GraphMatch: Efficient Large-Scale Graph Construction for Structure from Motion

Qiaodong Cui[1] Victor Fragoso[2] Chris Sweeney[3] Pradeep Sen[1]

[1]University of California, Santa Barbara
{qiaodong@umail, psen@ece}.ucsb.edu

[2]West Virginia University
victor.fragoso@mail.wvu.edu

[3]University of Washington
csweeney@cs.washington.edu

Abstract

We present GraphMatch*, an approximate yet efficient method for building the matching graph for large-scale structure-from-motion (SfM) pipelines. Unlike modern SfM pipelines that use vocabulary (Voc.) trees to quickly build the matching graph and avoid a costly brute-force search of matching image pairs, GraphMatch does not require an expensive offline pre-processing phase to construct a Voc. tree. Instead, GraphMatch leverages two priors that can predict which image pairs are likely to match, thereby making the matching process for SfM much more efficient. The first is a score computed from the distance between the Fisher vectors of any two images. The second prior is based on the graph distance between vertices in the underlying matching graph. GraphMatch combines these two priors into an iterative "sample-and-propagate" scheme similar to the PatchMatch algorithm. Its sampling stage uses Fisher similarity priors to guide the search for matching image pairs, while its propagation stage explores neighbors of matched pairs to find new ones with a high image similarity score. Our experiments show that GraphMatch finds the most image pairs as compared to competing, approximate methods while at the same time being the most efficient.*

1. Introduction

Recently, structure-from-motion (SfM) algorithms have achieved impressive reconstructions from large photo collections [3, 15, 18, 38, 40]. These exciting results are possible thanks to recent improvements in bundle adjustment [2], motion/camera parameter estimation [6, 7, 11, 13, 14, 16, 17, 21, 22, 26, 27, 32, 44, 46], and feature matching [8, 28].

All modern large-scale SfM pipelines [1, 27, 38] require the building of a *matching graph* in order to reconstruct a scene from a large photo collection. In this graph, images are nodes and edges represent matching image pairs that share visual content that could be used for 3D-reconstruction. Despite the aforementioned advancements, however, the problem of finding high-quality, matching image pairs to form the matching graph remains a major bottleneck for large-scale SfM pipelines. The reason for this is that the vast amount (75 - 95%) of image pairs do not match in most photo collections [47].

An effective, but prohibitively expensive, approach to filter non-matching image pairs is the brute force method: an exhaustive test of all possible $\mathcal{O}(N^2)$ image pairs. To accelerate the search for suitable image pairs to match, several state-of-the-art large-scale SfM pipelines use image-retrieval techniques [9, 30, 31], assuming that image pairs with high visual similarity scores are likely to match. Among the adopted image retrieval techniques, the most implemented in publicly available large-scale SfM pipelines [1, 27, 38] is Vocabulary (Voc.) Trees [30].

Although Voc. Trees help SfM pipelines find suitable image pairs to match more quickly, constructing a Voc. Tree is computationally expensive (due to the computation of a tree encoding visual words and the image index) and can demand a large memory footprint [36, 42]. Specifically, creating the Voc. Tree requires an expensive clustering pass (*e.g.*, k-means [23]) on a large set or subset of local image features (*e.g.*, SIFT [25]) for every node in the tree. For this reason, SfM pipelines construct this tree in an independent process before executing the reconstruction pipeline. In addition, creating an image index requires passing each of the local features of every image in a dataset through the vocabulary tree, which is also computationally expensive. Another disadvantage of Voc. trees is that SfM pipelines have to load the image index in memory to find suitable image pairs to match in a large photo collection. In this scenario, substantial memory footprints are necessary since the image index is large as well. SfM pipelines can reduce the memory footprint by creating a coarse Voc. tree. However, a coarse Voc. tree typically decreases matching performance.

Since Voc. Trees were devised for image-retrieval applications and not for SfM, they only find a fraction of the

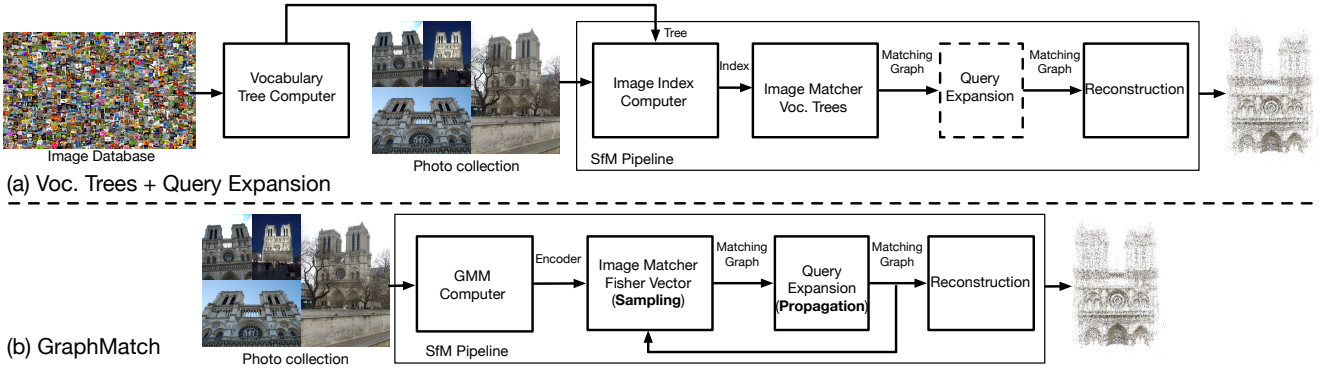


Figure 1. **(a)** Modern, state-of-the-art SfM pipelines, such as the Building-Rome-in-a-Day (BRIAD) pipeline [1], accelerate the search for potential image pairs using Vocabulary (Voc.) Trees. However, this requires an expensive offline stage that clusters a large database of image features to construct the Voc. Tree, and also has to build an index of the images which consume a large amount of memory. Furthermore, finding potential image matching pairs with Voc. Trees can produce a non-dense matching graph that can affect reconstruction quality. To reduce this effect, pipelines such as BRIAD use a *query-expansion* step, which uses the current matching graph to identify new potential pairs. **(b)** On the other hand, our proposed GraphMatch approach does not use Voc. Trees, and hence eliminates this expensive offline stage. Instead, GraphMatch uses Fisher distances [33, 35] to measure image similarity, since we find that they are better priors for finding matching pairs than Voc. trees similarity scores. GraphMatch then follows an iterative "sample-and-propagate" scheme, where the sampling process finds suitable image pairs to match and the propagation step uses the current state of the matching graph and image similarities to discover new matching pairs.

