Fast Organization of Large Photo Collections Using CUDA

Tim Johnson, Pierre Fite-Georgel, Rahul Raguram, and Jan-Michael Frahm

University of North Carolina at Chapel Hill, Department of Computer Science

Abstract. In this paper, we introduce a system for the automatic organization of photo collections consisting of millions of images downloaded from the Internet. To our knowledge, this is the first approach that tackles this problem exclusively through the use of general-purpose GPU computing techniques. By leveraging the inherent parallelism of the problem and through the use of efficient GPU-based algorithms, our system is able to effectively summarize datasets containing up to three million images in approximately 16 hours on a single PC, which is orders of magnitude faster compared to current state of the art techniques. In this paper, we present the various algorithmic considerations and design aspects of our system, and describe in detail the various steps of the processing pipeline. Additionally, we demonstrate the effectiveness of the system by showing results for a variety of real-world datasets, ranging from the scale of a single landmark, to that of an entire city.

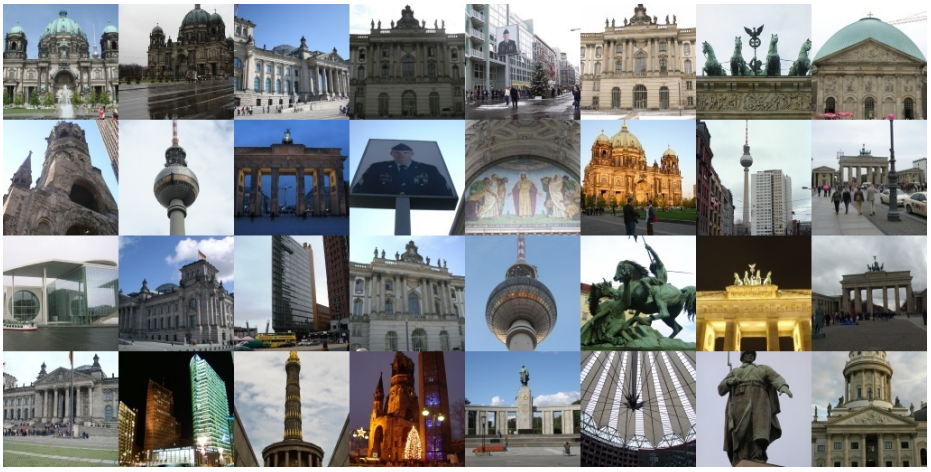

Fig. 1. A subset of the *iconic* images automatically found by our system, for the Berlin dataset

K.N. Kutulakos (Ed.): ECCV 2010 Workshops, Part II, LNCS 6554, pp. 463–476, 2012.

1 Introduction

Organizing Internet photo collections is an important task for many computer vision applications. For instance, partitioning a large set of photographs into clusters of similar images allows for more efficient post-processing tasks, such as structure from motion [1,2]. In addition, the organization of images into semantically consistent groups can also greatly enhance the browsing experience[1]. For the summarization strategy described in this paper, we define "similar" images as those which represent the same scene or landmark, taken from nearby vantage points, and under similar lighting conditions. Given these groupings of similar images, we then automatically extract a small subset of representative or *iconic* images that represent dominant aspects of the scene, and thus provide a concise visual summary of the dataset. This approach lends itself naturally to a hierarchical organization of the dataset into a form that is suitable both for 3D reconstruction as well as browsing.

With the ever-increasing abundance of images on the Internet, photo collections for a single search term now yield datasets on the order of millions of images – for example, a query for Rome on the photo sharing website Flickr yields approximately 3.4 million images. To operate on massive datasets of this form within a reasonable time-frame, it thus becomes essential to develop efficient algorithms that are capable of elegantly scaling to *Internet-scale* datasets. This is a particularly important consideration, given that the amount of digital information is predicted to increase exponentially in the years to come[2]. In this paper, we introduce an efficient method for the automatic organization of large scale photo collections ranging from several tens of thousands of images (the scale of a single landmark) to millions of images (representing an entire city). To our knowledge, this is the first system that runs completely on the GPU and scales to datasets on the order of millions of images.

In the following sections of the paper, we present previous work leading up to our approach (Section 2), followed by a high-level overview of our system (Section 3). We then provide an in-depth look at each step of the pipeline along with a discussion of important implementation details and design decisions (Section 4). The paper concludes with a presentation of results on three challenging real-world datasets (Section 5).

