An Iterative Method for Improving Feature Matches

Johannes Furch
Fraunhofer HHI / Humboldt University
Berlin, Germany
johannes.furch@hhi.fraunhofer.de

Peter Eisert
Fraunhofer HHI / Humboldt University
Berlin, Germany
peter.eisert@hhi.fraunhofer.de

Abstract

Finding reliable and well distributed keypoint correspondences between images of non-static scenes is an important task in Computer Vision. We present an iterative algorithm that improves a descriptor based matching result by enforcing local smoothness. During the optimization process, a Delaunay triangulation of the current set of matches is dynamically maintained. This 2D mesh provides natural neighborhoods and local affine transformations that are used to remove outliers and to resolve ambiguities. The optimization results in a decrease of incorrect correspondences and a significant increase in the total number of matches. The runtime of the overall algorithm is by far dictated by the descriptor based matching.

1. Introduction

Sparse keypoint correspondences are needed for many Computer Vision applications like 3D reconstruction, image-based rendering, tracking, augmented reality or object detection. The task is particularly challenging, if the input data contains dynamic content [11, 8]. The most common approach to address the problem is to employ a keypoint detector and descriptor [15, 1, 18]. The matching procedure usually assigns to each keypoint the nearest neighbor in the multidimensional descriptor space under some consistency constraints. Typical constraints are non-ambiguity (i.e. the nearest neighbor has a large enough distance ratio to the second one) or matching consistency in both directions (see Fig. 1a). Another approach that was mainly used before reliable descriptors were designed utilizes structural information between the keypoints to match similar clusters [19, 17, 7].

Depending on factors like baseline, texturization, scene depth and accuracy of nearest neighbor calculation, the distribution and reliability of the results vary significantly. If a global transformation model can be applied, one typical approach to improve results is to recover the underlying model, e.g. the fundamental matrix [10], by applying a Random Sample Consensus method (RANSAC) on the matching output and using it to remove outliers (see Fig. 1b). However, the fundamental matrix is based on the idealized geometry of a pinhole camera and therefore has to account for other parameters like distortion by introducing a large enough outlier threshold. Being defined globally, this leads to a high sensitivity to noise, which becomes very clear when trying to select correct correspondences from a number of preselected match candidates (see Fig. 1c). More advanced improvements can be achieved by running a sparse 3D reconstruction algorithm [14] and enforcing local smoothness e.g. by thresholding the Laplacian on a mesh spanned over those points. This approach can also be used to resolve ambiguities by selecting the match candidates that lie in close proximity to the 3D surface.

However, for dynamic image content, the above mentioned approaches are not sufficient. For local rigid motion of scene objects, a dominant fundamental matrix consensus can be found and multiple objects can be captured by solving for more than one model. The most common way to address the more general dynamic case is to combine the two matching strategies, i.e. filtering descriptor based matches by enforcing structural similarity between corresponding point clusters. If the structural similarity measure is carefully designed this approach can additionally be used to resolve ambiguities by deciding between a number of match candidates. Multiple methods facilitating rigid and non-rigid structural models have been published [3, 2, 5, 22].

Our approach follows the same general idea. Basic descriptor matching is used to identify match candidates and a selection of initial correspondences. For the points associated with this set, a dynamic Delaunay triangulation introduces a natural neighborhood and affine transformations between the triangles. Based on affine similarities to its neighborhood in the current selection, each match (candidate) can be evaluated. An iterative optimization procedure maximizes the number of selected correspondences while retaining local affinity between the neighbors. For all changes on the selection, an extended version of the dynamic Delaunay triangulation enables fast and localized recalculations of the

