GPU Accelerated Cascade Hashing Image Matching for Large Scale 3D Reconstruction

Tao Xu, Kun Sun, Wenbing Tao*

School of Automation, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract

Image feature point matching is a key step in Structure from Motion (SFM). However, it is becoming more and more time consuming because the number of images is getting larger and larger. In this paper, we proposed a GPU accelerated image matching method with improved Cascade Hashing. Firstly, we propose a Disk-Memory-GPU data exchange strategy and optimize the load order of data, so that the proposed method can deal with big data. Next, we parallelize the Cascade Hashing method on GPU. An improved parallel reduction and an improved parallel hashing ranking are proposed to fulfill this task. Finally, extensive experiments show that our image matching is about 20 times faster than SiftGPU on the same graphics card, nearly 100 times faster than the CPU CasHash method and hundreds of times faster than the CPU Kd-Tree based matching method. Further more, we introduce the epipolar constraint to the proposed method, and use the epipolar geometry to guide the feature matching procedure, which further reduces the matching cost.

Keywords: Image matching, SIFT, Cascade Hashing, GPU

1. Introduction

Image feature matching is to find the correspondence between images, which is widely used in image mosaic, object detection and 3D reconstruction. SIFT [1]

*Corresponding author.
Email address: wenbingtao@hust.edu.cn (Wenbing Tao)

Preprint submitted to arXiv.org May 24, 2018
is the most often used image feature since it has strong anti-interference ability and high matching accuracy. However, with the development of the Internet and information technology, the amount of image data is getting larger and larger in many computer vision applications. Especially in large scale Structure from Motion (SFM) [2, 3], image matching occupies the major computational cost because tens of thousands of images need to be handled. Therefore, fast image matching method is extremely important for accelerating the entire process of 3D reconstruction.

In 2014, Cheng et al. proposed a fast image feature matching method based on Cascade Hashing in 3D reconstruction [4]. The Cascade Hashing (CasHash) uses the Locality Sensitive Hashing [5] (LSH) to quickly determine and reduce a candidate point set, so CasHash avoids fully calculating Euclidean distance between all possible points. Compared with many other algorithms, such as Kd-Tree and LDAHash, CasHash has a quite low time complexity. As reported, on the same CPU computing platform, CasHash is able to achieve slightly better performance against Kd-tree and LDAHash in most cases.

Although CasHash is very fast for image matching, it is still time-consuming when the number of image pairs to be matched is extremely large. For example, for exhaustive matching of Dataset Rome16K [6], there are 115 million image pairs need to be matched and it will spend half a year by CasHash matching algorithm. Although Bag of Feature [7] and Vocabulary Tree [8] indexing are used to reduce the matching list to 30 neighbors, it still costs about 9 hours. Therefore it is necessary to further improve the matching efficiency. With the rapid development of hardware, GPU card is becoming more and more popular. It has been extensively used for high-performance and massively parallel computation. Considering the independence of the matching between different images, which is very suitable for parallelization, we intend to implement GPU-based CasHash to accelerate the matching procedure in large scale SFM 3D reconstruction in the work. However, there are some difficulties that need to be overcome when parallelizing CasHash:

(1) For large scale 3D reconstruction, massive image pairs need to be pro-
cessed. Limited by memory and GPU memory, the whole data can’t be loaded at once. So we need to design an efficient partially data loading and exchanging strategy, which also minimizes the data exchange cost at the same time. (2) In CasHash, there are many reduction (such as calculation of hashing code) and sorting (such as hashing ranking) operations which are the most time consuming parts of CasHash. In parallel computing, reduction and sorting are inefficient operations but can be specialized rewritten for CasHash.

In order to solve these problems, we propose a GPU accelerated Cascade Hashing Image Matching algorithm for large scale 3D reconstruction. A preliminary version of this work appeared in [9]. Firstly, a Disk-Memory-GPU data exchange strategy is proposed to optimize the load order for massive images. Secondly, an improved parallel reduction method is proposed to calculate hashing code which is used to determine and reduce a candidate point set. Then we use parallel hashing ranking to select a few nearest neighbors. Finally, the improved parallel reduction method is also used to calculate Euclidean distance between the nearest neighbors and copy output asynchronously.

