Stereo Correspondence with Compact Windows via Minimum Ratio Cycle

Olga Veksler
NEC Research Institute, 4 Independence Way Princeton, NJ 08540
olga@research.nj.nec.com

Abstract

One of the earliest and still widely used methods for dense stereo correspondence is based on matching windows of pixels. The main difficulty of this method is choosing a window of appropriate size and shape. Small windows may lack sufficient intensity variation for reliable matching, while large windows smooth out disparity discontinuities. We propose an algorithm to choose a window size and shape by optimizing over a large class of "compact" windows. The word compact is used informally to reflect the fact that the ratio of perimeter to area of our windows is small. We believe that this is the first area based method which efficiently constructs non rectangular windows. Fast optimization over compact windows is achieved via the minimum ratio cycle algorithm for graphs. The algorithm has only a few parameters which are easy to fix.

Index Terms — Stereo correspondence, adaptive windows, compact windows, minimum ratio cycle, graph algorithms.

1 Introduction

Area based matching is one of the oldest and still widely used approaches to dense stereo correspondence [11, 12, 13, 7, 6]. This approach makes a reasonable assumption that a pixel is surrounded by a patch of pixels which have approximately the same disparity. Thus to estimate how likely disparity d is for pixel p, a window of pixels centered at p in the left image is overlapped with the same window shifted by d[1] in the right image. Then the cost between the two windows is computed using, for example, sum of squared differences, normalized

[1]Here we assume that the stereo pair is rectified, so that d has dimension 1.

correlation, etc. See [16] for comparisons between different window costs. In the end, pixel p gets assigned the disparity d which gives the best window cost.

While the assumption that each pixel is surrounded by a patch at approximately the same disparity is usually valid, the shape and size of that patch is unknown beforehand, of course. Ignoring this, most methods use a rectangular window of fixed size. In this case the implementation is very efficient: the running time is linear in the number of pixels times the number of disparities, i.e. it is independent of the window size [3].

The popularity of window based correspondence is due in part to its efficiency and ease of implementation. However there is a well known problem with this method, which was realized by researchers as early as [11]. For a reliable estimate a window must be large enough to include enough intensity variation, but at the same time small enough not to cross a disparity discontinuity,[2] so that the assumption that the window contains only pixels at approximately the same disparity remains valid. This means that different pixels in the same image usually require windows of different sizes: for a small window the results are unreliable in low texture areas; as the window size is increased, the results in low texture areas become more reliable while disparity boundaries get increasingly blurred. In addition, no fixed window shape works well for all pixels. Pixels that are near a disparity discontinuity frequently require windows of different shapes to avoid crossing that discontinuity.

There are relatively few algorithms which vary window size or shape. Their common weakness is that they use naive optimization methods for the best window shape search. Previous methods can be roughly divided into two groups, according to the optimization method used. First there are methods such as [12, 9, 15] which use greedy local search. These algorithms improve an initial window estimate by expanding a window in a certain direction until a local maximum is reached. This is not only suboptimal but also quite slow. In the second group are the direct search methods [5, 4, 2]. They use direct search over several window shapes. For each pixel a small number of different window shapes are tried, and the one with best cost is retained. To be efficient, the number of windows is severely limited and cannot cover the whole range of different sizes and shapes needed.

The distinguishing property of our algorithm from the previous work is that we employ a powerful optimization technique to find the window with the best cost out of a huge class of windows. For each pixel we find an optimal window size and shape by optimizing an appropriate window cost over a large class of "compact" windows. We use the word "compact" loosely to reflect that our windows have small perimeter to area ratio. While our compact window class is not completely general, it is still rather large, its size is exponential

[2]Informally, a disparity discontinuity occurs between pixels with significantly different disparities.

in the height of the largest allowed window. Furthermore, our compact window class contains all possible rectangles, but the majority of shapes are not rectangles. As far as we know, this is the first area based method which can efficiently construct non rectangular windows.

Efficient optimization over compact windows is achieved via *minimum ratio cycle* (MRC) algorithm for graphs. If the largest allowed window is n by n, for our graphs optimization takes $O(n\sqrt{n})$ time in theory, but is linear in practice. MRC algorithm restricts the window cost, but it is still quite general. We can include normalization by the window size which is crucial since we compare windows of different sizes. Stated briefly, our window cost is the average measurement error over window pixels with slight bias towards larger windows.

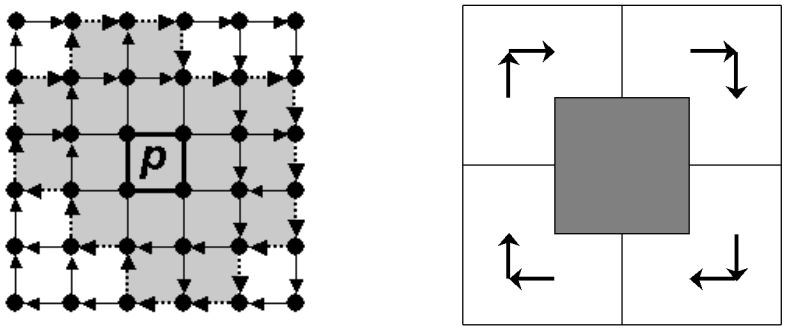
The straightforward algorithm computes the optimal window for each pixel-disparity pair. Even though we perform this step efficiently, it still depends on the window size, which is too slow for many applications. We devised simple heuristics which significantly reduce the number of optimal window computations while giving practically the same results.

