Stereo Matching by Compact Windows via Minimum Ratio Cycle

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

Abstract

Window size and shape selection is a difficult problem in area based stereo. We propose an algorithm which chooses an appropriate window shape by optimizing over a large class of "compact" windows. We call them compact because their ratio of perimeter to area tends to be small. We believe that this is the first window matching algorithm which can explicitly construct non-rectangular windows. Efficient optimization over the compact window class is achieved via the minimum ratio cycle algorithm. In practice it takes time linear in the size of the largest window in our class. Still the straightforward approach to find the optimal window for each pixel-disparity pair is too slow. We develop pruning heuristics which give practically the same results while reducing running time from minutes to seconds. Our experiments show that unlike fixed window algorithms, our method avoids blurring disparity boundaries as well as constructs large windows in low textured areas. The algorithm has few parameters which are easy to choose, and the same parameters work well for different image pairs.

1 Introduction

Area correlation is one of the oldest approaches to dense stereo matching. To estimate match quality for a pixel p at disparity d, sum of squared differences (SSD) or some other measure is computed between a window centered at p in the first image and the same window shifted by d in the second image. For efficiency most methods use a rectangular window of fixed size centered at p.

There is a well known problem with this method. For a reliable estimate window must be large enough to include enough intensity variation, but small enough to cover only pixels at equal depth. However different pixels in the same image frequently require windows of different sizes, and no single window size works well: for a small size results are unreliable in low-textured areas; as the size is increased, results in low-textured areas get progressively reliable while disparity boundaries get increasingly blurred.

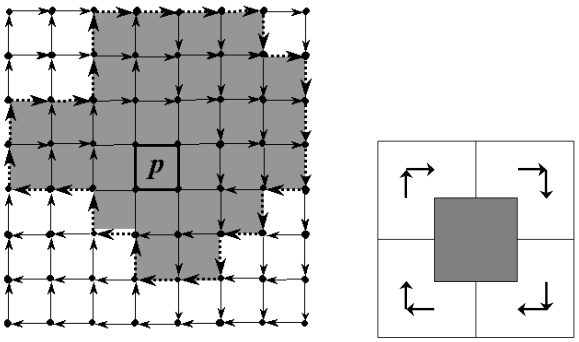
In addition, no fixed window shape works well for all pixels. Pixels that are close to a disparity discontinuity frequently require windows of different shapes to avoid crossing that discontinuity. That is why typical fixed window methods give especially bad results near discontinuities.

There were relatively few attempts to vary window's size or shape. The best known is the adaptive window by Kanade and Okutomi [7]. They use a model of intensity and disparity variation within a window to compute an uncertainty of disparity estimate. This allows a search for a window with locally minimum disparity uncertainty. The window shape is limited to rectangles due to computational cost. While this method is elegant, in our experiments in does not give sufficient improvement over fixed window methods. The problem might be its sensitivity to initial disparity estimate.

D. Geiger *et al.* [5] and A. Fusiello *et al.* [4] use a simpler multiple window method. For each pixel and disparity a limited number of distinct windows are tried, and the one with best correlation is retained. To be efficient, the number of windows is severely limited and cannot cover the whole range of different sizes and shapes needed.

We propose a method which is similar in spirit to [5] and [4]. However instead of trying a limited number of windows, for each pixel-disparity pair we compute the matching cost over a large class of "compact" windows. We use this term loosely and it reflects the fact that our windows have small perimeter to area ratio. The size of this class is exponential in the maximum window height. As far as we know, this is the first window-matching method which can explicitly construct non-rectangular windows. The exact description of our compact shapes is in section 2.

Efficient optimization over such a large class of windows is achieved via *minimum ratio cycle* algorithm (MRC) for graphs. Assuming that the largest window in the class is a n by n square, for our graphs optimization takes $O(n\sqrt{n})$ time in theory, but linear time in practice. The MRC algorithm puts constraints on the window match cost, but it is still quite general. In particular we can include normalization by the window size which is crucial since we compare windows of different sizes. The match cost of the window is described in detail in section 3, but in summary it is the av-



(a) Graph G_{pd} (b) Edge directions in each quadrant

Figure 1.

erage measurement error with bias towards larger windows.

