Scanline Optimization for Stereo On Graphics Hardware

Christopher Zach Mario Sormann Konrad Karner

VRVis Research Center
Graz, Austria
zach@vrvis.at

Abstract

In this work we propose a scanline optimization procedure for computational stereo using a linear smoothness cost model performed by programmable graphics hardware. The main idea for an efficient implementation of this dynamic programming approach is a recursive scheme to calculate the min-convolution in a manner suitable for the parallel stream computation model of graphics processing units. Since many image similarity functions can be efficiently calculated by modern graphics hardware, it is reasonable to address the final disparity extraction by graphics processors as well. Our timing results indicate that the proposed approach is beneficial for larger image resolutions and disparity ranges in particular.

1. Introduction

Utilizing modern programmable graphics processing units (GPU) for non-graphical computations has gained interest in the last years for several reasons: at first, the raw computational power of GPUs has increased much more than for traditional CPUs, partially because the specific programming model of GPUs enables strong parallelism to be employed. The underlying stream computation model is perfectly suited for several numerical applications. Additionally, the intrinsic texture mapping capabilities allow at least a linearly filtered subpixel access to image data, which is of substantial interest in image processing and early vision applications. Finally, the programming interfaces and feature sets of graphics hardware have recently stabilized, which allows easier developement of GPU-accelerated methods.

Depth and disparity estimation using the GPU as parallel processing unit is usually restricted to a final winner-takes-all extraction method, or alternatively a refinement procedure iteratively modifying the depth map is employed. Exact or approximate global inter- or intrascanline methods are almost always performed by traditional CPUs due to the required in-place (or destructive) value updates only poorly supported by GPUs.

In this work we propose a GPU-based computational stereo approach using scanline optimization to achieve optimal intrascanline disparity maps. Since we employ a linear discontinuity cost model, the central part of the procedure is the calculation of the appropriate min-convolution, which is usually implemented as two pass method using destructive array updates. We replace the in-place updates by a recursive doubling scheme suited for stream programming models. Consequently, the entire dense estimation pipeline from matching cost computation to global optimization to obtain the disparity resp. depth map is performed by the GPU and only the control flow is maintained by the CPU.

2. Related Work

An introduction to the computational model of graphics processing units and an overview of non-graphics related applications using the GPU can be found in [9]. Employing programmable hardware as a parallel stream processor and the mapping of a high level parallel language to GPUs is discussed in [2].

Using the efficient image interpolation capabilities and the parallel computation model of modern, programmable GPUs for real-time or near real-time multi-view depth estimation is addressed by several authors. Many of these approaches are based on a plane-sweep strategy typically combined with a winner-takes-all depth extraction method [13, 12, 11]. The initial depth map obtained by plane-sweeping can be refined incorporating adjacency information to generate higher quality depth images [3].

After computing the matching costs and an optional spatial aggregation step, the depth values are typically extracted by a winner-takes-all approach, which can be efficiently performed by the GPU using the z-buffer capability. Global, energy-based approaches to computational stereo are infrequently addressed: variational methods for

depth estimation on the GPU are described in [8] and [14]. Depth estimation on the GPU using the dynamic programming paradigm seems to be only addressed by Gong and Yang [6]. Their proposed approach is based on the Potts discontinuity cost model, which substantially simplifies (and accelerates) the dynamic programming step. They compared a pure GPU-based implementation with a mixed one, which uses the GPU only for dissimilarity calculation and aggregation. The authors concluded, that the mixed implementation is substantially faster than the pure GPU version. Timing results for a pure CPU version were not presented.

3. Scanline Optimization and Min-Convolution

Scanline optimization [10] searches for a globally optimal assignment of disparity values to pixels in the current (horizontal) scanline, i.e. it finds

$$\arg\min_{d_x} \sum_{x=1}^{W} \left(D(x, d_x) + \lambda V(d_x, d_{x-1}) \right),$$

where $D(x, d)$ is the image dissimilarity cost and $V(d, d')$ is the regularization cost. As in all dynamic programming approaches to stereo, different scanlines are treated independend from the neighboring ones (which may result in vertical streaks visible in the disparity image).

