

Building Rome in a Day

By Sameer Agarwal^a, Yasutaka Furukawa^a, Noah Snavely, Ian Simon^b, Brian Curless, Steven M. Seitz, and Richard Szeliski

Abstract

We present a system that can reconstruct 3D geometry from large, unorganized collections of photographs such as those found by searching for a given city (e.g., Rome) on Internet photo-sharing sites. Our system is built on a set of new, distributed computer vision algorithms for image matching and 3D reconstruction, designed to maximize parallelism at each stage of the pipeline and to scale gracefully with both the size of the problem and the amount of available computation. Our experimental results demonstrate that it is now possible to reconstruct city-scale image collections with more than a hundred thousand images in less than a day.

1. INTRODUCTION

Amateur photography was once largely a personal endeavor. Traditionally, a photographer would capture a moment on film and share it with a small number of friends and family members, perhaps storing a few hundred of them in a shoebox. The advent of digital photography, and the recent growth of photo-sharing Web sites such as Flickr.com, have brought about a seismic change in photography and the use of photo collections. Today, a photograph shared online can potentially be seen by millions of people.

As a result, we now have access to a vast, ever-growing collection of photographs the world over capturing its cities and landmarks innumerable times. For instance, a search for the term “Rome” on Flickr returns nearly 3 million photographs. This collection represents an increasingly complete photographic record of the city, capturing every popular site, façade, interior, fountain, sculpture, painting, and café. Virtually anything that people find interesting in Rome has been captured from thousands of viewpoints and under myriad illumination and weather conditions. For example, the Trevi Fountain appears in over 50,000 of these photographs.

How much of the city of Rome can be reconstructed in 3D from this photo collection? In principle, the photos of Rome on Flickr represent an ideal data set for 3D modeling research, as they capture the highlights of the city in exquisite detail and from a broad range of viewpoints. However, extracting high quality 3D models from such a collection is challenging for several reasons. First, the photos are *unstructured*—they are taken in no particular order and we have no control over the distribution of camera viewpoints. Second, they are *uncalibrated*—the photos are taken by thousands of different photographers and we know very little about the camera settings. Third, the *scale* of the problem is

enormous—whereas prior methods operated on hundreds or at most a few thousand photos, we seek to handle collections two to three orders of magnitude larger. Fourth, the algorithms must be *fast*—we seek to reconstruct an entire city in a single day, making it possible to repeat the process many times to reconstruct all of the world’s significant cultural centers.

Creating accurate 3D models of cities is a problem of great interest and with broad applications. In the government sector, city models are vital for urban planning and visualization. They are equally important for a broad range of academic disciplines including history, archeology, geography, and computer graphics research. Digital city models are also central to popular consumer mapping and visualization applications such as Google Earth and Bing Maps, as well as GPS-enabled navigation systems. In the near future, these models can enable augmented reality capabilities which recognize and annotate objects on your camera phone (or other) display. Such capabilities will allow tourists to find points of interest, driving directions, and orient themselves in a new environment.

City-scale 3D reconstruction has been explored previously.^{2, 8, 15, 21} However, existing large scale 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 computation. Images harvested from the Web have none of these simplifying characteristics. Thus, a key focus of our work has been to develop new 3D computer vision techniques that work “in the wild,” on extremely diverse, large, and unconstrained image collections.

Our approach to this problem builds on progress made in computer vision in recent years (including our own recent work on Photo Tourism¹⁸ and *Photosynth*), and draws from many other areas of computer science, including distributed systems, algorithms, information retrieval, and scientific computing.

2. STRUCTURE FROM MOTION

How can we recover 3D geometry from a collection of images? A fundamental challenge is that a photograph is a two-dimensional *projection* of a three-dimensional world. Inverting this projection is difficult as we have lost the depth of each point in the image. As humans, we can experience this problem by closing one eye, and noting our diminished

^a This work was done when the author was a postdoctoral researcher at the University of Washington.

^b Part of this work was done when the author was a graduate student at the University of Washington.

