HyperSfM

Kai Ni* and Frank Dellaert†
*Microsoft Corporation, Redmond, WA 98052, USA
†Georgia Institute of Technology, Atlanta, GA 30332, USA

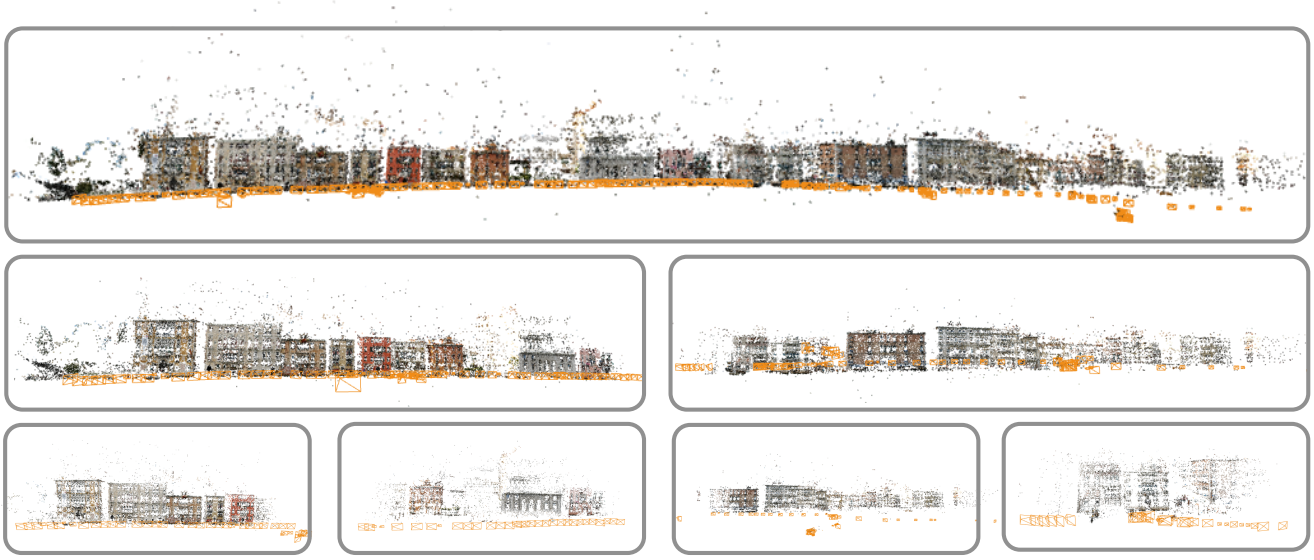

Figure 1. *We introduce HyperSfM, a divide-and-conquer approach for Structure from Motion. We represent an SfM problem by a hypergraph which we recursively partition, obtaining a tree of fully-constrained, nonlinear sub-problems. The top three levels are shown above for the " Grand Canal" data-set [1]. After partitioning, computation proceeds from the bottom up, recursively merging submaps.*

Abstract

We propose a novel algorithm that solves the Structure from Motion problem in a divide and conquer manner by exploiting its bipartite graph structure. Recursive partitioning has a rich history, stemming from sparse linear algebra and finite element methods, and are also appealing for solving large-scale SfM problems. However, an important and less explored question is how to generate good partitionings for SfM that divide the problem into fully-constrained sub-problems. Here we introduce HyperSfM, a principled way to recursively divide an SfM problem using a hypergraph representation, in which finding edge separators yields the desired "nested-dissection" style tree of nonlinear sub-problems. After partitioning, a bottom-up computation pass solves the SfM problem robustly (by having fully constrained sub-problems) and efficiently (because most nonlinear error is removed at lower levels of the tree). The performance of the algorithm is demonstrated for various indoor and outdoor standard data-sets.

I. Introduction

Large-scale structure from motion (SfM) problems have gained more and more attention lately, as SfM is becoming one of the key technologies in applications such as city-scale 3D reconstruction. A lot of effort has been made to push SfM algorithms towards collections of a large number of photos [2], [3]. In this paper we concentrate on the back-end optimization phase, after feature extraction and data association has been performed, which are daunting problems in their own right [1], [4].

In photogrammetry, divide and conquer approaches are a common and popular way to "bundle" data from a large area [5]. When images are taken sequentially from a plane or a ground vehicle, a feasible way to tackle the problem is to solve the subproblem within a relative coordinate system rather than a consistent global coordinate system [6]. However, this approach is decidedly suboptimal when there are a lot of "loop closures" in the camera trajectory, which is typically the case in unstructured photo-collections. Moreover, for such unordered, wide-baseline data-sets we

typically do not have knowledge of the capture ordering, making this approach unsuitable.

Another important problem worth investigating is how to avoid degeneracies when generating submaps. General partitioning algorithms [7] do not take into account the domain knowledge of SfM problems. Hence, directly applying those algorithms will easily introduce degeneracies to the state variables, especially 3D points, as each 3D point is typically only visible in a small number of cameras (two or three in practice). In fact, little work has been done on how to optimally divide the SfM problem while keeping all the individual sub-problems fully constrained.

