Real-Time Stereo by using Dynamic Programming

Sven Forstmann,Yutaka Kanou
3D Incorporated
Yokohama 221-0052,JAPAN
sven, kanou@ddd.co.jp

Jun Ohya
Waseda University
Tokyo 169-0051, Japan
ohya@waseda.jp

Sven Thuering,Alfred Schmitt
Karlsruhe University
Karlsruhe 76131,Germany
thuering, aschmitt@ira.uka.de

Abstract

A Dynamic Programming (DP) based algorithm that achieves real-time, high quality stereo-matching is presented. A special coarse to fine approach as well as the MMX based assembler implementation mainly contribute to the reached computation speed. A novel vertical smoothing approach by re-integrating paths inside the DP matching allows the avoidance of noisy horizontal strokes, so that high quality stereo-matching is achieved. The current implementation capable of running at about 30 FPS on an 2.2GHz PC, which is sufficient for the utilization in real-time applications.

1. Introduction

3D-Reconstruction from Stereo-Vision has a long history and is a topic of computer-based research for longer than 20 years. Usually, this technology is targetted to machine vision [12], but is also common in 3D reconstruction [8,9] or driving aid [10]. This paper proposes an algorithm that achieves real-time, high quality stereo matching, based on Dynamic Programming (DP). Firstly, the descision for the choice of the method will be explained, by comparing DP to existing alternatives like window-based correlation and graphcut. As for the sophisticated graphcut method which is computationally demanding and therefore is not applicable in real-time stereo. One as successful implementation here emphasized work is Layered Stereo [7], for leading the ranking in [4]. Correlation on the other hand, is capable of real-time and has already proved its feasibility in various software [5,6] and hardware [2,3] applications. In case of a little vertical displacement between both input images, correlation has the advantage, of finding correct disparities, whereas DP cannot. However, unlike DP, traditional correlation has the drawback of lacking in global horizontal optimization, as well as having difficulties inside low textured areas, and furthermore, its basic computational complexity is far above DP. An examination of the resulting depthmaps reveals inaccuracies of both algorithms. As for an investigation of DP, errorous horizontal strokes often arise, and in case of correlation, the result often emerges blurry. In comparison to correlation, DP incorporates some remarkable advantages, thus encouraging the descision to choose DP for the real-time implementation presented here. DP implementations appeared at times in history. In this context, the work of Kanade [1] is mentioned here as one of the fundamental. As for real-time applications, mostly correlation was chosen to be employed. In opposition to this, DP has been very uncommon in these approaches; [10] might be one of the rare examples. Hence, the opportunity to develop improvements seemed to be promising.
The organization of this paper is as follows. Section 2 outlines the proposed method. Sections 3 contains the steps applied to increase speed. Section 4 shows quality depending improvements and Section 5 demonstrates in experimental results the effectiveness of the proposed method. Section 6 summarizes this paper.

2 Proposed Method

This paper proposes a 2-step-DP based method for real-time, high-quality stereo matching. For efficient computation, the proposed system consists of a novel coarse to fine approach in combination with MMX, to achieve real-time performance. The coarse-to-fine configuration mainly contributes to computation efficiency as in table 3 but also to quality as table 1 demonstrates. As shown in fig.2 and 3, both, the coarse and the fine version, fully integrate the two common DP steps for evaluating the DP matrix and finding the path. The coarse step evaluates a stencil array that is used to reduce the DP matrix of the fine version. For achieving horizontal and mainly vertical smoothness, many modules had to be integrated, as table 1 reveals. They form a combination of novel and traditional ways to achieve overall smoothness. To handle vertical smoothness, especially the two following ideas deserve closer attention in regard to the final result. The first reintegrates already calculated DP-paths from earlier lines weighted into DP-matrices of the current scan line.The second applies smoothing by vertically blending the color difference values, which are involved in the evaluation of the DP matrix.

