Hierarchical Belief Propagation To Reduce Search Space Using CUDA for Stereo and Motion Estimation

Scott Grauer-Gray and Chandra Kambhamettu
University of Delaware
Newark, DE 19716
{grauerg, chandra}@cis.udel.edu

Abstract

This paper describes a hierarchical belief propagation implementation in which a 'rough' disparity map calculation or motion estimation in higher levels is used to limit the search space and enable the calculation of the desired disparity map/set of motion vectors using a smaller search space than traditional belief propagation. We implement our algorithm on the GPU using the CUDA architecture and explore a number of implementation details with promising results; it is clear that the storage requirements of belief propagation can be significantly reduced using our method without too large of a sacrifice in the accuracy of the results. In addition, we take advantage of the interpolation capabilities built into the GPU in order to retrieve the computed disparities/motion vectors at sub-pixel accuracy without making any change in implementation.

1. Introduction

Belief propagation for stereo processing is introduced and described by Sun et al. in [10]. Since then, it has developed into a popular algorithm for the computation of a disparity map from a pair of stereo images, as evidenced by the fact that three of the top five scoring implementations in the Middlebury Stereo evaluation described by Scharstein and Szeliski [9] use the algorithm in some manner as a part of their overall implementation (see http://vision.middlebury.edu/stereo/ for current rankings). Belief propagation is known to retrieve relatively accurate results, but the algorithm is limited by high storage requirements; each pixel in the image must hold 4 'message' values and a 'data cost' value for every disparity candidate. In this paper, we present a belief propagation implementation in which a hierarchical scheme is employed to reduce the disparity search space, lowering the storage requirement of the implementation.

2. Overview of Belief Propagation

Belief propagation is a general-purpose, iterative algorithm used for inference on Markov Random Fields (MRFs). Sun shows in [10] that the stereo vision problem can be converted to an NP-hard energy minimization problem which is equivalent to finding the maximum a posteriori (MAP) estimator of a MRF, and belief propagation can be used to retrieve an approximate solution. The total energy is equal to the sum of the data costs for each pixel at the assigned disparity plus the discontinuity costs for each pair of neighboring pixels with assigned disparity values. The data cost is computed using the brightness consistency assumption where the data cost increases as the difference in intensity between corresponding pixels increases, while the discontinuity cost increases as the disparity difference between neighboring pixels increases. In the experiments described in this paper, both the data and discontinuity costs are computed using the truncated linear model where the costs increase linearly with an increase in the intensity/disparity difference up to a given 'truncation' parameter, and a third parameter controls the weight of the data cost versus the discontinuity cost.

Belief propagation for stereo processing operates via the utilization of the data costs together with the four sets of message values corresponding to each possible disparity at each pixel. In each iteration, the current set of values are used to compute and send a vector of updated message values to each 4-connected neighboring pixel. After all iterations are complete, these values are used to compute the estimated disparity value at each pixel. Belief propagation generally gives good results, but the storage requirement needed in order to run the algorithm is O(5 * image height * image width * number of possible disparity values) due to the necessity of storing the data costs and message values for each possible disparity at each pixel.

3. Related Work in Belief Propagation

Felzenszwalb and Huttenlocher present various methods to speed up a belief propagation implementation in [4]. One of these methods is a hierarchical belief propagation algorithm: this method uses a pyramid scheme in which the width and height of the algorithm components are halved with each successive level of the pyramid, and message values from higher levels in the pyramid are used to initialize message values in the lower levels. However, this method is only designed to decrease the number of iterations necessary for the message values to converge; it does not reduce the disparity search space or the storage requirements of the implementation. In [6], Isard and MacCormick speculate that a hierarchical approach which reduces the search space at each pixel is possible, but postpone the work for a future paper.

Brunton et al. [3], Yang et al. [8], and Grauer-Gray et al. [5] each present hierarchical belief propagation implementations which take advantage of the parallel processing power of the graphics processing unit (GPU) in order to decrease the running time of the implementation. Two of these works ([3] and [8]) map their implementation to a graphics API in order to utilize the GPU, while the other ([5]) uses the GPU via the Compute Unified Device Architecture (CUDA) technology from nVidia. However, none of these approaches take steps to reduce the search space at each pixel.