The straightforward algorithm that computes the best window for each pixel-disparity pair depends on window size, which is too slow. We devised simple pruning heuristics which significantly reduce the number of optimal window computations while giving practically the same results.

Experiments on real imagery with ground truth show that not only our method outperforms fixed window algorithm, but it is also competitive with other algorithms which were designed to work well at discontinuities. In addition our parameters are easy to choose and the same parameters work well for different stereo pairs. In our framework we can handle brightness differences between images, as well as nonlinear noise, see section 6 for details.

2 The Compact Window Class

For each pixel-disparity pair (p, d), the compact window class is defined through the graph G_{pd}. An example G_{pd} is shown in Fig. 1(a). The squares correspond to image pixels, and the central thick square is pixel p. Black dots in the corner of pixel squares are the graph nodes, and directed arrows are the graph edges. Edges connect only the closest nodes. Fig. 1(b) summarizes the edges we include: the central gray region has no edges inside, and each of the four quadrants has only the edges in the direction shown.

Every directed cycle in G_{pd} encloses a connected area, and this connected area is a window in our class. An example window is shown in gray in Fig. 1(a), with corresponding cycle in dashed arrows.There is one to one correspondence between cycles in G_{pd} and compact windows, thus we say "cycle corresponding to the window" or vice versa.

The largest window size is limited by the graph size. The smallest window size is set by the central region with no arcs, we limit it since a single pixel window is too unreliable. If the largest window is a n by n square, then there are $O(2^n)$ windows in the compact class. Notice that our windows tend to have low perimeter to area ratio, hence the name compact (however they do not contain all shapes that one might call compact). All rectangles which contain the smallest allowed 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.

Although our window shape is not completely general it seems adequate for our purposes. Our goal is to construct a sufficiently large window of pixels that fit well to a single disparity while avoiding those that do not. In contrast our windows may not work for applications like image segmentation since the goal there is to extract all pixels in a region. Fig. 4(h) shows examples of the windows we find.

The compactness of our windows has an advantage over general shape. The best window of general shape may have thin subparts which do not belong to the disparity of the window but do match well at that disparity due to the image structure or noise. Unless there is a special treatment, the results may be plagued by these artifacts.

3 Window Matching Cost

There are three important terms in our window matching cost. The basic term accumulates over a whole window the measurement error for assigning disparity d to pixel p. Since we compare windows of different sizes, our second essential term normalizes by window size. The first two terms combined give the average window error. It is a good criterion to exclude outliers[1], since inclusion of outliers increases average error. We need more that that however. If two windows have different sizes but approximately equal average error we want to favor the larger one since larger windows are more reliable. Thus our last term implements a bias towards larger windows. It is particularly important in low texture areas where most windows have approximately equal low average error. Thus our window cost is the average measurement error with bias towards larger windows.

We now describe the general matching cost we can handle and narrow it down to the one we use in practice. Let (p, d) be a pixel-disparity pair for which we want to compute the matching cost. Let $\mathcal{S}_{pd}$ denote the set of all compact windows for the (p, d)-pair. For simplicity a window contains only pixels of the first image. For $W \in \mathcal{S}_{pd}$ let C_W be the corresponding cycle, and e be an edge of C_W.

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

[1] Outliers are pixels whose measurement error differs significantly from that of the other pixels in the window. The term comes from robust statistics, which deals with outliers by decreasing their weight in a window. In contrast we aim to avoid outliers altogether.

Here $err(q,d)$ models the measurement error for pixel q at disparity d; $n(q,d)$ is used for normalization and has to be positive; $b(e,d)$ can be arbitrary, and is used for bias towards larger windows. Notice that $b(e,d)$ and $n(q,d)$ may depend on the particular e, q and d, however we do not use this in practice. Our actual cost is:

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

which can be rewritten in words:

$$E(W) = \frac{\text{error of all pixels}}{\text{window size}} + b \cdot \frac{\text{window perimeter}}{\text{window size}} \quad (2)$$

The first term in equation 2 is just the average window error. The second term is smaller for larger compact windows since area scales approximately quadratically while perimeter scales approximately linearly. We set b to a low value since we want the bias term to differentiate only between windows with approximately equal average error.