2. Related Work

The development of feature descriptors has made a great contribution to the performance of image matching. In 1999, David Lowe proposed SIFT (Scale-invariant feature transform) feature [10] and concluded in 2004 [11]. SIFT feature is represented by a 128 dimensional vector and uses the Euclidean distances for similarity measurement. In 2004, Yan Ke et al. [11] proposed PCA-SIFT to reduce the complexity of matching. PCA-SIFT use PCA (Principal Component Analysis) to reduce the SIFT feature descriptor from 128 dimension to 20 dimension, which reduces the memory occupied and improves the efficiency of the matching. In 2004, Mikolajczyk et al. [12] proposed a variant of SIFT descriptor called GLOH (Gradient Location-Orientation Histogram) that uses log-polar bins instead of square bins to compute orientation histogram. GLOH improves the matching accuracy greatly, but it takes more time to extract features from
images. In 2006, Herbert Bay et al.[13] proposed SURF (Speeded Up Robust Features) descriptor that can improve the robust and speed of feature extract, but SURF is affine invariant under low angle. Based on SURF, Yanwei Pang et al.[14] proposed a fully affine invariant SURF algorithm with high computation efficient at the same time. In recent years, there are also some binary feature descriptors adapted to embedded systems and mobile platforms because of low memory usage and small computational complexity. In 2010, Calonder et al. [15] introduced the BRIEF (Binary Robust Independent Elementary Features) descriptor that relies on a relatively small number of intensity difference tests to represent an image patch as a binary string. There are also other binary features such as BRISK[16], FREAK[17] and etc.

Despite the improvement of descriptors, a series of SIFT feature matching methods are proposed in order to accelerate the speed of image matching. Based on data structure of the binary tree, Kd-Tree [18] is one of the ANN methods. The time complexity of Kd-Tree is $O(N \log N)$ [19], which is much faster than brute search. To solve the problem of high-dimensional SIFT feature points, LDAHash [20] was proposed based on the data structure of hashing. The high 128 dimensional vectors can be reduced into binary hashing codes. And then LDAHash uses the hashing codes as the key to find the nearest neighbors. CasHash is inspired by LDAHash and has more advantages, such as coarse-to-fine search strategy and etc. It makes CasHash faster and resistent to noise point pairs. Alhwarin et al. [21] proposed VF-SIFT that extend SIFT feature by a few pairwise independent angles. Then this method classifies SIFT features based on their introduced angles into different clusters. Only SIFT features that belong to same cluster are compared.

On the other hand, in Structure from Motion, there are some other methods to reduce the matching time by changing exhaustive matching into guided matching. Vocabulary Tree Clustering [8] uses a hierarchically quantized cluster tree to enable scalable clustering and recognition of millions of images. Recognition or classification is performed by running down the tree with each image feature and scoring based on branch traversal. Instead of exhaustive matching,
guided matching can be carried out between the images with high similarity score. However, Vocabulary Tree Clustering may miss a small part of correct matches. Li et al. [22] presents an approach that clusters images based on low-dimensional global appearance descriptors, and the clusters are refined using 3D geometric constraints. Thus this method reduces the matching pairs. Each valid cluster is represented by a single iconic view, and the geometric relationships between iconic views are captured by an iconic scene graph. Wu et al. [23] introduce a preemptive feature matching strategy that can reduce the matching pairs by up to 95% while still recovering sufficient good matches for reconstruction. For each image pair (parallelly), it do the following: (a) Match the top-scale features of the two images. (b) If the number of matches from the subset is small, return and skip the next step. (c) Do regular matching and geometry estimation. In this paper, we use GPU accelerated geometry estimation method to calculate fundamental matrix in parallel instead of using the classical RANSAC algorithm. Michal Havlena et al. [24] proposed VocMatch to establish point correspondences between all pairs of images by a sufficiently large visual vocabulary. Instead of matching each individual image pair, VocMatch directly gets the reconstruction trajectory.

In 3D reconstruction of large scale urban scenes, millions of images need to be processed. Even with the state-of-the-art algorithms, it takes large time to complete 3D reconstruction. As in the paper "Building Rome in a Day" [2], Sameer Agarwal et al. carried out the experiment that reconstructing cities consisting of 150K images spends less than a day on a cluster with 500 compute cores. With the development of computing platforms and devices, especially CUDA and GPU, a single super PC with GPU cards can achieve similar or better performance than a cluster of CPUs, while the cost is much smaller [25]. Jan-Michael Frahm et al. parallelize the processing of 3D reconstruction on modern multi-core CPUs and GPU cards. This paper processed 3 million images to reconstruction within the span of a day on a single PC.

There are also some parallelizing methods further reducing the matching time. Wu et al. [23] also parallelize the brute force matching on GPU called
SiftGPU matching. Shah et al. [26] proposed a parallelizing matching method mainly based on parallelizing Kd-tree. So combine GPU with low time complexity algorithm, the matching time will be reduced greatly.

