Pushing the Envelope of Modern Methods for Bundle Adjustment *

Yekeun Jeong[†*], David Nister[‡], Drew Steedly[‡], Richard Szeliski[‡], and In-So Kweon[†]
Robotics and Computer Vision Lab. KAIST[†]
Microsoft Research[‡]
ykjeong@rcv.kaist.ac.kr[*]

Abstract

In this paper, we present results and experiments with several methods for bundle adjustment, producing the fastest bundle adjuster ever published. The fastest methods work with the well known reduced camera system and handle the block-sparse pattern arising in the reduced camera system in a natural way. Adapting to the naturally arising block-sparsity allows the use of BLAS3, efficient memory handling, fast variable ordering, and customized sparse solving all at the same time. We present two methods, one using exact minimum degree ordering and block-based LDL solving, and one using block-based preconditioned conjugate gradient, both on the reduced camera system. We show experimentally that the adaptation to the natural block sparsity allows both these methods to perform better than previous ones. Further speed improvements are achieved by the novel use of embedded point iterations. The embedded point iterations take place inside each camera update step, yielding a higher cost decrease from each camera update step. This is especially true for points projecting far out on the flatter region of the robustifier.

1. Introduction

Bundle adjustment has become an essential part of structure from motion (SfM) and 3D reconstruction, attracting increased interest from the computer vision community. As a result, a number of approaches to bundle adjustment have been proposed in the last decade. These approaches can be divided into two groups: the first focuses on making the bundle adjustment algorithm as efficient as possible, while the second focuses on reducing the size or frequency of invocation of individual bundle adjustments.

Examples of the first group include [22, 14, 15, 17, 4, 2]. [22, 14] explain bundle adjustment and how to implement it. Additionally [22] discusses more theoretical issues, including gauge freedom, inner constraints, and the reliability of parameter estimates. The Levenberg-Marquardt algorithm (LM) [13] has been a popular choice for bundle adjustment. However, the authors of [15] question this choice and show that the dog leg algorithm (DL), which is designed to explicitly use a trust region concept, is a better alternative to LM. In [17], out-of-core bundle adjustment is proposed, which follows a divide-and-conquer approach. With that aspect only, this work may be classified into the other group, but they have an additional contribution: by caching linearizations of submaps for the full separator system, they enable reconstructing a large-scale system, given a good graph cut

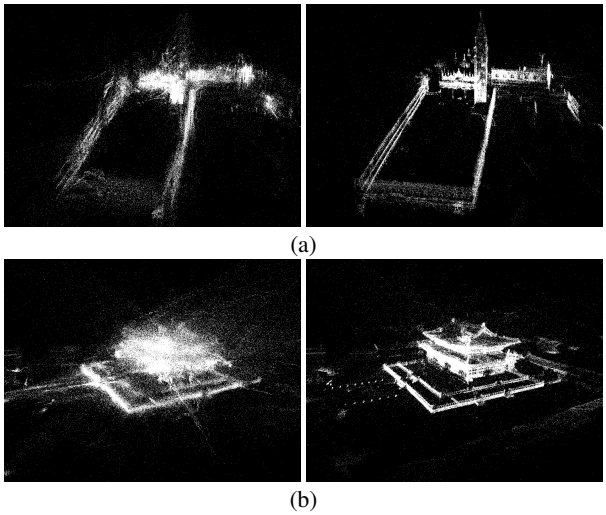

Figure 1. Perturbed(left) and adjusted(right) structures of (a) the **sanmarco** dataset (288 cameras, 144656 points, 468541 projections) and (b) the **korpalace** dataset (1195 cameras, 247500 points, 1035611 projections) by proposed bundler. With the **sanmarco** dataset, our new bundler using block-based preconditioning and embedded point iterations takes 31.8s (45 iterations) and achives 1.05 pixel RMS reprojection error, while the widely used conventional bundler [14] stops at 2.42 pixel error after 376.3s (104 iterations). For the **korpalace** dataset, the conventional bundle adjuster cannot run in 32-bit machine due to a huge amount of memory required. Our new bundler ends up with 0.98 pixel error after 190.9s (56 iterations), thanks to its efficient memory handling.

*This work was done while Yekeun Jeong was an intern at Microsoft Research and supported by Ministry of Knowlege Economy under Human Resources Development Program for Convergence Robot Specialists.

and initialization. Recently there has been an attempt to utilize structural layout of variables for a better preconditioning of Conjugate Gradients (CG) in bundle problems so that CG steps affect the explicit change of parameters more directly [4]. Another recent work in [2] suggested the use of a sparse direct method for Cholesky factorization and a block diagonal preconditioned conjugate gradient.