In this paper, we propose a principled way to partition the SfM problem. We exploit the bipartite structure of the SfM visibility graph and convert it to a simplified camera *hypergraph*. It is shown that vertex separators composed of only 3D points can be located from the hypergraph, and non-singularity can be strictly enforced by imposing a graph refinement step after partitioning.

Our algorithm is not only out-of-core but also naturally alleviates the initialization issue of bundle adjustment in SfM problems [8], [9]. We employ a bottom-up optimization using the submap tree obtained by recursive partitioning. The optimization over different subtrees in the same level can be carried out in parallel. The optimized submaps are aligned to one another after passing information up the tree to their common ancestor. As a beneficial consequence we never need to generate the initialization for the entire large-scale problem.

In the results section, we demonstrate the effectiveness of our algorithm using several indoor and outdoor data sets from Microsoft's PhotoSynth database. We show that the proposed partitioning scheme effectively decouples original problems, and that the resulting sub-problem structure greatly speeds up the bottom-up bundle adjustment phase.

II. Background and Related Work

A. Related Work

Divide-and-conquer methods to efficiently solve large-scale bundle-adjustment problems have long been favored in the photogrammetry community [5], [8]. In structure from motion problems, which are typically much more unstructured, dividing the entire problem into small pieces not only makes the bundle adjustment optimization more scalable, but also provides an easier way to generate good initializations, as has already been demonstrated to be a crucial capability for SfM problems [9]. Our work is related to [10], [11] in terms of hierarchical optimization, but ours does divide-and-conquer at the variable level, while most other work operates at the image level.

The divide-and-conquer idea for continuous optimization was first explored in the linear algebra community under the name "nested dissection" [12]. Lipton, Rose, and Tarjan subsequently showed that, for certain classes of graphs (e.g., planar), separator theorems exist that enable one to obtain theoretical guarantees on the worst case computational complexity [13]. In detail, the nested dissection (ND) algorithm recursively partitions the graph of the original problem and finds a *vertex separator* V_S which splits the graph into two parts A and B, such that there are no connections between any node in A and any node in B. More recently, Krauthausen et al. [14] applied the separator theorems of the nested dissection algorithm to large-scale urban mapping problems in robotics.

We have previously investigated a graph-based approach for SfM [15], introducing an out-of-core SfM algorithm which uses submaps to leverage a computational advantage. However, in that approach we only used a single-level submap structure, and the main bottleneck is solving the dense linear system corresponding to a large separator. Moreover, we did not explicitly handle degeneracies in the submaps. In this paper, we introduce a recursive partitioning approach to produce submaps of very small size, and also address the degeneracy problem by working on a hypergraph representation. Note that we have explored a hierarchical optimization scheme in robotics[16], but it only applies only to solving 2D reconstruction problems: due to the same degeneracy issue discussed above, is not suitable for solving SfM problems.

B. Problem Formulation

In SfM [17], we infer the structure of the scene and the motion of the camera by using the correspondences between features from different views. In particular, certain types of features [18] (points, lines, and so forth) are first extracted and matched across all the image pairs. Then the camera parameters and feature locations are optimized to minimize a cost function, such as the 2D projection errors.

The non-linear minimization of the projection errors is referred to as *bundle adjustment* in the literature [8], in which we jointly estimate the optimal 3D structure, as well as the camera parameters by minimizing a least-squares cost function. Typically, the measurement function $h_k(.)$ is non-linear, and one assumes a normally distributed measurement noise with associated covariance matrix Σ_k, leading to

$$\sum_{k=1}^{K} \|h_k(C_{i_k}, P_{j_k}) - z_k\|^2_{\Sigma_k} \tag{1}$$

Above, $C_i (i \in 1...M)$ represents the intrinsic and extrinsic camera calibrations, $P_j (j \in 1...N)$ represents the 3D structure, and $z_k (k \in 1 \dots K)$ represents the 2D measurement of the point P_{j_k} in camera C_{i_k}. The notation $\|.\|^2_\Sigma$ stands for the squared Mahalanobis distance with covariance matrix Σ.