potentially matching pairs (see Sec. 5), which decreases reconstruction quality. To increase the number of potentially matching image pairs computed by the Voc. trees, some SfM pipelines such as the Building-Rome-in-a-Day (BRIAD) pipeline [1] execute a *query-expansion* step after building an initial matching graph (see Fig. 1a). This idea, drawn from work on text/document retrieval [10], is based on the observation that if images i and j are matched, and image j is also matched with image k, then potentially image i could be matched with k.

In BRIAD, potentially-matching image pairs are found by applying query expansion a repeated number of times. This approach has a fundamental problem, however, in that repeated query expansions lead to *drift* [1], where the context of the images found rapidly diverge from that of the initial image, especially as it starts examining the neighbors of the neighbors, and so on. Although the geometric verification step in the matching process ensures that these pairs are not admitted into the graph as edges, it greatly reduces the efficiency of the overall matching process because it tests more pairs unlikely to share edges. In other words, in approaches like BRIAD, the more it performs query expansion, the less efficient the algorithm becomes.

To combat these problems, in this work we present a new algorithm called *GraphMatch* (see Fig. 1b), an efficient, approximate matching algorithm that significantly accelerates the search for image pairs in SfM pipelines. GraphMatch is inspired by the PatchMatch algorithm [4], which has been used successfully for accelerating the search for patch-wise correspondences between two images. Compared to algorithms such as BRIAD, GraphMatch has two key differences which greatly improve its performance.

First, GraphMatch does not use Voc. trees, so it does not require an offline expensive stage to construct them or the image index. Instead, we use a score computed from the distance between Fisher vectors [33, 35] of any two images. Our experiments demonstrate that Fisher vector distances are faster to compute and more indicative of a possible match than Voc. Tree similarity scores, or those of other descriptors such as VLAD [20] or GIST [31].

Second, we use an alternative approach to the query expansion step that is based on the PatchMatch algorithm [4]. This alternative approach maximizes the fraction of matching pairs and ensures high-quality reconstruction. PatchMatch makes a similar observation to that of query expansion: if patch i in one image and patch j in another are matched, then the neighboring patches of patch j are likely to be matches for patch i. This information is used by PatchMatch in an iterative "sample-and-propagate" scheme to efficiently compute the correspondence fields between patches of two images. First, the correspondences of each patch are initialized to point to random patches in the sampling step. Then in the propagation step, the valid correspondences found in the previous step are propagated to their neighbors. This algorithm iterates until the full, approximate correspondence field has been found.

GraphMatch implements a similar iterative "sample-and-propagate" algorithm which builds the matching graph incrementally. However, rather than doing random sampling as in PatchMatch, the sampling stage of GraphMatch uses Fisher scores to search for matching image pairs more effectively. The propagation stage then uses the current

graph of matched image pairs to explore the neighbors of images belonging to geometrically verified image pairs. Unlike the query expansion in BRIAD which is only executed after the graph has been constructed and therefore suffers from drift, GraphMatch alternates sampling and propagation in an iterative fashion to find suitable image pairs.

The "sample-and-propagate" scheme of GraphMatch provides an excellent balance between matching images with similar visual appearance (the sample stage) while also taking advantage of local connectivity cues (the propagate stage). As a result, GraphMatch is able to efficiently determine which image pairs are likely to produce a good match, avoiding unnecessary overhead in attempting to match image pairs which are unlikely to match.

2. Previous Work

Determining *correspondence* between pairs of images is a critical first key step in recovering scene geometry through SfM. Typical SfM pipelines first detect point features such as SIFT [24] features, then perform feature matching between SIFT descriptors in pairs of images to estimate their epipolar geometry. Pairs of images which produce a sufficient number of feature correspondences that pass an additional geometric verification step are then used during SfM to generate feature tracks. Given an input dataset of N images that we wish to reconstruct, brute-force matching approach exhaustively tests all possible image pairs leading to an $O(N^2)$ algorithm. However, most image pairs do not lead to successful matches. Large-scale SfM pipelines can gain efficiency by leveraging information that helps the pipeline identify image pairs that are more likely to match and filter image pairs that are unlikely to match. Wu [47] measures this likelihood by matching only a subset of features, observing that an image pair is likely to match when these subset-features produce a sufficient number of image correspondences. While this strategy provides a significant increase in efficiency, it can yield to disconnected reconstructions [47] and is still requires testing every image pair.

To mitigate the cost of exhaustive matching approaches, image retrieval techniques have been employed to efficiently determine a set of candidate image pairs for feature matching based on image similarity. Vocabulary (Voc.) trees [30] are frequently used to efficiently match datasets. A vocabulary is learned (typically from clustering image features), and the visual words are organized into a tree that may be efficiently searched. Term Frequency Inverse Document Frequency (TF-IDF) is used to determine the similarity of images using inverted files [30]. Voc. Trees are highly scalable due to efficient search of the tree structure. Additional image retrieval works have used a learning-based approach to predict which images contain visual overlap. Schönberger *et al*. [37] uses a machine-learning-based approach dubbed PAIGE to predict which images contain visual overlap. PAIGE trains a random forest (RF) [19] using image-pair features to predict the pairs with scene overlap.

