To appear in CVPR 2003

Multi-Resolution Real-Time Stereo on Commodity Graphics Hardware

Ruigang Yang and Marc Pollefeys*
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina, USA

Abstract

In this paper a stereo algorithm suitable for implementation on commodity graphics hardware is presented. This is important since it allows to free up the main processor for other tasks including high-level interpretation of the stereo results. Our algorithm relies on the traditional sum-of-square-differences (SSD) dissimilarity measure between correlation windows. To achieve good results close to depth discontinuities as well as on low texture areas a multi-resolution approach is used. The approach efficiently combines SSD measurements for windows of different sizes. Our implementation running on an NVIDIA GeForce4 graphics card achieves 50-70M disparity evaluations per second including all the overhead to download images and read-back the disparity map, which is equivalent to the fastest commercial CPU implementations available. An important advantage of our approach is that rectification is not necessary so that correspondences can just as easily be obtained for images that contain the epipoles. Another advantage is that this approach can easily be extended to multi-baseline stereo.

1 Introduction

Depth from stereo has traditionally been, and continues to be one of the most actively researched topics in computer vision. While some recent algorithms have obtained excellent results by casting the stereo problem as a global optimization problem, real-time applications today have to rely on local methods, most likely correlation-based ones, to obtain dense depth maps in real time and online.

It is only recently that real-time implementations of stereo vision became possible on commodity PCs, with the help of rapid progress in CPU clock speed, and assembly level optimizations utilizing special extensions of the CPU instruction set, such as the MMX extension from Intel. While it is quite amazing that some of them could perform up to 65 million disparity estimations per second (Mde/s) in software [3, 4, 5], there are few CPU cycles left to perform other tasks including high-level interpretation of the stereo results. In many real-time applications, such as robot navigation, to calculate a raw depth map is only the first step in the entire processing pipeline.

In this paper, we will take advantage of an ubiquitous component of every commodity PC – the graphics card. We present a multi-resolution stereo algorithm that allows a standard Graphic Processor Unit (GPU) to perform many tens of millions of disparity evaluations per second. Our method runs completely on the graphics hardware. Once the input images are downloaded to the graphics board, the CPU is essentially idle.

At the heart of our method is a multi-resolution approach to achieve good results close to depth discontinuities as well as on low texture areas. We combine the sum-of-square-differences (SSD) dissimilarity measures for windows of different sizes. This is, in fact, equivalent to using a large *weighted* correlation kernel with a pyramid shape. By utilizing the mipmap functionality [13] on the graphics hardware, we can compute this dissimilarity measure very efficiently.

2 Related Work

Stereo vision is one of the oldest and most active research topics in computer vision. It is beyond the scope of this paper to provide a comprehensive survey. Interested readers are referred to a recent survey and evaluation by Scharstein and Szeliski [12]. While many stereo algorithms obtain high-quality results by performing optimizations, today only correlation-based stereo algorithms are able to provide a dense (per pixel) depth map in real time on standard computer hardware.

Only a few years ago even correlation-based stereo algorithms were out of reach of standard computers so that special hardware had to be used to achieve real-time performance [2, 7, 15, 8, 6].

In the meantime, with the tremendous advances in computer hardware, software-only real-time systems begin to merge. For example, Mulligan and Daniilidis proposed a new trinocular stereo algorithm in software [10]

*This work was supported in part by the NSF grant ISS-0237533, and by a generous 2002-2003 Link Foundation fellowship.

to achieve 3-4 frames/second on a single multi-processor PC. Hirschmuler introduced a variable-window approach while maintaining real-time suitability [4, 3]. There is also a commercial package from Point Grey Research [5] which seems to be the fastest one available today. They reported 45Mde/s on a 1.4GHz PC, which extrapolates to 64Mde/s on a 2.0GHz PC.