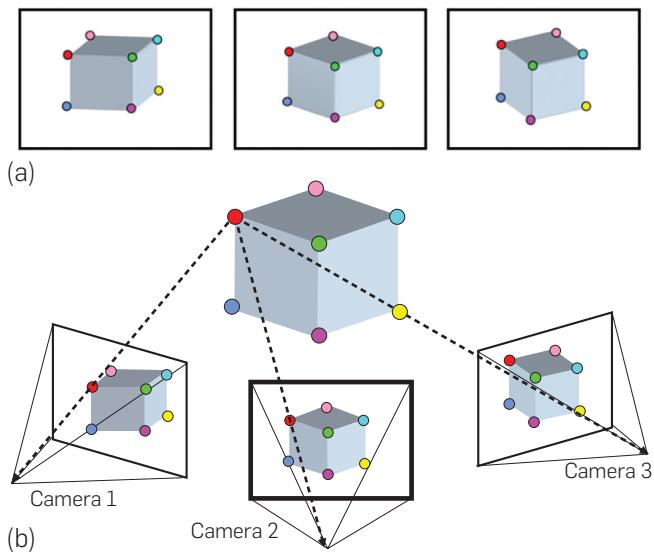
The original version of this paper was published in the *Proceedings of the 2009 IEEE International Conference on Computer Vision*.

depth perception. Fortunately, we have two eyes, and our brains can estimate depth by correlating points between the two images we perceive. This gives us some hope that from *multiple* photos of a scene, we can recover the shape of that scene.

Consider the three images of a cube shown in Figure 1a. We do not know where these images were taken, and we do not know *a priori* that they depict a specific shape (in this case, a cube). However, suppose we *do* know that the corners of the cube as seen in the images, i.e., the 2D projections of the 3D corners, are in correspondence: we know that the 2D dots with the same color correspond to the same 3D points. This correspondence gives us a powerful set of constraints on the 3D geometry of the cameras and points.¹⁰ One way to state these constraints is that given scene geometry (represented with 3D points) and camera geometry (a 3D position and orientation for each camera), we can predict where the 2D projections of each point should be in each image via the equations of perspective projection; we can then compare these projections to our original measurements.

Concretely, let X_i , $i = 1, \dots, 8$ denote the 3D positions of the corners of the cube and let R_j , c_j , and f_j , $j = 1, 2, 3$ denote the

Figure 1. (a) Three images of a cube, from unknown viewpoints. The color-coded dots on the corners show the known correspondence between certain 2D points in these images; each set of dots of the same color are projections of the same 3D point. (b) A candidate reconstruction of the 3D points (larger colored points) and cameras for the image collection shown above. Each image in the collection has an associated position and orientation. This reconstruction largely agrees with the observed 2D projections; when the red 3D point is projected into each image (depicted with the dotted lines), the predicted projection is close to the observed one. In the case of Camera 3 the projection is slightly off; the resulting residual is called the reprojection error, and is what we seek to minimize.



orientation, position, and the focal length of the three cameras. Then if x_{ij} is the image of point X_i in image j , we can write down the image formation equations as

$$x_{ij} = f_j \prod (R_j(X_i - c_j)), \quad (1)$$

where \prod is the projection function: $\prod(x, y, z) = (x/z, y/z)$. The *Structure from Motion* (SfM) problem is to infer X_i , R_j , c_j , and f_j from the observations x_{ij} .

The standard way to do this is to formulate the problem as an optimization problem that minimizes the total squared reprojection error:

$$\arg \min_{X_i, R_j, c_j, f_j} \sum_{i \sim j} \|x_{ij} - f_j \prod(R_j(X_i - c_j))\|^2. \quad (2)$$

Here, $i \sim j$ indicates that the point X_i is visible in image j . An example reconstruction, illustrating reprojection error, is shown in Figure 1. While this toy problem is easily solved, (2) is in general a difficult nonlinear least squares problem with many local minima, and has millions of parameters in large scenes. Section 5 describes the various techniques we use to solve (2) at scale.

3. THE CORRESPONDENCE PROBLEM

In the cube example above, we assumed that we were given as input a set of 2D correspondences between the input images. In reality, these correspondences are not given and also have to be estimated from the images. How can we do so automatically? This is the *correspondence* problem.

To solve the correspondence problem between two images, we might consider every patch in the first image and find the most similar patch in the second image. However, this algorithm quickly runs into problems. First, many image patches might be very difficult to match. For instance, a patch of clear blue sky is very challenging to match unambiguously across two images, as it looks like any other patch of sky, i.e., it is not *distinct*. Second, what happens if the second image is taken at a different time of day or with a different level of zoom?

