Building Rome in a Day

Sameer Agarwal[1,*] Noah Snavely[2] Ian Simon[1] Steven M. Seitz[1] Richard Szeliski[3]

[1]University of Washington [2]Cornell University [3]Microsoft Research

Abstract

We present a system that can match and reconstruct 3D scenes from extremely large collections of photographs such as those found by searching for a given city (e.g., Rome) on Internet photo sharing sites. Our system uses a collection of novel parallel distributed matching and reconstruction algorithms, designed to maximize parallelism at each stage in the pipeline and minimize serialization bottlenecks. It is designed to scale gracefully with both the size of the problem and the amount of available computation. We have experimented with a variety of alternative algorithms at each stage of the pipeline and report on which ones work best in a parallel computing environment. Our experimental results demonstrate that it is now possible to reconstruct cities consisting of 150K images in less than a day on a cluster with 500 compute cores.

1. Introduction

Entering the search term "Rome" on `flickr.com` returns more than two million photographs. This collection represents an increasingly complete photographic record of the city, capturing every popular site, facade, interior, fountain, sculpture, painting, cafe, and so forth. Most of these photographs are captured from hundreds or thousands of viewpoints and illumination conditions—Trevi Fountain alone has over 50,000 photographs on Flickr. Exciting progress has been made on reconstructing individual buildings or plazas from similar collections [16, 17, 8], showing the potential of applying structure from motion (SfM) algorithms on unstructured photo collections of up to a few thousand photographs. This paper presents the first system capable of *city-scale* reconstruction from unstructured photo collections. We present models that are one to two orders of magnitude larger than the next largest results reported in the literature. Furthermore, our system enables the reconstruction of data sets of 150,000 images in less than a day.

*To whom correspondence should be addressed. Email: `sagarwal@cs.washington.edu`

City-scale 3D reconstruction has been explored previously in the computer vision literature [12, 2, 6, 21] and is now widely deployed e.g., in Google Earth and Microsoft's Virtual Earth. However, existing large scale structure from motion systems operate on data that comes from a structured source, *e.g.*, aerial photographs taken by a survey aircraft or street side imagery captured by a moving vehicle. These systems rely on photographs captured using the same calibrated camera(s) at a regular sampling rate and typically leverage other sensors such as GPS and Inertial Navigation Units, vastly simplifying the computation.

Images harvested from the web have none of these simplifying characteristics. They are taken from a variety of different cameras, in varying illumination conditions, have little to no geographic information associated with them, and in many cases come with no camera calibration information. The same variability that makes these photo collections so hard to work with for the purposes of SfM also makes them an extremely rich source of information about the world. In particular, they specifically capture things that people find *interesting*, i.e., worthy of photographing, and include interiors and artifacts (sculptures, paintings, etc.) as well as exteriors [14]. While reconstructions generated from such collections do not capture a *complete* covering of scene surfaces, the coverage improves over time, and can be complemented by adding aerial or street-side images.

The key design goal of our system is to quickly produce reconstructions by leveraging massive parallelism. This choice is motivated by the increasing prevalence of parallel compute resources both at the CPU level (multi-core) and the network level (cloud computing). At today's prices, for example, you can rent 1000 nodes of a cluster for 24 hours for $10,000 [1].

The cornerstone of our approach is a new system for large-scale distributed computer vision problems, which we will be releasing to the community. Our pipeline draws largely from the existing state of the art of large scale matching and SfM algorithms, including SIFT, vocabulary trees, bundle adjustment, and other known techniques. For each stage in our pipeline, we consider several alternatives, as some

algorithms naturally distribute and some do not, and issues like memory usage and I/O bandwidth become critical. In cases such as bundle adjustment, where we find that the existing implementations did not scale, we have created our own high performance implementations. Designing a truly scalable system is challenging, and we discovered many surprises along the way. The main contributions of our paper, therefore, are the insights and lessons learned, as well as technical innovations that we invented to improve the parallelism and throughput of our system.

The rest of the paper is organized as follows. Section 2 discusses the detailed design choices and implementation of our system. Section 3 reports the results of our experiments on three city scale data sets, and Section 4 concludes with a discussion of some of the lessons learned and directions for future work.

2. System Design

Our system runs on a cluster of computers (nodes), with one node designated as the master node. The master node is responsible for the various job scheduling decisions.

