Stereo Correspondence by Dynamic Programming on a Tree

Olga Veksler
University of Western Ontario
Computer Science Department, Middlesex College 361
London ON N6A 5B7 Canada
olga@csd.uwo.ca

Abstract

Dynamic programming on a scanline is one of the oldest and still popular methods for stereo correspondence. While efficient, its performance is far from the state of the art because the vertical consistency between the scanlines is not enforced. We re-examine the use of dynamic programming for stereo correspondence by applying it to a tree structure, as opposed to the individual scanlines. The nodes of this tree are all the image pixels, but only the "most important" edges of the 4 connected neighbourhood system are included. Thus our algorithm is truly a global optimization method because disparity estimate at one pixel depends on the disparity estimates at all the other pixels, unlike the scanline based methods. We evaluate our algorithm on the benchmark Middlebury database. The algorithm is very fast, it takes only a fraction of a second for a typical image. The results are considerably better than that of the scanline based methods. While the results are not the state of the art, our algorithm offers a good trade off in terms of accuracy and computational efficiency.

1 Introduction

Stereo correspondence is an inherently ambiguous problem. To reduce ambiguities, most algorithms make assumptions about scene geometry. One reasonable and commonly made assumption is that a scene is piecewise smooth. This implies that the recovered stereo correspondence map d should also be piecewise smooth.

There are many ways to implement the requirement of a piecewise smooth disparity map, see [19] for an excellent review of stereo algorithms. Using the terminology of [19], one popular approach is called *global optimization* and it is based on directly optimizing an objective function.

In *global optimization*, the constraints on the disparity map d are formulated into an objective function *E(d)* which is then minimized over all image pixels. A typical objective function has the following form:

$$E(d) = E_{data}(d) + \lambda E_{smooth}(d). \quad (1)$$

The data term, E_{data} penalizes any disagreement of the disparity map d with the observed data. That is E_{data} is small if the pixels that d puts in correspondence have similar intensities, and large if the pixels that d puts in correspondence differ significantly in intensities. The smoothness term, E_{smooth} measures the extent to which d is not piecewise smooth. For computational feasibility, in most methods E_{smooth} measures the difference in disparity only between the pixels that are the nearest neighbours on the 4 connected pixel grid.

Except for some special cases [18, 13], optimizing the energy function in 1 is NP-hard. We will break the global optimization methods in two groups: the *1D* optimization methods and the *2D* optimization methods.

The *1D* optimization methods [1, 16, 8, 19] can be seen as drastically simplifying the objective function in equation (1). They enforce piecewise smoothness only in the horizontal direction, and so the optimization is reduced to one dimension. That is the $E_{smooth}(d)$ does not contain any terms based on neighbouring pixels in the vertical direction. Assuming that there are n scanlines, the energy function in equation (1) can be written as a sum of n energy functions, one for each scanline, and each one can be optimized separately from the others. This optimization can be performed efficiently and exactly using dynamic programming.

We should clarify that traditionally the methods which we here call *1D* optimization methods are called the dynamic programming methods, and that these methods typically start by formulating the objective function on an individual scanline[1], without ever formulating a global objective function of the type in equation (1). However any such dynamic programming method can be trivially reformulated with a global objective function, and it is conve-

[1] The Scanline Optimization method of [19] was the first method which directly formulated energy function on an individual scanline, as far as we are aware.

nient for us to refer to them as *1D* optimization methods to emphasise the one dimensional nature of the optimization that they perform. The *1D* optimization methods are not truly global optimization methods because the disparity estimate at a pixel depends only on the disparity estimate of pixels on the same scanline, but is completely independent of the disparity estimates on the other scanlines.

The performance of *1D* optimization methods is far from the state of the art [19], since piecewise smoothness in enforced only in the horizontal direction. The most noticeable artifact which distinguishes the resulting disparity maps of such methods is the horizontal "streaking" which results from the lack of coherence in the vertical direction. Most methods [16, 2, 4] try to improve results by post processing between the scanlines, with various degrees of success. The advantage of the *1D* optimization methods is that they are simple to implement and are efficient.