Image retrieval techniques offer efficient performance but typically are optimized based on precision and recall, not for the final SfM reconstruction quality. As such, these methods are suboptimal for use with SfM. Shen *et al*. [39] proposed a graph-based approach for SfM image matching where image pairs to match are chosen based on a combination of appearance similarity and local match graph connectivity. After an initial minimum spanning tree of matches is built, the minimum spanning tree is expanded to form strongly consistent loops (determined from pairwise geometry between images in the loops) in the match graph. Finally, community detection is used to densify the matching graph efficiently given the match graph structure after strongly consistent loops are added.

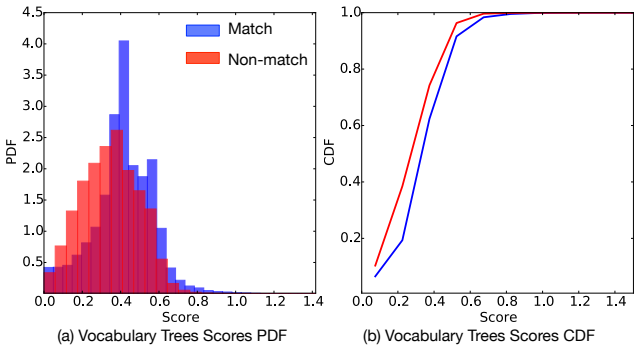
Our approach, GraphMatch, is of similar spirit to the method of Shen *et al*. because it also considers visual similarity and match graph connectivity priors. Shen's algorithm, however, aims to find the fewest number of matches possible that provide a strong loop consistency in the underlying matching graph while still yielding high quality reconstructions. Thus, they attempt to find a minimal stable set of edges that yield high quality reconstructions. GraphMatch, on the other hand, aims to accelerate the search for valid matches to be as efficient as possible (by minimizing the number of "bad" image pairs tested) so that as many good matches may be found as efficiently as possible. By finding as many valid matches as possible, GraphMatch is able to avoid disconnected reconstructions that plague alternative methods due to weak or missing connectivity [47] and recover more cameras in the resulting SfM reconstruction (*c.f*. Fig. 6). Indeed, our algorithm typically is able to recover 80-90% of the total good matches found by the baseline in just a fraction of the time. Further, GraphMatch uses simple-yet-effective priors that yield an easy-to-implement and efficient algorithm.

3. Finding Image Pairs with Informative Priors

The task of finding all geometrically verified, matching image pairs in a large database of N images can be posed as the problem of discovering a matching graph. In this graph, vertices represent images, and edges exist when pairs of images contain a sufficient number of geometrically verified feature matches.

The brute force approach leads to poor performance for large N because of its $O(N^2)$ complexity. However, it is possible to obtain high quality reconstructions by using *approximate matching algorithms* that only use subsets of potential edges [41]. These approximate methods are extremely useful for large-scale SfM applications when they discover a large percentage of edges quickly.

The complexity of finding suitable image pairs with ap-



(a) Vocabulary Trees Scores PDF (b) Vocabulary Trees Scores CDF

Figure 2. Probability density functions (PDFs) and cumulative distribution functions (CDFs) of geometrically-verified matching image pairs (edges) and non-matching image pairs (non-existing edges) as a function of Vocabulary Tree scores. The support of the PDFs and CDFs of valid and non-existing edges overlap significantly.

proximate methods depend on the density of the matching graph. The lower the density (or higher the sparsity), the more challenging for algorithms to find image pairs. To alleviate this issue, existing SfM pipelines use Voc. trees as a way to generate image pairs with a high similarity score. Unfortunately, the image representation that Voc. trees use provides limited information about matching image pairs. This is confirmed by observing the distributions of the Voc. trees similarity scores for matching and non-matching image pairs of several large-scale photo collections: NYC LIBRARY, MONTREAL NOTRE DAME, MADRID METROPOLIS, ALAMO, TOWER OF LONDON, ROMAN FORUM. We can observe in Fig. 2 that the distributions of Voc. trees similarity scores for matching and non-matching image pairs overlap significantly. The larger the separation between these distributions, the more useful information to find matching image pairs.

For our algorithm, we begin exploring better priors for edge-finding by studying metrics for global image similarity. In particular, we tested VLAD [20] and Fisher vectors [33, 35]. As Fig. 3 shows, we found that Fisher scores gave us the best separation between the distributions of edges and non-edges, meaning that they provide better guidance towards sampling edges. While the fisher vector is effective, other priors like PAIGE [37] may be used.

For our second prior, we were inspired by query-expansion [10] and the PatchMatch algorithm [4], which has been very successful in image processing tasks [5, 12]. PatchMatch accelerates the process of finding matching image patches with a clever, two-stage algorithm. First, it randomly samples the correspondence field by testing each patch in the first image with a *random* patch in the source. Then, the propagation step will exploit the few good matches that the sampling step found to generate new correspondences. The basic idea of propagation is that neighboring patches in the first image usually correspond to neighboring patches in the second image. Therefore, this stage propagates good matches throughout the image. The algorithm then iterates, repeating the sampling and propagation stages until the approximate correspondence field between the two images is quickly discovered.

In this work, if two vertices A and B are neighbors (*i.e.*, are connected by an edge in the graph), we hypothesize that the neighbors of B might also be neighbors for A and vice-versa. Indeed, in practice we see that this is the case (see Fig. 4), where we show the distributions for edges and non-edges as a function of the modified graph distance between the vertices. This fact makes sense for graphs in structure-from-motion applications, where vertices that share an edge usually represent images that are captured in close spatially proximity, and therefore tend to have commonality with other images that are also in close proximity because of spatial coherence.

Given these two priors, we propose to use them in an alternating algorithm we call *GraphMatch*. Unlike existing SfM pipelines that execute a sampling-like step using Voc. trees followed by a query-expansion step, GraphMatch iteratively executes a sampling-and-propagation steps until the current matching graph satisfies certain criteria.