In this section, we describe the detailed design of our system, which can naturally be broken up into three distinct phases: (1) pre-processing §2.1, (2) matching §2.2, and (3) geometric estimation §2.4.

2.1. Preprocessing and feature extraction

We assume that the images are available on a central store, from which they are distributed to the cluster nodes on demand in chunks of fixed size. This automatically performs load balancing, with more powerful nodes receiving more images to process.

This is the only stage where a central file server required; the rest of the system operates without using any shared storage. This is done so that we can download the images from the Internet independent of our matching and reconstruction experiments. For production use, it would be straightforward to have each cluster node to crawl the Internet for images at the start of the matching and reconstruction process.

On each node, we begin by verifying that the image files are readable, valid images. We then extract the EXIF tags, if present, and record the focal length. We also downsample images larger than 2 Mega-pixels, preserving their aspect ratios and scaling their focal lengths. The images are then converted to grayscale and SIFT features are extracted from them [10]. We use the SIFT++ implementation by Andrea Vedaldi for its speed and flexible interface [20]. At the end of this stage, the entire set of images is partitioned into disjoint sets, one for each node. Each node owns the images and SIFT features associated with its partition.

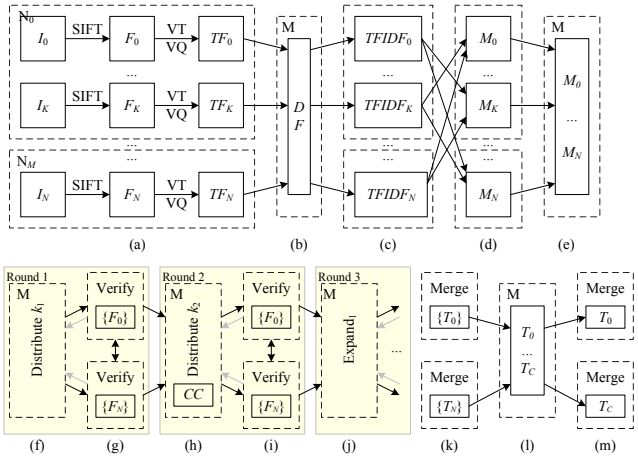

Figure 1. Our multi-stage parallel matching pipeline. The N input images are distributed onto M processing nodes, after which the following processing stages are performed: (a) SIFT feature extraction, vocabulary tree vector quantization, and term frequency counting; (b) document frequency counting; (c) TFIDF computation and information broadcast; (d) computation of TF-based match likelihoods; (e) aggregation at the master node; (f) round-robin bin-packing distribution of match verification tasks based on the top k_1 matches per image; (g) match verification with optional inter-node feature vector exchange; (h) match proposal expansion based on images found in connected components CC using the next k_2 best matches per image; (i) more distributed verification; (j) four more rounds of query expansion and verification; (k) track merging based on local verified matches; (l) track aggregation by into C connected components; (m) final distribution of tracks by image connected components and distributed merging.

2.2. Image Matching

The key computational tasks when matching two images are the photometric matching of interest points and the geometric verification of these matches using a robust estimate of the fundamental or essential matrix. While exhaustive matching of all features between two images is prohibitively expensive, excellent results have been reported with approximate nearest neighbor search. We use the ANN library [3] for matching SIFT features. For each pair of images, the features of one image are inserted into a k-d tree, and the features from the other image are used as queries. For each query, we consider the two nearest neighbors, and matches that pass Lowe's ratio test are accepted [10]. We use the priority queue based search method, with an upper bound on the maximum number of bin visits of 200. In our experience, these parameters offer a good tradeoff between computational cost and matching accuracy. The matches returned by the approximate nearest neighbor search are then pruned and verified using a RANSAC-based estimation of the fundamental or essential matrix [18], depending on the availability of focal length information from the EXIF tags.

These two operations form the computational kernel of our matching engine.

Unfortunately, even with a well optimized implementation of the matching procedure described above, it is not practical to match all pairs of images in our corpus. For a corpus of 100,000 images, this translates into 5,000,000,000 pairwise comparisons, which with 500 cores operating at 10 image pairs per second per core would require about 11.5 days to match. Furthermore, this does not even take into account the network transfers required for all cores to have access to all the SIFT feature data for all images.