3. Overview

3.1. Review of Cascade Hashing

CasHash [4] uses two layers of LSH for efficient nearest neighbor search in a coarse-to-fine manner. LSH [5] is a kind of ANN method, whose average time complexity is $O(1)$. Through hashing mapping, true neighbors are more likely to be assigned to the same bucket, that is, the same hashing code. It consists of the following steps.

**Step 1. Hashing lookup with multiple tables** All image feature points are mapped into $m$-bit short hashing codes by $m$ different LSH hashing functions. However, using only one hashing table will lead to comparatively large possibility that the real matching pair falls into different buckets. So the above hashing is carried out multiple times, returning multiple lookup tables. Each query point in image $I$ will treat all the points with the same hashing code in image $J$ as its matching candidate.

**Step 2. Hashing remapping** The query and all the candidate points are mapped into $n$-bit ($n > m$) hashing codes with LSH. Since these hash codes are longer than previous ones, the number of points falling into the same bucket with the query is quite small. CasHash computes the Hamming distances between the query point and its candidates.

**Step 3. Hashing ranking** The Hamming distance between the query and the candidates after hashing remapping ranges from 0 to $n$. CasHash then collects the top $k$ points with the smallest Hamming distance. These points are treated as the final matching candidates of the query point.

**Step 4. Euclidean distance calculation** Finally, the Euclidean distances of the SIFT feature descriptors between the query point in image $I$ and
its $k$ nearest neighbors on image $J$ are computed. Then the point with the smallest distance which also passes Lowe’s ratio test \cite{1} is selected as the matching point.

3.2. GPU accelerated Cascade Hashing Image Matching

In this paper, we propose a GPU accelerated Image Matching with Cascade Hashing. The flow chart of our method is shown in Fig.1. Firstly, we propose an improved parallel reduction on GPU. By making full use of shared memory and registers on GPU, we use registers to do the last $N_r$ rounds of the reduction instead of using shared memory in every round. In step 1, 2 and 4, the calculation of hashing code and Euclidean distance can make use of the improved parallel reduction for acceleration.

In 3D reconstruction of large scale scenes, there are tens of thousands of images to be processed. In this case, the memory usage is easy to exceed the capacities of the main memory and GPU memory. We propose a Disk-Memory-GPU data exchange strategy. A certain number of images are bundled into data blocks, and a certain number of blocks are bundled into data groups. And then we optimize the load order to avoid redundant the data exchange.

In step 3 of CasHash, in order to reduce the computational bottleneck in hashing ranking, we propose a improved parallel hashing ranking which use the threshold $\tau$ to filter the points in the middle of a few buckets. The computational bottleneck can be reduced because the points concentrating in the middle of a few buckets are filtered.

Further more, inspired by the preemptive feature matching strategy \cite{23}, we use top-scale SIFT features to do exhaustive image matching and compute the epipolar geometry for image pairs. Then the geometric information is used to guide the remaining matching.

4. Three-Level Data Scheduling Strategy

Most computational tasks of the proposed method are performed on the graphics card. However, when the number of images scales up, it is impossible
Figure 1. The flow chart of our proposed approach. It contains two modules: hash code calculating module and image matching module. An efficient data scheduling strategy between the disk, memory and graphics card is applied to adapt to large scale data.

to load all the data from the disk to memory and graphics card. Thereby in this section we propose a three-level data scheduling strategy, which satisfies the following two constraints: 1) loading data within the capacity of each device and 2) making full use of the computational units.

Generally speaking, the capacities of disk, memory and graphics card decrease in order. So we organize the whole data in three different granularities accordingly: the whole set, groups and blocks. As is shown in the last row of Fig. 2 and Fig. 3, a block contains data from \( N_p \) images and such \( M \) blocks constitute a group. The whole data set consists of \( N \) groups and is stored on the disk. For clarity, we will denote \( G_i \) as the \( i - th (i \in [1,N]) \) group and \( B_{ij} \) as the \( j - th (j \in [1,M]) \) data block in \( G_i \). In the following part we will introduce two slightly different data scheduling strategies for hash code calculation and image matching because the former operates on a single image while the latter operates between two images.
4.1. Data Scheduling Strategy for Hashing Code Calculation

When calculating hash code, we just need to progressively load data to the graphics card in two lines. The first line loads group $G_i$ from the disk to the memory, and then loads one of its un-processed block $B_{ij}$ from memory to the graphics card. It is shown as the green arrows in Fig. 2. The block $B_{ij}$ will be replaced with the next un-processed block $B_{ij+1}$ if all the data in it has been processed. Similarly, the group $G_i$ will be replaced with another group $G_{i+1}$ if all its blocks have been processed. The second line runs in parallel with the first line on another thread, which is shown as the red arrows in Fig. 2. It loads $G_{i+1}$ from the disk when $G_{i}$ is still in memory, and loads $B_{ij+1}$ from memory when $B_{ij}$ is in the graphics card.

