Stereovision on GPU

Ruigang Yang, Liang Wang
University of Kentucky
{ryang,liang}@cs.uky.edu

Greg Welch, Marc Pollefeys
University of North Carolina at Chapel Hill
{welch,marc}@cs.unc.edu

1. INTRODUCTION

Depth from stereo has traditionally been, and continues to be one of the most actively researched topics in computer vision. Recent development in this area has significantly advanced the state of the art in terms of quality. However, in terms of speed, these best stereo algorithms typically take from several seconds to several minutes to generate a single disparity map, limiting their applications to off-line processing. There are many interesting applications, such as robot navigation and augmented reality, in which high-quality depth information at video rate is crucial.

In this short paper we summarize a number of techniques we have developed over the past few years to improve both the speed and quality of real-time stereo algorithms. Common to these techniques is to map the computation to commodity graphics hardware (i.e., GPUs) to exploit its parallelism and massive bandwidth there. Evaluation using the benchmark Middlebury database shows that our approaches are among the best and fastest stereo algorithms so far.

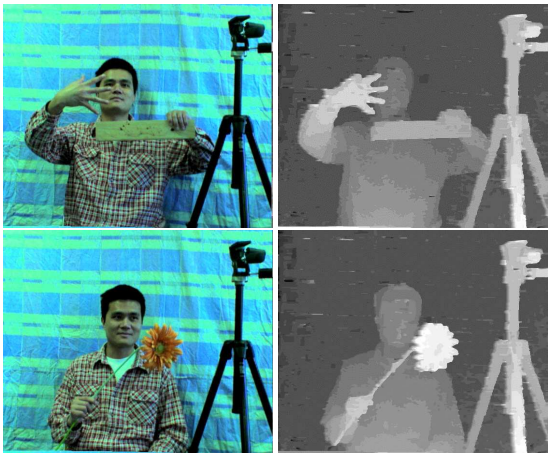

Figure 1: Two sample images and their depth maps from our live system on a 3.0GHz PC with an ATI's Radeon XL1800 graphics card. With this quality, we can achieve 43 fps with 320×240 input images and 16 disparity levels.

2. METHOD

Given a pair of images, the goal of a stereo algorithm is to establish pixel correspondences between the two images. The correspondence can be expressed in general as a disparity vector, i.e., if $P_L(x, y)$ and $P_R(x', y')$ are corresponding pixels in the left and right image respectively, then the disparity of $P_L(x, y)$ and $P_R(x', y')$ is defined as the difference of their image coordinates–$[x - x', y - y']$. Therefore, the output of a stereo algorithm is a disparity map, i.e., a map that records the disparity vector for every pixel in one image (the reference image) – the disparity map for the other image is automatically defined because of the symmetry in disparity vectors.

Our stereo framework contains four major steps: rectification, matching cost computation, cost aggregation, and finally disparity selection. Rectification involves a 2D projective transformation for each image so that the epipolar lines are aligned with scan lines. In the second step, a matching cost for every possible disparity value for each pixel is computed. To reduce the ambiguity in matching, the cost is summed over a small neighboring window (support region) in the third aggregation step. The last disparity selection step picks an "optimal" disparity value for each pixel. In the next few sections, we present several techniques that fit well on the graphics hardware to receive maximum acceleration.

2.1 Rectification

The standard approach to perform image-pair rectification consist of applying 3×3 homographies to the stereo images that will align epipolar lines with corresponding scanlines. It is also a common practice to correct non-linear lens distortions at the same time. A common optimization is to create a look-up table that encodes the per-pixel offset resulting from lens distortion correction and the rectifying homography. The latest generation of graphics hardware supports dependent-texture look-up that makes precise per-pixel correction possible.

2.2 Matching cost computation

A widely used matching cost is the the absolute difference between the left and right pixel intensities:

$$|P_L(x, y) - P_R(x + d, y)| \ . \tag{1}$$

where d is the hypothesized disparity value. For every pixel $P_L(x, y)$ in the reference image, we loop through all disparity hypotheses to calculate their matching costs using equation 2.2. In the end, we obtain a matching cost volume **C** – a three-dimensional array indexed by x, y, and d. The computation can be implemented on GPU using fragment program and texture mappings. Furthermore pixels can be packed to use the vector processing capability of GPU. Details can be found in [5].

2.3 Cost Aggregation

The task of cost aggregation is sum over the per-pixel matching cost over a small window to reduce the ambiguity in matching cost. Toward this end, we have developed three different approaches: *MIPMAP*, *adaptive window*, and *color-weighted*.

MIPMAP Modern GPUs have built-in box-filters to efficiently generate all the mipmap levels needed for texturing. Starting from a base image P^0 the following filter is recursively applied:

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

where (u, v) and (p, q) are pixel coordinates. Therefore, it is very efficient to sum values over $2^n \times 2^n$ windows. The final aggregated matching cost is the sum over differently-sized windows. This approach combines the global characteristics of the large windows with the well-localized minima of the small windows. Details can be found in [4].