4. Proposed Algorithm: GraphMatch

Like PatchMatch, GraphMatch has two main steps: *sampling* and *propagation*. The purpose of its sampling stage is to find previously unexplored matching pairs, which will connect regions of the graph that had not been connected before. Unlike PatchMatch, rather than exploring completely at random, we guide our search towards candidate edges that are more likely to be valid by using the Fisher-distance prior discussed in the last section. The propagation step in GraphMatch aligns well with a query-expansion step and serves a similar purpose to the one in PatchMatch. The goal of the propagation step is to exploit current matching image pairs to discover new matching pairs.

There are some fundamental differences between PatchMatch and GraphMatch. For example, in PatchMatch the neighborhood of each patch is clearly defined by the parametrization of the underlying pixel grid of an image. However, in the graph discovery case, it is not clear what the "neighborhood" of a vertex is, especially since the graph has not been fully discovered. Therefore, inspired by the query-expansion, GraphMatch uses all the current, direct neighbors of a vertex (*i.e.*, the set of vertices that currently share a valid edge with the vertex) as the potential neighborhood of candidates that will be tested for more matches. Unlike query-expansion, however, GraphMatch identifies the candidate image pairs to test, ranks them based on their Fisher distances, and selects a subset as the pairs to geometrically verify. Thus, the propagation step in GraphMatch

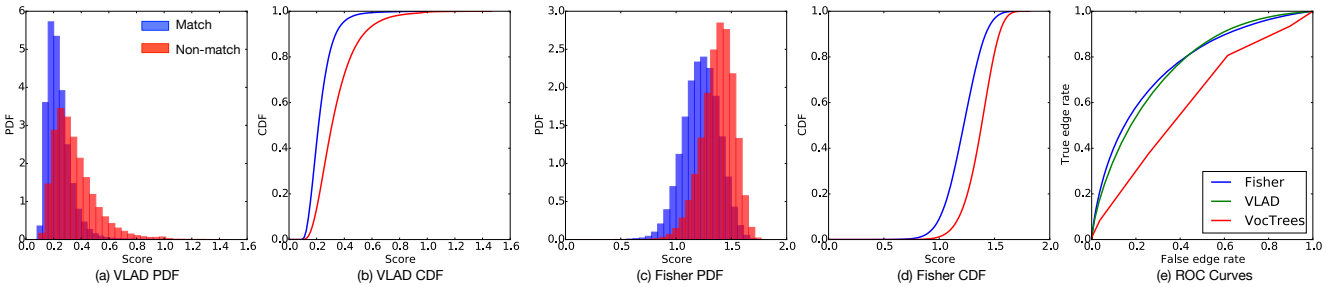


Figure 3. PDFs ((**a**) and (**c**)) and CDFs ((**b**) and (**d**)) of edges and non-edges over scores of VLAD ((**a, b**)) and Fisher ((**c, d**)). The overlap between the PDFs of the edges and non-existing edges corresponding to Fisher and VLAD scores is less than that of the Voc. Trees scores. The ROC curves shown in Fig. **(e)** confirm this. The curves corresponding to Fisher (blue) and VLAD (green) are above of the curve corresponding to Voc. Trees scores (red). Also, Fig. **(e)** shows that Fisher scores tend to be better than VLAD scores for predicting edges when the false edge rate is less tha 0.4. Consequently, Fisher scores are the best prior for edge prediction considering a low false edge rate.

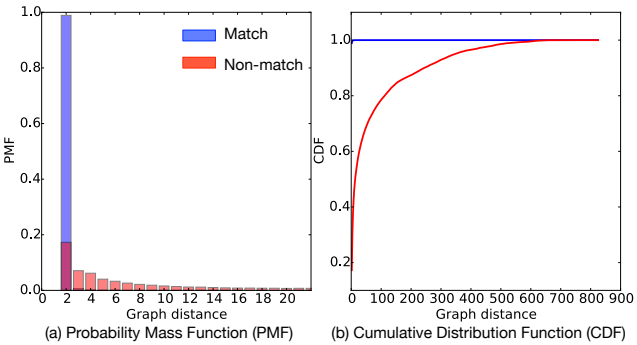


Figure 4. Probability Mass Functions (PMFs) and CDFs for edges and non-edges as a function of a modified graph distance between every pair of vertices (A, B) in the graph. If no edge exists betwen A and B, we simply use their normal graph distance (number of edges in their shortest path). But if an edge exists, we remove it before computing their graph distance (otherwise it would always be '1'). The PMF plot **(a)** shows that (A, B) are much more likely to share an edge when the distance is 2 (i.e, they have a neighbor in common), which constitutes the foundation of the propagation step in GraphMatch. The CDFs plot **(b)** shows that non-matching pairs have a higher modified graph distance since its CDF (red curve) grows slower than that of the matching pairs (blue curve).

combines cues from the current state of the matching graph and image similarity priors.

Another fundamental difference between GraphMatch and PatchMatch, is that PatchMatch finds a single correspondence for a given query patch. In contrast, in the graph discovery case, each image can have multiple matching images. Thus, GraphMatch generalizes PatchMatch in this sense by adjusting the sampling and propagation algorithms to test multiple images in the same iteration, and allow each image to have multiple matches.

In the following subsections, we describe the different stages of our algorithm in detail, which takes in as input a large collection of images $\mathbf{I} = \{I_1, \ldots, I_N\}$ and outputs an approximate connectivity graph $\mathbf{G}$. Complete pseudocode can be found in the supplementary material.

4.1. Pre-processing

The GraphMatch algorithm first pre-computes SIFT features [25] for all N images in the collection to be used later on for the geometric verification process. Unlike existing SfM pipelines that use Voc. trees to represent images and measure image similarity, GraphMatch uses Fisher vectors [33, 35]. It uses them for the following reasons: 1) outperform Voc. trees-image-based representations for image retrieval [34]; 2) require only a single clustering pass instead of several passes as in Voc. trees; 3) reveal more discriminative information about matching image pairs than Voc. trees as shown in Fig. 2.

