Parallel Stereo Matching Based on Edge-Aware Filter

Fan Bu(✉) and Chunxiao Fan

Beijing Key Laboratory of Work Safety Intelligent Monitoring,
School of Electronic Engineering, Beijing University of Posts and Telecommunications,
Beijing 100876, People's Republic of China
stbbff99@gmail.com, fcxg100@163.com

Abstract. This paper presents a novel parallel stereo matching algorithm based on edge-aware filter with good performance in accuracy and speed. The initial matching cost is built with census transform and sobel operator. Then the aggregated cost is computed by rolling guidance filter and guided filter. The final disparity is computed by rolling guidance filter and weighted median filter. The key idea is to eliminate the influence of small scale structures when computing weights in aggregation step and post-processing step. The proposed method ranks 17th on Middlebury benchmark and the results cost 52.5ms on one GPU and 33.8ms on two GPUs.

Keywords: Stereo matching · Edge-aware filter · Cost aggregation · Disparity

1 Introduction

Stereo matching seeks the correspondence in stereo images. The method can be classified into two categories: local methods and global methods [1]. Global methods regard the problem as energy function and minimize the function with belief propagation, graph cuts and so on. They can produce accurate results but they consume too much time. Local methods compute disparity within a support region. The traditional representative local methods are AdaptWeight [2, 3], GeoSup [4] and so on.

Traditional local methods are not well paralleled. So they can't achieve real-time performance on GPU. Nowadays, with the development of edge-aware filter, some local methods utilize edge-aware filter to do cost aggregation step. In article [5, 8-10], joint bilateral filter, guided filter and domain-transform filter are used to compute the weights in cost aggregation step. These methods achieve good performance in both accuracy and speed.

But, there exists a serious issue in the above methods. When computing weights in the aggregation step, some small scale structures will influence the weights' distribution. Zhang proposed rolling guidance filter [11] in 2014. This filter has an edge-aware property while removing small scale structures. The filter is simple in implementation and consumes little time. It gives us some clues to solve the issue.

In this paper, we propose a novel stereo matching algorithm based on edge-aware filter. We adopt a Census-Sobel measure to compute initial matching cost. Then we use rolling guidance filter and guided filter to do cost aggregation. Finally we utilize

H. Liu et al. (Eds.): ICIRA 2015, Part III, LNAI 9246, pp. 310–321, 2015.
DOI: 10.1007/978-3-319-22873-0_27

rolling guidance filter and weighted median filter to get the disparity. Our method is well-paralleled and we give an efficient implementation on CUDA. The results on Middlebury benchmark demonstrate the great performance in accuracy and speed.

The rest of the paper is organized as follows: Section 2 provides a summary of rolling guidance filter. Section 3 gives a full description of our algorithm. Section 4 gives the key of the implementation on CUDA. Section 5 shows the experimental results. Section 6 gives the conclusion.

2 Rolling Guidance Filter

Fig. 1 illustrates the work flow of rolling guidance filter [11]. This filter has two steps. Step 1 uses Gaussian filter to remove small scale structures. Step 2 uses edge-aware filter to do edge recovery, such as joint bilateral filter, guided filter and so on. Step 2 is an iterative step. The final J^n is the output image. Here we take guided filter for example. The equations are listed in Eq. (1-4). Step 1 uses Eq. (1-2) and step 2 uses Eq. (3-4). Guided filter takes one-channel image as input. But in Fig. 1, the input image I can be a 3-channel image (RGB). The solution is to handle each channel separately. Then we get three 1-channel images: J_R^n, J_G^n and J_B^n. Finally we combine the three 1-channel images to a 3-channel image. The detailed information about rolling guidance filter and guided filter can be found in [6, 7, 11].

$$J^1(p) = \frac{1}{K_p} \sum_{q \in N(p)} \exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2} \right) I(q) \tag{1}$$

$$K_p = \sum_{q \in N(p)} \exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2} \right) \tag{2}$$

$$J^{n+1}(p) = \sum_q W_{pq}(J^n) I(q) \tag{3}$$

$$W_{pq}(J^n) = \frac{1}{|w|^2} \sum_{k:(p,q) \in w_k} \left(1 + \frac{(J_p^n - \mu_k)(J_q^n - \mu_k)}{\sigma_k^2 + \varepsilon} \right) \tag{4}$$