The optimal assignment can be efficiently found using a dynamic programming approach to maintain the minimal accumulated costs $sumCost(x, d)$ up to the current position x:

$$sumCost(x+1, d) = D(x+1, d) + \min_{d_1} \left(sumCost(x, d_1) + V(d, d_1) \right).$$

In a linear discontinuity cost model we have $V(d, d_1) = \lambda |d - d_1|$ and the calculation of

$$\min_{d_1} \left(sumCost(x, d_1) + \lambda |d - d_1| \right)$$

for every d can be performed in linear time using a forward and a backward pass to compute the lower envelope [5]. The linear-time procedure to calculate the min-convolution is given in Algorithm 1.

This procedure is not directly suitable for GPU implementation, since it relies at first on in-place array updates and secondly, a linear number of passes is required to update the entire array h.[1]

The basic idea to enable a GPU implementation of min-convolution is utilizing a recursive doubling approach,

[1] Using the depth test with the same depth buffer as texture source and target buffer would allow the direct implementation, but this approach results in undefined behaviour according to the specifications. Such an approach would have additional disadvantages, mainly the reduced ability to utilize the parallelism of the GPU.

Algorithm 1 Procedure to calculate the lower envelope efficiently

Procedure Min-Convolution
Input: Output $h[]$
  for $d = 1 \dots k$ **do**
    $h[d] \leftarrow sumCost(x, d)$
  end for
  {Forward pass}
  for $d = 2 \dots k$ **do**
    $h[d] \leftarrow \min(h[d], h[d-1] + \lambda)$
  end for
  {Backward pass}
  for $d = k - 1 \dots 1$ **do**
    $h[d] \leftarrow \min(h[d], h[d+1] + \lambda)$
  end for

which is outlined in Algorithm 2. Recursive doubling [4] is a common technique in high-performance computing to enable parallelized implementations of sequential algorithms. This technique is frequently used in GPU-based applications to perform *stream reduction* operations like accumulating all values of a texture image [7].

If we focus on the forward pass in Algorithm 2, the procedure calculates the result of the forward pass for subsequently longer sequences ending in d. Initially, $h_0^+[d]$ contains the min-convolution of the single element sequence $[d, d]$. In every outer iteration with index L the handled sequence is extended to $[d - 2^L, d]$ and its length is doubled. Note, that $h^+[d]$ is defined to be ∞ (i.e. a large constant), if d is outside the valid range $[1 \dots k]$. After all iterations, $h^+[d]$ contains the correct result of the forward pass, which can be easily shown by induction. The same argument applies to the backward pass, hence this procedure yields to the desired result. In addition to the lower envelope h the disparity values for which the minimum is attained are tracked in the array $disp[]$.

Note that the updates in the loops over d are independent and can be performed as parallel loop. In GPGPU terminology, the bodies of these loops are *computational kernels* [2]. Additionally, the scanlines of the images are treated independently, therefore the min-convolution can be performed for all scanlines in parallel.

Figure 1 gives an illustration of the first few iterations in the forward pass of Algorithm 2. Since the next iteration of the outer loops in the min-convolution algorithm refers only to values generated in the previous iteration, only two arrays must be maintained (instead of a logarithmic number of arrays). The role of this two arrays is swapped after every iteration; the destination array becomes the new source and vice versa. In GPU terminology, these arrays correspond to render-to-texture targets, and alternating the roles of these textures is referred as *ping-pong rendering*.

Algorithm 2 Procedure to calculate the lower envelope using recursive doubling

Procedure Min-Convolution using Recursive Doubling

{Forward pass}
for $d = 1 \dots k$ **do**
  $h_0^+[d] \leftarrow sumCost(x, d)$
  $disp[d] \leftarrow d$
end for
for $L = 0 \dots \lceil \log_2(k-1) \rceil$ **do**
  for $d = 1 \dots k$ **do**
    $d_1 \leftarrow d - 2^L$
    $h_L^+[d] \leftarrow \min(h_{L-1}^+[d], h_{L-1}^+[d_1] + \lambda\, 2^L)$
    $disp[d] \leftarrow \arg\min_d(h_{L-1}^+[d], h_{L-1}^+[d_1] + \lambda\, 2^L)$
  end for
end for
{Backward pass}
for $d = 1 \dots k$ **do**
  $h_0^-[d] \leftarrow h_L^+[d]$
end for
for $L = 0 \dots \lceil \log_2(k-1) \rceil$ **do**
  for $d = 1 \dots k$ **do**
    $d_1 \leftarrow d + 2^L$
    $h_L^-[d] \leftarrow \min(h_{L-1}^-[d], h_{L-1}^-[d_1] + \lambda\, 2^L)$
    $disp[d] \leftarrow \arg\min_d(h_{L-1}^-[d], h_{L-1}^+[d_1] + \lambda\, 2^L)$
  end for
end for
Return $h_{\log_2(k-1)}^-$ and $disp$

