Pushing the Envelope of Modern Methods for Bundle Adjustment

Yekeun Jeong, *Student Member*, *IEEE*, David Nistér, *Member*, *IEEE*, Drew Steedly, *Member*, *IEEE*, Richard Szeliski, *Fellow*, *IEEE*, and In-So Kweon, *Member*, *IEEE*

Abstract—In this paper, we present results and experiments with several methods for bundle adjustment, producing the fastest bundle adjuster ever published in terms of computation and convergence. From a computational perspective, the fastest methods naturally handle the block-sparse pattern that arises in a reduced camera system. Adapting to the naturally arising block-sparsity allows the use of BLAS3, efficient memory handling, fast variable ordering, and customized sparse solving, all simultaneously. We present two methods; one uses exact minimum degree ordering and block-based LDL solving and the other uses block-based preconditioned conjugate gradients. Both methods are performed on the reduced camera system. We show experimentally that the adaptation to the natural block sparsity allows both of these methods to perform better than previous methods. Further improvements in convergence speed are achieved by the novel use of embedded point iterations. Embedded point iterations take place inside each camera update step, yielding a greater cost decrease from each camera update step and, consequently, a lower minimum. This is especially true for points projecting far out on the flatter region of the robustifier. Intensive analyses from various angles demonstrate the improved performance of the presented bundler.

Index Terms—Computer vision, bundle adjustment, structure from motion, block-based, sparse linear solving, point iterations.

1 Introduction

Bundle adjustment (BA) has become an essential part of structure from motion (SfM) and 3D reconstruction has attracted increased interest from the computer vision community despite its extremely complex nature. Although many reports have been presented on the subject, bundle adjustment remains the primary bottleneck in relevant applications and is problematic in large-scale reconstructions.

To resolve these problems, we present several methods that dramatically improve the performance of the bundle adjustment (i.e., the bundler). We exploit the block-sparsity pattern that arises in a reduced camera system (RCS) and enhance the computational speed of the bundler with BLAS3[1] operations, efficient memory handling, and fast block-based linear solving. Furthermore, novel embedded point iterations (EPIs) substantially improve the convergence speed by yielding a high cost decrease from each camera update step. The experimental analyses covering various bundlers and data sets comprise another important contribution of this paper. The experimental results show the improved performance of the proposed bundler and provide useful and detailed comparisons among various choices when compositing a bundler. Our approach does not require any special assumption or hardware and is identical to 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 of the other previous approaches demonstrates the generality and the applicability of our proposed method.

1. BLAS3 is a library of matrix-matrix operations.

- *Y. Jeong, D. Nistér, D. Steedly, and R. Szeliski are with Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399. E-mail: {yejeong, dnister, steedly, szeliski}@microsoft.com.*
- *I.-S. Kweon is with the RCV Lab, Department of Electrical Engineering, KAIST, 291 Daehak-ro, Yuseong-gu, Daejeon 305-701, Korea. E-mail: iskweon@ee.kaist.ac.kr.*

Manuscript received 2 Mar. 2011; revised 9 Sept. 2011; accepted 22 Nov. 2011; published online 19 Dec. 2011.
Recommended for acceptance by Y. Sato.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-2011-03-0138.
Digital Object Identifier no. 10.1109/TPAMI.2011.256.

1.1 Related Works

The previous approaches can be divided into two groups. The first focuses on making the bundle adjustment algorithm as efficient as possible, and the second focuses on reducing the size or frequency of the invocation of individual bundle adjustments.

Examples of the first group include [1], [2], [3], [4], [5], [6], [7], and [8]. Triggs et al. [1] and Lourakis and Argyros [2] explained bundle adjustment and how to implement it. Triggs et al. discussed more theoretical issues, including gauge freedom, inner constraints, and the reliability of parameter estimates [1], while Lourakis and Argyros offered detailed explanations on the standardized bundle adjustment procedures [2]. The Levenberg-Marquardt algorithm (LM) [9] has been the most popular choice for bundle adjustment. However, the authors of [3] questioned this choice and showed that the dog leg algorithm (DL), which is designed to explicitly use a concept of trust region, is a better alternative to LM, which reflects the fitness of the approximated linear model by checking if the cost decreases. In [4], an out-of-core bundle adjustment was proposed which follows a divide-and-conquer approach. In that aspect only, that work could be classified into the other group; however, it did make an additional contribution. By

caching submap linearizations for the full separator system, the authors were able to reconstruct a large-scale system if given a good graph cut and initialization.

Within the last few years, there have been several attempts to solve the linear system in the bundle adjustment more efficiently. Preconditioned conjugate gradients (PCG) replacing the Cholesky factorization that is used in the LM algorithm are the key to these attempts. Byröd and Åström utilized the structural layout of variables for the better preconditioning of conjugate gradients (CG) in bundle problems such that the CG steps affected an explicit change in the parameters more directly [5]. Agarwal et al. suggested the adaptive use of a sparse direct method for Cholesky factorization and a block diagonal PCG [6].

