Pacific Graphics 2014
J. Keyser, Y. J. Kim, and P. Wonka
(Guest Editors)

Volume 33 (2014), Number 7

Fast Feature-Oriented Visual Connection for Large Image Collections

Qingan Yan, Zhan Xu, Chunxia Xiao

Computer School, Wuhan University, China

Abstract

Deriving the visual connectivity across large image collections is a computationally expensive task. Different from current image-oriented match graph construction methods which build on pairwise image matching, we present a novel and scalable feature-oriented image matching algorithm for large collections. Our method improves the match graph construction procedure in three ways. First, instead of building trees repeatedly, we put the feature points of the input image collection into a single kd-tree and select the leaves as our anchor points. Then we construct an anchor graph from which each feature can intelligently find a small portion of related candidates to match. Finally, we design a new form of adjacency matrix for fast feature similarity measuring, and return all the matches in different photos across the whole dataset directly. Experiments show that our feature-oriented correspondence algorithm can explore visual connectivity between images with significant improvement in speed.

Categories and Subject Descriptors (according to ACM CCS): I.3.m [Computer Graphics]: computational photography—image-based modeling

1. Introduction

With the explosive growth in the availability of image data online, image collections have become an important medium in computer graphics and vision. These unorganized views show a lot of famous sites and are rich of geometric information. Over the years, we have witnessed the emergence of a number of novel Web-scale graphic systems that have been developed based on large-scale image collections, for instance, [SSS06] [TSH*11]. A central part for these applications is to construct a match graph exploring the connectivity relationships between overlapping images and indicating feature correspondences across the whole photo collection. However, constructing such match graph to an image collection usually involves massively exhaustive pairwise feature comparisons [MA10] [SBBF12] [ML14]. The computational cost for this process grows exponentially with the increasing of image number. Therefore, it is desirable to execute pairwise matching operations as minimal as possible, at the meantime efficiently obtain as complete a match graph as possible.

Internet photo collections usually represent very non-uniform samplings of viewpoint. Most image pairs exhibit sparse visual correspondence. It is often unnecessary to check all the image pairs, as a subset is sufficient for most applications. Many recent state-of-the-art image matching systems, for example [ASS*09] [HGO*10] [FFGG*10] [GBQG09], have been built on top of this idea. For large datasets, they make use of the image retrieval techniques such as bag-of-words models and inverted files to extract a small portion of similar image pairs on which they perform detailed feature matching. This kind of methods drastically reduces the overall computation, and produces a well approximated match graph. However, bag-of-words models require an additional learning phase and rely on query expansion stage to dense the final match graph. Both of them introduce overhead at the image retrieval stage which increases the total running time, especially when the image sets are not sufficiently large.

On the other hand, matching irrelevant features between overlapping images is also undesirable. For instance, Fig.1 shows three visual overlapping images from **St. Pauls** dataset. It is easy to see that, although there are some good feature matches linking these similar photographs, most features between them are unmatched. Hence querying nearest neighbors for such irrelevant features of each overlapping image pair is also a waste of time.

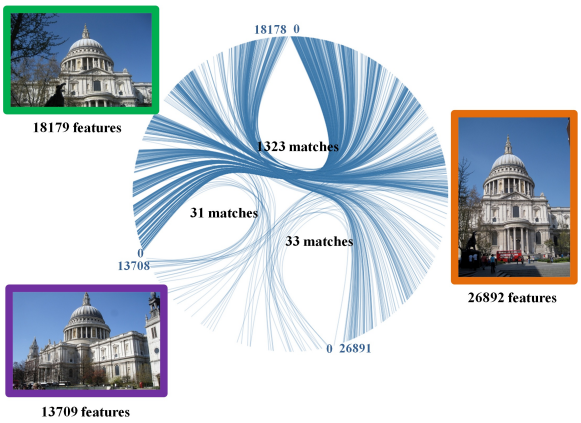


Figure 1: *Three photos from the **St. Pauls** dataset captured in different viewpoints. Although each image pair exhibits spatial overlap with some common visual features (shown as links with blue color between them), many features (white regions on the circle) are unmatched. Matching these features, even if they are in similar image pairs, is also unnecessary. Thus our motivation is, for each feature, to intelligently predict a small portion of feature candidates to match, rather than process at the level of images.*

Different from traditional *image-oriented* pairwise image matching systems, we approach this through a new *feature-oriented* way that treats the feature, rather than a whole image, as basic processing unit (a vertex in the match graph). Motivating our method is the observation that interactions between feature pairs further than a certain distance apart can be safely ignored, as the probability to be matches becomes very low. Intuitively, if each feature could successfully in advance find its all potential candidates from the given image collection, then we just need to verify a very small portion of vertices to form a dense match graph efficiently. Based on the match graph, a number of interesting works can proceed on [CS13] [TKKT12].

Thus in this paper, we propose a fast *feature-oriented* dense match graph construction algorithm for large photo collections. Our method mainly improves the construction procedure in three ways. Firstly, instead of repeatedly building kd-tree for every image pair, we directly put the feature points of the input photo collection into a single kd-tree, and each feature can be linearly represented by a constant number of nearby leaves (which act as anchor points in our paper). Secondly, image matching can be significantly accelerated by intelligently comparing each feature (not image) to a small portion of its potentially related candidates. It works by modeling the feature relationships using an anchor graph. This makes features only need to execute a few comparison operations with corresponding anchor points to select related candidates. Thirdly, based on the constructed anchor graph, we design a new form of adjacency matrix for fast feature similarity measuring, and return each feature's all visual correspondences across the whole dataset directly.

Our algorithm is non-iterative and learning-free. We demonstrate the effectiveness of our algorithm on several image collections with the sizes range from tens to thousands. Experiments show that our method can efficiently construct a dense match graph from the image collection within a significant short period of time, even performed on a single CPU core.

This paper is organized as follows. In section 2, we give the related work, and section 3 is the main part of our paper, where we describe our proposed *feature-oriented* matching algorithm and give a detailed analysis of its complexity. In section 4, we show the performance of our algorithm on efficiency and accuracy. Next we show a 3D reconstruction application in section 5 and conclude this paper in section 6.