The last 10 years have seen the development of algorithms for taking an image and detecting the most distinctive, repeatable features in that image. Such feature detectors not only reduce an image representation to a more manageable size, but also produce much more robust features for matching, invariant to many kinds of image transformations. One of the most successful of these detectors is SIFT (Scale-Invariant Feature Transform).¹³

Once we detect features in an image, we can match features across image pairs by finding similar-looking features. While exhaustive matching of all features between two images is prohibitively expensive, excellent results have been reported with approximate nearest neighbor search¹⁸; we use the ANN library.³ 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 if the nearest neighbor returned by ANN is sufficiently far away from the next nearest neighbor, it is declared a match.¹³

Despite their scale invariance and robustness to appearance changes, SIFT features are *local* and do not contain any global information about the image or about the location of other features in the image. Thus feature matching based on SIFT features is still prone to errors. However, since we assume that we are dealing with rigid scenes, there are strong geometric constraints on the locations of the matching features and these constraints can be used to clean up the matches. In particular, when a rigid scene is imaged by two pinhole cameras, there exists a 3×3 matrix F , the *Fundamental matrix*, such that corresponding points x_{ij} and x_{ik} (represented in homogeneous coordinates) in two images j and k satisfy¹⁰:

$$x_{ij}^\top F x_{ik} = 0. \quad (3)$$

A common way to impose this constraint is to use a greedy randomized algorithm to generate suitably chosen random estimates of F and choose the one that has the largest support among the matches, i.e., the one for which the most matches satisfy (3). This algorithm is called Random Sample Consensus (RANSAC)⁶ and is used in many computer vision problems.

4. CITY-SCALE MATCHING

Section 3 describes how to find correspondences between a pair of images. However, given a large collection with tens or hundreds of thousands of images, our task is to find correspondences spanning the entire collection. One way to think about this image matching problem is as a graph estimation problem where we are given a set of vertices corresponding to the images and we want to discover the set of edges connecting them. In this graph an edge connects a pair of images if and only if they are looking at the same part of the scene and have a sufficient number of feature matches between them. We will call this graph the *match graph*.

A naive way to determine the set of edges in the match graph is to perform all $O(n^2)$ image matches; for large collections, however, this is not practical. For a set 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, plus all of the time required to transfer the image and feature data between machines. Further, 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, i.e., the graph is sparse. We expect this to be the case for images from an entire city.

Instead, our system uses a multiround scheme: in each round we propose a set of edges in the match graph, then verify each edge through feature matching. If we find more than a minimum number of features, we keep the edge; otherwise we discard it. Thus, the problem reduces to that of formulating a method for quickly predicting when two images match. We use two methods to generate proposals: whole image similarity and query expansion.

4.1. Whole image similarity

A natural idea is to come up with a compact representation for computing the overall *similarity* of two images, then use this metric to propose edges to test.

For text documents, there are many techniques for quickly comparing the content of two documents. One common method is to represent each document as a vector of weighted word frequencies¹¹; the distance between two such vectors is a good predictor of the similarity between the corresponding documents.

Inspired by this work in document analysis, computer vision researchers have recently begun to apply similar techniques to visual object recognition with great success.^{5, 14, 16, 17} The basic idea is to take the SIFT features in a collection of photos and cluster them into “visual words.” By treating the images as documents consisting of these visual words, we can apply the machinery of document retrieval to efficiently match large data sets of photos. We use a fast tree-structured method for associating visual words with image features.¹⁴ Each photo is represented as a sparse histogram of visual word which we weight using the well-known “Term Frequency Inverse Document Frequency” (TFIDF) method¹¹; we compare two such histograms by taking their inner product. For each image, we determine the $k_1 + k_2$ most similar images, and verify the top k_1 of these. This forms the proposals for the first round of matching.

At this stage, we have a sparsely connected match graph. To derive the most comprehensive reconstruction possible, we want a graph with as few connected components as possible. To this end, we make further use of the proposals from the whole image similarity to try to connect the various connected components in this graph. For each image, we consider the next k_2 images suggested by the whole image similarity and verify those pairs which straddle two different connected components. We do this only for images which are in components of size two or more.^c

4.2. Query expansion

After performing the two rounds of matching based on whole image similarity, we have a sparse match graph, but this graph 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 original form, query expansion takes a set of documents that match a user’s query, then queries again with these initial results, *expanding* the initial query. The final results are a combination of these two queries. If we define a graph on the set of documents (including the query), with similar documents connected by an edge, then query expansion is equivalent to finding all vertices that are within two steps of the query vertex.