The *2D* optimization methods enforce piecewise smoothness in both horizontal and vertical directions. Traditional *2D* optimization approaches to approximate d include simulated annealing [12], continuation methods [3], mean-field annealing [11]. While interesting from the theoretical point of view, these methods are rather inefficient. Recently, graph-cuts [6, 15] and belief propagation methods [20, 9] have been applied quite successfully to optimize equation (1). These methods are relatively efficient and produce excellent results according to the recent stereo evaluation on data with ground truth conducted by [19]. Still these methods are far from real time, and their theoretical complexity is not quite clear because they are iterative.

The motivation behind our work is to use the powerful and efficient optimization tool provided by dynamic programming, but apply it to a structure more suited to enforce piecewise continuity than a scanline. Dynamic programming can be applied to graphs without loops, in particular, to trees [17]. Dynamic programming on a tree is almost as efficient as that on a one dimensional array.

It is best to minimize the energy in equation (1) on a *2D* grid, but it an NP-hard problem in general. If we think of *1D* optimization algorithms as using a collection of scanlines to approximate a grid, then this is an obviously poor approximation. A tree structure is a significantly better approximation to a *2D* grid. The first (small) advantage is that a tree contains $m - 1$ more edges of the original grid than the collection of scanlines, where m is the number of rows. The second, more important, advantage is that since a tree structure is connected, the estimate of disparity at one pixel depends on the estimate of disparity at all the other pixels. Thus dynamic programming on a tree is a truly global optimization algorithm. Contrast this with the individual scanlines approximation to a grid, where the disparity at one pixel depends only on pixels at the same scanline. The last, and the most important advantage is that out of huge number of possible tree structures we can choose the tree structure which contain "most important" edges of the grid. Contrast this again with the individual scanline approximation, where there is no choice but to take all the horizontal edges.

Our algorithm is not a *1D* optimization method because it operates across both vertical and horizontal dimensions. However it is also not a true *2D* optimization method, because it operates only among the chosen directions in the 2 dimensions, that is the directions given by the tree structure.

Our algorithm is simple to describe and implement. We start with modelling an objective function of the type in equation (1). Ideally, it has to be optimized on a graph structure which is a grid of pixels. However dynamic programming is not applicable to a grid. Thus we remove the "least important" edges from this grid until the remaining graph is a tree, and then we apply the dynamic programming to the resulting tree. The most important and interesting question is what are these "least important" edges for our problem. We present possible approaches to this question in section 3. For now we just say that these are the edges between pixels which are less likely to have the same disparity.

To implement dynamic programming efficiently, we use the methods developed by [10]. Typically, if a tree has m nodes and the number of possible disparity values is h, then the straightforward dynamic programming takes $O(nh^2)$ time. However for a certain restricted but still quite useful type of energy functions, the running time can be reduced to $O(nh)$, which is the complexity of our method.

Recently, tree structures have been used for energy minimization in tree-reweighted message passing, see, for example [21, 14]. These approaches are quite different from our work, they are based on iteratively passing tree-reweighted messages, and in certain cases there are some optimality guarantees. Our approach is much simpler, but also much more efficient.

We evaluate our algorithm on the benchmark Middlebury database. The results fall in the middle range, as expected. The state of the art results are given by the more computationally costly *2D* optimization algorithms. Our results are by far better than those of methods based on *1D* optimization. The running time is excellent, just a fraction of a second for the images in this database. Thus our algorithm should be suitable for a real time implementation.

This paper is organized as follows. We start in section 2 by explaining how dynamic programming can be implemented efficiently using methods in [10] . In section 3 we discuss how we choose a tree structure. In section 4 we present our experimental results. Future work is in section 5.