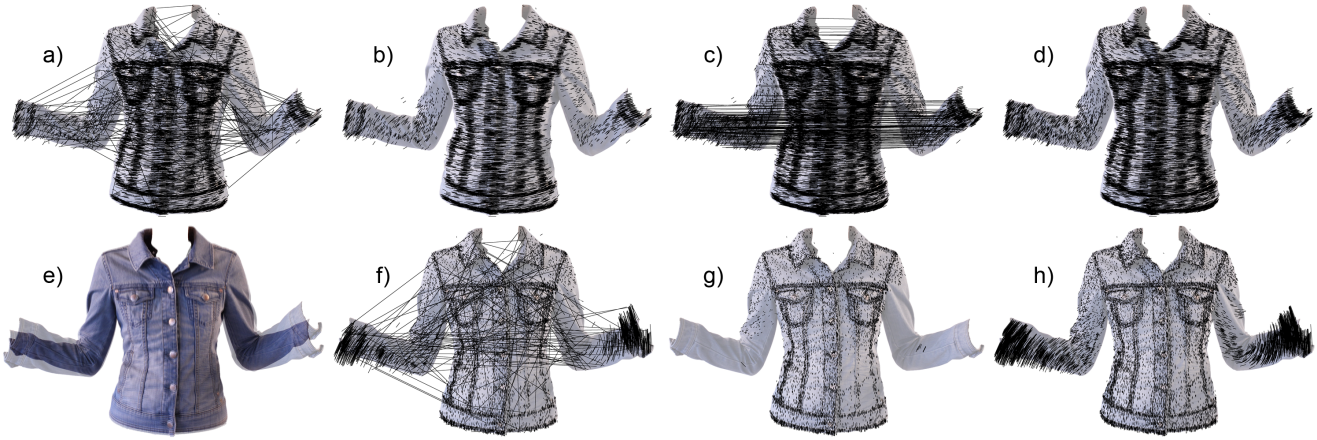

Figure 1. Rigid Camera Motion (a-d) and Non-Rigid Scene Motion (e-h); a) Basic SIFT Matching (10442 matches); b) Fundamental Matrix Filtering (10203 matches); c) Fundamental Matrix Augmentation (15748 matches); d) Our Filtering/Augmentation (10157/14992 matches); e) Input Images; f) Basic SIFT Matching (11621 matches); g) Fundamental Matrix Filtering (10282 matches); h) Our Method (11353/15960 matches)

affinities. The optimization results in a decreased number of incorrect correspondences and a significant increase in the total number of matches. While not being as fast and simple as outlier rejection using epipolar geometry, the runtime of our approach is still by far governed by the keypoint extraction and basic matching process. The only assumption we make in addition to those of the descriptor is local smoothness of the scene content between the images.

2. Iterative Method for Improving Feature Matches

To define the objective, we first construct a graph consisting of keypoints (vertices) and match candidates (edges). Throughout the algorithm, we maintain a set of selected match candidates which represent the most likely matches known in the current iteration. The points associated with the matches in this selection are used to construct a dynamically maintainable Delaunay triangulation in one of the images. When transforming this mesh to the corresponding points, every triangle undergoes an affine transformation. This transformation and the natural neighborhood of the Delaunay triangulation are used to design a weight that is assigned to each match (candidate). The weights reflect the quality of a match (candidate) given the current selection. We introduce extensions to the dynamic Delaunay triangulation that allow us to efficiently update those weights when changes are made to the selection and therefore the mesh. To find an optimal set of correct correspondences, we first remove matches from the selection until all remaining ones have a predefined quality as reflected by the weight. This selection is then augmented by incrementally introducing match candidates that retain the quality of the set.

2.1. Match Candidate Graph

We first extract keypoint sets $\mathcal{P}$ and $\mathcal{Q}$ from the two input images. While our method is independent of the feature point detector and descriptor, we decided to use SIFT due to its objective qualities [16]. We then construct an undirected bipartite graph $G = (\mathcal{P}, \mathcal{Q}, \mathcal{E})$ and assign a weight function $w : \mathcal{E} \rightarrow \mathbb{R}$ to all edges. The edges correspond to matches and match candidates, the weight is a quality measure for their correctness.