In the second category, [18, 20, 21, 16, 7, 11, 8] have proposed various approaches to apply bundle adjustment. The authors of [18] recover 3D structure from a long sequence by performing bundle adjustment hierarchically, from segment-wise to global, and also suggest an efficient approach that reduces the number of frames in the global system by introducing virtual key frames. In [20], the authors reduce the redundancies of the brute force bundling by checking which variables need to be optimized after every new frame. [21] proposes a spectral partitioning approach, which divides a large-scale bundle problem into smaller subproblems and preserves the low error modes of the original system. [16, 7] investigate the proper application of the local bundle adjustment. While [16] suggests applying local bundle adjustment after every new keyframe is found, [7] suggests applying it after every added frame. Both of them also analyze what number of recently added frames is optimal for given datasets. [8] recently proposed a method which includes uncertainty propagation and the maximum likelihood estimation of the local bundle adjustment given a particular noise model. In the case of [11], a relative frame representation is introduced; instead of representing cameras in common global coordinates. They use relative motions between cameras combined with a sequential SfM and efficiently loop closing.

Although many previous works have been presented, bundle adjustment is still the primary bottleneck in relevant applications and is problematic in very large-scale reconstructions. In this paper, we present several methods for dramatically improving the performance of the bundle adjustment (i.e., bundler). We exploit the block-sparsisy pattern arising in a reduced camera system by carrying out BLAS3[1] operations, efficient memory handling, and fast block-based linear solving. Furthermore, the novel embedded point iterations (EPIs) considerably improve the computational speed by yielding a high cost decrease from each camera update step and allowing the convergence to appear within a small number of iterations. Our approach also does not require any special assumption or hardware, and is exactly the same as conventional bundlers in terms of functionality and usage in related applications. The fact that it is possible to use our approach in combination with all other previous approaches demonstrates the generality and the applicability of our proposed method.

The rest of the paper is organized as follows. Section 2 briefly explains the conventional bundler. Section 3 discusses the details of how to use the block structure and Section 4 describes our contributions to solving the reduced camera system. The motivation and the implementation of EPIs are in Section 5. We experimentally show that our proposed methods perform substantially faster than previous methods in Section 6.

[1] BLAS3 is a library of matrix-matrix operations

2. Bundle Adjustment

Bundle adjustment (BA) is the problem of refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameter [22] by minimizing the robustified squared sum of all possible reprojection errors. At the outermost layer, BA is carried out using Levenberg-Marquardt (LM) algorithm [13], one of the dampened Newton methods. The LM algorithm assumes the cost function to be locally quadratic and dampens the Hessian by controlling a dampener λ when the function is not fitted well. In our case, to save more computation load, we use a LM variant that differs in how the dampener affects the hessian as in [7]. In general, the computed Jacobian J is a $2 \times (kM + 3N)$ matrix and the resultant Hessian $H = J^T J$ is a $(kM + 3N) \times (kM + 3N)$ matrix, where M, N, k are the number of cameras, points, and the parameters for one camera respectively. This means that one needs to solve $(kM + 3N) \times (kM + 3N)$-sized linear system and this normally takes too much time. Fortunately, in most cases, M is much smaller than N and the Schur complement can be used to reduce the large system to a smaller $kM \times kM$ linear system H_{rc}, the so called reduced camera system (RCS) [22, 10]. For more details on bundle adjustment, please refer to [22, 14, 7]. Our implementation is similar in spirit to [7], including computing outer products and keeping sine values of rotation.

2.1. The Reduced Camera System

The Schur complement transforms the linear solving of H to another linear solving of H_{rc} and a back-substitution. Because the size of H_{rc} only depends on the size of camera parameters, bundle adjustment does not suffer from a very large number of points anymore. Note that if this Schur complement is computed explicitly, many advantages may be lost. Constructing the RCS implicitly is very important and allows faster speed and smaller memory usage [7]. How to manage the structure of RCS and how to solve RCS are explained in Section 3 and 4.

3. Block Structure

The fact that a block-sparse pattern arises naturally in a reduced camera system is very well known [22, 14, 7, 6, 17]. A fair amount of research has already been done on the topic of re-ordering techniques for the cameras, designed to

reduce fill-in during a direct solution, but considerable improvement is still required. We suggest several ways to significantly enhance the efficiency and the speed of the dominant processes such as building a reduced camera system and LM optimization.

It is easier if we treat all camera parameter blocks identically, since this allows a very simple and efficient block solver to be applied to the homogeneous block structure of the reduced camera matrix. However, practical applications of SfM should deal with various unknown cameras at the same time, as in [19, 1, 2]. A mixed set of cameras containing partially known, fully known and unknown intrinsics may be an important case. Fixed parameters need to be omitted, and the parameters shared by several cameras need to be joined. This breaks the homogeneous block structure. To resolve this conflict, we choose the smallest block size that always contains free parameters as the block size, and put the other variables, which are sometimes fixed, or free, or joined, on the right hand side of the Hessian. This results in an arrow matrix with the extra free variables along the right and bottom sides, and the upper-left block consisting of fixed-sized blocks.

