REAL TIME STEREO VISION USING EXPONENTIAL STEP COST AGGREGATION ON GPU

Wei Yu[1], *Tsuhan Chen*[2], *James C. Hoe*[1]

[1]Carnegie Mellon University, [2] School of Electrical Computer and Engineering, Cornell University

ABSTRACT

In this paper, we propose a local cost aggregation approach for real time stereo vision on a graphics processing unit (GPU). Recent research shows that local approaches based on carefully designed cost aggregation strategies can outperform many global approaches. Among those local aggregation approaches, adaptive-weight window produces the best quality disparity map under real-time constraint, but it is slower than other local approaches. We propose a very fast adaptive-weight aggregation method based on exponential step information propagation. The basic idea is to propagate information from long distance pixels within a few iterations. We also discuss important techniques of efficient implementation on GPU platform, which result in 10.5x speed up than a straightforward implementation. Compared to existing real time adaptive-weight approach, our technique reduces the computation time by more than half at improved accuracy. Detailed experimental results show that our technique is Pareto-optimal among existing real time or near real time stereo algorithms in the accuracy-speed trade-off space.

Index Terms— real time stereo, GPU, adaptive-weight

1. INTRODUCTION

The goal of stereo vision is to reconstruct disparity map from a stereo image pair. A taxonomy and evaluation system proposed by [1] is now widely used in evaluating the disparity accuracy of stereo matching algorithms using a set of benchmark datasets. As [1] suggests, most stereo algorithms consists of four steps: (1) cost initialization (matching cost of each pixel at different disparity hypothesis) (2) cost aggregation (aggregate initial costs over spatial support) (3) disparity optimization (decide the disparity level that optimizes a target cost function) (4) disparity refinement (post-processing to remove mismatch).

Most existing stereo algorithms can be classified into two categories: local and global approaches. Their major differences lie in step (3). Local algorithms use Winner-Take-All (WTA) strategy, simply taking the disparity level that minimizes the aggregation cost. Global algorithms optimize more complicated target functions that consider interactions among pixels, and thus need global optimization techniques like belief propagation or graph cut to compute the optimal solution. Generally speaking, local algorithms are computationally less expensive, and global algorithms produce higher quality disparity map. The top performing stereo algorithms in terms of accuracy are all global methods [2]. However, those algorithms typically take several to tens of minutes to compute a disparity map of image size 288×384 with 16 dispairty levels, which are not suitable for real-time applications.

Most real time stereo systems use local approaches on graphics processing unit (GPU). [3] presents interesting accuracy-speed trade-off of six recent cost aggregation approaches with WTA optimization under the real time constraint on an ATI Radeon X800 graphics card. Their experiments show that a modified version of adaptive-weight window approach proposed in [4] performs the best in terms of accuracy, but it's relatively slower than other local methods. The adaptive-weight approach proposed in [4] demonstrates high accuracy, but it's too expensive in computation to run in real time. [3] simplifies the method in [4] for a better trade-off between accuracy and speed.

There are several papers about real time stereo using semi-global or global approaches on GPU. [5] proposes a semi-global real time stereo algorithm on ATIs Radeon XL1800 graphics card. It integrated adaptive-weight aggregation along vertical direction with dynamic programming optimization along horizontal scanlines. The disparity accuracy is slightly better than to [3] , and the computation time is about 2x of [3]. [6] proposes a near real time global stereo matching using hierarchical belief propagation. Better disparity accuracy is achieved compared to [5], and computation time is about 2.7x of [5].

[7] proposes a segmentation-based cost aggregation strategy that can achieve near real time processing speed on Intel Core Duo 2.14 GHz CPU, with the best accuracy among the state-of-art near real time local approaches on CPU. However, both accuracy and speed in [7] are worse compared to any of the above real time stereo on GPU platforms, showing a certain gap between CPU/GPU processing power for stereo vision.