Recently, more developed ways to apply the conjugate gradient to bundle adjustment with less memory requirements have been presented [7], [8]. In [8], the authors suggested applying conjugate gradients for least squares (CGLS) to bundle adjustment with a block QR preconditioning. The CGLS requires the Jacobian matrix only and does not build the Hessian matrix. In this case, a reduced camera system is not used, but it is still advantageous when the memory requirement becomes the main concern. Agarwal et al. proposed using PCG with a generalized symmetric successive over-relaxation (SSOR) preconditioner for the Hessian matrix to implicitly use PCG on an RCS without any explicit construction of the RCS [7]. These approaches can be useful when the given problem has as many as 10,000 cameras.

In the second category, [10], [11], [12], [13], [14], [15], [16], and [17] proposed various approaches to apply bundle adjustment. The authors of [10] recovered a 3D structure from a long sequence by performing bundle adjustment hierarchically from segment-wise to global and also suggested an efficient approach that reduces the number of frames in the global system by introducing virtual key frames. In [11], the authors reduced the redundancies of the brute force bundling by checking which variables were required to be optimized after every new frame. The authors of [12] proposed a spectral partitioning approach, which divided a large-scale bundle problem into smaller subproblems and preserved the low error modes of the original system.

The authors of [13], [14], [15], [16], and [17] paid particular attention to how to efficiently apply bundle adjustment to incremental or real-time SfM systems. Mouragnon et al. [13] and Engels et al. [14] investigated the proper application of local bundle adjustment, which only considers cameras and points within a certain time range. Mouragnon et al. [13] suggested applying local bundle adjustment after each new keyframe was found, whereas Engels et al. [14] suggested applying it after every frame was added. Klein and Murray [15] presented a real-time augmented reality system using two independent threads for tracking and mapping. The main task of the mapping thread is to optimize keyframes and points locally with high priority and globally with low priority. This strategy is effective when the camera does not explore new scenes continuously and the expected number of total keyframes is fairly low. Eudes and Lhuillier [17] proposed a method that included uncertainty propagation and the maximum likelihood estimation of the local bundle adjustment given a particular noise model. In the case of [16], a relative frame representation was introduced instead of representing cameras and points in common global coordinates. Those authors used the relative motions between cameras for efficient loop closing in the incremental SfM.

1.2 Overview

According to our survey of the literature, block structure has been understood equivalently as the sparse structure of SfM problems only and has been used mainly to form the RCS efficiently. The use of CG, which is the key component to reducing the cubic complexity of a bundler, has not been investigated by any full performance analysis. No effort has been made to understand how cameras and points move inside bundle iterations, which may provide an important clue to accelerating the convergence. Addressing these issues are the key contributions of this paper.

In Section 2, we briefly explain the standard bundler and several different implementations of the proposed bundler. We discuss how to fully utilize the block structure by employing the BLAS3 (Section 3) and how to efficiently solve the reduced camera system using block-based reorderings and preconditioned conjugate gradients (Section 4) in detail, which contributes to improving the computational efficiency of the proposed method. The novel EPIs that were inspired by understanding the general movements of cameras and points during bundle iterations are introduced in Section 5. They mainly contribute to the faster convergence of the bundler. We experimentally demonstrate that our proposed methods perform substantially faster than previous methods in Section 6. We present our conclusions in Section 7.

The performance of the proposed method is summarily compared with that of the conventional bundler [2] in Fig. 1. Twenty bundle iterations are performed for both methods, and the proposed method achieves a lower (RMS) reprojection error with a shorter processing time than the conventional one. This demonstrates that the proposed method is a more efficient way to perform the procedures of the bundler as well as a faster way to converge. A preliminary version of this paper appeared in [19].

2 Bundle Adjustment

Bundle adjustment is the problem of refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameter [1] by minimizing the robustified squared sum of the reprojection errors. At the outermost layer, BA is performed using the Levenberg-Marquardt (LM) algorithm [9], which is a damped Newton method. The LM algorithm assumes the cost function to be locally quadratic and dampens the Hessian matrix by controlling a dampener λ when the function is not fitted well.

In general, the computed Jacobian J is a $2N_P \times (kM + 3N)$ matrix, and the resultant Hessian $H = J^T J$ is a $(kM + 3N) \times (kM + 3N)$ matrix where M, N, N_P, k are the numbers of cameras, points, projections, and parameters for one camera, respectively. Solving a $(kM + 3N) \times (kM + 3N)$-sized linear system simply is infeasible when the problem includes too many cameras and points. However, M is