Note that there are two blocks/groups in the graphics card/memory at the same time, but only one of them is in progress. Although $G_{i+1}$ and $B_{i+1}$ are coexistent with $G_i$ and $B_{i}$, they will not be processed until $G_i$ and $B_{i}$ are finished and released. Keeping the next group or block loaded in advance ensures that there is no need to wait for new data after the current data is finished, which is more efficient.
4.2. Data Scheduling Strategy for Image Matching

Since image matching operates on a pair of images, the data scheduling strategy becomes more complex. In order to cover all possible image pairs, we need to match all possible group pairs and all possible block pairs. As is shown in Fig.3, the strategy here has two lines as well. The first line loads a pair of groups \( G_i \) and \( G_k \) from the disk to the memory, and then loads one block from each group \( B_j^i \) and \( B_l^k \) to the graphics card. Data exchanges more frequently between the graphics card and memory than between the disk and memory. After images in \( B_l^k \) have been fully matched to images in \( B_j^i \), it will be replaced with another new block \( B_l^{k+1} \). If \( B_j^i \) has been matched to all the blocks in \( G_k \), we match images within \( B_j^i \) itself and then replace \( B_j^i \) with the next block \( B_j^{i+1} \). The same rule applies to groups, too. After all the blocks in group \( G_k \) have been matched to all the blocks in \( G_i \), it will be replaced with another new group \( G_{k+1} \). If \( G_i \) has been matched to all the other groups, we match blocks within \( G_i \) and then replace \( G_i \) with the next group \( G_{i+1} \). For the same reason described in the previous subsection, the second line which runs on another thread loads \( B_l^{k+1} \) and \( G_{k+1} \) in advance while existing data is in progress.

The difference with previous subsection is that we focus on pairs rather than individuals. There are three blocks/groups maintained in the graphics card/memory at the same time, and two out of three are in progress. Matching is carried out not only between different blocks/groups, but also within the same block/group.

5. Fast Computation of Reduction on GPU

5.1. The Improved Parallel Calculation of Hashing Code

In [5], Charikar defined the hashing function by using inner product similarity. In step 1 and step 2 of CasHash, when calculating the hashing code of every SIFT point, we need to compute the inner product of the high dimensional vector. Let \( d \) be the dimension of the query vector, so the CUDA kernel
The function of the hashing map is $Hashing \_kernel<<<N \ast n, d>>>(PR, PQ)$, where $PR$ and $PQ$ are the pointers of random vectors $r_k$ and SIFT feature descriptors in the GPU global memory, respectively. Since the SIFT features point is 128 dimensional, i.e. $d = 128$. $N$ is the number of points, and $n$ is the length of hashing code. The kernel function totally calls $N \ast n$ blocks and each block calls $d$ threads. The inner product vector needs to do $d$ accumulation. Since the GPU global memory access is relatively slow and GPU needs to access GPU global memory repeatedly in the accumulation step, we use the GPU shared memory to reduce access delay [27]. The shared memory is a kind of GPU cache. The products are stored in the shared memory temporarily when doing accumulation, thus the arithmetic speed is improved.

There are 128 threads participating in operation at the same time in each block. Each thread reads a component of $PR$ and a component of $PQ$, and then multiplies them on shared memory. Data access in shared memory is very fast, but using serial accumulation in each block will activate only one thread, which will reduce arithmetic efficiency. The way to deal with vector accumulation in general parallel computing is parallel reduction [28], which is also calculated in the shared memory.

The classic GPU parallel reduction is shown in Fig. 4. The accumulation of each block added in the shared memory makes full use of threads. For example,
in the first round, the accumulation of 128 elements calls 64 threads, and then obtains 64 results. Consequently, the numbers of elements and threads are reduced by half after every round of operation until the last round, which uses one thread to add two elements. Accumulation is a process of reducing the number of threads used.

The access time for GPU global memory, GPU shared memory and GPU registers are hundreds of operational cycles, 10 operational cycles and a single operation cycle [29], respectively. So if registers instead of shared memory is used in the parallel reduction, it can further reduce the delay. However, all the registers could only be accessed by a single thread. If we use the registers in the early stage of parallel reduction, most threads will be idle. In this paper, we improve the parallel reduction by using both the shared memory in the early stage and the registers in the final stage to achieve faster speed.