We then calculate nearest neighbors for all descriptors in both matching directions. Similar to the basic matching procedure described above, we define a distance ratio threshold t_{dr} to detect ambiguities. A directed correspondence is non-ambiguous if the ratio of the descriptor distances between the first and the second nearest neighbor is smaller than t_{dr}. The aim of our method is to filter out wrong correspondences and to include correct match candidates that are rejected as ambiguous by the basic matching procedure. For this purpose, we also account for the ambiguous match candidates by constructing for each point a set of all k-nearest neighbors whose descriptor distance ratio with respect to the first nearest neighbor is greater than t_{dr}. If two vertices $p \in \mathcal{P}$ and $q \in \mathcal{Q}$ contain each other in their sets of nearest neighbors, a match (candidate) $e = (p, q)$ is added to $\mathcal{E}$. If those sets contain only a single item, the match is added to the initial selection $\mathcal{S}$, which corresponds to the non-ambiguous result of the basic matching procedure.

For a set of matches to be consistent, no two points can have the same corresponding point. In graph terminology, this is reflected by the term "matching". A subset of edges of a bipartite graph is called a matching, if none of the vertices of the graph is adjacent to more than one of those edges [6]. Naturally, the initial selection meets this condi-

tion. Any subset of a matching is again a matching. Therefore, filtering can never destroy the matching property.

The weights $w(e)$ are calculated from structural properties of the points associated with the current selection. If the transformation of neighboring points validates or "supports" a match (candidate), its weight will be high. We say that a match (candidate) e is valid if $w(e) \geq t_v$ for a predefined validity threshold t_v.

We define $\mathcal{S}$ to be a valid selection if it meets the following criteria:

1. $\mathcal{S}$ is a matching
2. $\forall(e \in \mathcal{S})(w(e) \geq t_v)$, i.e. all matches in $\mathcal{S}$ are valid

Note that while every selection has to be a matching to be consistent, not every matching contains only valid correspondences.

The second condition treats the weight to be a binary property that is not affected by its magnitude, because global loss of weight is not necessarily undesirable, as long as all matches $e \in \mathcal{S}$ stay valid. Optimizing the sum of the weights instead would exclude weaker local structures (e.g. borders, dynamics, etc) in favor of strongly connected bigger ones (e.g. planes, etc).

Since our goal is to find as many correct matches as possible, the objective is to find a maximal valid selection. The challenge of this optimization is that potentially all weights can change whenever a match is introduced or removed from the current selection.

2.2. Designing a Meaningful Weight

One very simple example to illustrate the concept of the weight is the perpendicular distance to the epipolar line $w_{fm}(e) = d_{\perp}(\mathbf{F}p, q)$. If the fundamental matrix $\mathbf{F}$ as a global model is estimated robustly from a selection of correspondences using the RANSAC method, its parameters and therefore the weights should not change during filtering. As mentioned above, augmentation with this kind of weight introduces a considerable amount of noise and is therefore not applicable (see Fig. 1c).

Another straightforward weighting scheme assumes that at least some points that are close to each other will keep this property when being transformed to the other image. So we count the number c of k-nearest neighbors of a point whose matches stay k-nearest neighbors of its corresponding point. We then define the weight to be $w_{nn}(e) = c/k$. Those weights can be calculated for the current selection as well as for the candidates and they change dynamically whenever $\mathcal{S}$ changes. Although this is a locally defined measure, it has some major drawbacks. First, even if some points have matches in the same general area, they could still be distributed in an undesirable way due to noise or a reoccurring pattern. Also, proximity is an undirected measure that is drawn to dense clusters of keypoints (i.e. textured areas). Therefore, points in adjacent, less textured areas will not be supported by "natural neighbors" around them, but by the more dense region close by.

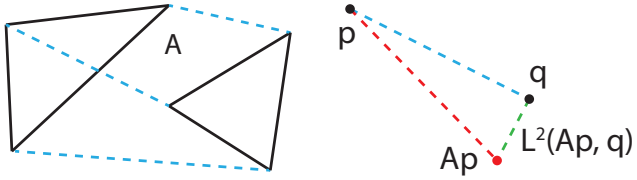

Figure 2. An affine matrix $\mathbf{A}$ calculated from two corresponding triangles and the error distance between the affine transformation of p and its corresponding point q.

