Optimized Stereo Reconstruction Using 3D Graphics Hardware

Christopher Zach, Andreas Klaus, Bernhard Reitinger* and Konrad Karner

August 28, 2003

Abstract

In this paper we describe significant enhancements of our recently developed stereo reconstruction software exploiting features of commodity 3D graphics hardware. The aim of our research is the acceleration of dense reconstructions from stereo images. The implementation of our hardware based matching procedure avoids loss of accuracy due to the limited precision of calculations performed in 3D graphics processors. In this work we target on improving the time to obtain reconstructions for images with small resolution, and we covered other enhancements to increase the speed for highly detailed reconstructions. Our optimized implementation is able to generate up to 130 000 depth values per second on standard PC hardware. Additionally we extend the original epipolar reconstruction approach to estimate disparities between two images without known orientation. Therefore we can calculate e.g. the optical flow between successive images using 3D hardware.

1 Introduction

The automatic creation of digital 3D models from images of real objects comprise a main step in the generation of virtual environments and is still a challenging research topic. Many stereo reconstruction algorithms to obtain models from images were proposed, and almost all approaches are based on traditional computations performed in the main CPU. Very recently few methods were proposed, that exploit the parallelism and special-purpose circuits found in current 3D graphics hardware to accelerate the reconstruction process.

In our earlier work [20] we proposed a hierarchical, area-based stereo matching procedure on programmable 3D graphics hardware. Although the method is still significantly slower than the plane sweeping approach proposed by Yang et al. [19] (using a five camera setup), our approach provides good results with a calibrated stereo setup. Additionally the domain of achieved depth values is larger due to the local mesh refinements employed in our method.

In this work we present substantial performance enhancements of our matching procedure without losing accuracy. The key idea is the amortization of computations across several iterations of the algorithm. Further, we implement a disparity matching method to calculate the optical flow between two images.

After a section on related work we briefly outline our original hardware-based stereo reconstruction method in Section 3. Our recent improvements are presented in Sections 4–7. Section 8 presents benchmark comparisons between the original and new method to illustrate the achieved performance gain, and finally we outline future work.

2 Related Work

2.1 Reconstruction from Stereo Images

Stereo matching is the process of finding corresponding or homologous points in two or more images and is an essential task in computer vision.

We distinguish between feature and area based methods. Feature based methods incline to sparse but accurate results, whereas area based methods de-

liver dense point clouds. The former methods use local features that are extracted from intensity or color distribution. These extracted feature vectors are utilized to solve the correspondence problem. Area based methods employ a similarity constraint, where corresponding points are assumed to have similar intensity or color within a local window. Frequently it is necessary to incorporate a continuity constraint to obtain better reconstructions in homogeneously textured regions. If the relative orientations of the images are known, the search for corresponding points can be restricted to epipolar lines and the search problem is only one-dimensional. This leads to significant improvements in accuracy as well in performance. A good collection and comparison of different matching methods can be found in [3]. Brown et al. [1] give an overview of recent developments of stereo vision methods. In order to obtain faster convergence and to avoid local minima we employ a hierarchical approach for matching [8, 10], which is in particular inspired by the work of Redert et al. [15]

Some applications for stereo matching require fast or even real-time reconstructions, e.g. robot navigation or real-time view synthesis. The Triclops vision system [7] consists of a hardware setup with three cameras and appropriate software for realtime stereo matching. The system is able to generate depth images at a rate of about 20Hz for images up to 320x240 pixels on current PC hardware. The software exploits the particular orientation of the cameras and MMX/SSE instructions available on current CPUs. In contrast to the Triclops system our approach can handle images from cameras with arbitrary relative orientation.

Yang et al. [19] developed a fast stereo reconstruction method performed in 3D hardware by utilizing a plane sweep approach to find correct depth values. The number of iterations is linear in the requested resolution of depth values, therefore this method is limited to rather coarse depth estimation. Further, their approach requires a true multi-camera setup to be robust, since the error function is only evaluated for single pixels. In later work the method was made more robust using trilinear texture access to accumulate error differences within a window [18].