2.1 Dynamic Programming

This section explains the basic steps for evaluating the DP matrix, also described in fig.5. As first step, it is necessary to obtain the elements of the matrix. The instructions for accomplishing this are described in pseudocode below, whereas i as well as j have to be counted from 0 to n-1 seperately, in order to cover all elements of matrix A. The value n is representing firstly the width of the input images but also stands for the dimension of the whole DP matrix. The connectivity between can be seen best in fig.1.

```
Minimum = Min( A[i-1,j], A[i,j-1], A[i-1,j-1] )
ColorL    = LeftImage[i,y]
ColorR    = RightImage[j,y]
A[i,j]    = Minimum + Dif(ColorR,ColorL)
```

At the total beginning, A[0,0] has to be initialized with 0. Afterwards, all others elements are evaluated in the order from the upper left to the lower right corner, as in fig.5. It is absolutely necessary just to include already initialized matrix elements for performing the Min function inside the pseudocode above. Once the matrix has been filled, a path of minimal cost can be calculated by tracing back through the DP matrix beginning from A[n-1,n-1], and ending in A[0,0]. This step can be seen in fig.6 as well as in the following pseudocode below. The horizontal difference between path- and diagonal-elements will return the left disparity map, and the vertical difference will return the right disparity map. The chosen path always follows the smallest values inside A, wereas three directions are allowed to be chosen in each position: up, upleft and left. In case of an unclear descision, upleft will be chosen as default.

```
DisparityMapL [i,y] = j-i
DisparityMapR [j,y] = i-j
Up          = A[i-1,j]
Left        = A[i,j-1]
UpLeft      = A[i-1,j-1]
Minimum   = Min( Up,Left,UpLeft )
case Minimum of
    UpLeft    : i=i-1;j=j-1
    Left      : j=j-1
    Up        : i=i-1
end
```

As for the code above, it will be initialized by setting i=0 and j=0, then repeated until i==0 and j==0. The variables i and j are representing the current path-position inside the matrix, as well as the horizontal position inside the disparity map. The difference of choosing i or j for addressing the depth-map, is equal to choosing the left or right disparity map. The variable y defines the vertical position inside the displarity-map and remains constant during the upper loop.

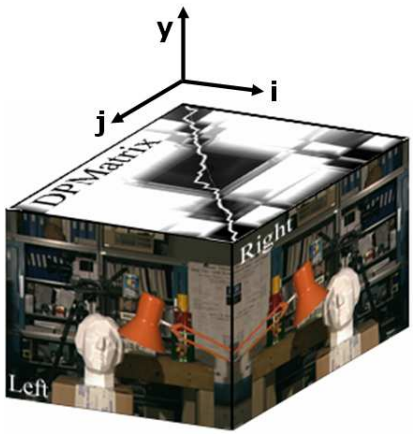

Figure 1: 3D View of the DP matching view, with axes.

3 Speed Optimizations

The most relevant optimizations of this paper are of course focused on speed, because the algorithm aims on real-time. The contribution of each of the following steps to the final result is referenced in table 3.

3.1 Reduction of the Matrix Size

The first applied optimization was the reduction of the the DP matrix to absolutely necessary portions, as fig.7 shows. The amount of these portions can be bound by the maximal expected disparity. The average speedup factor for this optimization is estimated to be approximately image width divided by the maximal allowed disparity range. It is recommended to append all used parts of the matrix to obtain the smallest memory block possible and improve cache-performance.

3.2 Assembly Language

The main parts of the algorithm were ported to assembly language by integrating the multimedia extension commands, MMX, available on any recent CPU. By exploiting these extensions, only 4 assembly commands are necessary for calculating the color difference of 2 pixel (i.e. 6 color values). It is recommended to consult [14], page 78, for further information. Finally, assembler brought a speedup of approximately 30% in comparison to C. In the past, MMX has already been used in real-time stereo, but there, it was applied to correlation and used for parallel adding instead of evaluating differences [17].

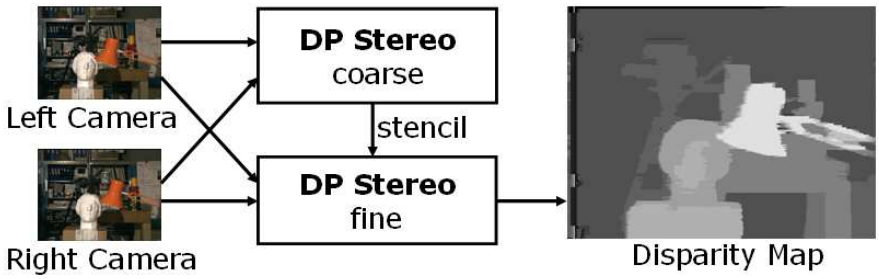

Figure 2: This graph shows the overview above the general coarse to fine DP process