All these methods used a number of techniques to accelerate the calculation, most importantly, assembly level instruction optimization using Intel's MMX extension. While the reported performance of 35-65Mde/s is sufficient to obtain dense-correspondences in real-time, there are few CPU cycles left to perform other tasks including high-level interpretation of the stereo results. Furthermore, most approaches use an equal-weight box-shaped filter to aggregate the correlation scores, so the result from the previous pixel location can be used in the current one. While this simplifies the implementation and greatly reduces computational cost, the size of the aggregation window has a significant impact on the resulting depth map.

We in 2002 proposed a completely different approach [16]. We presented a real-time multi-baseline system that takes advantage of commodity graphics hardware. The system was mostly aimed at novel view generation, but could also return depth values. We used the programmability of modern graphics hardware to accelerate the computation. But at that time it was limited to use a 1×1 correlation window, so that multiple images had to be used to disambiguate matching and achieve reliable results.

In this paper we extend our original method. Aimed to overcome the limitation of the support size, we propose to use a pyramid-shaped correlation kernel that strikes a balance between large windows (more system errors) and small windows (more ambiguities), and can very efficiently be evaluated on graphics hardware. The extended method can still be implemented on commodity graphics hardware and works well with two cameras in a general configuration, without the need for rectification.

3 Method

Comparing images To efficiently compute dense correspondence maps between two images using graphics hardware a plane-sweep algorithm can be used [16, 1]. Given a plane in space, it is possible to project both images onto it using projective texture mapping. If the plane is located at the same depth as the recorded scene, pixels from both images should be consistent. This can be verified by evaluating the square difference between the pixel intensities:

$$(I'_{x,y} - I_{x,y})^2, \tag{1}$$

where $I'_{x,y}$ and $I_{x,y}$ denote pixels in two images. To estimate a dense set of correspondences, a plane hypothesis is set up for every possible disparity value. In the standard stereo case consisting of two fronto parallel cameras (and the optical axis aligned with the Z-axis), the planes should be placed at $Z = -\frac{fb}{d}$, with d ranging over the desired disparity range, f the focal length measured in pixels and b the baseline.

Note that if the images are in a more general configuration the same strategy can still be followed. It is interesting to note that a priori rectification is not necessary since this is effectively provided in the projective texture mapping step. An important advantage of our strategy compared to rectification is that configuration with the epipole contained in the image can also be addressed. This is due to the fact that the projective texture mapping operation is carried out independently for every depth hypothesis[1].

As typical for real-time approaches we use a Winner-Takes-All strategy to select the best match along a ray in space. This ray can correspond to the line of sight of a pixel in one of the two images (in which case our approach is equivalent to the traditional left-to-right or right-to-left stereo matching) or be orthogonal to the family of planes used as hypotheses (which is more typical for plane-sweep algorithms, especially when more than two view are considered). In this paper we will computed correspondences for pixels of one of the two images.

Hardware-based SSD aggregation Although the simple consistency measure might be sufficient in multi-view approaches [9, 16], it is necessary to use larger support region in the two-view stereo case. Therefore, one of the typical dissimilarity measures used in real-time stereo is the Sum-of-Square-Differences (SSD):

$$SSD(x,y) = \sum_{p=-n/2}^{n/2} \sum_{q=-n/2}^{n/2} (I'_{x+p,y+q} - I_{x+p,y+q})^2, \tag{2}$$

where n is the support size. This type of measured can be evaluated very efficiently on a CPU. By reusing data from previous pixels, the algorithm becomes independent of the window size. However, on todays graphics hardware it is not so simple to efficiently implement this type of optimization. On the other hand todays GPUs have build in box-filters to efficiently generate all the mipmap levels needed for texturing [14]. Starting from a base image J^0 the following filter is recursively applied:

$$J^{i+1}_{u,v} = \frac{1}{4} \sum_{q=2v}^{2v+1} \sum_{p=2u}^{2u+1} J^i_{p,q},$$

[1] For standard rectification the two images are also warped to one of the depth hypotheses, but then it is assumed that other hypothesis can be generated by a simple horizontal shift of the image.

where (u, v) and (p, q) are pixel coordinates. Therefore, it is very efficient to sum values over $2^n \times 2^n$ windows. Note that at each iteration of the filter the image size is divided by two. Therefore, a disadvantage of this approach is that the SSD can only be evaluated exactly at every $2^n \times 2^n$ pixel locations. For other pixels, approximate values can be obtained by interpolation. Note that given the low pass characteristic of the box-filters the error that is induced this way is limited.