2.2 Accelerated Computations in Graphics Hardware

Even before programmable graphics hardware was available, the fixed function pipeline of 3D graphics processors was utilized to accelerate numerical calculations [5, 6] and even to emulate programmable shading [13]. The introduction of a quite general programming model for vertex and pixel processing [9, 14] opened a very active research area. The primary application for programmable vertex and fragment processing is the enhancement of photorealism in interactive visualization systems (e.g. [2, 4]) and entertainment applications ([11, 12]).

Recently several authors identified current graphics hardware as a kind of auxiliary processing unit to perform SIMD (single instruction, multiple data) operations efficiently. Thompson et al. [17] implemented several non-graphical algorithms to run on programmable graphics hardware and profiled the execution times against CPU based implementations. They concluded that an efficient memory interface (especially when transferring data from graphics memory into main memory) is still an unsolved issue. For the same reason our implementation is designed to minimize memory traffic between graphics hardware and main memory. Strzodka [16] discusses the emulation of higher precision numbers with several 8 bit color channels. We faced a similar problem of manipulating large integer values and storing them in the frame buffer and texture maps for later use.

3 Overview of Our Matching Procedure

The original implementation of our 3D graphics hardware based matching procedure is described in [20]. The matching algorithm essentially warps the left and the right image onto the current triangular 3D mesh hypothesis according to the relative orientation between the images. The projection of the mesh into the left image constitutes always a regular grid (i.e. the left image is never distorted). The warped images are subtracted and the sum of absolute diffenences is calculated in a window around the mesh vertices. New

mesh hypotheses are generated by moving mesh vertices along the camera rays for the left image (epipolar constraint). Different mesh hypotheses generate different local errors and the matching procedure determines the point-wise optimal depth component of vertices.

The main steps of the algorithm, namely image warping and differencing, sum of absolute differences calculation, and finding the optimal depth value are performed on the graphics processing unit. The main CPU executes the control flow and updates the current mesh hypothesis. In order to minimize transfer between host and graphics memory most changes applied to the current mesh geometry are only applied virtually without requiring real update of 3D vertices. For details we refer to our earlier work.

The control flow of the matching algorithm consists of 3 nested loops:

1. The outermost loop adds the hierarchical approach to our method and reduces the probability of local minima e.g. imposed by repetitive structures. In every iteration the mesh and image resolutions are doubled. The refined mesh is obtained by linear (and optionally median) filtering of the coarser one.

2. The inner loop chooses a set of independent vertices to be modified and updates the depth values of these vertices after performing the innermost loop. In order to avoid influence of depth changes of neighboring vertices, only one quarter of the vertices can be modified simultaneously.

3. The innermost loop evaluates depth variations for candidate vertices selected in the enclosing loop. The best depth value is determined by repeated image warping and error calculation wrt. the tested depth hypothesis. The body of this loop runs entirely on the 3D graphics processing unit (GPU).

Figure 1 illustrates the key steps of the algorithm. The result of the procedure is a 3D mesh or point cloud consisting of the sampling points. In the current framework we use one sampling vertex every four pixels, hence images with one megapixel resolution yield to a mesh with 256×256 vertices.

4 Amortized Difference Image Generation

For larger image resolutions (e.g. 1024×1024) rendering of the corresponding mesh generated by the sampling points takes a considerable amount of time. In the 1-megapixel case the mesh consists of approximately 131 000 triangles, which must be rendered for every depth value (several hundred times in total). Especially on mobile graphic boards mesh processing implies a severe performance penalty: stereo matching of two 256×256 images has comparable speed on a desktop GPU and on a usually slower mobile GPU intended for laptop PCs, but matching 1-megapixel images requires two times longer on the mobile GPU.