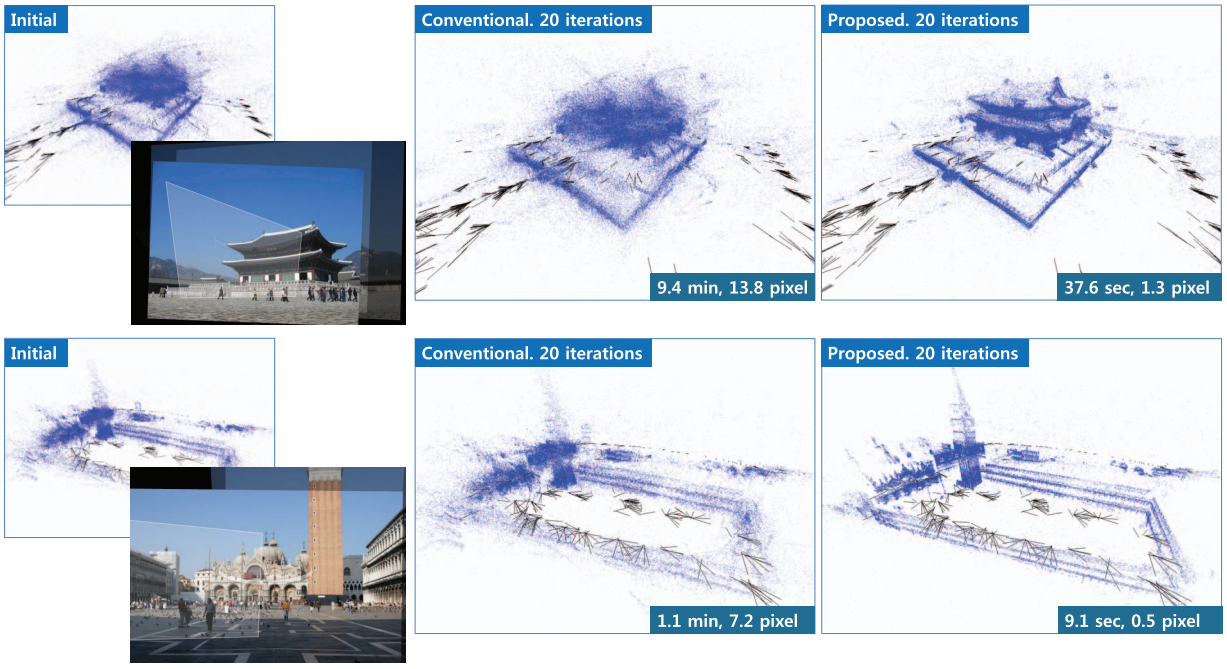


Fig. 1. A brief comparison between the conventional bundler [2] and the proposed bundler with preconditioned conjugate gradient as a linear solver. Two real data sets, **korpalace** (top) and **sanmarco** (bottom), were used. For each row, "initially perturbed cameras and points with a snapshot captured on the web [18]," "an adjusted result by the conventional method," and "another result by the proposed method" are shown from left to right. For each bundler, 20 iterations were completed. The timings (in min. or sec.) and the final RMS reprojection errors (in pixels) are shown with the adjusted reconstructions. The conventional bundler for **korpalace** was especially compiled on an x64 platform because of its large memory requirement.

considerably smaller than N in most cases, and the Schur complement can be used to reduce the large system to a smaller $kM \times kM$-sized linear system H_{rc}, which is the so-called reduced camera system [1], [20]. If N is much smaller than M, computing the reduced structure system (RSS) is more efficient. For more details on bundle adjustment, refer to [1], [2], [14].

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} depends on the number of cameras, the bundle adjustment does not have to suffer due to the large number of points. Note that if this Schur complement is computed explicitly and the sparse structure of the Hessian matrix is not considered, many advantages may be lost. The importance of RCS creation and its sparse structure were already studied by Brown several decades ago [21]. Constructing the RCS implicitly allows for faster speed and less memory usage [14]. Methods for managing the structure of RCS and solving RCS are explained in Sections 3 and 4.

2.2 Implementation Details

Aside from the key contributions that we will explain in the rest of this paper, several implementation details are worthy of mention. Our implementation is similar in spirit to [14], including computing the outer products, keeping the sine values of rotation, and augmenting the scaled diagonal of the Hessian matrix. By computing the outer products, i.e., accumulating the point tracks directly into the RCS, we do not form any intermediate matrices such as U, W, and V in [2] to explicitly construct the RCS.

The Rodriguez representation is a well-known parameterization of rotation, and the sine values of rotation have similar partial derivatives to those of the Rodriguez representation. By permutating the axes and inverting the signs of the derivatives of the sine values, the derivatives of the Rodriguez representation can be obtained. Therefore, we can speculate that the performances of the two parameterizations are equivalent. We conducted an experiment on this assertion and found that the two parameterizations provided nearly equivalent decreases until convergence and were better than the Euler angles. Moreover, keeping the sine values allows us to compute an exact rotation matrix more efficiently, even for noninfinitesimal changes.

Although the LM algorithm simply rescales the dampener according to the decrease/increase of the cost, the fitness of the approximation can be carefully measured by comparing the expected decrease and the actual decrease, which is also known as the trust region control. An example where the trust region method was applied to BA is given in [3]. We employ the concept of trust region control by carefully checking the discrepancy between the predicted and actual costs. We decrease the dampener when the cost reduction is above a certain fraction (currently set to 70 percent of the cost reduction predicted by the approximated model).

To further reduce computational load, we use an LM variant that differs in how the dampener affects the Hessian matrix. Normally, the diagonal augmentation of the LM is given by

$$H_{aug} = H + \lambda I, \tag{1}$$

where H and H_{aug} are the Hessian and the augmented Hessian matrices, respectively. However, we augment the

Hessian in a different way that adds the diagonal of the Hessian matrix multiplied by the dampener:

$$H_{aug} = H + \lambda \, diag\{H\}. \quad (2)$$

This augmentation helps the bundler to avoid repeatedly solving the linear system for newly increased dampeners when the computed step fails to reduce the cost [14].

3 Block Structure

The fact that a block-sparse pattern arises naturally in a reduced camera system is very well known [1], [2], [4], [14], [22]. A fair amount of research has already been performed on the topic of reordering techniques for cameras and has been designed to reduce fill-in during a direct solution. However, linearly solving the Hessian matrix still accounts for most of the complexity of the bundler [14]. 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.