Even if we were able to do all these pairwise matches, it would be a waste of computational effort, since an overwhelming majority of the image pairs do not match. This is expected from a set of images associated with a broad tag like the name of a city. Thus, we must be careful in choosing the image pairs that the system spends its time matching.

Building upon recent work on efficient object retrieval [15, 11, 5, 13], we use a multi-stage matching scheme. Each stage consists of a proposal and a verification step. In the proposal step, the system determines a set of image pairs that it expects to share common scene elements. In the verification step, detailed feature matching is performed on these image pairs. The matches obtained in this manner then inform the next proposal step.

In our system, we use two methods to generate proposals: vocabulary tree based whole image similarity §2.2.1 and query expansion §2.2.4. The verification stage is described in §2.2.2. Figure 1 shows the system diagram for the entire matching pipeline.

2.2.1 Vocabulary Tree Proposals

Methods inspired by text retrieval have been applied with great success to the problem of object and image retrieval. These methods are based on representing an image as a bag of words, where the words are obtained by quantizing the image features. We use a vocabulary tree-based approach [11], where a hierarchical k-means tree is used to quantize the feature descriptors. (See §3 for details on how we build the vocabulary tree using a small corpus of training images.) These quantizations are aggregated over all features in an image to obtain a term frequency (TF) vector for the image, and a document frequency (DF) vector for the corpus of images (Figure 1a–e). The document frequency vectors are gathered across nodes into a single vector that is broadcast across the cluster. Each node normalizes the term frequency vectors it owns to obtain the TFIDF matrix for that node. These per-node TFIDF matrices are broadcast across the network, so that each node can calculate the inner product between its TFIDF vectors and the rest of the TFIDF vectors. In effect, this is a distributed product of the matrix of TFIDF vectors with itself, but each node only calculates the block of rows corresponding to the set of images it owns. For each image, the top scoring $k_1 + k_2$ images are identified, where the first k_1 images are used in an initial verification stage, and the additional k_2 images are used to enlarge the connected components (see below).

Our system differs from that of Nister and Stewenius [11], since their system has a fixed database to which they match incoming images. They can therefore store the database in the vocabulary tree itself and evaluate the match score of an image on the fly. In our case, the query set is the same as the database, and it is not available when the features are being encoded. Thus, we must have a separate matrix multiplication stage to find the best matching images.

2.2.2 Verification and detailed matching

The next step is to verify candidate image matches, and to then find a detailed set of matches between matching images. If the images were all located on a single machine, the task of verifying a proposed matching image pair would be a simple matter of running through the image pairs, perhaps with some attention paid to the order in which the verifications are performed so as to minimize disk I/O. However, in our case, the images and their feature descriptors are distributed across the cluster. Thus, asking a node to match the image pair (i, j) require it to fetch the image features from two other nodes of the cluster. This is undesirable, as there is a large difference between network transfer speeds and local disk transfers. Furthermore, this creates work for three nodes. Thus, the image pairs should be distributed across the network in a manner that respects the locality of the data and minimizes the amount of network transfers (Figure 1f–g).

We experimented with a number of approaches with some surprising results. We initially tried to optimize network transfers before any verification is done. In this setup, once the master node has all the image pairs that need to be verified, it builds a graph connecting image pairs which share an image. Using `MeTiS` [7], this graph is partitioned into as many pieces as there are compute nodes. Partitions are then matched to the compute nodes by solving a linear assignment problem that minimizes the number of network transfers needed to send the required files to each node.

This algorithm worked well for small problem sizes, but as the problem size increased, its performance became degraded. Our assumption that detailed matches between all pairs of images take the same constant amount of time was wrong: some nodes finished early and were idling for up to an hour.

The second idea we tried was to over-partition the graph into small pieces, and to parcel them out to the cluster nodes on demand. When a node requests another chunk of work, the piece with the fewest network transfers is assigned to it. This strategy achieved better load balancing, but as the size

of the problem grew, the graph we needed to partition grew to be enormous, and partitioning itself became a bottleneck.

The approach that gave the best results was to use a simple greedy bin-packing algorithm (where each bin represents the set of jobs sent to a node), which works as follows. The master node maintains a list of images on each node. When a node asks for work, it runs through the list of available image pairs, adding them to the bin if they do not require any network transfers, until either the bin is full or there are no more image pairs to add. It then chooses an image (list of feature vectors) to transfer to the node, selecting the image that will allow it to add the maximum number of image pairs to the bin, This process is repeated until the bin is full. This algorithm has one drawback: it can require multiple sweeps over all the image pairs needing to be matched. For large problems, the scheduling of jobs can become a bottleneck. A simple solution is to only consider a subset of the jobs at a time, instead of trying to optimize globally. This windowed approach works very well in practice, and all our experiments are run with this method.

