Towards Internet-scale Multi-view Stereo

Yasutaka Furukawa[1] Brian Curless[2] Steven M. Seitz[1,2] Richard Szeliski[3]
[1]Google Inc. [2]University of Washington [3]Microsoft Research

Abstract

This paper introduces an approach for enabling existing multi-view stereo methods to operate on extremely large unstructured photo collections. The main idea is to decompose the collection into a set of overlapping sets of photos that can be processed in parallel, and to merge the resulting reconstructions. This overlapping clustering problem *is formulated as a constrained optimization and solved iteratively. The merging algorithm, designed to be parallel and out-of-core, incorporates robust filtering steps to eliminate low-quality reconstructions and enforce global visibility constraints. The approach has been tested on several large datasets downloaded from Flickr.com, including one with over ten thousand images, yielding a 3D reconstruction with nearly thirty million points.*

Figure 1. Our dense reconstruction of Piazza San Marco (Venice) from 13,703 images with 27,707,825 reconstructed MVS points (further upsampled x9 for high quality point-based rendering).

1. Introduction

The state of the art in 3D reconstruction from images has undergone a revolution in the last few years. Coupled with the explosion of imagery available online and advances in computing, we have the opportunity to run reconstruction algorithms at massive scale. Indeed, we can now attempt to reconstruct the entire world, i.e., every building, landscape, and (static) object that can be photographed.

The most important technological ingredients towards this goal are already in place. Matching algorithms (e.g., SIFT [17]) provide accurate correspondences, structure-from-motion (SFM) algorithms use these correspondences to estimate precise camera pose, and multi-view-stereo (MVS) methods take images with pose as input and produce dense 3D models with accuracy nearly on par with laser scanners [22]. Indeed, this type of pipeline has already been demonstrated by a few research groups [11, 12, 14, 19], with impressive results.

To reconstruct everything, one key challenge is *scalability*.[1] In particular, how can we devise reconstruction algorithms that operate at *Internet-scale*, i.e., on the millions of images available on Internet sites such as Flickr.com? Given recent progress on Internet-scale matching and SFM (notably Agarwal *et al*.'s Rome-in-a-day project [1]), we focus our efforts in this paper on the last stage of the pipeline, i.e., Internet-scale MVS.

MVS algorithms are based on the idea of correlating measurements from several images at once to derive 3D surface information. Many MVS algorithms aim at reconstructing a global 3D model by using all the images available simultaneously [9, 13, 20, 24]. Such an approach is not feasible as the number of images grows. Instead, it becomes important to *select* the right subset of images, and to *cluster* them into manageable pieces.

We propose a novel view selection and clustering scheme that allows a wide class of MVS algorithms to scale up to massive photo sets. Combined with a new merging method that robustly filters out low-quality or erroneous points, we demonstrate our approach running for thousands of images of large sites and one entire city. Our system is the first to demonstrate an unstructured MVS approach at city-scale.

We propose an *overlapping view clustering* problem [2], in which the goal is to decompose the set of input images into clusters that have small overlap. Overlap is important for the MVS problem, as a strict partition would undersample surfaces near cluster boundaries. Once clustered, we apply a state-of-the-art MVS algorithm to reconstruct dense 3D points, and then merge the resulting reconstructions into

[1]There are other challenges such as handling complex BRDFs and lighting variations, which we do not address in this paper.

a single dense point-based model. Robust filtering algorithms are introduced to handle reconstruction errors and the vast variations in reconstruction quality that occur between distant and nearby views of objects in Internet photo collections. The filters are designed to be out-of-core and parallel, in order to process a large number of MVS points efficiently. We show visualizations of models containing tens of millions of points (see Figure 1).

1.1. Related Work

Scalability has rarely been a consideration in prior MVS algorithms, as prior datasets have been either relatively small [22] or highly structured (e.g., a video sequence which can be decomposed into short time intervals [19]).

Nevertheless, some algorithms lend themselves naturally to parallelization. In particular, several algorithms operate by solving for a depth map for each image, using a local neighborhood of nearby images, and then merge the resulting reconstructions [11, 12, 18, 19]. Each depth map can be computed independently and in parallel. However, the depth maps tend to be noisy and highly redundant, leading to wasted computational effort. Therefore, these algorithms typically require additional post-processing steps to clean up and merge the depth maps.