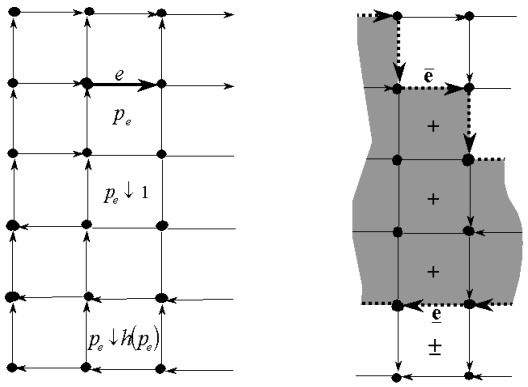
4 Minimization via Minimum Ratio Cycle

The MRC algorithm was first introduced to the vision community by Jermyn and Ishikawa in their interesting image segmentation work [6]. 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 full description of MRC algorithms see [1].

4.1 Minimum Ratio Cycle

Suppose $G = (V, E)$ is a directed graph with functions $w : E \to \mathcal{R}$ and $\tau : E \to \mathcal{R}$ on its edges. Function w can be arbitrary while $\sum_{e \in C} \tau(e)$ must be positive over every cycle C. The problem then is to find a directed cycle C which minimizes the ratio: $\mu(C) = \frac{\sum_{e \in C} w(e)}{\sum_{e \in C} \tau(e)}$. The MRC problem can be reduced to a detection of a negative cycle. Suppose μ^* is the optimal value of $\mu(C)$, and $\hat{\mu}$ is a guess at μ^*. Set the new edge weights $l(e) = w(e) - \hat{\mu} \cdot \tau(e)$. 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 $\hat{\mu} < \mu^*$.

We can use negative cycle detection and either binary or sequential search to find μ^*. Theory favors binary search, but in practice we found sequential search to be faster. It works as follows. Suppose $\overline{\mu}$ is an upper bound on μ^*. We start with $\hat{\mu} = \overline{\mu}$, and compute $l(e)$. If there is a zero weight cycle, then $\mu^* = \hat{\mu}$ and we terminate the search. If there is a negative cycle W, we set $\hat{\mu} = \mu(W)$ as our next guess and continue. There can be no positive cycle since $\hat{\mu}$ is always an upper bound. In practice it takes 3 iterations on average.



(a) Edge e lies above pixel p_e

(b) Cycle in dashed arrows, window in gray color

Figure 2.

For general graphs with $O(n)$ nodes and edges, negative cycle detection worst case complexity is $O(n^2)$, see [1]. We have shown in [2] how to reduce the running time to $O(n\sqrt{n})$ worst case for our graphs, the proof is omitted here due to the space constraints. However in practice it takes 2 passes over the graph on average. Thus on average we find the minimum ratio cycle after 6 passes over the graph.

4.2 Conversion to Minimum Ratio Cycle

To find the optimum window in our class for a pixel-disparity pair (p, d), we should search for a minimum ratio cycle in the graph G_{pd} with properly defined weight functions w and τ. First we need more notation. 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. [2]

The weight functions $w = w_1 + w_2$ and τ are as follows:

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

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

[2] 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.

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 $\underline{e}$. From this fact we deduce that the sum over all the edge weights w_1 in the cycle C_W gives the total error in the window W.

Similarly we can define τ so that the sum of weights τ over a cycle gives the total number of pixels in the corresponding window:

$$\tau(e) = \begin{cases} h(p_e) + 1 & \text{if } e \text{ points right} \\ -(h(p_e) + 1) & \text{if } e \text{ points left} \\ 0 & \text{if } e \text{ is vertical} \end{cases}$$

Now we only have to add a bias towards larger windows, which we do by defining $w_2(e) = b$. It is easy to check that with $w(e) = w_1(e) + w_2(e)$ and τ as defined, $\mu(C_W) = E(W)$ for window W and corresponding cycle C_W.