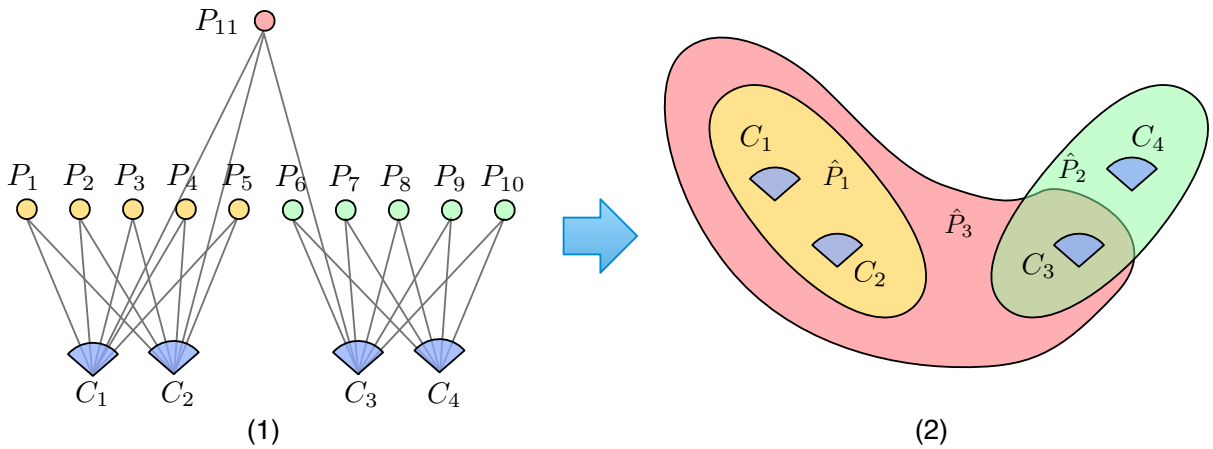

Figure 2. **The visibility graph of an exemplar SfM problem on the left is converted to the corresponding hypergraph representation on the right**. *The cameras are shown in blue, and the 3D points are shown in yellow. Each edge in the left graph indicate that a 3D point is visible from a certain camera. The contours in the right graph represent the hyperedges in the hypergraph, and each contour connects multiple cameras. Note that the colors of 3D points in the visibility graph share the same colors as the corresponding hyperedges in the hypergraph. For simplicity, we ignore any singularity problem in the toy example.*

C. The Bipartite Visibility Graph

With every SfM we can associate a *visibility graph*, i.e., the bipartite graph $G_{\text{SfM}} = (C, P, E)$ where the sets of cameras C and points P appear as vertices, and there is an edge e_{ij} corresponding to every measurement z_k, indicating that point P_j is visible in camera C_i. A small example is shown in Figure 2.(1). Note that dividing the original SfM problem is equivalent to partitioning the corresponding visibility graph. Generally speaking, we desire that the sub-problems have similar sizes, and each sub-problem is also self-contained, i.e. non-singular.

III. HyperSfM

Here we introduce our main contribution, HyperSfM, which consists of three major parts:

1) A hierarchical partitioning based on hypergraphs.
2) A refinement step that deals with degeneracies.
3) A bottom up optimization step that merges submaps.

A. Hierarchical Partitioning

Our algorithm works by finding a small edge separator in the *camera hypergraph*, which corresponds to a vertex separator in the original visibility graph consisting solely of 3D points. A *hypergraph* $\mathcal{H} = (\mathcal{X}, \mathcal{E})$ is composed of a set of vertices $\mathcal{X}$ and a set of hyperedges $\mathcal{E}$, where each hyperedge connects multiple vertices. We define the camera hypergraph $\mathcal{H}_{\text{cam}} = (C, P)$ of an SfM problem as the hypergraph obtained by treating the cameras C as vertices $\mathcal{X}$ and the 3D points P as hyperedges $\mathcal{E}$.

To reduce the complexity of hypergraph $\mathcal{H}_{\text{cam}}$ and speed up later graph operations, we introduce *compressed edges*. In SfM problems, it is typical that two or more images perceive the same cloud of 3D points, e.g. points on a building facade. Those 3D points correspond to the edges in $\mathcal{H}_{\text{cam}}$ that connect to the same camera vertices. Hence, it is straight-forward to introduce compressed edges and generate a much simplified graph $\hat{\mathcal{H}}_{\text{cam}} = (\mathcal{X}, \hat{\mathcal{E}})$. For example, points P_1 to P_5 in Figure 2.(1) all connect to C_1 and C_2, so they will be combined to a compressed edge $\hat{P}_1$ in $\hat{\mathcal{H}}_{\text{cam}}$, as shown in Figure 2.(2). We also specify the weights of the resulting hyperedges using the number of 3D points each compressed edge represents, e.g. $w_{\hat{P}_1} = 5$.

We partition the hypergraph by finding a small edge separator $\hat{\mathcal{E}}_S$ (defined below). This hypergraph partitioning is done recursively, which leads to a tree structure as shown in Figure 3. Note that the partitioning is general regardless of graph types (e.g. loopy or not). The recursive

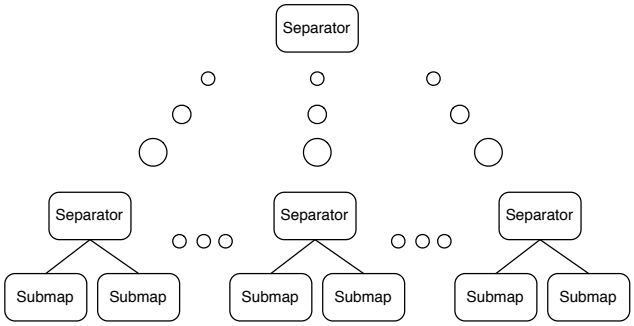

Figure 3. **By partitioning the hypergraphs and finding vertex separators in the visibility graph, the original SfM problem can be partitioned recursively**. *The resulting tree structure has submaps on its leaves and has separators along the path from the leaves to the root. Note that two subtrees are independent given their common ancestor on the tree.*