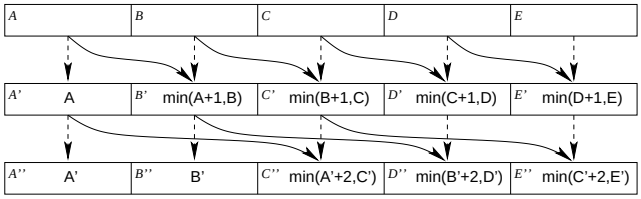

Figure 1. Graphical illustration of the forward pass using a recursive doubling approach.

The full linear discontinuity cost model is often not appropriate and a *truncated* linear cost model with $V(d, d_1) = \lambda \min(T, |d - d_1|)$ is preferable. If T is chosen to be a power of two, the truncated cost model can be incorporated without an additional performance penalty into Algorithm 2 by replacing the $\lambda\, 2^L$ smoothness cost term in the min-convolution algorithm by $\lambda \min(T, 2^L)$. For other values of T an additional pass over the $sumCost(x, \cdot)$ array is required [5]. We restrict our implementation to the pure linear model resp. to the truncated model with power-of-two thresholds.

4. Overall Procedure

This section describes the basic procedure for scanline optimization on the GPU, which consists of several steps. The outline of the overall procedure is presented in Algorithm 3. The input consists of two rectified images with resolution $W \times H$. The range of potential disparity values is $[d_{min}, d_{max}]$ with k elements.

The procedure traverses vertical scanlines positioned at x from left to right. At first the dissimilarity of the current scanline at x in the left image with the set of vertical scanlines $[x + d_{min}, x + d_{max}]$ is calculated, resulting in a texture image with dimensions H and k. The dissimilarity is either a sum of absolute differences aggregated in a rectangular window or the sampling insensitive pixel dissimilarity score proposed in [1].

If the first scanline is processed, the texture storing $sumCost$ is initialized with the dissimilarity score. For all subsequent scanlines the lower envelope of $sumCost$ is computed using Algorithm 2 to obtain $\min_{d_1}(sumCost(x-1, d_1) + \lambda|d - d_1|)$ for every row y and disparity value d. The computation of the lower envelope keeps track of the disparity value, where the mininum is attained (we refer to Section 5.2 for a detailed description of the efficient disparity tracking). These tracked disparities are read back into main memory for the subsequent optimal disparity map extraction. Afterwards, the $sumCost$ array is incremented by the dissimilarity score of the current vertical scanline.

If the final scanline is reached, the total accumulated $sumCost$ is read back in order to determine the optimal disparities for the last column given by $\arg\min_d sumCost(W, d)$. With the knowledge of the disparities for the final column, the disparities for previous columns can be assigned by a backtracking procedure.

5. GPU Implementation Enhancements

The basic method outlined in the last section does not utilize the free parallelism of fragment program operations, which work on four component vectors simultaneously. Consequently, the performance of the method can be substantially improved if this inherent parallelism is taken into account.

5.1. Fewer Passes Through Bidirectional Approach

Essentially, W passes of the min-convolution procedure are required to obtain the final $sumCost$ values and the corresponding disparity map. This number can be effectively halved, if scanline optimization is applied at two opposing horizontal positions simultaneously finally meeting in the