Observe that τ satisfies the restriction of the MRC algorithm, i.e. its sum over any cycle is positive since it is just the number of pixels in the corresponding window. This holds because by construction our cycles are always clockwise. For any counterclockwise cycle τ would accumulate negative window size, and we could not use MRC algorithm. If MRC placed no restrictions on τ, then we could find the best window of general shape (on the graph which contains all possible edges between the closest vertices).

5 Compact window algorithms

The time to compute the optimal window for a pixel-disparity pair (p, d) depends on the largest window in $\mathcal{S}_{pd}$. Thus the straightforward algorithm which finds the best window for all (p, d) pairs is too slow. We develop two simple pruning heuristics. The first one is based on the following observation. If the optimal window for some (p, d) pair has low cost, then not only pixel p should have a low matching cost at disparity d, but other pixels in this window should have a low matching cost at d. This idea allows us to significantly reduce the number of optimal window computations. We now explain it in details.

More notation first. Let W^o_{pd} denote the optimal window in $\mathcal{S}_{pd}$ and W^s_{pd} the smallest window in $\mathcal{S}_{pd}$. Obviously $E(W^s_{pd})$ is an upper bound on $E(W^o_{pd})$. Sometimes we need to compute just the average error over a window, without the bias term. Let $E_a(W)$ be such a window cost, given by the first term in the sum of equation 2.

for all (p, d) **do**
  $V(p, d) = 0$; $MIN(p) = \infty$; $MIN_a(p) = \infty$
Sort all (p, d) in increasing order of $E(W^s_{pd})$.
for each (p, d) in sorted order **do**
  if $V(p, d) = 0$ **and** $E_a(W^s_{pd})/c < MIN_a(p)$ **then**
    find optimal $W^o_{pd} \in \mathcal{S}_{pd}$.
    for all $q \in W^o_{pd}$ **do**
      $V(q, d) = 1$;
      if $E(W^o_{pd}) < MIN(q)$ **then**
        $MIN(q) = E(W^o_{pd})$; $Best_D(q) = d$;
        $MIN_a(q) = E_a(W^o_{pd})$;

Figure 3. Compact window with pruning.

Now consider a pair (p, d) with low $E(W^s_{pd})$ and suppose $q \in W^o_{pd}$. Then $E(W^o_{pd})$ is a good approximation to $E(W^o_{qd})$. Indeed, since $E(W^s_{pd})$ is low and $E(W^o_{pd}) \leq E(W^s_{pd})$, window W^o_{pd} consists mostly of pixels with low error. Thus there are many pixels around q with low error, and so $E(W^o_{qd})$ is likely to be small. In contrast suppose now that $E(W^s_{pd})$ is large. Then there still may be many pixels in W^o_{pd} with low error, since such pixels bring the cost down from the initial high upper bound. Thus some $q \in W^o_{pd}$ may have many pixels nearby with low error. Therefore $E(W^o_{qd})$ may be significantly lower than $E(W^o_{pd})$. Note that since average error carries much more weight than the bias term, we omitted analysis of bias in E.

There is another way to see the point above. Suppose we expand our window class by allowing graphs G_{qd} which are not necessarily centered at q. Then W^o_{pd} is a low cost window for q in the expanded class. Thus placing q at d should have a low cost which we approximate by $E(W^o_{pd})$.

The discussion above leads us to the our first pruning heuristic. 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 the optimal window for (q, d). We may get several estimates on $E(W^o_{qd})$ in which case we retain the lowest one.