Adaptive Window (AW) Another cost aggregation scheme is to use an adaptive window that can be accurately evaluated at every pixel location. Our scheme has six different support windows, each corresponding to a different shape configuration–corner, edge, etc. The one with the minimum score is used as the aggregated matching cost. To efficiently implement this on GPU, we use the hardware support for bilinear texture interpolation. By sampling in the middle of 4 pixels, it is possible to average those pixels. To sum over a large window, we implement a two-pass algorithm. More details can be found in [5].

Color-weighted (CW) We adopted the adaptive weight approach from [6]. The basic idea is to aggregate the matching cost based on both color and geometric proximity. Given a pixel p (we dropped the coordinate indices for simplicity in notation), and a pixel l in its support region, the matching cost from l is weighted by the color difference (Δc_{pl})) between p and l, and the Euclidian distance (Δg_{pl}) between p and l on the image plane. The formulation for the weight $w(p, l)$ is:

$$w(p, l) = \exp\left(-(\frac{\Delta c_{pl}}{\gamma_c} + \frac{\Delta g_{pl}}{\gamma_g})\right), \qquad (2)$$

where γ_c and γ_g are weighting constants.

The aggregated cost is computed as a *weighted* sum of the per-pixel cost. This simple approach has shown to be remarkably effective. However, this aggregation scheme is very expensive in terms of computation. Unlike many schemes that utilize a rectangular window, which can be efficiently computed either incrementally or through separate passes, the weighting mask varies from pixel to pixel because the center pixel has different colors. As reported in [6], it took about one minute to produce a small depth map (i.e., about 0.03 Mde/second), which makes real-time application impossible. We designed two approximation schemes on GPU to speed up the process over two orders of magnitude. More details can be found in [2].

2.4 Disparity Selection

Given the real-time constraint, there are only two options for disparity selection: "winner-take-all" (WTA) that assigns each pixel to the disparity value with the minimum cost, or to use Dynamic Programming (DP) that optimizes the results on a scanline by scanline basis. WTA is very simple from a computational standpoint but the result is more sensitive to image noise and calibration errors. DP offers horizontally optimized result but the inter-scanline consistency is not enforced. Both WTA and DP can be implemented on GPU. We found out that DP does not receive significant speed up in GPUs, probably due to its complex branching and looping structure.

3. RESULTS AND CONCLUSION

We evaluated our algorithm using the Middlebury data set [1]. The color-weight-based aggregation combined with dynamic programming yields the best results, as shown in Figure 2. In terms of speed, as shown in Table 1, MIPMAP+WTA is the fastest, achieving close to 1000 millions of disparity evaluations per second (MDE/s)[1]. In contrast, commercial stereo packages can only achieve about 150 MDE/s. Some video results can be found in [3].

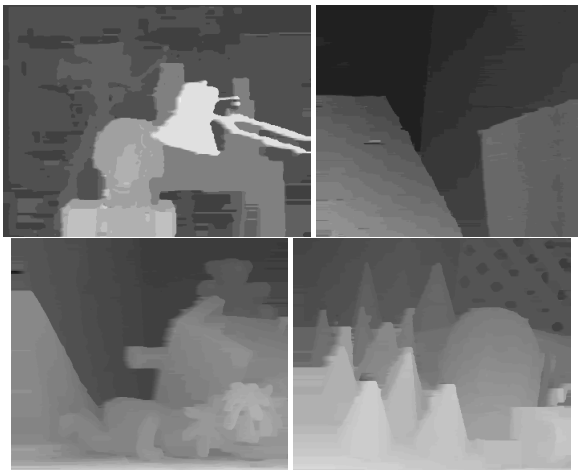

Figure 2: Resulting depth maps using color-weight+DP.

	MIPMAP+WTA	AW+WTA	CW+WTA	CW+DP
MDE/s	980	560	47	53

Table 1: Speed of Different Algorithms.The test system is a 3.0Ghz PC with a Radeon XL1800 graphics card from ATI.

We already started to further improve the quality and speed, including global-optimization based approaches and exploring temporal consistency. We believe that more flexible writing operations in commodity hardware can further allow more complex algorithms to exploit their full potentials.

4. REFERENCES

[1] D. Scharstein and R. Szeliski. www.middlebury.edu/stereo.

[2] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister. High-quality Real-time Stereo using Adaptive Cost Aggregation and Dynamic Programming. In *Proceedings of 3DPVT*, 2006.

[3] L. Wang, M. Liao, and R. Yang. `http://www.vis.uky.edu/liaomiao/GPUstereo.htm`.

[4] R. Yang and M. Pollefeys. Multi-Resolution Real-Time Stereo on Commodity Graphics Hardware. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 211–218, 2003.

[5] R. Yang, M. Pollefeys, and S. Li. Improved Real-Time Stereo on Commodity Graphics Hardware. In *IEEE Workshop on Real-time 3D Sensors and Their Use (in conjunction with CVPR'04)*, 2004.

[6] K.-J. Yoon and I.-S. Kweon. Locally Adaptive Support-Weight Approach for Visual Correspondence Search. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 924–931, 2005.

[1]The number of disparity evaluations per second corresponds to the product of the number of pixels times the disparity range times the obtained frame-rate and therefore captures the performance of a stereo algorithm in a single number.