Fast and Accurate Image Matching with Cascade Hashing for 3D Reconstruction

Jian Cheng, Cong Leng, Jiaxiang Wu, Hainan Cui, Hanqing Lu
National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences
{jcheng,cong.leng,jiaxiang.wu,hncui,luhq}@nlpr.ia.ac.cn

Abstract

Image matching is one of the most challenging stages in 3D reconstruction, which usually occupies half of computational cost and inaccurate matching may lead to failure of reconstruction. Therefore, fast and accurate image matching is very crucial for 3D reconstruction. In this paper, we proposed a Cascade Hashing strategy to speed up the image matching. In order to accelerate the image matching, the proposed Cascade Hashing method is designed to be three-layer structure: hashing lookup, hashing remapping, and hashing ranking. Each layer adopts different measures and filtering strategies, which is demonstrated to be less sensitive to noise. Extensive experiments show that image matching can be accelerated by our approach in hundreds times than brute force matching, even achieves ten times or more than Kd-tree based matching while retaining comparable accuracy.

1. Introduction

3D reconstruction is one of the classic and challenging problems in computer vision, and finds its applications in a variety of different fields. In recent years, large scale 3D reconstruction from community photo collections (e.g. Photo Tourism [18]) has become an emerging research topic, which is attracting more and more researchers from academy and industry. However, 3D reconstruction is extremely computational expensive. For example, it may cost more than a day in a single machine to reconstruct an object with only one thousand pictures. In the Structure from Motion (SfM) model [1, 4], 3D reconstruction pipeline can be divided into several steps: feature extraction, image matching, track generation and geometric estimation, etc. Among them, image matching occupies the major computational cost, even more than half of all in some case. Moreover, inaccurate matching results might lead to failure of reconstruction. Therefore, fast and accurate image matching is crucial for 3D reconstruction.

Image matching techniques can be roughly divided into three categories: point matching, line matching and region matching. Due to its robustness to changes of illumination, affine transformation and viewpoint changes, point matching has received much attention and many effective algorithms have been proposed in the past decades [13, 2]. However, point matching is usually very time consuming. For pairwise image matching, the computational complexity of exhaustively comparing all feature points in two images is $O(N^2)$, where N is the average number of feature points in each image.

As an alternative, approximate nearest neighbor (ANN) search has been studied to replace the exhaustive matching in 3D reconstruction. One classical paradigm is the tree-based approaches which tend to store data with efficient data structures, and makes the search operation quite fast, typically with the complexity of $O(log(N))$. The representative tree based algorithms include Kd-tree, M-tree, and ball tree [15], among which the Kd-tree may be the most widely used strategy in literature of 3D reconstruction [1, 4]. However, the high dimensionality will significantly corrupt the efficiency of tree-based methods, and make them even perform worse than the naive methods (e.g. linear scan).

Because of the efficiency in both storage and speed, hashing based ANN search methods have attracted much attention in the past years [9, 7]. Hashing converts all feature representation of images into binary codes and then conducts a bitwise XOR operation in very fast speed. Hashing strategy is first introduced into matching process for 3D reconstruction in LDAHash [19]. LDAHash performs Linear Discriminant Analysis (LDA) on the descriptors before binarization. However, LDAHash use an exhaustive linear search to find the matching points, which reduces significantly its efficiency. Besides, LDAhash is a supervised and data-dependent approach which needs additional human labeling in training stage.

Inspired by LDAHash, in this paper, we propose a Cascade Hashing structure to speed up image matching for 3D reconstruction, named CasHash. The advantages of the proposed approach are two-fold. On the one hand, the proposed Cascade Hashing structure contains three layers which map the image representation into binary codes from coarse to

fine, resulting in significant speedup of image matching. On the other hand, each layer of the Cascade Hashing adopts different measure and filtering strategy, which is demonstrated to be less sensitive to noise of feature points.

Compared with the closest LDAHash, our approach has three major advantages. First, our cascade hash structure is coarse-to-fine and faster in accelerating image matching. Second, our approach can be considered as a multi-stage filter which is resistent to noise point pairs. Third, we utilize an unsupervised and data-independent method, i.e. Locality Sensitive Hashing (LSH) [3], to generate hash functions. Therefore, our approach is data-independent and training-free. Extensive experiments demonstrate that image matching can be accelerated by our approach in ten times or more than Kd-tree based method while retaining comparable accuracy.