Now we turn to the second pruning step. Before computing the optimal window for a (p, d) pair, we check if p already has a good match at disparity d' with computed or estimated $E(W^o_{pd'})$. If $E_a(W^s_{pd})/c > E_a(W^o_{pd'})$, we exclude (p, d) from optimal window computation. Constant c should be ≥ 1, and we set $c = 1.5$. This step improves efficiency by excluding unlikely pixel-disparity pairs from computation. In addition it improves results for thin objects. Suppose we do not make this pruning. Consider a pixel p on a thin object against a background, and suppose the true disparities of the object and background are d_o and d_b. At

d_o the optimal window for p is small and has low average error. At d_b the optimal window for p is large, with large errors for pixel p and other object pixels and small errors for the background. So the average error at d_b may also be low. Since we have bias towards larger windows, $E(W^o_{pd_o})$ may be larger than $E(W^o_{pd_b})$, and p may be placed at d_b, which is wrong. The pruning above significantly improves results for thin objects, since the average window error cannot stray too far from the average error in the window center.

The algorithm is summarized in Fig. 3. Variable $MIN(p)$ holds the minimum of $E(W^o_{pd})$ found so far for pixel p, with $MIN_a(p)$ holding corresponding $E_a(W^o_{pd})$, and $Best_D(p)$ the corresponding disparity. Sorting can be performed in linear time with the bucket sort.

6 Experimental Results

6.1 Measurement Error

Different image pairs have varying degree and types of noise. For low noise image pairs our algorithm performs very well with $err(p,d)$ set to SSD or SAD (sum of absolute differences). However frequently there is brightness difference between corresponding image patches or even nonlinear errors, especially in areas with fine textures when cameras' baseline is large. For such image pairs it is beneficial to use $err(p,d)$ which models the above distortions. We develop one such $err(p,d)$ below.

We use the smallest window W^s_{pd} to estimate the average brightness in the left and right image patches around p, i.e.

$$\overline{I}_L(p) = \frac{1}{|W^s_{pd}|} \sum_{q \in W^s_{pd}} I_L(q)$$

$$\overline{I}_R(p \oplus d) = \frac{1}{|W^s_{pd}|} \sum_{q \in W^s_{pd}} I_R(q \oplus d)$$

Here $I_L(p)$ is the intensity of p in the left image, and $I_R(p \oplus d)$ is the intensity of p shifted by d in the right image. The number of pixels in W^s_{pd} is given by $|W^s_{pd}|$.

Now suppose there is a shift in brightness between the left and right corresponding image patches. That is $I_L(q) \approx I_R(q \oplus d) + s$. Then $\overline{I}_L(p) \approx \overline{I}_R(p \oplus d) + s$, and we can get rid of brightness shift s by subtracting $\overline{I}_L(p)$ from the left and $\overline{I}_R(p \oplus d)$ from the right image pixels. Thus we define

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

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 we do not know W^o_{pd} in advance. In our experiments $|W^s_{pd}|$ is 9, and performs reliably. Note that err_1 model is another reason why we put a limit on the smallest window we allow.

There are frequently nonlinearities in the corresponding intensities. We develop one more error function $err_2(q,d)$ which works well in textured areas of an image. 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:

$$sgn_i(q) = \begin{cases} -1 & \text{if } I_L(q) - I_L(q \to i) < 0 \\ 1 & \text{if } I_L(q) - I_L(q \to i) > 0 \\ 0 & \text{if } I_L(q) - I_L(q \to i) = 0 \end{cases}$$

Here $q \to i$ stands for the pixel to the left, right, above, or below of q if $i = l, r, a, b$ correspondingly. 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

$$err_2(q,d) = f\Big(\sum_{i \in \{l,r,a,b\}} |sgn_i(q) - sgn_i(q \oplus d)|\Big),$$

where

$$f(x) = \begin{cases} x & \text{if x } \le 4 \\ \infty & \text{otherwise} \end{cases}$$

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 very robust to many 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. This is expected in low textured areas, and so our final measurement error is a combination:

$$err(q,d) = \min\left(err_1(q,d), err_2(q,d)\right)$$

6.2 Results

Our algorithm works well with the same parameters for all image pairs we tried. For all the experiments, we set the minimum window to a 3 by 3 square, the maximum window to a 31 by 31 square, both centered at p. The remaining parameters are $c = 1.5$, and $b = 1$.