To design a more meaningful weight, we make the assumption that close by points lie on a common plane or at least on a smoothly changing surface and therefore undergo a similar transformation between two images. Given that projective effects are negligible in small regions, a set of close points should then undergo a very similar affine transformation from one image to the other. Since an affine 2×3 matrix $\mathbf{A}$ is uniquely defined by the transition of three points, a nearby point can be evaluated using the Euclidean distance $d_{\mathbf{A}}(p, q) = L^2(\mathbf{A}p, q)$ between the affine transformation and the matching point (see Fig. 2).

As mentioned above, it is desirable to include supporting structures from all directions, regardless of their absolute distance. The Delaunay triangulation provides such "natural" neighborhoods. We construct a dynamically maintainable Delaunay mesh for all points $p \in \mathcal{P}$ that are adjacent to matches in the current selection. This introduces an affine transformation for each triangle, which is stored for all faces of the mesh. Choosing a dynamic triangulation reflects the need for dynamic weight recalculation whenever $\mathcal{S}$ is changed.

We say that a face supports $e = (p, q)$ if $d_{\mathbf{A}}(p, q) \leq t_a$ for a predefined affinity threshold t_a and an affine matrix $\mathbf{A}$ calculated from the triangle transformation. For a match e we define the weight $w_{dl}(e)$ to be the count of the supporting outer faces of the star polygon of p (orange triangles in Fig. 4).

2.3. Iterative Optimization Method

Our iterative optimization method consists of two steps. Starting with the initial selection $\mathcal{S}$, a filtering procedure consecutively removes matches from S until the selection reaches a valid state. Following that, an augmentation procedure consecutively inserts match candidates into S that preserve its validity. Augmentation terminates when no such candidates exist anymore.

As for the RANSAC method, the assumption for the it-

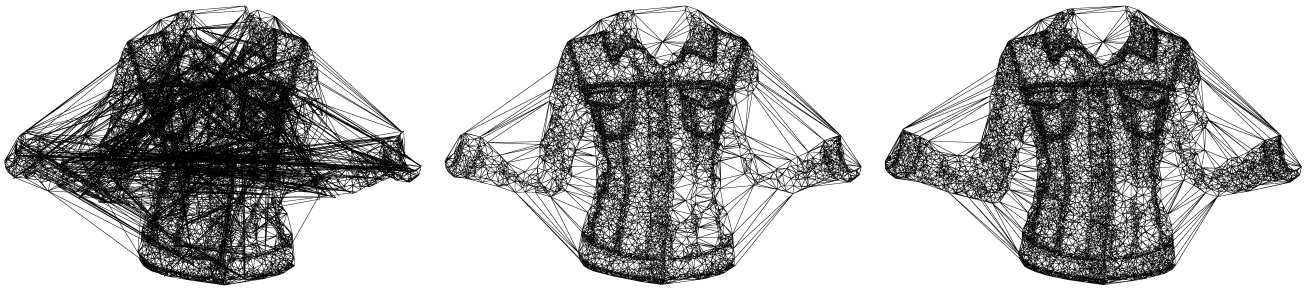

Figure 3. Our iterative optimization ran on the images in Fig. 1e. The mesh is calculated for the first image and and shown for the second one. The left mesh represents the initial selection, the center one the end of filtering and the right one the final solution.

erative procedure is that based on the effectiveness of descriptor based feature matching, the initial selection is fairly well distributed and contains only a few (up to $\approx 20\%$) false correspondences. We therefore assume that $> 80\%$ of the matches in the initial selection can be used to form a valid selection. Considering that the wrong correspondences might "disturb" the rest, the number of invalid matches in the initial selection might be lower than the real number of incorrect matches. It is also assumed that the sparse and well distributed nature of the keypoints and hence the match candidates will in most cases result in at most one candidate being supported by its neighborhood.

For filtering, an ascending, weight sorted, mutable priority queue is filled with all matches of the current selection. In each step, the next match is taken from the queue and checked for validity. If it is not valid, it is removed from $\mathcal{S}$ and the weights in the priority queue are updated. If it is valid, the procedure terminates and the current selection $\mathcal{S}$ is in a valid state.