Many of the best performing MVS algorithms instead reconstruct a global 3D model directly from the input images [9, 13, 20, 24]. Global methods can avoid redundant computations and often do not require a clean-up post-process, but scale poorly. One exception is Jancosek *et al*. [14] who achieve scalability by designing the algorithm out-of-core. However, this is a sequential algorithm. In contrast, we seek an out-of-core algorithm that is also parallelizable.

With depth-map based MVS algorithms, several authors have succeeded in large-scale MVS reconstructions [18, 19]. Pollefeys *et al*. [19] present a real-time MVS system for long image sequences. They estimate a depth map for each input image, reduce noise by fusing nearby depth maps, and merge the resulting depth maps into a single mesh model. Micusik *et al*. [18] propose a piece-wise planar depth map computation algorithm with very similar clean-up and merging steps. However, both methods have been tested only on highly structured, street-view datasets obtained by a video camera mounted on a moving van, and not the unstructured photo collections that we consider in this paper, which pose additional challenges.

Besides scalability, variation in reconstruction quality is another challenge in handling large unorganized image collections, as surfaces may be imaged from both close up and far away. Goesele *et al*. [12] proposed the first MVS method applied to Internet photo collections, which handles variation in image sampling resolutions by selecting images with the most compatible resolution. Gallup *et al*. [10] select images at different baselines and image resolutions to control depth accuracy. Both of these methods handle variation by selecting images prior to reconstruction. These techniques may be used in conjunction with the methods proposed here, but the major difference in our work is that we also handle the variation in a post-processing step, when merging reconstructions. We note that some prior depth map merging algorithms take into account estimates of uncertainty, e.g., by taking weighted combinations of depth samples to recover a mesh [4, 25]. While such approaches can handle noise variation, we find they do not perform well for large Internet photo collections, where resolution variation is a major factor, because combining high and low resolution geometries in the standard ways will tend to attenuate high resolution detail. We instead propose a simple merging strategy that filters out low resolution geometry, which we have found to be robust and well-tailored to recovering a point-based model as output.

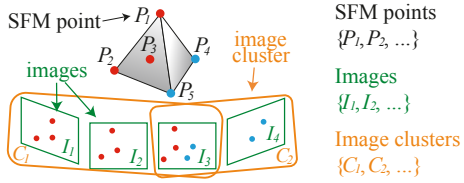


Figure 2. Our view clustering algorithm takes images $\{I_i\}$, SFM points $\{P_j\}$, and their associated visibility information $\{V_j\}$, then produces overlapping image clusters $\{C_k\}$.

The rest of the paper is organized as follows. The view-clustering algorithm is explained in Section 2, and details of the MVS point merging and rendering are given in Section 3. Experimental results are provided in Section 4 and we conclude the paper in Section 5. Our implementation of the proposed view-clustering algorithm is available at [6].

2. View Clustering

We assume that our input images $\{I_i\}$ have been processed by an SFM algorithm to yield camera poses and a sparse set of 3D points $\{P_j\}$, each of which is visible in a set of images denoted by V_j. We treat these SFM points as sparse samples of the dense reconstruction that MVS will produce. As such, they can be used as a basis for view clustering. More specifically, the goal of view clustering is to find (an unknown number of) overlapping image clusters $\{C_k\}$ such that each cluster is of manageable size, and each SFM point can be accurately reconstructed by at least one of the clusters (see Figure 2).

2.1. Problem Formulation

The clustering formulation is designed to satisfy the following three constraints: (1) redundant images are excluded from the clusters (*compactness*), (2) each cluster is small enough for an MVS reconstruction (*size constraint*); and

(3) MVS reconstructions from these clusters result in minimal loss of content and detail compared to that obtainable by processing the full image set (*coverage*). Compactness is important for computational efficiency but also to improve accuracy, as Internet photo collections often contain hundreds or thousands of photos acquired from nearly the same viewpoint, and a cluster consisting entirely of near-duplicate views will yield a noisy reconstruction due to insufficient baseline.

More concretely, our objective is to minimize the total number of images $\sum_k |C_k|$ in the output clusters, subject to the following two constraints. The first is an upper bound on the size of each cluster so that an MVS algorithm can be used for each cluster independently: $\forall k, |C_k| \leq \alpha$. α is determined by computational resources, particularly memory limitations.

