Disparity Estimation in Real-Time 3D Acquisition and Reproduction System

Artem Ignatov, Victor Bucha, Michael Rychagov
Samsung Research Center,
1-st Brestskaya St. 29, Moscow, 125047, Russia
{a.ignatov, v.bucha, michael.rychagov}@samsung.com

Abstract

The paper concerns the method of disparity estimation and refinement based on iterative filtration of raw disparity estimate. Raw disparity estimate is obtained by conventional stereo-matching methods. In suggested algorithm, 6-8 iterations are sufficient to compute high-quality disparity map, suitable for virtual views rendering, which is essential step of any 3D reproduction system. The method is suitable for highly-parallel processing on modern GPU. In contrast to previous methods of stereo matching implemented on GPU, the proposed approach provides correct disparity computation for "problem areas", i.e. large uniform areas, occlusion areas and areas with repetitive patterns. Presented method was implemented in a real-time 3D acquisition and reproduction system. For GPU programming, an extension of C programming language provided by NVIDIA Compute Unified Device Architecture (CUDA) was used. Experimental results confirm the usefulness and robustness of the method.

Keywords: *disparity estimation, real-time processing, GPU computing, 3D reproduction, video processing.*

1. INTRODUCTION

Nowadays, humanity enters new era of digital television, i.e. the 3D television. The attention of academic and business environment is absorbed by the great opportunities for new application possibilities, which advanced techniques may provide. The 3D TV may bring an effect of real presence of participants in real-time video conferences. Also, the reality of computer games can be significantly improved by playing in 3D. Any synthetic video rendered by computer graphics applications can be immediately viewed on modern 3D auto-stereoscopic displays. However, for the real scenes, captured by stereo-camera or multi-camera setups there are still a lot of tasks to be solved. For example, for video-conferencing, all calculations should be computed in real-time. The necessary computations include cameras calibration, disparity estimation, and several views generation according to 3D reproduction device requirements. Cameras calibration could be performed off-line, if the cameras geometry is fixed. However, the disparity estimation of participants, and simultaneous view rendering should be performed on acceptable frame-rate, convenient to the users.

To cope with high volumes of calculations, some researches have tried to realize stereo-matching algorithms on modern GPU of video-boards. Recent works aiming stereo analysis proposed to exploit very efficient parallel Single Instruction Multiple Data (SIMD) architecture of modern GPU. Real-time implementation of computationally intensive methods of disparity estimation becomes possible thanks to SIMD architecture. J. Mairal et al. [1] reported that he achieved nearly video frame rate stereo reconstruction. The approach is from family of variational methods based on deformable models. Proposed method computes dense stereo from 3 cameras and entirely implemented on a GPU.

R. Yang et al. in [2], tried to force OpenGL toward conventional stereo computation, based on SAD window calculation with WTA optimization. The authors applied several OpenGL features for speeding-up of the algorithm, namely mip-map mechanism of texture generation for aggregation raw costs of different scale; p-buffers, which are the user-allocated off-screen buffers for fragment output. Unlike the frame buffers, they can be used directly as a texture, thereby eliminating excessive CPU memory read-ins.

One more example of using GPU for stereo analysis is the work of A. Brunton et all [3], which presents a novel implementation of Bayesian belief propagation for graphics processing units found in most modern desktop and notebook computers.

Most recent results of disparity estimation are obtained using NVIDIA CUDA technology. S. Grauer-Gray et all [4] described an efficient CUDA-based GPU implementation of the belief propagation algorithm that can be used to speed up stereo image processing and motion tracking calculations without loss of accuracy. Achieved acceleration in comparing with CPU realization is reported as a factor of five.

J. Gibson et al. [5] described how to accelerate the calculation of depth from stereo images by using a GPU. The CUDA was employed in novel ways to compute disparity using BT (Birchfield–Tomasi) cost matching. The challenges of mapping a sequential algorithm to a massively parallel thread environment and performance optimization techniques were considered.

2. REAL-TIME 3D ACQUISITION AND REPRODUCTION SYSTEM

We developed the system for acquisition of the dynamic 3D scene and simultaneous reproduction on auto-stereoscopic display (Figure 1). To manage the high demands on computation, the system uses parallel computation architecture of modern GPU and proposed method on disparity estimation was highly optimized for such architecture.