Table 1 summarize all exisitng real time or near real time stereo. We are inspired by the real time adaptive-weight approach in [3]. The adaptive width window approach exhibits high data parallelism, which is very suitable to be accelerated

Table 1. Summarize real time or near real time stereo (quantitative results are given later in Table 3)

GPU	local[3]	very fast, good accuracy
	semi-global[5]	slower and better accuracy than local
	global[6]	slower and better accuracy than semi-global
	local(proposed)	faster than[3, 5] at comparable accuracy
CPU	local[7]	slower and worse accuracy than any GPU method

on GPU. Though graphics hardware has many more processing units than general purpose computing hardware, writing fast code to maximize the GPU hardware utilization is not trivial. Our goal is to improve the Pareto efficiency of GPU based real time stereo. That is, to achieve the same accuracy with less computing time, or higher accuracy with the same computing time.

In the following, section 2 explains the proposed algorithm and GPU implementation techniques. Section 3 compares our technique with existing real time or near real time approaches in terms of both accuracy and speed. Section 4 concludes the paper.

2. PROPOSED ALGORITHM AND GPU ACCELARATION

2.1. Algorithm Description

Original adaptive-weight in [4]. The basic idea of the adaptive-weight approach is to adjust the support-weight of each pixel in a given support window. The weight of each pixel is computed based on color dissimilarity and geometric relationship with the pixel under consideration. The advantage of the adaptive-weight approach is that it can preserve arbitrarily shaped depth discontinuities using a fixed window size [4]. The adaptive weight of pixel q in the support window of pixel p is computed as following :

$$w(p,q) = k \cdot f_s(\Delta c_{pq}) \cdot f_p(\Delta g_{pq}) \quad (1)$$

$$\Delta c_{pq} = \sqrt{(L_p - L_q)^2 + (a_p - a_q)^2 + (b_p - b_q)^2} \quad (2)$$

$$\Delta g_{pq} = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2} \quad (3)$$

$$f_s(\Delta c_{pq}) = \exp(-\frac{\Delta c_{pq}}{\gamma_c}), f_p(\Delta g_{pq}) = \exp(-\frac{\Delta g_{pq}}{\gamma_p}) \quad (4)$$

Δc_{pq} measures color similarity of two pixels, and Δg_{pq} measures Euclidean distance of two pixels. The more similar in color, the closer in distance, the higher the assigned weight. The aggregated cost of pixel p at disparity level d is computed using:

$$C(p,d) = \frac{\sum_{q \in N_p} w(p,q) w(\overline{p_d}, \overline{q_d}) C_0(q,d)}{\sum_{q \in N_p} w(p,q) w(\overline{p_d}, \overline{q_d})} \quad (5)$$

$\overline{p_d}$ and $\overline{q_d}$ are corresponding pixels of p and q in the target image at assumed disparity level d. $C_0(q,d)$ is the initial matching cost of pixel q at assumed disparity level d. According to [4], considering weights in both reference and target images can help preserving depth discontinuities in both views. The computational cost of the algorithm is very high, taking about one minute to generate a disparity map [4].

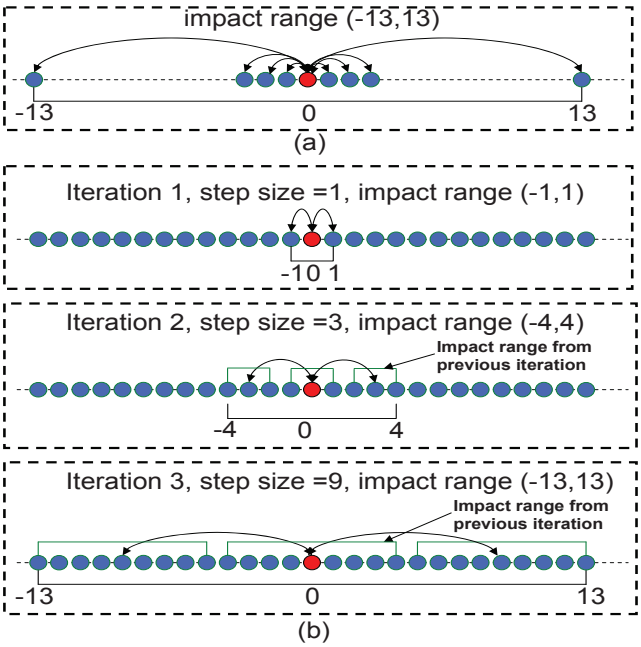

Fig. 1. A simple example of cost aggregation: (a) aggregation used in [3] (b) exponential step information propagation