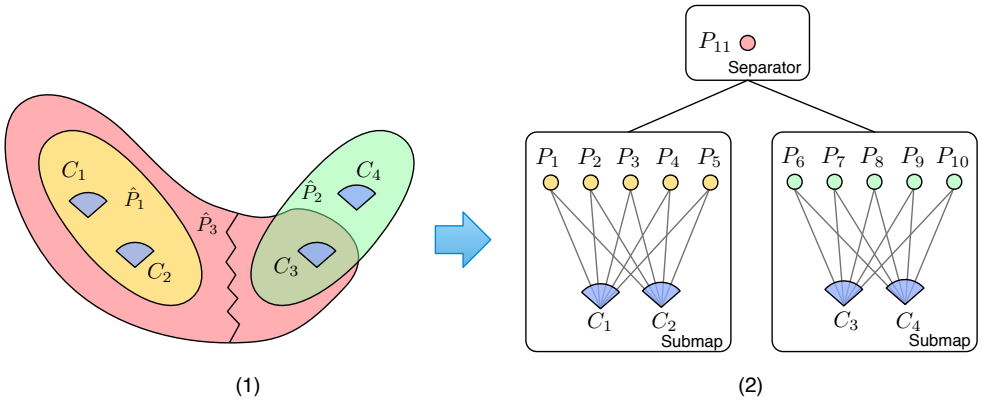

Figure 4. **Partitioning a hypergraph for a SfM problem (left) using an edge separator is equivalent to finding a vertex separator composed solely of 3D points in the original visibility graph (right)**. *The edges in the hypergraph are weighted according to the number of 3D points they correspond to, and $\hat{P}_3$ is chosen as the edge separator here because it has the smallest weights. The two resulting submaps are independent to each other given their vertex separator and can be optimized in an out-of-core manner.*

partitioning stops when the size of the current submap is smaller than a certain threshold (5000 for all our experiments). Note that the sizes of the separator are typically very small (details will be described in the result section) In fact, although there are no theoretical guarantees for the separator size in SfM problems, we found we can in practice always find good separators by exploiting the underlying structure of SfM.

An *edge-separator* $\hat{\mathcal{E}}_S$ of a hypergraph $\hat{\mathcal{H}}_{\text{cam}}$ is a subset of its edges $\hat{\mathcal{E}}$ that disconnects $\hat{\mathcal{H}}_{\text{cam}}$ into two or more separate connected components $\hat{\mathcal{H}}_A = (C_A, \hat{P}_A), \hat{\mathcal{H}}_B = (C_B, \hat{P}_B) \ldots$. Because of the definition of camera hypergraph $\hat{\mathcal{H}}_{\text{cam}}$, any edge separator $\hat{\mathcal{E}}_S$ in $\hat{\mathcal{H}}_{\text{cam}}$ automatically corresponds to a subset P_S of the 3D points in the original visibility graph G_{SfM}. For example, in Figure. 4, edge separator $\hat{\mathcal{E}}_S = \hat{P}_3$ corresponds to the 3D point set $P_S = \{P_{11}\}$ from the visibility graph.

Theorem 1. *If $\hat{\mathcal{E}}_S$ is the edge separator of hypergraph $\hat{\mathcal{H}}_{\text{cam}}$, and P_S is the set of its corresponding 3D points in the visibility graph G_{SfM}, we have that P_S is the vertex separator of G_{SfM}, disconnecting the visibility graph $G_{\text{SfM}} = (C, P, E)$ in two or more components $(C_A, P_A, E_A), (C_B, P_B, E_B), \ldots$*

Proof. The cameras in C_A and C_B in G_{SfM} are not connected, because G_{SfM} is bipartite. But moreover, no point in P_A is visible from any camera in C_B. To see this, assume there is point $P_1 \in P_A$ connected to a camera $C_1 \in C_B$. Because (C_A, P_A, E_A) is a connected component (by definition), P_1 is also connected to some camera, say C_2 in C_A. But then P_1 must be in the edge separator $\hat{\mathcal{E}}_S = P_S$, and not in P_A, which is a contradiction. QED.

B.Degeneracy Issues and Graph Refinement

To be able to optimize the partitioned submaps separately, we need to make sure that each submap is well constrained. That is to say, each landmark is visible in at least n cameras, and each camera perceives at least m landmarks. For example, in a SfM problem where camera calibrations are known, we have $m = 5$ and $n = 2$. Combining both requirements, we call such a partitioning a constrained (m, n)-*cut*. Note that the cut itself does not guarantee the non-singularity of a submap. However, we found that it is typically the case, otherwise more advanced pruning methods similar to [19] can be utilized.