Verifying an image pair is a two-step procedure, consisting of photometric matching between feature descriptors, and a robust estimation of the essential or fundamental matrix depending upon the availability of camera calibration information. In cases where the estimation of the essential matrix succeeds, there is a sufficient angle between the viewing directions of the two cameras, and the number of matches are above a threshold, we do a full Euclidean two-view reconstruction and store it. This information is used in later stages (see §2.4) to reduce the size of the reconstruction problem.

2.2.3 Merging Connected Components

At this stage, consider a graph on the set of images with edges connecting two images if matching features were found between them. We refer to this as the *match graph*. To get as comprehensive a reconstruction as possible, we want the fewest number of connected components in this graph. To this end, we make further use of the proposals from the vocabulary tree to try and connect the various connected components in this graph. For each image, we consider the next k_2 images suggested by the vocabulary tree. From these, we verify those image pairs which straddle two different connected components (Figure 1h–i). We do this only for images which are in components of size 2 or more. Thus, images which did not match any of their top k_1 proposed matches are effectively discarded. Again, the resulting image pairs are subject to detailed feature matching. Figure 2 illustrates this. Notice that after the first round, the match graph has two connected components, which get connected after the second round of matching.

2.2.4 Query Expansion

After performing two rounds of matching as described above, we have a match graph which is usually not dense enough to reliably produce a good reconstruction. To remedy this, we use another idea from text and document retrieval research – query expansion [5].

In its simplest form, query expansion is done by first finding the documents that match a user's query, and then using them to query the database again, thus *expanding* the initial query. The results returned by the system are some combination of these two queries. In essence, if we were to define a graph on the set of documents, with similar documents connected by an edge, and treat the query as a document too, then query expansion is equivalent to finding all vertices that are within two steps of the query vertex.

In our system, we consider the image match graph, where images i and j are connected if they have a certain minimum number of features in common. Now, if image i is connected to image j and image j is connected to image k, we perform a detailed match to check if image j matches image k. This process can be repeated a fixed number of times or until the match graph converges.

A concern when iterating rounds of query expansion is drift. Results of the secondary queries can quickly diverge from the original query. This is not a problem in our system, since query expanded image pairs are subject to detailed geometric verification before they are connected by an edge in the match graph.

2.3. Track Generation

The final step of the matching process is to combine all the pairwise matching information to generate consistent tracks across images, i.e., to find and label all of the connected components in the graph of individual feature matches (the *feature graph*). Since the matching information is stored locally on the compute node that it was computed on, the track generation process proceeds in two stages (Figure 1k–m). In the first, each node generates tracks from all the matching data it has available locally. This data is gathered at the master node and then broadcast over the network to all the nodes. Observe that the tracks for each connected component of the *match* graph can be processed independently. The track generation then proceeds with each compute node being assigned a connected component for which the tracks need to be produced. As we merge tracks based on shared features, inconsistent tracks can be generated, where two feature points in the same image belong to the same track. In such cases, we drop the offending points from the track.

Once the tracks have been generated, the next step is to extract the 2D coordinates of the feature points that occur in the tracks from the corresponding image feature files. We also extract the pixel color for each such feature point, which is later used for rendering the 3D point with the average

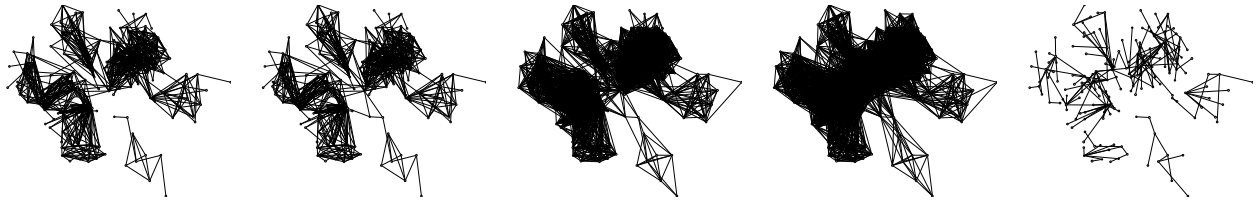

Figure 2. The evolution of the match graph as a function of the rounds of matching, and the skeletal set corresponding to it. Notice how the second round of matching merges the two components into one, and how rapidly the query expansion increases the density of the within component connections. The last column shows the skeletal set corresponding to the final match graph. The skeletal sets algorithm can break up connected components found during the match phase if it determines that a reliable reconstruction is not possible, which is what happens in this case.

