Geodesic Tree-Based Dynamic Programming for Fast Stereo Reconstruction

Chin-Hong Sin Chia-Ming Cheng Shang-Hong Lai
National Tsing Hua University
Hsinchu, Taiwan, R.O.C.
g9662553@oz.nthu.edu.tw, {cmcheng,lai}@cs.nthu.edu.tw

Shan-Yung Yang
Institute for Information Industry
Taiwan, R. O. C.
syyang@iii.org.tw

Abstract

In this paper, we present a novel tree-based dynamic programming (TDP) algorithm for efficient stereo reconstruction. We employ the geodesic distance transformation for tree construction, which results in sound image over-segmentation and can be easily parallelized on graphic processing unit (GPU). Instead of building a single tree to convey message in dynamic programming (DP), we construct multiple trees according to the image geodesic distance to allow for parallel message passing in DP. In addition to efficiency improvement, the proposed algorithm provides visually sound stereo reconstruction results. Compared with previous related approaches, our experimental results demonstrate superior performance of the proposed algorithm in terms of efficiency and accuracy.

1. Introduction

Stereo matching is one of the most fundamental problems in computer vision. In the past decade, this problem has attracted much attention from computer vision researchers, and they have advanced the state of the art in stereo matching considerably on pursuit of high quality results. By using the advanced techniques, such as MRF optimization, image segmentation, visibility reasoning, robust plane fitting, bilateral filtering, image matting, etc, integrated stereo algorithms seem to be successful for solving this well known ill-posed problem. However, multi-stage algorithms [10,14,17,22] usually not only suffer from high computational cost but also require nontrivial parameter settings, making them impractical to use.

One of the basic requirements for practical stereo matching solutions is the real-time response. To meet this requirement, previous efficient algorithms are too simple to provide satisfactory results. These simple solutions were usually performed in a single-pass process. Without multiple stages and/or post-refinement processing, the qualities of the real-time stereo approaches are generally incomparable to those of the approaches mentioned above.

Thanks to the modern hardware development, the use of GPU provides another possible solution that compromises the above two extreme approaches. The main challenge becomes how to take advantage of the current computing hardware system to achieve real-time computation. Thus the problem becomes how to transform the sequential process in the multiple stages of the associated techniques into a parallel design, so that the advanced stereo matching algorithm can be implemented on the modern multi-core systems with high computational efficiency

Motivated by the impacting fact in the semiconductor society that the well-known Moore's law [1] was broken due to the fundamental limitation of the physics, the trend is to develop practical vision applications on multi-core hardware devices in the near future. The computer hardware manufactures started to produce the multi-core architecture in a chip. For example, Intel Corporation released its first dual-core CPU in 2006, and the number of cores has increased to four within the three years. This tendency is still in progress, and it will continue in the foreseeable future. To take advantage of this trend, we devote ourselves to developing a new stereo matching algorithm, which is designed for running on the multi-core PC systems.

The dynamic programming (DP) based stereo matching algorithm, first introduced by Cox [2], successfully employs the uniqueness constraint and ordering constraint in the matching problem. The DP optimization can be performed individually along each scanline, which makes it very efficient and straightforward. However, the horizontal streaking effect occurs to the boundary of depth discontinuity, because the piecewise smoothness constraint is only enforced along the scanline. To improve this drawback, Veksler [3] proposed tree-based dynamic programming (TDP) that applied the DP optimization through the tree structure constructed by the conventional minimum spanning tree (MST) algorithm.