In the partitioning algorithm we introduced above, we explicitly choose a vertex separator P_S consisting solely of 3D points. The main reason for doing this is that it prevents the 3D points in the resulting submaps $(C_A, P_A, E_A), (C_B, P_B, E_B), \ldots$ from becoming singular. In SfM problems, the 3D points are typically only visible to a small number of cameras; if one or more of their cameras end up in the vertex separator after the partitioning, the related 3D points can easily become under-constrained in their submaps. Hence, to generate a valid (m, n)-cut, we try to use only 3D point vertices as the separator for G_{SfM}. In this case, only cameras in the submaps "lose" constraints if their observable 3D points are part of the separator. As each camera usually sees a lot of 3D points, losing some of those constraints because of vertex separator P_S usually does not make the camera singular. Also note that *all* the 3D points in the resulting submaps after partitioning remain fully constrained, as their connections to the neighboring cameras stay the same.

We employ a partition refinement step to enforce non-singularity and handle the rare case that some cameras become singular after we split the graph using a certain vertex separator. In graph theory, graph refinement refers to the local improvement approaches such as the widely-used KL refinement by Kernighan and Lin [20] and the algorithm by Fiduccia and Mattheyses [21]. In this paper, given hypergraph partitioning results, we locate the under-constrained

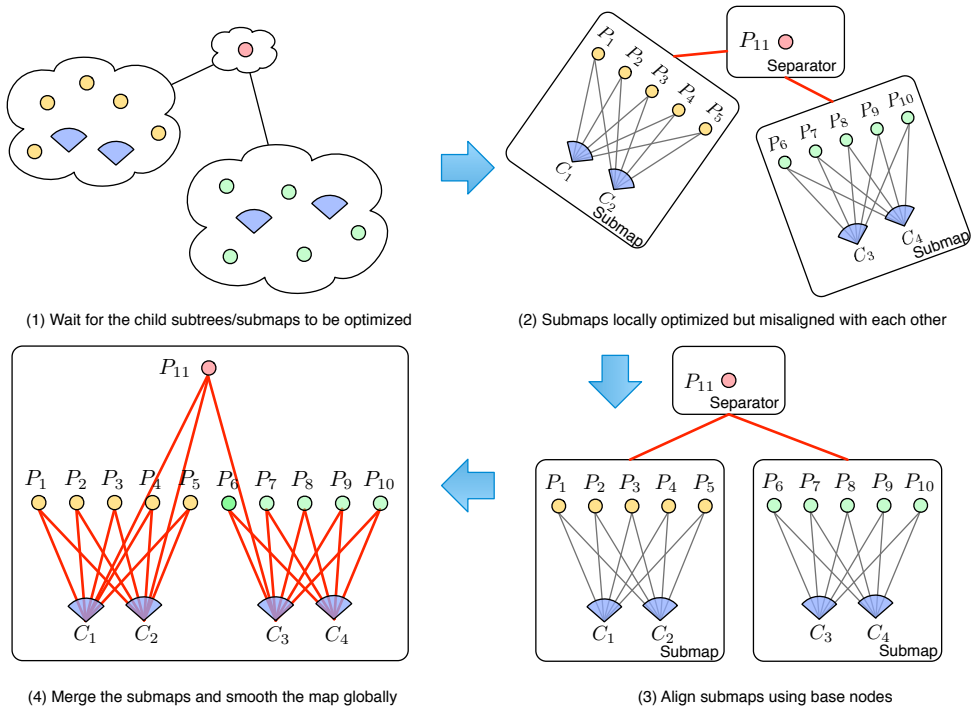

Figure 5. **The bottom-up optimization is carried out recursively.** *The red edges indicate the constraints used in each optimization. Readers may refer to [15] for more details of optimization using base nodes.*

cameras in submaps$(C_A, P_A, E_A), (C_B, P_B, E_B), \ldots$ and put them into the separator P_S. Up to this point, we also need to check all the affected 3D points in $P_A, P_B, \ldots$ and put the under-constrained ones to P_S as well. We iterate over cameras and 3D points until all the vertices in the submaps are fully constrained. For all the data sets we tested in this paper, it took at most two iterations, which makes graph refinement efficient.

C.Bottom-up Optimization

Given the tree structure after recursive partitioning, we employ a bottom-up process to optimize and merge all the submaps into a final global reconstruction. The process is inherently recursive (illustrated in Figure 5): for each subtree, the separator waits for its children to complete their own optimization. All the optimized child submaps are then aligned with each other as *rigid* maps. At last, all the child submaps as well as their separators are optimized together in the unified coordinate system. Such a process is carried out for each separator, and the entire SfM problem is solved in a bottom-up fashion.

The bottom-up process can be done efficiently in an out-of-core manner. Note that the multi-level tree structure induces more submaps with smaller sizes compared to the single-level submap based approaches such as [15], and this enables us to distribute the computation to more cores.

The rigid alignment of submaps is achieved using base nodes, as shown in step (3) of Figure 5. Each submap is assigned a base node, which represents the 6DoF transformation between the submap and its separator. Such an alignment is fast because only several base node variables as well as a small separator are involved. In fact, the number of base nodes under the same separator is typically two or three, and the sizes of the separators are also much smaller compared to the size of the original problem (the details will be presented in the results section).