A system for three-dimensional video acquisition and reproduction includes a stereo content acquisition stage, disparity

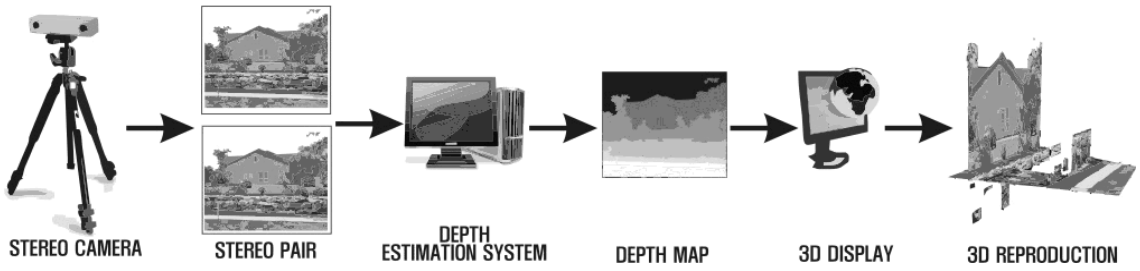


Figure 1: 3D acquisition and reproduction system

estimation stage, virtual view synthesis stage and 3D reproduction stage. The acquisition stage includes stereo cameras to acquire multiple video streams of dynamic scene. The disparity estimation stage includes computation units for real-time disparity estimation using obtained video streams. The view synthesis stage includes computation units for real-time virtual views synthesis on the basis of the computed disparity. The 3D reproduction stage includes computation units and 3D display for volumetric reproduction based on several virtual views generated on the previous stage.

The present system provides high-quality real-time 3D video acquisition and reproduction on 3D display.

Developed 3D acquisition and reproduction system includes stereo-camera Bumblebee 2 by Point Grey Research, modern PC with video-board NVIDIA GeForce 8800GTX, which has 128 stream processors, and auto-stereoscopic 3D monitor by Phillips.

For effective control, playback and record of 3D video, the corresponding software was developed. The acquisition part was written on the basis of Point Grey Research SDK. The stereo-matching, i.e. disparity/depth estimation was implemented using CUDA technology. This is the software/hardware architecture, which grants access to GPU stream processors for development of any tasks which could be effectively parallelized. Stereo-matching as image and video processing technique is the exemplar of application, which is well suited for the parallel-computation architecture. Since every pixel of image or video frame could be processed independently of other pixels. One of the main benefits of CUDA is a support of programmers-friendly environment for easy application development.

To decrease excessive memory reads/writes the CUDA application result was mapped to OpenGL Pixel Buffer Object (PBO) with simultaneous rendering on 3D monitor.

3. PROPOSED METHOD FOR DISPARITY ESTIMATION AND REFINEMENT

The dense disparity estimation problem consists of finding a unique mapping between the points belonging to the two images of the same scene (stereo pair). This is an ill-posed problem especially for textureless and occluded image areas. The mapping between corresponding points is called a *disparity vector*. In case of rectified geometry, the vector is scalar, and it is called a *disparity*. Thus, a *depth* is function of disparity with invert proportional dependence.

According to recent taxonomy [6], disparity estimation (stereo matching) algorithms generally perform the following four steps:

1. matching cost computation;
2. cost (support) aggregation;
3. disparity computation / optimization;
4. disparity refinement.

According to taxonomy, proposed method of disparity estimation concerns to the methods of disparity refinement. And could be applied for raw disparity estimate, obtained by disparity computation with matching cost calculation.

Proposed method relies on an idea of convergence from rough estimate toward the consistent disparity map through subsequent iterations of the disparity filter. On each iteration, the current disparity estimate is refined by filtration with accordance to images from stereo-pair. The reference image is defined as a color image from a stereo-pair, for which the disparity is estimated. And, the matched image is defined as other color image from the stereo-pair.

The disparity filter applies weighted average of neighboring pixels to current pixel of disparity map. Pixels, which are participated in filtration, are defined as *disparity reference pixels*. Corresponding pixels of color image are defined as *color reference pixels*. And pixels which are mapped by disparity values of color reference pixels are defined as *target pixels*.

In previous approaches the filter weights were reflected by proximity and similarity measures [7-8]. Proximity measure assigns weight of reference pixel, based on distance between current and reference pixels in spatial domain. While similarity measure assigns weight of reference pixel based on similarity between pixels of color image corresponding to reference and current disparity pixels.