For augmentation, a descending, weight sorted, mutable priority queue is filled with all match candidates that are not in the current selection. In each step, the next candidate is taken from the queue and checked for validity. If it is valid, it is added to $\mathcal{S}$ and the weights of all candidates are updated in the priority queue. If it is not valid, the procedure terminates and $\mathcal{S}$ is still in a valid state. In fact, the selection stays in a valid state throughout the procedure.

Several exceptions have to be handled during augmentation:

- Inserting a match candidate into $\mathcal{S}$ can destroy the validity of potentially every match in the selection.
- If multiple valid candidates are connected to the same point, their ambiguity is not resolved.
- If a candidate shares an adjacent point with a match in $\mathcal{S}$, its insertion would destroy the matching criterion.

To keep the basic augmentation procedure as simple as possible, we set the weight of a candidate to zero if its insertion would lead to the loss of validity for a match in the current selection, if its ambiguity is not resolved by the weight or if it shares an adjacent point with a match in the current selection. It should be noted that based on the insertion heuristic, the first two cases barely ever occur and the last one can easily be dealt with, whenever a new match is inserted.

Note that while the actual magnitude of the weights is not considered in the objective function, it is used to determine the order in which matches are filtered and augmented. This is due to the assumption that low weighted matches might prohibit other weak ones from getting valid during filtering and that more solidly weighted matches provide a better base for subsequent insertions during augmentation. Also note that this does not prevent weakly supported matches from insertion, they are just added later in the process and with a potentially higher weight.

2.4. Efficient Weight Calculation

Since weights have to be reevaluated whenever the current selection changes, efficient weight calculation is crucial for the efficiency of the overall algorithm. To achieve this, we make use of the fact that changes due to insertion and deletion of a point in a dynamic Delaunay mesh are limited to a fairly small neighborhood of this point. We track those changes and recalculate the weights only for the affected area.

Extended Dynamic Delaunay Triangulation

The implementation used for the dynamic Delaunay triangulation is an extended version of the methods described by [9] and [13]. For insertion of a new point, the algorithm finds the triangle containing this point, includes the point in the existing mesh and flips edges to enforce the Delaunay criterion. For deletion of a point, only edges inside the star polygon of the point have to be flipped to restore a valid Delaunay state (see Fig. 4). Since the insertion procedure requires a fast localization of the triangle containing a certain point, a "history graph" is maintained during the

construction and destruction process. Whenever new faces are added to the triangulation, e.g. through edge flips, they are added as children to the graph nodes representing the faces that were destroyed by the operation. For triangle localization, the graph is traversed until a leaf, i.e. a face that is currently in the triangulation, is reached.

To optimize the tracking of local changes, two major additions have been made to the basic dynamic Delaunay implementation:

For each insertion and deletion operation, sets of faces that were removed ($\mathcal{F}_{rem}$) and inserted ($\mathcal{F}_{ins}$) are provided. Faces that are temporally created during the mesh manipulation are not part of those sets, since they do not contain any valuable information. This way, affine matrices only have to be calculated for the faces in $\mathcal{F}_{ins}$ and weights only have to be updated if they were based on faces in $\mathcal{F}_{rem}$.

To calculate the weight for a match candidate that is not part of $\mathcal{S}$, the associated point has to be inserted into the current mesh. However, since weights are frequently recalculated during optimization, the history graph grows through the repeated insertion and deletion and therefore successively slows down triangle localization, independent on the actual size of the mesh. Also, restoring the original state of the mesh involves recalculation of affine matrices and weights that are already known.

To speed up the weight recalculation, we introduce two additional procedures to the dynamic Delaunay triangulation. They simulate deletion and insertion of a point without changing the original mesh and return a submesh containing the changed faces. The deletion simulation is a straight forward operation. It only involves calculating a Delaunay mesh for the points of the star polygon of the point to be deleted (see red polygon on the left side of Fig. 4). Simulating the insertion is not as easy, since the points involved are not known in advance. However, every iteration of the regular sequential insert function [9] involves checking a single point that has not been visited yet for being in accordance with the Delaunay property. If it is not, it will be affected by the insertion and has to be included into the submesh. The faces are constructed analogously to the regular insertion process. The resulting submesh equals the star polygon of the point when inserted into the current mesh (see red polygon on the right side of Fig. 4). Both functions also return the $\mathcal{F}_{rem}$ set as defined above.