2 Previous Work

Organizing large scale photo collections has been of interest to many researchers in recent years [3,4,1,5,6,2]. The various approaches can be broadly classified into two categories: the first group uses two-view geometric constraints between images to determine similarity, while the second category uses appearance cues,

[1] For instance, tag clusters on Flickr: `http://www.flickr.com/photos/tags/berlin/clusters/`

[2] `http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf`

or constraints from object/scene recognition to measure image similarity. The notable departures from this classification are the approaches introduced in [1] and [2], which are hybrid approaches combining both appearance and geometric constraints to perform scene summarization, reconstruction and recognition. The method presented in this paper is similar in spirit to the approach of [1], but scales each of the techniques used to the true scale of internet photo collections, as in [2]. However, in addition to [2], we also propose a way to parallelize the computationally expensive two-view geometric verification step on the GPU, in addition to the appearance grouping steps.

The first work that performed 3D reconstruction of landmarks from internet photo collections containing a few thousand images was the *Photo Tourism* system [4]. This method yields high-quality reconstruction results with the help of exhaustive pairwise image matching and global bundle adjustment after inserting each new view. Both of these steps are computationally prohibitive on large scale datasets as the exhaustive matching grows exponentially and is no longer practical given contaminated real-world photo collections. To improve the performance of their system, Snavely et al. [7] find *skeletal sets* of images from the collection, whose reconstruction provides a good approximation to a reconstruction involving all the images. One remaining limitation of this work is that it still requires the exhaustive computation of all two-view relationships in the dataset as a prerequisite. Most recently, Agarwal et al. [5] address this computational challenge by using a computing cluster with up to 500 cores in order to process larger datasets. Similar to our system, the work in [5] uses image recognition techniques such as approximate nearest neighbor search [8] and query expansion [9] to reduce the number of candidate relations from the full exhaustive set. However, in contrast to Agarwal et al., our system uses a *single* PC, thereby achieving an even higher effective processing rate. The main source of the computational advantage we achieve is through the cascaded application of appearance and geometric constraints. By first using computationally cheaper 2D appearance cues, we identify consistent clusters of images that are likely to be spatially related. In turn, this leads to an overall decrease in the number of candidates to be considered for pairwise registration.

Our main goal in this work is to leverage the computing power of the GPU in order to develop a high performance system capable of efficiently organizing large image collections. Strong and Gong present such a system in [10,11], however our system differs in that it does not require a training step. Our system also scales to millions of images, whereas theirs scales to only thousands. Computation of gist vectors on the GPU was proposed in [12], however their results for large datasets were extrapolated from results on smaller datasets. Our paper presents results that were collected from actual runs on datasets of millions of images. To our knowledge, this is the first attempt that implements certain algorithms (binary code generation, RANSAC) on the GPU. In addition to these, since we operate on binary vectors, we have also developed a k-medoids implementation for the GPU, as an alternative to existing k-means implementations [13,14]. Finally, for

the geometric verification step, we make use of SiftGPU, a publicly available GPU implementation of SIFT extraction and matching[3].

3 System Overview

In this section, we present a brief description of the main components of our system. At a high-level, the system operates in a hierarchical manner, using both 2D image appearance cues as well as 3D scene geometry in order to solve the task of partitioning images from large Internet photo collections into clusters of similar photographs. The input to the system is a raw Internet photo collection, downloaded using keyword searches on Flickr. It has been observed [15] that these collections can often be significantly contaminated, with a substantial fraction of the images being semantically unrelated to the query. Thus, our system has been designed with a view towards being robust to the significant amount of clutter present in community photo collections. The downloaded images are then subjected to several algorithms in a pipeline, first enforcing a loose grouping of the images based on 2D appearance cues, and subsequently refining the initial partitioning using stricter 3D constraints. This procedure allows for the efficient processing of very large datasets, since the more computationally demanding geometric verification steps are carried out only on smaller subsets of images that are already grouped together by similarity. The end result is a clustering where each image within a cluster is similar in geometric structure, vantage point, and color. The main steps in the pipeline are outlined below.