2. Related Work

The problem of image collections matching has recently gained great attention in both computer vision and computer graphic areas. It is a central element for image-based modeling [BNB13], image-based rendering [CDSHD13], and image editing [ZGW*14] [HSGL13]. However, the quadratic computational time for pairwise image matching significantly constrains their application range. In this section, we discuss two relevant techniques that have been proposed to facilitate image collections matching: approximate nearest neighbor models and image retrieval based methods.

2.1. Approximate nearest neighbor models

Approximate nearest neighbor (ANN) models have been widely used in feature matching. For each pair of images, [MA10] works by directly inserting the features of one image into a kd-tree and querying the tree using the features from the other image. Recently, many novel ANN methods are also designed which can be employed for fast pairwise image matching, e.g. LDAHash [SBBF12] and FLANN [ML14]. However, to handle collections with thousands to millions of images, directly using these ANN algorithm in the pairwise manner are still incompetent in reality.

In order to improve matching efficiency, it is reasonable to reduce comparison operations. [FFG09] [GFF10] employ a constant number of feature descriptors in each image and build a similarity linkage by matching each feature to its k nearest neighbors. Then, each image is matched to a small constant of photographs that have greatest match numbers. However the final match graph produced in this paper is sparse which means that there may be a portion of important image correspondences losing.

In a similar way, [Wu13] present a preemptive feature matching algorithm by exploiting the scales of invariant features. He discovers that the features with top-scale values often have more chances to be matched. This means the more

similar pairwise images are, the more top-scale features are on par with other features. So it is desirable to test only a few large-scale features to decide whether the image pair is necessary to be fully matched. This algorithm is efficient. Nevertheless, for some image collections, this method may produce poor results as it has a small chance of losing weak links due to improper thresholds or occlusion.

An interesting notion is employed in [TL09] and [NYS11] that, when detecting features from images, many features which are meaningless are extracted. Rejection of useless features can provide significant savings in ANN searching. While these may also give rise to sparse visual connectivity, as for some applications like [BNB13], the features around leaves and trees are informative descriptors.

2.2. Image retrieval based methods

Another intuitive way to speed up image matching is to directly retrieve similar candidates for each image. Image retrieval based methods leverage special classifiers or distance metrics ideally learned from relevant datasets to efficiently determine likely images in large collections.

Many recent state-of-the art image matching systems are based on Bag-of-Visual-Features (BoVF) models which are inspired from text retrieval. For instance, in order to build the scenes from city-scale image collections in a day, [ASS*09] first retrieve a constant of candidates for each images with a vocabulary tree [NS06], then dense the connectivity of the match graph by using query expansion [CPS*07], and finally match each image to their selected candidates through ANN searching on a cluster with 500 computer cores. In a similar way, [HGO*10] designs an image tourism system called *Image Webs* which divides the matching procedure into two phases. First, a standard BoVF model is used to produce a set of connected components. In the next phase, algebraic connectivity from spectral graph theory is applied to dense each connected component.

Although, BoVF models have been widely used in collection matching problems, these metrics are noisy according to different training sets. [LSG12] proposes an algorithm identifying large connected components in large image collections and introduce the idea of relevance feedback to improve decision making over time. [KTT*12] poses the process of matching as a link prediction problem and increases the match graph by iterating between the estimation of potential links and their verification. While in order to leverage this kind of matching algorithm, a good training set and cloud platform are recommended.

Unlike above approaches based on a massively parallel cluster, [RWFL11] and [FFGG*10] leverage GIST features to extract iconic images, for fast image clustering on a single commodity PC using GPUs. They also show that the geo-tag is a good tool for image linkage verification. However, they may break a connected scene into several separate components, which is undesirable in 3D reconstruction.

Current image matching methods focus on the whole image too much, and ignore the underlying structure of features, which is of great importance for boosting image matching, especially when the number of feature points is considerable. Instead of querying only one nearest neighbor for each feature at a time, we directly extract the whole matches of this feature from the photo collection. Hence in this paper, we introduce a *feature-oriented* dense match graph construction method which can significantly speed up the performance of image matching for large photo datasets, even on a single CPU core.

3. Efficient Match Graph Construction

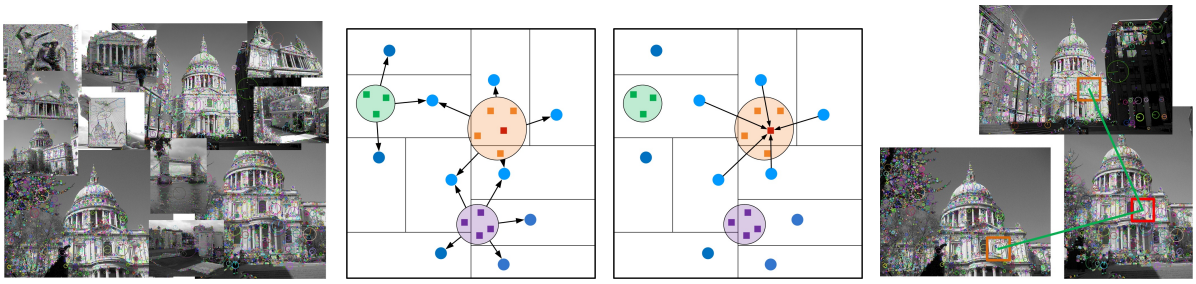
Given a set of features extracted from the input photo collection, as mentioned in the introduction, our goal of image matching is equivalent to construct a match graph from the feature set, as efficiently as we can, with edges linking corresponding feature vertices.

To formalize, a match graph is an undirected graph that can be denoted as $Z = (V, E, W)$, where $v \in V$ is a set of n vertices in which each vertex v represents a feature point rather than a single image that needs to be matched, $(v_i, v_j) \in E$ is a set of edges connecting related vertices which indicate these vertices are matches, and $W \in R^{n \times n}$ denotes the adjacency matrix with the element w_{ij} recording the corresponding weight between the point v_i and v_j. In the case of image collections matching, the weight is non-negative and is the similarity measure between that feature pair, usually in the simple form of $w_{ij} = \|v_i - v_j\|$. Thus the problem of image matching is to link V with E according to W.