The initialization problem on each level of the tree is sidestepped by integrating the optimized submaps from the children. It has been shown that a reasonably good initialization is crucial for the final convergence of SfM problems [8], [9]. In large-scale SfM problems, the initialization is more difficult due to errors accumulated in local reconstructions, e.g. a maximum spanning tree of pair-wise reconstructions, greatly reduce the overall initialization quality. By splitting the original data into many smaller parts, the initialization in the global level is avoided. Instead, small maps propagate their optimized states to the parent separator, where that information is integrated together by submap alignment. In this way, more reliable initialization is achieved from the bottom up, which makes the algorithm more robust.

The bottom-up optimization proposed here is also exact. A smoothing step [22] is executed for all the subtrees including the entire tree when reaching the root level (step

	$\lvert P_S\rvert$ / $\lvert G_{\text{SfM}}\rvert$	Nr. Submap	Time (sec.)
Brown House	2.48%	2	0.57
Old House	1.61%	3	1.28
Grand Canal	0.99%	2	3.12
San Marco	12.5%	3	3.71
St. Peters	4.00%	2	5.10

Table I. **The partitioning results for the five data sets**. $|P_S|$is the number of vertices in the root separator P_S, and $|G_{\text{SfM}}|$ is the total number of vertices in the original problems. The second column indicates the number of submaps after the first-level partitioning. The timing results in the last column is the total time cost of the entire recursive partitioning.

(4) in Figure 5), which improves the quality of the most recent subtree estimate and supplies an initialization for the parent level. More importantly it guarantees the exactness of the current subtree and is the same as regular bundle adjustment with *all* variables involved. For levels other than the root level, the optimization does not need to fully converge, as its results only serve as the initialization for the next level. In our experiments, we limit the number of iterations for all subtree smoothing to seven except for the root level, where we use the same convergence criteria as we would for regular bundle adjustment.

IV. Experimental Results

A. Hypergraph Partitioning

We demonstrate the hypergraph partitioning using five indoor and outdoor data-sets (Brown House, Old House, Grand Canal, San Marco, St. Peters) from the PhotoSynth database, as shown in the first column of Figure 6. After generating the camera hypergraph $\hat{\mathcal{H}}_{\text{cam}}$, we use the Metis graph partitioner [7] (the default settings are used in our experiments) to find the edge separators, which are shown in red in the last two columns of Figure 6. All the resulting first-level submaps are shown in other colors. To make the comparisons fair, we initialized both BA and HyperSfM using the same pairwise reconstruction results from the PhotoSynth implementation.

First, we evaluate how well our algorithm is able to divide SfM problems. As listed in Table I, most of the data sets have a separator smaller than 5% of the total data size. This means that given a small part of the points and cameras (mostly points), the entire 3D world can be decoupled into two or three submaps *without* discarding any measurements from the original problem.

We investigated both indoor and outdoor scenarios, and the characteristics of the data sets vary from one to another. For the Grand Canal, there is no loop in the camera trajectory, hence the partitioning is straightforward: the set is split into two parts along the camera trajectory. In Brown House, Old House and St. Peter data sets, there are loops in the trajectory, but the cameras do not have many 3D objects in common. Hence the separator size is still very small. On the other hand, in the San Marco data set, the separator size is bigger as most of pictures were taken along the direction of St Mark's basilica and Campanile town, hence there are far more overlapping between camera views, which leads to a bigger separator. Note that overall it is still a small portion of the original data.

The partitioning on the hypergraph is very efficient, as shown in the last column of Table I. For all five data sets, the longest execution time is about five seconds. Compared to the total SfM timing shown in the next section, the overhead caused by partitioning is less then 1%, hence it can often be neglected in the whole SfM pipeline.

	Cameras	BA(sec.)	HyperSfM(sec.)
Brown House	61	725	456
Old House	178	1279	789
Grand Canal	270	3237	1553
San Marco	237	N/A	1465
St Peters	285	N/A	1823

Table II. **The timing results for the five data sets**. BA indicates our own implementation of regular bundle adjustment, which uses AMD ordering [23] to solve the linear systems. Note that HyperSfM uses exactly the same implementation as regular bundle adjustment to optimize the individual submaps.

Level	Submap Alignment	Subtree Smoothing
1st	6.0 iter., 0.48sec.	3.0 iter, 186.40 sec.
2nd	6.5 iter., 0.72 sec.	3.5 iter., 28.08sec.
3rd	8.2 iter., 1.28 sec.	4.3 iter., 7.67 sec.
4th	8.9 iter., 1.50 sec.	4.5 iter., 3.45 sec.
5th	N/A	6.3 iter., 0.78 sec.

Table III. **The timing results for optimizing submaps in Grand Canal data-set on each level of bottom-up optimization**. Results are averaged over all the operations that happen at the save level. In the first column, the two numbers are the average nonlinear iteration numbers and the average time per iteration for aligning the child submaps with respect to each separator. In the second column, the numbers are the corresponding iteration numbers and time per iteration for smoothing each submap. At the 5th level, submap alignment results are not available because there are no child submaps at this level.