During our development we concluded that proximity measure is less important, than similarity one. Since the filter usually operates in small local area of the image, the spatial measure constraint could be defined implicitly. And more attention should be paid on color similarity evaluation.

We propose the following methods for similarity computation.

1. Similarity computation with two pixel area comparison.
2. Similarity computation with one pixel area comparison.
3. Similarity computation with two single pixels comparison.

First two methods of similarity computation are concern to pixel area comparison rather then comparison of single pixels. This is done to strengthen the pixel similarity criterion. We studied that similarity computation based on pixel areas comparison gives more visually pleasant results, rather than individual pixels comparison.

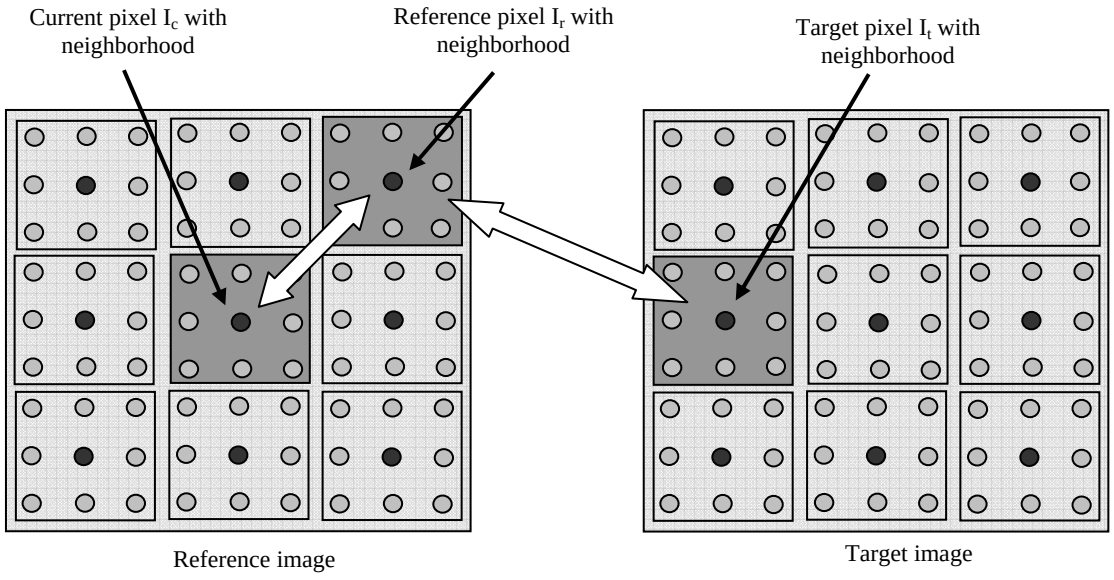


Figure 2: Pixels similarity measure computation**:** a) two comparisons of pixel neighborhoods, b) one comparison of pixel neighborhoods

In the first method of similarity computation, the weight is twofold and reflects the degree of similarity of current pixel with reference ones and target ones. First comparison is done between current pixel neighborhood with reference pixel neighborhood. The second one is carried out between reference pixel neighborhood with target pixel neighborhood (Exemplars of comparisons are shown in Figure 2 a) by wide white arrows). In this case the weights of disparity filter are computed as follows

$$w_r = e^{\frac{-C_1(x_r,y_r)}{\sigma_r} - \frac{C_1(x_t,y_t)}{\sigma_t}}, \qquad (1)$$

where C_1 () stands for a function used for pixel neighborhood comparison, σ_r and σ_t are the parameters of filter strength adjustment.

The second method of similarity computation is based on single comparison of pixel neighborhood areas of current and target pixels (Exemplar of comparison is shown in Figure 2 b) by wide white arrow). This type of similarity computation assumes that the current pixel is more or like similar with reference one due to their proximity to each other. And penalty is given only when the reference pixel is mapped to target pixel which is not similar with current pixel. In this case the weights of disparity filter are computed as follows

$$w_r = e^{\frac{-C_1(x_t,y_t)}{\sigma_t^2}},$$

where $C_1()$ stands for a function used for pixel neighborhood comparison. It is defined as

$$C_1(x_r, y_r) = \frac{1}{N \cdot M} \times \times \sqrt{\sum_{T\in\{R,G,B\}} \sum_{j=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{i=-\lfloor M/2 \rfloor}^{\lfloor M/2 \rfloor} \left(I_T(x_{c+i}, y_{c+j}) - I_T(x_{r+i}, y_{r+j}) \right)^2}$$

where $I_T(x_c, y_c)$ stands for current pixel intensity with coordinates x_c and y_c for color channel T, $I_T(x_r, y_r)$ denotes the reference pixel intensity with coordinates x_r and y_r for color channel T, N and M – are pixel area dimensions.