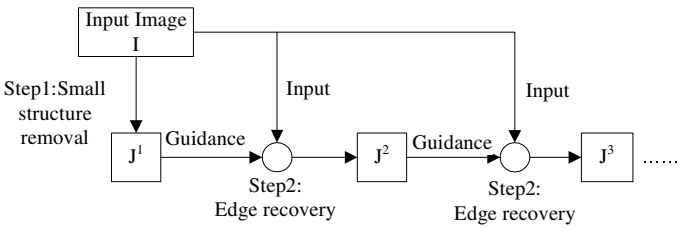


Fig. 1. Work flow of rolling guidance filter

Fig. 2 gives an example of rolling guidance filter. Figure (a) shows the original image while figure (b) shows the result image after rolling guidance filter. The small scale structures like small dots are well removed. And the edges of large scale structures like number 8 and number 2 are well preserved.

(a) (b)

Fig. 2. (a) Input image. (b) Output image after rolling guidance filter.

3 Stereo Matching

Fig. 3 illustrates the work flow of our method. Like most local methods, our method is organized by four parts: matching cost computation, cost aggregation, disparity selection and post-processing [1]. Here we do rolling guidance filter on original stereo images and utilize the filtered images to do cost aggregation and post-processing.

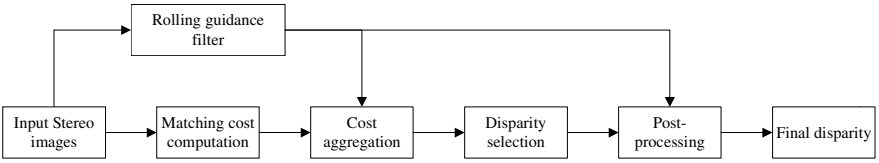


Fig. 3. Work flow of stereo matching

3.1 Matching Cost Computation

Rhemann et al. introduced a simple but effective measure in [8]. The cost is combined of differences of color and gradient, and we name it ad-gradx. It is formally expressed in Eq. (5). $C(p,d)$ is the matching cost. $\nabla_x(I_p)$ in Eq. (5) represents the x-direction gradient of pixel p in image I and the gradient is computed as Eq. (6).

$$C(p,d)=(1-\alpha)\cdot\min\left[\left\|I_p^L-I_q^R\right\|,\tau_1\right]+\alpha\cdot\min\left[\left\|\nabla_x I_p^L-\nabla_x I_q^R\right\|,\tau_2\right] \tag{5}$$

$$\nabla_x I_p=\left(I_{p+(1,0)}-I_{p-(1,0)}\right)/2 \tag{6}$$

Fig. 4 shows the disparity from Teddy image. From figure (d) and figure (e), we can see that the ad-gradx measure performs bad in large textureless regions. To solve this problem, we adopt a new cost measure named Census-Sobel described in Eq. (7-10). Here, $census(p)$ is the census transform of pixel p. $sobelx(p)$ is the

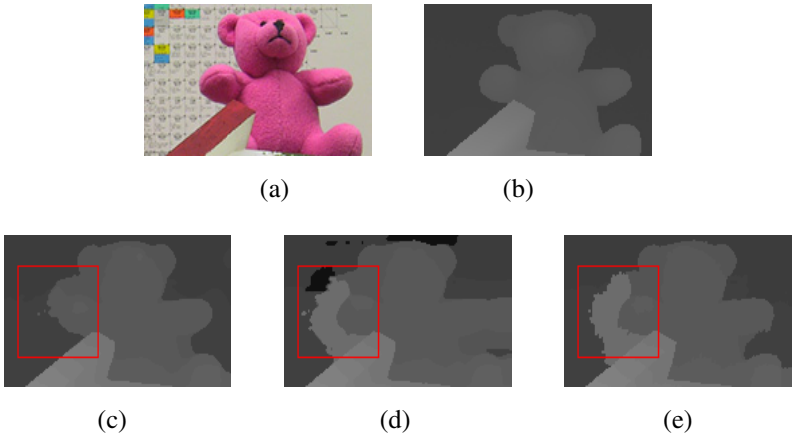
x-direction gradient of pixel p computed by sobel operator and $sobely(p)$ is the y-direction gradient. Pixel q is the corresponding pixel in the right image when the disparity of pixel p is d. *Ham* means hamming distance. γ_{census}, γ_{sobelx} and γ_{sobely} are parameters. Figure (c) proves the effectiveness of our Census-Sobel measure.