The block structure allows efficient memory handling, variable reordering, 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 a block is either completely empty 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 the bundle adjustment process.

Before starting the iterations, the pattern is computed as an upper triangular bit-mask in our implementation. Then, a sparse block matrix is prepared. This is accessed through a matrix of pointers with valid pointers only on the nonzero blocks. This design is chosen for its speed of access to the blocks during accumulation to the reduced camera matrix, and it improves the speed of solving problems containing the tens of thousands of cameras. When a common camera model is employed, each block is a square with a sidelength b that is equal to the number of the camera parameter ($b = 9$ for an 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 preconditioned conjugate gradients and back-substitution [23]. The solver essentially spends all of its time multiplying the $b \times b$ blocks, which is an operation that can be completely unrolled and optimized.

However, practical applications of the SfM should deal with various unknown cameras at the same time, such as in [6], [18], [24]. A mixed set of cameras containing partially known, fully known, and unknown intrinsics is an important case. The fixed parameters need to be omitted, and the parameters shared by several cameras need to be joined. This also breaks the homogeneous block structure. For example, the inhomogeneous sparse structure of the RCS shown in Fig. 2a may arise. In principle, the variable block structure can be allowed. However, it is easier if we treat all of the camera blocks identically because this allows a very simple and efficient block solver to be applied to the homogeneous block structure of the RCS.

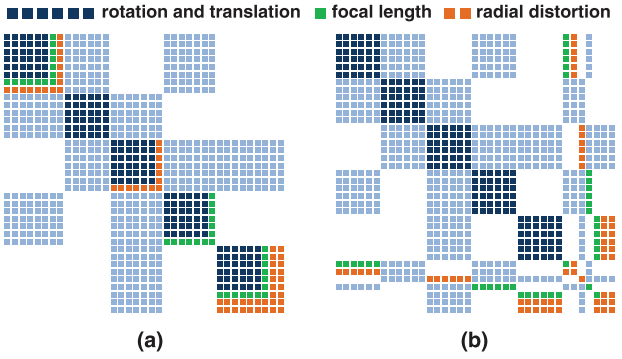


Fig. 2. An example of keeping the homogeneous block structure in the RCS when an SfM problem deals with various unknown cameras at the same time. (a) The original and (b) the reordered sparse patterns of the RCS are compared.

To maintain the homogeneous block structure, we choose the smallest block size that always contains free parameters such as the block size, and place the other variables, which are sometimes fixed, free, or joined, on the right-hand side of the Hessian matrix. 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 as shown in Fig. 2b. To utilize the simplified block structure, we also use hand-coded BLAS3 routines for small-sized matrix operations.

With the same motivation of maintaining 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 that is used in the optimization. A globally free gauge is known to generally improve convergence speed [1]. Other alternatives, such as fixing selected cameras and baselines, have been attempted with no discernible effect; therefore, the floating gauge is preferred to preserve the homogeneous block structure. However, this floating gauge could cause rank deficiency, which is critical in the Cholesky factorization when the dampener approaches zero. We handle this rank deficiency in the factorization by checking whether zero is reached in the diagonal and skipping zero divisions.

Our experiments indicate that even for dense systems where each pair of cameras shares a track, the improved cache-locality and the BLAS3 nature of the block-solver produce a four times speed up.

4 Solving The Reduced Camera System

As mentioned in Section 2.1, the reduced camera system is obtained using the Schur complement trick that compacts the dimensionality. Several methods to efficiently solve the RCS, such as reorderings, sparse Cholesky factorization, and preconditioned conjugate gradients, have been employed in previous works [1], [6]. An efficient accumulation of an RCS has also been suggested [14]. In this section, we explain how the proposed block structure and resultant block-based computation assist in improving the performance of the bundler. We also clarify the application of conjugate gradients and other related topics, which are ambiguous and have not been investigated sufficiently, to suggest the best method for constructing the fastest

bundler. Finally, we introduce the block-based preconditioned conjugate gradients.

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[2]) is a popular and powerful modern reordering technique [22], [25].

Here, we use exact minimum degree (MD) ordering. This is a better choice than AMD because although 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. Therefore, the time taken to find the ordering is swamped by the core factorization, and any improvement on the fill-in repays itself b^3-fold. The situation for scalar sparse factorizations is different. In that case, the exact MD ordering takes roughly the same amount of time as the subsequence core factorization, whereas the AMD can be faster (for example, in 20 percent of the time) without sacrificing much quality in the ordering, which results in efficiency improvements of the entire process of up to 40 percent.

4.2 Preconditioned Conjugate Gradients

The conjugate gradients algorithm is the most widely used iterative method for solving large sparse linear systems in the form of $Ax = b$. Most of the cameras that are distant from each other observe different scenes, which results in zero blocks of the Hessian matrix H and the reduced camera system H_{rc}. A larger image set normally results in a sparser system.

The Hessian matrix H has been a typical choice for "A" in BA. However, the direct application of CG to H is generally known to be inefficient because of its slow convergence [1] even though recent research has revealed its potential applicability to large SfM problems consisting of tens of thousands of cameras [7], [8]. We apply CG to H_{rc} to replace the role of the Cholesky factorization, which grows along with the cube of the number of cameras. The implicit method proposed in [7] also applies CG to H_{rc} without explicitly constructing H_{rc}. This method is algebraically equivalent to ours, but the implicit method has a stability issue that requires further investigation.