In our system, for every vertex j in the match graph, if vertices i and k are connected to j , we propose that i and k are also connected, and verify the edge (i, k) . This process can be

^c We use $k_1 = k_2 = 10$ in all our experiments.

repeated a fixed number of times or until the match graph converges.

4.3. Distributed implementation

We now consider a distributed implementation of the ideas described above. Our matching system is divided into three distinct phases: (1) pre-processing (Section 4.3.1), (2) verification (Section 4.3.2), and (3) track generation (Section 4.3.3). The system runs on a cluster of computers (nodes) with one node designated as the master node, responsible for job scheduling decisions.

4.3.1. 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. Each node down-samples its images to a fixed size and extracts SIFT features. This automatically performs load balancing, with more powerful nodes receiving more images to process. This is the only stage requiring a central file server; the rest of the system operates without using any shared storage.

At the end of this stage, the set of images (along with their features) has been partitioned into disjoint sets, one for each node.

4.3.2. Verification and detailed matching

The next step is to propose and verify (via feature matching) candidate image pairs, as described in Section 3.

For the first two rounds of matching, we use the whole image similarity (Section 4.1), and for the next four rounds we use query expansion (Section 4.2).

If we consider the TFIDF vectors corresponding to the images to be the rows of a huge matrix T , then the process of evaluating the whole image similarity is equivalent to evaluating the outer product $S = TT'$. Each node in the cluster evaluates the block of rows corresponding to its images, chooses the top $k_1 + k_2$ entries in each row and reports them to the master node. Query expansion is a simple and cheap enough operation that we let the master node generate these proposals.

If the images were all located on a single machine, verifying each proposed pair would be a simple matter of running through the set of proposals and performing SIFT matching, perhaps paying some attention to the order of the verifications so as to minimize disk I/O. However, in our case, the images and features are distributed across the cluster. Asking a node to match the image pair (i, j) may require it to fetch the image features from two other nodes of the cluster. This is undesirable due to the large difference between network transfer speeds and local disk transfers, as well as creating work for three nodes. Thus, the candidate edge verifications should be distributed across the network in a manner that respects the locality of the data.

We experimented with a number of approaches with surprising results. Initially, we tried to optimize network transfers before performing any verification. In this setup, once the master node knows all the image pairs that need to be verified, it builds another graph connecting image pairs which share

an image. Using MeTiS,¹² 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 problems, but not for large ones. Our assumption that verifying every pair of images takes the same constant amount of time was wrong; some nodes finished early and idled for up to an hour.

Our second idea was to over-partition the graph into small pieces, then parcel them out to nodes on demand. When a node requests a chunk of work, it is assigned the piece requiring the fewest network transfers. This strategy achieved better load balancing, but as the problem sizes grew, the graph we needed to partition became enormous and partitioning itself became a bottleneck.

The approach that gave the best result was to use a simple greedy bin-packing algorithm where each bin represents the set of jobs sent to a node. 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. A drawback of this algorithm is that it can require multiple sweeps over all the remaining image pairs: for large problems this can be a bottleneck. A simple solution is to consider only a fixed sized subset of the image pairs for scheduling. This windowed approach works very well in practice and our experiments use this method.

4.3.3. Track generation

Until now, we have only compared two images at a time. However, when a 3D point is visible in more than two images and the features corresponding to this point have been matched across these images, we need to group these features together so that the geometry estimation algorithm can estimate a single 3D point from all the features. We call a group of features corresponding to a single 3D point a *feature track* (Figure 2); the final step in the matching process is to combine all the pairwise matching information to

Figure 2: A track corresponding to a point on the face of the central statue of Oceanus (the embodiment of a river encircling the world in Greek mythology).



generate consistent tracks across images.

The problem of track generation can be formulated as the problem of finding connected components in a graph where the vertices are the features in all the images and edges connect matching features. Since the matching information is stored locally on the compute node where the matches were computed, the track generation process is distributed and proceeds in two stages. First, each node generates tracks from its local matching data. This data is gathered at the master node and then broadcast over the network to all the nodes. Second, each node is assigned a connected component of the match graph (which can be processed independently of all other components), and stitches together tracks for that component.

