Fig. 6. Search directions in the case $d_{max} > d_{min} \geq 0$. Left figure: from left to right. Right figure: right to left. The search area has to be split in two parts only (compare to figure 5).

If d_{min} and d_{max} have the same sign, $d = 0$ is outside of the slice and only two regions have to be distinguished. The case of positive disparities is shown in figure 6.

F. Left-Right Consistency Check

A valid correspondence must match in both directions. The right-left disparity for the corresponding pixel in the right image must point back to the pixel in the left image.

$$d_{RL}(x + d_{LR}(x, y), y) = -d_{LR}(x, y)$$

Using the left-right check to detect occlusions is superior to a large number of methods, according to [3] it has the highest hit and lowest false positive rate. The authors mention a high computational cost, in our implementation it takes approximately 20%–25% of the total computing time. A much higher quality of the disparity map easily justifies this effort.

G. Sub-pixel Disparity Estimation

We use the common technique of fitting a parabola to the three SAD values around the integer disparity and taking its minimum as sub-pixel estimate (figure 7).

$$d_s(x, y) = d_i(x, y) + \frac{f(d_i - 1) - f(d_i + 1)}{2(f(d_i - 1) - 2f(d_i) + f(d_i + 1))}$$

with

$$f(d) = SAD(x, y, d) .$$

While this is correct for the sum of squared differences, in our case of absolute differences one would have to fit the absolute value function to the data, which is much more difficult. Nevertheless using the parabola still gives a better estimate for the disparity than integer values.

The interpolation is done immediately after the integer valued disparity for a pixel has been found, because then the three SAD values are available in the L1-cache.

H. Uniqueness of the Minimum

As with all correlation based stereo algorithms, in regions with repetitive or without texture, no reliable disparity estimate can be made. In these cases it is better to produce a sparse disparity map than a wrong one.

Instead of just seeking the minimum, we keep track of the three smallest SAD values. The minimum defines a threshold above which the third smallest value

must lie, otherwise the pixel is marked as not having a unique minimum (figure 7). A threshold of 5%–20% above the minimum SAD value has been a reasonable choice in our applications.

Regarding the second smallest instead of the third smallest value is not sufficient, because there are pixels whose SAD is nearly equal for two neighboring disparities. These "double minima" (figure 7(b)) are perfectly valid and occur frequently between regions that differ in their integer disparity by one.

III. MEDIAN FILTER

Median filtering is a well-known technique for removing salt-and-pepper noise from images. There are two cases where filtering the disparity map can improve the results:

- In weakly textured regions the signal to noise ratio is low and often some pixels are rejected although the disparity can correctly be estimated in the neighborhood.
- Conforming with [5] we found that isolated matches that were not rejected by our checks, almost always were false ones.

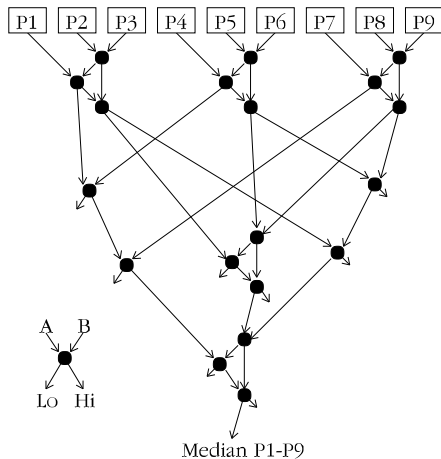


Fig. 8. Minimum sorting network to calculate the median of nine elements [13]. Every node has two inputs on its top, the smaller element is output at the bottom left, the bigger one at the bottom right.

For small kernels it is faster to calculate the median directly rather than maintaining a sorted list with insertion and deletion [2]. It is only necessary to partially sort the elements until the element in the middle position can be identified. Figure 8 shows the minimum sorting network for nine elements. Every node in the network is implemented as swapping operation so that the sorting can be done in place.