We make comparisons with a fixed window method and the adaptive window algorithm in [7][3]. For these algorithms we chose parameters which gave the smallest error in disparity compared to ground truth. Adaptive window algorithm was initialized with the results of the fixed window method. We also compare our results with the graph cuts algorithm in [3]. This algorithm was designed to accurately localize discontinuities, and it gives the best published results on the Tsukuba data. The algorithm imposes a prior on the distribution of a disparity map. We chose the Potts prior which has only one parameter λ. Larger λ values encourage less discontinuities in the disparity map. We found that graph-cuts algorithm works very well for a correctly chosen λ. However the optimal λ value depends on a particular scene and may vary significantly for different scenes. The paper in [3] gives no way to choose λ automatically, and it is obviously hard since it depends on unknown scene content. We compare our results with the graph cuts algorithm

[3] Implementation found on Internet

for the best value of λ. Besides the efficiency, our advantage over the graph cuts algorithm is that our parameters are easy to choose and the same parameters work well on different scenes. In addition we can model brightness differences between two images.

Figs. 4(a,b) show the left image of a stereo pair and its ground truth from Tsukuba university. Figs. 4(c-g) show the results of a fixed window, adaptive window, graph-cuts, our compact window and compact window with pruning algorithms. Under each image we show running time, percent of exactly correct disparities, and percent of disparities off by ± 1 from the correct answer.

The adaptive window algorithm actually worsens results of the fixed window method. Our methods are significantly better in ± 1 correct disparities than the fixed and adaptive window algorithms. Although the percentage of the exactly found disparities is almost the same, keep in mind that for our algorithms parameters are fixed, while for all other algorithms parameters are manually optimized to give the best performance. Superiority of our methods is most obvious around disparity discontinuities, which are localized significantly better. The table below lists the percent of correct and ± 1 correct disparities for pixels which are within distance $1, 2, 3$ and 4 from a discontinuity. We omit the adaptive window results in this table, its results are slightly worse than that of the fixed window.

distance from disc.	1	2	3	4
Compact window	57,79	62,84	65,86	67,87
Fixed window	46,69	50,72	53,73	56,75

Compared with the graph cuts method, our algorithms give significantly less exact disparities, however the ± 1 correct disparities are the same. We have no prior on disparity map, and for this stereo pair there are many large regions where the measurement error for the disparity that our algorithm chooses is slightly but consistently better than the measurement error at the correct disparity.

We now compare our compact window algorithms with and without pruning. The number of pixels different between the two is 10%. However most of these differences are ± 1 disparity and are due to close matching costs in low textured areas. Thus disparity error counts are equal for these algorithms. Note also that as predicted, thin objects are found better by the algorithm with pruning.

Fig. 4(h) shows several optimal windows found for pixels at their computed disparities. Windows are black while pixels for which windows were constructed are white. Observe that windows grow as large as they can without crossing disparity continuities. Notice especially how the windows in the corners of the lamp confine to its shape, the small thin windows on the lamp and camera handle, and the large window in the right upper textureless corner.

Fig. 5 shows another stereo pair with ground truth from the Tsukuba database. Due to space constraints we omit the picture for the fixed window method, but its best results are for window size 5×5 with 45% exact and 61% ± 1 correct disparities. Here our algorithm gives the same percentage of exact disparities and even better ± 1 correct disparities than graph cuts method. There are more discontinuities in this scene than in the previous one. Therefore the optimal λ for graph cuts algorithm is 4 times smaller than for the previous stereo pair. The table below summarizes exact and ± 1 correct disparities for these two scenes and different λ.

λ	1	5	10	20	100
Fig 4	71,93	84,93	89,93	90,95	85,92
Fig 5	53,79	56,82	42,72	44,72	32,34

For the Tsukuba ground truth data we conclude that our algorithm performs better than the fixed and adaptive window algorithms and it is competitive with graph cuts algorithm. Note that our parameters are fixed while parameters for other algorithms are manually optimized.