The second encourages the coverage of the final MVS reconstructions as follows. We say an SFM point P_j is *covered* if it is sufficiently well reconstructed by the cameras in at least one cluster C_k. To quantify this notion of "well-reconstructed," we introduce a function $f(P, C)$ that measures the expected reconstruction accuracy achieved at a 3D location P by a set of images C. This function depends on the camera baselines and pixel sampling rates (see the Appendix for our definition of f). We say that P_j is covered if its reconstruction accuracy in at least one of the clusters C_k is at least λ times $f(P_j, V_j)$, which is the expected accuracy obtained when using all of P_j's visible images V_j:

$$P_j \text{ is covered if } \max_k f(P_j, C_k \cap V_j) \geq \lambda f(P_j, V_j),$$

where $\lambda = 0.7$ in our experiments. The coverage constraint is that for each set of SFM points visible in one image, the ratio of covered points must be at least δ (also set to 0.7, in our experiments). Note that we enforce this coverage ratio on each image, instead of on the entire reconstruction, to encourage good spatial coverage and uniformity.

In summary, our overlapping clustering formulation is defined as follows:

$$\text{Minimize } \sum_k |C_k| \text{ subject to} \qquad \text{(compactness)}$$

- $\forall k \;\; |C_k| \leq \alpha,$ (size)
- $\forall i \;\; \dfrac{\{\#\text{ of covered points in } I_i\}}{\{\#\text{ of points in } I_i\}} \geq \delta.$ (coverage)

There are a couple of points worth noting about this formulation. First, the minimization causes redundant images to be discarded, whenever constraints can be achieved with a smaller set of images. Second, the proposed formulation automatically allows overlapping clusters. Finally, the formulation implicitly incorporates image quality factors (e.g., sensor noise, blur, poor exposure), as poor quality images have fewer SFM points, and are thus more costly to include with the coverage constraint.

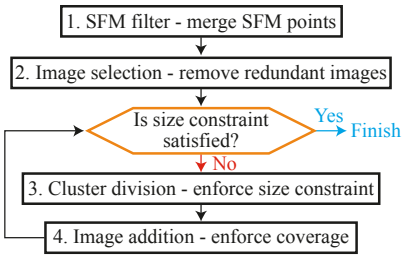


Figure 3. View clustering algorithm consists of four steps, where the last two steps are iterated until all the constraints are satisfied.

2.2. View Clustering Algorithm

Solving the proposed clustering problem is challenging, because the constraints are not in a form readily handled by existing methods like k-means, normalized cuts [16, 23], etc. Before presenting our algorithm, we first introduce some neighborhood relations for images and SFM points. A pair of images I_l and I_m are defined to be *neighbors* if there exists an SFM point that is visible in both images. Similarly, a pair of image sets are neighbors if there exists a pair of images (one from each set) that are neighbors. Finally, a pair of SFM points P_j and P_k are defined to be neighbors if 1) they have similar visibility, that is, their visible image sets V_j and V_k are neighbors according to the above definition, and 2) the projected locations of P_j and P_k are within τ_1 pixels in every image in $(V_j \cup V_k)$, where $\tau_1 = 64$ is used.

Figure 3 provides an overview of our approach, which consists of four steps. The first two steps are pre-processing, while the last two steps are repeated in an iterative loop.

1. SFM filter – merging SFM points: Having accurate measures of point visibility is key to the success of our view clustering procedure. Undetected or unmatched image features lead to errors in the point visibility estimates V_j (typically in the form of missing images). We obtain more reliable visibility estimates by aggregating visibility data over a local neighborhood, and merging points in that neighborhood. The position of the merged point is the average of its neighbors, while the visibility becomes the union. This step also significantly reduces the number of SFM points and improves running time of the remaining three steps. Specifically, starting from a set of SFM points, we randomly select one point, merge it with its neighbors, output the merged point, and remove both the point and its neighbors from the input set. We repeat the procedure until the input set is empty. The set of merged points becomes the new point set, which, with some abuse of notation, is also denoted as $\{P_j\}$.[2] See Figure 4 for a sample output of this step.

2. Image selection – removing redundant images: Start-

[2] An even better approach would be to re-detect and match new image features to improve visibility information as in [8]. However, this algorithm would be significantly more expensive.

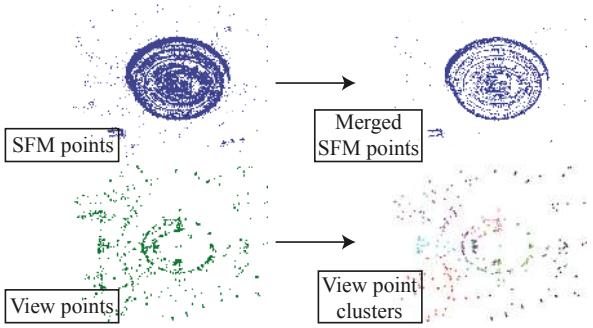


Figure 4. Top: The first step of our algorithm is to merge SFM points to enrich visibility information (SFM filter). Bottom: Sample results of our view clustering algorithm. View points belonging to extracted clusters are illustrated in different colors.

ing with the full image set, we test each image and remove it if the coverage constraint still holds after the removal. The removal test is performed for all the images enumerated in increasing order of image resolution (# of pixels), so that low-resolution images are removed first. Note that images are permanently discarded in this step to speed up the following main optimization steps.