The proposed method is shown in Fig. 5, the registers are used instead of the shared memory in the last $N_r$ rounds of the reduction. Since the number of threads is small in the last several rounds, using registers will not waste too
many threads but save much shared memory access time. In Fig. 5 in the last two rounds, the classic method of parallel reduction requires to access shared memory 2 times, need to do addition 2 times. So it needs about 22 cycles to complete the operation of the last two calculations. And after the use of registers, the proposed method need to access registers 3 times, and need to do addition 3 times. So it needs about 6 cycles to complete the operation of the last two calculations. Thus the proposed method greatly improves the speed of operation. In section 4, we show the relationship between the $N_r$ rounds registers involved in and the time of image matching.

5.2. Improved Parallel Euclidean Distance Calculation

In step 4 of CasHash, calculating the Euclidean distances still spends a lot of time in CPU, and the Euclidean distance formula is $\sum_{i=0}^{d-1} (x_1^i - x_2^i)^2 = \Delta x \cdot \Delta x$. Therefore, the Euclidean distance is essentially a vector inner product, which is the same as that in the section 3.2. After Euclidean distance calculation and Lowe’s ratio test [1], the output (SIFT matching pairs) can be copied into hard disk. Since CPU and GPU are different devices, the operations of CPU and GPU are asynchronous [30]. When doing image matching on GPU, CPU can copy the output from GPU memory to host memory and then to hard disk. This
method hides the output time, and improves the overall operation efficiency.

6. Improved Parallel Hashing Ranking

In step 3 of CasHash, we propose an improved parallel hashing ranking method. Each GPU block ranks the Hamming distance between a query point and its candidate points in an image, and the process of constructing a index table in a GPU block is shown in Fig.6. As the histogram showed, the horizontal axis is the Hamming distance and the vertical axis is the number of points that fall in the bucket. On the one hand, since the candidate points have passed through rough filtering, few outliers are included. So few Hamming distances are large. On the other hand, there are $2^{128}$ kinds of different hashing codes, there are few Hamming distances near 0. Therefore in experiments, the candidate points are concentrated in the middle of the hashing bucket. If two candidate points have the same Hamming distance, there will be conflict when storing them into the same hashing bucket. In order to avoid the error caused by the conflict, it is necessary to use the atomic operation of CUDA. The atomic operation can make the conflict become serial task[^31], which will reduce the efficiency of operation. And the time of conflict in the middle of the hashing bucket is most. The efficiency of parallel computation follows the bucket principle, so the computation time depends on the points number in the middle of the hashing bucket.

However, we only need the first $k$ nearest neighbors. $k$ is less than the number of hashing buckets, so most of the $k$ nearest neighbors is in the first few buckets. In Fig.6 we set a threshold $\tau$ in parallel hashing ranking. The candidate points whose Hamming distances to the query are greater than the threshold $\tau$ are discarded and not stored in the bucket and the candidate points whose Hamming distances to the query are smaller than the threshold $\tau$ are stored in the bucket. Therefore the threshold $\tau$ will reduce the computational bottleneck and improve efficiency. But if the threshold $\tau$ is too small, we may filter the real matching point which will reduce matching accuracy. In this
paper, the influence of the threshold on the matching accuracy and matching time is illustrated by experiments in section 4.

7. Image Matching with Geometry-aware Cascade Hashing on GPU

In order to further improve the matching speed, we introduce the epipolar constraints to the above mentioned CasHashGPU framework. The new method performs in two steps. In the first step, only few top scale features are matched and a GPU accelerated fundamental matrix estimation method is implemented to compute the epipolar geometry between a pair of images. In the second step, the epipolar geometry is used to guide the matching for the remaining features. The improved geometry-aware two-step feature matching method is called Ga-CasHashGPU.

7.1. First Stage of Image Matching

Inspired by [32][23], we use the top 20% scale SIFT features to do exhaustive image matching by CasHashGPU. Since only a small portion of feature points are involved, this stage is very fast. If at least 16 matches are found between images, then we also use GPU to estimate the fundamental matrix $F$ between
the images. And if at least $2/3$ of the initial matches are inliers, the image pair will be processed in the guided matching.

7.2. Second Stage of Guided Matching with Geometry-aware CasHashGPU

The epipolar geometry estimated before is further used to guide the remaining matching procedure, which is called Geometry Guided CasHashGPU. It consists of the following steps. step 1. hashing lookup with multiple tables on GPU; step 2. hashing remapping on GPU; step 3. epipolar constraints to reduce the candidate points; step 4. improved parallel hashing ranking; step 5. improved parallel Euclidean distance calculation.