- **Global Descriptor Extraction** (Sec. 4.1): In order to cluster images based on similarity, we first need to compute feature vectors for each image. We choose to compute gist descriptors [16] for each image, which has been shown to effectively capture perceptual similarity and has been used to retrieve structurally similar scenes [17]. We combine the gist descriptor with low resolution color descriptions of the image to produce a 368 dimensional feature vector for each image.
- **Conversion to Binary Codes** (Sec. 4.2): To efficiently store the feature vectors on the GPU, we compress them into binary codes. The compressed representation allows all of the data to be simultaneously stored on the GPU, which significantly reduces the amount of data transfer necessary between device and host during the clustering step.
- **Clustering on the Binary Codes** (Sec. 4.3): Given a set of compressed descriptor vectors, we cluster the dataset with a parallel implementation of the k-medoids algorithm [18]. This provides a rough grouping of the dataset into clusters that are similar in global appearance. In addition, this step also filters out a large fraction of unrelated images, since these fall into small and isolated clusters.
- **Geometric Verification** (Sec. 4.4): Once a loose grouping of the images has been obtained, strict geometric constraints are enforced to ensure that

[3] `http://www.cs.unc.edu/~ccwu/siftgpu`

the images within a cluster represent the same scene or structure. This step requires the extraction of SIFT features [19], followed by a RANSAC [20] procedure for estimating the pairwise epipolar geometry [21,22]. Geometrically consistent images, or images that capture the same 3D structure, are retained, while the others are discarded. This produces clean clusters that are consistent both in terms of appearance as well as geometry.

Each of the above steps is implemented in CUDA and runs on one or multiple GPUs. We also plan to make these implementations freely available on the web, for use by the community.

4 Image Organization for Internet Photo Collections

In this section we introduce the details of our proposed processing pipeline. Along with the algorithmic considerations we will discuss design decisions to ensure a efficient use of the highly parallel GPU architecture. In particular we explore the considerations made in regards to memory access patterns, maximizing hardware utilization and minimizing I/O between the host and GPU.

4.1 Global Descriptor Extraction

To boost computational efficiency, we aim at compactly describing each image with a single global image descriptor. We choose the powerful gist descriptor proposed by Oliva and Torralba [16], which was shown to achieve good results for the tasks of scene matching and retrieval [17]. The gist vector is a description of the oriented edges in an image. It is an aggregation of image convolutions that have been downsampled to a resolution of 4x4. Each convolution picks up edge responses at a certain orientation and scale. The convolution kernel used in our implementation is the Gabor filter. In our implementation, we perform a total of 20 convolutions at three different scales. The images we convolve are greyscale thumbnails with a resolution of 128x128, and are cropped accordingly to preserve their original aspect ratio. Performing these operations leads to a 320-dimensional gist vector. It is also typical to augment the gist descriptor with color information, usually by appending an additional vector that carries color information. In contrast to the downsampled L*a*b color space used to incorporate colour information as in [17], our method directly uses the RGB representation of the image. For the task of organizing image collections, we empirically found the two representations to perform comparably. Thus, a 4×4 RGB representation is then appended to the gist feature vector. The final descriptor has 368 dimensions, stored as floats.

Computationally, the convolution with the Gabor filter and the downsampling of the images are the most demanding tasks in this step. We improve GPU utilization during the convolution step by efficiently processing the images in batches. Since each of the input images is only of size 128×128, we combine the processing of multiple images to achieve greater occupancy of the GPU. Images

are convolved in batches of 256 (determined empirically, we measure no gain with bigger batch sizes) to reduce the number of memory transfers from host to GPU. The images are laid out in row major order, one after the other and are passed to the CUDA kernel for convolution.

The CUDA kernel for convolution assigns 16x16 thread blocks to compute the convolution of some 16x16 patch of the virtual image, and each thread within a block computes one pixel of the convolved image. Prior to dispatching the CUDA kernel, the convolution kernel is transferred into a constant memory buffer. We chose to use constant memory for two main reasons. First, constant memory is cached, so reads from the buffer will be fast. Secondly, constant memory is useful for memory accesses when each thread in a half-warp accesses the same index. In this case, each thread reads from the same location in the convolution kernel simultaneously, providing a slight advantage over textured memory.

Each thread block loads its corresponding image patch into shared memory. It also needs to load in a portion of the image that borders this patch, and the size of this portion is dependent on the size of the convolution kernel. All of these memory loads are coalesced by carefully controlling which threads load a particular part of the image. Once all necessary image data is loaded into shared memory, the patches are convolved with a standard double nested loop, and the sum is written out to global memory.