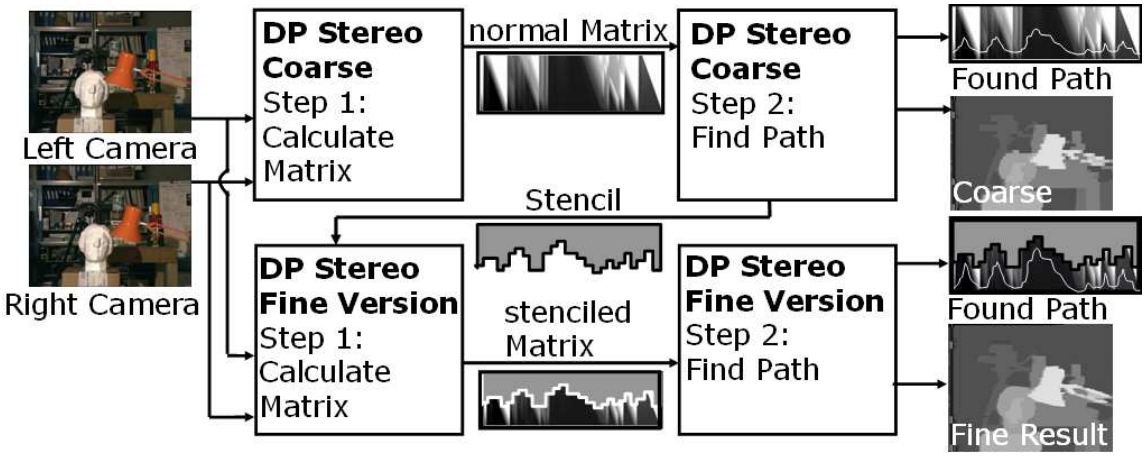

Figure 3: Detailed overview of the coarse to fine DP approach, with both DP steps and the DP matrices