There exists another efficient possibility to sum the content of multiple pixels which does not reduce resolution. By enabling bilinear texture interpolation and sampling in the middle of 4 pixels, it is possible to average those pixels. Note that in a single pass only a summation over a 2×2 window can be achieved. Both approaches can be combined if desired.

Multi-resolution approach Choosing the size of the aggregation window is a difficult problem. The probability of a mismatch goes down as the size of the window increases [11]. However, using large windows leads to a loss of accuracy and to the possibility of missing some important image features. This is especially so when large windows are placed over occluding boundaries. This problem is typically dealt with by using a hierarchical approach [2], or by using special approaches to deal with depth discontinuities [4].

Here we will follow a different approach that is better suited to the implementation on a GPU. By observing correlation curves for a variety of images, one can observe that for large windows the curves mostly only have a single strong minimum corresponding in the neighborhood of the true depth, while for small windows often multiple equivalent minima exist. However, for small windows the minima are typically well localized. Therefore, one would like to combine the global characteristics of the large windows with the well-localized minima of the small windows. The simplest way to achieve this in hardware consist of just adding up the different curves. In Figure 1 some example curves are shown for the Tsukuba dataset.

Summing two SSD images obtained for windows differing by only a factor of two (one mipmap-level) is very easy and efficient. It suffices to enable trilinear texture mapping and to enforce the right mipmap-level bias. More SSD images can easily be summed by using multiple texturing units (that can all refer to the same texture data, but using different mipmap-level biases).

In fact, this approach corresponds to using a large window, but with larger weights for pixels closer to the center. An example of a kernel is shown in Figure 2. The peaked region in the middle allows good localization while the broad support region improves robustness.

We will call this approach the Multiple Mip-map Level

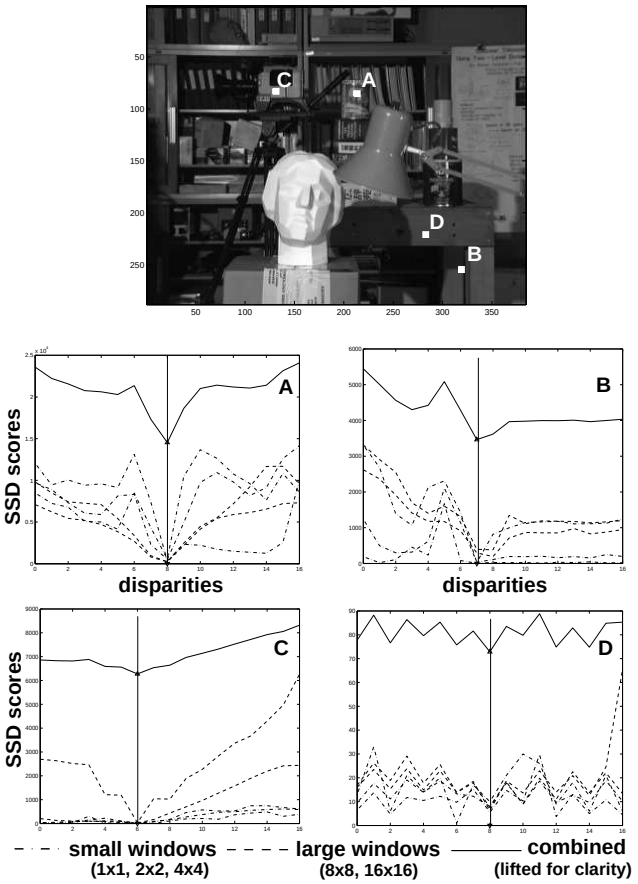

Figure 1: Correlation curves for different points of the Tsukuba stereo pair. Case A represents a typical, well-textured, image point for which SSD would yield correct results for any window size. Case B shows a point close to a discontinuity where SSD with larger windows would fail. Case C and D show low-texture areas where small windows do not capture sufficient information for reliable matching.