An alternate approach to reducing the storage requirements of belief propagation is presented by Liang et al. in [7]; this work presents a tile-based approach to belief propagation that requires only 1-5% of the storage requirements of 'regular' belief propagation while achieving similar results. However, this approach does not reduce the search space at each pixel, instead reducing the storage requirements by repeatedly running belief propagation iterations on individual 'tiles' and storing the outbound message values that are needed for input to neighboring tiles.

4. Outline of Our Belief Propagation Implementation

In our implementation, we use a hierarchical scheme in which 'rough' results computed at higher levels are used to initialize the search space at lower levels, where the results are 'fine-tuned'. We begin at the top level with a relatively large increment between disparity candidates and decrease the increment by some proportion with each succeeding level, while the number of disparity candidates remains constant. As a result, the magnitude of the range between the minimum and maximum disparity candidates decreases with each level down the hierarchy.

In each level, we retrieve the estimated disparity at each pixel, and use this estimated disparity as the midpoint of the disparity range for the corresponding pixel in the next level, though adjustments may be necessary to account for the condition that the entire disparity range of the lower level pixel must be within the disparity range of the corresponding pixel at the higher level. The implementation is designed and executed using the parallel processing power of the graphics processor unit (GPU) via the Compute Unified Device Architecture (CUDA) from nVidia. We omit an extensive CUDA introduction in this paper; the interested reader is invited to look at the documentation provided by nVidia [1].

As discussed in section 3, Grauer-Gray et al. [5] previously presented a belief propagation implementation using the GPU and CUDA, and the results show that the architecture can be used to speed up a related implementation without losing accuracy. In addition to taking advantage of the GPU's processing capabilities, our implementation takes advantage of the hardware's interpolation capabilities to retrieve sub-pixel disparities without making any changes to the general implementation.

Initially, we hold the image resolution of each level constant, making for a 'cuboid' level hierarchy rather than a pyramidal one, and we initialize the message values at each level to 0 rather than using the message values from upper levels to initialize the message values at lower levels. Later, we adjust the implementation to use a pyramidal hierarchy and initialize the message values at lower levels with message values calculated at upper levels in order the reduce the number of iterations needed for message value convergence. Our implementation takes advantage of the checkerboard scheme presented by Felzenszwalb and Huttenlocher [4] to half the number of messages passed in each iteration, dividing the pixels into two 'sets' as represented by a checkerboard and passing message values from one set to the other in each iteration.

One issue with this hierarchical scheme is that disparity candidates are 'skipped' at higher levels where the disparity candidate increment is greater than the increment at the bottom level. In order to mitigate this factor, we use the disparity candidate increment of the bottom level when computing the data costs at each level, setting the data cost corresponding for each candidate disparity to the minimum of the data costs that this disparity could correspond to at the bottom level. In a two-level scheme where the disparity increment is 2 at the upper level and 1 at the bottom level, the data cost for each candidate disparity at the upper level is the minimum of data costs corresponding to three different disparities: the candidate disparity minus 1, the candidate disparity itself, and the candidate disparity plus 1.

5. Results Using Initial Scheme

We run our initial implementation on the Tsukuba stereo set using a data cost cap of 15.0, a discontinuity cost cap of

Start Disparity Incr. / Num Levels	% Bad Matches w/ Error Threshold 1.0
1 / 1	4.85
2 / 2	4.88
4 / 3	5.71
8 / 4	6.56

Table 1. Results using the cuboid hierarchy when varying the starting disparity candidate increment and number of levels such that the disparity candidate increment is 1 at the bottom level.

1.7, and a data weight set to 0.07 after smoothing the images with a CUDA-implemented Gaussian filter of sigma 1.0. The reference image of the stereo set and the ground truth disparity map are shown in the top row of figure 1. First, we run the implementation using a single level with 200 iterations and a disparity range from 0 to 16 in increments of 1. The resulting disparity map is shown in the left side of the second row of figure 1.