For every pixel we check if at least five disparities are given in the 3×3 neighborhood. If not, the resulting

pixel is rejected. For five to nine good pixels, we take the median of their disparities and store the result. An ANSI C implementation of the optimal median of three to nine elements is given in [2].

IV. RESULTS

All calculations for this papers were performed under Linux (kernel 2.4.6) on a dual processor 800MHz Pentium III system (Intel OR840) with 512MB RDRAM. The code was compiled with gcc 2.96, using the multi-threaded libc. Compiler flags for the optimizer were "-O6 -funroll-loops -fschedule-insns2 -mcpu=pentiumpro -march=i686". To make the results comparable to non SMP systems, only one thread was used for calculation.

Performance evaluation has been done with the image pairs listed in table 1, covering a wide range of sizes and disparity search ranges. The resulting disparity maps are comparable to those of every other implementation using the sum of absolute differences and left-right consistency checking (figure 9).

image pair	$x \times y \times d$	ms	$\frac{ns}{pix \times disp}$
QSIF	$160 \times 120 \times 32$	50	82
tsukuba	$384 \times 288 \times 20$	218	98
pentagon	$512 \times 512 \times 21$	552	100
PAL	$720 \times 576 \times 50$	1890	91
image0005 ¹	$2048 \times 1536 \times 551$	124900	72

Table 1: Execution times for sample image pairs.

Figure 10 shows the resulting disparity map after applying the 3×3 median filter from section III, which took 48ms. Many small regions that were rejected in the original disparity map have been closed with disparities from neighbor pixels and single pixels without support in the neighborhood have been discarded. The median filtered disparity map yields a more continuous surface when the points are triangulated afterwards.

Figure 11 illustrates that our algorithm can handle very large images. The correlation window has been 19×19 pixels. To cover the whole depth range, a disparity search range of approximately 550 pixels is needed. Keeping the memory consumption low is very important with this image pair, because storing the whole volume would require 6.45GB(!). By using the ring buffer of 21 slices for the disparity space volume, the memory consumption can be reduced to manageable 90.4MB. Except for some mismatches at the checkerboard patterns in the front, the map does

¹taken from the "Tea pot" test sequence at <http://www.eecs.lehigh.edu/~tboult/STEREO/DataSets.htm>

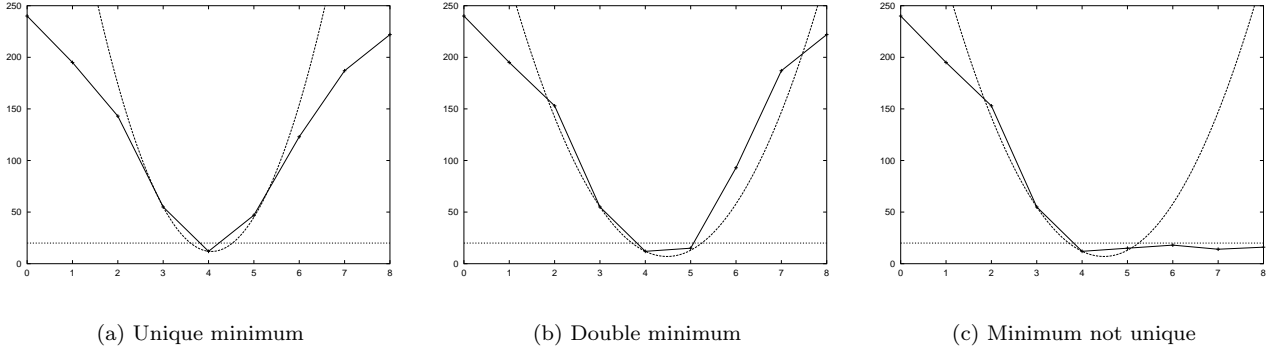


Fig. 7. SAD values, threshold and interpolation parabola plotted over disparity. One unique minimum (a) or a double minimum (b) pass the uniqueness test. If the third smallest value does not lie above the threshold (c), the pixel is marked as bad.