$$C(p,d) = C_{census}(p,d) + C_{sobelx}(p,d) + C_{sobely}(p,d) \tag{7}$$

$$C_{census}(p,d) = 1.0 - \exp\left(-\frac{Ham\left[census(p), census(q)\right]}{\gamma_{census}}\right) \tag{8}$$

$$C_{sobelx}(p,d) = 1.0 - \exp\left(-\frac{\left\|sobelx(p) - sobelx(q)\right\|}{\gamma_{sobelx}}\right) \tag{9}$$

$$C_{sobely}(p,d) = 1.0 - \exp\left(-\frac{\left\|sobely(p) - sobely(q)\right\|}{\gamma_{sobely}}\right) \tag{10}$$



(a) (b) (c) (d) (e)

Fig. 4. (a) A part of Teddy image. (b) Groundtruth. (c) Our disparity with census-sobel cost. (d) Our disparity with ad-gradx cost. (e) Disparity in article [8], using ad-gradx cost.

3.2 Cost Aggregation

Rhemann et al. utilized guided filter to do cost aggregation [8] in Eq. (11-12). Here $C_2(p,d)$ is the aggregated cost of pixel p at disparity d. $W_{pq}(I)$ is the weight of pixel p and q in image I. I is the left image in the stereo images and $C(p,d)$ is the initial matching cost in Eq. (7). Details of Eq. (12) are included in article [6-9].

$$C_2(p,d) = \sum_q W_{pq}(I) C(p,d) \tag{11}$$

$$W_{pq}(I)=\frac{1}{|w|^2}\sum_{k:(p,q)\in w_k}\left(1+\left(I_p-\mu_k\right)^T\left(\Sigma_k+\varepsilon U\right)^{-1}\left(I_q-\mu_k\right)\right) \tag{12}$$

When pixel p and q locate on the same side of the edge, $W_{pq}(I)$ computed by Eq. (12) is larger than the case locating on the opposite side of the edge. Locating on the same side means that pixel p and q have more opportunity to locate on the same disparity plane while locating on the opposited side of the edge means that pixel p and q have less opportunity. So the aggregation way in Eq. (11-12) is effective.

However, small scale structures may influence the weights' distribution. In Fig. 5-(a), all the pixels in the red square centered at pixel B locate on the same disparity plane so the weights in the red square should be almost the same. But in fact, pixel B is surrounded by some different color pixels and these pixels are small scale structures in figure (g). Here small scale structures mean they only occupy few pixels. Figure (i) shows the effects of these structures, the weights' distribution is disordered. Figure (h) shows the zoom of pixel B after rolling guidance filter. The small scale structures are removed while the weights are computed correctly in figure (j). Meanwhile, rolling guidance filter doesn't destroy the edges of large scale structures, as shown in figure (c) and (d). In figure (e) and (f), the weights of pixel A is almost the same.

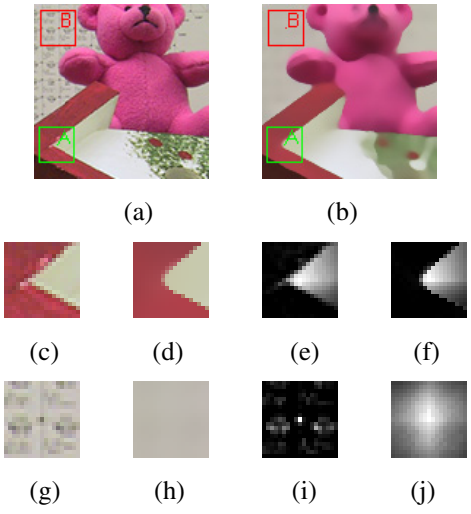


Fig. 5. (a) A part of Image Teddy. (b) Output image of figure a after rolling guidance filter. (c) Zoom of pixel A in figure a. (d) Zoom of pixel A in figure b. (e) Weights of pixel A in figure c. (f) Weights of pixel A in figure d. (g) Zoom of pixel B in figure a. (h) Zoom of pixel B in figure b. (i) Weights of pixel B in figure g. (j) Weights of pixel B in figure h.