Before the next convolution is performed, the current convolved images are downsampled to reduce the amount of storage necessary. Downsampling is performed on the virtual image, and care is taken to ensure that sub-image boundaries within the virtual image are preserved. Downsampling is performed in two one-dimensional passes over the virtual image. The thread blocks, accordingly, are 1-dimensional. Since each thread block outputs one downsampled pixel, the size of each thread block is determined by the level of downsampling.

The downsampling kernel essentially performs a sum reduction along a contiguous portion of the image, with a final division to achieve an average value over its portion of the image. Each thread block outputs its result to global memory buffer in a transposed fashion, so that the next pass can read in the image in a set of coalesced memory accesses. After two passes the virtual image has been downsampled in both the x and y directions, and the result remains in the row-major storage format. See Figure 2 for a pictorial explanation.

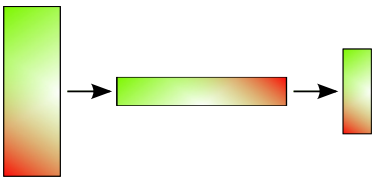

Fig. 2. Visualization of the downsampling process. It is performed in two one-dimensional passes with transpositions after each pass.

Intuitively, the extracted gist descriptors describe the viewpoint, through the edge structure, and the illumination, by means of the subsampled image, of the scene. These gist vectors can thus be leveraged to roughly group together similar images, as outlined in subsequent sections.

4.2 Conversion to Binary Codes

To group together similar images, we would like to perform a clustering procedure on the gist vectors. However, there are some important memory limitations that must be overcome in order to achieve good performance. In order to achieve efficient clustering on the GPU, we require that all feature vectors fit into the GPU memory. Given today's GPU memory limitations, we could only fit about 600K to 700K of the 368 dimensional descriptors into the GPU memory, which is significantly smaller than the dataset sizes that we seek to operate on. Processing the dataset in batches would require a large number of transfers between host and device for each iteration of the clustering algorithm, thus leading to significant overhead. To overcome this limitation, we compress each feature vector into a string of binary numbers, known as a binary code. The particular compression technique we implemented is based on the method of [23]. Since we would like to retain the appearance relationships of the gist vectors, these vectors are compressed in such a way that the Hamming distance between the resulting binary codes approximates the Euclidean distance between the original feature descriptors. Each bit of the code describes on which side of a randomly generated hyper-plane the original GIST descriptor is located. This method of compression has been compared to a simpler locality sensitive hashing method in [2], and it has been shown to preserve feature vector distances sufficiently. The ability to choose the number of bits in our binary codes provides flexibility for the clustering step. We may decrease the number of bits if we wish to manage GPU memory more conservatively, or we may increase the number of bits if we wish to approximate the feature vectors more accurately. To ensure high computational performance, our technique employs the highly optimized CUBLAS library, with the exception of a kernel for converting float vectors into binary strings. This kernel dispatches 1-dimensional thread blocks of length 32, and each thread block produces one unsigned integer of output by performing a bitwise OR reduction on 32 bits. The reduction represents the sgn of 32 floats. For Berlin, using a 512-bit binary code scheme reduces the storage requirement of the features from 3.7GB to 164MB.

4.3 Clustering

Once the binary codes have been generated we can cluster them using a parallel implementation of *k-medoids*. Similar to k-means, the standard k-medoids algorithm [18] takes n features as input, and outputs k clusters. It differs from k-means in that the cluster centers are the most central data elements of the respective clusters, instead of the mean of the cluster center. This accommodates our binary representation of the image descriptors for which the mean is

not meaningful. Since the Hamming distance of our binary codes approximates the Euclidean distance of the original gist descriptors, our k-medoids implementation uses this as the distance metric. K-medoids consists of iterations of an assignment step and update step. It is initialized by randomly selecting k distinct binary codes as cluster centers, or *medoids*. During the assignment step, each binary code is associated with the closest medoid by Hamming distance. In the update step, the binary code that minimizes the sum of distances to all other codes in its cluster becomes the new medoid center. We define convergence as the number of medoid changes falling below a defined threshold (we use $0.01k$).

