Hierarchical Disparity Estimation with Programmable 3D Hardware

Christopher Zach
VRVis Research Center
Inffeldgasse 16
A–8010 Graz
Austria
zach@vrvis.at

Konrad Karner
VRVis Research Center
Inffeldgasse 16
A–8010 Graz
Austria
karner@vrvis.at

Horst Bischof
University of Technology Graz
Inffeldgasse 16
A–8010 Graz
Austria
bischof@icg.tu-graz.ac.at

ABSTRACT

This work presents an implementation of a hierarchical disparity estimation algorithm entirely executed on programmable 3D graphics hardware. In contrast to previous hardware based implementations of computational stereo algorithms, our method calculates disparities either for rectified stereo images or uncalibrated pairs of stereo images without known epipolar geometry. We exploit features of modern graphics hardware to search for correct disparity vectors efficiently. The hierarchical approach increases the speed and the robustness of the algorithm. Additionally, we use bidirectional matching to remove false matches. We observe up to 50 fps for input images with 256×256 pixels resolution on desktop graphics cards and up to 30 fps on mobile 3D hardware.

Keywords

Programmable Graphics Hardware, Accelerated Image Processing, Stereo Vision

1 INTRODUCTION

Since programmable 3D graphics hardware has increased its algorithmic and computational power, several research teams exploit the SIMD (single instruction, multiple data) facilities of 3D graphic processing unit (GPU) to accelerate applications not related to rendering and visualization. Using graphics hardware for image processing and computer vision tasks is very natural, since graphics hardware is optimized to access and manipulate 2D images. Even with very early 3D boards it was possible to perform image warping efficiently using the texture mapping feature. With the current generation of GPUs supporting floating point channels and highly programmable fragment programs it is possible to execute more advanced scientific algorithms on such hardware.

In this work we propose a hierarchical algorithm to estimate disparities between two stereo images, which can be executed on DirectX 8.1 class 3D hardware (e.g. on an ATI Radeon 9000 Mobility). The result of the procedure is a dense disparity map, which assigns a vector pointing to the corresponding pixel in the right image for every pixel in the left image. The disparity map can be optionally verified by a back-matching procedure, which identifies and discards false matches.

2 RELATED WORK

The demand for more complex and more realistic virtual realities has accelerated the development of highly programmable 3D graphics hardware (e.g. [Lindh01]). The programmability and computational power of recent 3D hardware is used for non-graphical purposes as well, e.g. for numerical calculations and simulations ([Hopf99b], [Hopf99a], [Harri02], [Thomp02], [Krü03], [Bolz03]).

Programmable graphics hardware is used in the field of image processing for linear and non-linear filtering ([Hadwi01], [Yang02a], [Sugit03], [Colan03], [Viola03]). Some authors present per-

formance comparisons between optimized CPU and GPU implementations of image processing methods. In general, if the GPU's facilities for parallel computation and fast texture access can be exploited, GPU based algorithms can be several times faster than the CPU based counterpart.

Yang et al. [Yang02b] present a real-time depth estimation method utilizing graphics hardware working with several calibrated cameras to increase robustness. Depth values are obtained by a plane sweep through the 3D world space, therefore the time complexity is linear in the desired depth resolution. Later, Yang and Pollefeys refined the method to work even for two camera setups using mipmapping facilities of graphics hardware to calculate a more robust similarity function [Yang03].

Sugita et al. [Sugit03] compare the performance of CPU and GPU implementations of several image filters and implemented a rather simple stereo matching algorithm. Their method is a block-based matching procedure using rectified images as input. The GPU implementation of the stereo matcher runs about twice as fast as the optimized CPU implementation using the Intel Performance Primitives library.

This work extends our earlier work [Zach03a] in several aspects. Our previous implementation iteratively refines the vertices of the current 3D mesh hypothesis and keeps track of the best local modification so far. Evaluating the current mesh hypothesis requires rendering a large mesh, which decreases the overall performance. In later work we improved the speed of the algorithm significantly [Zach03b], but the basic idea remained the same. In contrast to our current method the previous one calculates depth (or disparity) values only for mesh vertices, which are placed every 4 pixels. We still utilize the hierachical approach already used in the earlier implementations and the idea of iterated local refinement of the current hypothesis.