3. Cluster division – enforcing the size constraint: Next, we enforce the size constraint by splitting clusters, while ignoring coverage. More concretely, we divide an image cluster into smaller components if it violates the size constraint. The division of a cluster is performed by the Normalized-Cuts algorithm [23] on a visibility graph, where nodes are images. The edge weight e_{lm} between an image pair (I_l, I_m) measures how much I_l and I_m together contribute to MVS reconstruction at relevant SFM points: $e_{lm} = \sum_{P_j \in \Theta^{lm}} \frac{f(P_j, \{I_l, I_m\})}{f(P_j, V_j)}$, where Θ^{lm} denotes a set of SFM points visible in both I_l and I_m. Intuitively, images with high MVS contribution have high edge weights among them and are less likely to be cut. The division of a cluster repeats until the size constraint is satisfied for all the clusters.

4. Image addition – enforcing coverage: The coverage constraint may have been violated in step 3, and we now add images to each cluster in order to cover more SFM points and reestablish coverage. In this step, we first construct a list of possible actions, where each action measures the effectiveness of adding an image to a cluster to increase coverage. More concretely, for each uncovered SFM point P_j, let $C_k = \text{argmax}_{C_l} f(P_j, C_l)$ be the cluster with the maximum reconstruction accuracy. Then, for P_j, we create an action $\{(I \rightarrow C_k), g\}$ that adds image $I (\in V_j, \notin C_k)$ to C_k, where g measures the effectiveness and is defined as $f(P_j, C_k \cup \{I\}) - f(P_j, C_k)$. Note that we only consider actions that add images to C_k instead of every cluster that could cover P_j for computational efficiency. Since actions with the same image and cluster are generated from multiple SFM points, we merge such actions while summing up the measured effectiveness g. Actions in the list are sorted in a decreasing order of their effectiveness.

Having constructed an action list, one approach would be to take the action with the highest score, then recompute the list again, which is computationally too expensive. Instead, we consider actions whose scores are more than 0.7 times the highest score in the list, then repeat taking an action from the top of the list. Since an action may change the effectiveness of other similar actions, after taking one action, we remove any *conflicting* ones from the list, where two actions $\{(I \rightarrow C), g\}$, $\{(I' \rightarrow C'), g'\}$ are conflicting if I and I' are neighbors. The list construction and image addition repeat until the coverage constraint is satisfied.

After the image addition, the size constraint may be violated, and the last two steps are repeated until both constraints are satisfied.

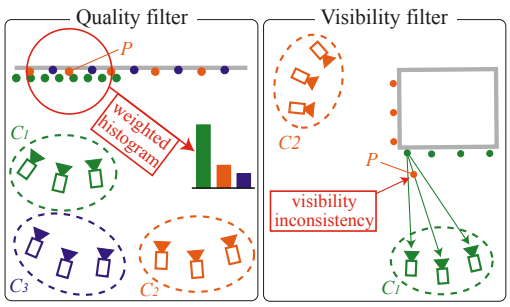
We note that the size and coverage constraints are not difficult to satisfy; indeed, an extreme solution is to creating a small cluster for each SFM point with sufficient baseline/resolution. In this extreme case, the resulting clusters will likely contain many duplicates and therefore have a poor compactness score. Typically, our approach of splitting clusters then adding a few images (usually at boundaries) tends to rapidly and easily satisfy the constraints while achieving reasonable (though not optimal) compactness scores; it terminates in a couple of iterations in all of our experiments. While our approach is not globally optimal, we note that achieving optimal compactness is not critical for our application.

3. MVS Filtering and Rendering

Having extracted image clusters, Patch-based MVS software (PMVS) by Furukawa *et al*. [7] is used to reconstruct 3D points for each cluster independently. Any MVS algorithm could be used, but we chose PMVS, which is publicly available. In this section, we propose two filters that are used in merging reconstructed points to handle reconstruction errors and variations in reconstruction quality (see Figures 5 and 6). Our filtering algorithms are designed to be out-of-core and operate in parallel to handle a large number of MVS points efficiently. We now describe the two filtering algorithms, discuss their scalability, and explain how merged MVS points are rendered.