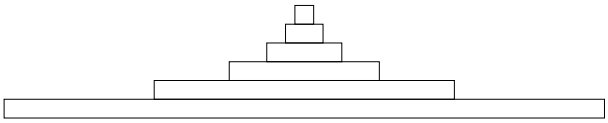

Figure 2: Shape of kernel for 6-level SSD.

(MML) method. In contrast, the approach that only uses one mip-map level will be called the Single Mip-map Level (SML) method.

In addition to these two variations, we also implemented an additional min filtering step. The use of min filter was proposed in [12]. It is equivalent to replace each pixel's disparity with the one from its local neighborhood that has the minimum SSD score in the *final* disparity map. So it is suffice to overlay the final disparity map with different offsets and select the minimum at each pixel location. This can be applied to both the SML method and the MML method, and it incurs little additional computation.

Implementation Details We outline our implementation as follows. There are basically three stages at each step, the

Algorithm 1 Outlines for a real-time implementation

```
for (i = 0; i< steps; i++) {
   // the scoring stage;
   computeSSD();

   // the aggregation stage;
   // MML is the Maxinum Mipmap Level
   if (MML > 0)
       sumAllMipLevels(MML);

  //  the selection stage;
  SelectMinimumSSD();
}
// optional, only needed once
MinFilter();
```

scoring stage to compute the SSD scores, the optional *aggregation* stage to sum up all scores from different mipmap levels, and the final *selection* stage to select the depth with minimum SSD scores. The depth value is encoded in the RGB channel while the SSD score is stored in the alpha channel. An optional min filter can be applied to the final disparity map. Interested readers can refer to the appendix of [16] for implementation details about the scoring and the selection stage. Note that if we only want to use a single mipmap level, which is equivalent to aggregate the SSD scores over a fix-sized support region, we only need reduce the rendered image size proportionally (by changing the viewport setting) in the last selection stage. The automatic mipmap selection mechanism will select the correct mipmap level. Texture shift can also be applied in the last selection stage to increase the effective resolution.

In the aggregation stage, we first have to copy the frame buffer to a texture, with automatic mipmap generation *enabled*. Then we use the multi-texturing functionalities to sum up different mipmap levels. Note that all texture units should bind to the same texture object but with different settings of the mipmap bias. This is possible because the mipmap bias is associated with per texture unit, not per texture object, as defined in the OpenGL 1.4 specification.

In each stage, there is a texture copy operation that copies the frame buffer to the texture memory. We found that texture copies are expensive operations, especially when the automatic mipmap generation is enabled. The frame rate can be doubled if we bypass the second stage. We have tested in a separate program the use of *P-buffer*, an OpenGL extension that allows to render directly to an off-screen texture buffer. We found that no performance gain could be obtained (in fact, performance is even slightly worse in some cases). We suspect that this extension is still a "work in progress" in the current drivers from NVIDIA. We expect to see a dramatic performance boost when this work is done.

Furthermore, the last texture copy for the selection stage can be eliminated with small changes in the graphics hardware. There is the alpha test in the graphics pipeline that only allows to compare the alpha value (which is the SSD score in our case) to a constant. It would be ideal to change the alpha test to compare to the current alpha value in the frame buffer, in a fashion similar to the depth test. We believe these are opportunities for graphics hardware venders to improve their future products.

Though we have not implemented yet, it is also possible to remove lens distortions as an additional texture mapping step. Today's graphics card supports texture indexing through a lookup table, so the lens distortions can be removed precisely. The performance impact is minimal since only two additional passes (one for each image) are required.