2 Efficient Dynamic Programming on a Tree

In this section, we first describe the energy function we can optimize, then we show how to optimize it exactly with dynamic programming on a tree, and lastly we show how we use methods in [10] to significantly reduce the complexity for a restricted types of energy functions.

2.1 Energy Function

Let $G(V, E)$ be a tree graph with vertices V and edges E. By definition of a tree, vertices are connected and there are no cycles. All pixels of the left image form the vertices in V. For the edges E, we choose only a subset of the standard 4-connected grid formed by the edges between the pixels which are the nearest neighbours in the image. This subset is chosen so that it forms a tree, we give details in section 3.

We can now write the energy in equation (1) that we optimize on a tree structure more explicitly. We match pixels in the left image to the pixels in the right image. Thus our setup is not symmetric, unlike most scanline based dynamic programming algorithms. We also do not handle the uniqueness and ordering constraints.[2] This is a small price to pay for the much improved accuracy. Let p be a pixel in the left image and d_p be the value of disparity map d at pixel p. Let $m(d_p)$ be the matching penalty for assigning disparity d_p to pixel p. For example, $m(d_p)$ can be the absolute difference between the pixel p in the left image, and pixel p shifted by d_p in the right image. In our framework, $m(d_p)$ can be arbitrary. We describe the $m(d_p)$ that we actually use in section 4. The data term in equation (1) can be now written as $\sum_{p \in V} m(d_p)$.

Let $s(d_p, d_q)$ be the smoothness penalty for assigning disparities d_p and d_q to p and q which are connected by an edge in our graph. To enforce smoothness on the disparity map, $s(d_p, d_q)$ should be a monotonically nondecreasing function in the absolute disparity difference $|d_p - d_q|$. To preserve discontinuities in the disparity map, $s(d_p, d_q)$ should not grow too big so that the penalty for a large discontinuity is not prohibitively heavy. In our framework, $s(d_p, d_q)$ can be arbitrary, but some choices of $s(d_p, d_q)$ lead to more efficient implementation. We will further discuss our choice for $s(d_p, d_q)$ in section 2.3. The smoothness term for the whole image can be now written as $\sum_{(p,q) \in E} s(d_p, d_q)$. Thus the energy function that we seek to optimize is given by equation (2).

$$E(d) = \sum_{p \in V} m(d_p) + \lambda \sum_{(p,q) \in E} s(d_p, d_q). \qquad (2)$$

[2] Loosing the ordering constraint is actually not a loss, in our opinion, since the ordering constraint is violated in practice. Most dynamic programming algorithms use the ordering constraint for efficient handling of symmetric stereo correspondence formulation.

2.2 Optimization with Dynamic Programming

Dynamic programming on a tree is a trivial generalization of dynamic programming on a linear array structure. We follow [10] in describing how it can be done. Let $r \in V$ be the root vertex of our tree. Obviously the minimum of the energy in (2) is independent of the choice of r. Since the dynamic programming algorithm presented in this section finds the minimum of the energy in 2, any choice of r leads to the same solution.[3] Let depth of the r be 0, and depth of all other $v \in V$ be the number of edges between the root r and v on the shortest path between r and v.

Each node v, except the root, has a parent $p(v)$, and the depth of $p(v)$ is equal to the depth of v minus 1. If node v is not a root, then the minimum value of the energy in (2) on the subset of the graph consisting of a subtree rooted at v and the edge between v and $p(v)$ can be written recursively as a function of $d_{p(v)}$ (the depth assigned to $p(v)$):

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

where C_v is the set of children of v. Let $L_v(d_{p(v)})$ be the optimum disparity assignment to v as a function of $d_{p(v)}$. It can be defined by replacing min with $argmin$ in equation (3).