3.1. Quality Filter

The same surface region may be reconstructed in multiple clusters with varying reconstruction quality: nearby clusters produce dense, accurate points, while distant clusters produce sparse, noisy points. We want to filter out the latter, which is realized by the following *quality filter*. Let P_j and V_j denote an MVS point and its visibility information estimated by the MVS algorithm, respectively. Sup-




```
Quality Filter
For each reference cluster Ck on each node
    Initialize histograms of MVS points
      reconstructed from Ck;
    For each cluster Cl
        For each point P in Ck
            Compute histogram entry Hl for P;
    Filter out points in Ck using histogram;
```

```
Visibility Filter
For each reference cluster Ck on each node
    For each image I in Ck
        Compute a depth-map for I;
    For each remaining cluster Cl
        For each point P in Cl
            For each image I in Ck
                If P and I conflicts
                    Increment conflict count for P;
        Save conflict counts for Cl to file;
For each cluster Ck on each node
    Read files containing conflict counts for Ck;
    Filter out points in Ck with conflict counts;
```

Figure 5. Quality and visibility filters are used in merging MVS reconstructions, where both filters are designed to be out-of-core as well as parallel. Left: MVS point P is tested against the filters. Right: Pseudo codes, where loops highlighted in blue can be executed in parallel.

pose P_j has been reconstructed from cluster C_k (a reference cluster). We first collect MVS points $\{Q_m\}$ and their associated visibility information $\{V_m\}$ from all the clusters 1) that have compatible normals with P_j, i.e., angle difference being less than $90°$; and 2) whose projected locations are within τ_2 pixels from that of P_j in every image in V_j ($\tau_2 = 8$ in our experiments). From the collected MVS points, we compute a histogram $\{H_l\}$, where H_l is the sum of reconstruction accuracies $f(Q_m, V_m)$ associated with MVS points reconstructed from C_l. Since a cluster with accurate and dense points should have a significantly larger value than the others, P_j is filtered out if the corresponding histogram value H_k is less than half the maximum: $H_k < 0.5 \max_l H_l$. We repeat this procedure by examining each reference cluster in turn, which can be executed in parallel.

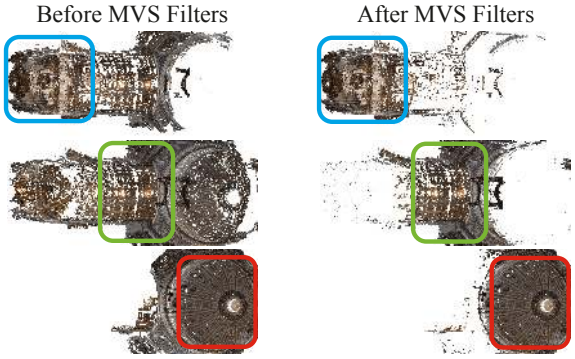


Figure 6. MVS points reconstructed from three view clusters before and after the MVS filters for St Peter's Basilica. Before the filtering, MVS points have large overlaps. Each cluster has its own 3D space, highlighted in color rectangles, where reconstructions become the most accurate over all the clusters. Points outside such a space are mostly removed by our filters.

3.2. Visibility Filter

The *visibility filter* enforces consistency in visibility information associated with MVS points over the entire reconstruction. The filter is, in fact, very similar to the one used in PMVS [7, 9]. The difference is that PMVS enforces the intra-cluster consistency inside each cluster, while our filter enforces inter-cluster visibility consistency over an entire reconstruction by comparing PMVS outputs from all the clusters. More concretely, for each MVS point, we count the number of times it *conflicts* with reconstructions from other clusters. The point is filtered out if the *conflict count* is more than three. Conflict counts are computed as follows.

Let Θ_k denote a set of MVS points reconstructed from cluster C_k (a reference cluster). We construct depths maps for images in C_k by projecting Θ_k into their visible images. Depth maps also store reconstruction accuracies associated with MVS points.[3] We compute the conflict count of each MVS point P in non-reference clusters as the number of depth maps in C_k that conflict with P. P is defined to conflict with a depth map if P is closer to the camera than the depth map by a small margin, and the reconstruction accuracy of P is less than half the value stored in the depth map. Note that we repeat this procedure by changing the reference cluster one by one, which can again be executed in parallel. The conflict count of the same MVS point is computed multiple times from different executions of this step, and their sum is tested against the threshold.