3 OUR METHOD

3.1 Overview of Block Matching

The search for corresponding points in two images is based on block matching, i.e. the method determines the best matching region in the right image for every pixel in the left image. Typically, a rectangular template window is moved over the right image and a similarity (or correlation) function with a fixed window in the left image is evaluated. Since programmable graphics hardware offers only limited control and data flow, only simple correlation functions are applicable for hardware accelerated implementations. In particular, the sum of absolute differences (SAD) between the pixels in the windows, and the sum of squared differences (SSD) are suitable candidates. The result of the block matching procedure is a usually dense disparity image, which maps pixels in the left image to corresponding pixels in the right image.

If the relative orientation between the camera positions is known, the search for corresponding regions can be restricted to a one-dimensional line search. In these cases the source images can be resampled (rectified), such that a search along horizontal scan-lines is sufficient. Furthermore, the search range can be bounded.

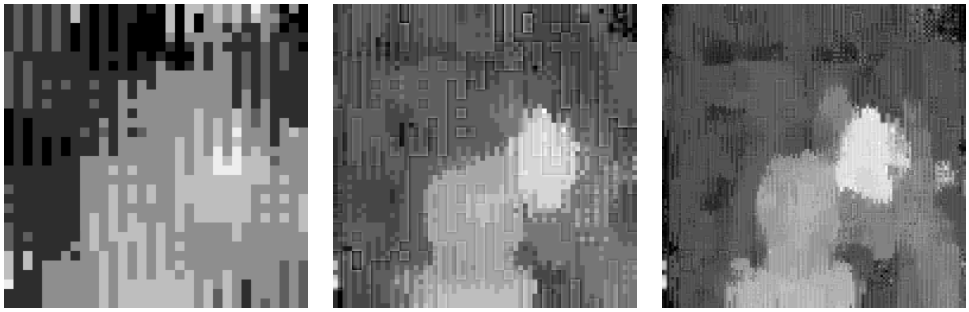
3.2 Hierarchical Matching

The input of the matching procedure consists of two gray-level images. After transfering the input images into textures in graphics memory, an image pyramid is created (either automatically by the graphics hardware or explicitly, if automatic mipmap generation is not available or non-functional). Starting from a coarse image representation (32×32 in our current implementation), the current disparity map is refined in every level of the pyramid. The initial disparity map used in the next (finer) level is the upsampled map obtained in the previous level. Figure 1 shows several intermediate disparity maps generated in the hierarchy. The final disparity map is shown in Figure 3(d). The number of disparity variations evaluated is currently the same in all levels, and the tested disparity range is halved from one level to the next to enable more precise search.

3.3 Disparity Refinement

In this section we describe the procedure executed within a given level in the image pyramid. The disparity map generated in the previous level is refined to obtain a map with higher resolution. This procedure essentially searches for the optimal disparity variation to minimize the sum of absolute differences (SAD) accumulated in the template window.

The procedure consists of two passes: In the first pass the absolute difference pixels between the left image and the warped right image wrt. the current disparity hypothesis are calculated and accumulated over a window. In the second pass a comparison step is performed to determine the optimal disparity value for each pixel.



(a) Coarsest level (32×32 pixels)

(b) Intermediate level (64×64 pixels)

(c) Intermediate level (128×128 pixels)

Figure 1: Disparity maps generated in successive levels of the hierarchy.

3.3.1 Image Warping and SAD accumulation

The right image is warped using a dependent texture lookup. The left image, the right image and the disparity map are bound to texture units. The currently evaluated disparity variation is an additional argument for the fragment program, which performs the required texture lookups and calculates the absolute difference of single pixels between the left image and the warped right image. We allow matching with subpixel accuracy by optionally scaling the disparity values.

In order to calculate the sum of absolute differences within a window, we execute the warping pass repeatedly with suitably jittered texture coordinates. The generated absolute differences between single pixels are accumulated using frame buffer blending with source and destination factors set to `GL_ONE`. Using bilinear texture access we perform SAD computation for an $n \times n$ window using only $n/2 \times n/2$ passes. The original input images are sampled at corner positions to obtain the average value of four adjacent pixels (see Figure 2). For example, a 4×4 window generates 4 instead of 16 texture accesses and the number of blending passes is reduced accordingly. Since absolute differences are calculated after sampling the input images, the result is only an approximation of the true SAD within an $n \times n$ window. Nevertheless we observed that this approach is a reasonable tradeoff between speed and robustness using larger template windows.