color of the feature points associated with it. Again, this procedure proceeds in two steps. Given the per-component tracks, each node extracts the feature point coordinates and the point colors from the SIFT and image files that it owns. This data is gathered and broadcast over the network, where it is processed on a per connected component basis.

2.4. Geometric Estimation

Once the tracks have been generated, the next step is to run structure from motion (SfM) on every connected component of the match graph to recover a pose for every camera and a 3D position for every track. Most SfM systems for unordered photo collections are incremental, starting with a small reconstruction, then growing a few images at a time, triangulating new points, and doing one or more rounds of nonlinear least squares optimization (known as *bundle adjustment* [19]) to minimize the reprojection error. This process is repeated until no more cameras can be added. However, due to the scale of our collections, running such an incremental approach on all the photos at once is impractical.

The incremental reconstruction procedure described above has in it the implicit assumption that all images contribute more or less equally to the coverage and accuracy of the reconstruction. Internet photo collections, by their very nature are redundant—many photographs are taken from nearby viewpoints and processing all of them does not necessarily add to the reconstruction. It is thus preferable to find and reconstruct a minimal subset of photographs that capture the essential connectivity of the match graph and the geometry of the scene [8, 17]. Once this is done, we can add back in all the remaining images using pose estimation, triangulate all remaining points, and then do a final bundle adjustment to refine the SfM estimates.

For finding this minimal set, we use the skeletal sets algorithm of [17], which computes a spanning set of photographs that preserves important connectivity information in the image graph (such as large loops). In [17], a two-frame reconstruction is computed for each pair of matching images with known focal lengths. In our system, these pairwise reconstruction are computed as part of the parallel matching process. Once a skeletal set is computed, we estimate the SfM parameters of each resulting component using the incremental algorithm of [16]. The skeletal sets algorithm often breaks up connected components across weakly-connected boundaries, resulting in a larger set of components.

2.4.1 Bundle Adjustment

Having reduced the size of the SFM problem down to the skeletal set, the primary bottleneck in the reconstruction process is the non-linear minimization of the reprojection error, or bundle adjustment (BA). The best performing BA software available publicly is Sparse Bundle Adjustment (SBA) by Lourakis & Argyros [9]. The key to its high performance is the use of the so called Schur complement trick [19] to reduce the size of the linear system (also known as the normal equations) that needs to be solved in each iteration of the Levenberg-Marquardt (LM) algorithm. The size of this linear system depends on the 3D point and the camera parameters, whereas the size of the Schur complement only depends on the camera parameters. SBA then uses a dense Cholesky factorization to factor and solve the resulting reduced linear system. Since the number of 3D points in a typical SfM problem is usually two orders of magnitude or more larger than the number of cameras, this leads to substantial savings. This works for small to moderate sized problems, but for large problems with thousands of images, computing the dense Cholesky factorization of the Schur complement becomes a space and time bottleneck. For large problems however, the Schur complement itself is quite sparse (a 3D point is usually visible in only a few cameras) and exploiting this sparsity can lead to significant time and space savings.

We have developed a new high performance bundle adjustment software that, depending upon the size of the problem, chooses between a truncated and an exact step LM algorithm. In the first case, a block diagonal preconditioned conjugate gradient method is used to solve approximately the normal equations. In the second case, CHOLMOD [4], a sparse direct method for computing Cholesky factorization is used to exactly solve the normal equations via the Schur comple-

Data set	Images	Cores	Registered	Pairs verified	Pairs found	Time (hrs) Matching	Skeletal sets	Reconstruction
Dubrovnik	57,845	352	11,868	2,658,264	498,982	5	1	16.5
Rome	150,000	496	36,658	8,825,256	2,712,301	13	1	7
Venice	250,000	496	47,925	35,465,029	6,119,207	27	21.5	16.5

Table 1. Matching and reconstruction statisics for the three data sets.

ment trick. The first algorithm has low time complexity per iteration, but uses more LM iterations, while the second one converges faster at the cost of more time and memory per iteration. The resulting code uses significantly less memory than SBA and runs up to an order of magnitude faster. The exact runtime and memory savings depend upon the sparsity structure of the linear system involved.