Fig. 9. Disparity map of the "tsukuba" image pair for a window size of 7×7 pixels. Pixels that were rejected by the left-right or uniqueness test are shown in yellow.

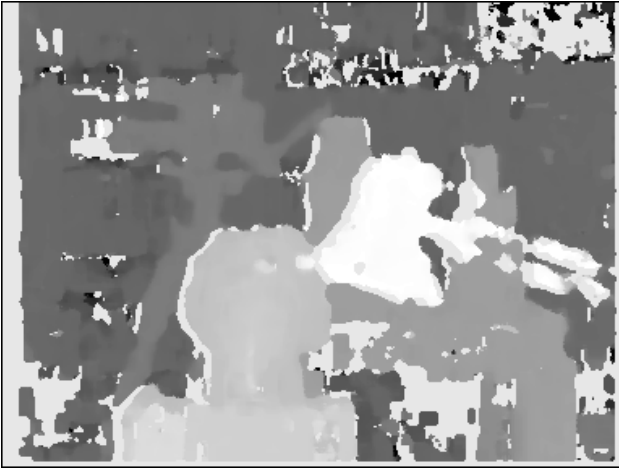


Fig. 10. Median filtered disparity map of the "tsukuba" image pair (compare to figure 9).

not contain gross errors, parts of the gray surface of the table could be matched quite densely. For this disparity map, median filtering took 836ms.

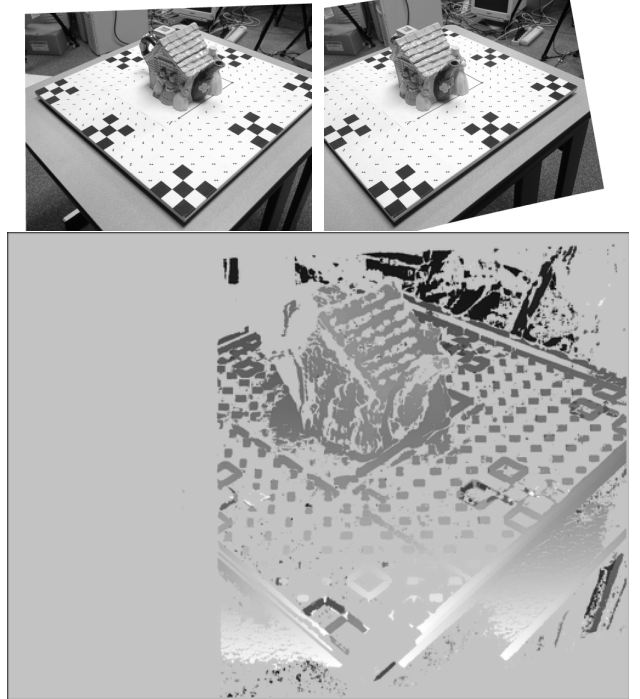


Fig. 11. Image pair "image0005" and its disparity map.

V. SIMD CONSIDERATIONS

Some concluding remarks about the potential and difficulties of accelerating the algorithm with SIMD instructions, in particular the MMX and SSE instruction sets of the Intel Pentium III and its successors [7], [8]:

- The `psadbw` instruction for the sum of absolute differences of up to eight 8bit values simplifies the SAD

calculation. However, when reading two consecutive pixels (32bit RGB) into MMX registers, some shuffling of the data has to be done to get the difference of single pixels. As an alternative `movd` can be used, but this doesn't reduce the number of memory accesses.

- Unfortunately there exist only instructions for calculating the minimum of two values that operate on 16bit or 8bit data, we would need it for 32bit.
- To avoid pollution of the caches with write-only data, non temporal stores can be used to bypass the cache when storing the disparity values.
- Parallel data access is difficult, because the sliding window sum in x and the minimum search in d are orthogonal directions. We would like to see an instruction like `pshufw` that can shuffle data among two registers, not only within one.
- Using the same memory region with different granularities is suboptimal, [8] suggests to avoid small loads after a large store and vice versa.
- Using MMX for correlation and SSE for subpixel refinement in parallel reduces register pressure and enables parallel data flow in the FPU and integer units. In addition it reduces the number of switches between floating point and MMX usage of the floating point registers significantly.