3.3. Scalability

MVS reconstruction and filtering are the most computationally expensive and memory intensive parts of our system. Here, we focus on memory complexity, which is the more critical factor for scalability.[4]

The memory expense of the MVS reconstruction step depends on the choice of the core MVS algorithm, but is not an issue with our system, because the number of im-

[3] PMVS recovers a point for every abutting set of 2×2 pixels in our setting, and depth maps are computed at half resolution. For an image belonging to multiple clusters, multiple depth maps are computed.

[4] Memory consumption is more critical for MVS, because PMVS (and some other MVS algorithms [12]) can utilize SFM visibility information to restrict sets of images to be matched instead of exhaustively trying every possible pair. Therefore, running-time of such algorithms is more or less linear in the amount of surface area to be reconstructed, while memory limitation is an unavoidable issue.

ages in each cluster is upper-bounded by the constant α. We now discuss the average case memory requirements of the MVS filtering algorithms as the worst-case analysis is difficult. Let N_P, N_I and N_M denote the average numbers of pixels per image, images per cluster, and reconstructed MVS points per cluster, respectively. N_C denotes the number of extracted image clusters. The amount of memory required for the quality and visibility filters are $2N_M$ and $2N_M + N_P N_I$, respectively, because 1) MVS points from at most two clusters need to be stored at any instant in both filters, and 2) the visibility filter needs to store depth maps for only one cluster at a time. Note that the total amount of data processed by the two filters are $N_M N_C$ and $N_M N_C + N_P N_I N_C$, respectively, which will not fit into memory for very large datasets.

3.4. Rendering

Having reconstructed MVS points, we associate a color with each point by taking the average of pixel colors at its image projections in the visible images. QSplat [21] is used to visualize 3D colored points with small modifications and enhancements as described in the supplementary material, including 3×3 upsampling of MVS points to improve point-based rendering quality.

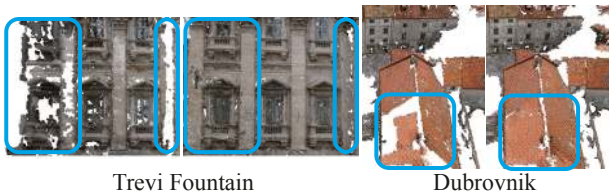
4. Experimental Results

The proposed algorithm is implemented in C++ and a PC with Dual Xeon 2.27 GHz processors is used for the experiments. We thank Agarwal *et al*. [1] for providing us with the datasets and SFM outputs. We use Graclus [5, 16] for Normalized Cuts, PMVS [7] for the core MVS reconstruction, and QSplat [21] for the final visualization. Our view clustering algorithm has several parameters. We use the same set of parameters for all the datasets: $\alpha = 150$, $\beta = 4$, $\lambda = 0.7$ and $\delta = 0.7$ (see the Appendix for the definition of β).

Sample input images and final renderings of the reconstructed point models are shown in Figures 1 and 7. The Dubrovnik dataset, with more than 6,000 images, covers an entire old city. Piazza San Marco is our largest dataset, with nearly $14,000$ images. Table 1 provides some statistics, i.e., the number of input images, the number of images after image selection and the total number of images in the final clusters (including multiple counting of images appearing in multiple clusters). Note that the image selection discards nearly 90% of the images for Trevi Fountain, whose scene geometry is relatively simple and view configurations have high overlap. The bottom of the table lists the running time of the three steps of our system. The MVS reconstruction and filtering steps can be parallelized, so we provide running time for both serial and parallel executions (assuming one node per cluster), with the latter numbers inside parentheses. Note that we simulate the parallel execution with a set of processes run serially on a single node in our experiments, with timings recorded for each process, and their maximum being the running time of the parallel execution for each step. The quality filter typically takes five to tens time longer than the visibility filter. Note that the numbers of points in the final rendering are nine times the numbers of points after the MVS filters, due to upsampling (supplementary material), yielding nearly 130,000,000 points for Dubrovnik and 250,000,000 points for Piazza San Marco.

Table 1. Statistics. See text for details.

