Extracting Dense Features for Visual Correspondence with Graph Cuts

Olga Veksler
NEC Laboratories America, 4 Independence Way Princeton, NJ 08540
olga@nec-labs.com

Abstract

We present a method for extracting dense features from stereo and motion sequences. Our dense feature is defined symmetrically with respect to both images, and it is extracted during the correspondence process, not in a separate preprocessing step. For dense feature extraction we use the graph cuts algorithm, recently shown to be a powerful optimization tool for vision. Our algorithm produces semi-dense answer, with very accurate results in areas where features are detected, and no matches in featureless regions. Unlike sparse feature based algorithms, we are able to extract accurate correspondences in some untextured regions, provided that there are texture cues on the boundary. Our algorithm is robust and does not require parameter tuning.

1 Introduction

Visual correspondence is a key task in many vision applications. Two images of the same scene are given, and the task is to find pixels in different images which are projections of the same world point. We develop an algorithm for stereo and motion correspondence. In stereo, images are taken simultaneously from different view points, and correspondence gives depth cues. In motion, images are taken at different times, and correspondence gives motion cues.

There has been a wealth of approaches to visual correspondence. Most give dense estimates, that is they establish correspondence for all or almost all pixels. These include methods based on optical flow [6], correlation [4], dynamic programming [8], graph-cuts [2], etc. While many of these algorithms perform well under good conditions, such as low noise and reasonably textured data, they fail under unfavorable conditions. Indeed, in many realistic scenes it may be impossible to find reliable correspondences in some regions no matter which algorithm is used. A robust vision system needs to disregard any correspondences in such regions. Many dense methods however do not provide a confidence measure in their correspondences at all. If they do provide a confidence measure it usually something simple which in essence marks correspondences near texture as more reliable. Depending on scene texture, such a confidence measure may dismiss most of the correspondences.

There are also methods which establish correspondence only for parts of the scene, in order to gain a more reliable performance. For example most feature based methods match sparse image features, like edge pixels [5] or edge segments [7]. These methods are more robust, but can produce quite sparse results.

Similar to the methods in the previous paragraph, we want to develop a highly accurate, but not necessarily dense algorithm. We want correspondences only in regions where reliable correspondences can be found. To achieve this goal, we look for *dense features* which are easy to match reliably. Compared to sparse feature algorithms, we produce denser results and in addition find correspondences in some untextured regions, if there is reliable texture on their boundary.

Our algorithm is based on the idea in [11]. In [11] they introduced a notion of dense features for stereo. They define a dense feature as a connected set of pixels in the left image and the corresponding set of pixels in the right image such that intensity edges on the boundary of these sets are stronger than the matching error on the boundary (which is the absolute intensity difference between corresponding boundary pixels). They call this "the boundary condition". The idea is that even for untextured region, its boundary can give a cue for correspondence. The boundary cue is good enough if the intensity change on the boundary is stronger than noise, and noise is reflected by the matching error. In addition, the insides of the left and right pixel sets should match, as checked through thresholds. Note that a dense feature is associated with a disparity equal to the displacement between the left and right pixel sets.

The main limitation of [11] is the way they extract dense features. They are extracted using a local algorithm which processes each scanline independently from the other. As a result, in [11] are able to enforce the boundary condition only on the left and right boundary, but not on the top and bottom, which reduces reliability. We borrow the idea of dense features and the boundary condition from [11], but

our overall dense feature definition is different. Our main advantage over [11] is that we use graph cuts for dense feature extraction, which is a global optimization algorithm shown to be a powerful tool for vision [2]. As a result we are able to enforce the boundary condition for the whole boundary of a dense feature, which significantly improves accuracy compared to [11]. Also using optimization framework we avoid hard thresholds that are necessary in [11]. In addition we extend our algorithm to handle motion data.

Our dense features have all of the desirable properties of the dense features in [11]. That is they are extracted during the correspondence process, not a separate preprocessing step, and they are symmetric with respect to both images. Unlike sparse features, our dense features are very descriptive, so that after all dense features are computed, there is little disambiguation to be done. That is if a pixel belongs to more than one dense feature, it is likely that either one of these dense features is due to noise and is very small in size, or that the pixel is in a repeated texture region.