Weight Calculation

The set of supporting faces is saved for each point and the weight is calculated as the size of this set. The weight will only change if any of the faces of the star polygon or its outer faces are deleted. The deletion of any of those faces will trigger a recalculation of the weight.

To calculate the weight and the effect on $\mathcal{S}$ for a match candidate $e = (p, q) \in \mathcal{S}$, we simulate the insertion of p. We extend the resulting submesh, by adding the outer faces of the star polygon from the original mesh. This has some major advantages:

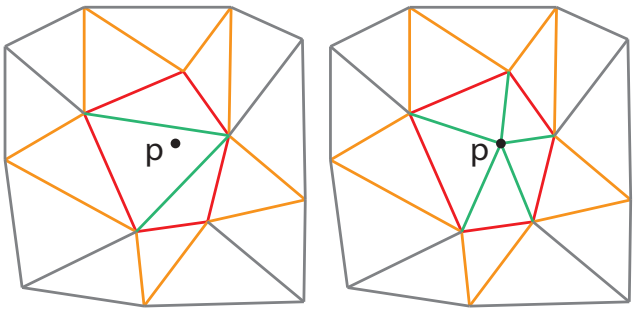

Figure 4. The mesh on the right side shows the insertion of point p into the mesh on the left side. The star polygon of p is drawn in red, the outer faces in orange.

- The weight of e can be fully calculated inside the submesh.
- All points associated with matches whose weights are affected by the insert operation are part of the submesh.
- All weight changes in the original mesh can easily be derived from the "inner weights" of the submesh and $\mathcal{F}_{rem}$.

Since the weight of a match candidate and the structure of the submesh can potentially change whenever a face in $\mathcal{F}_{rem}$ or one of the outer faces is removed, the submesh has to be recalculated whenever this happens. Especially in highly textured regions, which potentially contain a lot of match candidates, this can result in a large number of recalculations for every insertion. Naturally, the faces that trigger reevaluation of affected weights is significantly higher. However, unlike the insertion simulation, this calculation is relatively cheap.

To avoid unnecessary recalculations for very unlikely candidates, we allow for an optional efficient weight estimation. If the estimation stays below the validity threshold t_v, the comparably slow submesh extraction for accurate weight calculation is omitted. Although we do not know which faces will be the outer ones of the star polygon for a candidate point p, we know that they are somewhere close to where p will be inserted into the original mesh. So since we already have a fast triangle localization method available, a meaningful estimation of the weight for the match e associated with p can be derived by looking for support in the surrounding of this triangle. Even if a relatively large area is inspected, the estimation is still considerably faster than the insert simulation. We found that in most cases the estimated weight is very similar to the accurate calculation.

Figure 5. Local Rigid Motion of Multiple Objects; Top to Bottom: Input Images; Original and Transformed Mesh of the Final Solution; Basic SIFT Matching (4585 matches) and Our Method (3556/4682 matches)

3. Experimental Results

For our test results, we used the SIFT and KD-tree implementations of VLFeat [20]. The modified dynamic Delaunay triangulation was implemented utilizing the halfedge data structure of OpenMesh [4].

Since a good keypoint density over the whole image is desired, we do not limit ourselves to keypoints with high local contrast. The errors and ambiguities that are commonly associated with that will be dealt with by our method. Other than that we use standard parameters for SIFT: $noctaves(-1)$, $nlevels(3)$, $o\_min(-1)$, $peak\_thresh(0.001)$, $edge\_thresh(10.0)$, $magnif(3.0)$, $window\_size(2.0)$, $norm\_thresh(0.0)$. To improve the runtime of the nearest neighbor calculation while retaining a good matching performance, we set the maximum number of comparisons for the search to $1/8$th of the number of points in the KD-tree. The number of k-nearest neighbors and therefore the number of potential ambiguous match candidates is set to 8. Since we want a comparably restrictive initialization, we lower the commonly used distance ratio threshold t_{dr} from 0.8 [15] to 0.7. Note that this also affects the lengths of the candidate lists.