The optimal disparity assignment for the root node r can be written as

$$L_r^* = arg \min_{d_r \in D} \left(m(d_r) + \sum_{w \in C_r} E_w(d_r) \right). \qquad (4)$$

If v is a leaf node (that is the node without children), then C_v is empty. Therefore for a leaf v, functions E_v and L_v are not recursive and can be evaluated directly. Let M be the maximum depth in the tree. The optimization of the energy in equation (2) starts by evaluating the functions E_v and L_v for each node v at depth M. Now we can evaluate E_v and L_v for all the nodes at depth $M - 1$ because any child w of such a node has depth M, and therefore we have already evaluated E_w and L_w at the previous step. We proceed evaluating E_v and L_v in order of decreasing depth until the root is reached. Once the root is reached, we can compute its optimal disparity assignment. Then we use the optimal value at the root and go down the tree in order of increasing depth, computing the optimal disparity assignments for each node, using the already computed functions L_v.

If h is the size of set D, then computing L_v and E_v takes $O(h^2)$ time each, because for each possible value of $d_{p(v)}$ we have to cycle over all possible values of d_v when searching for minimum in equation (3). Thus the overall complexity is $O(h^2 n)$, where n is the number of nodes in the tree.

[3] Unless the global minimum is not unique, in which case choice of different roots could lead to different solutions, but each of these solutions gives a global minimum of the energy function.

The *1D* optimization methods also have theoretical complexity $O(h^2 n)$. However dynamic programming on a tree is slightly slower in practice, because tree traversal is less efficient than an ordered array traversal.

2.3 Improving Efficiency

For larger h, complexity $O(h^2 n)$ is not desirable. We use the methods in [10] to significantly reduce the complexity of dynamic programming for a restricted types of energies. In particular, consider the following smoothness penalty:

$$s_P(p_d, q_d) = \begin{cases} 0 & \text{if } d_p = d_q \\ w_{pq} & \text{otherwise} \end{cases} \quad (5)$$

This $s_P(p_d, q_d)$ is often used for energy based stereo correspondence [6, 9], and is a simple discontinuity preserving smoothness term. We use subscript P to denote that $s_P(p_d, q_d)$ comes from the Potts model in Markov Random Fields. There is no penalty if neighbouring pixels are assigned the same disparity. If neighbouring pixels are assigned different disparities, then there is a fixed penalty w_{pq} which is independent of $|d_p - d_q|$. This penalty may, however, depend on the individual pixels p and q as expressed by coefficients w_{pq}. We have used s_P to evaluate our algorithm, although any energy function of the type in equation (2) can be used. Note that the efficiency of dynamic programming can be improved for more general energy functions [10].

Let $E^P(d)$ be the energy function in equation (2) with the smoothness penalty given by $s_P(p_d, q_d)$ in equation (5). For this energy function, computation of L_v and E_v in equation (3) can be reduced from $O(h^2)$ time to $O(h)$ time. For general energy functions $E(d)$, given fixed disparity value of the parent node d_{p_v} we have to search over all possible values of d_v to find the one which minimizes $E_v(d_{p(v)})$. Consider now the energy function given by $E^P(d)$. The smoothness penalty $s_P(d_{p(v)}, d_v)$ is binary, it is either 0 or a constant independent of $d_{p(v)}$ and d_v. Therefore the optimum disparity for the child v can be found in constant time, without searching over all possible range of disparity values in D. We first compute is the disparity value d_{v^*} which minimizes $m(d_v) + \sum_{w \in C_v} E_w(d_v)$. Note that d_{v^*} is independent of $d_{p(v)}$ and can be computed in $O(h)$ time. Then for each possible disparity value of a parent, $d_{p(v)}$, there are only two choices for the optimum disparity assignment for v. First choice is the disparity of the parent, $d_{p(v)}$, in which case $s_P(d_{p(v)}, d_v) = 0$ in equation (3). The second choice is d_{v^*}, in which case $s_P(d_{p(v)}, d_v) \neq 0$, unless $d_{p(v)} = d_{v^*}$.