B. SfM Timing Results

In this section, we measure and compare the timing results of the regular bundle adjustment algorithm and the proposed HyperSfM algorithm. All the experiments were carried out on a 2.8GHz Intel Core 2 Duo machine with 8GB memory. We used the same feature correspondences as inputs to both algorithms, and we also made HyperSfM use the same implementation and settings as regular bundle adjustment when doing the nonlinear optimization for all the submaps and the final global reconstruction. Hence, the differences in the timing results are only due to HyperSfM's divide-and-conquer scheme.

In Table II, we can observe that solving the decoupled problems yields a great improvement in terms of both

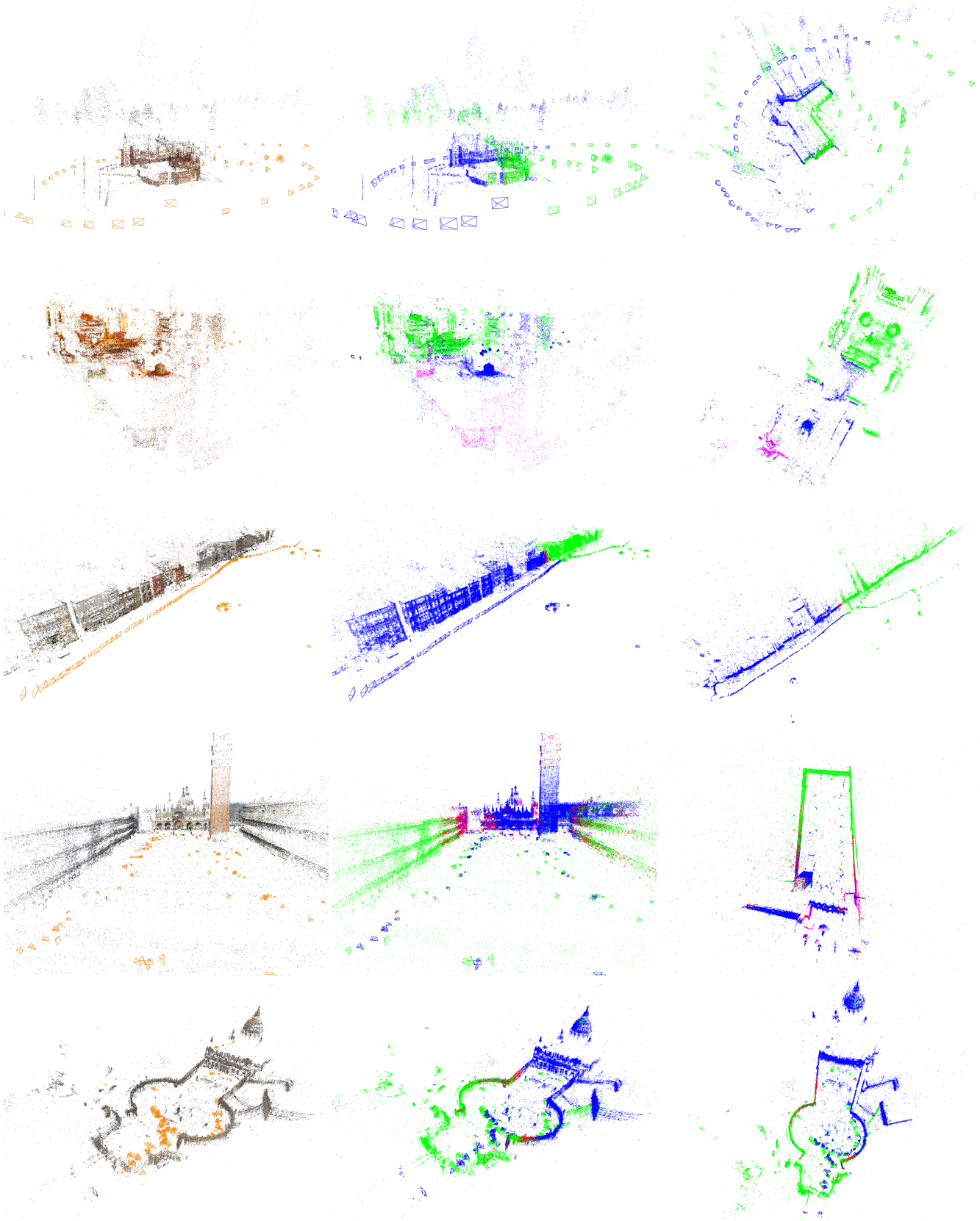

Figure 6. **The partitioned results for the five data sets** *(from top to bottom: Brown House, Old House, Grand Canal, San Marco, and St. Peters). The first column show the cameras and the point clouds in their original color, and the last two columns are the front view and the top view of the root separator (labelled in red) and the first-level submaps (labelled in the other colors).*

speed and robustness. For the Grand Canal data-set and the St. Peters data set, regular sparse bundle adjustment does not converge properly because the initialization we obtained from pairwise reconstructions is very noisy. On the other hand, the proposed HyperSfM approach behaviors more robustly and always converged successfully, benefited from the bottom-up initialization using submaps. For the easier data sets (Brown House, Old House, and Grand Canal), HyperSfM shows 37% to 53% speed improvement over the regular bundle adjustment algorithm.