4 Results

We have implemented our proposed method in OpenGL and tested on a variety of image pairs. Our first test set is the Tsukuba set that has been widely used in the computer vision literature. The results are shown in Figure 3, in which we show the disparity maps with two variations of our method. For disparity maps on the left column, we used the SML method so that the SSD image was only rendered at a single mipmap level to simulate a fix-sized box filter. Note that texture shift trick effectively doubles the size of filter. So it is equivalent to use a 2×2 kernel at mipmap level zero, and a 4×4 kernel at mipmap level one, etc. For the right column, we used the MML method, i.e. we summed up different mipmap levels and show different disparity maps by changing the only parameter – the maximum mipmap level (abbreviated as MML to abuse the notion) from zero up to six. We can see that the result using a 1×1 kernel is almost meaningless, as in the second row. This would be the result achieved by the method described in [16] with only two input images. If we use a higher mipmap level or in other words increase the MML, results are getting better. But the image resolution drops dramatically with the SML method. The disparity map seems to be the best when $MML = 4$ (i.e. 16×16 kernels). Little was gained when $MML > 4$. Results from another widely used stereo pair using $MML = 4$ are shown in Figure 4.

In term of performance, we tested our implementation on an NVIDIA GeForce4 Card – a card with four multi-texture units. We found virtually no performance difference when MML is set from one to four. This is not surprising since we can use all four texture units to sum up all four levels in a single pass. If MML is set to over four, another

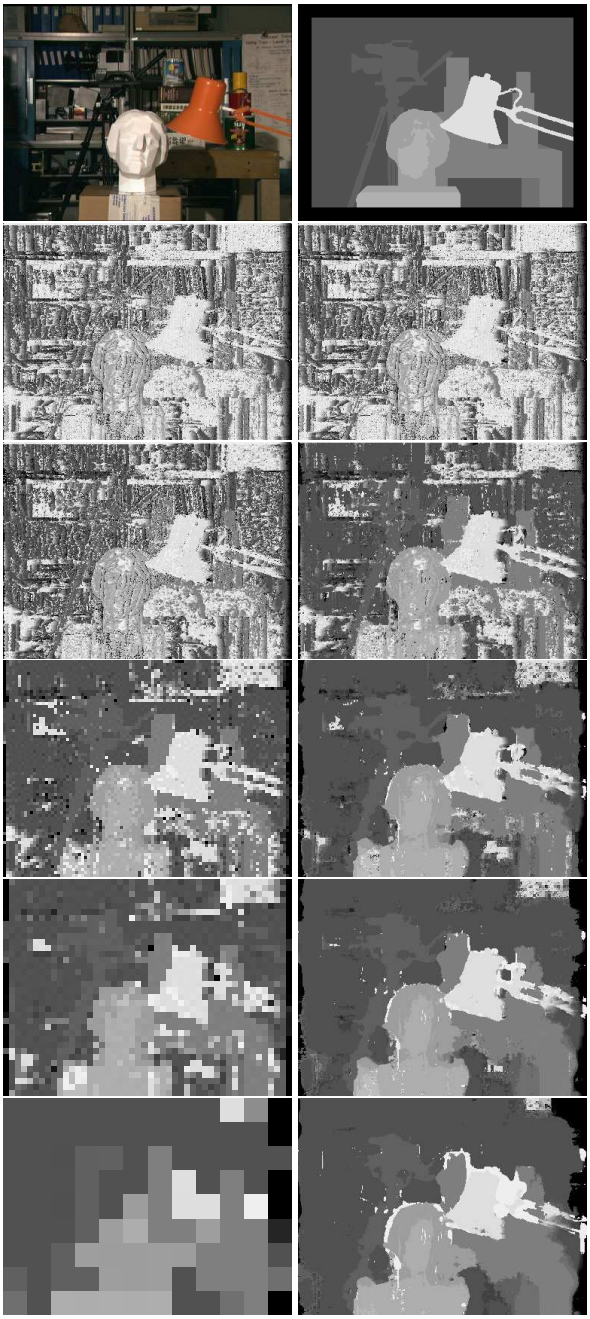

Figure 3: Results on the Tsukuba data set. The second image shows the ground-truth disparity map. For the remaining rows, on the left column we show the disparity map using a single mipmap, while on the right we show the ones with summed scores. The mipmap levels are set to 1, 3, 4, and 6, respectively.