Next, we run our implementation on the same images using two levels with 200 iterations per each level. The increment in disparity candidates is set to 2 at the upper level and is halved to 1 in the lower level. This results in a search space of 9 possible disparities at each level, nearly halving the space needed for the data cost and message values in the implementation. The resulting disparity map is shown on the right side of the second row of figure 1. We continue to run our implementation on the Tsukuba set using 3 and 4 levels, with disparity candidate increments starting at 4 and 8 and search spaces of 5 and 3 possible disparities, respectively. The disparity increment is halved with each succeeding level, with a final disparity increment of 1 in each experiment. The results of these experiments are shown in bottom row of figure 1.

Based on the resulting disparity maps, the sacrifice of any accuracy in each successive experiment does not appear to be that high given the saving of storage space. We compute the error in each experiment in terms of the percent of 'bad pixels' with a difference in disparity greater than 1.0 from the ground truth (excluding a border region of 18 pixels on each edge); the error is 4.85% when the disparity candidate increment starts at 1 and gradually increases up to 6.56% when the disparity candidate increment starts at 8, with all the results in table 1.

6. Results Using Pyramid Hierarchy

Next, we modified our implementation to use a 'pyramid' hierarchy in which each pixel at each level (other than the bottom level) corresponds to a 2 X 2 block of pixels in the next level down in the hierarchy. The algorithm is modeled after the multi-grid belief propagation described by Felzenszwalb and Huttenlocher in [4] such that the data

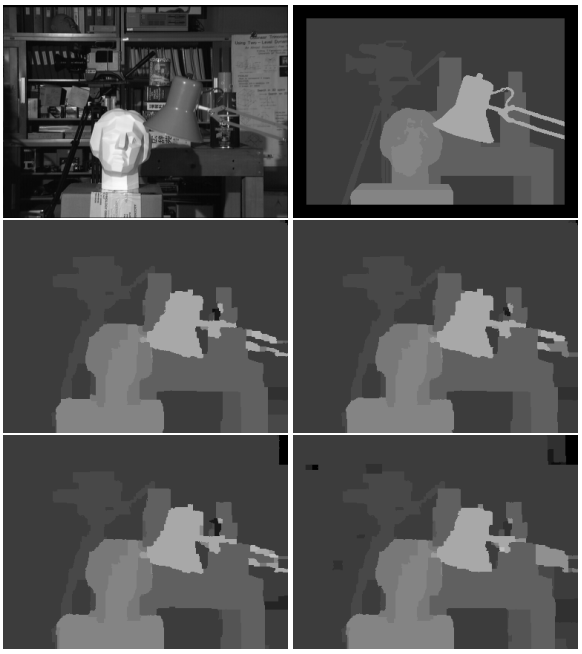

Figure 1. Top level (left to right): Tsubuka reference image and ground truth disparity map. Middle and lower levels (left to right then top to bottom): Resulting disparity map after running implementation using 1, 2, 3, and 4 levels with the disparity candidate increment beginning at 1, 2, 4, and 8, respectively, and cut in half with each successive level.

Starting Disparity Incr. / Num Levels	Bad Matches w/ Error Threshold 1.0
1 / 1	4.85
2 / 2	5.37
4 / 3	5.55
8 / 4	7.43

Table 2. Results using the pyramid hierarchy when varying the starting disparity candidate increment and number of levels such that the disparity candidate increment is 1 at the bottom level.

cost corresponding to each disparity candidate at each pixel is equal to the sum of the data costs corresponding to the disparity candidate at all the bottom level pixels corresponding to the pixel at the current level. The results of running pyramidal belief propagation using 1, 2, 3, and 4 levels with disparity candidate increments starting at 1, 2, 4, and 8, respectively, is shown in figure 2 and table 2; note that the results given in the table exclude a border region of 18 pixels on each edge.