To take full advantage of the modern hardware support on PC, our stereo matching algorithm simultaneously performs tree construction and disparity estimation through parallelization on both CPU and GPU. Concurrently, CPU computes the efficient geodesic distance transformation for tree construction according to the input image intensity

2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops
978-1-4244-4441-0/09/$25.00 ©2009 IEEE

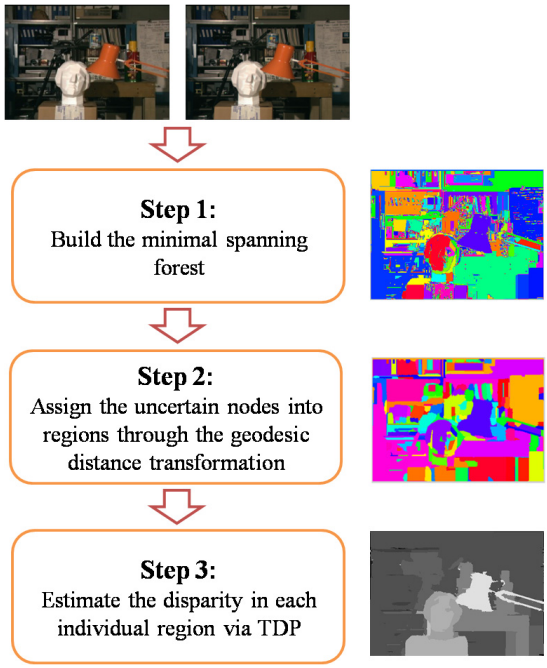


Figure 1: System flow of the proposed algorithm

content, while GPU takes charge of data cost for each pixel. After that, we apply enhanced TDP to individual trees for both of the stereoscopic images and visibility constraint to obtain the final results. On the PC equipped with AMD Athlon dual core processor with nVidia GT-9800, the total execution time for both image segmentation and disparity estimation of both images can reach up to 6 fps for the 384x288 stereo image pair with 16 disparity levels.

The main contributions of this paper can be briefly summarized in the following three aspects. First of all, we propose a parallel tree-based stereo matching algorithm based on efficient geodesic image segmentation. Secondly, the proposed algorithm is designed suitably for advanced multi-core architecture on PC. Last but not least, our algorithm provides the image segmentation result in addition to the disparity map estimation, which is done in near real-time response. Through the experimental results on both synthetic and real data, we demonstrate that the estimated disparity map outperforms those of previous closely related approaches in terms of efficiency and accuracy.

2. System Overview

The flow diagram of our system is shown in Figure1. The first step is to construct the minimal spanning forest according to intensity similarity. In practice, uncertain pixels in the unexpected small trees may appear due to inevitable noises. Thus, pixels in these unexpected trees are detected and re-assigned to the neighboring reliable regions. A possible solution [7] is to hierarchically calculate the mean and standard variance of the tree to decide whether to merge the two adjacent trees. However, this method is too time-consuming to meet the real-time response requirement. We perform the tree merging in one step. The strategy is to apply the geodesic distance transformation [6] from the uncertain pixels to the target trees. It is worth noting that this strategy not only reassigns the uncertain pixels to a reliable tree with the shortest path, but also preserves a tree structure to the MST. Finally, we perform an energy minimization on each local MST by using TDP optimization.

We divide the task to different computing platforms for gaining the optimal performance. Our algorithm performs the computationally intensive operations on GPU by using CUDA. Concurrently, the remaining operations are assigned to multi-core CPU. In the beginning, a minimal spanning forest is constructed. In this process, associating each pixel to a thread parallelizes the calculation of the intensity gradient, thus the computation of the weights of the edges in the MST is performed on GPU. This information is passed to CPU for constructing the minimal spanning forest. Geodesic distance is done efficiently by a two-pass minimum filter on CPU. Meanwhile, the data terms in the energy function is pre-calculated on GPU. As the same strategy as intensity gradient calculation, we associate each pixel to a thread. Hence, in the final step of our algorithm, i.e. the energy minimization on each local MST by using tree-based dynamic programming, the optimization can be executed very fast because the data terms are already calculated on GPU. This process is currently implemented on the multi-core CPU by using OpenMP.

3. Geodesic Tree Construction

We improve the method introduced by Criminisi et al. [6] to calculate the geodesic distance of the uncertain pixels. Their method requires the user to provide foreground and background strokes. Instead, our algorithm automatically constructs the minimal spanning forest. Pixels in a MST which depth less than a threshold are marked as uncertain pixels. These uncertain pixels are reassigned to the neighboring MSTs according to their geodesic distance.