Algorithm 3 Outline of the scanline optimization procedure on the GPU

Procedure Scanline optimization on the GPU

for $x = 1 \dots W$ **do**
  Compute the image dissimilarity for the vertical scanline at x and all possible disparities, resulting in `scoreTex`
  if $x = 1$ **then**
    `sumCostTex` := `scoreTex`
  else
    Calculate the lower envelope h of `sumCostTex` resulting in `lowerEnvTex`.
    Read back tracked disparities from `lowerEnvTex`.
    `sumCostTex` := `lowerEnvTex` + `scoreTex`
  end if
  if $x = W$ **then**
    Read back the accumulated cost for the final column from `sumCostTex`.
  end if
end for
Extract final disparity map by backtracking

central position. More formally, let $sumCost_{fw}(x, d)$ be the accumulated cost starting from $x = 1$ and $sumCost_{bw}$ the cost beginning at $x = W$, which are computed simulatenously using parallel fragment operations. If we assume even W, in every iteration the values for $sumCost_{fw}(x, d)$ and $sumCost_{bw}(W - x + 1, d)$ are determined. The iterations stop at $x_0 := W/2 + 1$ and the total cost for optimal paths with disparity d at position x_0 is

$$sumCost_{fw}(x_0, d) + sumCost_{bw}(x_0, d) - D(x_0, d).$$

Hence the initial disparity assigned to x_0 is the disparity attaining the minimum of this sum and the complete disparity map can be extracted by the backtracking procedure as already outlined. Since multi-pass rendering comes with substantial costs, this modification of the procedure reduces the total runtime by approximately 45% for 384×288 images.

5.2. Disparity Tracking and Improved Parallelism

Using a bidirectional approach does not only reduce the number of passes, but the parallelism of the fragment processor is employed to some extent – two $sumCost$ values are handled in parallel ($sumCost_{fw}$ and $sumCost_{bw}$). Since GPUs are designed to operate on vector values with four components, an additional performance gain can be expected if four $sumCost$ values are stored in the color channels for every pixel.

Note that the calculation of the lower envelope for $sumCost$ is not enough, since the appropriate disparity values attaining the minimum must be stored as well in order to enable an efficient backtracking phase. If one assumes integral disparity values, image dissimilarity scores and an integral smoothness weight λ, then $sumCost$ and h are integer numbers as well. Hence, the associated disparity can be encoded in the fractional part of h. Furthermore, no additional operations are needed to track the disparities attaining the minimal accumulated costs. Of course, in case of ties in the min-convolution procedure, disparities with smaller encoded fractions are prefered (which is as good as any other strategy).

Encoding the disparity value in the fractional part of floating point numbers limits the image resolution in order to avoid precision loss. If the dissimilarity score is an integer from the interval $[0, T]$, then the total accumulated cost is at most $(W/2 + 1) \times T$, where W is the source image width. If the dissimilarity score is discretized into the range $[0, 255]$, 16 bit of the mantissa are required to encode $sumCost$ for half PAL resolution ($W = 384$), which leaves enough accuracy to encode the disparities in the fractional part. The sign bit of the floating point representation can be additionally incorporated by centering the range of dissimilarity scores around 0.

Utilizing this compact representation for accumulated cost/disparity pairs allows us to handle two horizontal scanlines in parallel, thereby reducing the effective image height to the half for the min-convolution. Figure 2 illustrates the parallel processing of two vertical scanlines in the bidirectional approach, and the assignment of the RGBA channels to pixel positions.

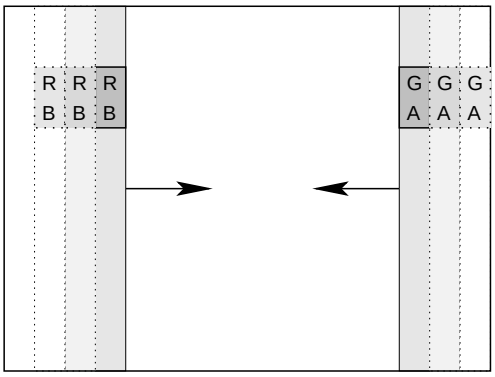

Figure 2. Parallel processing of vertical scanlines using the bidirectional approach for optimal utilization of the four available color channels. The arrows indicate the progression of the processed scanlines in consecutive passes.