3.3.2 Comparison Step

After the SAD for every pixel wrt. the current disparity hypothesis is computed, the current errors

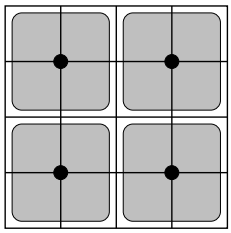


Figure 2: In order to calculate the SAD in a 4×4 region, only the positions indicated by the dots are sampled to interpolate between four adjacent pixels (designated by the gray region).

are compared with the errors of the optimal disparity found so far. The previously optimal disparity values (stored in the red and green channel) together with the corresponding SAD (stored in the alpha channel) are represented by an off-screen pixel buffer, which is bound to a texture unit. The SAD values associated with the currently evaluated disparities (as computed in Section 3.3.1) are bound to a second texture unit and a fragment program calculates the pixel-wise minimum of the SAD values and selects the superior disparity. On more powerful graphics hardware providing depth value assignment inside the fragment program, a more efficient method can be used for pixelwise comparison (see Section 3.5.2).

3.4 Disparity Verification

The role of the input images can be exchanged and the matching procedure can be applied to the reversed pair of images (back-matching, bidirectional matching, see [Egnal02] for a comparison of disparity verification methods). In addition to the

left-to-right disparity map D_L a map D_R of right-to-left disparities is obtained. We discard the disparity value for pixel p, if the following consistency condition is false:

$$|D_R(p + D_L(p)) + D_L(p)| < \varepsilon$$

The disparity value of the corresponding pixel in the right image should point back to the original pixel (within some allowed threshold). Because of the hierarchical approach we evaluate the disparity range at sparse positions and the optimized right-to-left disparity calculation described in [Müh02] cannot be applied. In our framework right-to-left disparity estimation is completely independent from left-to-right disparity calculation and requires the same amount of operations.

3.5 Optimizations

3.5.1 Batched Evaluation

Evaluating exactly one disparity modification in every iteration does not exploit the parallel execution of fragment instructions working simulateously on each channel. Therefore our implementation evaluates four disparity modifications in one step and stores the accumulated errors in the red, green, blue and alpha channel. Technically, the four tested disparity variations are stored in four sets of texture coordinates, which are used by the fragment program to warp the right image four times. The obtained (gray-level) pixel values from the right image are shuffled into one register and the absolute difference with the left image pixel is stored in the generated color fragment. This batched approach reduces the number of time consuming pixel buffer switches (see also [Zach03b]) and uses the fill rate to full capacity.

3.5.2 Depth Replacement for Minimum Determination

The `ARB_fragment_program` OpenGL extension allows depth values to be assigned by the fragment program.[1] This feature can be exploited to accelerate the comparison step described in Section 3.3.2. Moving the error value stored in the appropriate color channel into the depth component allows the depth test to be used to determine the disparity change with the smallest error efficiently [Zach03b], [Sugit03]. This feature is not available on the ATI Radeon 9000 Mobile, therefore the search for the optimal disparity requires two alternating buffers, which are swapped after every iteration. In combination with the batched evaluation described in the previous section it is possible to process two disparity variations within one pass.

[1] Traditionally, depth values are interpolated from vertex geometry and cannot be changed on a fragment basis.

4 RESULTS

Table 1 presents performance results for the Tsukuba dataset on mobile and desktop graphics hardware. This dataset consists of rectified small-baseline stereo images, therefore a 1-dimensional disparity search is sufficient. The input images were converted to gray-scale images and rescaled to 256×256 pixels resolution. The number of evaluated disparity variation is 44 for unidirectional matching and 88 for bidirectional matching. We set the range of disparities to the interval $[-75/4, 75/4]$ containing 151 values (but not every value is actually tested due to the hierarchical approach). The desktop 3D hardware (ATI Radeon 9700 Pro) outperforms the mobile hardware (Radeon 9000 Mobility) for two reasons: (a) the fill rate is higher because of increased clock speed and raised number of parallel pixel pipelines and (b) the depth replace optimization desribed in Section 3.5.2 can be applied. Surprisingly, the timings observed for bidirectional matching are slightly better than the expected timings (i.e. doubled time for unidirectional matching). Using larger window templates increases the matching time sublinearly, too.