2. The Proposed Approach

In this paper, we propose a Cascade Hashing structure, named CasHash, to speed up image matching for 3D reconstruction. In our method, a simple hashing algorithm, Locality Sensitive Hashing (LSH), is adopted to generate binary code (refer to Fig.1). Consequently, we will give a brief introduction to LSH algorithm.

2.1. Locality Sensitive Hashing

Let $X = \{x_1, x_2, \cdots, x_n\}$ be a set of data points, where $x_i \in \mathbb{R}^d$. Given a query vector q, we are interested in finding the most similar items in X to the query. LSH is perhaps the most well known hashing based ANN search scheme, which relies on the existence of locality sensitive hashing functions. Assume $\mathcal{H}$ be a family of hashing functions mapping $\mathbb{R}^d$ to Hamming space $\mathbb{B}$. For any two points x and y, it chooses a function h from $\mathcal{H}$ uniformly at random and is confined to the probability $h(x) = h(y)$. The function family $\mathcal{H}$ is locality sensitive if it satisfies the following conditions:

Definition 1: A family $\mathcal{H}$ of functions from $\mathbb{R}^d$ to $\mathbb{B}$ is called (R, cR, P_1, P_2)-sensitive for $D(\cdot,\cdot)$ if for any $x, y \in \mathbb{R}^d$

- $Pr_{h\in\mathcal{H}}(h(x) = h(y)) \geq P_1$, if $D(x, y) \leq R$
- $Pr_{h\in\mathcal{H}}(h(x) = h(y)) \leq P_2$, if $D(x, y) \geq cR$

where $D(\cdot,\cdot)$ is a distance function in the original space $\mathbb{R}^d$. Obviously, a family $\mathcal{H}$ is valid only when $c > 1$, and $P_1 > P_2$. Given valid LSH functions, Gionis *et al.* proved that the query time for retrieving $(1+\epsilon)$-near neighbors is bounded by $O(n^{\frac{1}{1+\epsilon}})$ for Hamming distance [7].

Charikar defined a hashing function $h \in \mathcal{H}$ for the widely used inner product similarity [3]. More specifically, choosing a random vector r from a d-dimensional Gaussian

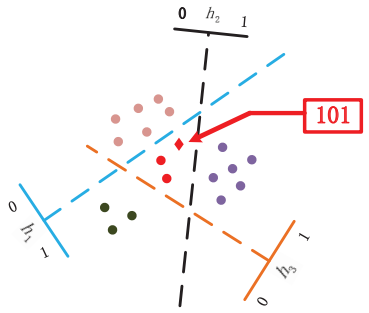


Figure 1. Locality Sensitive Hashing

distribution $\mathcal{N}(0, I)$, with this hyperplane r, hashing function h is defined as:

$$h_r(q) = \begin{cases} 1, & \text{if } r \cdot q > 0 \\ 0, & \text{if } r \cdot q < 0 \end{cases} \tag{1}$$

In [8], Goemans *et al.* proved that for any data point x and y ,

$$Pr(h_r(x) = h_r(y)) = 1 - \frac{\theta(x, y)}{\pi} \tag{2}$$

where $\theta(x, y) = cos^{-1}(\frac{x^T y}{\|x\|\|y\|})$ is the angle between x and y.

Defining $D(x, y) = \frac{\theta(x,y)}{\pi}$, with Eq.(2) it is easy to find that if $D(x, y) \leq R$, then $Pr(h(x) = h(y)) \geq 1 - R$ and if $D(x, y) \geq cR$, then $Pr(h(x) = h(y)) \leq 1 - cR$. Therefore, if $P_1 = 1 - R$ and $P_2 = 1 - cR$, as long as the approximation factor c is greater than 1, we have $P_1 > P_2$. These satisfy the property in Definition 1.

Since the gap between the probability P_1 and P_2 could be quite small, in real cases, an "amplification" process is needed by concatenating the output of multiple different hashing functions. Because of its widely use, LSH algorithm is also extended to p-norm distance [5], Mahalanobis distance [10] and kernel similarity [11].