3.1. Problem reformulation

As for large databases which may contain billions of features, we expect to make the construction time of W to be approximately linear corresponding to the feature number. This means it would be fine that, for each feature point $\boldsymbol{v_i}$, we could intelligently obtain a candidate set $C_i = \{\boldsymbol{v_j}\} \subset V$ across the whole set, and determine which elements in C_i are good matches. However, it is difficult to know in advance which features are potentially related, since directly using $k$nn is both inefficient and inaccurate here. In order to achieve this requirement, we construct an additional graph G by just comparing each feature to a few selected anchor points, where anchor points are a set of landmark samples sharing the same dimension with image features. Fig.2 shows a comprehensive description about our algorithm.

The object of an anchor-based model is to locally represent each data point on the manifold in the form of a linear combination of its nearby anchor points, and see the linear weights as its local coordinate coding [LHC10]. This notion



(a) Input feature set (b) Splatting (c) Slicing (d) Output feature correspondences

Figure 2: *In large photo collections, it is desirable to find a small number of candidates for each feature to match. To achieve this target, we first place the whole features into a kd-tree. This makes similar features cluster locally (quadrangles in middle figures) and be surrounded by a few anchor points (small circles in blue color). Therefore, matching is done by firstly scattering feature* ***records*** *(as described in section 3.4) into their nearby anchors (****splatting****), and then gathering at each feature from corresponding anchors, by combining identical* ***records*** *contained in these anchors, to score the related candidates around it (****slicing****), finally returning the all images that contain visual correspondences to current feature.*

can be also used in our case. Since similar features are locally adjacent which are usually bounded by their neighboring anchors, and the linear weights are a helpful indication of their similarity. Now we introduce how to use anchor graph to model our features. Given a feature set of a large image collection $V = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_n}\} \subset \mathbb{R}^d$, suppose we additionally have a data set denoted by $X = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_m}\} \subset \mathbb{R}^d$ with the size of $m (m \ll n)$, in which each $\boldsymbol{x_i}$ acts as an anchor point. The anchor graph G is a bipartite graph with nodes are the features V and anchors X. Then the idea is to approximately reconstruct each feature point in V from anchors of X as follow:

$$\boldsymbol{v_i} = \sum_{k=1}^{m} u_{ik} \boldsymbol{x_k}, \quad i = 1, \ldots, n \tag{1}$$

where $U \in \mathbb{R}^{n \times m}$ is the weight matrix of G that measures the relations between V and X with the constraints that, for each row i of U, $\sum_{k=1}^{m} u_{ik} = 1$ and the element $u_{ik} \geq 0$.

A key problem for constructing an anchor graph is how to compute the weight matrix U. As for large-scale feature sets, it is computationally expensive to optimize a convex problem for each feature $\boldsymbol{v_i}$ and impose a heavily computational overhead on building matrix U. Instead, in some other applications (e.g., image filtering [CPD07] [ABD10]), the weights can be estimated based on the distances between data points. This idea is useful, as it is reasonable to consider that one feature, in a high probability, tends to be mainly reconstructed by its neighboring anchors.

Hence similar to [XBC*11], we simply represent each $\boldsymbol{v_i}$ by its k nearby anchors with the weights computed as:

$$u_{ij} = \frac{\mathcal{H}(\boldsymbol{v_i}, \boldsymbol{x_j})}{\sum_{l \in L_{\langle i \rangle}} \mathcal{H}(\boldsymbol{v_i}, \boldsymbol{x_l})} \quad \forall j \in L_{\langle i \rangle} \tag{2}$$

where $\mathcal{H}(\boldsymbol{v_i}, \boldsymbol{x_l}) = \exp(-\|\boldsymbol{v_i} - \boldsymbol{x_l}\|^2 / 2\delta_s^2)$ is the Gaussian kernel and $L_{\langle i \rangle} \subset \{1, \ldots, m\}$ is the set saving the indexes of k nearest anchors of $\boldsymbol{v_i}$. Parameter δ_s determines the size of the local region upon which each anchor can have an impact.

Assume we have already built an anchor graph, by connecting each feature to its k nearest anchors and assigning the weights according to the Eq.2. Calculating the similarity between a potentially related feature pair $\boldsymbol{v_i}$ and $\boldsymbol{v_j}$ is thus equivalent to compute w_{ij} as follow:

$$w_{ij} = \| \sum_{l \in L_{\langle i \rangle}} u_{il} \boldsymbol{x_l} - \sum_{l' \in L_{\langle j \rangle}} u_{jl'} \boldsymbol{x_{l'}} \| \quad s.t. L_{\langle i \rangle} \cap L_{\langle j \rangle} \neq \emptyset. \tag{3}$$

While directly computing the distance may be even more inefficient, as many additional operations have to be done. Hence our problem is naturally reformulated into how to quickly build the anchor graph G, and based on this graph, how to design a new form of adjacency matrix W to efficiently measure the similarity instead of Eq.3. Next, we will discuss these two aspects respectively in more detail.

3.2. Anchor graph construction

In order to construct an anchor graph, we should know the set X of anchor points ahead. How to select a number of proper anchors from the feature set is a key problem, as it directly influences our matching accuracy. In our paper, we identify the anchors by placing the whole features into a single Gaussian kd-tree.

The procedure starts from the root cell T which represents all features in a dataset. To each cell, we compute the bounding box (the set of minimal and maximal values for each dimension of features in the cell) and choose its longest dimension, along which we split halfway a cell into two child cells adaptively. The feature set is partitioned recursively until each terminal cell (a leaf node) has a diagonal

length of its bounding box less than the threshold δ_l. Similar to [AGDL09] [XL10] [XLX*14], in each inner node of the tree we need to store six variables: the cutting dimension θ_d and corresponding cutting value θ_{val}, the bounding values θ_{min} and θ_{max} in cutting dimension, and pointers to its children θ_{le} and θ_{ri}. Leaf nodes only maintain a data point which is considered as an anchor point with the same dimension to features. Each anchor point $\boldsymbol{x_i}$ is computed by the mean of feature descriptors covered in this leaf domain. Hence the kd-tree has adaptively produced m anchors $\{\boldsymbol{x_i}\}_{i=1}^{m}$ one anchor per leaf.