A fundamental difference between Voc. trees-based image matchers and GraphMatch is that Voc. trees-based methods compute the tree using a large database of image features (*e.g.*, SIFT) and several clustering passes in an offline stage. In contrast, GraphMatch only uses the input photo-collection to compute image representations via Fisher vectors as part of the same SfM pipeline. Thus, after computing the local image features of the input photo collection, GraphMatch proceeds to compute a Fisher vector for every image. To do so, GraphMatch first constructs a database of image features by taking a random sample of features for every image in the photo collection. Then, it estimates the cluster priors, diagonal-covariance matrices, and centroids that parametrize the Gaussian Mixture Model (GMM) [29]. Note that the GMM parameter estimation phase is the only clustering pass to the data in GraphMatch. Using the estimated GMM, GraphMatch computes a Fisher vector for every image in the input photo collection using a Fisher-vector encoder. GraphMatch uses the efficient and multi-threaded GMM estimator and Fisher encoder included in the VlFeat [45] library.

Once every image has its Fisher vector, GraphMatch computes a distance matrix between every pair of images. To compute this matrix efficiently given its $O(N^2)$ complexity, GraphMatch exploits the symmetry in the matrix

(*i.e.*, $d(A, B) = d(B, A)$), avoids computing the distance of an image with itself, and computes the matrix in parallel. Consequently, GraphMatch ends up computing half the matrix quickly by using multi-threaded schemes. The distance estimation only involves subtracting two vectors of 4096 floats and taking the dot product of the result with itself. We observed in our experiments that this matrix evaluation takes at most 3% of the total matching time, and it is not a bottleneck in our approach. The clustering approach in [15] can be leveraged to further decrease the complexity of this step.

Finally, the last step of the pre-processing stage uses the Fisher distances to create a list for every image that ranks all other images based on proximity to the given image. In our case, images at the beginning of the list have smaller Fisher distances (are closer or similar) to the image in question. GraphMatch implements this ranking step also using efficient multi-threaded schemes. Once the algorithm has completed the pre-processing step, it begins the main body of the algorithm, which iterates between a sampling step and a propagation step and continues until no sampling and propagation has occurred.

4.2. Sampling Step

The sampling step attempts to find connections between new regions in the graph by testing new potential edges. In the original PatchMatch algorithm this testing was done at random, but in our case we find that the Fisher distance prior (see Fig. 3) helps guide the sampling and improves the efficiency of the algorithm by increasing the probability that edges are found.

In order to sample based on Fisher distance, GraphMatch uses the ranked lists for every image that were precomputed in the pre-processing step and pulls a fixed number of candidate images from each list at every iteration to be used for sampling. The number of images to be pulled is controlled by two parameters: MAXTESTSFORSAMPLING, which controls the maximum number of times GraphMatch can test an image before it stops sampling altogether, and NUMBERSAMPLEITERATIONS, which governs the maximum number of iterations for which it does sampling. The number of samples for each image per iteration is simply set by the division of these two numbers.

Note that GraphMatch only samples from a vertex if it has less than MAXNUMNEIGHBORS, which is part of our termination criteria. However, this vertex can still be chosen to sample to (*e.g.*, if it is high on another vertices sampling list). Once the list of sampling pairs has been computed for all vertices in the graph, GraphMatch passes it to a function that will test each one for edges using geometric verification. Those that are found to have edges are added to the graph and to a data structure that tracks all neighboring vertices for each image. Furthermore, the vertices tested are removed from the ranked lists of the appropriate vertices so they are not tested again.

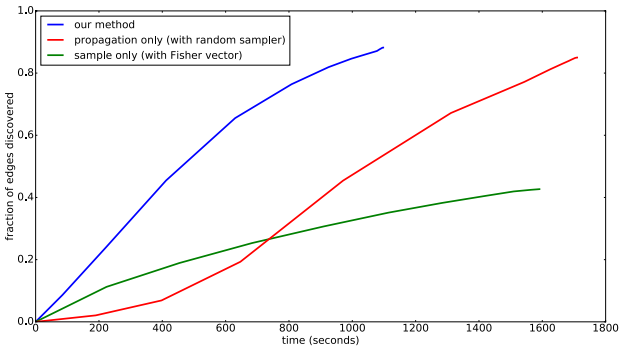


Figure 5. The fraction of discovered edges for GraphMatch with different configurations: 1) GraphMatch with sample and propagation stages on (blue); 2) GraphMatch with only the sampling stage (green); and 3) GraphMatch with only propagation stage (red). GraphMatch with both stages on outperforms the other two configurations. GraphMatch leverages the information from both stages to discover edges quickly.

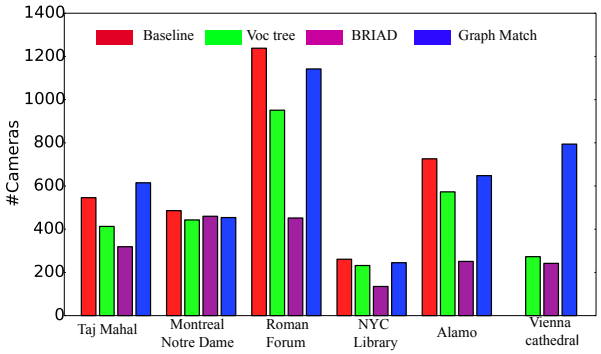


Figure 6. Number of cameras reconstructed for each method for all scenes in Table 1. GraphMatch consistently reconstructs most cameras and is very close to baseline.