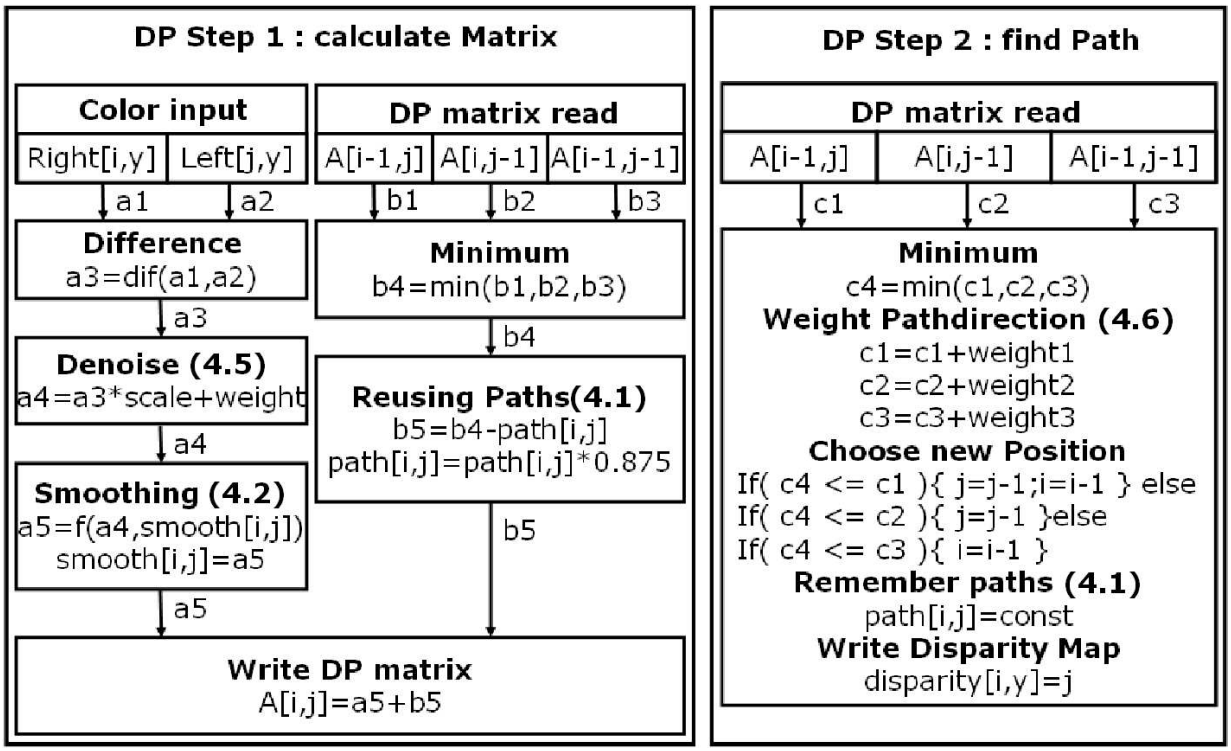

Figure 4: This figure shows the two DP steps with all necessary formulas

		Right				
	S	T	E	R	E	O
S	0	1	2	3	4	5
T	1	0	1	2	2+1	...
R	2	1	1	1+0	1+1	...
E	3	2	1	1+1	1+0	...
O	4	3	2	1+1	1+1	
E	5	4	2+0	2+1	...	

Left

Figure 5: This figure is a simple example for the construction of a basic DP matrix. Two strings shall be matched here, STEREO and STREOE, instead of color values. The formula for the evaluation is the basic one of section 2.1: A[i,j]=min(A[i-1,j-1],A[i-1,j],A[i,j-1])+dif(left[i],right[j]). To simplify the difference function, the return values were reduced to 0 and 1. Dif returns 0 for equal inputs (e.g. dif(T,T)) and 1 for unequal inputs (e.g. dif(S,T)). Before the evaluation is started, A[0,0] is initialized with 0.

		Right				
	S	T	E	R	E	O
S	**0**	**1**	**2**	**3**	**4**	**5**
T	**1**	**0**	**1**	**2**	**3**	**4**
R	**2**	**1**	**1**	**1**	**2**	**3**
E	**3**	**2**	**1**	**2**	**1**	**2**
O	**4**	**3**	**2**	**2**	**2**	**1**
E	**5**	**4**	**2**	**3**	**2**	**2**

Left

Figure 6: The path inside the DP matrix is evaluated by starting at the lower right corner of the matrix and tracing back along the smallest numbers to the upper left corner.

3.3 Coarse to Fine

In this algorithm, the preview for coarse to fine is calculated by line skipping. At first, every n'th horizontal line is calculated to find bounding space for possible disparities in between. The DP matrices of these lines between are now reduced to this space, which is spanned between a certain upper and lower boundary. The upper boundary is calculated as following:

Function to evaluate alignments
g(x,m) = x - (x mod m)

Pseudocode to evaluate the upper bound
```
up=-infinity
for(a=0;a!=16;a+=1)
 for(b=0;b!=8;b+=8)
  up = max( up , disparitymap[ g(x,16)-a, g(y,8)-b ] )
upperbound[x,y]=up;
```

In the pseudocode above, g represents the function which is necessary to snap the x and y coordinate to a grid with cellsize 16x8. At these gridpoints, bounding positions are calculated in the coarse version so that they are able to be applied as stencil to the fine version. These 16x8 cells can

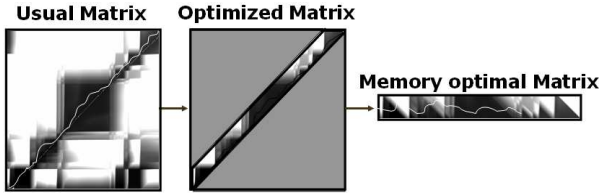

Figure 7: This figure shows an effective way to speed up the matching process. The left matrix represents the common DP matrix. In the middle, just absolutely necessary space is calculated. As this figure reveals, there is only a small portion required to be calculated around the diagonal line. The right matrix represents the final memory and cache optimal version. The values inside these matrices have similarities to the values calculated in fig.5 and 6 (brightness is proportional to values) The white line inside the matrices points out the found path.

be imagined best as 3D bounding boxes, whereas the depth is determined by expected disparity space. As for the variables, x and y are the coordinates inside the disparity map, a and b are temporary counting variables to calculate the bounding box and the variable "up" temporary holds the upper bound, until it is stored inside the upperbound table. For evaluating the lower bound, the pseudocode is identical, except the replacements of -infinity by +infinity and max by min which have to be made. By this novel coarse to fine optimization, speed could have been doubled in experimental results as table 3 proves. The idea, to calculate only half of the horizontal resolution in the preview, brought even a better speedup, but was dropped because lots of details were missing afterwards. Coarse to fine is also used in other papers, as in [5,16] eg., but there it was applied on correlation and the image was scaled in width and height instead of scaling the height only.