Fig. 6 shows results of our algorithm with pruning on other common stereo images. Two results are shown, one for narrow and one for wide baselines. For the wide baseline, the shrub sequence has significant brightness differences, and the tree sequence has nonlinear errors especially in the grass region. Our algorithm performs well, the fine branch detail is preserved and the slopes of the ground planes are captured.

Acknowledgments

We thank Dr. Y. Ohta and Dr. Y. Nakamura from the University of Tsukuba for providing the images with the dense ground truth.

References

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

[2] Authors. Stereo matching by compact windows via minimum ratio cycle. In *Technical report*, 2000.

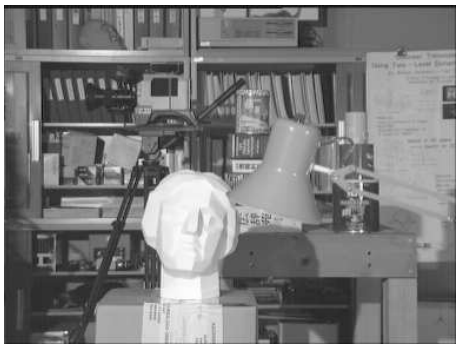
[3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *International Conference on Computer Vision*, pages 377–384, 1999.

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

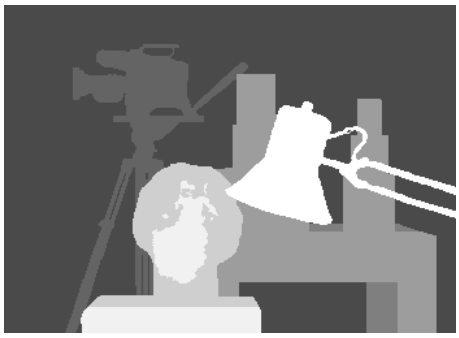
[5] D. Geiger, B. Ladendorf, and A. Yuille. Occlusions and binocular stereo. *International Journal of Computer Vision*, 14:211–226, 1995.

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

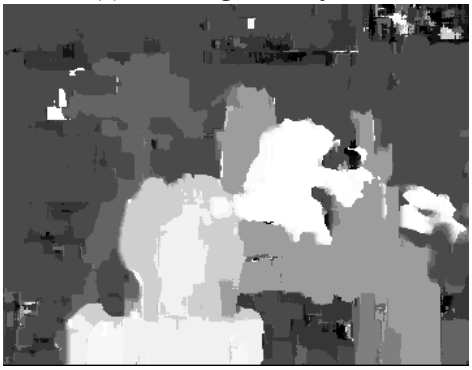
[7] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:920–932, 1994.



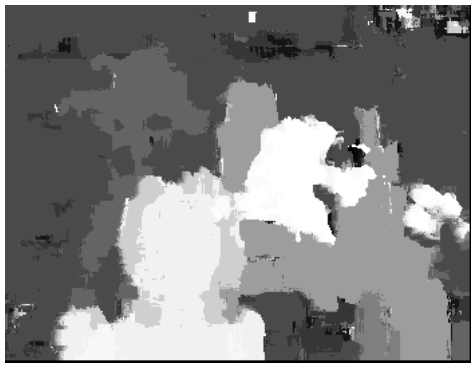
(a) Left image: 384 by 288



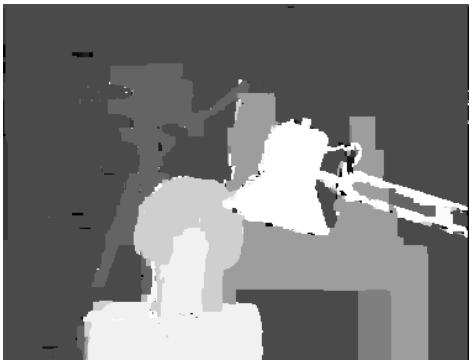
(b) Ground truth: 14 disparities



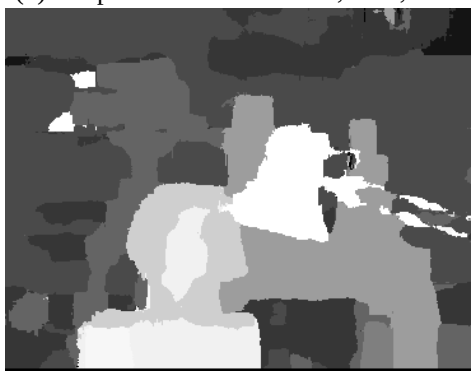
(c) Fixed window (11$\times$ 11): 1 sec; 72%,88%