3.1. Minimal Spanning Forest Construction

We modified Kruskal's algorithm [8] to construct the minimal spanning forest. The modified Kruskal's algorithm is given as follows:

Algorithm 1. Modified Kruskal's algorithm

- Initially, we treat each pixel as a separated tree, and construct the edges, $\boldsymbol{e}_i$, of four-connected neighbors
- For each $\boldsymbol{e}_i$, use GPU to compute the edge weights, $\boldsymbol{w}_i$, according to the color dissimilarity
- Use CUDA *radix_sort* to sort all the edge weights in the ascending order and store the ordered edges in the queue $\boldsymbol{S}$.
- *While* $\boldsymbol{S} \notin \varnothing$
  - Pop out $\boldsymbol{e}_i$ from $\boldsymbol{S}$
  - *If* $\boldsymbol{w}_i < \lambda$ *then* add $\boldsymbol{e}_i$ to connect the two nodes of the edge.
  - *Otherwise* discard $\boldsymbol{e}_i$

First, we use radix sort [18][19] performed on GPU to sort the edge weights in an increasing order. By using a queue, $\boldsymbol{S}$, to store the sorted edges, we sequentially pop out the edge from $\boldsymbol{S}$ and check whether it satisfies the criterion to add the edge to connect the two nodes of the edge by a user-defined threshold.

3.2. Geodesic Distance Transformation

We improve the method introduced by Criminisi in [6] to calculate the geodesic distance of the uncertain pixels. Their method requires the user to provide foreground and background strokes. Instead, our algorithm automatically constructs the minimal spanning forest [7]. Pixels in a MST with depths less than a threshold are marked as uncertain pixels. These uncertain pixels are reassigned to the neighboring MSTs according to their geodesic distance.

3.3. Energy Minimization via Tree-based Dynamic Programming

In each region, we can define its own energy function for disparity estimation. Similar to the previous work, the energy function can be defined as

$$E(D) = \sum_{p \in v} m(d_p) + \lambda \sum_{(p,q) \in E} s(d_p, d_q) \qquad (1)$$

We optimize this energy function with respect to disparity map D in the dynamic programming framework as described in [3]. Here, d_p is denoted as a disparity value at pixel p in the left image, and $m(d_p)$ is the matching penalty for assigning disparity d_p to pixel p. For simplicity, it can be the absolute difference between the pixel p in the left image, and the pixel p shifted by d_p horizontally in the right image. In this work, we use the sum of modified absolute intensity differences [9] inside a local window as the matching penalty. Here, we use GPU to calculate this part in the middle of CPU performing geodesic tree construction. In Eq. (1), $s(d_p,d_q)$ is a smoothness penalty for assigning disparities d_p and d_q to p and q which are connected by an edge in the graph. For the purpose of enforcing the smoothness constraint, $s(d_p,d_q)$ should be a monotonically non-decreasing function defined as

$$s(d_p, d_q) = \left| d_p - d_q \right| / (w_{pq} + \mu) \qquad (2)$$

where w_{pq} is the edge weight between pixel p and q, and μ is a small positive value (set to 10^{-5} in our experiments) to avoid dividing by zero. Instead of setting a fixed constant, the smoothness penalty is dynamically changed based on the content, which is similar to [10].

Dynamic programming is a technique that solves an optimization problem from the solutions of its sub-problems. Here, we compute the energy function from leaf nodes through the root node in a tree. Each node v has a parent node, denoted by $p(v)$, except the root node, and the minimal energy of the node v is a subset of the graph consisting of a subtree rooted at v and the edge between v and $p(v)$, thus it can be written in a recursive form as a function of $d_p(v)$, given by

$$E_v\left(d_{p(v)}\right) = \min_{d_v \in H} \left(m(d_v) + s(d_v, d_{p(v)}) + \sum_{w \in C_v} E_w(d_v) \right) \qquad \ldots (3)$$

where C_v is the set of children of node v and H is the set of all disparity levels considered here. For the leave node, because it does not have a child, its associated energy function can be simplified as follows:

$$E_v\left(d_{p(v)}\right) = \min_{d_v \in H} \left(m(d_v) + s(d_v, d_{p(v)}) \right) \qquad (4)$$