5. CITY-SCALE SfM

Once the tracks are generated, the next step is to use a SfM algorithm on each connected component of the match graph to recover the camera poses and a 3D position for every track.

Directly solving Equation 2 is a hard nonlinear optimization problem. 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*²⁰) to minimize the reprojection error. This process is repeated until no more images can be added. However, due to the scale of our collections, running such an incremental approach on all the photos at once was impractical.

To remedy this, we observed that Internet photo collections by their very nature are redundant. Many photographs are taken from nearby viewpoints (e.g., the front of the Colosseum) and processing all of them does not necessarily add to the reconstruction. Thus, it is preferable to find and reconstruct a minimal subset of photographs that capture the essential geometry of the scene (called a *skeletal set* in Snavely et al.¹⁹). Once this subset is reconstructed, the remaining images can be added to the reconstruction in one step by estimating each camera's pose with respect to known 3D points matched to that image. This process results in an order of magnitude or more improvement in performance.

Having reduced the SfM problem to its skeletal set, the primary bottleneck in the reconstruction process is the solution of (2) using bundle adjustment. Levenberg–Marquardt (LM) is the algorithm of choice for solving bundle adjustment problems; the key computational bottleneck in each iteration of LM is the solution of a symmetric positive definite linear system known as the *normal equations*.

We developed new high-performance bundle adjustment software that, depending upon the problem size, chooses between a truncated or an exact step LM algorithm. In the first case, a preconditioned conjugate gradient method is used to approximately solve the normal equations. In the second case, CHOLMOD,⁴ a sparse direct method for computing Cholesky factorizations, is used. The first algorithm has low time complexity per iteration,

but uses more LM iterations, while the second converges faster at the cost of more time and memory per iteration. The resulting code uses significantly less memory than the state-of-the-art methods and runs up to an order of magnitude faster. The runtime and memory savings depend upon the sparsity of the linear system involved.¹

6. MULTIVIEW STEREO

SfM recovers camera poses and 3D points. However, the reconstructed 3D points are usually sparse, containing only distinctive image features that match well across photographs. The next stage in 3D reconstruction is to take the registered images and recover dense and accurate models using a multiview stereo (MVS) algorithm.

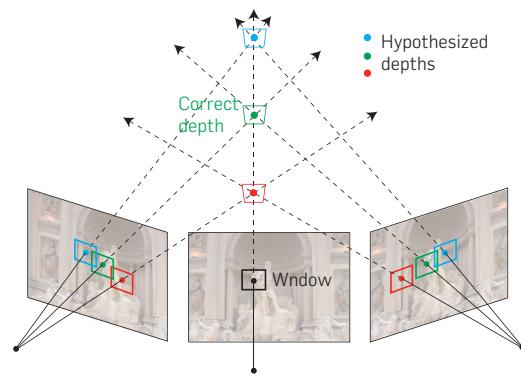
MVS algorithms recover 3D geometric information much in the same way our visual system perceives depth by fusing two views. In the MVS setting, we may have many images that see the same point and could be potentially used for depth estimation. Figure 3 illustrates how a basic algorithm estimates a depth value at a single pixel. To recover a dense model, we estimate depths for every pixel in every image and then merge the resulting 3D points into a single model.

For city-scale MVS reconstruction, the number of photos is well beyond what any standard MVS algorithm can operate on at once due to prohibitive memory consumption. Therefore, a key task is to group photos into a small number of manageable sized clusters that can each be used to reconstruct a part of the scene well.

Concretely, if we consider the SfM points as a sparse proxy for the dense MVS reconstruction, we want a clustering such that

1. Each SfM point is visible from enough images in a cluster.
2. The total number of clusters is small.
3. The size of each cluster is constrained to be lower than

Figure 3. A standard window-based multiview stereo algorithm. Given a pixel and an image window around it, we hypothesize a finite number of depths along its viewing ray. At each depth, the window is projected into the other images, and consistency among textures at these image projections is evaluated. At the true depth (highlighted in green), the consistency score is at its maximum.



a certain threshold, determined by the memory limitations of the machines.

The resulting clustering problem is a constrained discrete optimization problem (see Furukawa et al.⁹ for algorithmic details).

After the clustering, we solve for scene geometry within each cluster independently using a MVS algorithm, and then combine the results.⁹ This strategy not only makes it possible to perform the reconstruction, but also makes it straightforward to do so in parallel on many processors.