2 Optimization with Graph Cuts

Let $\mathcal{P}$ be a set of pixels and suppose we want to assign a binary label to each pixel $p \in \mathcal{P}$. Let f_p denote the label assigned to pixel p, and let $f = \{f_p | p \in \mathcal{P}\}$. In this paper we need to optimize the following binary function.

$$E(f) = \sum_p D_p(f_p) + \sum_{(p,q)\in\mathcal{N}} u_{(pq)} T(f_p \neq f_q) \qquad (1)$$

Here $D_p(f_p)$ is a function which depends on the individual label assigned to pixel p. It is used to encode a label preference for pixel p. $\mathcal{N}$ is a neighborhood system defined on $\mathcal{P}$, consisting of pairs of pixels. In this paper $\mathcal{N}$ is the standard 4-neighborhood system, that is each pixel has its top, left, bottom and right pixels as neighbors. Function $T(\cdot)$ is 1 if its argument is true, and 0 otherwise. Summation is over all ordered neighbor pairs (p, q), and finally $u_{(pq)}$ is a constant depending on the ordered pair (pq).

This type of energy is commonly used in vision, and it is a balance of two terms. The first sum encourages labelings where each pixel is assigned the label it likes according to D_p, thus encouraging labelings consistent with the observed data. The second sum encourages labelings where most nearby pixels have the same label, thus encouraging spatial consistency frequently exhibited by visual data.

Equation (1) can be optimized exactly with a minimum graph cut [2]. We use the new fast max-flow algorithm in [1]. The running time is nearly linear in practice.

3 Dense Feature Motivation

In this section we motivate our dense features. Our dense feature is associated with a certain displacement vector which is one dimensional for stereo (it is usually called disparity) and two dimensional for motion. That is a dense feature is a connected set of pixels which are likely to undergo the same displacement between the two images. Our goal is to find the properties of a connected set of pixels which make it a good candidate for a dense feature, that is properties that allow reliable matching of this pixel set.

Here is the overall idea. Dense features exist at some displacement, so first we fix a displacement. Texture gives a good cue for correspondence. Using texture cues for each pixel we decide if it is likely to undergo the fixed displacement between the two images. If yes, then there is a positive cue at that pixel. We also evaluate whether a pixel is not likely to undergo the fixed displacement, in which case there is a negative cue at that pixel. A concentration of positive cues indicates a possible dense feature presence in their neighborhood. A concentration of negative cues indicates that there should be no dense features in their neighborhood. Now we face a binary segmentation problem, dividing all pixels into two groups: those which undergo the fixed displacement and those which do not. To find a suitable place for dense feature boundaries, we use the boundary condition. Finally we convert the binary segmentation problem into the energy minimization problem in equation (1), and we segment (or label) all pixels using a graph cut.

Here is roughly how we compute positive and negative cues. Let p be a pixel, p_l its left neighbor, d a fixed displacement, and $L(p)$, $R(p)$ the intensities of p in the left and right images. Our basic measure of texture is simply $|L(p) - L(p_l)|$ in the left image and $|R(p) - R(p_l)|$ in the right image. In addition to texture cue, there is also a matching error for p at the displacement d. This matching error is $e(p, d) = |L(p) - R(p + d)|$. If the texture cues of p in the left image and of $p + d$ in the right image are larger than $e(p, d)$ and $e(p_l, d)$, then there is a positive cue (its strength is actually variable, details in Section 4.1). The intuition is that a texture cue is reliable only if it is stronger than the noise in the images, and noise is reflected in the matching error. For example, if $L(p) = 55$, $L(p_l) = 65$, $R(p + d) = 60$, and $R(p_l + d) = 67$, then there is a positive cue, since texture cues are 10 and 7 for the left and right images, and $e(p, d) = 5$ and $e(p_l, d) = 2$, which are smaller than both texture cues. If $R(p + d) = 65$ and $R(p_l + d) = 70$, then there is no positive cue. Definition of a negative cue is simpler. Roughly a negative cue is given by any pixel whose matching error is too large.

We found that with our definition of positive and negative cues the algorithm works quite well. It is possible to come up with a totally different definition of positive and negative cues. They should work well provided that it is significantly more likely for a pixel to give a positive cue rather than negative cue at the pixel's correct displacement.

Consider a stereo scene whose left image is in Fig. 1(a).