Thus for each node v, both E_v and L_v are computed in $O(h)$ time, and the total complexity of dynamic programming reduces to $O(nh)$, a considerable saving for larger h.

3 Choosing a Tree Structure

The biggest advantage of our algorithm over *1D* optimization is that we get to choose the subset of edges over which to optimize with dynamic programming. This allows to choose the most "important" edges for our tree, rather than being limited to all the horizontal edges, as in *1D* optimization. This section explains how we find these most "important" edges. We discuss two possible tree choices, named, respectively, the MID tree and the MIDDT tree.

3.1 MID Tree

By including an edge between pixels p and q in our tree, we are enforcing the constraint that pixels p and q should have similar disparities. Since we are limited to a tree structure, unlike the *2D* optimization methods, we must choose a subset of edges that link the pixels which are most likely to have the same disparity a priori. To find such pixels, we make use of the intensity information provided by the left image. Let $I(p)$ denote the intensity of pixel p in the left image. If neighbouring pixels p and q have similar intensity values $I(p)$ and $I(q)$, then they are more likely to have the same disparity a priori. This is because disparity discontinuities tend to align with intensity discontinuities.

Thus the first approach to choosing a tree structure is quite simple but works surprisingly well. Let $G' = (V, E')$ be a graph with vertices V consisting of all the image nodes and edges E' consisting of all the edges between the nearest neighbouring pixels, that is E' is simply the standard 4 connected grid. For each pair of neighbouring pixels p and q, assign weights $v_{pq} = |I(p) - I(q)|$ to the edge between p and q. Construct the minimum spanning tree of G' and let that tree to be the tree structure for our algorithm.[4] Let us call such a tree the minimum intensity difference tree (abbreviated the MID tree), because it is the tree with the minimum sum of intensity differences across the edges. Since edge weights v_{pq} are integers in a small range, the edge sorting can be performed in linear time. Therefore the minimum spanning tree can be computed in basically linear time, see any standard algorithms book, for example [7].

Let us come back to our smoothness penalty $s_P(d_p, d_q)$ in equation (5). For stereo correspondence, the weight coefficient w_{pq} is often made to be a monotonically decreasing function of intensity difference $|I(p) - I(q)|$. This is done with exactly the purpose of reflecting the fact that disparity discontinuities tend to align with intensity discontinuities. Thus the larger is the difference $|I(p) - I(q)|$, the smaller is $s_P(d_p, d_q)$. Consider a *2D* optimization problem on a grid with the energy as in equation (2) and w_{pq} set to be a monotonically decreasing function of intensity difference

[4] The minimum spanning tree is simply the tree that connects all the vertices of G' and the sum of its weights is minimum out of all such trees.

$|I(p) - I(q)|$. We can regard a tree structure as an approximation to the the *2D* grid. Then our MID tree chooses the edges of the maximum weight as an approximation to the *2D* grid. This is reasonable because edges with smaller weights w_{pq} contribute less to the energy, and if we have to remove any edges from the grid, the edges with smallest w_{pq} should be discarded first.

3.2 MIDDT Tree

Since the number of edges is by far larger than the number of intensities, in any image, there are large sets of edges with the same absolute intensity difference. Thus there are large subsets of edges which have equal weights $v_{pq} = |I(p) - I(q)|$. Therefore the MID tree is usually not unique, and the particular MID chosen for our algorithm will depend on the implementation details, rather than some meaningful criterion.

This leads us to a more interesting choice of a tree structure. For the MID tree, we used our prior belief which was based on a purely local criterion: if two pixels have similar intensities, then they are likely to have similar disparities. We can extend this criterion to a less local one: the deeper pixels p and q are inside a homogeneous intensity region, the more likely they are to have the same disparity. This prior belief is based on observation that disparity discontinuity is less likely to lie across a homogeneous intensity region. Of course we have to somehow define "deep inside a homogeneous region".