In order to reduce the number of mesh drawings up to four depth values are evaluated in one pass. We use multitexturing facilities to generate four texture coordinates for different depth values within the vertex program. The fragment shader calculates the absolute differences for these deformations simultaneously and stores the results in the four color channels (red, green, blue and alpha). Note that the mesh hypothesis is updated infrequently and the actually evaluated mesh is generated within the vertex shader by deforming the incoming vertices according to the current displacement.

The vertex program has now more work to perform, since four transformations (matrix-vector multiplications) are executed to generate texture coordinates for the right image for each vertex. Nevertheless, the obtained timing results (see Section 8) indicate a significant performance improvement by utilizing this approach. Several operations are executed only once for up to 4 mesh hypotheses: transferring vertices and transforming them into window coordinates, triangle rasterization setup and texture access to the left image.

5 Chunked Image Transforms

In contrast to Yang and Pollefeys [18] we calculate the error within a window explicitly using multiple passes. In every pass four adjacent pixels are accumulated and the result is written to a temporary

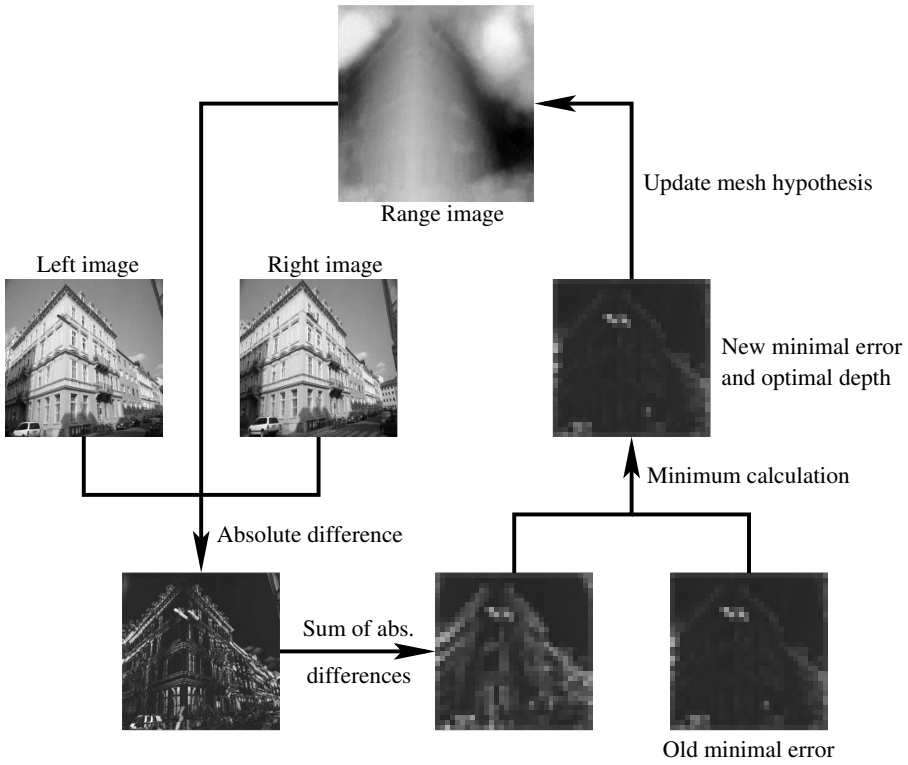


Figure 1: The basic workflow of the matching procedure. For the current mesh hypothesis a difference image between left image and warped right image is calculated in hardware. The error in the local neighborhood of the modified vertices are accumulated and compared with the previous minimal error value. The result of these calculations are minimal error values (stored in the red, green and blue channel) and the index of the best modification of vertices so far (stored in the alpha channel). All these steps are executed in graphics hardware and do not require transfer of large datasets between main memory and video memory.

off-screen frame buffer (usually called pixel buffer or P-buffer for short). It is possible to set pixel buffers as destination for rendering operations (write access) or to bind a pixel buffer as a texture (read access), but a combined read and write access is not available. In the default setting the window size is 8×8, therefore 3 passes are required. Note that we use specific encoding of summed values to avoid overflow due to the limited accuracy of one color channel. We refer to [20] for details.