In geometry-aware feature matching [26], epipolar constraints are defined as following. $pq = (x_q y_q 1)$ is a query point in image $I$, and $p'q = (x'_q y'_q 1)$ is the corresponding matching point in image $J$ of $pq$. $l_q = (a_q b_q c_q)$ is the corresponding epipolar line in image $J$, thus the candidate matching feature set of $pq$ $C$ is,

$$C = \{p'|dist(p', l_q) \leq d\} \quad (1)$$

$$dist(p', l_q) = \frac{a_q x' + b_q y' + c_q}{\sqrt{a_q^2 + b_q^2}} \quad (2)$$

where $d$ is the distance of the candidate point $p'$ to the epipolar line $l_q$, and $d$ is the threshold of the epipolar constraints. Thus the epipolar constraints are showed in Fig.7. The epipolar constraints reduce the candidate points, so the proposed method reduce the points which will be sorted by improved parallel hashing ranking. Thus this way improves the overall method efficiency.

8. Experiments and Results

In this paper, we extracted SIFT features from the testing image data sets and perform image matching. In the experiment, the two proposed methods(CasHashGPU and Ga-CasHashGPU) are compared with Kd-Tree [18], Sift-GPU [33] and the original CasHash [4]. The experiments of Kd-Tree and the
original CasHash are tested on E5-2630 v3@2.4GHz CPU without parallel computation. SiftGPU and the proposed method are tested on one GTX Titan X GPU. All the experiments are running in Ubuntu 14.04 operating system. In addition, in order to test the acceleration of the proposed method on multi-GPU, we test the CasHashGPU on a PC with 8 GTX 1080Ti GPUs.

Table 1 and 2 show the comparison of four methods above. The Ga-CasHashGPU performs epipolar-geometry guided exhaustive matching and other method perform exhaustive matching without priors. CasHashGPU is about 20 times faster than SiftGPU on the same graphics card, nearly 100 times faster than the CPU CasHash method and hundreds of times faster than the CPU Kd-Tree based matching method. Ga-CasHashGPU calculates much fewer matching pairs than CasHashGPU. So Ga-CasHashGPU is about 3 to 7 times faster than CasHashGPU on the same graphics card. The experiments show that the speedup of our method is more obvious when the number of images is larger, such as the "pozzoveggiani" data set. This is because when the number of images increases, the scale of the problem increases quadratically.
Table 1. Results on Matching Experiments. Kd-Tree, CasHash, Sift-GPU and CasHashGPU performs exhaustive matching without any geometric priors. Ga-CasHashGPU performs two-stage guided exhaustive matching.

(a) Data-pozzoveggiani (54 images) 8306 mean points; exhaustive matching 1431 pairs, guided matching 34 pairs

| Method          | time(s)  | speed(pairs/s) | speedup |
|-----------------|----------|----------------|---------|
| Kd-Tree         | 698.921  | 2.05           | 1.00×   |
| CasHash         | 183.218  | 7.81           | 3.81×   |
| SiftGPU         | 30.103   | 47.54          | 23.22×  |
| CasHashGPU      | 2.496    | 573.32         | 280.02× |
| Ga-CasHashGPU   | 0.736    | 1990.49        | 949.63× |

(b) Data-erpbero (259 images) 8641 mean points; exhaustive matching 33411 pairs, guided matching 780 pairs

| Method          | time(s)  | speed(pairs/s) | speedup |
|-----------------|----------|----------------|---------|
| Kd-Tree         | 1.479×10^4 | 2.26         | 1.00×   |
| CasHash         | 1461.525 | 22.86         | 10.12×  |
| SiftGPU         | 752.164  | 44.42         | 19.66×  |
| CasHashGPU      | 34.394   | 971.42        | 429.91× |
| Ga-CasHashGPU   | 15.701   | 2177.63       | 941.77× |

(c) Data-Aos_Hus (811 images) 7768 mean points; exhaustive matching 328455 pairs, guided matching 7337 pairs

| Method          | time(s)  | speed(pairs/s) | speedup |
|-----------------|----------|----------------|---------|
| Kd-Tree         | 1.456×10^5 | 2.26         | 1.00×   |
| CasHash         | 2.800×10^4 | 11.73        | 5.20×   |
| SiftGPU         | 6971.441 | 47.11         | 20.89×  |
| CasHashGPU      | 292.541  | 1122.77       | 497.81× |
| Ga-CasHashGPU   | 90.189   | 3723.20       | 1614.652× |
Table 2. Results on Matching of Large Data Sets. Because Data-Dubrovnik6K and Data-Rome16K are large data sets, Kd-Tree, CasHash and SiftGPU will spend a lot of time, we only compare and our two method.