7. EXPERIMENTS

We report the results of running our system on three city-scale data sets downloaded from Flickr: Dubrovnik, Rome, and Venice.

The SfM experiments were run on a cluster of 62 nodes with dual quad-core processors, on a private network with 1GB/s Ethernet interfaces. Each node had 32GB of RAM and 1TB of local hard disk space with the Microsoft Windows Server 2008 64-bit operating system. For encoding the images as TFIDF vectors, we used a set of visual words, created from 20,000 images of Rome. The images used to create the visual word vocabulary were not used in any of the experiments.

Figure 4 shows reconstructions of the largest connected components of these data sets. Due to space considerations, only a sample of the results are shown here. Complete result are posted at <http://grail.cs.washington.edu/rome>.

For whole image similarity proposals, the top $k_1 = 10$ were used in the first verification stage, and the next $k_2 = 10$ were used in the second component matching stage. Four rounds of query expansion were done. In all cases, the ratio of the 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 SfM timing numbers in Table 1 bear some explanation. It is surprising that running SfM on Dubrovnik took so much more time than for Rome, and is almost the same as Venice, both of which are much larger data sets. The reason lies in the structure of the data sets. The Rome and Venice sets are essentially collections of landmarks which mostly have a simple geometry and visibility structure. The largest connected component in Dubrovnik, on the other hand, captures the entire old city. With its complex visibility and widely varying viewpoints, reconstructing Dubrovnik is a much more complicated SfM problem. This is reflected in the sizes of the skeletal sets associated with the largest connected components shown in Table 2.

Figure 4 also shows the results of running our MVS⁹ on city-scale reconstructions produced by our matching and SfM system. Figure 4 shows MVS reconstructions (rendered as colored points) for St. Peter's Basilica (Rome), the Colosseum (Rome), Dubrovnik, and San Marco Square (Venice), while Table 3 provides timing and size statistics.

The largest dataset—San Marco Square—contains 14,000 input images which were processed into 67

clusters and yielded 28 million surface points in less than 3 h. While our system successfully reconstructs dense and high quality 3D points for these very large scenes, our models contain holes in certain places. For example, rooftops where image coverage is poor, and ground planes where surfaces are usually not clearly visible. On the other hand, in places with many images, the reconstruction quality is very high, as illustrated in the close-ups in Figure 4.

8. DISCUSSION

A search on Flickr.com for the keywords “Rome” or “Roma” results in over 4 million images. Our aim was to be able to reconstruct as much of the city as possible from these photographs in 24 h. Our current system is about an order of magnitude away from this goal. Since the original publication of this work, Frahm et al. have built a system that uses the massive parallelism of GPUs to do city scale reconstructions on a single workstation.⁷

In our system, 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. We are currently exploring ways of parallelizing all three of these steps, with particular emphasis on the SfM system.

Another issue with the current system is that it produces a set of disconnected reconstructions. If the images come with geotags/GPS information, our system can try and geo-locate the reconstructions. However, this information is frequently incorrect, noisy, or missing.

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 meta-data associated with these images. A more sophisticated strategy would exploit all the textual tags and geotags associated with the images to predict what images are likely to match distributing the data accordingly.

Finally, our system is designed with batch operation in mind. A more challenging problem is to make the system incremental.

Acknowledgments

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 thank Microsoft Research for generously providing access to their HPC cluster and Szymon Rusinkiewicz for Qsplat software. The authors would also like to acknowledge discussions with Steven Gribble, Aaron Kimball, Drew Steedly and David Nister.

Figure 4. From left to right, sample input images, structure from motion reconstructions, and multiview stereo reconstructions.



Table 1. Matching and SfM statistics for the three cities.

Data set	Images	Cores	Registered	Pairs verified	Pairs found	Time (h)		
						Matching	Skeletal sets	SfM
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 2. Reconstruction statistics for the largest connected components in the three data sets.

Data set	CC1	CC2	Skeletal set	Reconstructed
Dubrovnik	6,076	4,619	977	4,585
Rome	7,518	2,106	254	2,097
Venice	20,542	14,079	1,801	13,699

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.

Table 3. MVS reconstruction statistics for the four view clusters.