With the same motivation of keeping a homogeneous block structure, the gauge is left globally free. The gauge freedom of rotation, translation and scale is simply handled for each step by the dampener used in the optimization. A globally free gauge is known to improve convergence speed in general [22]. Other alternatives, such as fixing selected cameras and baselines were tried, with no discernible effect, and hence the floating gauge is preferred in order to preserve the homogeneous block structure.

The block structure allows for efficient memory handling, variable re-ordering, and customized sparse solving while maintaining the use of BLAS3, which is a library of matrix-matrix operations. The sparsity pattern is block-based, in that every block corresponds to a pair of cameras, and that block is either completely zero or completely filled in depending on whether the two cameras have a point track directly in common. The pattern persists across all the iterations of a bundle adjustment process. Before starting the iterations, the pattern is computed as an upper triangular bit-mask. Then, a sparse block matrix is prepared. This is accessed through a matrix of pointers, with valid pointers only to the non-zero blocks. This design is chosen for speed of access to the blocks during accumulation to the reduced camera matrix, and improves the speed on problems containing the tens of thousands of cameras. Each block is a square with sidelength b equal to the number of camera parameter ($b = 9$ for the uncalibrated camera model with two radial distortion parameters). Each whole block is stored in consecutive memory. The matrix is solved by a sparse block-based LDL factorization or conjugate gradient and back-substitution [9]. The solver spends essentially all of their time performing multiplication of the $b \times b$ blocks, a multiplication operation that can be completely unrolled and optimized. Our experiments indicate that even for dense systems where each pair of cameras share a track, the improved cache-locality and the BLAS3 nature of the block-solver produces a four times speed-up.

4. Solving Reduced Camera System

As mentioned in section 2.1, the reduced camera system (RCS) is obtained by a special Schur complement trick that makes the dimensionality compact. Moreover, several methods to efficiently solve the RCS, such as reorderings, sparse Cholesky factorization, and preconditioned conjugate gradient have been employed [22, 2]; an efficiently accumulating RCS has also been suggested [7]. However, several aspects exist that can still be improved further. In this paper, we explain how the proposed block structure and resultant block-based computation assist to improve these aspects. We also clarify CG and related topics, which are slightly ambiguous to use and have never been investigated by carrying out full performance analysis, to suggest the best method to construct the fastest bundler. Finally we introduce the block-based preconditioned CG.

4.1. Variable Ordering

The cameras are ordered to minimize the amount of blocks that are filled in during the LDL block factorization. While doing this completely optimally is NP-complete, several good and efficient approximate techniques exist. Approximate minimum degree (AMD) is a popular and powerful modern reordering technique [5, 6]. We actually perform exact minimum degree (MD) ordering. The reason why this is a better choice than AMD is that while the ordering takes place on the block level, the core factorization performs block operations that consist of b^3 scalar operations, where b is the size of one camera block. Hence, the time taken to find the ordering is swamped by the core factorization, and any improvement on the fill-in will repay itself b^3-fold. The situation for scalar sparse factorizations is different. In that case the exact MD ordering takes roughly the same amount as the subsequence core factorization, while the AMD can be found faster (say for example in 20% of the time) without sacrificing much quality on the ordering, resulting in efficiency improvements on the whole process of up to 40%. In this paper, we use reverse Cuthill-McKee ordering (REV) as an AMD in the experiments.

4.2. Preconditioned Conjugate Gradient

Conjugate gradient algorithm (CG) is the most widely used iterative method for solving large sparse linear systems and is the preferred technique for solving $H_{rc}x = -g$ when the system is large. Most of cameras distant to each other

observe different scenes, and this results in zero blocks of reduced camera matrix. Therefore, the larger image set normally causes the sparser system. In fact, CG reduces the time spent for solving the system and the size of required memory as well. However, one important thing that remains is that the convergence of CG is highly affected by the condition number of the system H.

In order to reduce the condition number, a preconditioning step is essential. If we fully (optimally) precondition the entire reduced system (which means that H^{-1} is used as the preconditioner P), CG will converge in a single iteration, but this is same as solving the full LDL. Consequently, we need to find a better trade-off between no preconditioning ($P = I$) and the full preconditioning ($P = H^{-1}$). The Jacobi preconditioner,

$$P = D^{-1}, \;\; D = diag\{H\}, \tag{1}$$

and the Symmetric Successive Over Relaxation (SSOR) preconditioner,

$$P = (D + L)^{-T} D (D + L)^{-1}, \;\; H = D + L + L^T, \tag{2}$$

have been conventionally used for the scalar-based CG. We propose preconditioning with limited bandwidth (truncated diagonals) as an alternative. To indicate the limited bandwidth, let H_n be the matrix containing 1 to n-th block diagonals and zeros for the rest block diagonals. When H is a $N \times N$ block matrix, H_0 and H_N equal to I and H. The band-limited block-based preconditioner is notated as,

$$P = H_n^{-1}, \quad 0 \leq n \leq N, \tag{3}$$

and is implemented with our customized block operations. Note that as with LDL factorization, the ordering of blocks makes a difference to the amount of (within-band) fill-in. Furthermore, the ordering also affects which out-of-band block are dropped and hence the efficacy of the preconditioner, which then affects the convergence rate of CG. Based on the performance comparison in Fig. 2, we decide which preconditioner is good. A detailed discussion is presented in the section 6.1.