For the root node, it does not have a parent, so its associated energy function can be written as,

$$E_{root} = \min_{d_{root} \in H} \left(m(d_{root}) + \sum_{w \in C(root)} E_w(d_{root}) \right) \qquad (5)$$

From this equation, we know that the disparity at the root node can be decided directly. Once it is decided, the disparity of the corresponding child node can be easily decided by passing the information of the estimated disparity from the parent node to child node. This process can be recursively applied to determine the disparities of all the nodes in a tree. The complexity of this DP process is $O(h^2 n)$, where h is the total number of disparity levels, and

n is the total number of nodes. We can reduce the complexity to *O(hn)* by simplifying the smoothness penalty function as follows, which is similar to [20][21]:

$$s\left(p_{p(v)}, q_v\right) = \begin{cases} 0, & \text{if } d_{p(v)} = d_v \\ w_{p(v),v}, & \text{otherwise} \end{cases} \quad (6)$$

Therefore, there are only two choices for the optimal disparity assignment for v. The first case is when $s(d_{p(v)},d_v)=0$, and this leads to the following equation

$$E_v\left(s(d_{p(v)}, d_v) = 0\right) = m(d_v) + \sum_{w \in c_v} E_w(d_v) \quad (7)$$

The second case is when $s(d_p,d_v)\neq 0$, and it corresponds to the following energy function:

$$E_v\left(s(d_{p(v)}, d_v) \neq 0\right) = m(d_{v^*}) + \sum_{w \in c_{v^*}} E_w(d_{v^*}) + const \quad (8)$$

The disparity value d_{v^*} is determined by minimizing $m(d_{v^*}) + \sum_{w \in C_{v^*}} E_w(d_{v^*})$, thus the final energy equation can be written as:

$$E(d_{p(v)}) = \min\left(E_v(s(d_{p(v)}, d_v) \neq 0), E_v(s(d_{p(v)}, d_v) = 0)\right) \quad \ldots (9)$$

Therefore, the computational complexity can be reduce to *O(hn)*.

We use OpenMp to perform the energy minimization in parallel with each thread assigned to a region, and there is no data interchange between regions during the energy minimization for each tree.

3.4. Occlusion Handling

Occluded pixels are the crucial pixels that need to be specially handled, because the pixels visible in one input image have no corresponding matched pixels in the other input image. To tackle this problem, we first treat the left image as the reference image and calculate its disparity map. Once we have the disparity map, we warp the disparity map of the right image to the left image view. The pixels without correspondences after the warping are treated as occluded pixels for the left image. Thus, the data terms for the occluded pixels are set to a very small value or zero in the progress of performing energy minimization.

In addition, we need to handle the pixels in homogeneous regions in a special way. Most of the time, the estimated disparities in homogeneous regions are incorrect because the associated cost function values are almost the same for different disparity levels. Hence, similar to the way of handling the occlusion pixels, we set the data term for a pixel in a homogeneous region to a very small value. We use a very simple way to detect homogeneous pixels by calculating the sum of absolute intensity differences between neighboring pixels inside a local window and comparing this value with a threshold.

4. Experiment Result

All of our experiments were performed on a PC equipped with AMD Athlon 64 X2 Dual core processor 3600+ of 2.01 GHz. The GPU model was nVidia GT-9800 with 112 stream processors.

We applied the proposed algorithm to the Middlebury datasets [12] to evaluate the performance of our algorithm. The results are shown in Table 2. There are four parameters to be set in our algorithm; namely, T_1 is the threshold of discarding the edge in the tree construction, T_2 is the threshold of destroying the trees with depth less than this value, T_3 is the threshold for the smoothness constraint, and T_4 is the threshold for indicating is a pixel is located in a homogeneous region. The window size used to calculate the data term and homogeneous pixel was set to be 7x3. Table 1 lists the parameter setting for the four datasets.