The use of CG could reduce the time needed to solve sparse systems and the size of the required memory as well. However, one important issue that remains is the condition number of the system A, which highly affects the convergence speed of CG. To reduce the condition number, a preconditioning step is essential. This can be done by applying preconditioned conjugate gradients in which H^{-1} is used. Another alternative is split preconditioner conjugate gradient, in which a preconditioner is split into $H^{-1} = L^{-T}L^{-1}$ form before preconditioning [26]. In the rest of this section, we use H for the augmented RCS for simplicity instead of using H_{rc} or A.

If we fully (optimally) precondition the entire system (which means that H^{-1} is used as the preconditioner P), CG will converge in a single iteration, but this is the same as solving the full LDL. Consequently, we must find a better tradeoff between no preconditioning ($P = I$) and full preconditioning ($P = H^{-1}$). The Jacobi preconditioner

$$P = D^{-1}, \quad D = diag\{H\}, \tag{3}$$

and the SSOR preconditioner

$$P = \left(\frac{D}{\omega} + L\right)^{-T} \frac{\omega}{2-\omega} D \left(\frac{D}{\omega} + L\right)^{-1}, \tag{4}$$

where $H = D + L + L^T$ and $0 \leq \omega < 2$, have conventionally been used for the scalar-based CG [26].

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 nth block diagonals and zeros for the rest of the block diagonals. When H is an $N_H \times N_H$ block matrix, H_0 and H_{N_H} are equal to I and H. The band-limited block-based preconditioner is notated as

$$P = H_n^{-1}, \quad 0 \leq n \leq N_H, \tag{5}$$

and is implemented using our customized block operations. Note that as with LDL factorization, the ordering of blocks makes a difference in the amount of (within-band) fill-in. Furthermore, the ordering also affects which out-of-band blocks drop out and hence the efficacy of the preconditioner, which then affects the convergence rate of CG. We decide which preconditioner is suitable based on the performance comparison shown in Fig. 5. A detailed discussion of this is presented in Section 6.1.

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

$$\epsilon \geq \frac{r^{k^T} r^k}{r^{1^T} r^1}, \tag{6}$$

where r^k is the residual after the kth iteration. For the stopping criterion, some loose thresholds may cause no problems. However, we set ϵ to 10^{-8}, which is quite tight, to guarantee that CG obtains similar steps than those of the Cholesky factorization. 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 practice to back-substitute for the point update. We have the option to make that point update step, but we can also iterate on each of the points separately p times before the entire camera+point update step is completed and scored. In other

2. In this paper, we use reverse Cuthill-McKee ordering as an AMD in the experiments.

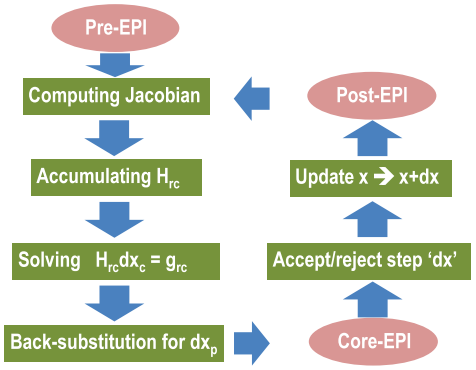


Fig. 3. A summarized flow of the proposed bundler. Ellipses indicate the EPIs that are optional, whereas boxes correspond to the standard procedures of bundle adjustment.

words, we optionally optimize each point with newly updated cameras using a tiny LM optimization. This is the core Embedded Point iteration, which is one of three EPIs that are introduced in this section. 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 correctly computed based on allowing both cameras and points to move together; however the point updates, given the correct camera movements, use full optimization rather than just a first-order prediction.

The idea behind this is that, for large, dense systems, EPIs whose complexity is linear in the number of points are much cheaper than the full update step. Moreover, the camera steps are based on many point measurements and are therefore stable, whereas 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 performing multiple point iterations to bring the points back to rest and to get the most out of each camera update step.

Occasionally, the points become more stable than the cameras in long image sequences or images that are captured by distant cameras with a narrow field of view. In this case, the situation can be reversed. Although we have not examined these cases, embedded camera iterations (ECIs) could be more efficient than EPIs, as seen in the reduction of the Hessian matrix to an RCS or an RSS.

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; this is due to the thin tails of the Gaussian distribution [1]. To avoid this situation, we apply a robustifier to the squared errors to model the heavier tails in the error distribution [27]. However, these robustifiers have flat (or near-flat) regions and points in this area 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 call them pre, core, and postpoint iterations (Fig. 3). 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 ensure that the given 3D points are optimal for the current cameras. This step is very useful for both the robustified/nonrobustified cost functions. The core-EPI reduces the cost further after the back-substitution and affects the acceptance of the current updates. The post-EPI is applied after each complete bundle iteration and naturally replaces the pre-EPI of the next bundle iteration.

Throughout the experiment, we terminate EPIs by limiting the maximum number of iterations as well as the minimum relative cost decrease. These stopping criteria are chosen based on an experiment in which the convergence curves of bundlers with various combinations of three maximum numbers of iterations and the minimum relative decrease are compared. We allow five, two, and 10 iterations for the pre, core, and post-EPIs as the maximum. We terminate the iterations if the relative cost decrease becomes less than 1 percent.