How many iterations should be used inside each Levenberg-Marquardt step needs to be addressed. CG guarantees the convergence and exact solution as well as LDL after N iterations for an $N \times N$ matrix, but N CG iterations usually take as much time as the complete LDL. On the other hand, if we force CG to stop earlier, we may lose the accuracy of the solution for normal equation. We solve this problem by adopting a stopping criterion on relative decrease of squared residual. The criterion is,

$$\epsilon \geq \frac{{r^k}^T r^k}{{r^1}^T r^1}, \tag{4}$$

where r^k is the residual after the k-th iteration, where we set ϵ to 10^{-8}. This works reasonably well. In our experiments, the accuracy loss causes a negligible effect and does not degrade the convergence of any bundlers. As a result, the proposed block-based preconditioned CG achieves a remarkable improvement.

5. Embedded Point Iterations

After the camera update step is computed, it is standard to back-substitute for the point update. We have the option to take that point update step, but also to iterate on each of the points separately p times before the complete camera+point update step is considered complete and scored. Note that this is different from the vastly inferior procedure of alternation, in which points and cameras are moved independently. Instead, the camera update step is computed correctly based on allowing both cameras and points to move together, but the point updates given the correct camera moves use full optimization rather than just a first order prediction.

The intuition behind this is that for large, dense systems, EPIs are cheap compared to the full update step. Moreover, the camera steps are based on many point measurements and are therefore stable, while the point updates, which are based on as few as two observations, can be more erratic. Therefore, it is sometimes worth paying the small price of multiple point iterations in order to bring the points back to rest and to get the most out of each camera update step.

The EPIs are much more effective when a robustifier is applied to the cost. The main problem with least squares is its high sensitivity to outliers and this comes from the thin tails of Gaussian distribution [22]. To avoid this situation, we apply a robustifier to the squared errors in order to model heavier tails in the error distribution [3]. However, these robustifiers have flat (or near-flat) regions and points lying here move slowly during bundle iterations. Therefore, our EPIs are very helpful and usually save a few bundle iterations.

In practice, we apply point iterations at three different places and name them pre, core and post point iterations, respectively. These three point iterations commonly sync up the points to the current cameras. The pre-EPI is only performed before the first bundle iteration to make sure the given 3D points are optimal for the current cameras and this step is very useful for both of the robustified/non-robustified cost. The core-EPI reduces the cost further after the back-substitution and affects the decision on accepting the current updates. The post-EPI is applied after each complete bundle iteration and naturally replaces the pre-EPI of the next bundle iteration.

6. Experimental Results

We performed experiments using two kinds of datasets−synthetic and real. For the synthetic datasets, we randomly generate cameras and points around a sphere

set	cameras	points	projections	variables	error
9	111	47193	134142	142356	10.70
26	288	144656	468541	435984	18.76
29	383	283343	889911	852710	5.34
39	1083	411913	1225846	1243320	9.15
40	1195	247500	1035611	750865	39.37

Table 1. Detailed information about 5 out of our 40 datasets. All datasets are randomly perturbed. Listed sets are named as (**afternoon**, **sanmarco**, **annecy**, **cliffhouse**, **korpalace**) in order. The values in the error column are the RMS reprojection errors in pixel. The entire table is available in [12].

with a given radius. The values 1.0 and 0.5 are taken as the radius for the cameras and points, respectively. Basically, all the cameras are looking at the center of the sphere, and their extrinsic parameters are randomly perturbed. Points are distributed inside the small sphere. A hundred points are generated for a camera and shared with ten other cameras chosen randomly. To generate more realistic camera networks, five out of the ten cameras are selected from near neighbors and the other five from far ones. All synthetic datasets are processed on a 2.93GHz quad core PC without multi-threading.

The real datasets consist of mainly 35 synths[2] from the web [1] and five more datasets (Table 1), which are free to use, because the synths from the web cannot be published. All the cameras and the points of synths are recovered by [1]. To obtain a reliable experimental result, we select the synths whose number of cameras is widely distributed and is a maximum of 1200. The average number of projections per point ranges from 3 to 7. We ask the readers to refer to Table 1 in [12] for detailed information on all the real datasets. Figure 1 in [12] also shows three views and the filled-in hessian patterns of five free datasets. Each BA of real datasets is performed with a single 2.8 GHz core of a computer cluster.