Table 1. Parameter settings for different image pairs

	T_1	T_2	T_3	T_4	λ
Tsukuba ,Venus	15	30	50	10	20
Teddy, Cones	25		70		

Compared to the conventional DP-based stereo algorithm [2], our results were free from streaking effect. However, we restrict the smoothness penalty to be either zero or a constant, so that most of the bad pixels occur in the slant surface, as shown in Figure2.

In Table 2, we list the result of our method to compare with those of the state-of-the-art quality-oriented methods [14][17], speed-oriented methods [15][16], and the most closely related DP-based methods [2][3][10]. First, compared with the state-of-the-art stereo algorithms, such as [14][17] listed in the Table 2, they normally take a couple of minutes to perform iterative multi-stage refinement to obtain a high quality depths, while our method is much faster. Second, compared with previous real-time stereo matching methods, e.g. [15][16], the proposed algorithm provides the additional image segmentation result in addition to the disparity map estimation. At last, we compare our method to the most related DP-based methods [2][3][10]. Although we observe that the region-based TDP [10] seems better in terms of quality, we have to point out that this method requires an additional over-segmentation algorithm as a preprocessing

Table 2. The quantitative evaluation on Middlebury dataset, numbers in the table indicate average bad pixels over indicated region. n-occ : non-occlusion region, all : all region, disc : discontinues region.

	Tsukuba			**Venus**			**Teddy**			**Cones**		
	n-occ	all	disc	n-occ	all	disc	n-occ	all	disc	n-occ	all	disc
DoubleBp[14]	0.83	1.24	4.49	0.10	0.35	1.46	1.41	4.13	4.73	1.71	7.02	5.16
AdaptingBP [17]	0.77	1.31	5.66	0.10	0.18	1.42	1.05	2.00	3.74	1.89	6.42	5.69
RegionTreeDP[10]	1.20	1.43	6.01	0.09	0.30	1.11	4.22	6.09	10.7	2.98	8.10	7.31
RealTimeBP [15]	1.25	3.04	6.66	0.63	1.53	7.68	5.68	8.27	10.2	2.9	9.11	8.27
Our Method	**1.52**	**2.28**	**7.53**	**0.58**	**0.82**	**3.06**	**5.15**	**8.45**	**11.6**	**4.24**	**9.92**	**10.1**
RealTimeGPU[16]	1.34	3.27	7.17	1.02	1.90	12.4	3.90	8.65	10.4	4.37	10.8	12.3
TreeDP[3]	1.73	2.55	8.82	1.21	1.89	7.35	8.80	17.2	17.2	5.57	14.1	12.5
DP [2]	3.43	4.23	9.85	6.50	7.43	17.4	7.11	17.9	13.4	6.52	15.1	15.1

step as well as a multi-stage refinement process to obtain the final results, which is much more time consuming compared to our method.

The proposed algorithm provides a very efficient segmentation-based stereo matching algorithm, as shown in Table 3. The segmentation information helps us to improve the stereo matching accuracy, especially around the edge region. Our algorithm is practical for applications that require both disparity estimation and image segmentation especially when computational speed is a primary consideration.

Table 3. The execution time for different image pairs.

	Segmentation	Energy minimization	Total time
Tsukuba	0.122 sec	0.047 sec	0.169 sec
Venus	0.467 sec	0.110 sec	0.577 sec
Teddy	0.532 sec	0.203 sec	0.735 sec
Cones	0.313 sec	0.218 sec	0.534 sec

We also performed the energy minimization with a single core to compare the execution time, though multi-core processors can gain more computational efficiency because the optimization process is parallelized in multi-thread framework. Table 4 compares the computational time required for using single-core and dual–core CPU for energy minimization.

Table 4. Different running time of energy minimization with single-core and dual-core CPU.