The third method of similarity computation is derived from the first one, setting pixel area dimensions to one. The difference from the first method is reflected in following equation for disparity filter weight computation

$$w_r = e^{\frac{-C_2(x_r,y_r)}{\sigma_r} - \frac{C_2(x_t,y_t)}{\sigma_t}}. \quad (2)$$

In comparison with Eq. 1, Eq. 2 uses different function for pixel comparison, which is defined as

$$C_2(x_r, y_r) = \sqrt{\sum_{T\in\{R,G,B\}} \left(I_T(x_{c+i}, y_{c+j}) - I_T(x_{r+i}, y_{r+j}) \right)^2},$$

where $I_T(x_c, y_c)$ stands for current pixel intensity with coordinates x_c and y_c for color channel T, $I_T(x_r, y_r)$ denotes the reference pixel intensity with coordinates x_r and y_r for color channel T.

According to selected method for similarity computation between pixels in filter window the disparity map at k-th iteration is given as

$$d_k(x_c, y_c) = \frac{1}{Norm} \cdot \sum_{s=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \sum_{p=-\lfloor L/2 \rfloor}^{\lfloor L/2 \rfloor} w_r \cdot d_{k-1}(x_r, y_r),$$

where $d_k(x_c, y_c)$ stands for the disparity map at k-th iteration for current pixel with coordinates (x_c, y_c), $d_{k-1}(x_r, y_r)$ denotes the disparity map at (k-1)-th iteration for reference pixel with coordinates $(x_r = x_{c+p}, y_r = y_{c+s})$, w_r denotes the weight of reference pixel, normalization factor is computed as

$$Norm = \sum_{s=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \sum_{p=-\lfloor L/2 \rfloor}^{\lfloor L/2 \rfloor} w_r \cdot$$

4. GPU IMPLEMENTATION OF PROPOSED METHOD

Proposed method was implemented for running on GPU. The following modifications of the algorithm were considered:

1. Adaptation of filter strength according to iteration;
2. Filter kernel size estimation according to iteration;
3. Separable processing of image rows and columns;
4. Histogram-based implementation of post-processing median filter.

Figure 3 represents algorithm flow-chart. According to flow-chart, the first step of the algorithm is to compute a raw disparity estimate. This could be done by utilization of conventional stereo-matching methods based on window correlation computation. We used SAD metric in 5x5 window. The exemplar of raw disparity estimate is presented in Figure 7 a). It is worth to mention high level of noise presented in raw disparity maps. Such noise will cause eye-fatigue when viewing this content on? auto-stereoscopic displays.

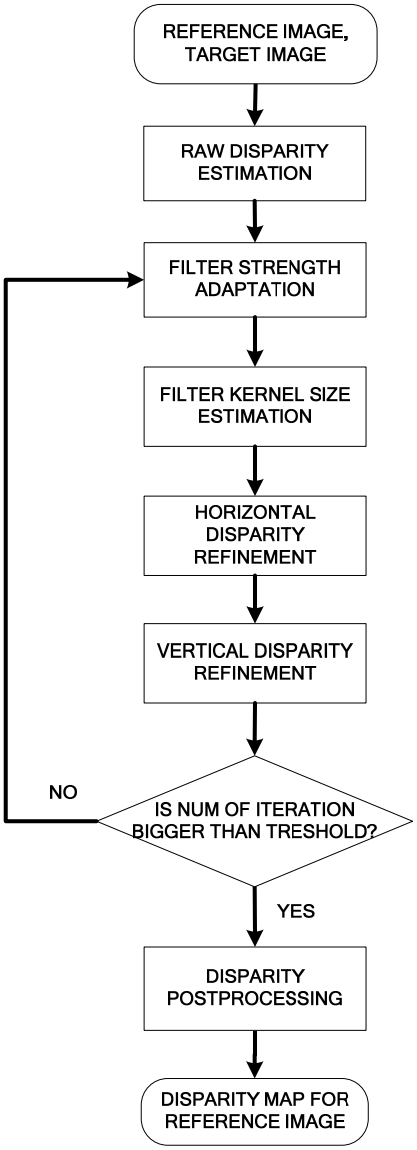


Figure 3: Disparity estimation algorithm flow-chart

After computation of raw disparity, the algorithm proceeds to series of iteration of disparity filter. On each iteration, filter strength (σ_r and σ_t are the parameters) is adjusted according to following formula