We show results on real imagery including the ground truth database compiled by D. Scharstein and R. Szeliski. They also performed an extensive evaluation of different stereo correspondence algorithms. For the results, as well as taxonomy of different stereo algorithms, see [14]. Our method performs better than all other local methods that they evaluated. It is inferior only to some of the global methods, but global methods are less efficient. Besides the speed, the advantages of our algorithm is that it has few parameters which are easy to choose and the same parameters work well for different imagery.

2 The Compact Window Class

In this Section we define a compact window class. It is defined with respect to each pixel-disparity pair (p, d), since we use this class to estimate the likelihood of disparity d for pixel p. We denote the compact window class for pixel-disparity pair (p, d) by $\mathcal{C}_{pd}$. A natural way to define $\mathcal{C}_{pd}$ is through a certain directed graph, which we denote by $\mathcal{G}_{pd}$. This graph is embedded into the image patch around pixel p. An example of $\mathcal{G}_{pd}$ and its embedding is in Fig. 1(a). The squares correspond to image pixels, and the central thick square is the pixel p. The black dots in the corner of each pixel square are the graph nodes, and the directed arrows are the graph edges. Edges connect only the closest nodes, but not all such edges are in $\mathcal{G}_{pd}$. The edges that are included have a special structure, which is easiest to see in Fig. 1(b). The central gray region has no edges inside, and each of the four quadrants has only the edges in the directions shown. Notice that the edges trace out clockwise cycles, where in each quadrant a cycle follows a path in the general direction shown in Fig. 1(b).

We define windows in $\mathcal{C}_{pd}$ through the cycles of the graph $\mathcal{G}_{pd}$. Every directed cycle in



(a) Graph $\mathcal{G}_{pd}$ (b) Edge directions in each quadrant

Figure 1: Graph structure

$\mathcal{G}_{pd}$ encloses a connected area, and this connected area is a window in $\mathcal{C}_{pd}$. An example of a compact window is shown in gray in Fig. 1(a), with the corresponding cycle in dashed arrows. There is one to one correspondence between cycles in $\mathcal{G}_{pd}$ and compact windows, thus we frequently say "cycle corresponding to the window"and vice versa.

First we state a few obvious facts about $\mathcal{C}_{pd}$. By construction, each window of $\mathcal{C}_{pd}$ contains pixel p. This is of course crucial because we use $\mathcal{C}_{pd}$ to estimate how likely disparity d is for pixel p. The largest window in $\mathcal{C}_{pd}$ is limited by the graph size. It is useful to limit the largest size because we may wish to prevent windows to be as large as the whole image. The smallest window in $\mathcal{C}_{pd}$ is the central gray region with no arcs in Fig. 1(b). A limit on the smallest window is helpful because matching with a single pixel window is too unreliable. In practice we set the largest and the smallest windows to be 31 by 31 and 3 by 3, respectively.

Now we state the less obvious properties of $\mathcal{C}_{pd}$. If the largest window in $\mathcal{C}_{pd}$ is n by n, then there are $O(2^n)$ windows in $\mathcal{C}_{pd}$[3], so direct search is ruled out. All rectangles which contain the smallest one are in our class. However our class is much more general than the rectangles, the rectangles form only a small $O(n^4)$ part. Notice that our windows shapes are compact[4], that is they have low perimeter to area ratio.

Although window shapes in $\mathcal{C}_{pd}$ are not completely general, they are adequate for our purposes. Assume that there is a patch of pixels around pixel p with approximately the same disparity as p. We do not need to extract all the pixels in that patch, as long as we construct a sufficiently large window of pixels from that patch. In contrast our windows do not work for applications like image segmentation since the goal there is to extract all pixels in a region. The compactness of our windows may be an advantage over general shape. The best

[3]Count the number of paths in our graph; at each step, there are 2 possible directions for a path.

[4]However they do not contain all shapes that one might call compact

general shape window may have thin subparts which do not belong to the disparity of most pixels in the window, but do match well at that disparity due to the image structure. Thus the results may be plagued by these artifacts. Fig. 3(a) shows examples of our windows.

3 Window Cost

In this section we describe the general window cost that we can handle, and the one that we actually use. Let $err(q,d)$ be the measurement error if pixel p has disparity d. Approximately $err(q,d)$ is the absolute difference in intensities between pixel q in the left image and q shifted by d in the right image. For the exact description of $err(q,d)$ see Section 6.