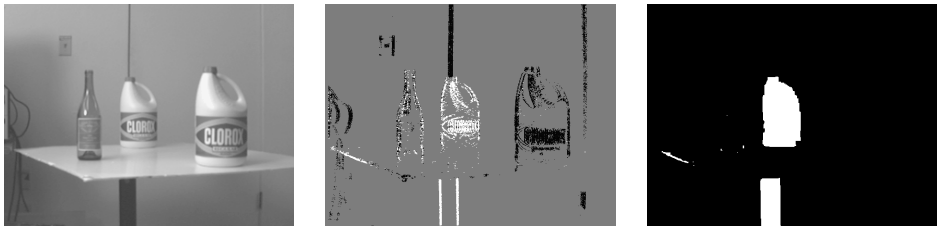

Figure 1. (a) Left image; (b) positive and negative cues; (c) dense features

We fix a displacement at the disparity six, which is the disparity of the table leg, one of the bottles and part of the table. Consider Fig. 1(b). Positive cues are in white, negative cues are in black, and gray pixels give neither positive no negative cues. In Fig. 1(c) are the extracted dense features in white. They correspond, pretty accurately to the table leg, one of the bottles and table edge. The inside of the table leg is almost completely textureless, however we are able to extract the whole leg as a dense feature.

4 Extraction via Graph Cuts

In this section we translate dense feature extraction into energy minimization of the form in equation (1), which can be minimized exactly with a graph cut [2].

Let a displacement d be fixed. We will divide all pixels into two groups: those which undergo displacement d and those which do not. We approach this as a labeling problem. Each pixel p is assigned a binary label. If p has label 1, then p undergoes displacement d, and if p has label 0 then p does not undergo displacement d. Now we need to define $D_p(f_p)$'s and $u_{(pq)}$ in equation (1). $D_p(f_p)$'s encode positive and negative cues. The smaller $D_p(l)$ is, the more likely is a label l for p. The likely places for dense feature boundary are encoded through $u_{(pq)}$'s. A smaller $u_{(pq)}$ means that a dense feature boundary is likely between p and q.

After $D_p(f_p)$'s and $u_{(pq)}$'s are defined, one graph cut labels all pixels with 0's and 1's. The connected sets of pixels labeled 1 are our dense features. Notice that with one graph cut we usually extract multiple dense features.

4.1 Definition of $D_p(f_p)$'s

Recall that we use $D_p(f_p)$ to encode pixel p's preference for labels 0 and 1. Label 1 corresponds to a positive cue at pixel p, and label 0 corresponds to a negative cue. We first describe $D_p(f_p)$'s for the stereo case. Let $L(p)$, $R(p)$ be as defined in Section 3. Let us fix a displacement d. First let us consider a positive cue, that is $D_p(1)$. Since we are minimizing the energy, the smaller $D_p(1)$ is, the more likely label 1 is for pixel p. There are two components that go into $D_p(1)$. First there is a texture cue at pixel p, and second we make sure that the matching error is not too large for p.

First consider the texture cue. For stereo, vertical texture is more reliable than the horizontal one, since the displacement is horizontal. Let p_l be the left neighbor of p. There is a good texture cue if the intensity change between p and p_l is larger than their matching error. We first measure the intensity change between pixel p and p_l in the left image: $\delta_l = |L(p) - L(p_l)|$. For symmetry, we also measure intensity change between the corresponding pixels in the right image: $\delta_r = |R(p+d) - R(p_l+d)|$. The symmetric measure of intensity change is $\delta = \min\{\delta_l, \delta_r\}$. Then we compute the matching error for p and p_l: $e(p) = |L(p) - R(p+d)|$, $e(p_l) = |L(p_l) - R(p_l+d)|$. Finally the texture cue is $t\_cue = 10 - h(\delta - e(p)) - h(\delta - e(p_l))$, where

$$h(x) = \begin{cases} 10 & \text{if } x < 0 \\ 10 - x^2/2.5 & \text{if } 0 \le x \le 5 \\ 0 & \text{if } x > 5 \end{cases}$$

That is $t\_cue \le 0$ if $\delta < e(p)$ or if $\delta < e(p_l)$, then it increases quadratically and stops increasing at 10, when δ is sufficiently larger than both $e(p)$ and $e(p_l)$.

Another component for a positive cue is the matching error, which should not be too large. We define $m\_cue = g(e(p)) + g(e(p_l))$, where $g(x) = 10 - x^2/160$. Finally we have to convert from cues to penalties, since the smaller $D_p(1)$ is, the more likely label 1 is for pixel p:
$D_p(1) = \max\{0, \min\{10, (10 - t\_cue) + (10 - m\_cue)\}\}$.