We use distance transform [5] to measure how deep a pixel p is inside a homogeneous intensity region. Let t be a threshold on intensity difference. Let B be the set of all pixels in the image which have the absolute intensity difference with one of its neighbours larger than t. That is

$$B = \{p \mid \ |I(p) - I(q)| > t \ \text{ for some neighbour } \ q\}.$$

Thus the set B consists of pixels which are not inside a homogeneous intensity region, and so B includes pixels which are right on the border of a homogeneous intensity region. Now for a pixel p, define

$$\mathcal{D}(p) = \min_{q \in B} dist(p, q), \tag{6}$$

where $dist(p, q)$ is the distance between pixels p and q. We use the Manhattan or block distance, which is just the absolute difference in coordinates of p and q. In this case, $\mathcal{D}$ can be computed particularly efficiently. For all pixels, $\mathcal{D}$ can be computed in just two passes over the image [5].

If $\mathcal{D}(p)$ is large, then it is inside a homogeneous intensity region. Let $M = \max_p \mathcal{D}$. Define edge weights $u_{pq} = M - \frac{1}{2}(\mathcal{D}(p) + \mathcal{D}(q))$ and use them instead of weights v_{pq} for the minimum spanning tree[5] algorithm in section 3.1.

[5]abbreviated MST

In practice, however, weights u_{pq} work worse than v_{pq} because of the sensitivity to the parameter t. Suppose t is too large. Consider the extreme case when t is equal to the maximum intensity difference observed in the image, Suppose that there is only one edge with this maximum intensity difference. Then set B contains only 2 pixels (those with the maximum intensity difference between them), and weights u_{pq} measure the average distance of p and q to the two pixels in B, completely ignoring all the other intensity information in the image.

If t is small, weights u_{pq} are reasonable, but there are many edges with weight 0. Thus, again, there are many possible minimum T_u, and a particular choice will depend on implementation rather than some meaningful criterion.

We found that it is best to combine edge weights v_{pq} and u_{pq} as follows. Let v_{pq} be the most important factor and u_{pq} be the secondary factor used to break ties when v_{pq} is equal for two or more edges. Let $k = \max_{(p,q)\in E'} u_{pq}$. Define the combined edge weights as

$$c_{pq} = k \cdot v_{pq} + u_{pq} \tag{7}$$

Thus when searching for the next edge to insert in the spanning tree, we first select edges with the smallest value of v_{pq} (that is the edges with the smallest absolute intensity difference) and any ties are broken by weights u_{pq}, that is the edges deeper inside a uniform region are given a preference. We call the resulting MST as MIDDT tree, because we first use the intensity difference information, and then the distance transform information.

4 Experimental Results

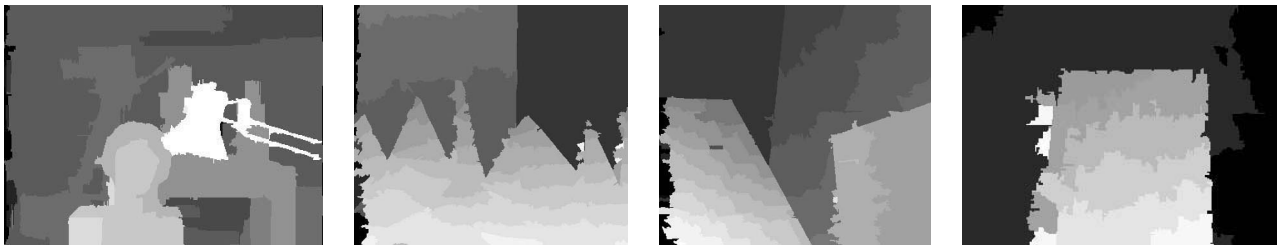
For the matching term we used

$$m(d_p) = min\left\{|L(p) - R(p - d_p)|, \tau\right\},$$

where $L(p)$ is the intensity of pixel p in the left image, and $R(p - d_p)$ is the intensity of pixel p shifted by disparity d_p in the right image, and τ is a parameter used to make $m(d_p)$ robust to outliers. In practice, we set $\tau = 10$. Our smoothness term, $s_P(d_p, d_q)$ is in equation (5). The parameter λ in equation (2), which measures the relative importance between E_{data} and E_{smooth} was set to 130. For the MID version of the algorithm, there are no additional parameters to set. For the MIDDT version, we used $t = 6$.