For the purpose of matching, for each feature point, besides the feature descriptor, we have to preserve two additional variables: η_{im} and η_{fe} which respectively indicate the image it belongs to and its feature index in this image.

In order to build the anchor graph, we connect each data point to its k nearest anchors using a Gaussian query by spilling a set of s querying samples. In each querying, the samples travel the tree from root to leaves, and recursively split into two sets where their sizes are determined by a Gaussian distribution. The Gaussian query quickly returns at most s results from which we seek the k nearest neighbors with the weights u_{ik} assigned by Eq.2.

The advantage of constructing an anchor graph is obvious. If two data points $(\boldsymbol{v_i}, \boldsymbol{v_j})$ are correlative, they share at least one common anchor point and the candidates of $\boldsymbol{v_i}$ can be regarded as $C_i = \{\boldsymbol{v_j} \in V : L_{\langle i \rangle} \cap L_{\langle j \rangle} \neq \emptyset\}$, otherwise $L_{\langle i \rangle} \cap L_{\langle j \rangle} = \emptyset$ and there is no need to match these points.

3.3. Similarity scoring

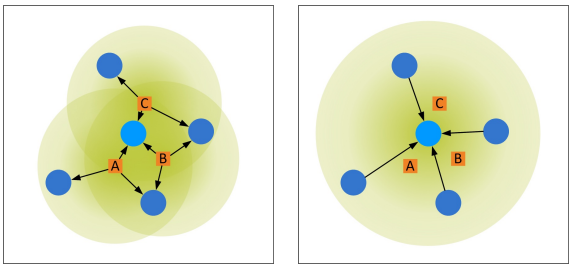
After constructing an anchor graph, the rest of computational cost for feature matching is the pairwise similarity metric measuring between features and their candidates. We present a new approach to design the adjacency matrix W and make an intuitive explanation for it.

Naturally with respect to the distribution of features, there is a useful observation that closer features usually share more anchor points in common. This gives us two important implications: For a good feature correspondence, first, the common anchors should have strong representation power to them; second, the quantity of anchors in common should also be large.

Given a feature $\boldsymbol{v_i}$ and its candidates $C_i = \{\boldsymbol{v_j}\}$, let us define in advance two vectors $\boldsymbol{b_i} = [u_{il_1}, \ldots, u_{il_k}]$ and $\boldsymbol{f_j} = [u_{jl_1}, \ldots, u_{jl_k}]$, where $\forall l_k \in L_{\langle i \rangle}$ is the anchor indexes of $\boldsymbol{v_i}$, $u_{il} \in U$. Then we compute the similarity score of a feature pair $(\boldsymbol{v_i}, \boldsymbol{v_j})$ as:

$$w_{ij} = (\mathbf{1} \cdot \boldsymbol{f_j}^T)^{\alpha} \frac{N}{k} \quad \forall j \in C_i. \tag{4}$$

The first term $(\mathbf{1} \cdot \boldsymbol{f_j}^T)^{\alpha}$ of Eq.4 indicates the representation power of common anchors, with a factor $\alpha \in [0,1]$ controlling the effect of this term. We have already known that $\mathbf{1} \cdot \boldsymbol{b_i}^T = 1$, if $(\boldsymbol{v_i}, \boldsymbol{v_j})$ is adjacent, it is reasonable to expect $\mathbf{1} \cdot \boldsymbol{f_j}^T$ also has a high value close to 1. While directly leveraging this term is noisy, thus we import an additional term N/k, where the variable $N = |L_{\langle i \rangle} \cap L_{\langle j \rangle}|$ is the amount of shared anchor indexes. If $\alpha = 1$, the score is collectively determined by these two terms, otherwise the second term plays a more important role. This formula favors those candidates with the high weight and large number of anchors in common to current feature $\boldsymbol{v_i}$.



(a) Anchor graph (b) Blurring

Figure 3: *(a) A anchor graph with three nearby features bounded by different anchor points. (b) The **blurring** step which gathers **records** at each anchor from nearby ones.*

We now obtain a score that captures the intuition about what makes a pair of features matched or not. This measure naturally preserves two-fold properties of W. First, the relationship between two data is always nonnegative. If they are related, the value $0 < w \leq 1$, otherwise $w = 0$. Second, the highly sparse matrix U also makes W sparse. This is particularly meaningful, since each feature only needs to be compared to a small number of related candidates subject to $L_{\langle i \rangle} \cap L_{\langle j \rangle} \neq \emptyset$.

3.4. Algorithm for feature-oriented correspondence

Now we describe our full pipeline driven by the above analysis. Our algorithm begins by running a standard feature detection (SIFT [Low04] is used in our paper) to generate a set of feature points. Since kd-tree performs poorly at dimension above 50, we reduce the feature dimension to $d = 24$. After that, we build a Gaussian kd-tree and regard the produced leaves as anchor points. Then each feature scatters its *record* which contains the η_{im}, η_{fe} and weight u into k nearest anchors to construct an anchor graph (*splatting*).

However, in some cases, an additional issue may arise. Looking more closely at a simple anchor graph as shown in Fig.3(a). Suppose there are three nearby features denoted by A, B and C. While after *splatting*, they are represented by different anchors, where point A and B share two common anchors, as do B and C, but A and C have only one. In this situation, the similarity score between feature pair

Table 1: *Performance statistics of our algorithm tested on four different photo collections using a single CPU core.*

Dataset	#images	#features	#anchors	Time (hr)		
				splatting	*blurring*	*slicing*
Louvre	5,494	60,130,581	38,506,745	11	7	1.5
St. Peter's	1,468	10,946,425	5,728,071	1.2	0.7	0.2
Colosseum	1,164	26,429,139	18,064,027	3.6	2.7	0.8
NotreDame	841	8,765,236	5,048,493	0.9	0.6	0.2

(A,C) estimated from Eq.4 would be very low, even though these points are mutually nearby. This is because the distribution of anchors is uneven and parameter δ_l or k is too small, which lead to redundant anchors around some feature clusters.