additional rendering pass is required, which results in less than 10% increase in calculation time[2]. In practice, we find

[2]We do not use trilinear interpolation in our performance testing, and it seems that in practice setting MML over four has an detrimental effect on the final result.

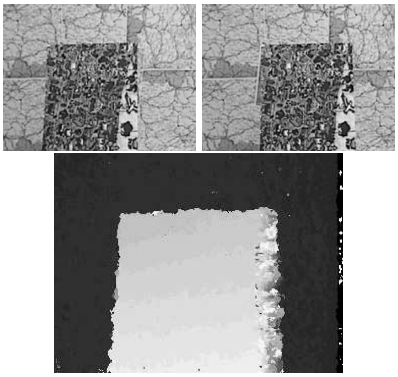

Figure 4: Calculated disparity map from another widely-used stereo pair.

Output Size	Search Range	Times (ms)	Times (Hz)	Img. Update (ms)	Read (ms)	Disp. Calc. (M/sec)
512^2	20	71.4	14	(VGA) 5.8×2	6.0	58.9
	50	182	5.50			65.6
	100	366	2.73			68.3
256^2	20	20.0	50	(QVGA) 1.6×2	1.5	53.1
	50	49.9	20			60.0
	100	99.0	10.1			63.2

Table 1: Performance on an NVIDIA GeForce4 card when summing all mipmap levels. The two input images are 640×480, the maximum mipmap level (MML) is set to 4 in all tests.

that setting MML to four usually strikes a good balance between smoothness and preserving small details. Details of the performance data for the MML method can be found in Table 1. Plotting these data in Figure 5, we can see that our algorithm exhibits very good linear performance with respect to the image size.

We also tested our SML method (results shown in Table 2). In this case the frame-rates are higher, especially when going to a higher mipmap level. Note that for higher mipmap levels the number of evaluated disparities per seconds drop because in this case the output disparity map has a smaller resolution. This method might be preferred for some applications where speed is more important than detail.

We also implemented a real-time system that captures and processes live data online. Our current prototype performs a few additional steps in software, such as radial distortion correction and segmentation[3]. As a proof of concept, these yet-to-be-optimized parts are not fully pipelined with the reconstruction. These overheads slow down the overall reconstruction rate to 6-8 frames per second at 256×256 resolution with 100 depth planes. In Figure 6, we show a sample stereo pair and a reconstructed depth map. In

[3]The cameras are facing a white wall with little texture. So we segment the images to fill the background with different colors.

Base Size	Output Size	Search Range	Times (ms)	Times (Hz)	Overhead (ms)	Disp. Calc. (M/sec)
512^2	128^2 (4×4)	20	2.5	40	12.0	11.7
		50	6.4	15.6		10.7
		100	12.8	7.8		8.86
512^2	256^2 (2×2)	20	28.3	35.3	13.1	31.7
		50	71.4	14.0		38.8
		100	144	6.9		41.7
512^2	512^2	20	40.8	24.5	17.6	89.8
		50	106	9.4		106
		100	207	4.8		117
256^2	128^2 (2×2)	20	12.7	78.7	3.58	20.1
		50	31.6	31.6		23.3
		100	63.1	15.8		24.6
256^2	256^2	20	16.2	61.7	4.7	62.7
		50	40.3	24.8		72.8
		100	80.7	12.4		76.7

Table 2: Performance on an NVIDIA GeForce4 card when using only a single mipmap level with texture shift enabled. Throughput decreases proportionally to the output resolution because the majority of the time is spent on computing the SSD score. The overhead includes both the image update time and the time to read back the depth map from the frame buffer.