Let (p,d) be a pixel-disparity pair for which we want to find the optimal window. Recall that $\mathcal{C}_{pd}$ denotes the set of all compact windows for the (p,d)-pair. For convenience, a window contains only pixels of the left image. For window $W \in \mathcal{C}_{pd}$ let C_W be the corresponding cycle, and e be an edge of C_W. Then the most general window cost that we can handle is

$$E(W) = \frac{\sum_{q \in W} err(q,d) + \sum_{e \in C_W} b(e,d)}{\sum_{q \in W} n(q,d)}.$$

Functions $err(q,d)$ and $b(e,d)$ can be arbitrary. For $n(q,d)$ there is a restriction that it has to be positive. Notice that $b(e,d)$ and $n(q,d)$ may depend on the particular edge e, pixel q and disparity d, however we currently we do not make use of it. Our actual window cost is:

$$E(W) = \frac{\sum_{q \in W} err(q,d)}{\sum_{q \in W} 1} + \frac{\sum_{e \in C_W} b}{\sum_{q \in W} 1} \tag{1}$$

The first term in the sum of equation (1) is the average window error. That is it accumulates $err(q,d)$ over all pixels q in a window and normalizes by the window size. This first term is a good criterion to exclude outliers[5], since inclusion of outliers in a window increases average error. Exclusion of outliers means that the disparity discontinuities are not smoothed out. We need more than that however. In low texture areas two windows may have different sizes but approximately equal average error. We want to favor the larger one since larger windows are more reliable. Thus the second term in the sum of equation (1) implements a bias towards larger windows. It is the ratio of window perimeter to window area multiplied by a parameter b. The second term is smaller for larger compact windows since area scales approximately quadratically while perimeter scales approximately linearly. Constant b has low positive value, since we want the bias term to differentiate only between window with

[5] Outliers are pixels whose error differs significantly from that of the other window pixels. Robust statistic deals with outliers by decreasing their weight in a window. In contrast we aim to avoid outliers altogether.

approximately equal average error. Thus our window cost is the average measurement error with bias towards larger windows. We found that this window cost discourages windows from crossing a disparity discontinuity while encouraging large windows in textureless areas.

4 Minimization via Minimum Ratio Cycle

The minimum ratio cycle algorithm (MRC) was first introduced into vision community by Jermyn and Ishikawa in their interesting image segmentation work [8]. In this Section we sketch the MRC problem and its complexity, and also describe how we use it to minimize the cost function in equation 1. For a thorough description of MRC algorithms see [1].

4.1 Minimum Ratio Cycle

Suppose $G = (V, E)$ is a directed graph with integer valued functions $w : E \to \mathbb{Z}$ and $t : E \to \mathbb{Z}$ on its edges. Function w can be arbitrary while t has the following restriction: $\sum_{e \in C} t(e) > 0$ for every cycle C. The problem then is to find a directed cycle C which minimizes the ratio: $\mu(C) = \frac{\sum_{e \in C} w(e)}{\sum_{e \in C} t(e)}$.

The MRC can be reduced to a repeated detection of a negative cycle. This algorithm was first described by Lawler [10]. Suppose μ^* is the optimal value of $\mu(C)$, and $\hat{\mu}$ is a guess at μ^*. Set the new edge weights $l(e) = w(e) - \hat{\mu} \cdot t(e)$. It is easy to see that if there is a negative cycle with the new weights, then $\hat{\mu} > \mu^*$. Similarly if there is a zero weight cycle then $\hat{\mu} = \mu^*$, and if there is no negative cycle then $\hat{\mu} < \mu^*$.

Since w and t are integral, μ^* must lie in the interval $[-W, W]$, where $W = \max\{|w(e)| : e \in E\}$. We use linear search to find μ^*. We start with $\hat{\mu} = W$, and compute the new edge weights $l(e) = w(e) - \hat{\mu} \cdot t(e)$. Then we run a negative cycle detection. If there is a negative cycle C, we set $\hat{\mu} = \mu(C)$ as the next guess and continue the search. If there is a zero weight cycle, then $\mu^* = \hat{\mu}$ and we terminate the search. There is always a negative or zero weight cycle, since $\hat{\mu}$ is an upper bound. This search must terminate after $O(WT^2)$ cycles, see [8].

A generic negative cycle detection takes $O(n^2)$ time for a graph with $O(n)$ edges, see [1]. However for our graphs, the worst case complexity is $O(n\sqrt{n})$, see [17]. Combining the negative cycle and the linear search complexity, the MRC has pseudopolynomial complexity. That is if we assume that the weights t and w are independent of the graph size (which they are for our problem), then the running time is polynomial, $O(n\sqrt{n})$ in our case. In practice we found that the average number of iteration to complete the linear search is around 3, and the average number of passes over the graph for the negative cycle detection is around 5. Thus the MRC algorithm needs about 15 passes over the graph to complete.