2.2. Cascade Hashing

In order to speed up the image matching as fast as possible, the proposed cascade hashing structure is designed to consist of three layers: hashing lookup (Section 2.2.1), hashing remapping (Section 2.2.2), and hashing ranking (Section 2.2.3). The flowchart of our method is shown in Fig.2. The designed 3-layer hashing maps the feature representation of images into binary codes from coarse to fine, resulting in faster image matching. Moreover, each layer of the cascade hashing adopts different measure and filtering strategy, which may filter out some feature point noise in cross-validation manner. In this sense, the proposed CasHash is less sensitive to noise in 3D reconstruction.

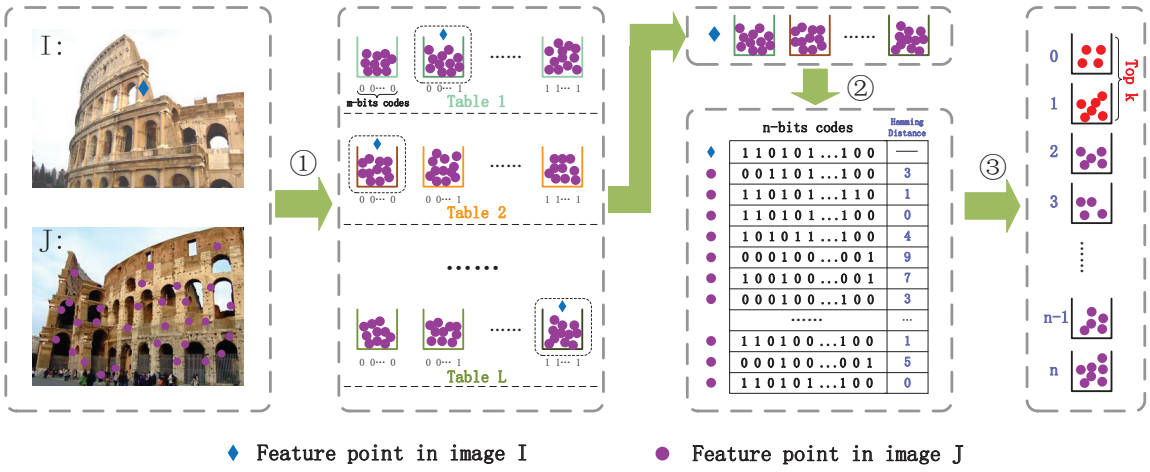


Figure 2. The flowchart of our proposed Cascade Hashing approach. For a feature point in Image I, three steps are involved to find its matching points in Image J. First, a multi-table hashing lookup with short codes is employed to conduct a coarse search. Second, the returned candidates will be remapped into higher dimensional Hamming space and the Hamming distances to the query are calculated. Final, we build hashing table with Hamming distance as keys in order to find the most exact (top k) candidate sets.

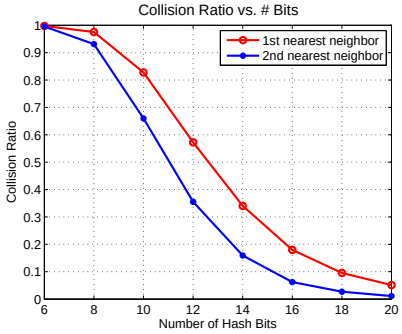


Figure 3. The ratio that the two nearest neighbors collides in the same bucket with the query while the number of bits changes.

2.2.1 Hashing Lookup with multiple tables

In the first step, we apply hashing lookup with short codes to do a coarse search. Particularly, all the feature points in all images are embedded into m bits binary codes with LSH. For feature point p in image I, in order to find its matching point in image J, a lookup table with a set of buckets is constructed using the m bits binary codes. All the points that fall into the same bucket of point p in image J will be returned. This above procedure is called as hashing lookup.

Hashing lookup emphasizes on the practical search speed since its complexity is constant time. However, using a single hashing table with long bits, hashing lookup often fails because the Hamming space becomes increasingly sparse and very few samples fall in the same bucket. In this paper, we adopt a multi-table strategy to address this problem. In particular, we generate L functions $g_l(q) = (h_{1,l}(q), h_{2,l}(q), \cdots, h_{m,l}(q))$, $l = 1, 2, \cdots, L$, where $h_{s,l}(1 \leq s \leq m, 1 \leq l \leq L)$ are generated independently and uniformly at random from $\mathcal{H}$. In summary, the data structure is constructed with L hashing tables which have m bits, and then each point p is assigned to a bucket $g_l(p)$, for $l = 1, \cdots, L$.