6 Experimental Results

We performed experiments using two types of data sets, synthetic and real. For the synthetic data sets, we randomly generated cameras and points around a sphere with a given radius. Values of 1.0 and 0.5 were taken as the radius for the cameras and points, respectively. Essentially, all of the cameras were looking at the center of the sphere, and their extrinsic parameters were randomly perturbed. Points were distributed inside the small sphere. One hundred points were generated for each camera and shared with 10 other cameras chosen randomly. To generate more realistic camera networks, five out of the 10 cameras were selected from nearby neighbors, and the other five were selected from far away ones. All of the synthetic data sets were processed on a 2.93 GHz quad core PC without multithreading.

The real data sets consisted of mainly 35 synths[3] from the web [18] and five more data sets (Table 1), which are free to use, because the synths from the web cannot be published. All of the cameras and the points of the synths were recovered in [18]. To generate randomly perturbed data sets, random noises were added to the recovered cameras, and all of the points were retriangulated. The image feature tracks were used without any modification. Detailed information, i.e., the original and noisy reconstructions and filled-in patterns of the reordered RCS, from the five data sets is shown in Fig. 4. To avoid biased experimental results, we selected the synths with a varying number of cameras. Detailed information on the real data sets is listed in Table 1. All of the data sets were divided into six groups according to their number of cameras for statistical analysis. The average number of projections per point ranged from three to seven. Each BA of the real data sets was performed with a Xeon 2.8 GHz quad core PC without multithreading.

We classified the tested settings of the bundler into three groups. Bundlers using the proposed block-based solvers were marked as "B_" and the others using scalar-based solvers [14] were marked as "S_". Although the bundlers marked as S_ are our own implementations, the conventional bundler marked as "L_" is a public and widely used implementation from [2]. "LDL" or "CG" were assigned next according to the linear solver. Six different bundlers were tested. In addition, we used "P" or "P*" for the cases in which the proposed EPIs were applied with or without

3. The term "synth" was used for a set of cameras and points that were reconstructed from a set of unordered images by the SfM solution in [18].

TABLE 1
Detailed Information of Our 40 Data Sets

set	cameras	points	projections	variables	error
1	40	17183	51627	51829	9.81
2	41	19346	126731	58325	8.04
3	89	56413	206026	169862	17.50
4	93	17480	51604	53091	14.12
5	94	7015	23999	21703	12.52
6	100	46241	191326	139423	15.98
7	105	34829	92991	105222	15.32
8	108	26519	85921	80313	10.62
9	**111**	**47193**	**134142**	**142356**	10.70
10	133	68114	176056	205273	5.92
11	134	15836	97446	48446	17.05
12	134	44977	117103	135869	6.01
13	165	85439	443947	257472	12.42
14	170	67548	360519	203834	19.14
15	179	49976	155499	151181	7.54
16	184	85533	243763	257887	9.61
17	190	33660	96863	102310	10.04
18	215	102346	329781	308543	28.19
19	218	102718	288163	309680	9.51
20	242	161812	664348	487130	11.70
21	245	52965	198872	160610	5.95
22	264	123276	647996	371676	21.15
23	275	137097	431231	413216	7.17
24	279	108350	566407	327003	8.79
25	287	78746	249074	238247	48.70
26	**288**	**144656**	**468541**	**435984**	18.76
27	296	128191	436350	386645	21.19
28	298	106503	308635	321595	7.74
29	**383**	**283343**	**889911**	**852710**	5.34
30	427	188990	870721	569959	9.05
31	432	187053	529780	564183	17.73
32	442	321216	1248290	966742	9.32
33	443	220506	716678	664619	43.97
34	457	149104	669109	450511	12.73
35	500	138369	593847	418607	24.59
36	532	99579	325242	302461	15.39
37	667	122898	359851	373363	13.92
38	849	668140	2857277	2010363	38.04
39	**1083**	**411913**	**1225846**	**1243320**	9.15
40	**1195**	**247500**	**1035611**	**750865**	39.37

*All data sets are randomly perturbed. Sets (**9, 26, 29, 39, 40**) are called (**afternoon**, **sanmarco**, **annecy**, **cliffhouse**, and **korpalace**) in order. The values in the error column are the RMS reprojection errors in pixels.*

back-substitution, respectively. In the case of P*, we allowed one more iteration for the core-EPI to compensate for the absence of the back-substitution. No robustifier was used in any bundler. The above notation for describing our tested algorithms is used in Figs. 6, 7, 8, and 9. In Fig. 5, we used a different notation that describes the type of preconditioning that we attempted as explained in the next section.

6.1 Experiments on Synthetic Data Sets

Prior to performing experiments that test various settings for the bundler, it is necessary to fix an appropriate preconditioner for each CG solver. To determine the best preconditioner, we investigated the total time (Fig. 5a) and the number of CG iterations (Fig. 5b) that were required to solve the RCS in one bundle iteration. In the "(A)_(B)_(C)" format used in Fig. 5, (A) is the block bandwidth for the block-based preconditioner H_n or a type of scalar-based preconditioner (JACOBI or SSOR), (B) is the block or scalar type, and (C) is the variable reordering algorithm used.