We classify the tested settings of the bundler into three groups. The bundler that uses the proposed block-based solvers is marked as "B_" and the one that uses scalar-based solvers [7] is marked as "S_". While S_ is our own implementation, the conventional bundler marked as "L_" is a public and widely used implementation of [14]. "LDL" or "CG" is assigned in the second place, according to the linear solver. Consequently, six different bundlers are tested. In addition, we use "P" or "P*" for the cases in which the proposed EPIs are applied with or without the back-substitution, respectively. The above notation for describing our tested algorithms is used in Figures 3-6. In Figure 2, we use a different notation that describes the kind of preconditioning we tried, as described in the next paragraph.

[2]The term 'synth' is used for a set of cameras and points reconstructed from a set of unordered images by the SfM solution at [1]

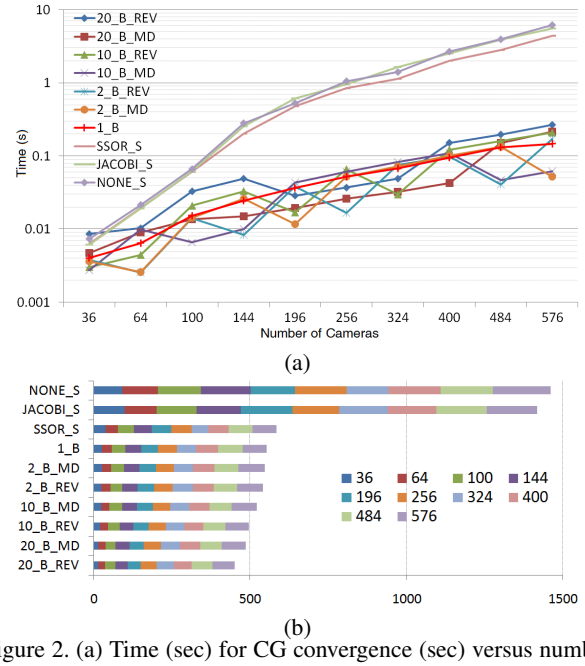

Figure 2. (a) Time (sec) for CG convergence (sec) versus number of cameras. For the '(A)_(B)_(C)' format of legend, (A) the number of the block bandwidth for block-based preconditioner or type of scalar preconditioner, (B) block or scalar type, and (C) variable orderings are coded. (b) The number of CG iteration to converge for various CG settings. SSOR and block-based preconditioners require about 3 times fewer iteration for convergence than NONE and JACOBI.

6.1. Experiments on Synthetic Datasets

Prior to performing experiments that test various settings, including a scalar or a block-based CG linear solver, it is necessary to fix an appropriate preconditioner for each CG solver. In order to find the best preconditioner, we investigate the total time (Fig. 2a) and the number of CG iterations (Fig. 2b) required to achieve convergence over all synthetic datasets. For the '(A)_(B)_(C)' format used in Fig. 2, (A) is the block bandwidth for block-based preconditioner or type of scalar-based preconditioner, (B) is the block or scalar type, and (C) is the variable re-ordering algorithm used. According to Fig. 2, "SSOR_S" (eqn. 2) is clearly better than "JACOBI_S" (eqn. 1) when the scalar CG is being used, whereas it is very difficult to decide which one is the fastest preconditioner among the tested block-based preconditioners. All block-based preconditioned CGs are around ten times faster than scalar CGs, and even the simplest one "1_B" ($P = H_1^{-1}$, i.e., block-based Jacobi preconditioning), which preconditions the main block diagonal only, saves more iterations than the scalar SSOR preconditioner. Although the number of CG iterations differs, all the tested block-based preconditioners show a similar overall performance. "1_B" is a good choice for experimental purposes, since it is independent of orderings and provides a predictable performance (shows a stably increasing plot in

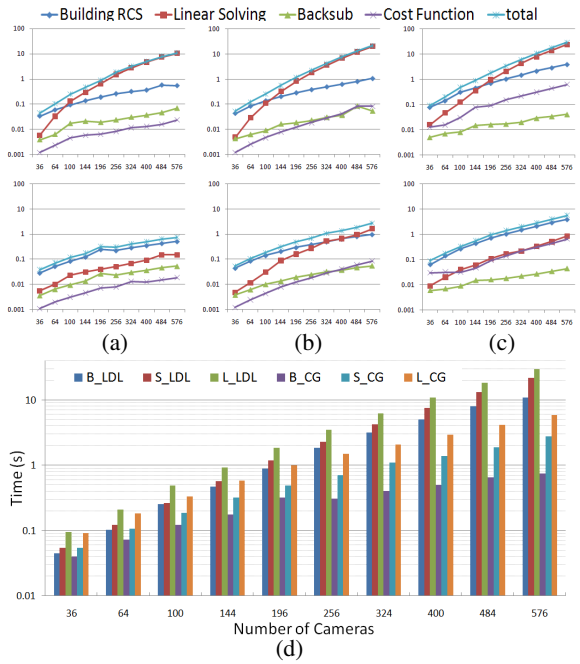

Figure 3. Time (sec) of each step in one iteration for the synthetic datasets. (a) Block-based(B), (b) Scalar(S), and (c) Conventional (L) bundlers with LDL(upper) and CG(lower) as linear solver. The minimum degree ordering is used for B_LDL and S_LDL.