On the one hand, larger value of m leads to a larger gap between the probabilities of collision for the similar points (P_1) and dissimilar points (P_2). The benefit of this amplification is that the hashing functions are more selective. On the other hand, if m is larger then P_1 is smaller, which means that L have to be large to ensure that the real neighbors collide with query point at least once. Particularly, the probability will be $1 - (1 - P_1)^L$. In practice, we can tade off parameter m and L for different applications.

In feature matching of 3D reconstruction, in order to find the matching point for feature point p in image I, we usually need to seek two nearest neighbors from image J. An experiment is conducted on toy data, and as shown in Fig.3. When the number of bits m increases, the ratio that the nearest (second nearest) neighbor collides in the same bucket of the query decreases, and they will be close to zero when $m > 20$. At the same time, as m increases, the Hamming space will be more sparse and less candidates will be returned. This is very beneficial for the following step in terms of computation cost. To find a tradeoff between matching accuracy and computational cost, in the experiments, we set $m = 8$ or 10 and $L = 6$ to construct multiple hashing tables.

2.2.2 Hashing Remapping

After the coarse search in hashing lookup stage, one can typically carry out accurate search, e.g. calculate the Euclidean distance between every candidate with the query.

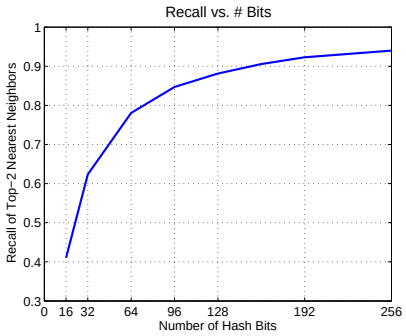


Figure 4. The recall of the two nearest neighbors when top 10 are returned while the number of bits changes.

However, the direct calculation of Euclidean distance requires a large amount of computation since the number of candidates is still large. To address this issue, a new n-bit (n $>$ m) hashing function is used to remap candidates provided by Hashing lookup stage into Hamming space. Afterwards, we sort the candidates according to their Hamming distance to the query and the top k items will be selected as the final candidates (described in Section 2.2.3).

When applying LSH to the feature points, e.g. SIFT, the ranking performance is closely related to the number of bits. In LSH, it can be theoretically guaranteed that longer binary codes will result in more discriminative hashing expression, and thus better performance of ranking. We conduct an experiment on toy data with different number of bits. Recall of top 2 (in terms of Euclidean distance) is shown in Fig.4 when top 10 (Hamming distance) is returned. In this paper, we remap the candidates into a 128-d Hamming space.

2.2.3 Top k Ranking via Hashing

In order to execute more precise search, in the third step, we expect to seek the k nearest neighbors in Hamming space, and then find the two nearest neighbors among them in Euclidean space. As the most commonly used method, the time complexity of linear search to find top k items is $O(kN)$. A tree-based data structure called min-heap was proposed to find the top k items and the time complexity is $O(log(k)N)$ [12]. In this paper, we propose a hashing based scheme to find the top k items, the time complexity of which is $O(N + k)$.

In detail, as computing the Hamming distance between the query and all points in the database, we use Hamming distance as keys to establish hashing buckets. The points with the same Hamming distance to the query will fall into the same bucket. For example, all points with Hamming distance 1 to the query will fall into the bucket whose key id is #1. When we use n bits for Hamming ranking, $n + 1$ buckets will be established at most. It worths to note that we only need to scan the database once to accomplish this process, therefore the time complexity is $O(N)$.

After this process is completed, we access points in the bucket in which the points' Hamming distance to the query is 0. If the number of points in this bucket is less than k, then access to next bucket in which the points' Hamming distance to the query is 1. This process is repeated until the number of obtained points summing to k. Obviously, the obtained k points are the top k candidates, and the time complexity is $O(k)$. Thus, the time complexity of the whole procedure is $O(N+k)$. And in most applications, k is much less than N.

After the top k candidates are found, we can find the two nearest neighbors among them according to the Euclidean distance, and matches that pass Lowes ratio test will be accepted [13].

3. Computational Complexity Analysis

Here we compare the computational complexity of our CasHash method to other related image matching methods. Assume we need to perform image matching between two images and each image contains N SIFT keypoints. We denote the consuming time for computing Euclidean distance and Hamming distance as T_E and T_H, respectively. In modern CPUs, it only takes one CPU cycle to compute Hamming distance for 128-bit hash code in Hamming space, thus T_H is much smaller than T_E.