Executing this multipass pipeline to obtain the sum of absolute differences within a window requires several P-buffer activations to select the correct target buffer for writing. These switches turned out to be relatively expensive (about 0.15ms per switch). In combination with the large number of switches the total time spent within these operations comprise a significant fraction of the overall matching time (about 50% for 256×256 images). If the number of these operations can be optimized, one can expect substantial increase in performance of the matching procedure.

Instead of directly executing the pipeline in the innermost loop (requiring 5 P-buffer switches) we reorganize the loops to accumulate several intermediate

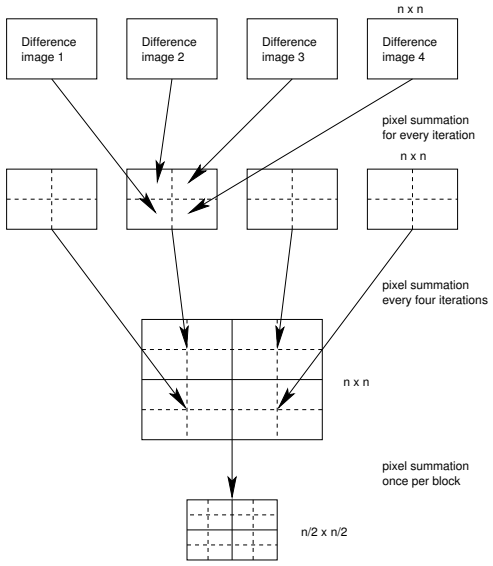


Figure 2: The modified pipeline to minimize P-buffer switches. Several temporary results are accumulated in larger pixel buffers arranged like tiles. Later passes operate on all those intermediate results and are therefore executed less frequently.

results in one larger buffer with temporary results arranged in tiles (see Figure 2). Therefore P-buffer switches are amortized over several iterations of the innermost loop. This flexibility in the control flow is completely transparent and needs not to be coded explicitly within the software. Those stages in the pipeline waiting for the input buffer to become ready are skipped automatically.

6 Minimum Determination Using the Depth Test

In our original implementation we utilized a fragment program to update the minimal error and optimal depth value found so far. This approach works on mobile graphic cards, but it is rather slow due to necessary P-buffer activations (since the minimum computation cannot be done in-place). Z-buffer tests allow conditional updates of the frame buffer in-place, but only very current graphics cards support user-defined assignment of z-values within the fragment shader (e.g. by using the `ARB_FRAGMENT_PROGRAM` OpenGL extension). Older hardware always interpolates z-values from the given geometry (vertices).

We use the rather simple fragment program shown in Figure 3 to obtain one scalar error value from the color coded error and move this value to the depth register used by graphics hardware to test the incoming depth against the z-buffer. Using the depth test provided by 3D graphics hardware, the given index of the currently evaluated depth variation and the corresponding sum of absolute differences is written into the destination buffer, if the incoming error is smaller than the minimum already stored in the buffer. Therefore a point-wise optimum for evaluated depth values for mesh vertices can be computed easily and efficiently.

```
PARAM depth_index = program.env[0];
PARAM coeffs = { 1/256, 1/16, 1, 0 };
TEMP error, col;
TEX col, fragment.texcoord[0],
      texture[0], 2D;
DP3 error, coeffs, col;
MOV result.color, depth_index;
MOV result.depth, error;
```

Figure 3: Fragment program to transfer the incoming, color coded error value to the depth component of the fragment. The dot product (`DP3`) between the texture element and the coefficient vector restores the scalar error value encoded in the color channels.