To remedy this problem, we introduce an additional step before scoring, where we smooth the initial anchor graph (*blurring*). As shown in Fig.3(b), it is easy to discover that the anchor points used to represent A, B and C are also close to each other. This reminds us neighboring anchors should also have an impact on current anchor. Based on the observation, we define another graph G^* which is an extended version of G. Since the anchor points are the leaves of tree T, we query the neighboring leaves for each anchor within the scope of δ_b, and compute the new weight matrix U^* by merging the corresponding *records* included in those selected anchors. If the *records* with η_{im} and η_{fe} have already existed in current anchor, there is nothing we need to do, otherwise we insert the *records* with a new weight $u^* = g \times u$, where g is a Gaussian weight decided by δ_b as well. The merging operations are efficient as the *records* contained in each anchors are ordered by η_{im} and η_{fe}.

By doing so, nearby anchors could propagate their *records* to each others. However, this step is optional. As in most applications, the *splatting – slicing* scheme provides sufficient accuracy as discussed in section 4.2.

After *blurring*, we gather *records* at each feature point from its k nearest anchors by accumulating the weight u with identical η_{im} and η_{fe} according to Eq.4 (*slicing*). For images labeled with different η_{im} to current matching feature, if the η_{fe} with largest u and the one with second largest u' satisfy $u - u' > \delta_m$, where δ_m is a threshold, then we regard the feature η_{fe} contained in image η_{im} is a good correspondence to current matching feature.

When the image collection is very large, we find that saving the whole feature set and corresponding anchor graph simultaneously in RAM is memory-consuming. In our experiments, for this situation, we commence *splatting* with dividing the image set into multiple parts, and operate single part at each time in following steps, then save the middle outputs to files in an out-of-core manner with a relatively small amount of the I/O budget.

3.5. Complexity analysis

In this subsection, we make a complexity analysis for our algorithm in detail, including the computation cost and storage cost. As feature points are our basic unit, thus the algorithm complexity is based on the size of feature points instead of images.

Given an image collection with n d-dimensional feature points, we first intelligently select m anchor points in a kd-tree. At each level of the tree, we need to process $O(n)$ nodes and do $O(d)$ comparisons per node. Since a kd-tree is approximately balanced, it is reasonable to expect a depth about $O(\log m)$. Therefore the anchor selection requires $O(nd \log m)$ time. Then we do a Gaussian query for the n input feature points to respectively scatter its *records* into k nearest leaf nodes. A Gaussian query takes a runtime about $O(s(\log m + d))$, and thus the *splatting* step spends $O(ns(\log m + d) + k^2 ns)$ time. Here, we have to store the whole feature sets in memory which cost $O(dn)$ storage.

In the *blurring* step, we blur each leaf node from its neighboring anchor points involving a Gaussian query and some merging operations. The merging process has a runtime about $O(kn/m)$. So the computational cost of this step can be bounded by $O(ms(\log m + d) + kn)$. Finally we assemble the *records* for each feature from corresponding anchor points to obtain matching results. This *slicing* step only requires I/O and merging operations with a time complexity of $O(kn/m)$. Generally processing these two steps theoretically requires us to store an anchor graph whose memory cost is at the level of $O(kn)$.

Consequently, as analysed above, the total computation complexity of our algorithm is $O(\log m(nd + ns + ms) + n(sd + k^2 s + k + k/m) + msd)$. While we have known that $m < n, k = 5, d = 128$ for SIFT descriptor, and s is a constant used for Gaussian querying which usually $s < 256$. This makes our time complexity be mainly decided by the first term, and results in a simplified expression of time complexity in $O(n \log n)$. As we expect, this bound shows an approximately linear relationship corresponding to the sizes of feature sets.

With regard to the storage complexity, the most memory-consuming step of our algorithm is reading all features for

Table 2: *A statistics of accuracy performance of our algorithm.*

Dataset	MA10	Our method								
		$\delta_l = 0.6$			$\delta_l = 2.0$			without blurring		
	M_G	M_T	M_I	$Prec$	M_T	M_I	$Prec$	M_T	M_I	$Prec$
Louvre	2,475	2,423	2,289	0.94	2,708	1,758	0.65	2,342	2,117	0.90
NotreDame	1,534	1,511	1,426	0.94	1,966	1,173	0.60	1,469	1,301	0.89

constructing a Gaussian kd-tree which requires $O(dn)$ storage; since in following steps, neither the whole feature set nor anchor graph is fully required in RAM. This results in a storage complexity of $O(dn)$.

4. Experiments

In this section, we evaluate the efficiency and accuracy of our algorithm on several image collections collected from Flickr. Their sizes are varied from hundreds to thousands and some are summarized in Table 1. Each collection corresponds to a popular landmark with a large disparity in the number of images that capture different parts of a scene. In addition, these datasets also contain some noisy images which have no meaning to the landmark structures, such as a close-up of baby faces or cute dogs. The timings recorded in this paper exclude the cost for SIFT extraction.

4.1. Efficiency Evaluation

Here we would like to answer how efficient our algorithm is in processing image collections matching. There are six parameters used in our algorithm. We find the values $\alpha = 0.7, k = 5, \delta_s = 0.6, \delta_l = 0.6, \delta_b = 0.4$ and $\delta_m = 0.3$, which achieve good performance in our experiments.

We first investigate the efficiency performance of our algorithm tested on single processing core. The timings are obtained on a normal PC with quad-core Xeon E3 3.3 GHz processor and 32GB RAM. Table 1 summarizes the typical running time of each step of our algorithm performed on four different scale datasets. Given a feature set large as **Louvre**, our approach constructs its dense match graph just in one day including the overhead of I/O and PCA processes.

The threshold δ_l directly determines the number of anchor points produced in our paper. Large value of δ_l leads to a small amount of anchor points. Although decreasing anchors in size can facilitate the querying operations in *splatting*, it also causes each anchor linking more features, and consequently increases the merging cost invested in *blurring* and *slicing*. Moreover, large δ_l reduces our accuracy of image matching as well. Since there would be more than one cluster locally bounded by the same anchor points, and the features in these clusters would obtain too ambiguous scores to distinguish from each others. On the other hand, if δ_l is too small, most features would individually become leaf nodes. This makes our algorithm be reduced to normal k-nn manner.