Fig. 2). Therefore, for all the experiments that we carry out, "1_B" and "SSOR_S" are applied to every block-based and scalar CG, respectively.

In Fig. 3, it can be observed how the computation times increase for six different bundlers with an increasing number of cameras. The partial and total times for one iteration are measured and plotted in order to compare the distribution of time over four steps: building the RCS, linear solving, back-substitution, and computing costs. The top row of Fig. 3 shows that the computation time for LDL quickly dominates the total time as the number of cameras increases. The bottom row of Fig. 3 shows that CG takes considerably less time than LDL. Moreover, the block-based CG does not dominate the total time. Figure 3d shows the time for 'total' of Figures 3a-c, i.e., the top curve, grouped by the number of cameras, so that the relative performance of different algorithms can be more easily compared. It should be noted that the proposed "B_LDL" and "B_CG" are the fastest in each category and "B_CG" takes less than 1 second even with a highly occupied 576×576 block(6×6) matrix.

6.2. Experiments on Real Datasets

In this section, we show that the previously mentioned results are valid for real datasets and also demonstrate the effect of EPIs and its variant. In practice, waiting until the bundler converges completely is unacceptable and requires too much time for a massive experiment. We therefore stop the process if the relative cost improvement is lower than 0.01% for every bundler and EPIs. With this criterion, we compare the elapsed time to reach a specific level of error. One or two pixel reprojection error is considered a satisfactory level for 640×480 or 800×600 imagery in the structure from motion literature. Since our datasets are obtained at different image resolutions, we re-scale every image so that its longer side has an 800-pixel dimension and calculate the RMS reprojection error in pixel. A 1.5 pixel RMS reprojection error is selected for the comparison on Table 2.

We tested about forty different block-based and scalar bundler variants with various linear solvers, orderings, preconditionings, EPIs, and lambda control strategies. Table 2 shows a set of results for eight representative bundlers. For several datasets like **cliffhouse(39)** and **korpalace(40)**, scalar bundlers are not able to be applied because of the memory limit, as discussed in Fig. 1. The columns $\{1, 4, 7, 8\}$ should show an identical number of iteration at each row (the values in parentheses), if a scalar or block-based CG computes an ideal solution as LDL does, and so should the columns $\{2, 5\}$ and $\{3, 6\}$. Nevertheless, the columns with CG show differences for several datasets, because of the stopping criterion. Those differences, however, cause no crucial effect and still provide a faster convergence (Table 2 in [12]).

The EPIs, as expected, reduce the iterations very effectively. Allowing one more core-EPI iteration, instead of the back-substitution ("P*"), increases the computational speed. We do not explicitly investigate the reason for this increase, but it is probably due to the different linearizations where the point updates are computed. As we mentioned in Section 5, the cameras are better constrained, and by nature, they settle down quickly. The core-EPI uses the linearization computed with the updated cameras, whereas the back-substitution uses the linearization computed with the previous cameras. Therefore, we speculate that the proposed EPIs perform better because it makes the points follow the recently updated cameras, which are more reliable than the points and are placed better than the previous cameras. Considering the fact that the bundlers with EPIs rarely fail to reach 1.5 pixel error, EPIs are helpful not only for saving iterations, but also for achieving better convergence.

Figure 4 visualizes the entire version of Table 2. The lower is the better, and lines of "B_CG_P*" and "B_LDL_P*" are obviously two lowest ones. The actual gaps between the plotted lines are substantial (the 'z' axis for time is in a logarithmic scale).

The overall performance differs significantly between the scalar and the proposed approaches, and "B_CG_P*" finally shows the best result over all datasets in terms of time, iteration, and convergence.

Times for one iteration of the five publishable datasets are also compared in Figure 5a, as in Figure 3d on the

set	Block-based LDL (B_LDL)			Block-based CG (B_CG)			S_LDL	S_CG
	BS only	_P	_P*	BS only	_P	_P*	BS	BS
9	6.64 (22.9)	0.91 (2.0)	**0.44 (1.0)**	5.92 (22.9)	1.64 (4.3)	0.45 (1.0)	8.75 (22.9)	7.35 (22.9)
26	N/A	17.61 (10.9)	6.58 (5.0)	N/A	14.04 (10.9)	**5.07 (5.0)**	N/A	N/A
29	N/A	4.81 (2.0)	**2.53 (0.9)**	N/A	10.54 (4.7)	**2.53 (0.9)**	N/A	N/A
39	364.46 (70.3)	85.05 (13.6)	6.27 (1.0)	186.1 (70.3)	49.39 (13.6)	**4.53 (1.0)**	N/T	N/T
40	N/A	N/A	196.0 (2.9)	N/A	N/A	**26.9 (5.2)**	N/T	N/T

Table 2. Elapsed time and number of iteration (in parentheses) for reducing RMS reprojection error to 1.5 pixel. Bold font indicates the minimum time for each row, and N/A stands for the case that 1.5 pixel error is not achieved under the stopping criteria. N/T means 'not tested'. The entire table is available in [12].