Real time adaptive-weight in [3]. To achieve real time implementation on GPU, two simplifications are proposed in [3]. First, only weights in the reference view is used in cost aggregation. Second, instead of using a fixed window of size $N \times N$, [3] use a two pass approach - first pass aggregates cost along horizontal scanline, followed by a pass aggregating cost along vertical scanline. This reduces the arithmetic complexity from $O(N^2)$ to $O(N)$. Though the accuracy of the simplified adaptive-weight in [3] is not so good as in [4], it already tops the accuracy among all local approaches discussed in [3]. It's slower than some other local methods in [3], taking about 64 ms to process Teddy and Cones dataset.

Proposed ESAW It takes $O(N)$ computations to aggregate costs of all pixels within $N/2$ offset to the center pixel along 1-D scanline in [3]. Fig 1(a) shows a simple example of aggregating pixels within range (-13,13). Using the approach in [3] needs computation on 27 pixels. Fig 1(b) shows another way of aggregating cost in 3 iterations with much less computation. In each iteration, every pixel aggregates costs of three pixels, itself and pixels at $-os$ and $+os$ offset. The "impact range" is defined as the furthest pixel offset where the pixel matching cost is aggregated into the center pixel. $os = 1, 3, 9$ for three iterations. After each iteration, the im-

pact range grows, from (-1,1) to (-4,4) to(-13,13). In this way, propagating matching cost within 13 pixels' offset to the center pixel just need computation on $3 \times 3 = 9$ pixels.

Putting the toy example more formally, if at iteration t, the impact range of previous iteration is $(-r(t-1), r(t-1))$, the maximum step size at iteration t to avoid "holes" in aggregation (missing costs of some pixels in the impact range) is $s(t) = 2r(t-1)+1$. And with this step size, the impact range can be enlarged to $(-r(t), r(t))$, where $r(t) = 3r(t-1)+1$. It can be derived that starting from $r(0) = 0$, the maximum step size at iteration t should be $s(t) = 3^{t-1}$, the impact range after iteration t is $(-r(t), r(t))$, where $r(t) = (3^t - 1)/2$. Therefore by propagation using exponential step size, aggregating cost of N pixels need only $O(logN)$ computations. Generalizing the above idea, we design the exponential step adaptive weight (ESAW) cost aggregation scheme:

---

for $t = 1$:max_iteration

1. Compute offset:

$$os = \text{round}(base^{t-1}); \tag{6}$$

2. Aggregate cost horizontally of center pixel p at (x, y), p_l at $(x - os, y)$ and p_r at $(x + os, y)$:

$$C^h(p) = \sum_{q \in \{p_l, p, p_r\}} w(q, p) C^{(t-1)}(q); \tag{7}$$

3. Aggregate cost vertically of center pixel p at (x, y), p_u at $(x, y - os)$ and p_d at $(x, y + os)$:

$$C^t(p) = \sum_{q \in \{p_u, p, p_d\}} w(q, p) C^h(q); \tag{8}$$

note:$C^t(p)$ is the aggregated cost of pixel p after iteration t. $C^h(p)$ is the intermediate horizontally aggregated cost.

end

Fig 2(a) shows average percent of bad pixels (the same as the last column "Average percent of bad pixels" in Middlebury stereo evaluation online system [2]) of all four benchmark datasets, for varying number of iterations (3 to 10)and base (1.5 to 3). It's clear that increasing the number of iterations helps improving the overall accuracy, though improvement gets very marginal after iteration 6. The processing time of ESAW is proportional to the number of iterations, which is plotted in Fig 2(b). Since images in different datasets are of different sizes and disparity levels, usually MDS (Millions of Disparity per Second) is used as a normalized measurement of computational efficacy. MDS is computed as $\frac{(\text{image size}) \times (\text{disparity levels})}{\text{processing time}}$. Table 2 summarizes the base giving the best accuracy for each iteration number, and corresponding accuracy and MDS.