(a) Data-Dubrovnik6K [6] (6044 images) 7438 mean points; exhaustive matching 1.826×10^7 pairs, guided matching 58611 pairs

| Method          | time(s)  | speedup |
|-----------------|----------|---------|
| CasHashGPU      | 1.054×10^4 | 1.00×   |
| Ga-CasHashGPU   | 1548.89  | 6.80×   |

(b) Data-Rome16K [6] (15178 images) 7891 mean points; exhaustive matching 1.152×10^8 pairs, guided matching 145101 pairs

| Method          | time(s)  | speedup |
|-----------------|----------|---------|
| CasHashGPU      | 1.565×10^5 | 1.00×   |
| Ga-CasHashGPU   | 20863.68 | 7.50×   |

Fig.8 shows the matching time on the "erpbero" dataset when using registers in different last $N_r$ rounds. When $N_r$ increases, the matching time will first decrease and then rise after $N_r$ is larger than a certain value. This is because using registers in the last few rounds will reduce memory access time, but if registers are used too early in the parallel reduction a lot of threads will be idle. We empirically set $N_r = 3$ to achieve the best performance in our experiments.

Fig.9(a) shows the number of matching points and matching time on the "erpbero" dataset for different $\tau$. Fig.9(b) gives the mean matching accuracy for different $\tau$ on "data-boat", "data-trees" and "data-ubc" (each group use 5 image pairs) of public Oxford dataset [34], which offers the Homography transformation as the ground truth. If the match pairs satisfy $x_1 - Hx_2 < \epsilon$, where $H$ is the given Homography matrix between image pairs and we set $\epsilon = 6$, we assume that the two keypoints compose accurate matching pairs. In Fig.9 when the threshold $\tau$ is about 40, the matching points don’t decrease and we can guarantee high matching accuracy and less matching time. Fig.10 shows the
analysis and comparison of time cost on ”Aos_Hus” and ”Data-Dubrovnik6K”
data set in each matching stage. The experiments show that the GPU acceler-
ated fundamental matrix estimation method is about 160 times faster than the
classical RANSAC algorithm on CPU. And the epipolar constraints can make
CasHashGPU faster.

Fig[11] shows the relationship between the number of GPU card and matching
speed. In the experiment, the number of GPU varies from 1 to 7. In Fig[11]
the matching speed is linear with the number of GPUs, so CasHashGPU can
linearly accelerate image matching on multi-GPU devices.

9. Conclusions

In this paper, we proposed a GPU accelerated image matching algorithm
with CasHash for fast feature matching of massive images. We proposed an
improved parallel reduction and an improved parallel hashing ranking to opti-
mize and parallelize the Cascade Hashing. For massive images, we proposed a
disk-memory-GPU data exchange strategy and we optimize the load order of
image data.
Figure 9. The influence of the threshold $\tau$. The influence of the threshold $\tau$ on the matching points and matching time is showed in (a). The influence of the threshold $\tau$ on the matching accuracy is showed in (b).
Figure 10. The diagram of the analysis and comparison of time cost on "Aos_Hus" and "Data-Dubrovnik6K" data set in each matching stage.
Figure 11. The relationship between the number of GPU card and matching speed. The experiment on Data-Dubrovnik6K time is showed in (a). The experiment on Data-Rome16K time is showed in (a).
References

[1] D. G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision 60 (2) (2004) 91–110. doi:10.1023/B:VISI.0000029664.99615.94

[2] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, R. Szeliski, Building Rome in a day, in: Proceedings of the IEEE International Conference on Computer Vision, 2009. doi:10.1109/ICCV.2009.5459148

[3] D. Crandall, A. Owen, N. Snavely, D. Huttenlocher, Discrete continuous optimization for large-scale structure from motion, Computer Vision and Pattern Recognition (2011) 3001–3008.