3.4 Compiler Optimizations

By all improvement, also standard compiler optimizations shouldn't be forgotten; for the basic implementation, the o2 compiler switch already brought twice the speed. Newer compiler even support o3. Also code and data alignments to 16,32, etc. might be helpful. For variables in general, there can also be a different speed between local-, class- or global program variables.

4 Quality Improvements

Because common DP can not optimize vertically, which causes the typical horizontal strokes, the next optimizations will focus on the quality aspect. An overview of the effectivity can be seen in table 1. The interconnection of the

following modules can be seen in fig.4. This figure also contains all used formulas and references to the sections below.

4.1 Reusing Calculated Paths

This optimization provides vertical smoothness by trying to keep the current path close to the path of the former scan line. This is done memorizing the current path inside a seperate matrix, called path[i,j] inside fig.4. Afterwards, this seperate matrix is subtracted from the general DP matrix of the following scan line. This causes the path of the following scan line to be sightly attracted by the path of the current scan line. To achieve a wider smoothness, the separate matrix is multiplied by a factor less than 1 each line, to maintain also the paths of previous scan lines. In fig.4, these steps are called “reusing paths” (left side) and “remember paths” (right side).

4.2 Smoothing of Difference Values

The next optimization to achieve vertical smoothness, is the weighted vertial smoothing of color differences. On the left side in fig.4, this procedure is referenced as smoothing. The idea is, to blend current color difference values with ones of the former scan line via matrix smooth[i,j]. The matrix smooth is hereby used for the temporary storage of the former difference values. In fig.4, f(a,b) evaluates the smoothed value, bound by a and b, whereas a stands for the current difference value and b for the former scan lines difference. Whether the tendency is closer to a or b depends on the difference between a and b. In case of a small difference, the tendency would be to b and a in case of a big difference. Inside the final implementation, the function f roughly approximates the weighting by providing 3 different weight steps. The weighting is extremely important here in order to preserve sharp object boundaries.

4.3 Subpixel Accuracy

The calculation on subpixel can be advantageous if applied to the presented real-time implementation, as fig.10 demonstrates. However, because of the speed-loss, it has not been integrated as preprocessing step. The easiest way to apply this is, to stretch the input images x-times and apply blurring across x pixel afterwards. The resulting depthmap now consists of x-times as many depth steps as the original. Horizontal stretching is most effective here, because vertical stretching has no effect on the amount of depth-steps.

4.4 Two Pass Evaluation

In this case, the DP has to be applied twice. At first, the horizontal lines are matched from left to right, and afterwards right to left. The advantage is, that occlusions are handled more accurately. It has not been implemented for the final version, because this version is targetted for speed. However, there is much potential for future usage.

4.5 Noise Reduction

Often, images tend to be a little noisy after being captured from camera. To reduce this noise, either filter can be applied in advance, but it’s also possible to ”weight” the color difference. This is done by adding a weight to the calculated color difference between the pixels. In fig.4, this module is refereced as ”Denoise”. The advantage is a fast noise removal without loosing accuracy inside the depthmap. The optimal weight is image dependent. This is the most important quality improvement as table 1 reveals.

4.6 Weights for Pathdirections

Another image dependent option for improvement is, to influence the possible path-directions when tracing back thought the DP matrix. This can be done easily, by assigning three different weights to the three available path-directions. In fig.4, the weights are represented by weight1, 2 and 3.

5 Results

For calculating the qualitative results of table 1, groundtruth images provided by [15] were taken as reference. Compared to other methods that pursue real-time, the achieved results, shown in fig.9 and fig.8 as well as the ones submitted to [15], show promising results for REAL SCENES (see also table 2). The method discussed here is one of the rare DP based methods that runs at real-time speed. As demonstrated in table 4, we can prove to present one of the fastest methods in stereo vision. The middlebury stereo database [15] includes two other algorithms that focus on real-time/fast stereo matching, as Hirschmueller’s method [18] and Sun’s [16]. Hirschmueller utilized a correlation based method for stereo matching, but he didn’t provide any benchmark result about the algorithm’s computation speed inside [18]. As for a comparison to his other paper [20], where speeds have been delivered, the presented implemetation would be approx. 60% faster, by taking the test-systems speed difference in terms of bogomips into account. The benchmark in bogomips is more accurate than the comparison of megahertz, also its results [21] are equal to other CPU benchmarks [23] in regard to the speed factor between different CPUs. Sun’s paper on fast stereo matching is more informative. Their actual paper can claim real-time performance, but it is currently ranked last in [15]. The presented method demonstrates the capability of achieving high quality 3D reconstructions as well as the real-time capability.