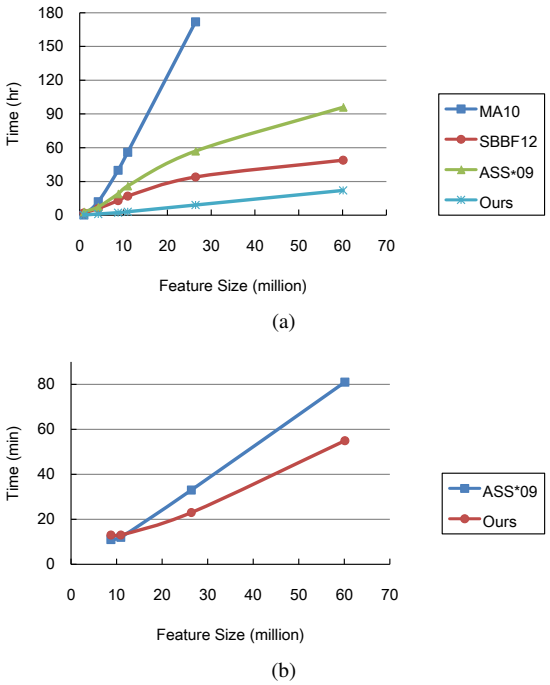


Figure 4: *(a)Computational time comparisons according to different feature sizes on single core. (b)Performance comparisons on the cluster.*

From experiments we find $\delta_l = 0.6$ (when $d = 24$) is a proper choice in our test sets, which also tells us if the distance of two features is much larger than 0.6, then the probability to be matches becomes very low.

Fig.4(a) shows some comparisons between our approach and recent state-of-the-art methods ([MA10], [ASS*09], and [SBBF12]). The comparisons are examined without any distributed optimization or GPU technique. As for [ASS*09], we train a vocabulary tree with the branching factor 10 from **St. Peter's**. For [SBBF12], we first use the vocabulary tree to find similar images, and then adopt its LAD-Hash model for fast ANN searching. Datasets used here include two additional image collections – **St. Pauls** and **Pantheon** respectively with image sizes 121 and 356. As shown in Fig.4(a), our *feature-oriented* matching scheme achieves significantly speedup compared to these three methods, and

performs well even if the collection is small-scale. It is also visible that the total computational time of our algorithm is more approximately linear to the sizes of feature sets.

To see the performance in parallel processing, we also implement a distributed version of [ASS*09] on a cluster with 6 nodes. Each node in the cluster has 2 12-core Xeon processors running at 2.2 GHz with 128 GB memory. Our algorithm can also be parallelized. A worker node first builds the Gaussian kd-tree and writes the structure into common storage device, then manager node averagely issues a set of images to worker nodes which separately execute our *splatting* and *blurring* stages. Before *slicing* a worker need to combine the separate anchor graphs produced by different workers, after that the *slicing* goes on. The tests are carried out on the four datasets summarized in Table 1. As shown in Fig.4(b), our method also exhibits better outperformance than [ASS*09] on cloud. These experiments demonstrate that our method is very efficient at finding feature correspondences from image collections.

4.2. Accuracy Evaluation

Evaluating the accuracy of feature matching is a difficult task, since there is no benchmark indicating which feature pairs are correct correspondences. Noisy is still remaining, even if carried out through exhaustive matching. In order to evaluate the accuracy of our algorithm, we define a simple form of matching precision that $Prec = M_I/M_T$. Denominator M_T is the size of set S_T (the feature pairs produced in our proposed algorithm from a given image set). $M_I = |S_T \cap S_G|$, where S_G represents the ground truth set generated by exhaustive matching [MA10] and $M_G = |S_G|$. However, in large photo collections, there usually contain millions of feature correspondences which are intractable to perform such accuracy estimation. Hence, instead of evaluating a whole dataset, we investigate the matching results of three overlapping images randomly chosen from **NotreDame** and **Louvre**. Moreover, to understand how much *blurring* improves the accuracy, we also make a comparison to the implementation without this step.

Table 2 summarizes the detailed accuracy statistics results of our approach following different schemes. As analyzed above, the selection of parameter δ_l is very important to the effectiveness of our algorithm, as larger δ_l may more easily lead to ambiguous similarity scores. When δ_l is close to 0.6 (when $d = 24$), our algorithm performs well in these image collections and produces highly precise feature correspondences similar to [MA10]. Additionally it is also obvious to see that, the *blurring* step brings at least 5% accuracy improvement. For those applications that matching accuracy is a primary requirement , the *blurring* is recommended. While for the datasets used in our experiments, the algorithm is also competent to produce satisfactory results without *blurring*.

In 3D reconstruction, image matching is a central element, as each feature linkage across different photos may potentially correspond to a spatial point. Hence, the number of point clouds generated through Structure-from-Motion (SfM) [SSS06], to some extent, represents the accuracy of visual connection as well. Such reconstructed points statistics of three matching methods are summarized in Table 3 for comparison. It is visible that, with regard to different photo collections, our algorithm could always maintain at an approximate level with [MA10], considering the amount of 3D point clouds produced by SfM, while requires a shorter period of computational time. We also show some 3D reconstruction results in section 5 in detail.

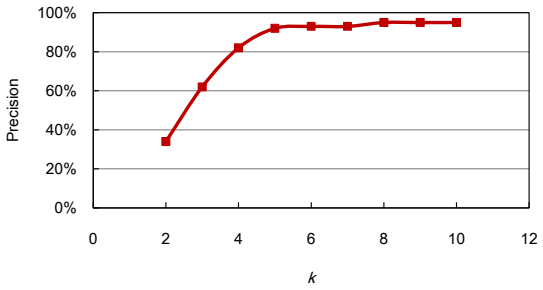


Figure 5: *Matching precision versus different values of parameter* ***k****.*

Fig.5 shows the performance of our algorithm at different values of k. Note that the precision is not sensitive to the selection of k when $k > 5$. With small k, we can guarantee the matrix U highly sparse, which could decrease our computation and memory consumption.