7 Disparity Estimation

A rather straightforward extension of our epipolar matching procedure is disparity estimation for two uncalibatrated images without known relative orientation. In order to address this setting the 3D mesh hypothesis (i.e. a 2D to 1D (depth) mapping) is replaced by a 2D to 2D (texture coordinates) mapping, but the basic algorithm remains the same. Instead of an 1D search along the epipolar line the procedure

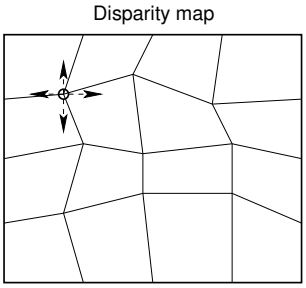


Figure 4: 2D search for the appropriate disparity value to warp the right image. The arrows indicate search directions at the sampling point.

performs a 2D search for appropriate disparity values. In this setting the rather complex vertex shader to generate texture coordinates for the right image used for epipolar matching can be omitted. In every iteration the current 2D disparity map hypothesis is modified in vertical and horizontal direction and evaluated (Figure 4). With this approach it is possible to find corresponding regions in typical small baseline settings (i.e. only small change in position and rotation between the two images).

8 Results

We run timing experiments on a desktop PC with an Athlon XP 2700 and an ATI Radeon 9700 and on a laptop PC with a mobile Athlon XP 2200 and an ATI Radeon 9000 Mobility. Table 1 depicts timing results for those two hardware setups and illustrates the effect of the various optimizations mentioned in the previous sections. The timings were obtained for epipolar stereo matching of the dataset shown in Figure 5. The total number of evaluated mesh hypothesis is 224 (256x256), 280 (512x512) and 336 (1024x1024). In the highest resolution (1024x1024) each vertex is actually tested with 84 depth values out of a range of approximately 600 possible values. Because of limitations in graphics hardware we are currently restricted to images with power of two dimensions.

We compared the original unoptimized implementation with the methods incorporating only one of the enhancements described in Section 4 and Section 5 and with the fasted method using all enhancements (including the technique described in Section 6 on the desktop PC). The desktop PC is surprisingly slow for lower resolutions, but the faster vertex processing and increased fill rate is obvious for the highest tested resolution. Although the benchmark results are given only for the particular dataset, timings for other datasets are similar (but not always completely identical).

Figure 5 gives visual results for stereo images showing the Merton college. Figure 5(a) and (b) comprise the source images, and Figure 5(c) shows the depth image generated by the epipolar matching procedure, whereas Figure 5(d) displays the magnitude of disparity between the source images. Brighter regions indicate larger disparities. Finally, Figure 5(e) shows a screenshot of the 3D point cloud obtained by the matching procedure.

9 Conclusions and Future Work

We presented several significant performance enhancements of our 3D hardware based stereo matching procedure to obtain faster reconstructions without loss in accuracy compared to the original approach. Further, we illustrated graphics hardware based disparity estimation descending from the epipolar stereo matcher.

With the emergence of consumer level DirectX 9 graphic boards supporting floating point precision for color channels and rather orthogonal and powerful instruction sets, more advanced hardware based algorithms to obtain 3D models from multiple images can be implemented. We intend to utilize an adaptive mesh refinement approach instead of the pure regular grid to generate suitable 3D models in regions with little texture information. Alternatively we have initial support for a smoothing term incorporated into the error value to favor planar reconstructions in homogeneous regions. Especially for disparity matching it is an objective to obtain disparity vectors for every

Hardware	Method	256x256	512x512	1024x1024
Desktop				
	Original	0.315	0.57	1.34
	Amortized	0.227	0.358	0.747
	Chunked	0.249	0.496	1.28
	Fastest	0.106	0.198	0.501
Laptop				
	Original	0.295	0.816	2.75
	Amortized	0.169	0.37	1.10
	Chunked	0.235	0.75	2.66
	Fastest	0.095	0.31	1.05

Table 1: Timing results (in seconds) for the apartment building dataset on two different graphic cards. The original implementation is compared with methods incorporating the improvements described in Section 4–6. We show the influence of individual optimizations (amortized mesh rendering and chunked image transforms) as well.

single pixel (instead for every sampling vertex). This goal requires a very different approach to be feasible for graphics hardware.