In conclusion, we take Eq. (12-13) for cost aggregation step. Image J in Eq. (13) is the output image of image I after rolling guidance filter and the weight $W_{pq}(J)$ is calculated by Eq. (12). Namely, we use image J instead of image I in Eq. (11).

$$C_2(p,d) = \sum_q W_{pq}(J) C(p,d) \tag{13}$$

3.3 Disparity Selection and Post-Processing

We apply the Winner-Takes-all method (WTA) to choose the disparity for pixel p [1]. In Eq. (14), d_p is the disparity for pixel p, S is the set of the possible disparities. $C_2(p,d)$ is the aggregated cost.

$$d_p = \arg\min_{d \in S} C_2(p,d) \tag{14}$$

Once we get the initial disparity, we apply two strategies to do post-processing: left-right check and weighted median filter.

Left-Right Check

We compute two disparity images for this step. Disparity image 0 chooses left image as the reference view and disparity image 1 chooses right image as the reference view. Then we test the pixel's disparity by Eq. (15). Pixels which don't fulfill Eq. (15) will be marked as error pixels, the others will be marked as correct pixels. Here $d_L(x,y)$ represents the disparity for the pixel which coordinate is (x,y) in the left image. The error pixels' disparities in disparity image 0 will be corrected by the disparity of the closest correct pixel [9] while the correct pixels' disparities keep unchanged. Detailed information is stated in article [8-9].

$$d_L(x,y) = d_R\left(x - d_L(x,y), y\right) \tag{15}$$

Weighted Median Filter

The disparity after left-right check will have some streak-like artifacts. Ma et al. proposed a simple but effective solution for the above problem in article [12]. They use weighted median filter to do post-processing in Eq. (16). Here I is the left image. The weight $W_{xy}(I)$ is computed by Eq. (12). Details about Eq. (16) refer to article [12].

$$h(x,i) = \sum_{y \in N(x)} w_{xy}(I)\,\delta\left(V(y) - i\right) \tag{16}$$

But there exists the same problem described in section 3.2. Small scale structures will influence the weights' values. Here we use Eq. (17) instead of Eq. (16). In Eq. (17), we use image J instead of image I. Image J is the output image of I after rolling guidance filter. Here image J is the same as the image J in section 3.2.

$$h(x,i) = \sum_{y \in N(x)} w_{xy}(J)\,\delta\left(V(y) - i\right) \tag{17}$$

4 Implementation on CUDA

Guided filter can be transformed into several box filters and box filter can be computed by integral image [14]. Thus, the efficient implementation of integral image is the key to reduce running time. Here, we take NVIDIA Tesla K20c GPU for example and state the implementation of integral image on CUDA.

4.1 Prefix Sum(Scan)

The definition of integral image is listed in Eq. (18). Here $I_s(x, y)$ is the integral image of image I. Eq. (19-20) are another definitions of integral image. So we can implement integral image in two steps. The first step computes the prefix sum in row-major order. The second step computes the prefix sum in column-major order.

$$I_s(x, \mathrm{y}) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j) \tag{18}$$

$$I_X(x, j) = \sum_{i=0}^{x} I(i, j) \tag{19}$$

$$I_s(x, y) = \sum_{j=0}^{y} I_X(x, j) \tag{20}$$

Traditional prefix sum methods utilized shared memory to do threads' communication [15]. NVIDIA Kepler architecture GPUs give us a more efficient communication way using warp shuffle functions like __shfl(), __shfl_up() and so on. These functions can exchange a variable between threads within a warp without using shared memory. The example of 32 elements' scan is listed below [19].

```
__global__ void scan(void) {
  int laneId = threadIdx.x & 0x1f;
  int value = 31 - laneId;
  for (int i=1; i<32; i*=2) {
    int n = __shfl_up(value, i, 32);
    if (laneId >= i) value += n;
  }
}
```

Eq. (19) computes each row's scan in the image. We can use one warp to do one row's scan [16] as Fig. 6 shows. When warp 0 finishes the computation of the first 32 pixels' scan (Pixel 0-31), we use a temporary value to store the 31th pixel's prefix sum. When warp 0 finishes the second 32 pixels' scan (Pixel 32-63), we add the temporary value to the second 32 pixels' scan. Then we update the temporary value with the 63th pixel's prefix sum. Repeat the above procedures until all the elements' prefix sum is finished. Eq. (20) is implemented by the same way.