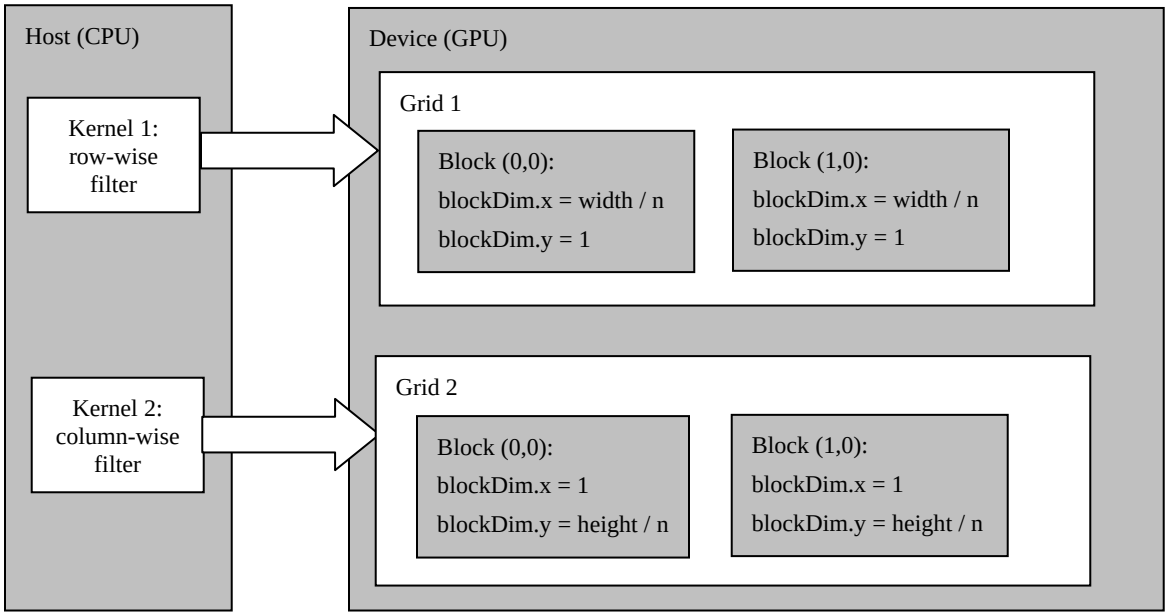


Figure 4: Computational grid arrangement for CUDA framework

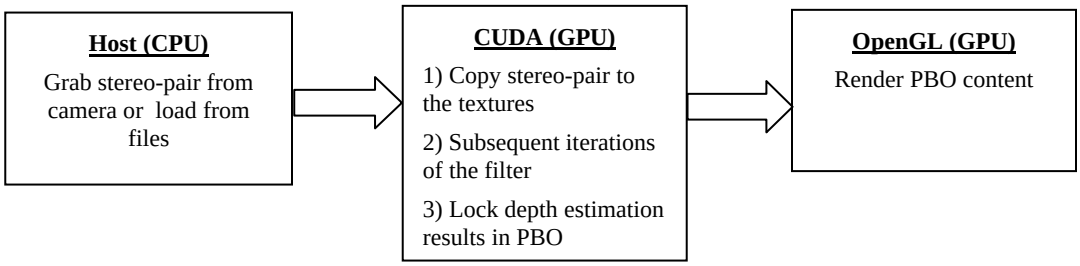


Figure 5: Data flow in proposed implementation

$$\sigma(k) = a_1 \cdot k + b_1, \quad (3)$$

where k is a number of iteration, a_1, b_1 are the linear coefficients.

After filter strength adaptation the algorithm for disparity refinement estimate the filter kernel radius by following formula

$$R(k) = a_2 \cdot k + b_2, \quad (4)$$

where k is a number of iteration, a_2, b_2 are the linear coefficients.