	Single Core	Dual Core
Tsukuba	0.078 sec	0.047 sec
Venus	0.157 sec	0.110 sec
Teddy	0.288 sec	0.203 sec
Cones	0.297 sec	0.218 sec

In the last experiment, we apply our algorithm to a stereo video clip to demonstrate the performance of the proposed algorithm on real stereo video. The results are shown in Figure 3.

5. Conclusions

In this paper, we proposed a geodesic tree-based dynamic programming algorithm for stereo matching. Multi-tree structures are constructed based on the image geodesic distance. The associated energy minimization for each tree corresponding to a region is performed by DP individually in parallel. Our experimental results showed the proposed algorithm can provide good disparity estimation and image segmentation very efficiently on a CPU+GPU PC system. As far as we know, the proposed algorithm is the fastest segmentation-based stereo matching algorithm. It is very useful for applications that require both the stereo matching and image segmentation results.

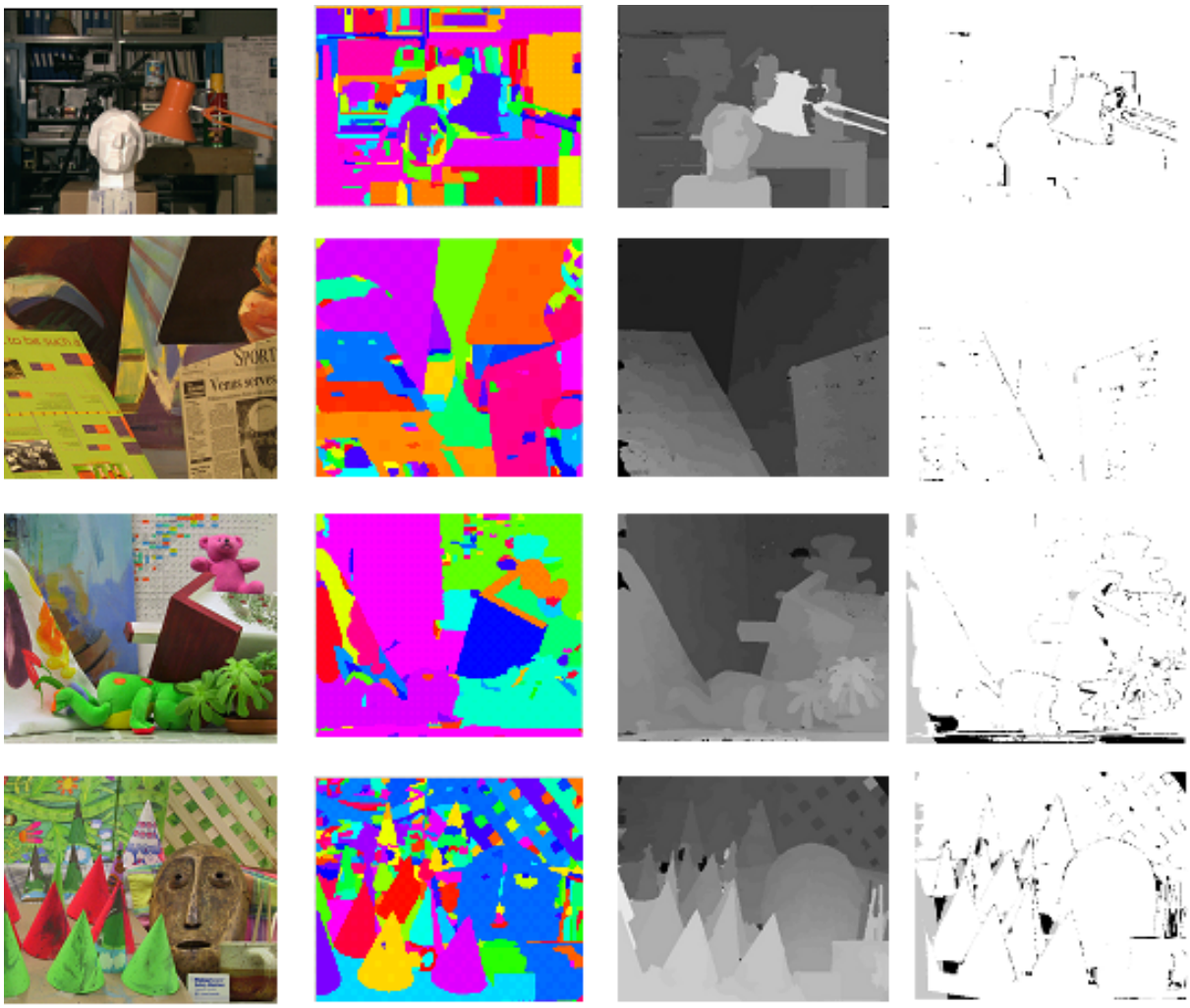
The limitation of our algorithm is confined on the construction of minimal spanning forest. It is not completely parallel so far. The other weakness of our algorithm is the smoothness constraint. Restricted smoothness penalty in two levels only sacrifices the accuracy of the stereo matching.