VI. CONCLUSION

We have presented an efficient implementation of correlation based stereo for color images that is capable of detecting occlusions and unreliable matches with a left-right consistency and a minimum uniqueness check. Sliding window summations and immediate reuse of intermediate results prevent redundant computations and unnecessary memory accesses. By choosing a special memory layout, the benefit of caches can be maximized.

A fast median filter removes the number of false matches even further with very little additional computation.

As long as there is a minimum of texture in the images, the algorithm gives dense disparity maps. We have used it with good results for example on images of fair-faced concrete.

The algorithm is fast and scales almost linearly with the size of the disparity space volume ($\text{width} \times \text{height} \times \text{disparity range}$). It can easily be multithreaded by dividing the images into horizontal stripes (with an overlap of win_y). Using two threads in parallel, we experienced a speedup of factor 1.6 to 1.8 depending on the image size.

REFERENCES

- [1] Luis Alvarez, Rachid Deriche, Javier Sánchez, Joachim Weickert, *Dense Disparity Map Estimation Respecting Image Discontinuities: A Pde and Scale-Space Based Approach*, technical report RR-3874, INRIA, January 2000.
- [2] Nicolas Devillard, *Fast Median Search: an ANSI C implementation*, <http://www.eso.org/~ndevilla/median/>
- [3] Geoffrey Egnal and Richard P. Wildes, *Detecting Binocular Half-Occlusions: Empirical Comparisons of Four Approaches*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2000.
- [4] Olivier Faugeras et. al., *Real time correlation-based stereo: algorithm, implementations and applications*, technical report RR-2013, INRIA, 1993.
- [5] P. Fua, *A Parallel Stereo Algorithm that Produces Dense Depth Maps and Preserves Image Features*, Machine Vision and Applications, 6(1), 1993.
- [6] A. Fusiello, V. Roberto, and E. Trucco, *Efficient stereo with multiple windowing*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 858–863, Puerto Rico, June 1997.
- [7] Intel Corporation, *IA-32 Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual*, <http://developer.intel.com/design/pentiumiii/manuals>, 2001
- [8] Intel Corporation, *Intel Architecture Software Optimization Reference Manual*, <http://developer.intel.com/design/pentiumiii/manuals>, 2001
- [9] S. S. Intille and A. F. Bobick, *Disparity-space images and large occlusion stereo*, in ECCV, pp. 179–186, 1994
- [10] R. Klette, A. Koschan, K. Schlüns, V. Rodehorst, *Surface Reconstruction based on Visual Information*, technical report 95/6, Dep. of Computer Science, The University of Western Australia, July 1995
- [11] K. Konolige, *Small Vision System: Hardware and Implementation*, Eighth International Symposium on Robotics Research, <http://www.ai.sri.com/konolige/svs/Papers>, Japan, October 1997
- [12] M.J. McDonnell, *Box-filtering techniques*, Computer Graphics and Image Processing, vol. 17, pp. 65–70, 1981.
- [13] C. Sun, *A Fast Stereo Matching Method*, Digital Image Computing: Techniques and Applications, pp. 95–100, Massey University, Auckland, New Zealand, December 1997.
- [14] John L. Smith, *Implementing Median Filters in XC4000E FPGAs*, Xcell: The Quarterly Journal for Xilinx Programmable Logic Users, Xcell 23, 1996
- [15] C. Zitnick and T. Kanade, *Cooperative Algorithm for Stereo Matching and Occlusion Detection*, technical report CMU-RI-TR-99-35, Robotics Institute, Carnegie Mellon University, October 1999.