10 Acknowledgements

We would like to thank Horst Bischof for careful reading of the manuscript and helpful suggestions.

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] M. Z. Brown, D. Burschka, and G. D. Hager. Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):993–1008, 2003.

[2] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Eurographics / SIGGRAPH Workshop on Graphics Hardware '01*, pages 9–16, 2001.

[3] O. Faugeras, J. Malik, and K. Ikeuchi, editors. *Special Issue on Stereo and Multi-Baseline Vision. International Journal of Computer Vision*, 2002.

[4] 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.

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

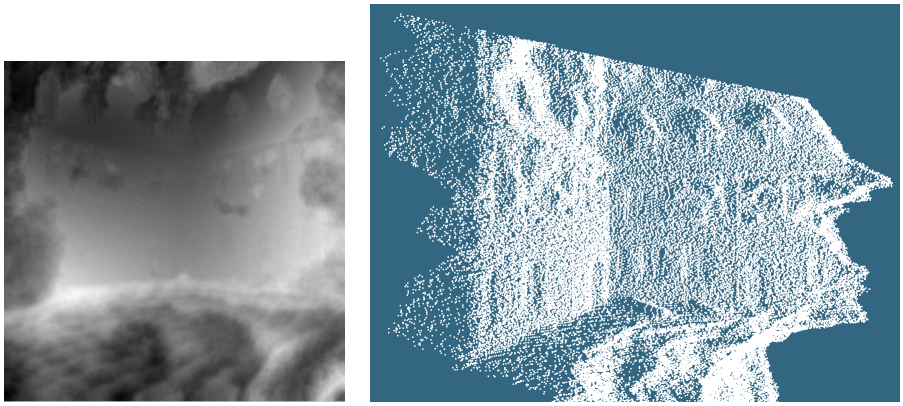
[6] M. Hopf and T. Ertl. Hardware-based wavelet transformations. In *Workshop of Vision, Modelling, and Visualization (VMV '99)*, pages 317–328, 1999.

[7] Point Grey Research Inc. http://www.ptgrey.com.

[8] H. S. Lim and T. O. Binford. Structural correspondence in stereo vision. In *Proc. Image Unterstanding Workshop*, volume 2, pages 794–808, 1988.

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

(a) Left image (b) Right image (c) The depth image



(d) The disparity image (e) The reconstructed model as 3D point cloud

Figure 5: Visual results for the Merton college dataset. The source images have a resolution of 1024×1024 pixels.

[10] C. Menard and M. Brändle. Hierarchical area-based stereo algorithm for 3D acquisition. In *Proceedings International Workshop on Stereoscopic and Three Dimensional Imaging*, pages 195–201, 1995.

[11] J. L. Mitchell. Hardware shading on the Radeon 9700. ATI Technologies, 2002.

[12] NVidia Corporation. Developer relations. http://developer.nvidia.com.

[13] M. S. Peercy, M. Olano, J. Airey, and P. J. Ungar. Interactive multi-pass programmable shading. In *Proceedings of SIGGRAPH 2000*, pages 425–432, 2000.

[14] K. Proudfoot, W. Mark, S. Tzvetkov, and P. Hanrahan. A real-time procedural shading system for programmable graphics hardware. In *Proceedings of SIGGRAPH 2001*, pages 159–170, 2001.

[15] A. Redert, E. Hendriks, and J. Biemond. Correspondence estimation in image pairs. *IEEE Signal Processing Magazine*, pages 29–46, 1999.

[16] R. Strzodka. Virtual 16 bit precise operations on RGBA8 textures. In *Proc. of Vision, Modeling and Visualization 2002*, pages 171–178, 2002.

[17] 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.

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

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

[20] C. Zach, A. Klaus, M. Hadwiger, and K. Karner. Accurate dense stereo reconstruction using graphics hardware. In *EUROGRAPHICS 2003, Short Presentations*, 2003. available as report from www.vrvis.at.