According to Fig. 5, "SSOR_S" ((4), $\omega = 1$) was clearly better than "JACOBI_S" (3) when the scalar CG was used, but it was very difficult to determine the fastest preconditioner among the tested block-based preconditioners. All of the block-based preconditioned CGs were approximately 10 times faster than the scalar CGs, and even the simplest one, "1_B" ($P = H_1^{-1}$, i.e., block-based Jacobi preconditioning), which preconditioned the main block diagonal only, saved more iterations than the scalar SSOR preconditioner. Although the number of CG iterations differed, all of the tested block-based preconditioners showed similar overall performance. "1_B" was a good choice for our experimental purposes because it is independent of orderings and provides predictable performance (showing a stably increasing plot in Fig. 5). It could also be favorable for practical implementation because of its simplicity. In all of our experiments, "1_B" and "SSOR_S" were applied to each block-based and scalar CG, respectively.

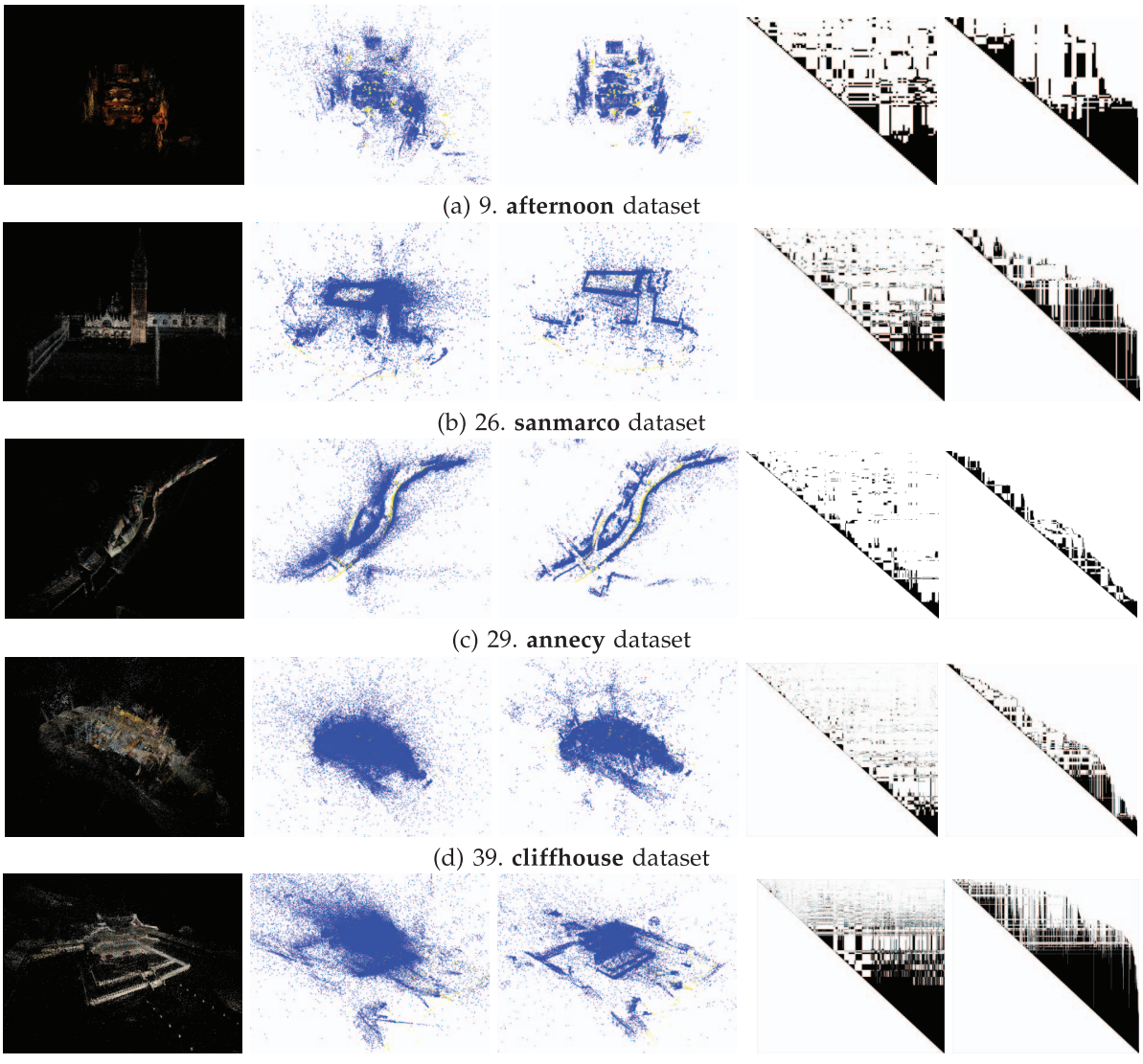
Fig. 6 shows how the computation times increased for six different bundlers with an increasing number of cameras. The partial and total times for one iteration were measured and plotted to compare the distribution of time over four steps. These were building the RCS, linear solving, back-substitution, and computing costs. As shown in the left column of Fig. 6a, the computation time of LDL quickly dominated the total time as the number of cameras increased. The right column in Fig. 6a shows that CG took considerably less time than LDL. The block-based CG was the fastest and did not dominate the total time. Fig. 6b shows the time for the "total" of the six bundlers in Fig. 6a, i.e., the top blue curve, which is grouped by the number of cameras. It should be noted that the proposed "B_LDL" and "B_CG" were the fastest in each category, and "B_CG" took less than 1 second, even with a highly occupied 576×576 block (6×6) matrix.