Since we did not interested in abrupt change of parameters from iteration to iteration, a linear dependency has provided us with simple solution to make parameters adaptable.

The idea behind the adaptation of parameters is the following. Since the reliability of source disparity is low, the algorithm starts with smallest filter strength and smallest kernel size. Then, the filter strength and kernel size are increased from iteration to iteration, allowing more resembling pixels to participate in filtration process. Also the adaptation of filter radius leads to decreasing of computational cost. This will be shown by concrete example in Section 5.

After adaptation of parameters the disparity refinement filter is applied. For speeding up of calculation the disparity filter could be defined in separable manner formulated in two passes. The first pass is row-wise processing. The second pass is column-wise one. The parameters of computational grid for CUDA framework is represented in Figure 4. The figure shows separable configuration of disparity processing. The host side resided on CPU has two kernels, which invoke corresponding two-dimensional grids of threaded blocks on the device (GPU). In the first grid, the blocks are organized to process rows of image. Parameters of blocks are set as (blockDim.x = width / n, blockDim.y = 1), where *n* is usually set to *halfwarp* value. This is HW-dependent parameter and it designates a number of stream processors on the multiprocessor. We set this value to 16, since we have used the NVIDIA GeForce 8800GTX. For the future generation of graphics boards this value will increase.

Since the method was implemented in GPU, it gave us the possibility of immediate rendering of disparity estimation results through CUDA – OpenGL interoperability option. More precisely, the disparity estimation results with reference image were locked in pixel-buffer object by CUDA. Then the OpenGL

part of program used that buffers for rendering. This operation has excluded additional data flow to CPU and again to GPU for rendering. Proposed data flow is described in Figure 5. Note that we used textures for storing the stereo-pair. The memory latency during fetching from texture is less, than from global memory. This happens due to texture caching.

5. EXPERIMENTAL RESULTS

Figure 6 represents result of disparity estimation from live stereo-video, while Figure 7 presents the result of disparity estimation for "Tsukuba" image from Middlebury image dataset.

Table 1 and Table 2 summarize objective evaluations of proposed methods. Two metrics were considered: root-mean square (RMS) distance to ground truth and bad pixel metric. Bad pixel is considered as the one, which differs from ground-truth pixel for 1 value. In the tables, the "bad_pixels_all" parameter represents the number of bad pixels in all image, while the "bad_pixels_nonocc" parameter corresponds to the number of bad pixels in non-occluded areas.

For "Tsukuba" image 4 types of filter were considered. The first three of them are classified according to method of similarity computation. For these experiments the radius and filter weight were kept constant for all iterations. The fourth method is the third method with adaptive parameters setting, according to Eq. 3 and Eq. 4.

According to Table 2, the best result for the "Tsukuba" was achieved when using third method with normalization. The resulted disparity map contains only 4 % of bad pixels. Here normalization means the forcing disparity values to neighboring disparity level. This operation decreases number of bad pixel about 1.5-2 times. However, during rendering into auto-stereoscopic display, the visual difference was not captured between normalized and non-normalized results. In our opinion the bad pixel metric does not reflect the quality of disparity properly, since resulted disparity map could have strong distortions of objects shape, and have small value of bad pixels simultaneously.

Proposed methods increase the quality of disparity map in occlusion areas also. This confirms by results in Table 1 and Table 2 (The "rms_error_occ" and "bad_pixels_occ" parameters correspond to value of RMS and number of bad pixels in occlusion areas). According to tables the number of bad pixels in occlusion areas is decreased by half in comparison with raw disparity map. And the RMS is improved by almost 3 times. Correct handling of disparity discontinues, especially in occluded regions greatly facilitates virtual view rendering. The comparative results of state-of-the-art stereo-matching methods and explanation of comparison parameters could be found in [6].

The visual quality of rendering results on the auto-stereoscopic monitor Phillips was nearly the same for 1-st, 3-rd and 4-th methods, while 2-nd method shows the artifacts in the area of lamp and bust. The 2-nd requires less computation than 1-st method. The method works well for smooth areas, while for strong depth discontinues it fails to compute disparity correctly.