We have tested our algorithm on the benchmark Middlebury database [19].[6] Figure 2 shows the accuracy of our algorithms along with the rank at the time of evaluation in the square brackets. There are 4 stereo pairs, named *Tsukuba*, *Sawtooth*, *Venus* and *Map*, respectively. The error is computed as the percentage of pixels far from the true disparity

[6]We used the colored images, except the *Map*, for which colored image is not available.



(a) Tsukuba (b) Sawtooth (c) Venus (d) Map

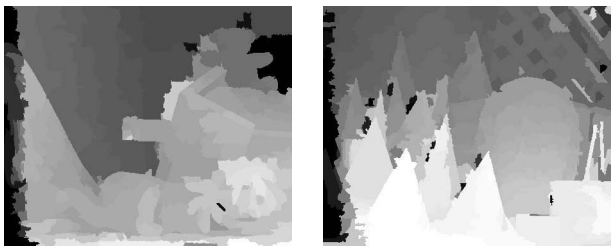
Figure 1.

by more than 1. These statistics are collected for all unoccluded image pixels (shown in column *all*), for all unoccluded pixels in the untextured regions (shown in column *untex.*), and finally for all unoccluded image pixels close to a disparity discontinuity (shown in column *disc.*).

As expected, the results fall in the middle range. The MIDDT algorithm gives better results, and was submitted officially to the Middlebury database The rank of MIDDT is 20 out of 36 algorithms currently in the database. The disparity maps for the MIDDT algorithm are in figure 1. There is very little horizontal "streaking" in the results, but there is also a little vertical "streaking" because there is no bias to either horizontal or vertical directions.

Our algorithm is very fast however, it runs in a fraction of a second for all the images in this database, whereas the algorithms which do give the state of the art performance are much slower and most of them are iterative. Our algorithm should be a good candidate for real time implementation. The Middlebury database has 2 new scenes which have much more complex geometry, called *teddy* and *cones*. These scenes are not included into the official evaluation yet. Our results (computed on our own by excluding the occluded pixels) are: for the *teddy* scene 14.26% and 8.9% error with threshold of 1 and 2, respectively; for the *cones* scene 12.81% and 8.23% error with threshold of 1 and 2, respectively. There are 4 methods based on *1D* optimization in the evaluation table, and by a coincidence they have consecutive ranks 25 to 28, which is almost at the very bottom of the table. This direct comparison of our method to the *1D* optimization methods may not be fair. Each of the *1D* optimization methods uses a different cost function and different types of post processing to alleviate the "horizontal streaking" artifacts. Thus the difference in performance between our method and the *1D* optimization methods could possibly be due to the differences in the cost functions.

To evaluate our method on an equal footing with the *1D* optimization, we implemented *1D* scanline optimization with our objective function, but optimized it on the scanlines instead of the tree. Now the objective functions



(a) Teddy (b) Cones

Figure 4.

are identical (up to the parameters), and any difference in the results is due to the fact that we optimize the objective function on the tree, rather than the scanlines. For our algorithm, we chose the same parameters for all the images in the database, and of course they were chosen to optimize the results. For the *1D* optimization, even though we are using the same objective function, a different set of parameters will optimize the results. To make our comparison more favourable to the *1D* methods (and to decrease the time for manual parameter search), for each image in the Middlebury database we found the best parameters separately. This would not be admissible if we were planning to submit the results of *1D* optimization method to the database officially. However it is perfectly acceptable for our purposes, since we create a bias in favour of *1D* optmization methods, but still will be able to conclude that our method is more accurate. The results for this *1D* optimization algorithm are in figure 3, we only show the error for all unoccluded pixels, so these numbers should be compared with our results in figure 2 under column *all*. If submitted, the algorithm would get rank 27. For all stereo pairs, except the *Map*, the *1D* optimization performs significantly worse than our algorithm. For the *Map* images, we were able to tune the parameters to get a slightly better accuracy than that of both MID and MIDDT algorithms. However the range of these parameters is far off the range from the other stereo pairs. The *Map*