Dataset		Basilica	Trevi	Colosseum	Dubrovnik	San Marco
# of images	Input	1275	1936	1167	6304	13709
	After image selection	298	212	490	2097	3535
	Sum over output clusters	333	218	528	2628	5917
# of clusters		4	2	7	28	67
# of SFM points	Input	515,259	636,290	483,621	483,621	4,316,202
	After SFM filter	14,619	7,467	20,958	68,023	197,658
# of MVS points	After PMVS	6,344,656	2,536,790	6,638,779	16,935,255	47,476,039
	After MVS filter	5,107,847	1,892,378	5,747,083	14,051,331	27,707,825
Running time [min]	View clustering	1.3	3.3	1.5	21.1	39.3
	PMVS	305 (84)	165 (84)	232 (45)	1202 (180)	2103 (165)
	MVS filter	39.9 (10.2)	12.6 (6.5)	73.0 (14.2)	474.6 (41.4)	666.5 (11.3)



Trevi Fountain Dubrovnik

Figure 8. For each pair, the left and right figures show reconstructed MVS points from mutually exclusive clusters and overlapping clusters found by the proposed algorithm.

Figure 8 illustrates the importance of the overlapping clustering formulation. Reconstructions from mutually exclusive clusters (without the image addition step) suffer from holes and possible loss of accuracy.

To compare our approach against previous work, we use a state-of-the-art depth-map based MVS algorithm by Goesele *et al*. [12], which is designed for Internet community photo collections, to process one of our datasets (using software provided by the authors). Their depth map estimation itself is scalable and parallelizable, while their chosen mesh extraction step (Poisson-based depth map merging [15]) is not.[5] We ran their algorithm on St. Peter's Basilica with an entire image set (1275 images) as well as images reduced by our image selection algorithm (298 images) (see Figure 9).

[5] Poisson-based merging can be run out-of-core [3], but not in parallel.

Figure 7. A sample input image is shown at the top for each dataset, followed by final renderings of the reconstructed point model.

As noted in the introduction, depth map MVS reconstructions tend to be noisy, and mesh extraction software [15] is essential for filtering out the noise. Goesele's method benefits from our image selection step, as the surface quality and execution time both improve (342 minutes with image selection, 640 minutes without, on a single node); this result indicates that the image selection step is useful for other MVS algorithms, not just PMVS. Figure 9 also shows shaded renderings of our point model and a surface mesh extracted by the same meshing software [15]. While the shaded point rendering is much noisier than the shaded mesh rendering (due to noisy normal estimates by PMVS), the *colored* point renderings yield high quality image-based rendering visualizations, as seen in Figures 1 and 7.

Our last experiment is to observe the running time of our system with different values of α (the upper bound on the cluster size) on both serial and parallel executions (the right of Figure 9). As α decreases, the number of view clusters increases, as does the amount of overlap, leading to some redundant effort and very slow serial execution. In contrast, the parallel execution becomes faster due to distributed computation. One interesting observation, however, is that when α becomes too small (when $\alpha = 30$), parallel execution becomes slower, likely due to excessive overlap, resulting in many redundant MVS points that are reconstructed and then processed by the MVS filters.

5. Conclusion

We have developed a novel MVS system for very large unorganized photo collections. Our system is the first to demonstrate an unorganized MVS reconstruction at city-scale. A key technical contribution is in the view clustering algorithm, which divides an image set into overlapping view clusters, after which an MVS algorithm can be used to reconstruct each cluster in parallel. We have also presented MVS filtering algorithms to handle quality variations and to enforce inter-cluster consistency constraints over the entire reconstruction; these algorithms are designed to run out-of-core and in parallel for scalability. Finally, using a point-based rendering algorithm, we demonstrate visualizing large architectural settings, automatically reconstructed from online photo collections.

Acknowledgments: This work was supported in part by National Science Foundation grant IIS-0811878, SPAWAR, the Office of Naval Research, the University of Washington Animation Research Labs, and Microsoft. We thank Sameer Agarwal and Noah Snavely for the datasets, Szymon Rusinkiewicz for Qsplat software, and Michael Goesele for the multi-view stereo software.

References

[1] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In *ICCV*, 2009.

[2] A. Banerjee, C. Krumpelman, J. Ghosh, S. Basu, and R. J. Mooney. Model-based overlapping clustering. In *SIGKDD*, pages 532–537. ACM, 2005.

[3] M. Bolitho, M. Kazhdan, R. Burns, and H. Hoppe. Multilevel streaming for out-of-core surface reconstruction. In *Symp. Geom. Proc.*, pages 69–78, 2007.

[4] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, pages 303–312, 1996.

[5] I. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *PAMI*, 29(11):1944–1957, 2007.