We can compare the performance results with the numbers presented by Sugita et al. [Sugit03]: They report 50ms required by an ATI Radeon 9700 Pro for 256×256 image resolution and a disparity search range of 50 (i.e. 50 disparity values are evaluated). Our implementation requires 28ms on the same graphics hardware for 44 iterations, but with a much higher potential accuracy (150 possible disparity values instead of 50).

Figure 3 gives visual results for the Tsukuba dataset. Figure 3(a) shows the left input image for this data and Figures 3(b)–(e) present the computed disparity maps using the hierarchical approach. Figure 3(f) shows the disparity map obtained by a non–hierarchical disparity search. The number of detected false matches in Figure 3(e) is 7.9%, whereas in Figure 3(f) 9.3% of the pixels are rejected. Since the hierachical approach has more isolated false matches, a median filter with small window size is well applicable. Note that this result should not indicate that hierarchical matching has higher accuracy in general, but the quality of disparity maps obtained by dense and hierarchical

Hardware	Resolution	Window size	Backmatching	Time
Radeon 9000 Mobile	256x256	4x4	-	29ms
			yes	51ms
		8x8	-	79ms
			yes	152ms
	512x512	4x4	-	94ms
			yes	187ms
		8x8	-	291ms
			yes	595ms
Radeon 9700 Pro	256x256	4x4	-	12ms
			yes	20ms
		8x8	-	28ms
			yes	51ms
	512x512	4x4	-	32ms
			yes	59ms
		8x8	-	92ms
			yes	180ms

Table 1: Timing results for the Tsukuba Head and Lamp dataset. The original input images are scaled to a resolution of 256×256 pixels and 512×512 pixels, respectively.

approaches should be comparable.

We tested our algorithm on an artificially generated dataset consisting of one textured sphere. Two small-baseline views of the sphere were rendered and captured. This dataset requires a two-dimensional search for correct disparities. In Figure 4 we show generated disparity maps with pixel and subpixel accuracy. We display 2D disparity maps as gray-level images showing the length of the disparity vector. Finally, Figure 5 presents results for small baseline stereo images obtained by a webcam. Since the images do not have known relative orientation, 2D disparity vectors are determined. The procedure performs 484 iterations and requires about $240ms$, and the range of potential disparity vectors has $151 \times 151 = 22801$ elements.

5 DISCUSSION

Beside the sum of absolute differences we have implemented and evaluated other window correlation functions. We tested the sum of squared differences, which resulted in worse disparity maps due to saturation effects and due to the limited accuracy of color channels and registers in the fragment shader. Using a multiresolution correlation function based on mipmapping facilities of graphics hardware [Yang03] generated inferior results as well. In summary, calculating the absolute difference between pixel values is beneficial on hardware with limited precision, since differencing does not change the magnitude of the result.

Non-hierarchical methods have the advantage, that correlation values generated during the search process can be reused e.g. for backmatching, since the correlation function is densely evaluated in the search space. Therefore computation of the correlation function for right-to-left matching can be omitted and replaced by an appropriate lookup of correlation values generated by the prior left-to-right matching. Consequently, the total computation time for bidirectional matching is only increased by approximately 20–25% [Müh02]. Additionally, the result of image differencing can be reused in these cases as well (see Sugita et al. [Sugit03]). Our hierarchical approach evaluates the window correlation function sparsely and these optimizations are not applicable. Nevertheless we consider a hierarchical approach more efficient for large search spaces, in particular for estimation of 2D disparity vectors. There are differences in the quality of obtained disparity maps for hierarchical and non-hierarchical approaches, too. Hierarchical refinement operating on an image pyramid may fail, e.g. if the source images contain solely high frequencies (random dot stereograms, for example), which are nonexistent in the coarser levels of the pyramid. On the other hand, hierarchical methods will have a higher probability to find correct disparity values in the case of repetitive textures than non-hierarchical approaches. Since real scenes have usually a wide range of frequencies, we favor hierachical matching for its efficiency.

6 CONCLUSION AND FUTURE WORK

We presented a fast hierarchical disparity estimation algorithm running entirely on programmable 3D graphics hardware. For relatively small 1D disparity search ranges we achieve even real-time behaviour on desktop PCs and still interactive rates on mobile hardware. Additionally, unreliable disparity values can be detected using bidirectional matching.