7. Results Using Pyramid Hierarchy and Message Propagation From Upper Levels

Next, we modify our implementation to use the message values from higher levels to initialize the message values at

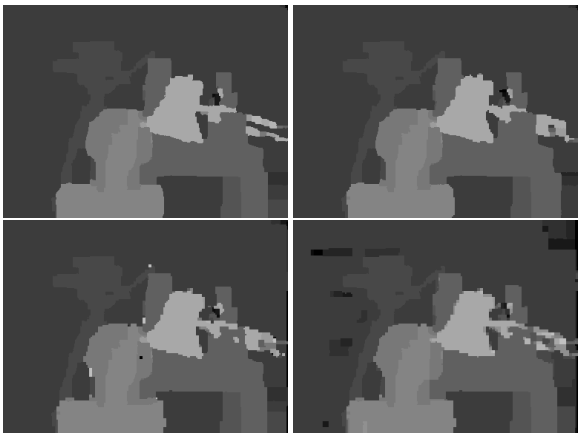

Figure 2. Resulting disparity maps (from left to right then top to bottom) after running implementation using the 'pyramid' hierarchy with 1, 2, 3, and 4 levels and disparity candidate increments beginning at 1, 2, 4, and 8, respectively, and then cut in half with each successive level.

lower levels. Since the increment between disparities is reduced with each succeeding level down the hierarchy, the message values are not available for every disparity candidate in the lower level; the disparity candidate in the lower level may be between two disparity candidates in the upper level. One potential method to retrieve the desired message values is to use interpolation between the nearest disparity candidates at the previous level. We run our implementation using this method with 2 levels, 200 iterations per level, and with a disparity candidate interval of 2 at the upper level and 1 at the lower level. The resulting disparity map is shown in left of figure 3; it is clear from these results that this method does not retrieve an accurate disparity map.

A likely explanation for the lack of accuracy using interpolation is that there is no guarantee that the message values obtained from the neighboring disparity candidates can predict the 'correct' message values corresponding to the candidates in-between. In the case where the 'middle' disparity candidate at the lower level is the 'best' candidate, the message values corresponding to that disparity are ideally less than the message values corresponding to all other disparity candidates. However, there is no way this will be reflected using interpolation from the neighboring disparities. One way to handle this scenario is to use the minimum of the surrounding message values, which prevents this 'middle' disparity candidate from starting with a disadvantage in terms of being the 'best' disparity value. The resulting disparity map using this approach is displayed in the right of figure 3, and appears to be much closer to the ground truth than the previous result.

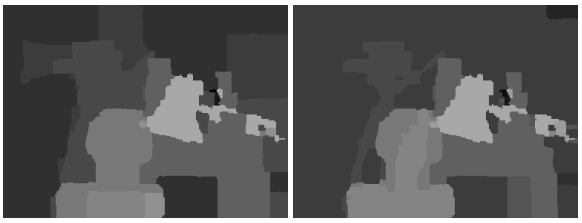

Figure 3. Computed disparity map from running implementation using the 'pyramid' scheme using 2 levels starting with a disparity candidate increment of 2 that is halved to 1 at the lower level. The left image shows the resulting disparity map when the lower level message values are retrieved using interpolation, and the right image shows the resulting disparity map when the lower level message values are retrieved using the minimum of the neighboring values.

8. Optimizing the Storage Requirements and Running Time of the Implementation

Thus far, our focus has been on accuracy of the results rather than running time. However, running time is often an important consideration in a stereo processing implementation, particularly when considering the possibility of real-time analysis. Our implementation is performed on the GPU using CUDA, in part because the results presented by Grauer-Gray et al. in [5] makes it clear that the processing power of the GPU can be used to decrease the running time of the implementation. In this section, we explore adjustments to the number of iterations, the starting disparity candidate increment, and the proportion at which to adjust the disparity candidate increment between levels in order to retrieve the optimal balance between the storage requirements, running time, and accuracy of our implementation. Unless otherwise specified, each of the following experiments use the pyramid hierarchy with message propagation from upper levels in order to minimize the number of iterations needed for message value convergence.

The results in this section are generated using a nVidia 8600M GT GPU with four multiprocessors on a laptop with a Intel Core 2 Duo CPU running at 2.00 GHz. As shown in [5], the running time could be further decreased by running the implementation on a computer with a higher-end GPU.