2.5. Distributed Computing Engine

Our matching and reconstruction algorithm is implemented as a two-layered system. At the base of the system is an application-agonstic distributed computing engine. Except for a small core, which is easily ported, the system is cross-platform and runs on all major flavors of UNIX and Microsoft Windows. The system is small and extensible, and while it comes with a number of scheduling policies and data transfer primitives, the users are free to add their own.

The system is targeted at batch applications that operate on large amounts of data. It has extensive support for local caching of application data and on-demand transfer of data across the network. If a shared filesystem is available, it can be used, but for maximum performance, all data is stored on local disk and only transferred across the network as needed. It supports a variety of scheduling models, from trivially data parallel tasks to general map-reduce style computation. The distributed computing engine is written as set of Python scripts, with each stage of computation implemented as a combination of Python and C++ code.

3. Experiments

We report here the results of running our system on three city data sets downloaded from `flickr.com`: Dubrovnik, Croatia; Rome; and Venice, Italy. Figures 3(a), 3(b) and 3(c) show reconstructions of the largest connected components in these data sets. Due to space considerations, only a sample of the results is shown here, we encourage the reader to visit `http://grail.cs.washington.edu/rome`, where the complete results are posted, and additional results, data, and code will be posted over time.

The experiments were run on a cluster of 62 nodes with dual quad core processors each, on a private network with 1 GB/sec Ethernet interfaces. Each node was equipped with 32 GB of RAM and 1 TB of local hard disk space. The nodes were running the Microsoft Windows Server 2008 64-bit operating system.

The same vocabulary tree was used in all experiments. It was trained off-line on 1,918,101 features extracted from 20,000 images of Rome (not used in the experiments reported here). We trained a tree with a branching factor of 10 and a total of 136,091 vertices. For each image, we used the vocabulary tree to find the top 20 matching images. The top $k_1 = 10$ matches were used in the first verification stage, and the next $k_2 = 10$ matches were used in the second component matching stage. Four rounds of query expansion were done. We found that in all cases, the ratio of number of matches performed to the number of matches verified starts dropping off after four rounds. Table 1 summarizes statistics of the three data sets.

The reconstruction timing numbers in Table 1 bear some explanation. It is surprising that the SfM time for Dubrovnik is so much more than for Rome, and is almost the same as Venice, both which is are much larger data sets. The reason lies in how the data sets are structured. The Rome and Venice data sets are essentially a collection of landmarks, which at large scale have a simple geometry and visibility structure. The largest connected component in Dubrovnik on the other hand captures the entire old city. With its narrow alley ways, complex visibility and widely varying view points, it is a much more complicated reconstruction problem. This is reflected in the sizes of the skeletal sets associated with the largest connected components as shown in Table 2. As we mentioned above, the skeletal sets algorithm often breaks up connected components across weakly-connected boundaries, therefore the size of the connected component returned by the matching system (CC1), is larger than the connected component (CC2) that what the skeletal sets algorithm returns.

4. Discussion

At the time of writing this paper, searching on Flickr.com for the keywords “Rome” or “Roma” results in over 2,700,000 images. Our aim is to be able to reconstruct as much of the city as possible from these photographs in 24 hours. Our current system is about an order of magnitude away from this goal. We believe that our current approach can be scaled to problems of this size. However, a number

(a) Dubrovnik: Four different views and associated images from the largest connected component. Note that the component captures the entire old city, with both street-level and roof-top detail. The reconstruction consists of 4,585 images and 2,662,981 3D points with 11,839,682 observed features.

Colosseum: 2,097 images, 819,242 points

Trevi Fountain: 1,935 images, 1,055,153 points

Pantheon: 1,032 images, 530,076 points

Hall of Maps: 275 images, 230,182 points

(b) Rome: Four of the largest connected components visualized at canonical viewpoints [14].

San Marco Square: 13,699 images, 4,515,157 points

Grand Canal: 3,272 images, 561,389 points

(c) Venice: The two largest connected components visualized from two different view points each

of open questions remain.