Notice that $0 \le D_p(1) \le 10$.

The definition of $D_p(0)$ is less involved, since we just look at the matching error. To make $D_p(0)$ slightly more robust, in addition to looking at the matching error for p, we also look at the matching error for p_l, since if a pixel does not undergo displacement d, in most cases, its left neighbor also does not undergo displacement d, and should have a large matching error as well. So we define

$$D_p(0) = \max\{0, 10 - \frac{\min\{e^2(p), e^2(p_l)\}}{30}\}.$$

With such a design, for $D_p(0)$ to be low for pixel p, not only $e(p)$ has to be large, but also $e(p_l)$ has to be large. Notice that $0 \le D_p(0) \le 10$, which is on the same scale as $D_p(1)$.

Figure 2. (a) left image; (b) dense features at disparity 14; (c) dense features at disparity 5

So far we defined $D_p(f_p)$'s for the stereo case. The motion case is very similar, except that we have to look not only for vertical texture cues (for stereo we looked for texture cue between pixel p and p_l), but also for horizontal texture cue, since in motion displacement is two dimensional and vertical texture cues alone are not reliable enough. We will not go into details for the lack of space, but for $D_p(1)$ to be small for motion, in addition to making sure the intensity difference between p and p_l is larger than the matching error, we also make sure that the intensity difference between p and its top neighbor is larger than the matching error.

4.2 Definition of $u_{(pq)}$'s

We use $u_{(pq)}$'s to enforce the segmentation boundary to lie at pixels which satisfy the boundary condition, and so $u_{(pq)}$'s should be small for such pixels. The boundary condition states that if the intensity change between two pixels is larger than the matching error, than this is a good place for the boundary, since the texture cue here is stronger than the noise. Due to various artifacts, there is often no contiguous set of pixels satisfying the boundary condition. So we actually allow the boundary to lie not necessarily at pixels satisfying the boundary condition, but close to such pixels.

First for each pixel p we compute how well the boundary between p and its left neighbor p_l satisfies the boundary condition. Similar to computing $D_p(1)$ in section 4.1, first we compute the minimum intensity difference: $\delta = \min\{|L(p) - L(p_l)|, |R(p+d) - R(p_l+d)|\}$. Then:

$$B_l(p) = \begin{cases} \infty & \text{if } \delta < e(p) \\ h(\delta - e(p)) & \text{otherwise} \end{cases}$$

Here $e(p)$, $h(x)$ are defined as in section 4.1. Thus $B_l(p)$ has a low value if it is likely that a left dense feature border goes between p and p_l. Similarly we compute $B_r(p)$, $B_u(p)$, $B_d(p)$ which have a low value if it is likely that a right, up, down dense feature border goes between p and its right, up, and down neighbors, respectively.

Next we use the generalized distance transform on the array $B_l(p)$ to compute how far each pixel is from a suitable place for a left boundary. That is we compute $T_l(p) = min_{q \in \mathcal{P}}(B_l(q) + dist(p,q))$, where $dist$ is the standard Manhattan distance. In [3] they show how to compute the generalized distance transform in two passes over all pixels. Similarly we compute T_r, T_u, T_d.

Finally, for each ordered pair (pq), if q is to the left of p,

$$u_{(pq)} = \begin{cases} 1 + B_l(p) & \text{if } B_l(p) \neq \infty \\ 1 + (T_l(p))^2 & \text{otherwise} \end{cases}$$

Cases when q is to the right, up, down of p are handled similarly with the corresponding arrays $B_r, T_r, B_u, T_u, B_d, T_d$. Note that having ordered pairs (pq) is important here, since it is possible that the left boundary between p and its left neighbor p_l is likely and thus $u_{(pp_l)}$ is low but the right border between p_l and p is not likely, and thus $u_{(p_l p)}$ is high.

5 Final Step: Disambiguation

In this section we describe the last step of our algorithm. After dense features for all displacements are computed, each pixel gets assigned a particular displacement. There are three cases. If a pixel does not belong to any dense feature, then it is left without correspondence in the final answer. If a pixel belongs to only one dense feature, then it gets assigned the displacement of that dense feature. Finally if a pixel belongs to more than one dense feature, we need to disambiguate between these dense features.