For brute force matching, linear search required for N^2 times computation of Euclidean distance, thus the time cost is approximately $O\left(T_E \cdot N^2\right)$. In Kd-tree based approximate nearest neighbor search, the average single search complexity is $O\left(\log N\right)$ and total time cost is $O\left(T_E \cdot N \log N\right)$. LDAHash based matching also requires exhaustive search, but its Hamming embedding of descriptors reduces time cost to $O\left(T_H \cdot N^2\right)$.

In CasHash based matching with three-layer structure, there will be $LN/2^m$ points get through hashing lookup stage on average. The calculation of Hamming distance for these points takes $O\left(T_H \cdot LN/2^m\right)$ for each query. Another major time expenditure is the final search among top-k candidates, which costs $O\left(T_E \cdot k\right)$. In general, L and k are no more than 10. The complexity of computing the hashes is $O(dmLN)$ in the lookup stage and $O(ndLN/2^m)$ in the remapping stage, respectively. In practice, the lookup and remapping stages can be offline implemented. Thus, the overall computation complexity is $O\left(T_H \cdot LN^2/2^m + T_E \cdot Nk + dmLN + ndLN/2^m\right)$. In theory, larger m will lead to faster matching but lower precision. In our experiments, $m = 8$ can result in rather fast speed with comparable precision. Overall, it is easy to realize 10 times or more acceleration than Kd-tree based image matching with comparable performance. Extensive experiments will provide encouraging results.

	boat	trees	ubc	wall
Brute	13.959s	28.003s	2.033s	18.326s
KDTree	1.278s	2.010s	0.566s	2.044s
LDAHash	0.924s	1.861s	0.144s	1.234s
CasHash	**0.128s**	**0.223s**	**0.029s**	**0.173s**

Table 1. Comparison of SIFT point matching time for four algorithms: brute force, Kd-tree, LDAHash and CasHash.

4. Experiments

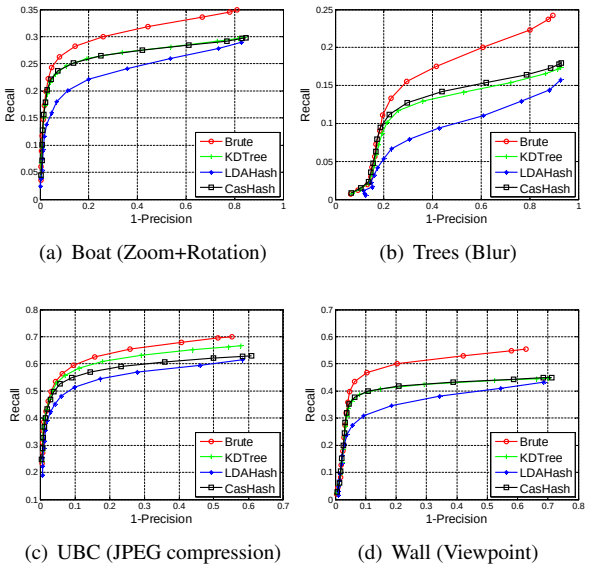
In our experiment, SIFT feature descriptors are extracted for pairwise images matching. In order to evaluate speed and accuracy of our proposed CasHash method, it is compared to three matching strategies, namely brute force matching, Kd-tree matching[15], and LDAHash matching[19]. We use Bundler[18] to process keypoints matching result and produce sparse point clouds. PMVS2 package[6] is used to generate dense reconstruction result for later visualization. 3D reconstruction results of several datasets are presented, ranging from high-quality image sets collected by specialists to low-quality image sets downloaded from Internet.

All experiments are carried out on a single desktop PC running Ubuntu 13.04 operating system with i7-2600 CPU and 8GB memory space. Neither parallel computation nor GPU acceleration technique is used in our experiments to ensure the fairness of comparison.

4.1. Results on Oxford Database

We use the standard Oxford dataset[14] to evaluate our SIFT keypoints matching performance and compare it with brute-force matching, Kd-tree matching and LDAHash matching. This dataset contains images with different geometric and photometric transformations, including blur, viewpoint change, zoom and rotation. Homography matrix between image pairs is given, thus for each SIFT keypoint detected in the first image, its expected position in the second image can be calculated. For any SIFT keypoint in the second image, if its distance to the expected position is lower than a certain threshold, then we assume that these two keypoints compose a ground-truth match pair. In our experiment, we set the distance threshold to 2.5.