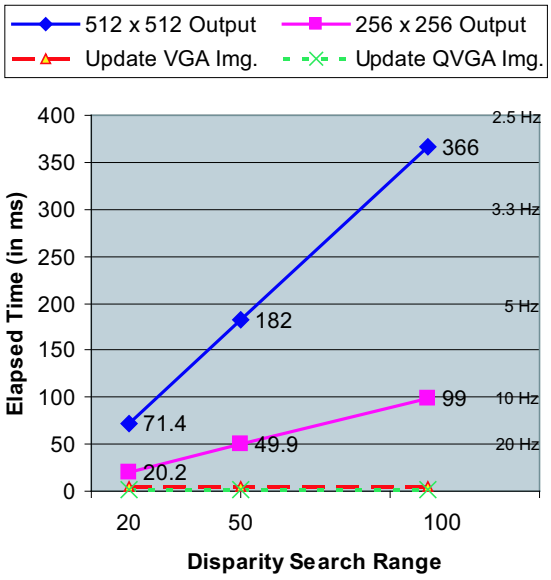

Figure 5: Performance on a NVIDIA GeForce4 Card. The data are from Table 1.

our real-time system, darker colors in the depth map mean that the object is closer to cameras while brighter colors mean further. The variation of disparity is about 20 pixels. We swept over 100 planes to achieve sub-pixel accuracy. To better illustrate our results, we also show the reconstructed 3D point cloud from different perspectives in

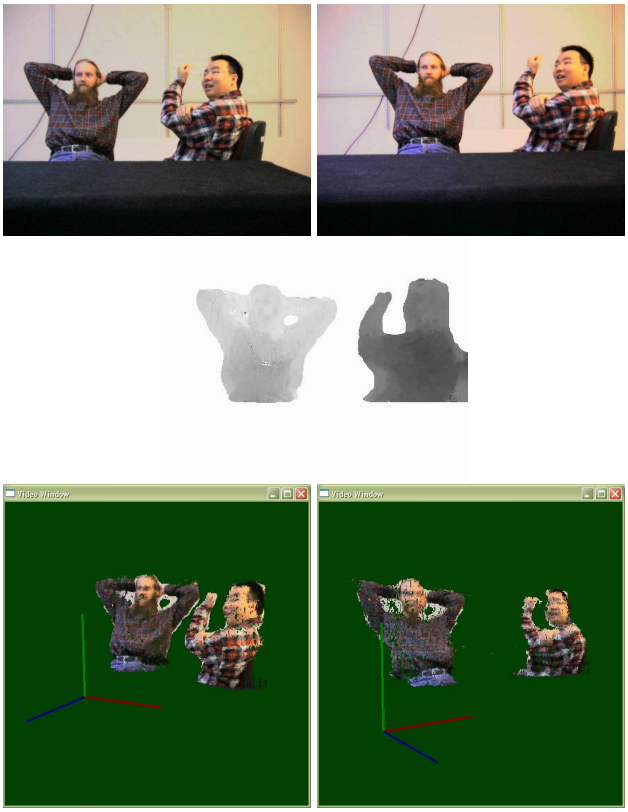

Figure 6: Typical results from our real-time online stereo system. The first row shows the two input images, while the second row shows the disparity map and the reconstructed 3D point cloud from different perspectives. Some holes in the 3D views are caused by the rendering. We simply render fix-size (in screen space) points using the GL_POINT primitive.

Figure 6. More scenes and their depth maps can be found in Figure 7.