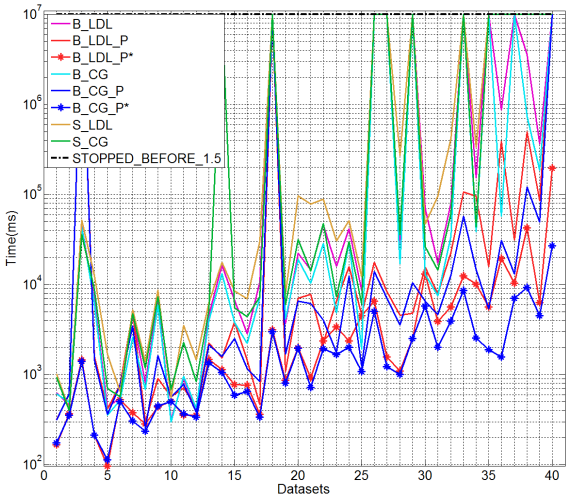

Figure 4. Time to reach 1.5 pixel RMS reprojection error. The plotted lines meet "STOPPED_BEFORE_1.5" when the corresponding setting of bundler stopped before reaching 1.5 pixel error for the datasets or was not tested.

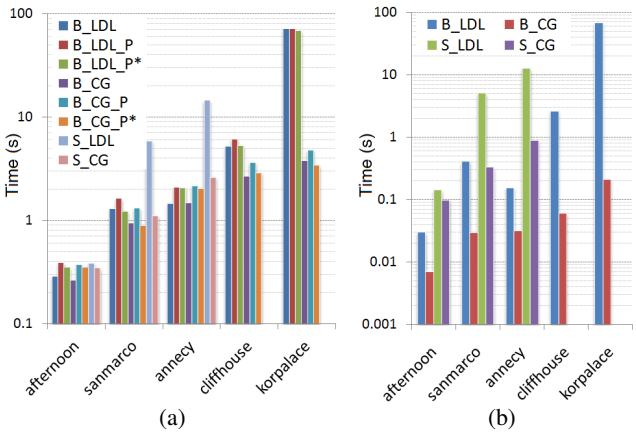

Figure 5. Time (sec) for (a) one bundle iteration and (b) one linear solving on the publishable datasets. As explained, the scalar solver is not able to be performed on two datasets on the right.

synthetic datasets. Besides the reduced number of iterations, EPIs show cheap cost and our block-based bundlers (both direct LDL and iterative CG) are faster than the scalar ones, even with EPIs. Two main factors, the number of cameras and the sparsity of the Hessian of RCS, are relevant. While the former affects the complexity of the entire bundle process, the latter mainly affects the linear solving. Therefore, the gap between the scalar bundlers and the block-based bundlers becomes larger as the number of cameras increases, and the dense Hessian widens the gap between LDL and CG. A direct comparison on the time for linear solving in Fig. 5b may help the reader to understand the effect. The **korpalace** and **cliffhouse** have similar complexity, but the **korpalace** has the denser Hessian than the **cliffhouse** (refer to Table 1 and Fig. 1 in [12]). The **korpalace** causes much larger gap between "B_LDL" and "B_CG" than the **cliffhouse** in Fig. 5. This means that the proposed block-based preconditioned CG gets more crucial when problems get bigger and denser.

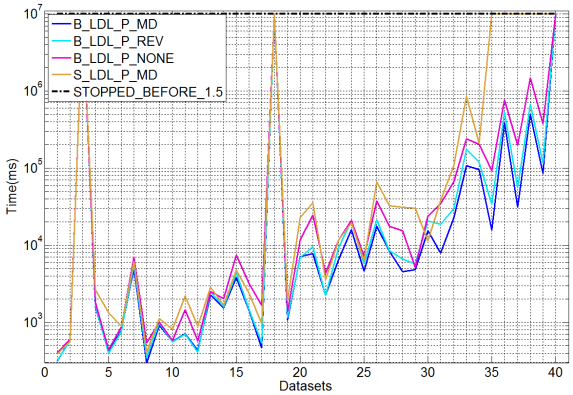

Figure 6. Time to reach 1.5 pixel RMS reprojection error for different ordering methods. The plotted lines meet 'STOPPED_BEFORE_1.5' when the corresponding setting of bundler stopped before reaching 1.5 pixel error for the datasets or was not tested.