5.3. Readback of Tracked Disparities

After the lower envelope is computed, the encoded tracked disparities are read back into main memory to be available for the final back tracking procedure. The tracked disparity values encoded in the fractional part of the lower envelope are extracted directly on the GPU into an 8-bit framebuffer (which is efficient, since fragment programs on NVidia hardware support native instructions to get the fractional part of a floating point number). The tracked disparities are now read back as byte channels. We discovered, that this approach is the fastest, since the usually expensive conversion from floating point numbers to integers is performed on the GPU without a performance penalty and the amount of data to be read back is substantially reduced.

6. Results

At first we give timing results for CPU and GPU implementation of scanline optimization software. The CPU version is a straighforward C++ implementation using the min-convolution as described in Algorithm 1. The disparity map is determined for successive scanlines. Code optimization is left to the compiler. The GPU implementation is based on OpenGL using the frame buffer extension and the Cg language.

The timing tests are performed on two hardware platforms: the first platform is a PC with a 3 GHz Pentium 4 CPU (CPU_A) and an NVidia Geforce 6800 graphics board (GPU_A) running Linux. The C++ source is compiled with gcc 3.4.3 and -O2 optimization. The second system is a PC with an AMD Athlon64 X2 4400+ CPU (CPU_B) and a Gefore 7800GT graphics hardware (GPU_B). The employed compiler is gcc 4.0.1 again with -O2 optimization.

Table 1 displayed the obtained timing results. Tsukuba 1x denotes the original well-known dataset with 384×288 image resolution and 15 possible disparity values. Tsukuba 2x and 4x denote the same dataset, which is resized to 768×288 resp. 1536×288 pixels. The possible disparity range consists of 30 and 60 values, respectively. We select horizontal stretching of the image to simulate sub-pixel disparity estimation.

The Pentagon dataset is another common stereo dataset with 512×512 pixels resolution and 16 potential disparity values (Pentagon 1x). Resizing the images to 1024×1024 resolution yields the Pentagon 2x dataset (32 disparities). The image similarity function in all datasets is the SAD using a 3×1 window calculated on grayscale images. In order to avoid the memory consuming 3D disparity image space the image dissimilarity is calculated on demand for the current vertical scanline.

The results in Table 1 clearly indicate that the multipass GPU method is significantly slower than the CPU version for small image resolutions. For higher resolutions the speed is roughly equal resp. the GPU version shows better performance depending on the hardware. Note that most time is actually spent in the scanline optimization procedure itself; only about 15-20% of the frame time is spent to calculate this particularly simple image dissimilarity. Additionally, the CPU-based backtracking part to extract the optimal disparities has a negligible impact on the total runtime.

	CPU_A	GPU_A	CPU_B	GPU_B
Tsukuba 1x	0.0462	0.1180	0.0373	0.0678
Tsukuba 2x	0.1891	0.2911	0.1387	0.1565
Tsukuba 4x	0.7257	1.0082	0.5655	0.4566
Pentagon 1x	0.1261	0.1877	0.0953	0.1165
Pentagon 2x	0.9458	1.0381	0.7065	0.4930

Table 1. Average timing result for various dataset sizes in seconds/frame.

The required time grows almost linearly on the CPU with increasing resolution, which is in contrast to the GPU curve. In theory, the 4 times stretched Tsukuba dataset should require the 16-fold runtime (fourfold number of disparities and horizontal pixels). The CPU version matches this expectation largely (15.1 and 15.7-fold runtime), whereas the GPU shows a sublinear behaviour (8.5 resp. 6.7-fold runtime). At low resolutions the setup times for frame buffers etc. become a more dominant fraction of the total runtime.

In order to provide a visual proof for the correctness of the proposed GPU implementation, the disparity maps for the Tsukuba dataset at different resolutions are shown in Figure 3.

7. Discussion

From the timing results presented in the previous section it can be concluded, that a GPU-based scanline optimization procedure is mostly suitable for larger images and disparity ranges, but not truly appropriate for realtime applications in particular. For small image resolutions the overhead of multipass rendering is still too significant to take advantage of the processing power of modern GPUs. Additionally, a scanline optimization procedure using a linear smoothness cost model is better dedicated for larger disparity ranges, where a (potentially truncated) linear model is preferable over the Potts model. If the disparity range contains only a few values, enforcing smooth disparity maps is futile, since consecutive values in the disparity range typically correspond to substantial depth discontinuties. Hence, a linear model is not effective in case of few potential disparities and a different approach like the near-realtime reliable dynamic programming (RDP) approach [6] is better suited. On the

(a) 1x (b) 2x (c) 4x

Figure 3. Disparity images for the Tsukuba dataset for increasing horizontal resolutions generated by the GPU-based scanline approach.

contrary, we believe that the Potts model used in the RDP approach is not appropriate for high-quality reconstruction applications. Consequently, future investigations will address non-realtime, but still interactive multi-view model generation.