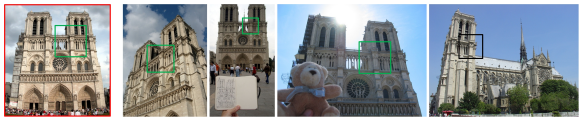
5. Application

In this section, we show 3D reconstruction results from large photo collections (**NotreDame** and **Louvre**) accelerated by our *feature-oriented* matching method. We begin with performing detailed SIFT detection on each image of a given set. Then this application requires a graph structure to be made to indicate feature correspondences between image pairs. Our algorithm provides this structure, for each feature, by directly finding all its matches from the whole set in an efficient way. In theory, a feature and its all correspondences form a *track* which reflects the same 3D point mapped into multiple 2D images. However, similar to other image matching method, the noise correspondences may also exist in our method, where a geometrically consistent verification is required for every overlapping image pair. Fig.6 exhibits two feature correspondence results across multiple images which are selected through our pipeline.

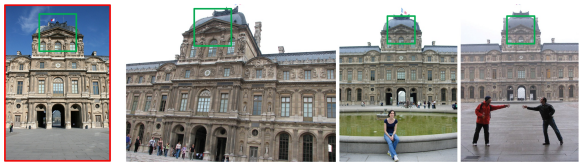
Given the *tracks*, we first identify a sparse skeleton of the whole scene related to the one in [SSS08], and reconstruct it with an incremental bundle adjustment algorithm [SSS06].

Table 3: *The comparison of reconstruction performance of three matching algorithms.*

Datasets	MA10		ASS*09		Ours	
	#Feature Matches	#Point Clouds	#Feature Matches	#Point Clouds	#Feature Matches	#Point Clouds
St. Peter's	12.7M	249,112	12.5M	**245,065**	12.5M	238,784
Colosseum	13.2M	321,731	12.7M	**311,121**	12.8M	307,691
NotreDame	7.5M	231,907	6.7M	210,267	7.2M	**218,596**



(a) NotreDame



(b) Louvre

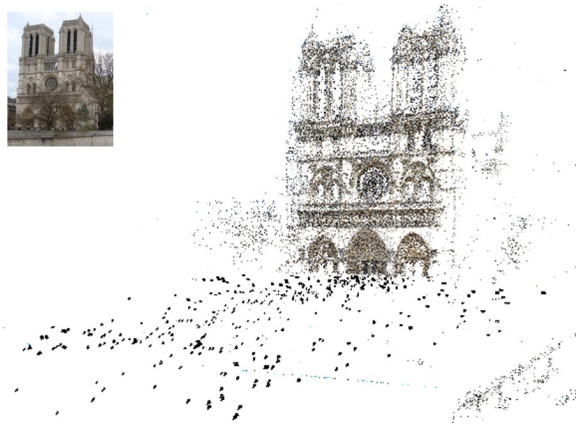
Figure 6: *Two feature correspondence results (the small squares in these pictures) in **NotreDame** and **Louvre** datasets. The black square is a noisy linkage which would be filtered in the geometrically consistent verification.*

Then the remaining images are hung onto this reconstructed skeleton. Although our image matching algorithm may bring in a few erroneous links compared to ground truth using [MA10], it is sufficient to obtain an approximate 3D model with more than ten times faster. Fig.7 shows the resulting point clouds of two landmark scenes which are respectively reconstructed from **NotreDame** and **Louvre** based on our match graph construction algorithm.

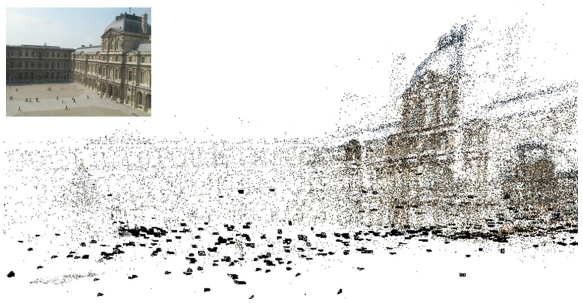
6. Conclusions and Future Work

In this paper, we have presented an efficient match graph construction algorithm for large image collections. We directly put the feature points into a single kd-tree and intelligently compare each feature to a small number of related candidates drawing support from an anchor graph. We also have shown that this method can significantly speed up the performance of image matching in different photo collections by up to one to two orders of magnitude, with little loss in accuracy.

The method we have described does have limitations. Although the proposed image matching avenue is very effective and efficient, this model is not very memory-efficient and completely sound in theory. Thus it might be fruitful to incorporate more sophisticated ideas from lattice and manifold to design more robust approaches. In addition, our method is also unable to distinguish ambiguous correspondences inferred from repeated structures. In future work, it would be interesting to extend our model to disambiguate duplicate scenes.



(a) NotreDame



(b) Louvre

Figure 7: *3D reconstruction results (**NotreDame** and **Louvre**) accelerated through our feature-oriented matching scheme.*

Acknowledgment

We wish to thank Noah Snavely and Changchang Wu for sharing their data and codes. We would also like to thank the anonymous reviewers for their valuable suggestions. This work was partly supported by the National Basic Research Program of China (No. 2012CB725303), the NSFC (No.

41271431, No.61472288), NCET (NCET-13-0441) and the Key Grant Project of Hubei province (2013AAA020).