We also investigated the behavior of bottom-up optimization at each level in the tree structure. As listed in Table III, we observed that submap alignment is much faster than the smoothing step even with more iterations on average. This is because submap alignment only involves the separator variables and several base-node variables, which are far fewer than the number of the variables in the entire subtree. The submap alignment gets a little more expensive for the low-level submaps, because submaps are less decoupled from each other at those levels. This is easily explained, as the partitioning algorithm will always choose the best vertex separator at each level, and hence the cost of partitioning, i.e. the number of edges it cuts, increase with each successive level in the tree.

HyperSfM also saves time by optimizing small lower-level submaps, thereby obviating the need for many iterations on the global reconstruction. For example, regular bundle adjustment takes 17 iterations to converge on the Grand Canal data-set. On the other hand, HyperSfM only takes 3 iterations on the same level of the global reconstruction (the first row in Table III), because most of nonlinear error has been removed during the low-level submap optimization. Note that HyperSfM runs bundle adjustment in each level, but the numbers of variables involved are much smaller in the lower levels. Even though HyperSfM spends part of time on the bottom-up optimization, it certainly affords great time-savings overall. Note that the three iterations spent in the smoothing step also guarantee the same exactness as regular bundle adjustment, which is another desired feature of the proposed algorithm.

V. Conclusions

We proposed a principled way to recursively partition SfM problems into small sub-problems based on a hypergraph representation, and applied a bottom-up optimization to exploit the tree structure so-obtained. This has been shown to have superior efficiency and scalability compared to traditional approaches. Furthermore, HyperSfM enables us to solve sub-problems of small size, which is demonstrated to be much more robust than directly working on original problems. The future work includes implementing a distributed system that runs over multiple threads and fully exploits the potential of the out-of-core optimization.

References

[1] N. Snavely, S. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3D," in *SIGGRAPH*, 2006, pp. 835–846.

[2] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski, "Bundle adjustment in the large," in *Eur. Conf. on Computer Vision*, 2010.

[3] Y. Jeong, D. Nister, D. Steedly, R. Szeliski, and I. Kweon, "Pushing the envelop of modern methods for bundle adjustment," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[4] J. M. Frahm, P. Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y. H. Jen, E. Dunn, B. Clipp, S. Lazebnik, and M. Pollefeys, "Building rome on a cloudless day," in *Eur. Conf. on Computer Vision*, 2010.

[5] D. C. Brown, "The bundle adjustment - progress and prospects," *Int. Archives Photogrammetry*, vol. 21, no. 3, 1976.

[6] G. Sibley, C. Mei, I. Reid, and P. Newman, "Adaptive relative bundle adjustment," in *Robotics: Science and Systems (RSS)*, 2009.

[7] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint graph partitioning," in *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*. Washington, DC, USA: IEEE Computer Society, 1998, pp. 1–13.

[8] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment – a modern synthesis," in *Vision Algorithms: Theory and Practice*, ser. LNCS, W. Triggs, A. Zisserman, and R. Szeliski, Eds. Springer Verlag, Sep 1999, pp. 298–375.

[9] C. Engels, H. Stewenius, and D. Nister, "Bundle adjustment rules," in *Symposium on Photogrammetric Computer Vision*, Sep 2006, pp. 266–271.

[10] M. Farenzena, A. Fusiello, and R. Gherardi, "Structure-and-motion pipeline on a hierarchical cluster tree," in *Intl. Conf. on 3D Digital Imaging and Modeling*, October 2009.

[11] M. F. R. Gherardi and A. Fusiello, "Improving the efficiency of hierarchical structure-and-motion," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2010.

[12] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, April 1973.

[13] R. Lipton and R. Tarjan, "Generalized nested dissection," *SIAM Journal on Applied Mathematics*, vol. 16, no. 2, pp. 346–358, 1979.

[14] P. Krauthausen, F. Dellaert, and A. Kipp, "Exploiting locality by nested dissection for square root smoothing and mapping," in *Robotics: Science and Systems (RSS)*, 2006.

[15] K. Ni, D. Steedly, and F. Dellaert, "Out-of-core bundle adjustment for large-scale 3D reconstruction," in *Intl. Conf. on Computer Vision (ICCV)*, Rio de Janeiro, October 2007.

[16] K. Ni and F. Dellaert, "Multi-level submap based slam using nested dissection," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[17] S. Ullman, *The interpretation of visual motion*. The MIT press, Cambridge, MA, 1979.

[18] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Intl. J. of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[19] J. Frahm and M. Pollefeys, "RANSAC for (quasi-)degenerate data (QDEGSAC)," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[20] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[21] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *DAC '82: Proceedings of the 19th Design Automation Conference*. IEEE Press, 1982, pp. 175–181.

[22] F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous localization and mapping via square root information smoothing," *Intl. J. of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, Dec 2006.

[23] P. Amestoy, T. Davis, and I. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.