[4] J. Cheng, C. Leng, J. Wu, H. Cui, H. Lu, Fast and accurate image matching with cascade hashing for 3D reconstruction, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 2014, pp. 1–8. doi:10.1109/CVPR.2014.8

[5] M. S. Charikar, Similarity estimation techniques from rounding algorithms, in: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing - STOC '02, 2002. doi:10.1145/509961.509965

[6] Cornell 3D Location Recognition Datasets URL http://www.cs.cornell.edu/projects/p2f/

[7] S. Lazebnik, C. Schmid, J. Ponce, Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006. doi:10.1109/CVPR.2006.68

[8] D. Nistér, H. Stewénius, Scalable recognition with a vocabulary tree, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006. doi:10.1109/CVPR.2006.264
[9] T. Xu, K. Sun, W. Tao, Gpu accelerated image matching with cascade hashing, in: J. Yang, Q. Hu, M.-M. Cheng, L. Wang, Q. Liu, X. Bai, D. Meng (Eds.), Computer Vision, Springer Singapore, Singapore, 2017, pp. 91–101.

[10] D. Lowe, Object recognition from local scale-invariant features, in: Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999. doi:10.1109/ICCV.1999.790410

[11] Y. Ke, R. Sukthankar, Pca-sift: a more distinctive representation for local image descriptors, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004, pp. 506–513.

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

[13] H. Bay, T. Tuytelaars, L. Van Gool, Surf: speeded up robust features, European conference on computer vision 3951 (2006) 404–417.

[14] Y. Pang, W. Li, Y. Yuan, J. Pan, Fully affine invariant SURF for image matching, Neurocomputing doi:10.1016/j.neucom.2011.12.006

[15] M. Calonder, V. Lepetit, C. Strecha, P. Fua, BRIEF: Binary robust independent elementary features, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2010. doi:10.1007/978-3-642-15561-1_56

[16] S. Leutenegger, M. Chli, R. Y. Siegwart, BRISK: Binary Robust invariant scalable keypoints, in: Proceedings of the IEEE International Conference on Computer Vision, 2011. doi:10.1109/ICCV.2011.6126542

[17] A. Alahi, R. Ortiz, P. Vanderheyndt, FREAK: Fast retina keypoint, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2012. doi:10.1109/CVPR.2012.6247715
[18] M. Muja, D. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, 2009, pp. 331–340. doi:10.5220/0001787803310340

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2009. doi:10.2307/2583667

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

[21] F. Alhwarin, D. Ristić-Durrant, A. Gräser, VF-SIFT: Very fast SIFT feature matching, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2010. doi:10.1007/978-3-642-15986-2_23

[22] R. Raguram, C. Wu, J. M. Frahm, S. Lazebnik, Modeling and recognition of landmark image collections using iconic scene graphs, International Journal of Computer Vision doi:10.1007/s11263-011-0445-z.

[23] C. Wu, Towards Linear-Time Incremental Structure from Motion, in: 2013 International Conference on 3D Vision, IEEE, 2013, pp. 127–134. doi:10.1109/3DV.2013.25

[24] M. Havlena, K. Schindler, VocMatch: Efficient multiview correspondence for structure from motion, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2014. doi:10.1007/978-3-319-10578-9_4

[25] A. Lee, C. Yau, M. B. Giles, A. Doucet, C. C. Holmes, On the Utility of Graphics Cards to Perform Massively Parallel Simulation of Advanced Monte Carlo Methods, Journal of Computational and Graphical Statistics 19 (4) (2010) 769–789. doi:10.1198/jcgs.2010.10039
[26] R. Shah, V. Srivastava, P. J. Narayanan, Geometry-aware feature matching for structure from motion applications, Proceedings - 2015 IEEE Winter Conference on Applications of Computer Vision, WACV 2015 (2015) 278–285. doi:10.1109/WACV.2015.44

[27] S. Hong, H. Kim, An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness, ACM SIGARCH Computer Architecture News 37 (3) (2009) 152. doi:10.1145/1555815.1555775

[28] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with CUDA, in: ACM SIGGRAPH 2008 classes on - SIGGRAPH ’08, 2008. doi:10.1145/1401132.1401152

[29] C. Shane, CUDA Programming, 2013. doi:10.1016/C2011-0-00029-7

[30] P. Micikevicius, 3D finite difference computation on GPUs using CUDA, in: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units - GPGPU-2, 2009. doi:10.1145/1513895.1513905

[31] J. Sanders, E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming, Concurrency Computation Practice and Experience doi:10.1002/cpe.1472

[32] R. Shah, A. Deshpande, P. J. Narayanan, Multistage SFM: Revisiting Incremental Structure from Motion, in: 2014 2nd International Conference on 3D Vision, IEEE, 2014, pp. 417–424. doi:10.1109/3DV.2014.95

[33] SiftGPU, http://cs.unc.edu/~ccwu/siftgpu/

[34] K. Mikolajczyk, C. Schmid, A performance evaluation of local descriptors, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (10) (2005) 1615–1630. doi:10.1109/TPAMI.2005.188