The time of SIFT keypoints matching with various methods are measured and listed in Table 1. Due to limited space, only four sets of results are reported here. In Fig.5, we observe that brute force matching achieves best performance as we expected, since its searching strategy is linearly check all keypoints. However, it also takes the longest matching time among all methods as indicated in Table 1. Kd-tree matching provides approximately 10 times boost in matching speed and still maintains satisfying performance. LDAHash matching is slightly faster than Kd-tree, but leads to a significant drop in recall versus 1-precision curves. Our proposed CasHash matching achieves comparable performance to Kd-tree's and only takes about $1/10$ of Kd-tree's matching time.



(a) Boat (Zoom+Rotation) (b) Trees (Blur)

(c) UBC (JPEG compression) (d) Wall (Viewpoint)

Figure 5. Recall versus 1-precision curves on Oxford database.

From the above experiment results, we can see that hashing technique is an effective method to accelerate keypoints matching. However, speed boosting by hashing often results in performance deterioration, which has been reported in many other applications. The performance of LDAHash matching in our experiment also proves this conclusion. However, with carefully designed cascade structure, our proposed CasHash matching not only significantly improves matching speed, but also ensures the matching performance is comparable with that of Kd-tree. This may be benefit from our cascade structure, whose multiple distance metric framework is helpful to filter out noise pairs.

4.2. Results on Tsinghua Database

To evaluate the effectiveness of different keypoints matching algorithms for the final 3D reconstruction, it is necessary to compare the results with the real-world 3D model. In our experiment, we use two datasets with 3D ground truth data acquired by laser scanner, containing 102 and 193 photographs respectively[1].

We run different matching methods on these datasets and use Bundler together with PMVS2 to generate the reconstruction results. The 3D reconstruction results of our matching method are shown in Fig.6. The matching time statistics are recorded in Table 2 for comparison.

We used the same evaluation methodology as described in [17]. For simplicity, the ground-truth model is denoted as G and the candidate model is denoted as R. Accuracy is

[1] http://vision.ia.ac.cn/data/index.html

(a) Life.Science.Building: 102 images.

(b) Tsinghua.School: 193 images.

Figure 6. 3D reconstruction results on dataset "Life.Science.Building" and "Tsinghua.School". Left: real-world photograph; Middle: CasHash-8Bit; Right: CasHash-10Bit.

Method	Dataset-Life.Science.Building		
	T_{Match}	Speed-Up	Points
Brute	76655s	1.00×	457907
KDTree	4689s	16.35×	435275
LDAHash	4782s	16.03×	490146
CasHash-8Bit	639s	119.96×	**497944**
CasHash-10Bit	**316s**	**242.58**×	407562
Method	Dataset-Tsinghua.School		
	T_{Match}	Speed-Up	Points
Brute	46488s	1.00×	680966
KDTree	4877s	9.53×	683461
LDAHash	2522s	18.43×	684524
CasHash-8Bit	402s	115.64×	676223
CasHash-10Bit	**243s**	**191.31**×	**697366**

Table 2. Reconstruction performance on Tsinghua database.

to assess how close R is to G and completeness is to assess how much of G is modeled by R.

The distances between points on R and their respective nearest points on G are computed to measure the accuracy. Accuracy at $X\%$ is defined as the minimal distance value so that $X\%$ of points on R are within this distance of G. Similarly, the completeness is measured by the distances between points on G and their respective nearest points on R. Completeness at $X\%$ is defined as the minimal distance value so that $X\%$ of points on G are within the distance of R. The accuracy and completeness evaluation results are demonstrated in Table 3 and Table 4.

From Table 3 we can see that our proposed CasHash-8Bit matching strategy is remarkably better than all other methods in dataset "Life.Science.Building". As for dataset "Tsinghua.School", the accuracy statistics of Kd-tree based matching are superior to the rest. CasHash-8Bit consistently outperforms the rest in completeness statistics as shown in Table 4 in both two datasets.

According to the experiment results above, it is shown that our CasHash matching strategy be able to achieve comparable (even slightly better) performance against Kd-tree and LDAHash in most cases.