Table 1: This table shows the efficientcy of the quality implementations; if one optimization was disregarded, the configuration was adjusted to the optimal result. The chosen bad pixel threshold is 0, so the percentages are higher as in [4] and [15]. In brackets (left side) are references to the sections in this paper.

Optimizations used	Measured Error
All optimizations (normal)	11.44 %
no noise reduction (4.5)	19.69 % (+8.25 %)
no path reusing (4.1)	14.55 % (+3.11 %)
no weighted paths (4.6)	13.56 % (+2.44 %)
no dif smoothing (4.2)	12.54 % (+1.10 %)
no coarse to fine (3.3)	12.43 % (+0.99 %)

Table 2: This table shows the result table of [4],page 31, fixed settings, extended by the algorithm of this paper(RT). These results are also included in [15]

Image	SSD	DP	RT	GC
Tsukuba	5.23	4.12	2.85	1.94
Sawtooth	2.21	4.84	6.25	1.30
Venus	3.74	10.10	6.42	1.79
Map	0.66	3.33	6.45	0.31

It can even reach graphcut level in comparison with Max flow [19] applied to the Tsukuba image. In [15], Max flow is ranked at the 17th place for quality, while the presented algorithm is ranked at place 14. The next better Graphcut can be found at place 12. Tsukuba might be also the most representative image in the survey of [4] and [15]. It contains only one real scene, while all the others were comuter-generated images. The standard Tsukuba images, obtained from [15], also used for building table 4,could have been matched in only 18 Milliseconds (55 fps) on the test system. This is sufficient for real-time usage, as camera systems have about 30 fps. The speed of the algorithm is also sufficient for pre- and postprocessing. The whole processing consisted of correcting the camera distortion, smoothing, stereo matching and vizualizing the depth-map.

Table 3: This table shows an overview of the applied speed optimizations and their effectiveness

Speed	Optimizations applied
8 fps	Nothing
18 fps	Compiler Optimizations
25 fps	Assember with MMX
42 fps	8 Step Preview
55 fps	8 Step Preview and MMX

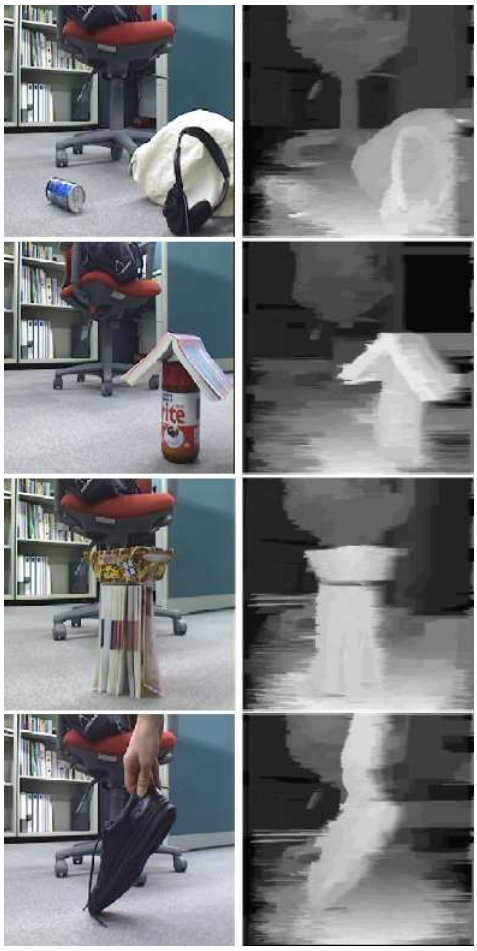

Figure 8: These images show example scenes, generated from a live-video stream. In this case, the version with dynamic parameter setting (DP parameters were adjustable on the fly) and optional backgound extraction was used, which has been about 50% slower as the reference version with constants. The resolution was 256x256 pixels with 27 disparity levels. The speed was about 15 fps for the whole system, starting from capturing over matching and finally rendering the depth-map as textured mesh.