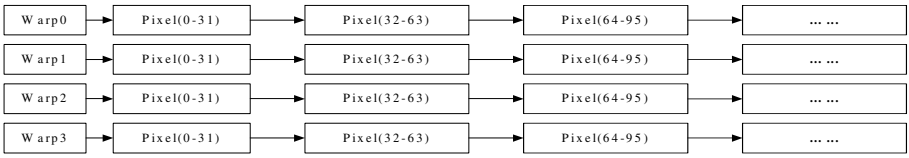


Fig. 6. Scan in row-major order

4.2 Novel Implementation of Integral Image

The two-dimension array is stored by row-major order in CUDA. So when computing scan in column-major order, the accesses of memory can't be coalesced. Thus much time is wasted in the accesses of memory. Bilgic et al. utilized matrix transpose to handle this problem in [17]. The method of integral image in [17] is listed below:

1. Compute the row-major scan
2. Matrix transpose
3. Compute the row-major scan
4. Matrix transpose

However, the matrix transpose procedure is not necessary. We propose a novel method for integral image. In Fig. 7, we will compute A's integral image. A is a 2-dimension array. The height of A is H and the width of A is W. B is the transpose of A. First we combine A's memory and B's memory with CUDA's texture memory. Texture memory has cache optimized for 2D spatial locality, so threads of the same warp that read texture addresses that are close together will achieve the best performance [19]. Firstly, we compute the column-major scan of A as section 4.1 shows but we write the results into B. Thus, we use one warp to compute A's one column's prefix sum and write results into one row of B. The row's number in B and the column's number in A should be the same. Secondly, we do the same procedure to array B. we use one warp to compute B's one column's prefix sum and write the result into one row of array C. Here C is another array which has the same size of A. Finally, C is the integral image of A. Our method only takes two column prefix sum, avoiding matrix transpose steps. So our method has better performance than the method in [17].

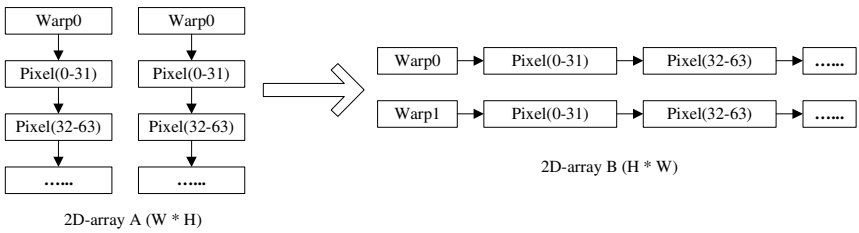


Fig. 7. Implementation of integral image

4.3 Stereo Matching on Two GPUs

Before doing left-right check as the description of Eq. (15), we should have two initial disparity maps. Map 0 is based on left image in stereo images and Map 1 is based on right image. The two maps' computation is independent, so we can compute the two maps on two GPUs to save about 1/2 time. Thus, we compute map 0 on device 0 and compute map 1 on device 1. The key of the implementation is minimizing memory's copy time between two GPUs. When the program is run as a 64-bit process, devices of Tesla GPU can use peer-to-peer access feature to accelerate the copy process [19]:

1. Call function cudaDeviceEnablePeerAccess() to enable peer-to-peer access.
2. Call function cudaMemcpyPeer() and so on to do the memory copy.

5 Experimental Results

We test our method with the Middlebury benchmark [18]. We use a server with Intel E5-2620-v2 2.1GHz CPU and NVIDIA Tesla K20c graphics card. The parameters are listed in Table 1. r_1 is the radius of Gaussian filter and r_2 is the radius of guided image filter. *Iters* are the times of iteration in rolling guidance filter.

Table 1. The parameters in our method.

σ_s	r_1	ε	r_2	*Iters*
1.06	1	0.000215	7	4
γ_{census}	γ_{sobelx}	γ_{sobely}	r_{census}	
7.5	2.5	10.3	1	

Fig. 8 shows the results of our method. Figure (a) shows the final disparity maps and figure (b) shows the error maps under pixel error threshold 1.