4.3. Propagation Step

The goal of the propagation step is to identify new edges by leveraging the spatial coherence normally found in the matching graphs for SfM and image similarities computed via Fisher vectors. The propagation step loops over every pair of vertices with known edges in graph $\mathbf{G}$. Given a pair A and B that share an edge, the propagation step takes the top *ranked* neighbors of B (based on their Fisher distance) and tests them against A and vice-versa. Note that GraphMatch propagates only from vertices that have less than MAXNUMNEIGHBORS for our termination criterion. The number of vertices selected to propagate from each neighbor is given by the parameter NUMTOPROPAGATE.

5. Experiments

We implemented GraphMatch in C++ using the Theia SfM [43] library's API, and incorporated it into Theia's re-

Dataset	# images / graph density	Alg.	# recon cameras	# edges	Position Error (m) avg.	Position Error (m) median	Time (min) pre	Time (min) match	Time (min) recon	Time (min) total	Speed Up match	Speed Up overall
TAJ MAHAL	1497	baseline	576	63,474	-	-	0	696.62	38.81	754.62	1×	1×
		voc. tree	413	9,368	0.04	0.03	10.04	37.39	53.60	109.23	12.80×	6.91×
	0.0437	BRIAD	319	21,332	0.04	0.03	10.04	26.98	13.14	58.19	15.79×	12.97 ×
		ours	615	48,949	**0.03**	**0.02**	0.87	41.78	57.62	108.54	14.02×	6.95 ×
MONTREAL N.D.	2298	baseline	486	33,836	-	-	0	1556.23	51.30	1643.39	1×	1×
		voc. tree	431	18,060	0.08	0.07	18.35	56.66	37.16	123.04	18.44 ×	13.36 ×
	0.0100	BRIAD	430	8,241	**0.06**	**0.03**	18.35	32.51	52.50	113.82	25.77×	14.44 ×
		ours	460	31968	0.06	0.05	1.39	48.90	48.62	109.65	26.01 ×	14.99 ×
ROMAN FORUM	2364	baseline	1247	52,155	-	-	0	1839.29	58.78	1939.45	1×	1×
		voc. tree	951	15,795	0.67	0.56	20.48	91.88	71.50	196.11	15.03×	9.89 ×
	0.0159	BRIAD	452	26,730	**0.03**	**0.02**	20.48	61.47	35.68	129.37	19.96 ×	14.99 ×
		ours	1142	50,216	0.12	0.09	1.39	78.80	52.81	145.04	20.35 ×	13.37 ×
NYC LIBRARY	2550	baseline	261	15,241	-	-	0	1065.78	24.98	1119.26	1×	1×
		voc. tree	232	7,639	**3.82**	**2.28**	20.18	56.94	18.37	107.18	12.25×	10.44 ×
	0.0039	BRIAD	135	4,327	4.18	2.76	20.18	39.44	9.02	80.03	15.30×	13.99 ×
		ours	245	13,427	4.21	3.11	1.40	41.46	24.53	78.82	20.09×	16.67 ×
ALAMO	2915	baseline	726	62,793	-	-	0	2506.32	85.31	2646.70	1×	1×
		voc. tree	573	19,932	0.21	0.08	27.64	73.37	44.93	160.47	22.04×	16.49 ×
	0.0122	BRIAD	251	12,490	0.20	0.10	27.64	41.28	6.33	89.25	30.64 ×	29.66 ×
		ours	648	50,943	0.13	0.04	1.61	61.29	65.82	143.04	33.07 ×	18.50 ×
VIENNA CATHEDRAL	6288	voc. tree	273	10,578	-	-	117.36	450.80	20.05	624.96	-	-
		BRIAD	242	17,578	-	-	117.36	216.60	22.94	389.75	-	-
	0.004	ours	794	79,394	-	-	3.37	367.58	44.28	450.60	-	-

Table 1. Results of timing and camera positions errors for different algorithms on different scenes. Note the preprocessing time does not include time to extract the sift features.

construction pipeline. All experiments were run on a machine with 128 GB of RAM, 2.6 TB of storage space using SSDs, and 2× Intel Xeon at 2.60GHz each with 8 cores. Since both our implementation and Theia library use multi-threading, we launched our SfM pipeline using 32 threads.

The experiments were performed using datasets formed from internet photo collections obtained from crawling Flickr [46]. These photo collections pose interesting challenges to the SfM pipelines because their size ranges from $1497 - 15685$ and the density of the underlying matching graph (*i.e.*, the number of good matches out of all possible image pairs) ranges from 0.04 to less than 0.004. As discussed in Section 3, the density determines the chances for an algorithm to find image pairs. The lower the density, the more challenging to find image pairs, and vice-versa.

We compare our approach to the brute-force method (the baseline), the commonly used Voc. trees method, and the building Rome in a day (BRIAD) matching scheme which implements a Voc. trees followed by a query-expansion. We compare their timings for pre-processing, matching, and total reconstruction. In addition, we measure the reconstruction quality using the discovered matching graph by computing the number of recovered cameras, number of edges in the matching graph (*i.e.*, good matches), and in most cases the mean and median distances of the cameras positions to their corresponding camera positions computed with the baseline method. For the baseline method, we used the multi-threaded brute-force (exhaustive search) implementation provided by Theia. For Voc. Tree-based methods (plain Voc. trees and BRIAD), we used a pre-computed and publicly available tree [1] with 1 million visual words, and indexed each dataset using the multi-threaded COLMAP SfM [38] library. Also, BRIAD retrieved $k = 40$ NN for each image while Voc. tree method use $k = 120$. The timings for the pre-processing phase of Voc. trees-based methods only consider the time for creating the image index, since we use a pre-computed tree. In BRIAD, we executed four times the query-expansion as suggested by Agrawal *et al*. [3]. We did not consider the pre-emptive method by Wu [47], since Schönenberger *et al*. [37] reported that Voc. trees outperform such a pre-emptive method.