To decrease computational burden without degrading in quality the adaption of filter radius was introduced. This means that the computational complexity of 4-th method is lower. This could be illustrated by following calculation. Since we used 6 iterations with radius 20 for the 1-st and 3-rd methods, this requires the 6 x 20 = 120 computations of weights. For the 4-th method with adaptive parameters $a_2 = 5$ and $b_2 = 1$, we get the following computation for the 6 iterations: 1+6+11+16+21+26 = 81. The computation of weight includes calculation of two color distances with exponent. So decreasing the number of weight computation introduces theoretical performance boost about 30% (1 – 81/120). This confirms by software execution time also (Table 3).

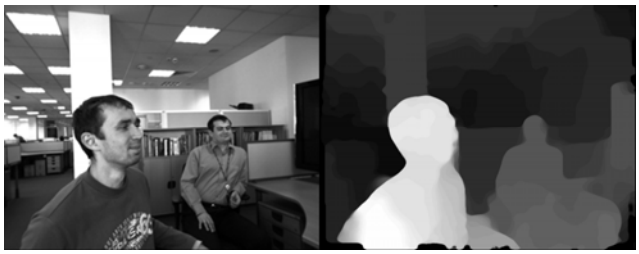


Figure 6: Live 3D capture from stereo camera:
Reference image with correspond disparity

Table 3 shows that maximum performance gain was achieved for 3-rd method. GPU implementation of the method executes 656 times faster than CPU one with float point arithmetic. Also the adaptation of radius in the 4-th method has lead to the least execution time with only 18 msec for frame. This corresponds to 50 fps. Correspondingly, the MDE (Million disparity estimation, MDE = width*height*disp_levels*fps) for tsukuba image with width=384, height =288 and disp_levels = 16 is equal to 97. At the same time, proposed approach shows higher throughput for real-time stereo-matching, and MDE is equal to 294 in this case.

6. CONCLUSION

Recent multi-view 3D displays require multiple views to be synthesized and rendered for 3D scene reproduction. Such virtual views can be generated from stereo content and disparity map. The key factor of high quality virtual view generation is a usability of disparity map which determined by the correct disparities values.

Proposed method of disparity estimation outputs disparity map of high quality computed on TV frame rates. Thus, developed technique could be used for various real-time applications, such as immersive videoconferencing or augmented reality.

7. REFERENCES

[1] J. Mairal, R. Keriven and A. Chariot, "Fast and efficient dense variational stereo on GPU", Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06), pp. 97-104.

[2] R. Yang, M. Pollefeys, "Multi-resolution Real-Time Stereo on Commodity Graphics Hardware" CVPR 2003, pp 211-218.

[3] A. Brunton, C. Shu, G. Roth "Belief Propagation on the GPU for Stereo Vision", Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06) 0-7695-2542-3/06, 2006.

[4] S. Grauer-Gray, C. Kambhamettu, and K. Palaniappan "GPU Implementation of Belief Propagation Using CUDA for

Cloud Tracking and Reconstruction", 5th IAPR Workshop on Pattern Recognition in Remote Sensing (PRRS 2008).

[5] J. Gibson and O. Marques "Stereo Depth with a Unified Architecture GPU", Computer Vision and Pattern Recognition Workshops (CVPRW 2008), pp. 1 – 6.

[6] D. Scharstein and R. Szeliski. "A Taxonomy and Evaluation of Dense two-frame stereo correspondence algorithms", IJCV, volume 47(1), pp. 7-42, 2002.

[7] K. J. Yoon and I. S. Kweon "Adaptive Supprot-Weight Approach for Correspondence Search" IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 28, No 4, April 2006

[8] F. Boughorbel "Adaptive Filters for Depth from Stereo and Occlusion Detection", Stereoscopic Displays and Applications XIX, Proceedings of the SPIE, Volume 6803, 2008.

Table 1. Comparative results for Tsukuba image

Evaluation criteria*	**Raw depth** (SAD 5x5 window)	**1-st method** (2 pixel areas comparison, radius 20)	**2-nd method** (1 pixel area comparison, radius 10)	**3-rd method** (2 pixels comparison radius 20)	**4-th method** (with adaptive parameters a_2=5, b_2=1)
rms_error_all	1.89	0.955	1.163	0.865	0.998
rms_error_nonocc	1.732	0.931	1.143	0.832	0.976
rms_error_occ	5.029	1.62	1.743	1.694	1.619
bad_pixels_all	0.154	0.099	0.159	0.096	0.135
bad_pixels_nonocc	0.137	0.09	0.149	0.086	0.126
bad_pixels_occ	0.826	0.444	0.573	0.458	0.478