The bottleneck of the k-medoids algorithm is the computation of the Hamming distance matrices for all clusters. Distance matrices are computed in both the assignment and the update stage. In the assignment stage, an $n \times k$ matrix is computed. In the update step, k smaller matrices are computed, one for each cluster. The dimension of each matrix is square and equal to the number of elements in that cluster. Fortunately, this computation is highly parallelizable. Our kernel for computing the distance matrix dispatches as many 16x16 thread blocks as needed to cover the full distance matrix. Each thread computes one entry of the distance matrix, and does so by processing 32 bits of the binary codes at a time. This way, each thread block only requires 128 bytes of shared memory at any given time. An overview of the clustering can be found in Algorithm 1.

Algorithm 1. K-Medoids

```
for i=1 to k do
  randomly assign medoid[i] to a binary code
end for
repeat
  for i=1 to n do
    compute distance of ith binary code to medoids in parallel
    do parallel min-reduce to assign binary code i to closest medoid
  end for
  for i=1 to k do
    compute distance matrix between all elements of cluster i
    do parallel sum-reduce over rows of distance matrix
    do parallel min-reduce of result to find new cluster center
  end for
until converged
```

4.4 Geometric Verification

The initial clusters provided by k-medoids may contain still images which are close in the compressed gist space, but they still may be visually or geometrically inconsistent as shown in Figure 3. Given that the desired output of our system only consists of clusters of images which have captured a consistent geometrical scene structure, we perform a final step to remove inconsistent images. This is performed by selecting the first r images of each cluster (the medoid and the images closest to it) and estimating the epipolar geometry of each image pair

within those r images. If any image has less than ρ inliers (we use $\rho = 18$ in all our experiments), they are replaced with the next closest image. Similarly, to [1] this process is repeated until r consistent images are found, or the cluster is rejected. In order to prevent extensive computation for large but inconsistent clusters, we reject a cluster if no consistent set has been found after $3r$ different images have been tested. When a cluster has been verified, the image with the most inliers over the set of r images is declared as the most representative view: the *iconic*. Afterwards, all remaining images are verified against the chosen iconic.

To compute the two-view geometry, we first extract SIFT [19] features using the efficient CUDA SiftGPU implementation. We limit the maximum number of extracted features to 4000 in the interest of computational efficiency. Following this, we compute pairwise putative matches using the CUBLAS library to perform fast matrix multiplication, followed by a distance ratio test to identify likely correspondences. The putative matches are then verified by estimating the fundamental matrix using the 7-point algorithm [24] in a RANSAC framework [20], both of which have been implemented in our system, using CUDA.

Algorithm 2. CUDA QR Decomposition Kernel

```
{Given A, compute matrices Q and R such that A = Q*R}
shared float *sR, *sQ
load A into sR, sQ = I
for k=1 to min(rows-1, cols) do
  compute kth Householder reflector in serial
  apply reflector to sR, sQ in parallel
end for
write sR, sQ to global memory
```

Due to the randomized nature of memory access patterns in RANSAC, implementing RANSAC efficiently on the GPU presents the challenge of achieving coalesced memory accesses. That is, nearby threads access nearby locations in memory. To overcome this, we push N random samples of 7 points onto the GPU, where N is the maximum number of iterations of RANSAC, set to 1024 in our experiments. This way, coalesced reads from memory can still reflect randomized reads of the data. These randomized reads of the data are used as input to the 7-point fundamental matrix estimator.

Estimating the fundamental matrix requires finding solutions to the fundamental matrix constraint $x'^T F x = 0$, where x and x' represent corresponding points across two views. At least seven of these constraints are required to solve for the fundamental matrix [24], so we randomly select 7 correspondences, which define a system of equations. The null space of this system of equations defines the fundamental matrix. To find the null space, we develop a QR decomposition algorithm in CUDA using Householder reflections. The pseudocode for our algorithm is shown in Algorithm 2. Since each thread block decomposes a separate matrix, our CUDA kernel only works for small matrices. This is not a

Fig. 3. An example cluster output from k-medoids on the Tower Bridge dataset. Note the inconsistent images near the bottom.

problem, as the system of linear equations is represented by a 7x9 matrix. The QR decomposition works by solving for one column of R at a time, and multiple columns cannot be solved simultaneously. However, all elements in one column can be solved simultaneously. Multiple blocks can be dispatched simultaneously, allowing for fast QR decompositions of multiple matrices.