5 Conclusion and Future Work

We have presented a stereo algorithm suitable for implementation on commodity graphics hardware. Our algorithm relies on the traditional sum-of-square-differences (SSD) dissimilarity measure between correlation windows. Unlike conventional approaches, we *combine* SSD scores from different resolutions, from coarse to fine. This is, in fact, equivalent to using a *center weighted* filter kernel, which achieves good results close to depth discontinuities as well as on low texture areas, and reduces estimate bias towards frontal planes.

Our algorithm can be efficiently implemented on current commodity graphics hardware. Performance tests have shown that our implementation running an NVIDIA GeForce4 graphics card is equivalent to the fastest commer-

Figure 7: More results from our real-time online stereo system. The first column shows the input images; The second column shows the disparity map.

cial CPU implementation available. We also demonstrated a real-time online prototype. Even with a rather rudimentary serial architecture and un-optimized code in C, we can calculate a 256×256 depth map with a 100 disparity search range at 6 to 8 frames per second.

Looking into the future, we are planning on optimizing our real-time system by also carrying out the radial distortion correction and background segmentation on the graphics hardware. We are also looking at ways to efficiently implement more advanced stereo algorithms on graphics hardware. This work will be eased with newer generations of graphics hardware providing more and more programmability. We are exploring the full potentials of the graphics hardware to make real-time vision faster, better, and cheaper.

References

[1] R. Collins. A Space-Sweep Approach to True Multi-Image Matching. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 358–363, June 1996.

[2] O. Faugeras, B. Hotz, H. Mathieu, T. Viville, Z. Zhang, P. Fua, E. Thron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real time sorrelation-based stereo: Algorithm, amplementations and application. Technical Report 2013, INRIA, August 1993.

[3] Heiko Hirschmuler. Improvements in Real-Time Correlation-Based Stereo Vision. In *Proceedings of IEEE Workshop on Stereo and Multi-Baseline Vision*, pages 141–148, Kauai, Hawaii, December 2001.

[4] H. Hirschmuller, P. Innocent, and J. Garibaldi. Real-Time Correlation-Based Stereo Vision with Reduced Border Errors. *International Journal of Computer Vision*, 47(1-3), April-June 2002.

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

[6] Tyzx Inc. Real-time Stereo Vision for Real-world Object Tracking. White paper, Tyzx Inc, April 2000. www.tyzx.com.

[7] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka. A Stereo Engine for Video-rate Dense Depth Mapping and Its New Applications. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, pages 196–202, June 1996.

[8] K. Konolige. Small Vision Systems: Hardware and Implementation. In *Proceedings of the 8th International Symposium in Robotic Research*, pages 203–212. Springer-Verlag, 1997.

[9] K. Kutulakos and S. M. Seitz. A Theory of Shape by Space Carving. *International Journal of Computer Vision (IJCV),*, 38(3):199–218, 2000.

[10] J. Mulligan, V. Isler, and K. Daniilidis. Trinocular Stereo: A New Algorithm and its Evaluation. *International Journal of Computer Vision (IJCV), Special Issue on Stereo and Multi-baseline Vision*, 47:51–61, 2002.

[11] H. Nishihara. PRISM, a Pratical Real-Time Imaging Stereo Matcher. Technical Report A.I. Memo 780, MIT, 1984.

[12] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, 47(1):7–42, May 2002.

[13] Lance Williams. Pyramidal Parametrics. In *Computer Graphics (SIGGRAPH 1983 Proceedings)*, volume 17, pages 1–11, July 1983.

[14] Mason Woo, Jackie Neider, and Tom Davic. *"OpenGL Programming Guide"*. Addison-Wesley, second edition, 1996.

[15] John Woodfill and Brian Von Herzen. Real-Time Stereo Vision on the PARTS Reconfigurable Computer. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 201–210, Los Alamitos, CA, 1997. IEEE Computer Society Press.

[16] R. Yang, G. Welch, and G. Bisop. Real-Time Consensus-Based Scene Reconstruction Using Commodity Graphics Hardware. In *Proceedings of Pacific Graphics 2002*, pages 225–234, Beijing, China, October 2002.