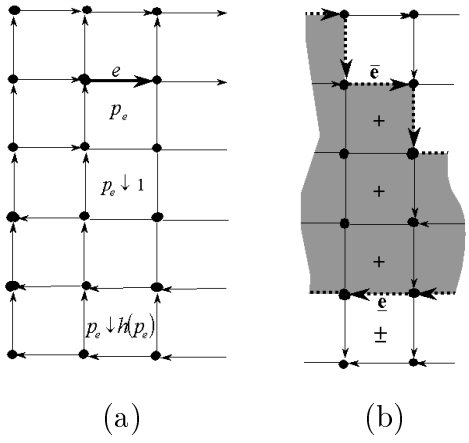


Figure 2: (a) edge e lies above pixel p_e; (b) cycle in dashed arrows, window in gray.

4.2 Conversion to Minimum Ratio Cycle

To find the optimum window in a $\mathcal{C}_{pd}$, we have to search for a minimum ratio cycle in the graph $\mathcal{G}_{pd}$, with properly defined edge weights w and t. First we need more notation. Recall the embedding of the graph $\mathcal{G}_{pd}$ in the image, shown in Fig. 1. We treat horizontal and vertical graph edges differently. A horizontal edge e lies above some pixel of the image, and we use notation p_e to denote this pixel. For example in Fig. 2(a) the thick edge e lies above pixel p_e. We use notation $p_e \downarrow i$ to denote the pixel directly below p_e by i spots, and $h(p_e)$ to denote the total number of pixels directly below pixel p_e. There is a special case when e lies on the very bottom edge of the image so that there is no pixel below it. In this case we set $h(p_e) = -1$, and we set the value of the empty sum to 0. We break up w into two functions $w = w_1 + w_2$. If edge e is vertical, then $w_1(e) = 0$. For horizontal e:

$$w_1(e) = \begin{cases} \sum_{i=0}^{h(p_e)} err(p_e \downarrow i, d) & \text{if } e \text{ points right} \\ -\sum_{i=0}^{h(p_e)} err(p_e \downarrow i, d) & \text{if } e \text{ points left} \end{cases}$$

That is an edge that points to the right accumulates the positive error in the column below it, and an edge pointing to the left accumulates the negative error in the column below.

Now consider a window W and its corresponding cycle C_W. Each column of W is bounded by cycle edges $\overline{e}$ on top and $\underline{e}$ on the bottom, as illustrated in Fig. 2(b). Edge $\overline{e}$ points to the right and $\underline{e}$ points to the left. The weight $w_1(\overline{e})$ accumulates the positive error in the column starting at pixel $p_{\overline{e}}$ and all the way down. All pixels that contribute to $w_1(\overline{e})$ are marked with "+" sign in Fig. 2(b). The edge weight $w_1(\underline{e})$ accumulates the negative error in the column starting at pixel $p_{\underline{e}}$ and all the way down. All pixels that contribute to $w_1(\underline{e})$ are marked with "-" sign in Fig. 2(b). Thus the sum of $w_1(\overline{e})$ and $w_1(\underline{e})$ gives exactly the total

error in the window column between edges $\overline{e}$ and $\underline{e}$. Now it easily follows that the sum over all the edge weights w_1 in the cycle C_W gives the total measurement error in the window W.

Similarly we can define t so that the sum of weights t over a cycle counts the total number of pixels in the corresponding window. For vertical e, $t(e) = 0$. For horizontal e:

$$t(e) = \begin{cases} h(p_e) + 1 & \text{if } e \text{ points right} \\ -(h(p_e) + 1) & \text{if } e \text{ points left} \end{cases}$$

Now we only have to add a bias towards larger windows, which we do by defining $w_2(e) = b$ for all edges e. It is easy to check that with $w(e) = w_1(e) + w_2(e)$ and t as defined, $\mu(C_W) = E(W)$ for window W and corresponding cycle C_W. Thus the cycle with minimum ratio μ corresponds to the window with the optimal cost E.

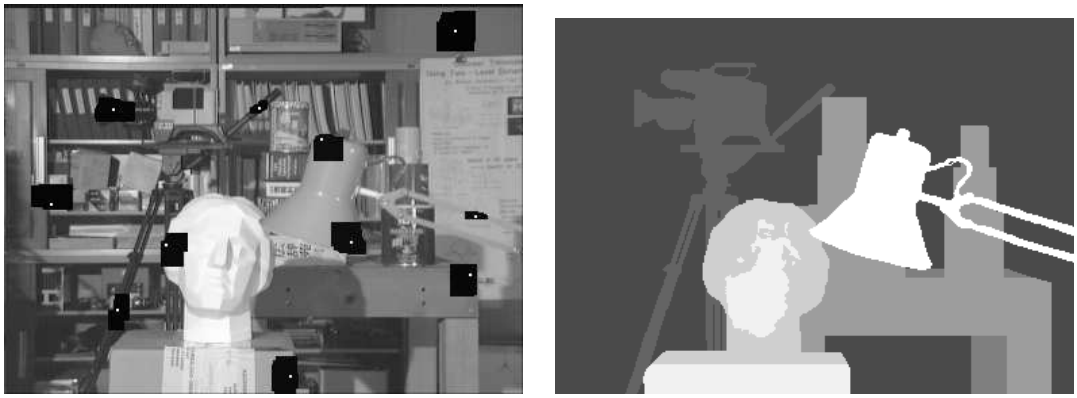
Observe that t satisfies the restriction of the MRC algorithm, that is its sum over any cycle is positive since it is just the corresponding window size. This holds because by construction our cycles are always clockwise. For any counterclockwise cycle, t would accumulate the negative window size, and we would not be able to use the MRC algorithm. If the MRC algorithm placed no restrictions on t, then we could find the best window of general shape.