Once the fundamental matrices have been generated, they must be evaluated against the entire set of correspondences. This data is stored on a GPU buffer, and we test the Sampson distance for each correspondence against a predefined threshold to identify inliers. We use the adaptive stopping criterion, where the number of iterations is updated based on the highest inlier ratio observed so far.

5 Results

In this section, we present results on three challenging real-world datasets. The experiments were run on a 2 Intel Xeon processor machine (8 available cores), with 50GB RAM and 4 Nvidia GTX 295 GPUs (8 GPU cores). The sift extraction, gist computation and RANSAC modules utilize all 8 GPUs, while the binary compression and clustering steps use 1 GPU, since multiple GPUs would require significant I/O between devices (eg., in the computation of the distance matrix). The number of clusters in k-medoids was chosen to be 10% of the dataset size, capped at a maximum of 100,000 centers.

The three datasets presented in this paper – Notre Dame (90,196 images), Tower Bridge (137,073 images), and Berlin (2,704,448 images) – were downloaded using keyword searches on Flickr. Figure 3 shows an example cluster output from k-medoids for the Tower Bridge dataset. Note that at this stage, only appearance-based cues have been employed. While the cluster demonstrates a appreciable degree of visual similarity, there exist incorrect images that are consistent in appearance, but that do not depict the same scene. Enforcing tighter geometric constraints helps "clean-up" these clusters, as shown in Figure 4.

Fig. 4. The same cluster as in Fig. 3 after it has passed through geometric verification. Note how the inconsistent images have been removed.

Figures 5 shows a subset of 120 iconic images for the Notre Dame dataset. It can be seen that this provides a concise summary of the "popular" aspects and viewpoints of the landmark. These iconic images can then be used to seed a structure-from-motion system, since they capture a variety of different camera locations covering the scene. Thus, the process of 3D reconstruction may be initialized using just a small, representative subset of the dataset, thus allowing for the processing of massive image collections. In addition, the iconic images can be used as the top level of a hierarchical browsing system, where each iconic may in turn be expanded to show all the images within the corresponding cluster, which are very similar in appearance and geometry to the iconic of interest. If desired, an additional level in the browsing hierarchy may be formed by grouping together iconic images into related "components" as in [1], thereby providing a three-level organization of the image collection.

To demonstrate some of the advantages of a GPU-based approach, we compared the performance of the proposed GPU RANSAC algorithm versus a very high performance real-time robust estimation technique called ARRSAC [25]. For this experiment, 50 random clusters were selected from the Berlin dataset and geometric verification was performed as outlined in Section 4.4. The results are tabulated in Table 1 for varying numbers of CPU and GPU cores. It can be seen from the table that the use of GPU-RANSAC for the geometric verification step results in a 2-8% improvement in speed, compared to ARRSAC. While this is not a large speedup, it should be noted that ARRSAC is a highly optimized framework, whereas our brute force implementation leaves much room for optimization.

Table 2 lists summary statistics for the complete pipeline operating on all three datasets. The table lists the number of iconics found by our system for each dataset, and it can be seen that these large datasets are efficiently reduced to a small, representative set of iconic images. The table also lists the number of geometrically consistent images that remain in the clusters following the robust geometric verification step of the processing pipeline. On average, roughly

Fig. 5. Subset of 120 iconic images for the Notre Dame dataset. The iconics denote dominant aspects of the scene or landmark, thus providing a concise but representative visual summary of the dataset.

5-35% of the images in each dataset are retained at this stage, though it must be noted this fraction can be increased through additional stages that attempt to match discarded images across different iconic clusters. Finally, Table 2 also lists runtimes for each stage of the pipeline. It can be seen that our GPU-based pipeline is able to process more than two million images in approximately 16

Table 1. Geometric verification performance: ARRSAC vs. our GPU-RANSAC. Timings were collected for different numbers of CPU and GPU cores used simultaneously.

Number of CPU/GPU cores	Geometric verification timing (seconds)	
	ARRSAC	GPU-RANSAC
1/1	155.38	152.40
4/4	38.5802	36.8281
8/8	28.299	25.892

Table 2. Summary statistics and timings for each processing step in the pipeline

Dataset	# Iconics	# Images Registered	Gist	Binary Code	Clustering	Geom Verif.
Notre Dame	3,566	27,496	87s	0.79s	20.4s	42min 8s
Tower Bridge	5,479	47,146	138s	1.29s	26.2s	59min 46s
Berlin	13,612	133,634	1hr 1min	28.9s	30min 46s	14hr 27min

hours, on a single PC equipped with graphics hardware. This represents an order of magnitude more data than current state of the art techniques [5].