For the fundamental matrix, the inlier threshold for the RANSAC estimation is set to 1.0 pixels, the outlier rejection threshold for filtering and augmentation to 4.0 pixels.

Our method has three parameters: the affinity threshold t_a, the validity threshold t_v and estimation depth for the augmentation weight calculation t_e. The last parameter defines the depth of the neighborhood of the containing triangle that is inspected for weight estimation. If t_e is set to 0, accurate calculation is performed for all match candidates. If not noted otherwise, the parameters were set to $(t_a, t_v, t_e) = (4.0, 1.0, 2.0)$.

Applying our approach to the two image pairs in Fig. 1 results in a significant gain of matches. For the rigid camera motion the increase amounts to about 44%, for the dynamic arm motion to about 37% of the initial result. Considering that the fundamental matrix filtering and our own filtering step suggest about $1 - 3\%$ of wrong matches for the basic SIFT matching, the percentage of real gain is even higher. The exceptionally low number of incorrect matches is due to the segmentation of the object and almost no occlusion between the images.

The shuffled books in Fig. 5 contain large untextured areas, leading to a lot of noise during basic SIFT matching. In the filtering step $\approx 25\%$ of the initial matches are removed, and about the same amount is reintroduced during augmentation. The transformed mesh (middle right) illustrates how topology is retained on the labels of the books while it is destroyed between them. For this problem domain, estimating multiple fundamental matrices for filtering could be considered. However, problems might arise due to bended books, small amounts of keypoints per subplane and the proximity of all books to a common plane.

Fig. 6 illustrates the results of our method applied to the well known "Graffiti" dataset [21]. Both, the trajectories and the transformed mesh show how filtering removes obvious mismatches and how augmentation fills in previously uncovered areas even beyond the border of the initial selection. Almost all matches in the final result agree with the ground truth provided by the dataset. However, the homography only reflects the upper part of the images, making it inadequate for a meaningful objective analysis. While for this scene filtering could also be facilitated using epipolar geometry, gaining $\approx 80\%$ of new correspondences clearly highlights the advantage of applying our method. For static scenes, a combination of both approaches could also be considered.

Our method unfolds its full potential on high resolution natural images containing dynamic information. As there is no ground truth for this kind of data, we use the Middlebury "stereo dataset" [12] to objectify the quality of our results. While for the images of "Rocks1" the number of keypoints per image (≈ 30000) and the number of total match candidates (≈ 40000) are considerably lower than for the images in Fig. 1 ($\approx 50000/90000$), the basic SIFT matching yields a significantly larger amount of matches (see Tab 1), suggesting that there is not much more to gain by augmentation. The same holds for other images in this dataset. This fact in mind, the statistics can still serve as a proof of concept for our method. Taking the quantization of the disparities, the localization errors of SIFT and the rounding to discrete pixel locations into account, we decided to classify all match points that are closer than 2 pixels to the ground truth location as "correct", those up to 4 pixels as "undecided" and all others as "wrong". Independent of the parameters used, there is a significant decrease in the number of wrong matches. Also, augmentation always yields more good matches than the basic matching approach. It should be noted that filtering and augmentation results in comparably small error distances (< 50 pixels), while for basic SIFT matching, they go up to ≈ 1500 pixels.

The first three results of augmentation show the effects of weight estimation on the matching performance. Since an estimation depth of $t_e = 2$ will include almost all outer faces of the expected star polygon, the results are very similar to those of the accurate calculation. On the other hand, lowering t_e to 1 prevents a lot of potentially supporting triangles and therefore many valid matches from being considered. However, since the effect on the inclusion of incorrect matches is limited, it is still a useful alternative if runtime is an issue. Compared to accurate calculation, the runtime of the optimization goes down to 13% and 20% for $t_e = 1$ and $t_e = 2$ respectively. For $t_e = 2$ this amounts to about 15% of the runtime of feature extraction for both images.