5 Approximation and Pruning

The straightforward compact window algorithm computes the optimal window for each pixel-disparity pair (p, d). Even though we devised a very efficient method to compute the optimal window, it still depends on the largest window size in $\mathcal{C}_{pd}$, which is too slow for many applications. To speed things up, we develop an approximation and a pruning heuristics.

The approximation heuristic is based on the following observation. Fig. 3(a) shows the scene in Fig. 5(a) with several computed windows, and the true disparities are in Fig. 3(b). The black areas are the optimal windows, and the white dots are the pixels for which they were computed. The disparity of the windows is the correct disparity for the white pixels. Notice an optimal window tends to include only pixels at the same disparity as the pixel for which the window was computed. This is because our window cost discourages crossing a disparity discontinuity. Thus we can use the optimal window cost computed for pixel p to approximate the optimal window cost for pixels q which belong to the optimal window of p.

That is we do the following. Let W^o_{pd} be the optimal window and W^s_{pd} be the smallest window in $\mathcal{C}_{pd}$. We start computing optimal windows for (p, d)-pairs in the order of increasing $E(W^s_{pd})$. For a computed W^o_{pd} we set $E(W^o_{qd}) = E(W^o_{pd})$ for all $q \in W^o_{pd}$ and do not compute W^o_{qd}. If we get several estimates on $E(W^o_{qd})$, we retain the lowest one. This approximation heuristic lets us to significantly reduce the number of optimal window computations.



(a) (b)

Figure 3: (a) sample optimal windows; (b) true disparities

The second heuristic is simple pruning. We do not compute the optimal window for pixel-disparity pairs (p, d) if $err(p, d)$ is too large. Before computing W^o_{pd}, we check if p already has a good match at disparity d' with computed or estimated $E(W^o_{pd'})$. If $E(W^s_{pd})/c > E(W^o_{pd'})$, we exclude (p, d) from optimal window computation. We set $c = 1.5$ in our experiments. This pruning improves efficiency by excluding unlikely pixel-disparity pairs from computation. As in a typical area based matching, after all $E(W^o_{pd})$'s are computed or approximated, pixel p gets assigned disparity d which gives the lowest $E(W^o_{pd})$.

6 Measurement Error

Before presenting our experimental results, we describe our model for the measurement error $err(p, d)$. For low noise image pairs our algorithm performs quite well with $err(p, d)$ set to squared intensity difference. However frequently there is brightness difference between corresponding image patches or even nonlinear errors, especially in fine textured areas. For such image pairs it is beneficial to use $err(p, d)$ which models the above noise types.

We use the smallest window W^s_{pd} to estimate the average brightness in the left and right image patches around p. Let $\overline{I_L}$ and $\overline{I_R}$ be the average intensity in W^s_{pd} in the left image and average intensity in W^s_{pd} shifted by d in the right image. Let $I_L(q)$ is the intensity of q in the left image, and $I_R(q \oplus d)$ is the intensity of q shifted by d in the right image. We correct for brightness differences between the left and right image patches:

$$err_1(q, d) = |(I_L(q) - \overline{I_L})) - (I_R(q \oplus p) - \overline{I_R})|.$$

This estimate would be more reliable if all pixels in W^o_{pd} were used to compute $\overline{I_L}$ and $\overline{I_R}$.

However W^o_{pd} is not known in advance. In our experiments $|W^s_{pd}| = 9$, and it seems sufficient.

Besides brightness differences, there are frequently nonlinear errors between the corresponding patches. We develop another function $err_2(q, d)$ which works well in such cases. This function exploits local differences in intensities, retaining only their signs and not the magnitude. Let us define functions $sgn_l(q)$, $sgn_r(q)$, $sgn_a(q)$, and $sgn_b(q)$ as follows: $sgn_i(q) = sign(I_L(q) - I_L(q \to i))$. Here $q \to i$ stands for the pixel to the left, right, above, or below of q if $i = l, r, a, b$ correspondingly. The $sign(x)$ function just retains the sign of the argument, i.e. $sign(x) = 1$ if $x > 0$, $sign(x) = -1$ if $x < 0$, and finally $sign(x) = 0$ if $x = 0$. Functions $sgn_l(q \oplus d)$, $sgn_r(q \oplus d)$, $sgn_a(q \oplus d)$, and $sgn_b(q \oplus d)$ are defined similarly on the right image. Now define $err_2(q, d) = f(\sum_{i \in \{l,r,a,b\}} |sgn_i(q) - sgn_i(q \oplus d)|)$, where $f(x) = x$ if $x \leq 4$ and $f(x) = \infty$ otherwise.