Landmark	Images	Images in clusters	Clusters	Time (min)		
				MVS points	Clustering	Reconstruction
St. Peter's Basilica	1,275	333	4	5,107,847	1.3	94.2
Colosseum	1,167	528	7	5,747,083	1.5	59.2
Dubrovnik	6,304	2,628	28	14,051,331	21.1	221.4
San Marco Square	13,709	5,917	67	27,707,825	39.3	176.3

References

- Agarwal, S., Snavely, N., Seitz, S.M., Szeliski, R. Bundle adjustment in the large. In *ECCV (2)*, volume 6312 of *Lecture Notes in Computer Science* (2010). K. Daniilidis, P. Maragos, and N. Paragios, eds. Springer, Berlin, Germany, 29–42.
- Antone, M.E., Teller, S.J. Scalable extrinsic calibration of omnidirectional image networks. *Int. J. Comput. Vis.* 49, 2–3 (2002), 143–174.
- Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM* 45, 6 (1998), 891–923.
- Chen, Y., Davis, T.A., Hager, W.W., Rajamanickam, S. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Softw.* 35, 3 (2008), 1–14.
- 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)*, IEEE, 1–8.
- Fischler, M.A., Bolles, R.C. Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography. *Commun. Assoc. Comp. Mach.* 24 (1981), 381–395.
- Frahm, J.-M., Georgel, P.F., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.-H., Dunn, E., Clipp, B., Lazebnik, S. Building Rome on a cloudless day. In *ECCV (4)*, volume 6314 of *Lecture Notes in Computer Science* (2010). K. Daniilidis, P. Maragos, and N. Paragios, eds. Springer, Berlin, Germany, 368–381.
- Früh, C., Zahor, A. An automated method for large-scale, ground-based city model acquisition. *Int. J. Comput. Vis.* 60, 1 (2004), 5–24.
- Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R. Towards internet-scale multi-view stereo. In *CVPR (2010)*, IEEE, 1434–1441.
- Hartley, R.I., Zisserman, A. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, U.K., 2003.
- Jones, K. A statistical interpretation of term specificity and its application in retrieval. *J. Doc.* 60, 5 (2004), 493–502.
- Karypis, G., Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392.
- Lowe, D. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vis.* 60, 2 (2004), 91–110.
- Nistér, D., Stewénius, H. Scalable recognition with a vocabulary tree. In *CVPR (2)* (2006). IEEE Computer Society, 2161–2168.
- Pollefeys, M., Nister, D., Frahm, J., Akbarzadeh, A., Mordohai, P., Clipp, B., Engels, C., Gallup, D., Kim, S., Merrell, P. et al. Detailed real-time urban 3d reconstruction from video. *IJCV* 78, 2 (2008), 143–167.
- Schindler, G., Brown, M., Szeliski, R. City-scale location recognition. In *CVPR (2007)*, IEEE Computer Society.
- Sivic, J., Zisserman, A. Video Google: A text retrieval approach to object matching in videos. In *ICCV (2003)*, 1470–1477.
- Snavely, N., Seitz, S.M., Szeliski, R. Photo Tourism: Exploring photo collections in 3d. *ACM Trans. Graph.* 25, 3 (2006), 835–846.
- Snavely, N., Seitz, S.M., Szeliski, R. Skeletal graphs for efficient structure from motion. In *CVPR (2008)*, IEEE Computer Society.
- Triggs, B., McLauchlan, P., Hartley, R.I., Fitzgibbon, A. Bundle adjustment—A modern synthesis. In *Vision Algorithms '99* (1999), 298–372.
- Zebelin, L., Bauer, J., Karner, K.F., Bischof, H. Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In *ECCV (4)*, volume 5305 of *Lecture Notes in Computer Science* (2008). D.A. Forsyth, P.H.S. Torr, and A. Zisserman, eds. Springer, Berlin, Germany, 873–886.

Sameer Agarwal (sameeragarwal@google.com), Google Inc., Seattle, WA.

Steven M. Seitz (seitz@cs.washington.edu), Google Inc. & University of Washington, Washington, Seattle, WA.

Yasutaka Furukawa (furukawa@google.com), Google Inc., Seattle, WA.

Ian Simon (iansimon@microsoft.com), Microsoft Corporation, Redmond, WA.

Richard Szeliski (szeliski@microsoft.com), Microsoft Research, Redmond, WA.

Brian Curless (curless@washington.edu), University of Washington, Washington, Seattle, WA.

Noah Snavely (snavey@cs.cornell.edu), Cornell University, Ithaca, NY.