4.3. Results on Flickr Database

With the rapidly increasing number of photographs uploaded to the Internet, it is possible to rebuild 3D scenes of famous tourist attractions solely based on these web images. We harvest photo collection of four famous landmarks from flickr website: Taj Mahal, Statue of Liberty, Notre Dame de Paris and Colosseum.

We present our reconstruction results in Fig.7 and reconstruction statistics of all matching methods in Table 5 for comparison. As shown in Table 5, our proposed CasHash-matching strategy achieves more than 10 times boost in speed for SIFT keypoints matching, comparing with commonly used Kd-tree matching strategy. At the same time, our reconstruction performance maintains at a comparable level with other keypoints matching methods, considering the amount of points used in the final dense reconstruction phase as an evaluation index.

5. Conclusions

In this paper, we proposed a Cascade Hashing method to speed up the image matching. Extensive experiments show that image matching can be accelerated by our approach in hundreds times than brute force matching, even achieves ten times or more than Kd-tree based matching while retaining comparable accuracy.

Method	Life.Science.Building									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Brute	4.95e-3	9.88e-3	1.49e-2	2.01e-2	2.58e-2	3.25e-2	4.17e-2	5.86e-2	1.05e-1	2.93
KDTree	4.68e-3	9.23e-3	1.38e-2	1.86e-2	2.41e-2	3.10e-2	4.07e-2	5.80e-2	1.03e-1	2.95
LDAHash	4.84e-3	9.56e-3	1.42e-2	1.92e-2	2.46e-2	3.14e-2	4.11e-2	5.72e-2	1.01e-1	2.94
CasHash-8Bit	5.39e-3	1.06e-2	1.58e-2	2.11e-2	2.68e-2	3.38e-2	4.40e-2	6.12e-2	1.05e-1	2.92
CasHash-10Bit	4.75e-3	9.47e-3	1.43e-2	1.96e-2	2.56e-2	3.30e-2	4.31e-2	6.10e-2	1.07e-1	2.92
Method	Tsinghua.School									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Brute	2.98e-3	5.93e-3	8.92e-3	1.21e-2	1.61e-2	2.13e-2	2.90e-2	4.17e-2	7.11e-2	2.25
KDTree	4.37e-3	8.98e-3	1.39e-2	1.91e-2	2.50e-2	3.19e-2	4.04e-2	5.29e-2	8.00e-2	1.93
LDAHash	2.99e-3	6.13e-3	9.61e-3	1.36e-2	1.85e-2	2.44e-2	3.23e-2	4.45e-2	7.35e-2	1.53
CasHash-8Bit	2.85e-3	5.70e-3	8.67e-3	1.21e-2	1.63e-2	2.19e-2	2.94e-2	4.11e-2	6.78e-2	1.43
CasHash-10Bit	3.87e-3	7.98e-3	1.25e-2	1.76e-2	2.34e-2	3.05e-2	3.97e-2	5.32e-2	8.18e-1	1.78

Table 3. Accuracy of 3D reconstruction on Tsinghua database.

Method	Life.Science.Building									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Brute	1.52e-2	2.06e-2	2.59e-2	3.18e-2	3.91e-2	4.93e-2	6.44e-2	8.96e-2	1.52e-1	1.95
KDTree	1.53e-2	2.06e-2	2.58e-2	3.15e-2	3.87e-2	4.88e-2	6.39e-2	9.04e-2	1.61e-1	1.90
LDAHash	1.50e-2	2.02e-2	2.52e-2	3.06e-2	3.75e-2	4.68e-2	6.07e-2	8.47e-2	1.45e-1	1.75
CasHash-8Bit	1.55e-2	2.09e-2	2.59e-2	3.15e-2	3.85e-2	4.82e-2	6.27e-2	8.71e-2	1.48e-1	1.52
CasHash-10Bit	1.56e-2	2.12e-2	2.66e-2	3.26e-2	4.00e-2	5.02e-2	6.58e-2	9.17e-2	1.58e-1	1.68
Method	Tsinghua.School									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Brute	1.33e-2	1.72e-2	2.05e-2	2.39e-2	2.77e-2	3.22e-2	3.82e-2	4.73e-2	6.62e-2	8.26e-1
KDTree	1.50e-2	1.94e-2	2.33e-2	2.71e-2	3.13e-2	3.61e-2	4.22e-2	5.12e-2	6.85e-2	8.28e-1
LDAHash	1.35e-2	1.75e-2	2.10e-2	2.45e-2	2.83e-2	3.28e-2	3.86e-2	4.73e-2	6.63e-2	7.85e-1
CasHash-8Bit	1.34e-2	1.73e-2	2.07e-2	2.41e-2	2.79e-2	3.23e-2	3.79e-2	4.65e-2	6.48e-2	9.63e-1
CasHash-10Bit	1.47e-2	1.91e-2	2.31e-2	2.72e-2	3.17e-2	3.71e-2	4.41e-2	5.46e-2	7.52e-2	7.11e-1