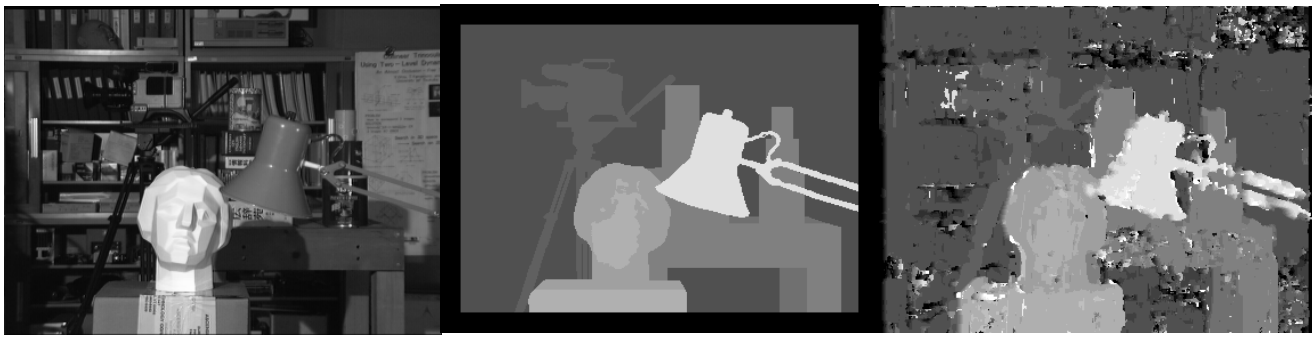
*) For explanation of the evaluation criteria, i.e. "rms_error_all" etc. see [6]

Table 2. Comparative results for Tsukuba image with normalization

Evaluation criteria	**Raw depth** (SAD 5x5 window)	**1-st method**	**2-nd method**	**3-rd method**	**4-th method**
rms_error_all	1.89	0.972	1.176	0.932	1.039
rms_error_nonocc	1.732	0.946	1.156	0.899	1.018
rms_error_occ	5.029	1.683	1.777	1.775	1.646
bad_pixels_all	0.154	0.073	0.092	0.042	0.073
bad_pixels_nonocc	0.137	0.065	0.083	0.034	0.065
bad_pixels_occ	0.826	0.382	0.434	0.323	0.361

Table 3. Performance analysis of Tsukuba image stereo matching

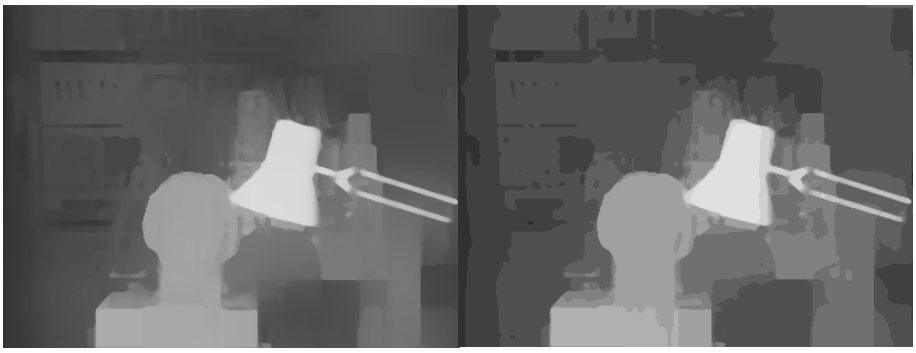
	Time (msec)			
	1-st method	**2-nd method**	**3-rd method**	**4-th method**
GPU implementation (nVidia 8800GTX)	3031.5	504.47	24.5	18.33
CPU implementation– 1 core (INTEL Core2Quad Q6600, 2.4 GHz)	720305.5	126290.6	16069.2	11024.32
Performance gain (times)	237	250	656	601



a)

b)

c)



d)

Figure 7: Experimental results for Tsukuba image **a)** color image, ground truth, raw depth (SAD with 5x5 aggregation window) (from left to right) **b)** proposed 1-st, 2-nd 3-rd methods according to Table 1 (from left to right) **c)** proposed 1-st, 2-nd 3-rd methods with normalization (from left to right) **d)** proposed 4-st method without and with normalization (from left to right)