As the future research directions, we plan to investigate a more elegant way to refine the smoothness constraint such that the accuracy of stereo matching can be improved and the corresponding energy function can still be efficiently minimized with DP. In addition, we plan to develop a truly parallel algorithm to construct minimal spanning forest. Furthermore, we are also interested in extending this technique to further speed up the depth estimation for stereo videos.

Acknowledgments: This study is conducted under the "III Innovative and Prospective Technologies Project" of the Institute for Information Industry which is subsidized by the Ministry of Economy Affairs of the Republic of China.

References

[1] C. E. Leiserson and I. B. Mirman. How to survive the multicore software revolution. In *CILK ARTS*, 2008.



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

Figure 2. Experimental results on synthetic data on Middlebury dataset. (a) column is the original image, different color in image of column(b) present different region, (c) column is the result of our algorithm, (d) column is the result of bad pixel.

[2] I. Cox, S. Hingorani, S. Rao, and B. Maggs. A maximum likelihood stereo algorithm. In *Computer Vision, Graphics and Image Processing*, 63(3): 542-567, 1996.

[3] O. Veksler. Stereo correspondence by dynamic Programming on a tree. In *CVPR*, 2005.

[4] nVidia. *NVIDIA CUDA Programming Guide 2.2*. 2008.

[5] The OpenMP API Specification for Parallel Programming: http://openmp.org/

[6] A. Criminisi, T. Sharp, and A. Blake. GeoS: geodesic image segmentation. In *ECCV*, 2008.

[7] O. Grygorash, Y. Zhou, and Z. Jorgensen. Minimum spanning tree based clustering algorithms. In *ICTAI*, 2006.

[8] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[9] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE TPAMI*, 20: 401-406, 1998.

[10] C. Lei, J. Selzer, and Y.-H. Yang. Region–tree based stereo using dynamic programming optimization. In *CVPR*, 2006.

[11] M. Gong and Y. H. Yang. Fast stereo matching using reliability-based dynamic programming and consistency constraints. In *ICCV*, 2003.