Table 4. Completeness of 3D reconstruction on Tsinghua database.

6. Acknowledgements

The authors would like to thank Prof. Zhanyi Hu and Dr. Wei Gao for their constructive suggestion. This work was supported in part by 973 Program Project under grant No.2010CB327905 and National Natural Science Foundation of China under grant No.61332016.

References

[1] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In *International Conference on Computer Vision (ICCV)*, 2009.

[2] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[3] M. Charikar. Similarity estimation techniques from rounding algorithm. In *ACM symposium on Theory of computing*, pages 380–388, 2002.

[4] D. Crandall, A. Owens, N. Snavely, and D. P. Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2011.

[5] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pages 253–262, 2004.

[6] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(8):1362–1376, 2010.

[7] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[8] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[9] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *ACM symposium on Theory of computing (STOC)*, pages 604–613, 1998.

[10] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[11] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

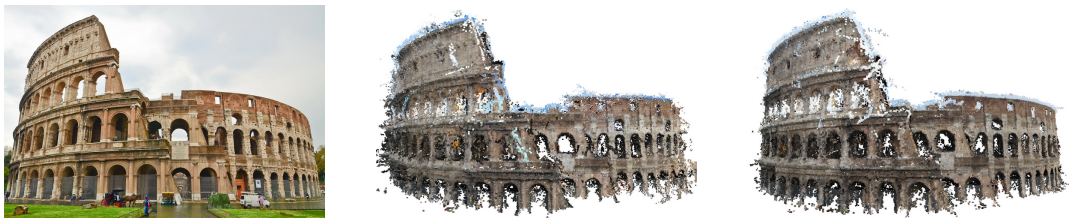
[12] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

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

(a) Taj.Mahal: 189 images.

(b) Statue.of.Liberty: 674 images.

(c) Notre.Dame: 715 images.



(d) Colosseum: 1357 images.

Figure 7. 3D reconstruction results on Flickr dataset "Taj.Mahal", "Statue.of.Liberty", "Notre.Dame" and "Colosseum". Left: real-world photograph; Middle: CasHash-8Bit; Right: CasHash-10Bit.

Method	Taj.Mahal			Statue.of.Liberty		
	T_{Match}	Speed-Up	Points	T_{Match}	Speed-Up	Points
Brute	52575s	1.00×	**124038**	30608s	1.00×	88001
KDTree	5136s	10.24×	119165	9765s	3.13×	98674
LDAHash	1774s	29.64×	120353	874s	35.02×	123274
CasHash-8Bit	301s	174.67×	118331	358s	85.50×	203587
CasHash-10Bit	**183s**	**287.30**×	116224	**231s**	**132.50**×	**246206**
Method	Notre.Dame			Colosseum		
	T_{Match}	Speed-Up	Points	T_{Match}	Speed-Up	Points
Brute	396729s	1.00×	358121	12307s	1.00×	**540308**
KDTree	60663s	6.54×	347056	2430s	5.06×	445774
LDAHash	13136s	30.20×	413348	851s	14.46×	492040
CasHash-8Bit	2266s	175.08×	**484960**	222s	55.44×	393408
CasHash-10Bit	**1354s**	**293.01**×	400673	**196s**	**62.79**×	512508

Table 5. Reconstruction statistics of four matching methods: brute force, Kd-tree, LDAHash and CasHash. Note: vocabulary tree technique has been used for dataset "Colosseum" to accelerate image matching [16].

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

[15] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, pages 331–340, 2009.

[16] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[17] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Computer vision and pattern recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 519–528. IEEE, 2006.

[18] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM transactions on graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.

[19] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):66–78, 2012.