We expect that using full color images instead of gray-level (intensity) images increases the accuracy and robustness for the uncalibrated setup, where the search space includes 2D disparity vectors. We would like to extend this work for calibrated stereo setups without the need to perform image rectification. In this setting epipolar lines do not coincide with horizontal scan-lines.

7 ACKNOWLEDGEMENTS

We would like to thank Roland Perko for helpful suggestions and discussion.

This work has been done in the VRVis research center, Graz and Vienna/Austria (http://www.vrvis.at), which is partly funded by the Austrian government research program Kplus.

References

[Bolz03] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In *Proceedings of SIGGRAPH 2003*, pages 917–924, 2003.

[Colan03] P. Colantoni, N. Boukala, and J. Da Rugna. Fast and accurate color image processing using 3D graphics cards. In *Proc. of Vision, Modeling and Visualization 2002*, 2003. to appear.

[Egnal02] G. Egnal and R. P. Wildes. Detecting binocular half-occlusions: Empirical comparison of five approaches. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(8):1127–1133, 2002.

[Hadwi01] M. Hadwiger, T. Theußl, H. Hauser, and M. E. Gröller. Hardware-accelerated high-quality filtering on PC hardware. In *Proc. of Vision, Modeling and Visualization 2001*, pages 105–112, 2001.

[Harri02] M. J. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 109–118, 2002.

[Hopf99a] M. Hopf and T. Ertl. Accelerating 3D convolution using graphics hardware. In *Visualization 1999*, pages 471–474, 1999.

[Hopf99b] M. Hopf and T. Ertl. Hardware-based wavelet transformations. In *Workshop of Vision, Modelling, and Visualization (VMV '99)*, pages 317–328, 1999.

[Krü03] J. Krüger and R. Westermann. Linear algebra operators for GPU implementation of numerical algorithms. In *Proceedings of SIGGRAPH 2003*, pages 908–916, 2003.

[Lindh01] E. Lindholm, M. J. Kilgard, and H. Moreton. A user-programmable vertex engine. In *Proceedings of SIGGRAPH 2001*, pages 149–158, 2001.

[Müh02] K. Mühlmann, D. Maier, J. Hesser, and R. Männer. Calculating dense disparity maps from color stereo images, an efficient implementation. *IJCV*, 47:79–88, 2002.

[Sugit03] K. Sugita, T. Naemura, and H. Harashima. Performance evaluation of programmable graphics hardware for image filtering and stereo matching. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology 2003*, 2003.

[Thomp02] C. J. Thompson, S. Hahn, and M. Oskin. Using modern graphics architectures for general-purpose computing: A framework and analysis. In *35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35)*, 2002.

[Viola03] I. Viola, A. Karnitsar, and E. Gröller. Hardware-based nonlinear filtering and segmentation using high-level shading languages. In *Proceedings of IEEE Visualization '03*, page to appear, 2003.

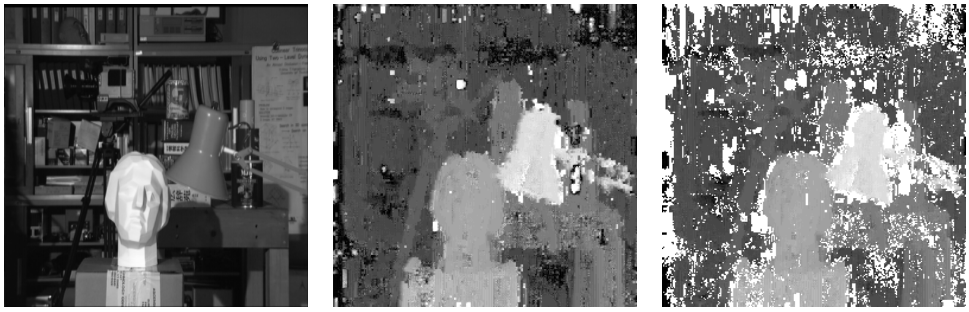
[Yang02a] R. Yang and G. Welch. Fast image segmentation and smoothing using commodity graphics hardware. *Journal of Graphics Tools*, 7(4):91–100, 2002.