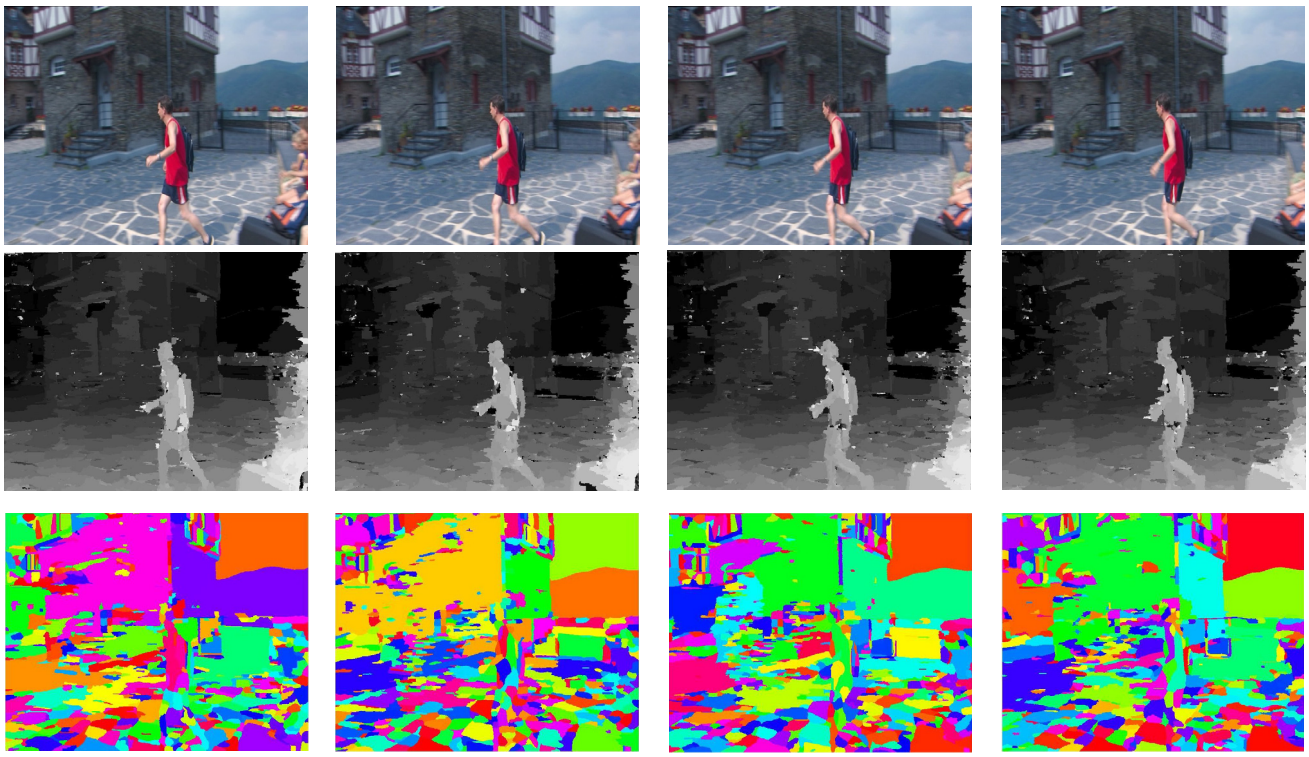
[12] Middlebury Stereo Vision Page: http://vision.middlebury.edu/stereo/

[13] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *ECCV*, 2002.

[14] Q. Yang, L. Wang, R. Yang, H. Stewénius, and D. Nistér. Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling. In *CVPR*, 2006.

[15] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nistér. Real-time global stereo matching using hierarchical belief propagation. In *BMVC*, 2006.

[16] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nistér. High-quality real-time stereo using adaptive cost aggregation and dynamic programming. In *3DPVT*, 2006.
[17] A. Klaus, M. Sormann and K. Karner. Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure. In *ICPR*, 2006.
[18] G. E. Blelloch. Prefix sum and their application. In *Technical Report CMU-CS-90-190*, School of Computer Science, Carnegie Mellon University, 1990.
[19] N. Satish, M. Harris and M. Garland. Designing efficient sorting algorithms for manycore GPUs. In *IEEE Proc. Int'l Parallel and Distributed Processing Symposium*, 2009.
[20] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. *IJCV*, 70: 41-54, 2006.
[21] A. Brunton, C. Shu, and G. Roth. Belief propagation on the GPU for stereo vision. In *CRV*, 2006.
[22] M. Bleyer, M. Gelautz, C. Rother, and C. Rhemann. A stereo approach that handles the matting problem via image warping. In *CVPR*, 2009.



(a) **(b)** **(c)** **(d)**

Figure 3. Experimental results on real data. First row is the original image, second row is the result of our algorithm, different color in image of third row present different region.