Table 4: This table contains the benchmarks, taken on an AMD AthlonXP 2800+ CPU (4456 bogomips). The results from other researcher's were scaled according to the relation of bogomips for being comparable, according to data of [21], however, Yang03 and Parts97 weren't able to be scaled because special hardware was used. The results are shown in frames per second, depending on resolution and disparity. As for the presented algorithm, the faster version with constant parameters was employed. The number in brackets, which is sometimes written behind the fps, represents the originally used disparity in case of a difference to the columns one. As for Hirschmueller, two versions were provided, whereas v1 has been the faster and v2 the more accurate one.

Method	Resolution	16 disp.levels	20 disp.levels	32 disp.levels	50 disp.levels	100 disp.levels
this paper	256x256	93,8 fps	87.5 fps	69.0 fps	53.1 fps	30.4 fps
	320x240	77.0 fps	71.8 fps	58.5 fps	46.0 fps	26.7 fps
	384x288	55.0 fps	50.0 fps	40.5 fps	33.1 fps	18.9 fps
	240x700	37,9 fps	35.0 fps	28.2 fps	21.2 fps	11.8 fps
	512x512	25.0 fps	21.6 fps	17.7 fps	14.7 fps	8.28 fps
	640x480	20.7 fps	18.4 fps	15.2 fps	12.3 fps	7.23 fps
	1024x1024	6.14 fps	5.62 fps	4.71 fps	3.45 fps	2.20 fps
Yang03 [5]	256x256		61.7 fps		24.8 fps	12.4 fps
	512x512		24.5 fps		9.40 fps	4.80 fps
Sun01 [16]	256x256			15.5 fps (26)		
	512x512		3.56 fps		3.11 fps (44)	
	1000x1000				0.66 fps (54)	
Hirschm02 [20]	320x240 (v1)			35.0 fps		
	320x240 (v2)			23.5 fps		
Depthfinder00 [10]	240x700	2.81 fps				
VIDET00 [17]	320x240	72.49 fps		45.6 fps		
Triclops(tm) [22]	320x240				26.0 fps	
	640x480					4.5 fps
Parts97 [3]	320x240			42.0 fps (24)		

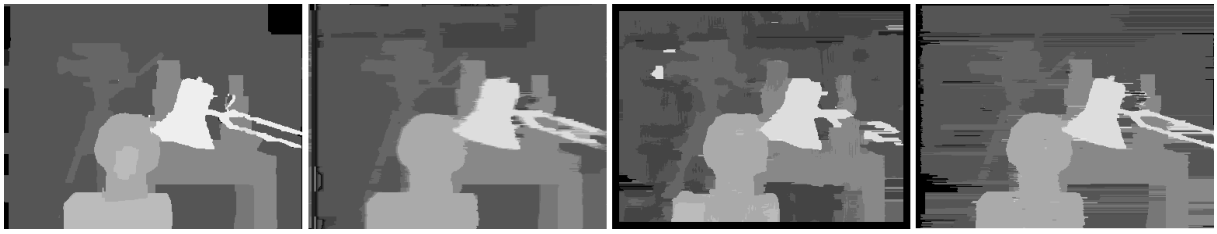

Figure 9: Here the different results from the most common methods applied to the Tsukuba image pair, shown from left to right as follows: Graphcut [13], the presented method, Correlation based stereo [6] and Dynamic Programming [4].

Figure 10: These images demonstrate the effectiveness of subpixel accuracy. The left side shows the input image (640x500), the middle the disparity map from direct evaluation and the right side the result of subpixel accuracy. For subpixel accuracy, the input image had to be streched in advance horizontally 4 times to a resolution of 2560x500. As can be seen, the depthmap from the stretched image pair has a much higher depth resolution by containing more details too.

6 Summary

The presented algorithm is one of the few DP approaches, targeted for real-time, because often correlation is used [2,3,5,6,8,9]. The achieved results showed qualitatively good disparity maps for real scenes, also by providing high performance, which is very important for real-time application usage. The achieved speed was sufficient, to match, filter and display 2 video streams on an AMD XP 2800+ PC in real-time; furthermore, generation of textured 3D-geometry in real-time was also possible. The motivation for this paper was to make stereo vision technologies more practical; i.e. we are interested in applying stereo vision to practical systems such as commercial products. As described earlier, main parts of the presented method are a combination of existing processing modules, but we propose new modules also. As a matter of fact, we are confident about the experimental results that show better performance than the other relevant works.