[Yang02b] R. Yang, G. Welch, and G. Bishop. Real-time consensus based scene reconstruction using commodity graphics hardware. In *Proceedings of Pacific Graphics*, 2002.

[Yang03] R. Yang and M. Pollefeys. Multi-resolution real-time stereo on commodity graphics hardware. In *Conference on Computer Vision and Pattern Recognition*, 2003.

[Zach03a] C. Zach, A. Klaus, M. Hadwiger, and K. Karner. Accurate dense stereo reconstruction using graphics hardware. In *EUROGRAPHICS 2003, Short Presentations*, 2003.

[Zach03b] C. Zach, A. Klaus, B. Reitinger, and K. Karner. Optimized stereo reconstruction using 3D graphics hardware. In *Workshop of Vision, Modelling, and Visualization (VMV 2003)*, pages 119–126, 2003.



(a) Left image

(b) Disparity obtained with a 4×4 window

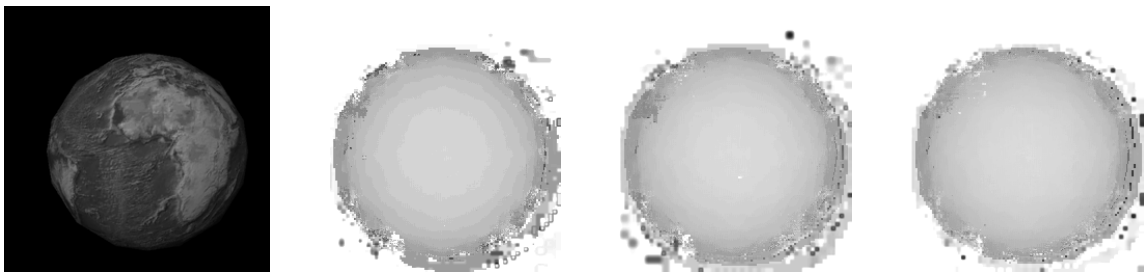
(c) Disparity obtained by bidirectional matching with a 4×4 window

(d) Disparity obtained with an 8×8 window

(e) Disparity obtained by bidirectional matching with an 8×8 window

(f) Disparity obtained by non-hierarchical matching with an 8×8 window

Figure 3: Matching results for the Tsukuba head and lamp data set. White pixels indicate false matches detected with backmatching. Disparities are discarded, if left-to-right and right-to-left disparities disagree by one pixel.



(a) Left image of the sphere dataset

(b) Disparity map with pixel accuracy

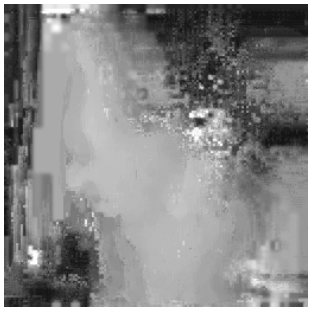
(c) Disparity map with half pixel accuracy

(d) Disparity map with quarter pixel accuracy

Figure 4: Results for the synthetic sphere dataset. This dataset requires a 2D search for disparities. The time for matching is about 240ms on a Radeon 9700 Pro. Note the Mach-banding in Figure (a) because of the single pixel accuracy. The silhouette of the sphere is noisy due to the textureless background.

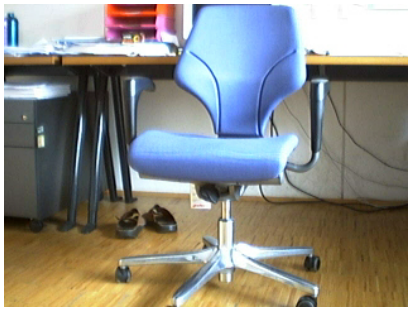
(a) Left image

(b) Right image

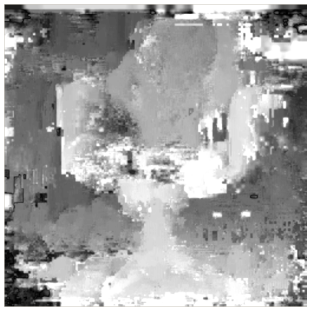


(c) Disparity map

(d) Left image



(e) Right image



(f) Disparity map

Figure 5: Disparity maps for webcam images. The images have unknown orientation, hence unconstrained matching is applied. The matching procedure is applied to a 256×256 image region cut out of the original image generated by the webcam.