2.2. GPU acceleration

The GPU we used for this study is NVIDIA GeForce 8800GTX (abbreviated as G80). The G80 contains a total of 128 pro-

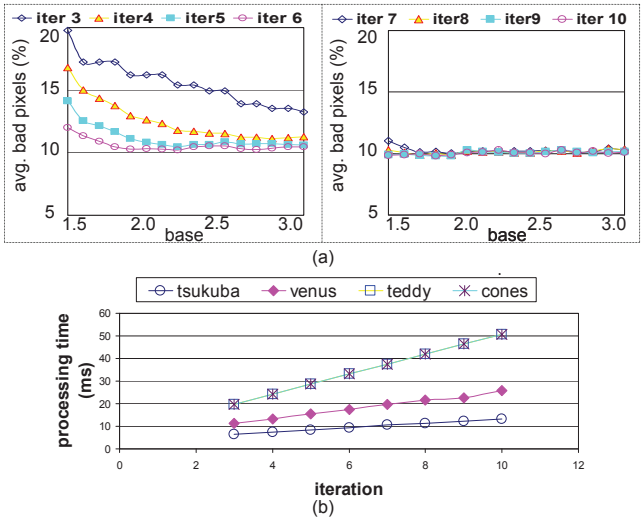

Fig. 2. Accuracy and speed for varying base and iterations.

Table 2. Optimal base for varying number of iterations.

# of iterations	3	4	5	6	7	8	9	10
Optimal base	3	2.8	2.2	2.2	1.9	1.9	1.9	1.8
Avg. % bad pixels	13.24	11.12	10.48	10.22	10.04	9.92	9.8	9.83
Avg. MDS	309.6	256.2	219.2	191.2	170.1	153.3	140.5	127.7

grammable processing units organized into 16 stream multiprocessors(SM). Each SM includes 8 stream processors executing SPMD (single program multiple data) program in lockstep. The architecture of GPU makes it appropriate for many image processing applications where a large number of pixels undergo similar computations independently.

In our application, proper memory management is crucial to high efficiency in hardware utilization. On G80, off-chip global and texture memory can be used to store large chunks of data. The aggregated cost of each iteration needs to be stored in the off-chip memory. The advantage of texture memory is that it supports clamp addressing mode, i.e., when coordinates go beyond the image size, texture reference automatically clamps the address. Using global memory, on the other hand, needs to deal with out-of-boundary address in the code explicitly. The downside of texture memory is that it's only readable to the GPU device. Writing to texture memory involves either interacting with host CPU which is very expensive, or using memory copy instruction on GPU device. However, tested memory copy time for the aggregated cost after each iteration takes about 7ms, while computing time takes less than 5ms. Choosing global memory to store the aggregated costs results in about 2.3x speed up.

Another important issue is the access pattern to global memory . Accessing off-chip memory involves long latency, thus must be compensated by high throughput. On G80, coa-

Table 3. Accuracy-speed comparison of real time or near real time stereo (error rate is measured by percentage of error pixels).

			tsukuba			venus			teddy			cones			Err	MDS
			vis	all	disc	vis	all	disc	vis	all	disc	vis	all	disc		
GPU	local	ESAW(iter 5)	1.8	3.3	8.7	2.9	4.0	20.0	10.1	16.9	21.2	6.8	14.3	15.9	10.5	219.2
		ESAW(iter 9)	2.0	2.8	9.9	2.5	3.5	15.7	10.0	16.5	21.3	6.0	13.0	14.4	9.8	140.5
		AW[3]	2.3	3.6	11.2	3.6	4.6	19.8	10.9	18.8	23.2	5.9	14.3	13.8	11.0	104.1
	semi-global	DP[5]	2.1	4.2	10.6	1.9	3.0	20.3	7.2	14.4	17.6	6.4	13.7	16.5	9.8	52.8
	global	BP[6]	1.5	3.4	7.9	0.8	1.9	9.0	8.7	13.2	17.2	4.6	11.6	12.4	7.7	19.7
CPU	local	aggregate[7]	3.0	4.4	13.2	3.5	4.6	25.5	10.7	17.5	23.4	4.9	12.7	11.3	11.2	18.9