8.1. Adjustments in the Iteration Count

We begin by looking at an implementation that begins with a disparity candidate increment of 8 and is cut in half in each subsequent level until 1, resulting in a total of 4 levels. In this experiment, we compare the running times and accuracy for varying numbers of iterations. We run these experiments using message propagation from upper level using the minimum message value corresponding to the surrounding disparity candidates as described in section 7, and also with reset message values in each iteration. The results

Num Iters	Running time (ms)	% Bad Matches w/ Error Threshold 0.0 (Min mess. vals prop. / Reset message vals)
200	2920	15.11 / 15.76
150	2268	15.03 / 15.79
100	1568	15.04 / 15.77
50	884	15.57 / 16.59
25	551	17.18 / 18.41
15	414	19.84 / 20.63
10	345	21.02 / 22.53
5	281	25.25 / 27.57

Table 3. Results when varying the number of message passing iterations in the implementation.

Starting Disparity Incr.	% Bad Matches w/ Error Threshold 0.0 / 1.0
1.0	15.6 / 7.44
1.5	17.0 / 7.30
2.0	17.1 / 7.59
3.0	17.8 / 7.17
4.0	19.1 / 9.62
6.0	22.0 / 8.00
8.0	24.1 / 8.84
12.0	24.8 / 7.47
16.0	22.5 / 10.8

Table 4. Results when varying the starting disparity candidate increments.

are shown in table 3 where the percent of bad matches represents the proportion of pixels where the integer calculated disparity value differs from the ground truth disparity value by a value greater than 0.

8.2. Adjustments in Disparity Increment Start With Constant Levels in Hierarchy

Next, we use the pyramid hierarchy with message propagation and hold the number of levels constant at 5, the number of iterations at each level constant at 10, and set the min and max disparities equal to 0 and 16, respectively. We adjust the starting disparity candidate increment between trials but continue to hold the proportional change in disparities between levels constant at 0.5, using the interpolation capabilities built into the GPU to retrieve the data costs at non-integer and sub-pixel disparities. The final calculated disparities are often non-integer values, but are rounded to integer values for final analysis. The results are shown in table 4.

Rate Incr. Adjusted	% Bad Matches w/ Error Threshold 0.5 / 1.0
0.4	16.5 / 11.4
0.5	14.9 / 9.56
0.6	14.8 / 8.08
0.7	13.8 / 6.44
0.8	24.2 / 7.54
0.9	17.1 / 8.44
1.0	11.7 / 4.86

Table 5. Results using a disparity candidate increment starting at 1.0 and adjusting the increment change between levels.

8.3. Adjusting the Disparity Increment Proportion Between Levels

The increment in disparity candidates has been cut in half when progressing down the hierarchy of levels in all experiments thus far. Now, we experiment with adjusting the rate at which the disparity candidate increment is adjusted between levels, holding the number of levels constant at 5 and the number of iterations per level constant at 10. In these experiments, we begin with disparity candidate increments of 1.0 and 2.0 and vary the rate at which the increment is adjusted between 0.4 and 1.0, with the rate representing the value that the increment at the previous level is multiplied by to retrieve the increment at the succeeding level.

In these experiments, the neighboring pixels may not have a message value that corresponds to each disparity candidate at each level, even when the candidate is within the pixel's disparity range. As a result, we use interpolation to retrieve the message value corresponding to the desired disparity. In addition, we stretch the disparity candidate sampling to the first 'whole' disparity candidate increment beyond the range in our sampling-invariant implementation, but then clamp the computations at the 'edge' samples to differ by no more than half the current disparity candidate increment of the 'target' candidate to prevent 'leaking' the data costs of the 'target' candidate into neighboring disparity candidates. We do not round the computed disparities in these experiments, allowing for results at the sub-pixel level. Our results show the percent of bad matches with error thresholds of 0.5 and 1.0. It should be noted that in the second set of trials where the disparity candidate increment begins at 2.0, the disparity increment remains above 1.0 when the increment adjustment rate is 0.9 and 1.0, making it impossible for the error to be below 0.5 for some pixels.