References

[ABD10] ADAMS A., BAEK J., DAVIS M. A.: Fast high-dimensional filtering using the permutohedral lattice. *Computer Graphics Forum 29*, 2 (2010), 753–762. 4

[AGDL09] ADAMS A., GELFAND N., DOLSON J., LEVOY M.: Gaussian kd-trees for fast high-dimensional filtering. *ACM Trans. Graph. 28*, 3 (July 2009), 21:1–21:12. 5

[ASS*09] AGARWAL S., SNAVELY N., SIMON I., SEITZ S., SZELISKI R.: Building rome in a day. In *ICCV 2009* (2009), pp. 72–79. 1, 3, 7, 8

[BNB13] BRADLEY D., NOWROUZEZAHRAI D., BEARDSLEY P.: Image-based reconstruction and synthesis of dense foliage. *ACM Trans. Graph. 32*, 4 (July 2013), 74:1–74:10. 2, 3

[CDSHD13] CHAURASIA G., DUCHENE S., SORKINE-HORNUNG O., DRETTAKIS G.: Depth synthesis and local warps for plausible image-based navigation. *ACM Trans. Graph. 32*, 3 (July 2013), 30:1–30:12. 2

[CPD07] CHEN J., PARIS S., DURAND F.: Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph. 26*, 3 (July 2007). 4

[CPS*07] CHUM O., PHILBIN J., SIVIC J., ISARD M., ZISSERMAN A.: Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV* (2007), pp. 1–8. 3

[CS13] CAO S., SNAVELY N.: Graph-based discriminative learning for location recognition. In *CVPR* (2013), pp. 700–707. 2

[FFG09] FARENZENA M., FUSIELLO A., GHERARDI R.: Structure-and-motion pipeline on a hierarchical cluster tree. In *ICCV Workshops* (2009), pp. 1489–1496. 2

[FFGG*10] FRAHM J.-M., FITE-GEORGEL P., GALLUP D., JOHNSON T., RAGURAM R., WU C., JEN Y.-H., DUNN E., CLIPP B., LAZEBNIK S., POLLEFEYS M.: Building rome on a cloudless day. In *ECCV* (2010), pp. 368–381. 1, 3

[GBQG09] GAMMETER S., BOSSARD L., QUACK T., GOOL L. V.: I know what you did last summer: object-level auto-annotation of holiday snaps. In *ICCV* (2009), pp. 614–621. 1

[GFF10] GHERARDI R., FARENZENA M., FUSIELLO A.: Improving the efficiency of hierarchical structure-and-motion. In *CVPR* (2010), pp. 1594–1600. 2

[HGO*10] HEATH K., GELFAND N., OVSJANIKOV M., AANJANEYA M., GUIBAS L.: Image webs: Computing and exploiting connectivity in image collections. In *CVPR* (2010), pp. 3432–3439. 1, 3

[HSGL13] HACOHEN Y., SHECHTMAN E., GOLDMAN D. B., LISCHINSKI D.: Optimizing color consistency in photo collections. *ACM Transactions on Graphics 32*, 4 (2013), 38. 2

[KTT*12] KIM K. I., TOMPKIN J., THEOBALD M., KAUTZ J., THEOBALT C.: Match graph construction for large image databases. In *ECCV*. 2012, pp. 272–285. 3

[LHC10] LIU W., HE J., CHANG S.-F.: Large graph construction for scalable semi-supervised learning. In *ICML* (2010), pp. 679–686. 3

[Low04] LOWE D. G.: Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2 (2004), 91–110. 5

[LSG12] LOU Y., SNAVELY N., GEHRKE J.: Matchminer: Efficient spanning structure mining in large image collections. In *ECCV* (2012). 3

[MA10] MOUNT D. M., ARYA S.: ANN: A library for approximate nearest neighbor searching. 1, 2, 7, 8, 9

[ML14] MUJA M., LOWE D.: Scalable nearest neighbour algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on PP*, 99 (2014), 1–1. 1, 2

[NS06] NISTER D., STEWENIUS H.: Scalable recognition with a vocabulary tree. In *CVPR* (2006), vol. 2, pp. 2161–2168. 3

[NYS11] NAIKAL N., YANG A., SASTRY S.: Informative feature selection for object recognition via sparse pca. In *ICCV* (2011), pp. 818–825. 3

[RWFL11] RAGURAM R., WU C., FRAHM J.-M., LAZEBNIK S.: Modeling and recognition of landmark image collections using iconic scene graphs. *International Journal of Computer Vision 95*, 3 (2011), 213–239. 3

[SBBF12] STRECHA C., BRONSTEIN A., BRONSTEIN M., FUA P.: Ldahash: Improved matching with smaller descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 34*, 1 (2012), 66–78. 1, 2, 7

[SSS06] SNAVELY N., SEITZ S. M., SZELISKI R.: Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH* (2006), pp. 835–846. 1, 8

[SSS08] SNAVELY N., SEITZ S. M., SZELISKI R.: Skeletal graphs for efficient structure from motion. In *CVPR* (2008), vol. 1, p. 2. 8

[TKKT12] TOMPKIN J., KIM K. I., KAUTZ J., THEOBALT C.: Videoscapes: Exploring sparse, unstructured video collections. *ACM Trans. Graph. 31*, 4 (July 2012), 68:1–68:12. 2

[TL09] TURCOT P., LOWE D.: Better matching with fewer features: The selection of useful features in large database recognition problems. In *ICCV Workshops* (2009), pp. 2109–2116. 3

[TSH*11] TUITE K., SNAVELY N., HSIAO D.-Y., TABING N., POPOVIC Z.: Photocity: Training experts at large-scale image acquisition through a competitive game. In *CHI* (2011), pp. 1383–1392. 1

[Wu13] WU C.: Towards linear-time incremental structure from motion. In *3D Vision, 2013 International Conference on* (2013), pp. 127–134. 2

[XBC*11] XU B., BU J., CHEN C., CAI D., HE X., LIU W., LUO J.: Efficient manifold ranking for image retrieval. In *SIGIR* (2011), pp. 525–534. 4

[XL10] XIAO C., LIU M.: Efficient mean-shift clustering using gaussian kd-tree. *Computer Graphics Forum 29*, 7 (2010), 2065–2073. 5

[XLX*14] XIAO C., LIU M., XIAO D., DONG Z., MA K.-L.: Fast closed-form matting using a hierarchical data structure. *Circuits and Systems for Video Technology, IEEE Transactions on 24*, 1 (2014), 49–62. 5

[ZGW*14] ZHANG C., GAO J., WANG O., GEORGEL P., YANG R., DAVIS J., FRAHM J.-M., POLLEFEYS M.: Personal photograph enhancement using internet photo collections. *Visualization and Computer Graphics, IEEE Transactions on 20*, 2 (2014), 262–275. 2