A mixed computation approach with the GPU calculating and aggregating the similarity scores and the CPU performing the final depth extraction is possible as well. Several image resampling operations like distortion removal and source image rectification can be performed efficiently by the GPU due to its dedicated texturing hardware. A mixed approach has the disadvantage that a significant amount of data is transfered between the GPU and the CPU, which may require additional data conversion. In our setting with the simple 3×1 SAD window the performance of a mixed approach (0.106s for the Tsukuba 1x dataset) is only slightly faster than the pure GPU method, but considerably slower than the pure software implementation. Conversely, if image dissimilarity calculation is more demanding and can be efficiently performed by the GPU, a pure GPU-based pipeline for dense stereo might be beneficial, since the CPU is free to perform other tasks (e.g. data transfer from a video camera, or high level interpretation of the obtained disparity maps). Definite timing results and conclusions require further investigations.

8. Summary and Future Work

In this paper we propose a scanline optimization procedure for disparity estimation suitable for stream architectures like modern programmable graphics processing units. Although the direct implementation of scanline optimization using destructive (i.e. in-place) value updates must be replaced by a more expensive recursive approach, the huge computational power of current GPUs turns out to be beneficial for larger image resolutions and disparity ranges. Consequently, the entire disparity estimation pipeline comprising of matching score computation and global disparity extraction can be performed on graphics hardware, thereby avoiding the relatively costly data transfer between the CPU and the GPU.

Future work will focus on the incorporation of the proposed method into our high-performance multi-view 3D reconstruction workflow, which uses currently a winner-takes-all depth extraction approach. Two items need to be addressed in particular: at first, the relatively small number of disparity values is replaced by a larger number of depth hypotheses in order to obtain highly accurate models. Together with the increased image resolutions in this setting the disparity/depth encoding scheme may require an adaption. Additionally, the simple SAD like dissimilarity measure needs to be replaced by a correlation function, which is more robust under changing lighting conditions (like the normalized cross correlation). The efficient calculation of such similarity measures without generating the full disparity space image is one goal of future studies.

Acknowledgments

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

[1] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):401–406, 1998.

[2] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: Stream computing on graphics hardware. In *Proceedings of SIGGRAPH 2004*, pages 777–786, 2004.

[3] N. Cornelis and L. Van Gool. Real-time connectivity constrained depth map computation using programmable graphics hardware. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1099–1104, 2005.

[4] P. Dubois and G. H. Rodrigue. An analysis of the recursive doubling algorithm. *High Speed Computer and Algorithm Organization*, pages 299–307, 1977.

[5] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, pages 261–268, 2004.

[6] M. Gong and Y.-H. Yang. Near real-time reliable stereo matching using programmable graphics hardware. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 924–931, 2005.

[7] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra. Fast summed-area table generation and its applications. In *Proceedings of Eurographics 2005*, pages 547–555, 2005.

[8] J. Mairal and R. Keriven. A GPU implementation of variational stereo. Technical Report 05-13, CERTIS, 2005.

[9] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–51, 2005.

[10] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.

[11] J. Woetzel and R. Koch. Real-time multi-stereo depth estimation on GPU with approximative discontinuity handling. In *1st European Conference on Visual Media Production (CVMP 2004)*, pages 245–254, 2004.

[12] R. Yang, M. Pollefeys, and S. Li. Improved real-time stereo on commodity graphics hardware. In *CVPR 2004 Workshop on Real-Time 3D Sensors and Their Use*, 2004.

[13] R. Yang, G. Welch, and G. Bishop. Real-time consensus based scene reconstruction using commodity graphics hardware. In *Proceedings of Pacific Graphics*, pages 225–234, 2002.

[14] C. Zach and K. Karner. PDE-based depth estimation on the GPU. Technical report, VRVis Research Center, 2005.