One peculiar result in each set of experiments is that the error rate increases between a disparity increment rate change of 0.8 and 0.9 and then drops dramatically when the disparity candidate increment rate change between levels is

Rate Incr. Adjusted	% Bad Matches w/ Error Threshold 0.5 / 1.0
0.4	17.1 / 11.1
0.5	16.0 / 9.58
0.6	15.3 / 7.46
0.7	25.2 / 8.03
0.8	38.3 / 8.10
0.9	33.1 / 15.9
1.0	67.2 / 6.25

Table 6. Results using a disparity candidate increment starting at 2.0 and adjusting the increment change between levels.

increased to 1.0 (when the disparity candidate increment is unchanged between levels). To explore this observation, we inspect the resulting disparity maps at the 0.9 and 1.0 rate change in the trials where the disparity candidate increment begins at 1.0. The resulting disparity maps are shown in figure 4. It appears that when the rate change is near 1.0 but not 'at' 1.0, the algorithm allows for too much flexibility in the disparity candidates between levels.

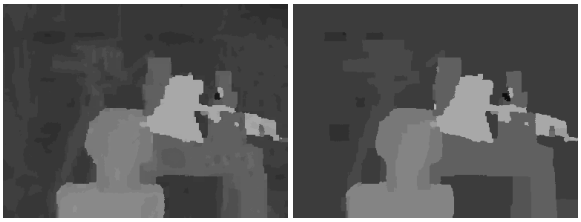

Figure 4. Computed disparity map resulting from running our implementation with a disparity candidate increment rate change of 0.9 (left) and 1.0 (right) between levels.

9. Subpixel Accuracy in Resulting Disparity Map

Next, we use our implementation to retrieve the resulting disparity maps at sub-pixel accuracy. A number of methods previously used to retrieve subpixel disparities are described and referenced by Scharstein and Szeliski [9]. Our implementation takes advantage of the interpolation capabilities of the GPU to retrieve the data costs at non-integer disparity candidates, treating a non-integer disparity candidate the same way as an integer disparity candidate without any change in implementation. Since the ground truth of the Tsukuba stereo set used in the previous experiments contains only integer disparities, we use the 'Venus' stereo set shown in figure 5 for the experiments in this section. The ground truth disparities in the 'Venus' stereo set range from 3.0 to 19.75 and are given in increments of one-eighth of a pixel.

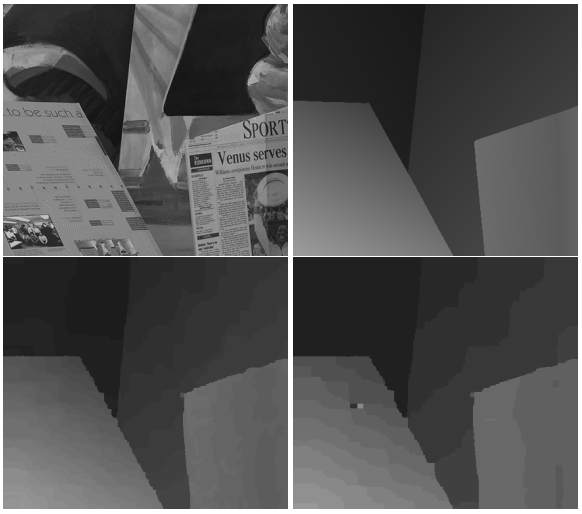

Figure 5. Top: Reference image (left) and ground truth disparity map (right) from the 'Venus' stereo set. Bottom: Resulting disparity maps from running one level of belief propagation with disparity increments of 0.5 (left) and 1.0 (right)

Disparity Incr.	% Bad Matches w/ Error Threshold 0.5 / 1.0
0.4	3.55 / 2.13
0.5	3.03 / 2.00
0.6	3.89 / 2.08
0.7	4.92 / 2.04
0.8	7.99 / 2.11
0.9	7.20 / 2.14
1.0	9.02 / 2.32

Table 7. Results of running a single level w / 200 iterations on the 'Venus' stereo set with varying disparity candidate increments.