The track generation, skeletal sets, and reconstruction algorithms are all operating on the level of connected components. This means that the largest few components completely dominate these stages. Currently, our reconstruction algorithm only utilizes multi-threading when solving the bundle adjustment problem. While this helps, it is not a scalable solution to the problem, as depending upon the connectiv-

Data set	CC1	CC2	Skeletal Set	Reconstructed
Dubrovnik	6,076	4619	977	4585
Rome	7,518	2,106	254	2,097
Venice	20,542	14,079	1,801	13,699

Table 2. Reconstruction statistics for the largest connected components in the three data sets. CC1 is the size of the largest connected component after matching, CC2 is the size of the largest component after skeletal sets. The last column lists the number of images in the final reconstruction.

ity pattern of the match graph, this can take an inordinate amount of time and memory. The key reason we are able reconstruct these large image collections is because of the large amount of redundancy present in Internet collections, which the skeletal sets algorithm exploits. We are currently exploring ways of parallelizing all three of these steps, with particular emphasis on the SfM system.

The runtime performance of the matching system depends critically on how well the verification jobs are distributed across the network. This is facilitated by the initial distribution of the images across the cluster nodes. An early decision to store images according to the name of the user and the Flickr ID of the image meant that most images taken by the same user ended up on the same cluster node. Looking at the match graph, it turns out (quite naturally in hindsight) that a user's own photographs have a high probability of matching amongst themselves. The ID of the person who took the photograph is just one kind of metadata associated with these images. A more sophisticated strategy would exploit all the textual tags and geotags associated with the images to predict what images likely to match and then distribute the data accordingly.

Finally, our system is designed with batch operation in mind. A more challenging problem is to start with the output of our system and to add more images as they become available.

Acknowledgement

This work was supported in part by SPAWAR, NSF grant IIS-0811878, the Office of Naval Research, the University of Washington Animation Research Labs, and Microsoft. We also thank Microsoft Research for generously providing access to their HPC cluster. The authors would also like to acknowledge discussions with Steven Gribble, Aaron Kimball, Drew Steedly and David Nister.

References

[1] Amazon Elastic Computer Cloud. `http://aws.amazon.com/ec2`.

[2] M. Antone and S. Teller. Scalable extrinsic calibration of omni-directional image networks. *IJCV*, 49(2):143–174, 2002.

[3] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *JACM*, 45(6):891–923, 1998.

[4] Y. Chen, T. Davis, W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *TOMS*, 35(3), 2008.

[5] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV*, pages 1–8, 2007.

[6] C. Früh and A. Zakhor. An automated method for large-scale, ground-based city model acquisition. *IJCV*, 60(1):5–24, 2004.

[7] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SISC*, 20(1):359, 1999.

[8] X. Li, C. Wu, C. Zach, S. Lazebnik, and J. Frahm. Modeling and Recognition of Landmark Image Collections Using Iconic Scene Graphs. In *ECCV*, pages 427–440, 2008.

[9] M. I. A. Lourakis and A. A. Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM TOMS*, 36(1):1–30, 2009.

[10] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[11] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, 2006.

[12] M. Pollefeys, D. Nister, J. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. Kim, P. Merrell, et al. Detailed real-time urban 3d reconstruction from video. *IJCV*, 78(2):143–167, 2008.

[13] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *CVPR*, pages 1–7, 2007.

[14] I. Simon, N. Snavely, and S. M. Seitz. Scene summarization for online image collections. In *ICCV*, pages 1–8, 2007.

[15] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *ICCV*, pages 1470–1477, 2003.

[16] N. Snavely, S. M. Seitz, and R. Szeliski. Photo Tourism: Exploring photo collections in 3D. *TOG*, 25(3):835–846, 2006.

[17] N. Snavely, S. M. Seitz, and R. Szeliski. Skeletal graphs for efficient structure from motion. In *CVPR*, pages 1–8, 2008.

[18] P. Torr and A. Zisserman. Robust computation and parametrization of multiple view relations. In *ICCV*, pages 727–732, 1998.

[19] B. Triggs, P. McLauchlan, H. R.I, and A. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Vision Algorithms'99*, pages 298–372, 1999.

[20] A. Vedaldi. Sift++. `http://vision.ucla.edu/~vedaldi/code/siftpp/siftpp.html`.

[21] L. Zebedin, J. Bauer, K. Karner, and H. Bischof. Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In *ECCV*, pages IV: 873–886, 2008.