There are two main reasons why a pixel can be in more than one dense feature. First there are some small spurious dense features, which are not reliable and can be ignored. Thus we we ignore all dense features smaller than 10.

Second reason is repeated texture. Fig. 2(a) shows the left image of a stereo scene, Figs. 2(b,c) show in white dense features at disparity 14 and 5, respectively. Consider Fig. 2(b). The largest dense feature found is the lamp, and 14 is its correct disparity. The next in size is an erroneous dense feature to the left of the lamp. It is due to repeated texture of the books, and its correct disparity is actually 5. Consider Fig. 2(c) now. At the correct disparity, the pixels in erroneous dense feature belong to a larger dense feature.

In principle, we could detect and declare ambiguous large overlaps between dense features. However we found

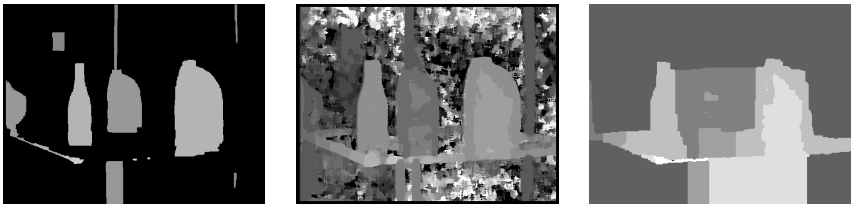

Figure 3. (a) Our algorithm; (b) window based algorithm; (c) graph-cuts algorithm

	Tsukuba			*Sawtooth*			*Venus*			*Map*		
Algorithm	error	density	time	error	density	time	error	density	time	error	density	time
our algorithm	0.36	75	6	0.54	87	13	0.16	73	13	0.01	87	6
method in [11]	0.38	66	1	1.62	76	6	1.83	68	5	0.22	87	2
method in [9]	1.4	45	-	1.6	52	-	0.8	40	-	0.3	74	-

Figure 4. Results on Middlebury stereo database

that the approach in [11] works well. If a pixel p belongs to more than one dense feature, then p gets the displacement of the "densest" feature. That is p chooses the feature which has more pixels in the immediate surrounding of p. Here is how they measure the density of a feature around p. Let $H^{nw}(d,p)$ be the Manhattan distance from p in the north-west direction to the nearest pixel q s.t. q is not in any dense feature. H^{nw} can be computed in one pass over the image for all pixels. Similarly define H^{ne}, H^{sw}, and H^{se} to be the Manhattan distance from p to the nearest q s.t. q is not in any dense feature. Let F_d be the dense feature at displacement d containing p. Then the density of F_d with respect to p is: $density(p, F_d) = H^{nw} + H^{nw} + H^{sw} + H^{se}$. In words, this density measure is the dimensions of the largest piecewise rectangular region around p which lies completely in F_d. So if p is in more than one dense feature, it chooses the feature with the largest density. This will tend to place the repeated texture regions a the correct displacement, since a the correct displacement the repeated texture region tends to be in a larger dense feature (at the wrong disparity, not all repeated texture region pixels get matched).

6 Experimental Results

All experiments are performed with parameters fixed as in Sections 4.1 and 4.2 on 600Mhz Pentium III PC. The running times for other algorithms are as given by their authors.

Consider a stereo pair whose left image is in Fig. 1(a). This is a very challenging scene since most of it is untextured. Figs. 3(a,b,c) show the results of our algorithm, a standard fixed window algorithm, and the global method of [2]. The pixels for which our algorithm does not find an answer are in black. The running time was 25 seconds. Our algorithm finds the disparity for the bottles, table edge and leg, and parts of the background. All the correspondences it finds are accurate, as checked by hand. We do not extract the textureless table top even though it has texture cues on the boundary. This is because the table overlaps several disparities and parts of it are occluded by bottles. The local window algorithm produces a lot of errors. Even the global algorithm in [2], one of the best according to the Middlebury evaluation[1] produces large regions with gross errors, such as between the two largest bottles, for most of the table, and below the table. For the algorithms in (b,c), varying the parameters changes the results somewhat, but does not lead to more accurate answers.