AW is abbreviation for "adaptive weight"; DP is abbreviation for "dynamic programming"; BP is abbreviation for "belief propagation".

lesced memory access is the most efficient way to read/write global memory. Coalesced access means 16 threads issued in the same instruction cycle must access continuous address and starting address must be aligned (multiples of 16). Since the offset value in each iteration may not guarantee to be multiples of 16, this may result violating the starting address alignment requirement. The solution is to use on-chip shared memory. Chunks of pixels are read from global memory into the on-chip shared memory in a coalesced way, further computation only involves accessing shared memory, whose access does not require alignment and is as fast as registers. This technique results in about 4.2x speedup. Using both techniques, the total speed up is about 10.5x. All experimental results reported in the following use both techniques.

3. EXPERIMENTAL RESULTS

We compare the ESAW with all real time local approaches in [3]. Fig 3 plots different approaches in the accuracy-speed space. For fair comparison, we use the same measurements as in [3]:x-axis is the average processing time per disparity evaluation on four datasets. y-axis in Fig 3(a) is the average error rate at non-occluded area, in Fig 3(b) is the average error rate at discontinuous area. Clearly the ESAW is Pareto-optimal among all local approaches in the accuracy-speed space.

Table 3 shows ESAW compared to the state-of-art real time or near real time stereo approaches, including local, semi-global and global approaches on GPU and CPU. The semi-global algorithm [5] is slower than local approaches, and global algorithm [6] is slower than the semi-global one, but the accuracy of the global algorithm is the best. ESAW can achieve comparable accuracy of adaptive-weight [3] at iteration 5 and accuracy of semi-global one [5] at iteration 9, with 2.1x and 2.7x speed up. CPU based real time stereo [7] is far lagged behind in both accuracy and speed.

4. CONCLUSION

In this paper, we propose a novel fast cost aggregation approach ESAW based on exponential step information propagation . The basic idea is to propagate information from long distance pixels to the center pixel within a few iterations. Experiments show ESAW is Pareto optimal in the accuracy-speed space compared to the state-of-art real time or near real time approaches.

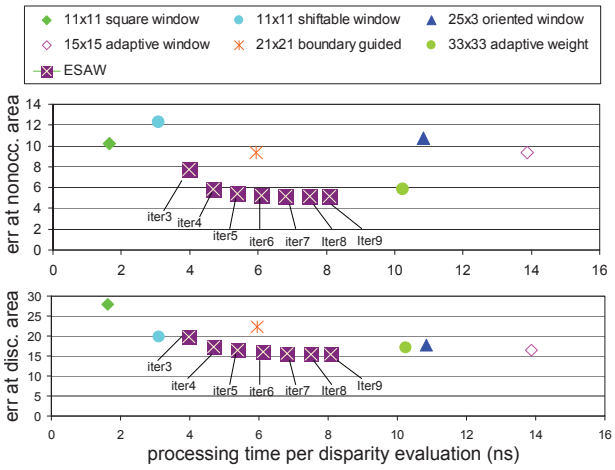

Fig. 3. Comparison of ESAW with real time local approaches in [3] at non-occluded and discontinuous area.

5. REFERENCES

[1] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," in *IJCV*, 2002.

[2] "Middlebury stereo evaluation online system," `http://vision.middlebury.edu/stereo/eval/`.

[3] Minglun Gong, Ruigang Yang, Liang Wang, and Mingwei Gong, "A performance study on different cost aggregation approaches used in real-time stereo matching," in *IJCV*, 2007.

[4] K. J. Yoon and I. S. Kweon, "Locally adaptive support-weight approach for visual correspondence search," in *CVPR*, 2005.

[5] Liang Wang, Miao Liao, Minglun Gong, Ruigang Yang, and David Nistr, "High-quality real-time stereo using adaptive cost aggregation and dynamic programming," in *3DPTV*, 2006.

[6] Qingxiong Yang, Liang Wang, Ruigang Yang, Shengnan Wang, Miao Liao, and David Nistr, "Real-time global stereo matching using hierarchical belief propagation," in *BMVC*, 2006.

[7] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Near real-time stereo based on effective cost aggregation," in *ICPR*, 2008.