The results of each method can be seen in Table 1. Among the approximate methods (*i.e.*, BRIAD and Voc. trees), our algorithm achieves the highest reconstruction quality. This is because it recovers the highest number of cameras and the highest number of edges in the matching graph; see #recon-cameras and #edges columns and Fig. 6. Our algorithm recovers a great number of cameras and is the most comparable to that of the baseline. Thus, GraphMatch is able to consistently find more edges than Voc. Tree-based methods because our algorithm is much more efficient than Voc. trees at finding good image pairs to match. This leads to more stable SfM reconstructions with longer feature tracks and better visual coverage [44]. This can be seen in the Taj Mahal dataset, where the Voc. Tree method recovers very few edges and is only able to reconstruct 68% of the number of cameras with respect to the baseline. See supp. material for results on more datasets.

We also computed the efficiency of the four algorithms as the ratio between the number of found matching image pairs and the total number of tested image pairs. The average efficiency of the baseline was 0.021, which is exactly the same as the average density for all scenes as expected. The average efficiency for Voc.Trees-based methods was slightly higher at 0.08, and ours was 0.27 (nearly $10\times$ more

[1] http://people.inf.ethz.ch/jschoenb/colmap/

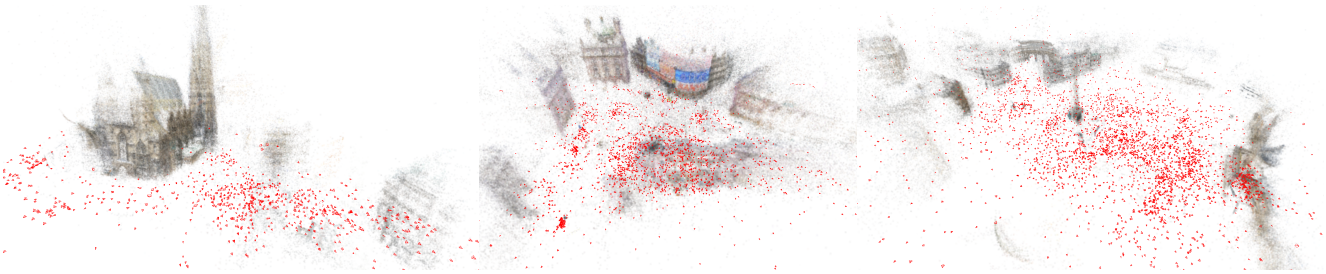


Figure 7. Reconstructions of three large datasets, from left to right: VIENNA CATHEDRAL (6,288 images), PICADILLY (7,351 images), TRAFALGAR (15,685 images).

than baseline). Furthermore, we broke down the efficiency by stage for our algorithm. We found that the sampling step was 0.08 efficient, while the propagation step was 0.37 efficient. The fact that nearly one third of the edges tested by propagation are valid given that the average density of all the scenes is around 0.021 underscores the importance of our propagation step. To understand the effect of iteratively alternating sampling and propagation steps, we measured the fraction of edges discovered over time with both stages alternating each other, with only sampling stage on and with only propagation stage on. As shown in fig. 5, our alternating scheme discovers edges better. Thus, both steps complement each other to boost the edge discovery.

To study whether a trained Voc. trees per datasets could improve its performance, we trained Voc. trees for the dataset TAJ MAHAL and ROMAN FORUM. We found that the matching efficiency and reconstruction quality remained basically the same. It reconstructed 422 and 975 cameras respectively, while the tree from COLMAP reconstructed 413 and 951 cameras. The training time for these two datasets was 0.9 and 1.5 hours, respectively. This brings the total pre-processing time to 1.1 and 1.8 hours, respectively, while fisher vectors required only 0.9 and 1.4 minutes to train for the same datasets, respectively; an approximately 70x speedup compared to Voc. Trees. This suggests that training the Voc. Tree on specific scenes is not a good option for large-scale SfM, especially when compared to the proposed approach.

To assess the reconstruction quality of large-scale photo collections using GraphMatch, we reconstruct three additional datasets: VIENNA CATHEDRAL (6288 images), PICADILLY (7531 images), and TRAFALGAR (15685 images). The SfM pipeline with our algorithm required the following reconstruction timings: PICADILLY in 9.7 hours (estimated 163.6 hours with the baseline) and TRAFALGAR in 16.0 hours (estimated 835.4 hours with the baseline). For more results and analysis please see the supp. material.

6. Conclusion

In this paper, we have presented a novel algorithm called *GraphMatch* for image matching in structure-from-motion pipelines that use scene priors to efficiently find high quality image matches. It iteratively searches for image matches with a "sample-and-propagate" strategy similar to that of PatchMatch [4, 5]. During the sampling stage, we use priors computed from Fisher vector [33, 35] distances between images to guide the search for image pairs which are likely to match. Inspired by the query-expansion step [3], the propagation stage exploits local connectivity and image similarities to find new matches based on neighbors of the current matches. This strategy is able to efficiently discover which image pairs are most likely to yield good matches without requiring an offline training process. Our experiments show that GraphMatch achieves the highest number of recovered cameras and the highest efficiency (*i.e.*, the ratio between the matching image pairs and number of pairs tested), while maintaining equivalent speed-ups compared to that of Voc. tree-based methods without considering the time of the tree construction. Consequently, GraphMatch becomes an excellent algorithm for discovering the matching graph efficiently in large-scale SfM pipelines without requiring an expensive offline stage for building a Voc. tree.

While our method is effective, it has a few limitations:

1. Optimal parameter tuning. Our method has four parameters that were optimized over some datasets, and we have not found theoretically the best parameters for each datasets; and

2. Image representation to guide the sampling stage. GraphMatch uses Fisher vectors and are effective. However, there may be other methods more suitable for this task (*e.g.*, PAIGE [37]).

Nevertheless, GraphMatch is effective for efficient image matching in structure-from-motion pipelines.

Acknowledgments. This work was supported in part by NSF grants IIS-1321168, IIS-1342931, IIS-1619376 and IIS-1657179. The authors would like to thank Atieh Taheri, who worked on an early version of the algorithm and conducted some preliminary experiments.