Thus $err_2(q, d)$ measures how well signs of local variations match around q in the left image and $q \oplus d$ in the right image. This is robust to all monotonic nonlinear changes. Notice that if the argument to function f is larger than 4, less than two of sgn_i functions match, so the use of err_2 is unreliable and it is set to infinity. Our final measurement error is a combination: $err(q, d) = \min(err_1(q, d), err_2(q, d))$.

7 Experimental Results on Stereo Data

Our algorithm works well with the same parameters for all image pairs we tried. For all the experiments, we set the minimum window to 3 by 3, the maximum window to 31 by 31 (both centered at p); the pruning parameter $c = 1.5$, and the bias parameter $b = 1$.

First we evaluate our algorithm on the Middlebury database with dense ground truth due to D.Scharstein and R.Szeliski. They computed ground truth for several stereo pairs and also took one stereo pair with ground truth from the Tsukuba University. They have implemented several major approaches to stereo correspondence themselves and invited researchers to submit their results. They evaluated 20 algorithms, including graph cuts, dynamic programming, and area based methods. Summary of the evaluation is in Fig. 4, for their full report see [14], and the results can be found on *http://www.middlebury.edu/stereo/results.html*.

The first column in Fig. 4 gives names of each of the 20 stereo algorithms. The algorithms are arranged roughly in the order of performance, with the better ones on top. The next 4 columns give percentage errors each algorithm makes on the 4 scenes from the database. A computed disparity is counted as an error if it is more than 1 away from the true disparity. Each of these 4 columns is broken into 3 subcolumns: the *all* column gives the total error percentage, the *untex* column gives error percentage in the untextured areas of the image (where intensity gradient is smaller than some threshold), and the *disc* column gives the

	Tsukuba			*Sawtooth*			*Venus*			*Map*	
Algorithm	all	untex	disc	all	untex	disc	all	untex	disc	all	disc
Layered	1.58	1.06	8.8	0.34	0.00	3.35	1.52	2.96	2.6	0.37	5.2
GraphCut	1.94	1.09	9.5	1.30	0.06	6.34	1.79	2.61	6.9	0.31	3.9
BeliefProp	1.15	0.42	6.3	0.98	0.30	4.83	1.00	0.76	9.1	0.84	5.3
GC+occl	1.27	0.43	6.9	0.36	0.00	3.65	2.79	5.39	2.5	1.79	10.1
GraphCut	1.86	1.00	9.4	0.42	0.14	3.76	1.69	2.30	5.4	2.39	9.4
MultiCut	8.08	6.53	25.3	0.61	0.46	4.60	0.53	0.31	8.0	0.26	3.3
CompWin	3.36	3.54	12.9	1.61	0.45	7.87	1.67	2.18	13.2	0.33	4.0
Realtime	4.25	4.47	15.0	1.32	0.35	9.21	1.53	1.80	12.3	0.81	11.4
Bay. diff.	6.49	11.62	12.3	1.45	0.72	9.29	4.00	7.21	18.4	0.20	2.5
Cooperative	3.49	3.65	14.8	2.03	2.29	13.41	2.57	3.52	26.4	0.22	2.4
SSD+MF	5.23	3.80	24.7	2.21	0.72	13.97	3.74	6.82	13.0	0.66	9.4
Stoch. diff.	3.95	4.08	15.5	2.45	0.90	10.58	2.45	2.41	21.8	1.31	7.8
Genetic	2.96	2.66	15.0	2.21	2.76	13.96	2.49	2.89	23.0	1.04	10.9
Pix-to-Pix	5.12	7.06	14.6	2.31	1.79	14.93	6.30	11.37	14.6	0.50	6.8
Max Flow	2.98	2.00	15.1	3.47	3.00	14.19	2.16	2.24	21.7	3.13	16.0
Scanl. opt.	5.08	6.78	11.9	4.06	2.64	11.90	9.44	14.59	18.2	1.84	10.2
Dyn. Prog.	4.12	4.63	12.3	4.84	3.71	13.26	10.10	15.01	17.1	3.33	14.0
Shao	9.67	7.04	35.6	4.25	3.19	30.14	6.01	6.70	43.9	2.36	33.0
MMHM	9.76	13.85	24.4	4.76	1.87	22.49	6.48	10.36	31.3	8.42	12.7
Max. Surf.	11.10	10.70	42.0	5.51	5.56	27.39	4.36	4.78	41.1	4.17	27.9