9.1. Results Using a Single Level

First, we run our implementation at a single level with 200 iterations using disparity candidate increments ranging from 0.4 to 1.0. We do not perform any smoothing of the input images and the disparity candidate range extends from 0.0 to 20.0; otherwise the parameters are the same as in the Tsukuba implementation. The resulting disparity maps using disparity increments of 0.5 and 1.0 are shown in figure 5, and the results showing the percentage of 'bad' pixels beyond an error threshold of 0.5 and 1.0 are shown in table 7; note that these results exclude a 'border' region of 20 pixels around the resulting disparity map.

9.2. Results Using a Hierarchical Implementation

Next, we run our implementation using the 'cuboid' hierarchy described in section 4 with message propagation between levels using the minimum surrounding message value method described in section 7. We begin with a disparity

Number of Levels	% Bad Matches w/ Error Threshold 0.25 / 0.5 / 1.0
1	41.1 / 9.02 / 2.32
2	16.2 / 4.22 / 2.18
3	8.74 / 3.33 / 2.02
4	8.11 / 3.31 / 2.02

Table 8. Results of running a single level w / 200 iterations on the 'Venus' stereo set with varying disparity candidate increments.

Number of Levels	Ave. End-point Error	% Bad Matches w/ Error Threshold 0.1 / 0.5 / 1.0
3	0.314	94.9 / 10.7 / 3.89
4	0.238	82.9 / 6.78 / 1.90
5	0.188	66.0 / 6.37 / 1.01

Table 9. Results of running our motion estimation implementation using 3, 4, and 5 levels.

candidate increment of 1.0 and halve the increment in each subsequent level; the storage requirements remain constant regardless of the number of levels. We run the implementation using 1, 2, 3, and 4 levels, with disparity candidate increments at the bottom level of 1.0, 0.5, 0.25, and 0.125, respectively. The results showing the percent of 'bad' pixels beyond error thresholds of 1.0, 0.5, and 0.25 are shown in table 7; these results exclude a 'border' region of 20 pixels on each edge of the resulting disparity map. The use of multiple levels of decreasing disparity candidate increments improves the results over a single level with a disparity candidate increment of 1.0, but none of these results are as accurate as the result presented in section 9.1 when the disparity candidate increment is initially set to 0.5. However, the disparity candidate search space and overall storage requirements in the implementation presented in this section are half that of the aforementioned single-level implementation that retrieved a slightly more accurate disparity map.

10. Using Hierarchical Implementation for Motion Estimation

Belief propagation can been used for motion estimation as well as stereo, as shown by Isard and MacCormick [6] and Grauer-Gray et al. [5]. However, the presence of a two dimensional motion candidate search space often leads to larger and possibly infeasible storage requirements when attempting traditional belief propagation for motion estimation. In this section, we show how our hierarchical belief propagation implementation makes motion estimation feasible.

We run our implementation on the 584 X 388 Dimetrodon image pair presented by Baker et al. [2] using the Manhattan distance between assigned motion vectors for the discontinuity cost. The first frame of the pair, the ground truth motion, and a legend showing the color coding of the motion is shown in figure 6. The movement present in this image pair is not that large in magnitude, but fine precision at the subpixel level is required for accurate results. We run our 'cuboid' hierarchy implementation described in section 4 using a motion range from -5.0 to 5.0 on the x and y axis and message propagation between levels using the minimum surrounding message value as described in section 7. The interval between each motion candidate is set to 2.5 in the top level, resulting in 5 possible motions on each axis for a total search space of 25 possible motions, and is halved with each subsequent level. We run the implementation using 3, 4, and 5 levels with 200 message passing iterations per level, a maximum data cost of 15.0, a maximum discontinuity cost set to infinity, and a data cost weight of 0.07. The resulting motion using 5 levels is shown in figure 6, and the results for each trial using the error in flow endpoint measure described in [2] is shown in table 9. Our results are closer to the ground truth than the results corresponding to the motion algorithms described in [2] for the Dimetrodon pair (see http://vision.middlebury.edu/flow/), but more experiments are necessary to benchmark our implementation in relation to the current state-of-the-art in motion estimation; one particular challenge in such a task is maintaining a constant set of parameters across image sets with different magnitudes of motion.