References

[1] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. Building Rome in a day. *Communications of the ACM*, 54(10):105–112, 2011. 1, 2

[2] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *Proc. of the European conference on computer vision*, 2010. 1

[3] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building Rome in a day. In *2009 IEEE 12th international conference on computer vision*, pages 72–79. IEEE, 2009. 1, 7, 8

[4] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3):24:1–24:11, July 2009. 2, 4, 8

[5] C. Barnes, E. Shechtman, D. B. Goldman, and A. Finkelstein. The generalized PatchMatch correspondence algorithm. In *Proceedings of the 11th European Conference on Computer Vision Conference on Computer Vision: Part III*, ECCV'10, pages 29–43, Berlin, Heidelberg, 2010. Springer-Verlag. 4, 8

[6] M. Bujnak, Z. Kukelova, and T. Pajdla. 3d reconstruction from image collections with a single known focal length. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1803–1810. IEEE, 2009. 1

[7] A. Chatterjee and V. Madhav Govindu. Efficient and robust large-scale rotation averaging. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 521–528, 2013. 1

[8] J. Cheng, C. Leng, J. Wu, H. Cui, H. Lu, et al. Fast and accurate image matching with cascade hashing for 3d reconstruction. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2014. 1

[9] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007. 1

[10] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, Oct 2007. 2, 4

[11] D. J. Crandall, A. Owens, N. Snavely, and D. P. Huttenlocher. Sfm with mrfs: Discrete-continuous optimization for large-scale structure from motion. *IEEE transactions on pattern analysis and machine intelligence*, 35(12):2841–2853, 2013. 1

[12] S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, and P. Sen. Image Melding: Combining inconsistent images using patch-based synthesis. *ACM Trans. Graph.*, 31(4):82:1–82:10, July 2012. 4

[13] V. Fragoso, P. Sen, S. Rodriguez, and M. Turk. EVSAC: Accelerating Hypotheses Generation by Modeling Matching Scores with Extreme Value Theory. In *Proc. of the IEEE International Conference on Computer Vision*, 2013. 1

[14] V. Fragoso, C. Sweeney, P. Sen, and M. Turk. ANSAC: Adaptive Non-Minimal Sample and Consensus. In *Proc. of the British Machine Vision Conference*, 2017. 1

[15] J.-M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, et al. Building Rome on a cloudless day. In *European Conference on Computer Vision*, pages 368–381. Springer, 2010. 1, 6

[16] V. M. Govindu. Robustness in motion averaging. In *Asian Conference on Computer Vision*, pages 457–466. Springer, 2006. 1

[17] R. Hartley, K. Aftab, and J. Trumpf. L1 rotation averaging using the weiszfeld algorithm. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3041–3048. IEEE, 2011. 1

[18] J. Heinly, J. L. Schonberger, E. Dunn, and J.-M. Frahm. Reconstructing the world* in six days*(as captured by the yahoo 100 million image dataset). In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2015. 1

[19] T. K. Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998. 3

[20] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311. IEEE, 2010. 2, 4

[21] N. Jiang, Z. Cui, and P. Tan. A global linear method for camera pose registration. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 481–488, 2013. 1

[22] K. Kanatani and C. Matsunaga. Closed-form expression for focal lengths from the fundamental matrix. In *Proc. 4th Asian Conf. Comput. Vision*, volume 1, pages 128–133. Citeseer, 2000. 1

[23] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. 1

[24] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999. 3

[25] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 1, 5

[26] D. Martinec and T. Pajdla. Robust rotation and translation estimation in multiview reconstruction. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007. 1

[27] P. Moulon, P. Monasse, and R. Marlet. Global fusion of relative motions for robust, accurate and scalable structure from motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3248–3255, 2013. 1

[28] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014. 1

[29] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. 5

[30] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2161–2168. IEEE, 2006. 1, 3

[31] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International journal of computer vision*, 42(3):145–175, 2001. 1, 2

[32] O. Ozyesil and A. Singer. Robust camera location estimation by convex programming. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 2674–2683, 2015. 1

[33] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007. 2, 4, 5, 8

[34] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed fisher vectors. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3384–3391. IEEE, 2010. 5

[35] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. In *European conference on computer vision*, pages 143–156. Springer Berlin Heidelberg, 2010. 2, 4, 5, 8

[36] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007. 1

[37] J. L. Schonberger, A. C. Berg, and J.-M. Frahm. Paige: pairwise image geometry encoding for improved efficiency in structure-from-motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1009–1018, 2015. 3, 4, 7, 8

[38] J. L. Schönberger and J.-M. Frahm. Structure-from-motion revisited. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 1, 7

[39] T. Shen, S. Zhu, T. Fang, R. Zhang, and L. Quan. Graph-based consistent matching for structure-from-motion. In *European Conference on Computer Vision*, pages 139–155. Springer, 2016. 3

[40] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008. 1

[41] N. Snavely, S. M. Seitz, and R. Szeliski. Skeletal graphs for efficient structure from motion. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, jun 2008. 3

[42] H. Stewénius, S. H. Gunderson, and J. Pilet. Size matters: exhaustive geometric verification for image retrieval accepted for eccv 2012. In *Computer Vision–ECCV 2012*, pages 674–687. Springer, 2012. 1

[43] C. Sweeney, T. Hollerer, and M. Turk. Theia: A fast and scalable structure-from-motion library. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 693–696. ACM, 2015. 6

[44] C. Sweeney, T. Sattler, T. Hollerer, M. Turk, and M. Pollefeys. Optimizing the viewing graph for structure-from-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 801–809, 2015. 1, 7

[45] A. Vedaldi and B. Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1469–1472. ACM, 2010. 5

[46] K. Wilson and N. Snavely. Robust global translations with 1DSfM. In *Proc. of the European Conference on Computer Vision*, 2014. 1, 7

[47] C. Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013. 1, 3, 7