References

[1] Yuichi Ohta, Takeo Kanade Stereo by Intra- and Inter-Scanline Search Using Dynamic Programming *IEEE Transactions on Pattern Analysis and Machine Intelligence,1985*

[2] Takeo Kanade,Hiroshi Kano,Shigeru Kimura,Atsushi Yoshida,Kazuo Oda Development of a Video-Rate Stereo Machine *Proc. of International Robotics and Systems Conference (IROS '95)*

[3] John Woodfill,Briam Von Herzen Real-Time Stereo Vision on the PARTS Reconfigurable Computer *Originally published in IEEE Symposium on FPGAs for Custom Computing Machines, IEEE Symposium on FPGAs for Custom Computing Machines,1997*

[4] Daniel Scharstein,Richard Szeliki A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence *In Proceedings of the IEEE Workshop on Stereo and Multi-Baseline Vision, Kauai, HI, Dec. 2001*

[5] Ruigang Yang and Marc Pollefeys Multi-Resolution Real-Time Stereo on Commodity Graphics Hardware *Multi-Resolution Real-Time Stereo on Commodity Graphics Hardware, page 211-217, CVPR 2003*

[6] Ruigang Yang and Marc Pollefeys Real-time Correlation Based Stereo Vision *CVPR 2001 Stereo Workshop / IJCV 2002*

[7] Michael H. Lin, Carlo Tomasi Surfaces with Occlusions from Layered Stereo *Stanford University, 2002.,CVPR 2003*

[8] Masatoshi, Takeo Kanade A Multiple Baseline Stereo *IEEE Transactions on Pattern Analysis and Machine Intelligence 1993*

[9] P.J. Narayanan, Peter W. Rander, Takeo Kanade Constructing Virtual Worlds Using Dense Stereo *Proc. ICCV, pp. 3-10, 1998.*

[10] Yi Lu Murphey, Jie Chen, Jacob Crossman, Janxin Zhang, Paul Richardson, Larry Sieh Depthfinder, A Real-time Depth Detection System for Aided Driving *Depthfinder, A Real-time Depth Detection System for Aided Driving IVS 2000*

[11] A.F. Bobick and S.S. Intille Large occlusion stereo *In Vismod, 1999*

[12] Steven B. Goldberg, Mark W.Maimone, Larry Matthies Stereo Vision and Rover Navigation Software for Planetary Exploration *IEEE Conference Proceedings,March 2002,Big Sky,Montana,USA*

[13] Vladimir Kolmogorov and Ramin Zabih Computing Visual Correspondence with Occlusions using Graph Cuts *In International Conference on Computer Vision (ICCV), July 2001*

[14] Intel Corporation Intel Architecture Optimization Manual *Chapter 4.6.7,page 78*

[15] Daniel Scharstein, Richard Szeliski Middlebury Stereo Vision Research Page *http://bj.middlebury.edu/ schar/stereo/web/results.php*

[16] C. Sun Fast stereo matching using rectangular subregioning and 3D maximum-surface techniques *CVPR 2001 Stereo Workshop / IJCV 2002*

[17] Luigi Di Stefano, Stefano Mattoccia Fast Stereo Matching for the VIDET System using a General Purpose Processor with Multimedia *2000*

[18] H. Hirschmueller Improvements in Real-Time Correlation-Based Stereo *Vision. CVPR 2001 Stereo Workshop / IJCV 2002*

[19] S. Roy and I. J. Cox A maximum-flow formulation of the N-camera stereo correspondence problem *ICCV 1998*

[20] H. Hirschmller, P. R. Innocent Real-Time Correlation-Based Stereo Vision with Reduced Border Errors *IJCV 20002*

[21] Steve.org Details about the Performance of several machines, date 04/2004 *http://www.steve.org.uk/Software/BogoMIPS/all.php?detail=1*

[22] PointGrey Research Triclops(tm) Software Development Kit, date 04/2004 *http://www.ptgrey.com/*

[23] Tom's Hardwareguide Benchmark Marathon of 65 CPUs, date 04/2004 *http://www20.tomshardware.com/cpu/20030217/cpu_charts-27.html*