4. Conclusion

We have presented an iterative optimization method to improve the results of descriptor based feature matching. The only assumption we make about the scene content is local smoothness. We have introduced important extensions to the dynamic Delaunay triangulation that are crucial for the efficiency of the optimization. We have presented results that show the effectiveness of our approach for different image domains. All results show a decrease in the number of wrong correspondences and a significant increase in the total number of matches. While our optimization slightly increases the runtime of feature matching, the execution time of the overall algorithm is still dictated by the descriptor based matching.

$d \leq 2$	$2 < d \leq 4$	$d > 4$	Parameters
Basic SIFT matching			
14118	202	112	
Filtering			(t_a, t_v)
13824	108	15	$(1, 1)$
14085	173	24	$(2, 1)$
14118	202	41	$(4, 1)$
13362	83	13	$(1, 2)$
14039	154	19	$(2, 2)$
14105	200	37	$(4, 2)$
Augmentation			(t_a, t_v, t_e)
15488	177	32	$(1, 1, 0)$
15335	147	31	$(1, 1, 1)$
15456	164	29	$(1, 1, 2)$
15841	293	71	$(2, 1, 0)$
15867	352	97	$(4, 1, 0)$
14814	111	22	$(1, 2, 0)$
15760	251	39	$(2, 2, 0)$
15850	349	83	$(4, 2, 0)$

Table 1. Middlebury "Rocks1" Ground Truth Results.

Acknowledgements

This work was funded by the European Commission under contract FP7-288238 SCENE.

References

[1] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[2] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(4):509–522, Apr.

[3] P. Besl and N. D. McKay. A method for registration of 3-D shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256, Feb.

[4] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. OpenMesh: A Generic and Efficient Polygon Mesh Data Structure. OpenSG Symposium 2002, 2002.

[5] H. Chui and A. Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2):114–141, 2003.

[6] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[7] A. D. J. Cross and E. Hancock. Graph matching with a dual-step EM algorithm. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(11):1236–1253, Nov.

[8] J. Fayad, C. Russell, and L. Agapito. Automated articulated structure and 3D shape recovery from point correspondences. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 431–438. IEEE, 2011.

Figure 6. The "Graffiti" Dataset [21]; Left to Right: Input Images; Basic SIFT Matching (721 matches); Our Filtering (622 matches); Our Augmentation (1123 matches); Top: Trajectories; Bottom: Mesh Calculated for the First and Shown for the Second Image

[9] L. Guibas, D. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[10] R. I. Hartley. In Defense of the Eight-Point Algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(6):580–593, June 1997.

[11] A. Hilsmann, P. Fechteler, and P. Eisert. Pose Space Image Based Rendering. *Comput. Graph. Forum (Proc. Eurographics)*, 32(2):265–274, May 2013.

[12] H. Hirschmüller. Evaluation of Cost Functions for Stereo Matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[13] T. Kao, D. M. Mount, and A. Saalfeld. Dynamic maintenance of Delaunay triangulations. Technical report, College Park, MD, USA, 1991.

[14] M. A. Lourakis and A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.

[15] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[16] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615 –1630, oct. 2005.

[17] H. Ogawa. Labeled point pattern matching by Delaunay triangulation and maximal cliques. *Pattern Recogn.*, 19(1):35–40, Jan. 1986.

[18] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[19] L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):504–519, 1981.

[20] A. Vedaldi and B. Fulkerson. VLFeat: An Open and Portable Library of Computer Vision Algorithms. In *Proceedings of the international conference on Multimedia*, MM '10, pages 1469–1472, New York, NY, USA, 2010. ACM.

[21] Visual Geometry Group, Department of Engineering Science, University of Oxford. Affine Covariant Regions Dataset. `http://www.robots.ox.ac.uk/~vgg/data/data-aff.html`, 2004. [Online; accessed 05/17/2013].

[22] Y. Zheng and D. Doermann. Robust point matching for nonrigid shapes by preserving local neighborhood structures. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):643–649, 2006.