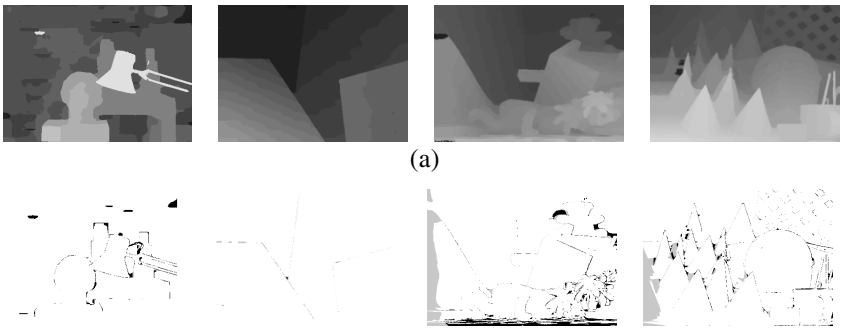


Fig. 8. (a) Final disparity maps. (b) Error maps under pixel error threshold 1.

Table 2 lists the average error ratio of several stereo matching methods on the Middlebury datasets. Our method has better performance over the other methods in the table. Up to May 18, 2015, our method ranks 17th of all the 162 methods. The key of our method is eliminating small structures' effect when computing weights.

Table 2. Evaluation for several local methods on Middlebury stereo images.

Algorithms	Avg. Error (%)	Rank
Ours	4.80	17
Guided agg+WM[12]	5.50	n/a
CostFilter[8]	5.55	50
GeoSup[4]	5.80	60
AdaptWeight[2]	6.67	98

For further proving our method's innovation, Table 3 shows contrast between our method and three other experiments. Experiment 1 only takes ad-gradx cost in Eq. (5) instead of Census-Sobel cost in Eq. (7) while other steps keep the same. Experiment 2 only takes Eq. (11-12) for cost aggregation instead of Eq. (12-13) and experiment 3 only takes Eq. (16) for weighted median filter instead of Eq. (17). From table 2, our method has better performance over the other three experiments.

Table 3. The contrast between our method and three experiments.

Algorithm	Avg. Error (%)
Ours	4.80
Ex. 1	5.77
Ex. 2	5.00
Ex. 3	4.84

Our method is easily implemented on CUDA. Table 4 gives the running time of our method on one GPU and on two GPUs. The running time includes all the steps in our method. The average time is 52.5ms on one GPU and 33.8ms on two GPUs.

Table 4. Running time of our method.

Image	Max Dis	Resolution	Time on 1 GPU (ms)	Time on 2 GPUs (ms)
Tsukuba	15	384*288	18.75	11.63
Venus	19	434*383	30.80	19.43
Teddy	59	450*375	80.24	52
Cones	59	450*375	80.24	52

Table 5 lists several other methods' average running time on Middlebury image sets. The data is selected from article [8, 12]. Our method is a little faster than the others on 1 GPU. The main reason is our fast implementation of integral image on GPU. The methods which rank higher than our method usually utilize Graph cuts, belief propagation and so on to get more accurate results, which will cost much time. So balancing the disparity's accuracy and running time, our method is a great choice.

Table 5. Evaluation of running time of several methods.

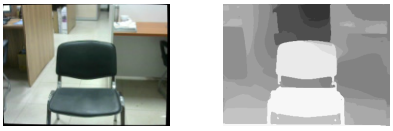
Algorithm	Avg. Time on GPU (ms)	Avg. Error (%)
Ours(1 GPU)	52.5	4.80
Ours(2 GPUs)	33.8	4.80
CostFilter(1 GPU)	65	5.55
Guided agg+WM(1 GPU)	54	5.50

Table 6 gives the running time of the integral image. Here the testing image's resolution is 1M (1024 * 1024) and the type of each pixel is float4. We can see our implementation described in section 4 has the best performance.

Table 6. Running time of integral image

Algorithm	Avg. Time (ms)
Ours	0.72
Our prefix scan + matrix transpose	2.1
Method in [17]	4.0

Fig. 9 gives the photos captured by Logitech C270 consumer cameras in real scenes. We can notice that our approach can produce accurate disparity image.