We tried several standard ways to extract reliable correspondences from the dense estimates. First we take disparity estimates only near textured pixels. We tried this with both the window and graph-cuts algorithms. The results are accurate, but significantly sparser than our algorithm. Another way is to take pixels with high local score for the window algorithm. This does not work for the scene above since in most untextured areas wrong correspondences have high local scores. Yet another way is to take pixels whose best local score is significantly higher than the second best score. This essentially reduces to the first approach, taking pixels near texture. Last thing we tried is the left and right consistency principle, that is the correspondence is performed first for the left, then for the right images, and only correspondences consistent across the two results are retained. We tried this with both the window and graph-cuts algorithms. The results improve, but not significantly.

[1] http://www.middlebury.edu/stereo

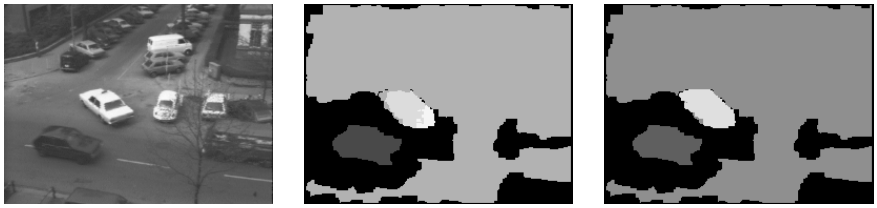

Figure 6. (a) taxi sequence; (b) horizontal motion; (c) vertical motion

Figure 5. Our algorithm

Next we evaluate our algorithm on the Middlebury database with ground truth. The results of our algorithm versus those in [11] and another semi-dense algorithm in [9] are in table 4. The last four columns name each of the four stereo pairs in the database. Each of these columns is broken into three parts. The "error" column gives the total error in the unoccluded regions. The "density" column gives the percentage of matched pixels, and the "time" column holds the running time in seconds. Results in [9] are obviously much worse than ours. We are significantly better both in density and accuracy than [11], especially for the *Venus* and the *Sawtooth* scenes. This is due to the use of graph cuts for feature extraction, our dense features are more reliable than those in [11]. Fig. 5 shows the results of our algorithm on the *Tsukuba* scene, whose left image is in Fig. 2(a). Our algorithm matches some untextured regions, like parts of the table and statue head stand. This is because it takes advantage of the texture cues on the boundaries of these regions to extract them as dense features. Notice that we do not find correspondences in the occluded areas. For the *tsukuba* scene, [10] reports semi-dense results. At density 73%, they have 4.0% errors, which is significantly worse than what we have at 75% density.

Fig. 6 shows the results of our method on the taxi motion sequence. The running time was 13 seconds, 72% of pixels are matched. The algorithm locates correctly two moving cars and most of the background. For the last experiment, we have run our algorithm for two completely unrelated images. As expected, no correspondences were found.

7 Future Work

Our biggest limitation is that a dense feature can only overlap one displacement. If a textureless region overlaps several displacement, we cannot extract it even if there is a strong texture cue on the boundary. Another improvement is occlusion reasoning. Currently if some region occludes a textureless region, we cannot extract the occluded textureless region, even if there is a texture cue on its boundary.

Acknowledgments

We would like to thank Dr. Birchfield, Prof. Scharstein, Dr. Szeliski, and Prof. Tomasi for providing stereo images.

References

[1] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *EMMCVPR02*, page 359 ff., 2002.

[2] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *ICCV99*, page 377ff.

[3] P. Felzenszwalb and D. Huttenlocher. Efficient matching of pictorial structures. In *CVPR00*, pages II:66–73, 2000.

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

[5] W. Grimson. A computer implementation of a theory of human stereo vision. *Royal*, B-292:217–253, 1981.

[6] B. Horn and B. Schunck. Determining optical flow. *AI*, 17(1-3):185–203, August 1981.

[7] N.Ayache and B. Faverjon. Efficient registration of stereo images by matching graph descriptions of edge segments. *IJCV*, 1, 1987.

[8] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search using dynamic programming. *PAMI*, 7(2):139–154.

[9] R. Sara. Finding the largest unambiguous component of stereo matching. In *ECCV02*, page III: 900 ff., 2002.

[10] R. Szeliski and D. Scharstein. Symmetric sub-pixel stereo matching. In *ECCV02*, page II: 525 ff., 2002.

[11] O. Veksler. Dense features for semi-dense stereo correspondence. *IJCV*, 47(1-3):247–260, April 2002.