Finally, we demonstrate that the exact minimum degree ordering becomes faster than the approximate one (REV) as BLAS3 becomes available. The comparison is depicted in Fig. 6 and is focused on variable ordering methods. It should be noted that the four bundlers that are compared follow one common convergence path and only differ in time because the ordering does not affect the result of LDL solving. Therefore, their plotted lines move together. Because of the increasing complexity, the absolute gap between lines widens for the latter datasets. The block-based LDL with the exact minimum degree ordering (MD) is clearly better

than the one with reverse Cuthill-McKee ordering (REV) and is the fastest one.

7. Conclusion and Future Work

This paper provides three main contributions−adapting the block structure to deal with fixed and tied variables, the resulting block-based linear solvers, and the novel EPIs. A carefully-managed block structure can maintain the desired homogeneity and allow the use of BLAS3 and efficient memory handling. It also supports fast reordering and customized sparse linear solving such as the block-based preconditioned CG, the effectiveness and the stability of which are proven by our experiments. At the same time, the EPIs successfully sync the points to the camera updates so that the entire bundle iterations are not wasted. Finally, the bundlers that simultaneously utilize all the proposed contributions outperform the previous bundlers tested in the experiments. The block-based preconditioned CG with EPIs, which is the best bundler, achieves a substantial improvement. Moreover the proposed bundler can be used in concert with other approaches that modify how and when the bundler is invoked.

There are several issues that need to be addressed in a future study. The effect of the λ control strategy has not been fully investigated. A comparison between the suggestion given in [15] and LMs with various strategies is also interesting. Another task is parallelizing the proposed method. All our sub-steps except for linear solving can be computed in parallel in a straightforward manner, with exception of linear solving. However, it is easy to parallelize CG and the block Jacobi preconditioner. Once parallelization has been implemented, the computational speed should further be improved.

References

[1] Photosynth. http://photosynth.net. 3, 5

[2] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building rome in a day. In *IEEE International Conference on Computer Vision (ICCV)*, 2009. 1, 2, 3

[3] M. J. Black and A. Rangarajan. The outlier process: Unifying line processes and robust statistics. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1994. 4

[4] M. Byrod and K. Astrom. Bundle adjustment using conjugate gradients with multiscale preconditiong. In *British Machine Vision Conference*, 2009. 1, 2

[5] T. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006. 3

[6] F. Dellaert and M. Kaess. Square root sam: Simultaneous location and mapping via square root information smoothing. *International Journal of Robotics Research*, 2006. 2, 3

[7] C. Engels, H. Stewénius, and D. Nistér. Bundle adjustment rules. In *Photogrammetric Computer Vision (PCV)*. ISPRS, Sept. 2006. 2, 3, 5

[8] A. Eudes and M. Lhuillier. Error propagations for local bundle adjustment. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2009. 2

[9] G. Golub and C. V. Loan. *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences, 3rd edition edition, 1996. 3

[10] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 2

[11] S. Holmes, G. Sibely, G. Klein, and D. W. Murray. A relative frame representation for fixed-time bundle adjustment in sfm. In *IEEE International Conference on Robotics and Automation*, 2009. 2

[12] Y. Jeong, D. Nister, D. Steedly, R. Szeliski, and I.-S. Kweon. Supplementary material. `0486_supp.pdf`. 5, 6, 7

[13] K. Levenberg. A method for the solution of certain nonlinear problems in least squares. *The Quarterly of Applied Mathematics*, 2:164–168, 1944. 1, 2

[14] M. Lourakis and A. Argyros. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm. Technical Report 340, Institute of Computer Science - FORTH, Heraklion, Crete, Greece, Aug. 2004. 1, 2, 5

[15] M. Lourakis and A. Argyros. Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment? In *IEEE International Conference on Computer Vision (ICCV)*, pages 1526–1531, 2005. 1, 8

[16] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Real time localization and 3d reconstruction. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, pages 363–370, 2006. 2

[17] K. Ni, D. Steedly, and F. Dellaert. Out-of-core bundle adjustment for large-scale 3d reconstruction. In *IEEE International Conference on Computer Vision (ICCV)*, 2007. 1, 2

[18] H.-Y. Shum, Z. Zhang, and Q. Ke. Efficient bundle adjustment with virtual key frames: A hierarchical approach to multi-frame structure from motion. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, pages 2538–2543, 1999. 2

[19] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph.*, 25(3):835–846, 2006. 3

[20] D. Steedly and I. A. Essa. Propagation of innovative information in non-linear least-squares structure from moion. In *IEEE International Conference on Computer Vision (ICCV)*, pages 223–229, 2001. 2

[21] D. Steedly, I. A. Essa, and F. Dellaert. Spectral partitioning for structure from motion. In *IEEE International Conference on Computer Vision (ICCV)*, pages 996–1003, 2003. 2

[22] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – a modern synthesis. In B. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer-Verlag, 2000. 1, 2, 3, 4