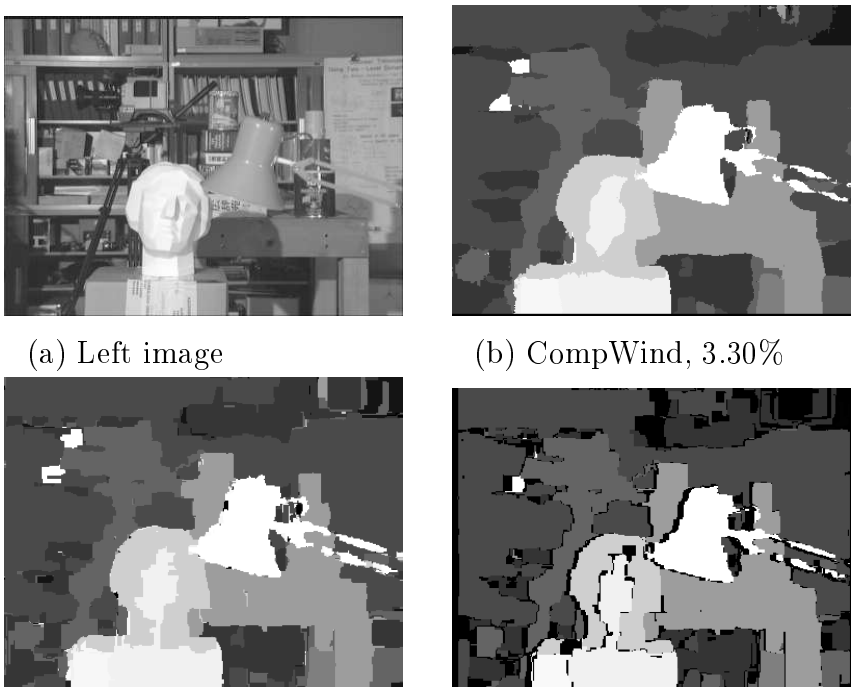
Figure 4: Middlebury stereo evaluation results

error percentage near discontinuities (at a small distance from some disparity discontinuity). For the *Map* scene there is not *untex* column because it is well textured everywhere.

The performance of our algorithm with approximation heuristics (CompWin) is highlighted by 2 horizontal lines. It performs better than all the local methods and some of the global ones[6]. In the table, above our algorithm are only to the graph cuts based methods and belief propagation. However our algorithm is faster than the global methods. The running time for the Tsukuba, Sawtooth, Venus, and Map scenes are 17, 29, 33, and 12 seconds, respectively. For the graph cuts the running times are approximately 3 times longer (when performed on the same machine). For the better textured scene *Map*, our algorithm performs better or just slightly worse than all the global methods which are ranked higher.

The Middlebury evaluation did not include a fixed window results because a shiftable window works much better. A shiftable window uses direct search over several window shapes, and its results are under the name SSD+MF. Our algorithm is significantly better

[6]Stereo algorithms can be roughly divided in 2 groups. Local methods make decision at a pixel by looking at just the local patch around that pixel. For global methods a pixel can influence a very distant pixel.



(a) Left image (b) CompWind, 3.30%

(c) CW with approx., 3.36% (d) left-right consistency, 1.95%

Figure 5: Tsukuba head and lamp scene

than SSD+MF, especially in the *disc* column, that is especially around discontinuities.

Now we evaluate if our error model in Section 6 helps on the Middlebury database. If instead of that model we use just the absolute difference in intensities, then the error percentages for *Tsukuba*, *Sawtooth*, *Venus*, and *Map* scenes are 3.72%, 1.92%, 2.27%, and 2.28% respectively. These percentages are slightly worse for the first three scenes, and significantly worse for the *Map* scene. The reason is that there is brightness differences between the left and right images of the *Map* scene, so modeling them helps.

The running time of our algorithm is influenced by image content. It is faster for the better textured images because the pruning heuristic is more effective then. For example for the better textured scene *Map* the algorithm processes 153,360 pixels per second. For the lowest textured scene *Tsukuba*, the algorithm processes only 97,581 pixels per second. Thus the running time is not completely predictable beforehand, but can be estimated depending on the type of imagery expected (for example textured outdoors versus low textured indoors).

In Fig. 5(a) is the left image of the Tsukuba stereo pair. In Figs. 5(b,c) are the results of our compact window without and with approximation heuristics, respectively. The ground truth for this scene is in Fig. 3(b). Under each image we show the percent of errors, 3.30% and 3.36% respectively. The running time without approximations is 22 minutes, and with

(a) Left image (b) True disparities (c) Compact window

Figure 6: Tsukuba plant scene

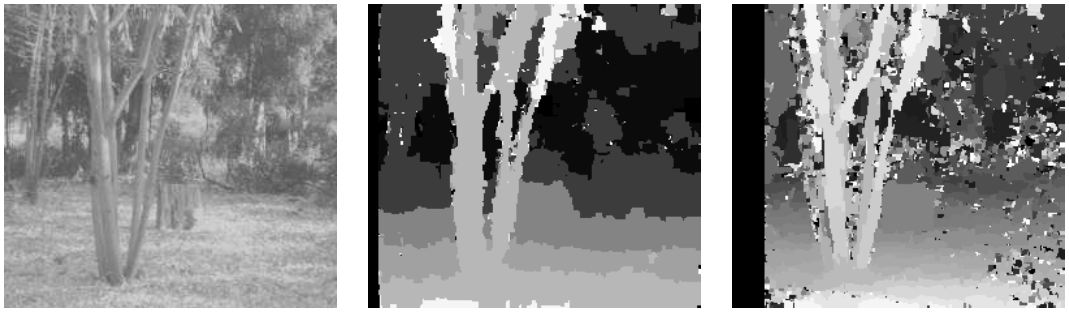
approximations it is 18 seconds. The number of pixels different between the two versions is 10%. However most of these differences are ± 1 disparity and are due to close window costs in low textured areas. Thus percentage error counts are almost equal for these algorithms.