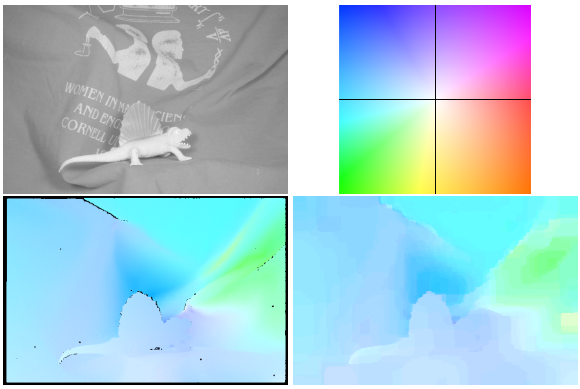

Figure 6. Top: First frame of Dimetrodon image pair (left) and legend showing color coding of motion (right) Bottom: ground truth motion (left) and computed motion using implementation with 5 levels (right).

11. Conclusions and Future Work

We have presented a method to reduce the storage requirements of a stereo or motion estimation belief propagation implementation without sacrificing much in terms of the accuracy of the results, as well as how our GPU im-

plementation can be used without modification to retrieve sub-pixel accuracy in the resulting disparity map or motion vectors. For image sets with a large disparity/motion range or with sub-pixel accuracy requirements, our belief propagation implementation may be used where traditional belief propagation is not feasible due to overwhelming storage requirements.

In the future, we plan to further explore tweaking the parameters and algorithm details to retrieve the optimal balance of storage requirements and running time versus accuracy in the results as well as explore other options to improve the results on different sets of images without increasing the storage requirements of the implementation. One possibility is to incorporate the gradient constancy assumption as presented by Zhang and Negahdaripour in [11] into our implementation; this addition has been shown to increase the robustness of a related belief propagation implementation. Finally, we plan to investigate how the ideas presented in this paper could be incorporated into related work that uses belief propagation, including the tile-based approach discussed in section 3 as well as the some of the top-scoring algorithms in the Middlebury stereo evaluation.

12. Acknowledgements

This work was partially supported by a U.S National Aeronautics and Space Administration award NASA NNX08AD80G under the ROSES Applied Information Systems Research program.

References

[1] *NVIDIA CUDA Programming Guide: Version 2.3.1*. NVIDIA Corporation, August 2009.

[2] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, Oct. 2007.

[3] A. Brunton, C. Shu, and G. Roth. Belief propagation on the gpu for stereo vision. In *Computer and Robot Vision, 2006. The 3rd Canadian Conference on*, pages 76–76, June 2006.

[4] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. *Int. J. Comput. Vision*, 70(1):41–54, 2006.

[5] S. Grauer-Gray, C. Kambhamettu, and K. Palaniappan. Gpu implementation of belief propagation using cuda for cloud tracking and reconstruction. In *2008 IAPR Workshop on Pattern Recognition in Remote Sensing (PRRS 2008)*, pages 1–4, 2008.

[6] M. Isard and J. Maccormick. Dense motion and disparity estimation via loopy belief propagation. Technical report, 2005.

[7] C.-K. Liang, C.-C. Cheng, Y.-C. Lai, L.-G. Chen, and H. H. Chen. Hardware-efficient belief propagation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 80–87, 2009.

[8] R. Y. S. W. M. L. Q. Yang, L. Wang and D. Nister. Real-time global stereo matching using hierarchical belief propagation. In *British Machine Vision Conf.*, page 989998, 2006.

[9] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3):7–42, 2002.

[10] J. Sun, N.-N. Zheng, and H.-Y. Shum. Stereo matching using belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(7):787–800, 2003.

[11] H. Zhang and S. Negahdaripour. Integrating bc & gc models in computing stereo disparity as markov random field. *Journal of Multimedia*, 2006.