(a) (b)

Fig. 9. (a) Left input image. (b) Disparity image.

6 Conclusion

This paper has presented a parallel stereo matching algorithm based on edge-aware filter and fast implementation on CUDA. Our method is based on three key points: using Census-Sobel cost measure, utilizing rolling guidance filter and guided filter to do cost aggregation and utilizing rolling guidance filter and weighted median filter to do post-processing. The effect of small scale structures is eliminated when computing weights, thus the disparity quality is improved. The results on Middlebury stereo images show the great performance in accuracy and speed. As heterogeneous computing technology develops, we will explore more efficient implementation on GPU.

Acknowledgement. The work was supported by the National Natural Science Foundation of China (Grants No. NSFC-61402046, NSFC-61170176), Fund for the Doctoral Program of Higher Education of China (Grants No.20120005110002), Fund for Beijing University of Posts and Telecommunications (No.2013XZ10, 2013XD-04).

References

1. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision **47**(1–3), 7–42 (2002)
2. Yoon, K.J., Kweon, I.S.: Locally adaptive support-weight approach for visual correspondence search. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005, vol. 2, pp. 924–931. IEEE (2005)
3. Yoon, K.J., Kweon, I.S.: Adaptive support-weight approach for correspondence search. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(4), 650–656 (2006)
4. Hosni, A., Bleyer, M., Gelautz, M., et al.: Local stereo matching using geodesic support weights. In: 16th IEEE International Conference on Image Processing (ICIP), pp. 2093–2096. IEEE (2009)
5. Mattoccia, S., Giardino, S., Gambini, A.: Accurate and efficient cost aggregation strategy for stereo correspondence based on approximated joint bilateral filtering. In: Zha, H., Taniguchi, R.-i., Maybank, S. (eds.) ACCV 2009, Part II. LNCS, vol. 5995, pp. 371–380. Springer, Heidelberg (2010)
6. He, K., Sun, J., Tang, X.: Guided image filtering. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part I. LNCS, vol. 6311, pp. 1–14. Springer, Heidelberg (2010)
7. He, K., Sun, J., Tang, X.: Guided image filtering. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(6), 1397–1409 (2013)
8. Rhemann, C., Hosni, A., Bleyer, M., et al.: Fast cost-volume filtering for visual correspondence and beyond. In: 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3017–3024. IEEE (2011)
9. Hosni, A., Rhemann, C., Bleyer, M., et al.: Fast cost-volume filtering for visual correspondence and beyond. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(2), 504–511 (2013)
10. Pham, C.C., Jeon, J.W.: Domain transformation-based efficient cost aggregation for local stereo matching. IEEE Transactions on Circuits and Systems for Video Technology **23**(7), 1119–1130 (2013)
11. Zhang, Q., Shen, X., Xu, L., Jia, J.: Rolling guidance filter. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014, Part III. LNCS, vol. 8691, pp. 815–830. Springer, Heidelberg (2014)
12. Ma, Z., He, K., Wei, Y., et al.: Constant time weighted median filtering for stereo matching and beyond. In: 2013 IEEE International Conference on Computer Vision (ICCV), pp. 49–56. IEEE (2013)
13. Zabih, R., Woodfill, J.: Non-parametric local transforms for computing visual correspondence. In: Eklundh, J.-O. (ed.) ECCV 1994. LNCS, vol. 801, pp. 151–158. Springer, Heidelberg (1994)
14. Crow, F.C.: Summed-area tables for texture mapping. ACM SIGGRAPH Computer Graphics **18**(3), 207–212 (1984)
15. Harris, M., Sengupta, S., Owens, J.D.: Parallel prefix sum (scan) with CUDA. GPU Gems **3**(39), 851–876 (2007)
16. Li, J.: High performance edge-preserving filter on GPU. NVIDIA GTC (2015)
17. Bilgic, B., Horn, B.K.P., Masaki, I.: Efficient integral image computation on the GPU. In: 2010 IEEE Intelligent Vehicles Symposium (IV), pp. 528–533. IEEE (2010)
18. http://vision.middlebury.edu/stereo
19. NVIDIA C Programming Guide Version 7.0