Currently our algorithm does not deal with occlusion, i.e. with pixels which are visible in only one of the images. A standard way to detect occluded pixels is to run the algorithm with the left and right images reversed, and then remove pixels which get inconsistent disparity assignments between the two answers. The results of such procedure are shown in Fig. 5(d). Excluding pixels found inconsistent, the error count drops to 1.95%.

In Figs. 6(a,b) is another stereo pair with ground truth from the Tsukuba University, which is not included in the Middlebury database. In Fig. 6(c) is the result of our algorithm. Here our algorithm gives 16% error count, which is slightly better than graph cuts algorithm whose error count is 18% for this scene. We omit the results of the graph cuts algorithm, but they look quite similar to ours. In this experiment for the graph cuts algorithm we manually picked the best parameters, while for our algorithm parameters are fixed. The running time for our algorithm was 24 seconds, while for the graph cuts algorithm it was 72 seconds.

Fig. 7 shows our results on another common stereo sequence from SRI. Two results are shown, one for narrow and one for wide baselines. For the wide baseline this sequence has significant nonlinear errors in the grass region. Our algorithm performs well, the fine branch detail is preserved and the slopes of the ground plane are captured. The running times are 4 seconds for the small and 22 seconds for the large baselines.

The algorithm is quite robust to noise and brightness differences between the images, in our experience. The scenes in this section have different degrees of noise. The *Map* scene has significant brightness distortions, the wide baseline SRI trees have severe noise especially in the grass region. However the algorithm performs quite well with the same parameters for all the imagery.



(a) Tree sequence (b) Small baseline (c) Large baseline

Figure 7: Results on SRI trees

8 Conclusions

We presented an algorithm which gives an efficient way to optimize a window cost over a huge class of compact windows. This class contains all rectangular shapes, but the majority of shapes are not rectangular. We believe that our algorithm is a first one to efficiently construct non rectangular windows. Experimental results on the Middlebury database show that our algorithm performs better than all the other local methods tested there. It is inferior only to some of the global methods, but global methods are less efficient. The compact window algorithm can be used for other applications where window matching approach is used, as long as the window cost is the one that we can handle.

Acknowledgments

We thank Dr. Y. Ohta and Dr. Y. Nakamura for providing ground truth imagery; and Dr. Sharstein and Dr. Szeliski for providing ground truth imagery and stereo evaluation results.

References

[1] K. Ahuja, Thomas L. Magnati, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

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

[3] O. Faugeras, B. Hotz, H. Mathieu, T. Viéville, Z. Zhang, P. Fua, E. Théron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real time correlation-based stereo: algorithm, implementatinos and applications. Technical Report 2013, INRIA, 1993.

[4] A. Fusiello and V. Roberto. Efficient stereo with multiple windowing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 858–863, 1997.

[5] D. Geiger, B. Ladendorf, and A. Yuille. Occlusions and binocular stereo. *International Journal of Computer Vision*, 14:211–226, 1995.

[6] D.B. Gennery. Modelling the environment of an exploring vehicle by means of stereo vision. In *Ph. D.*, 1980.

[7] M.J. Hannah. Computer matching of areas in stereo imagery. In *Ph. D. thesis*, 1978.

[8] I. Jermyn and H. Ishikawa. Globally optimal regions and boundaries as minimum ratio cycles. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2001.

[9] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:920–932, 1994.

[10] E. Lawler. Optimal cycles in doubly weighted directed linear graphs. In *Int. Symp. on Theory of Graphs*, pages 209–232. Gordon and Breach, 1966.

[11] M.D. Levine, D.A. O'Handley, and G.M. Yagi. Computer determination of depth maps. *CGIP*, 2:131–150, 1973.

[12] K. Mori, M. Kidode, and H. Asada. An iterative prediction and correction method for automatic stereocomparison. *CGIP*, 2:393–401, 1973.

[13] D.J. Panton. A flexible approach to digital stereo mapping. *PhEngRS*, 44(12):1499–1512, December 1978.

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

[15] S. Scherer, W. Andexer, and A. Pinz. Robust adaptive window matching by homogeneity constraint and integration of descriptions. In *ICPR98*, page CVP1, 1998.

[16] N. Sebe, M.S. Lew, and D.P. Huijsmans. Toward improved ranking metrics. *PAMI*, 22(10):1132–1143, October 2000.

[17] O. Veksler. Stereo correspondence with compact windows via minimum ratio cycle. *NEC Research Institute Technical Report*, 2001.