	Tsukuba			Sawtooth			Venus			Map	
Algorithm	all	untex.	disc.	all	untex.	disc.	all	untex.	disc.	all	disc.
MID	2.17 [14]	0.66[8]	11.53[14]	1.59 [19]	0.93 [22]	8.82 [18]	1.39 [11]	1.39 [10]	7.59 [11]	1.32 [20]	11.44 [23]
MIDDT	1.77 [13]	0.38 [5]	9.48 [14]	1.44 [21]	0.84 [25]	6.87 [17]	1.21 [11]	1.41 [13]	5.04 [6]	1.45 [23]	13.00 [27]

Figure 2. Our results on Middlebury Database

	Tsukuba	Sawtooth	Venus	Map
Best Parameters	$\lambda = 12, \tau = 12$	$\lambda = 19, \tau = 15$	$\lambda = 21, \tau = 11$	$\lambda = 82, \tau = 19$
all error	4.98	6.68	3.83	0.96

Figure 3. *1D* optimization results on Middlebury Database

stereo pair is highly textured, and that is the reason why *1D* optimization performs well on it, there is no need for disambiguation by using vertical smoothness constraint. The *1D* optimization runs approximately 3 times faster in our implementation because traversing a linear array is faster than traversing a tree.

5 Future Work

In the future, we plan on further investigating the tree structure selection. In particular, we are interested in using results from image segmentation for selecting a tree structure. Image segmentation algorithms give a collection of homogeneous intensity segments, thus we can use distance transform from segment border as a tree weight. Segmentation algorithms may give more robust results than our simple MIDDT approach. This also would provide an interesting link between our algorithm and segmentation based stereo correspondence algorithms.

References

[1] H. Baker and T. Binford. Depth from edge and intensity based stereo. In *IJCAI81*, pages 631–636, 1981.

[2] S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. *IJCV*, 35(3):1–25, December 1999.

[3] A. Blake and A. Zisserman. *Visual Reconstruction*. MIT Press, 1987.

[4] A. Bobick and S. Intille. Large occlusion stereo. In *Vismod*, 1999.

[5] G. Borgefors. Distance transformations in digital images. *CVGIP*, 34(3):344–371, June 1986.

[6] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision*, pages 377–384, 1999.

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

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

[9] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. In *CVPR04*, pages I: 261–268, 2004.

[10] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 61(1):55–79, January 2005.

[11] D. Geiger and F. Girosi. Parallel and deterministic algorithms from MRF's: Surface reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):401–412, May 1991.

[12] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[13] H. Ishikawa. Exact optimization for markov random fields with convex priors. *PAMI*, 25(10):1333–1336, October 2003.

[14] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. In *Tenth International Workshop on Artificial Intelligence and Statistics*, January 2005.

[15] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. In *ECCV02*, page III: 82 ff., 2002.

[16] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search. *TPAMI*, 2:449–470, 1985.

[17] L. Rabiner and B. Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1993.

[18] S. Roy. Stereo without epipolar lines: A maximum-flow formulation. *IJCV*, 34(2-3):147–161, August 1999.

[19] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1-3):7–42, April 2002.

[20] J. Sun, N. Zheng, and H. Shum. Stereo matching using belief propagation. *PAMI*, 25(7):787–800, July 2003.

[21] M. Wainwright, T. Jaakkola, and A. S. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on Information Theory*, 45(9):1120–1146, May 2001.