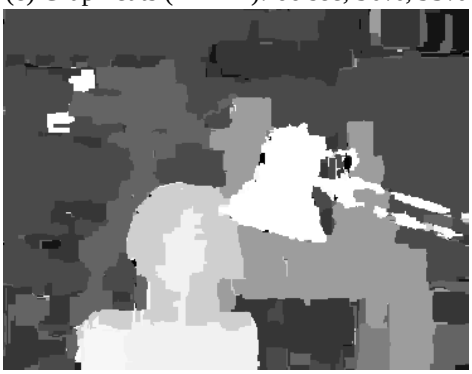
(d) Adaptive window: 300 sec; 72%, 86%



(e) Graph cuts ($\lambda = 20$): 66 sec; 90%, 95%



(f) Compact window: 22 min; 73%, 95%



(g) Compact window with pruning: 18 sec; 73%,95%

(h) Sample optimal windows

Figure 4. Imagery with ground truth

(a) Left image: 320 by 240

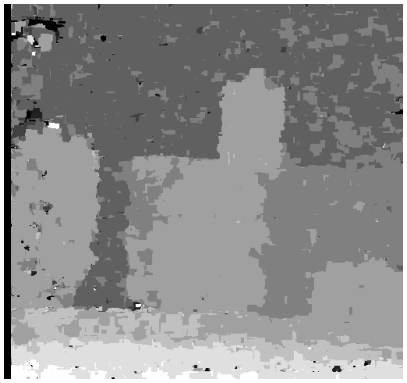
(b) Ground truth: 25 disparities

(c) Graph cuts ($\lambda = 5$): 72 sec; 56%, 82%

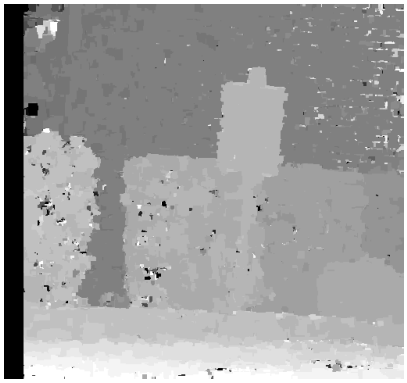
(d) Compact window with pruning: 24 sec; 56%, 84%

Figure 5. Imagery with ground truth

(a) Shrub sequence: 512 by 480

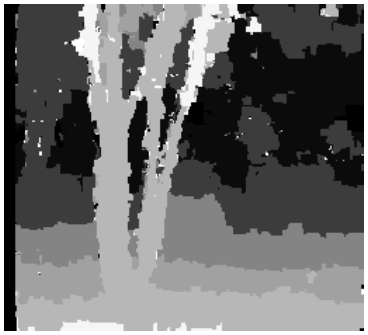


(b) Small baseline:56sec, 9 disp

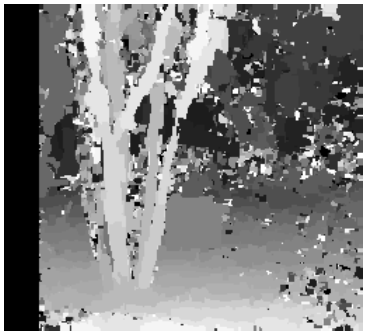


(c)Large baseline: 140 sec, 26 disp

(c) Tree sequence: 256 by 233



(d) Small baseline: 4 sec, 6 disp



(e) Large baseline: 22sec, 25 disp

Figure 6. Results of our algorithm on other real imagery