Figure 9. Left: A depth-map based MVS algorithm by Goesele *et al*. [12] is used to reconstruct mesh models, one from an entire image set and the other from images reduced by our image selection step. We also show shaded renderings of our point-set and mesh models extracted by the meshing software [15]. Right: Running time of our system on serial and parallel executions for different values of α.

[6] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. CMVS. `http://grail.cs.washington.edu/software/cmvs`.

[7] Y. Furukawa and J. Ponce. PMVS. `http://grail.cs.washington.edu/software/pmvs`.

[8] Y. Furukawa and J. Ponce. Accurate camera calibration from multi-view stereo and bundle adjustment. *IJCV*, 84(3):257–268, 2009.

[9] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. *PAMI*, 2009.

[10] D. Gallup, J. M. Frahm, P. Mordohai, and M. Pollefeys. Variable baseline/resolution stereo. In *CVPR*, 2008.

[11] D. Gallup, J. M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. In *CVPR*, 2007.

[12] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. M. Seitz. Multi-view stereo for community photo collections. In *ICCV*, 2007.

[13] C. Hernández Esteban and F. Schmitt. Silhouette and stereo fusion for 3D object modeling. *CVIU*, 96(3):367–392, 2004.

[14] M. Jancosek, A. Shekhovtsov, and T. Pajdla. Scalable multi-view stereo. In *3DIM*, 2009.

[15] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symp. Geom. Proc.*, 2006.

[16] B. Kulis and Y. Guan. Graclus software. http://www.cs.utexas.edu/users/dml/Software/graclus.html.

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

[18] B. Micusik and J. Kosecka. Piecewise planar city modeling from street view panoramic sequences. In *CVPR*, 2009.

[19] M. Pollefeys et al. Detailed real-time urban 3d reconstruction from video. *IJCV*, 78(2-3):143–167, July 2008.

[20] J.-P. Pons, R. Keriven, and O. Faugeras. Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score. *IJCV*, 72(2):179–193, 2007.

[21] S. Rusinkiewicz and M. Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH*, 2000.

[22] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. *CVPR*, 1:519–528, 2006.

[23] J. Shi and J. Malik. Normalized cuts and image segmentation. *PAMI*, 22(8):888–905, 2000.

[24] G. Vogiatzis, P. H. Torr, and R. Cipolla. Multi-view stereo via volumetric graph-cuts. In *CVPR*, pages 391–398, 2005.

[25] C. Zach, T. Pock, and H. Bischof. A globally optimal algorithm for robust tv-l^1 range image integration. In *ICCV*, 2007.

Appendix A: Measuring MVS Accuracy

$f(P, C)$ measures the MVS reconstruction accuracy at a 3D location P achieved by a set of images C. Two geometric factors are taken into account: baselines and pixel sampling rates. We first define an MVS reconstruction accuracy for a pair of images (I_l, I_m) at P:

$$f'(P, I_l, I_m) = g(\angle I_l P I_m) \cdot \min\left(1/r(P, I_l), 1/r(P, I_m)\right).$$

A baseline is measured by the angle $\angle I_l P I_m$ between two viewing rays emanating from P towards the two camera centers. We apply a Gaussian function to the angle: $g(x) = \exp\left(-\frac{(x-20)^2}{2\sigma_x^2}\right)$, whose peak is at 20°. We set $\sigma_x = 5^\circ$ for $x < 20^\circ$ and $\sigma_x = 15^\circ$ for $x > 20^\circ$. The pixel sampling rate of image I_l at P is defined as $1/r(P, I_l)$, where $r(P, I)$ is the diameter of a sphere centered at P whose projected diameter equals one pixel in I. Finally, an accuracy measure $f(P, C)$ for a set of images C is defined as

$$f(P, C) = \max_{p \in \mathbf{T}(C)} \sum_{I_l, I_m \in p} f'(P, I_l, I_m). \qquad (1)$$

$\mathbf{T}(C)$ denotes a set of every combination of β images in C, where β is set to $\min(4, |C|)$ in our experiments. For each combination, an accuracy is evaluated as a sum of pairwise functions f', and the maximum over all the combinations is set to $f(P, C)$. We choose this function because many MVS algorithms, including PMVS, which is used in our experiments, use only a few images in reconstructing each point (or surface region) for computational efficiency.

The evaluation of (1) can be very expensive for a large image set C. Instead of exhaustively testing every combination, we greedily construct a set of β images that approximately maximizes the score as follows. After initializing C by a pair of images that has the highest pairwise score f', we greedily add an image to C that maximizes $f(P, C)$ one by one (See [12] for a similar technique used for speed up).