6 Conclusion

This paper presents a high-performance GPU-based system for organizing large photo collections. The system employs recognition constraints along with 3D geometry, and exploits the underlying parallelism of the organization problem. The system is entirely implemented on the GPU in CUDA, and is capable of efficiently organizing massive image collections, containing millions of images, on a single computer while still producing high-quality results. To our knowledge, this is the first system that uses GPGPU computation to achieve the image organization task.

References

1. Li, X., Wu, C., Zach, C., Lazebnik, S., Frahm, J.-M.: Modeling and Recognition of Landmark Image Collections Using Iconic Scene Graphs. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part I. LNCS, vol. 5302, pp. 427–440. Springer, Heidelberg (2008)
2. 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: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part IV. LNCS, vol. 6314, pp. 368–381. Springer, Heidelberg (2010)
3. Schaffalitzky, F., Zisserman, A.: Multi-view Matching for Unordered Image Sets, or How Do I Organize My Holiday Snaps? In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) ECCV 2002, Part I. LNCS, vol. 2350, pp. 414–431. Springer, Heidelberg (2002)
4. Snavely, N., Seitz, S.M., Szeliski, R.: Modeling the world from Internet photo collections. International Journal of Computer Vision 80, 189–210 (2008)

5. Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building Rome in a day. In: ICCV (2009)
6. Berg, T.L., Berg, A.C.: Finding iconic images. In: The 2nd Internet Vision Workshop at IEEE CVPR (2009)
7. Snavely, N., Seitz, S.M., Szeliski, R.: Skeletal sets for efficient structure from motion. In: CVPR (2008)
8. Arya, S., Mount, D., Netanyahu, N., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. JACM 45, 891–923 (1998)
9. 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)
10. Strong, G., Gong, M.: Browsing a Large Collection of Community Photos Based on Similarity on GPU. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Remagnino, P., Porikli, F., Peters, J., Klosowski, J., Arns, L., Chun, Y.K., Rhyne, T.-M., Monroe, L. (eds.) ISVC 2008, Part II. LNCS, vol. 5359, pp. 390–399. Springer, Heidelberg (2008)
11. Strong, G., Gong, M.: Organizing and browsing photos using different feature vectors and their evaluations. In: CIVR, pp. 1–8 (2009)
12. Wang, Y., Feng, Z., Guo, H., He, C., Yang, Y.: Scene recognition acceleration using cuda and openmp. In: ICISE, pp. 1422–1425 (2009)
13. Shalom, S.A.A., Dash, M., Tue, M.: Efficient *K*-Means Clustering Using Accelerated Graphics Processors. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2008. LNCS, vol. 5182, pp. 166–175. Springer, Heidelberg (2008)
14. Hall, J.D., Hart, J.C.: Abstract gpu acceleration of iterative clustering (2004)
15. Kennedy, L., Chang, S.F., Kozintsev, I.: To search or to label?: Predicting the performance of search-based automatic image classifiers. ACM MIR (2006)
16. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. IJCV 42, 145–175 (2001)
17. Hays, J., Efros, A.A.: Scene completion using millions of photographs. SIGGRAPH (2007)
18. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data An Introduction to Cluster Analysis. Wiley Interscience, New York (1990)
19. Lowe, D.: Distinctive image features from scale-invariant keypoints. IJCV 60, 91–110 (2004)
20. Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24 (1981)
21. Beardsley, P., Zisserman, A., Murray, D.: Sequential updating of projective and affine structure from motion. Int. J. Computer Vision 23, 235–259 (1997)
22. Nistér, D., Naroditsky, O., Bergen, J.: Visual odometry for ground vehicle applications. Journal of Field Robotics 23 (2006)
23. Raginsky, M., Lazebnik, S.: Locality-sensitive binary codes from shift-invariant kernels. NIPS 22, 1509–1517 (2009)
24. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press (2004) ISBN: 0521540518
25. Raguram, R., Frahm, J.-M., Pollefeys, M.: A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part II. LNCS, vol. 5303, pp. 500–513. Springer, Heidelberg (2008)