6.2 Experiments on Real Data Sets

In this section, we show that the previously mentioned results of the block-based LDL and PCG are valid for a number of real data sets. We also demonstrate the effect of EPIs and their variants on the convergence of a bundler.

In practice, waiting until the bundler converges completely is unacceptable and requires too much time in a massive experiment. Therefore, we allowed every bundler to proceed for a fixed but sufficiently large number of iterations. To observe where each bundler converged, 200 iterations were forced to proceed.

We also simulated the case in which a bundler is stopped when it reaches a fairly low level of error. This type of analysis provides useful information for the practical use of bundlers in which the processing time is as important as the value of the final error. Because our data sets were obtained at different image resolutions, we rescaled every image such that its longer side had an 800-pixel dimension and calculated the RMS reprojection error in pixels. Considering the final errors obtained by all bundlers, 1.0 pixel RMS reprojection error was selected as a proper mid-point at which the performances of all of the bundlers could be compared and used for the comparison shown in Table 2. For a few data sets whose minimum error found by all bundlers were larger than 1.0 pixels, we set the satisfactory level of



(a) 9. **afternoon** dataset

(b) 26. **sanmarco** dataset

(c) 29. **annecy** dataset

(d) 39. **cliffhouse** dataset

(e) 40. **korpalace** dataset

Fig. 4. Five out of 40 data sets were allowed to be published. In each row, "original point cloud," "perturbed points and cameras," "adjusted points and cameras," and "filled-in Hessian patterns after the MD and AMD ordering" are depicted from left to right. While the Hessian patterns in (c) and (d) show high sparsity and imply a dominant sequential relation between cameras, other patterns have multidimensional links between many of the cameras. The sparsity of (e) is the lowest.

error as the minimum error with the addition of a 0.2 pixel margin. For the rest of this section, we use "1.0 pixel error" to represent all of the satisfactory levels of errors.

We tested approximately 40 different block-based and scalar-based bundler variants with various linear solvers, orderings, preconditionings, EPIs, and lambda control strategies. Table 2 shows the set of results for eight representative bundlers. For the data sets in Groups 5 & 6 (Table 3) including **cliffhouse(39)** and **korpalace(40)**, scalar bundlers could not be applied because of the memory limitations of the Win32 application. The columns {1, 4, 7, 8} should show an identical number of iterations in each row (the values in parentheses) if a scalar or block-based CG computes an ideal solution in the same way that an LDL does and so should columns {2, 5} and {3, 6}. Nevertheless, the columns with CG showed differences in several data sets because of its inexact solution caused by the stopping criterion. However, those differences had no crucial effect and still provided faster convergence. As shown in Table 3, the number of data sets for which CG-based bundlers reached the desired error first is substantially larger than that of the LDL-based bundlers. The group-wise statistic in the table also demonstrates that using CG is more efficient for larger data sets. However, it is noteworthy that using a block-based LDL is still favorable for data sets with low complexity (Groups 1 & 2) because it provides an exact solution.

The EPIs reduced the number of iterations very effectively despite the use of the nonrobustified cost. This could be caused by ill-constrained points or the points having relatively large reprojection errors. Allowing for one more core-EPI iteration instead of the back-substitution ("P*") increased the convergence speed. We did not explicitly investigate this phenomenon, but it was probably due to the different linearizations in which the point updates were computed. As mentioned in Section 5, the cameras were

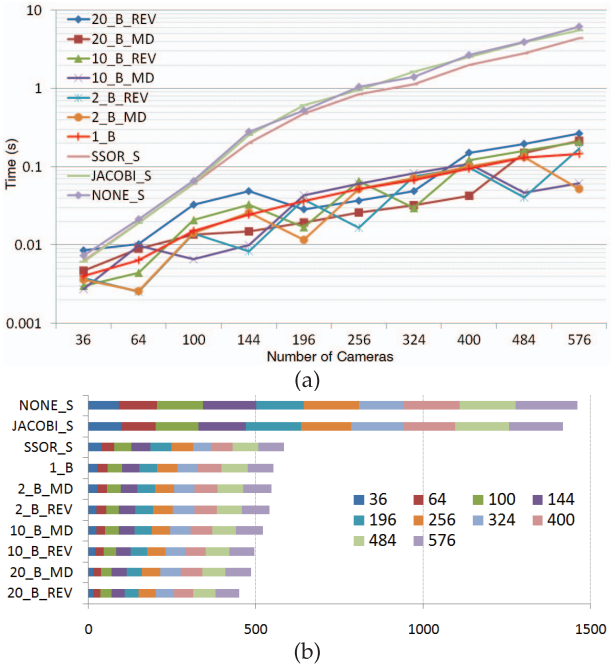


Fig. 5. (a) Time for CG convergence (sec) versus number of cameras. An explanation of the "(A)_(B)_(C)" format of legend is given in the beginning of Section 6.1. (b) The number of CG iterations needed to converge for various CG settings. SSOR and block-based preconditioners require approximately 3 times fewer iterations for convergence than NONE and JACOBI.

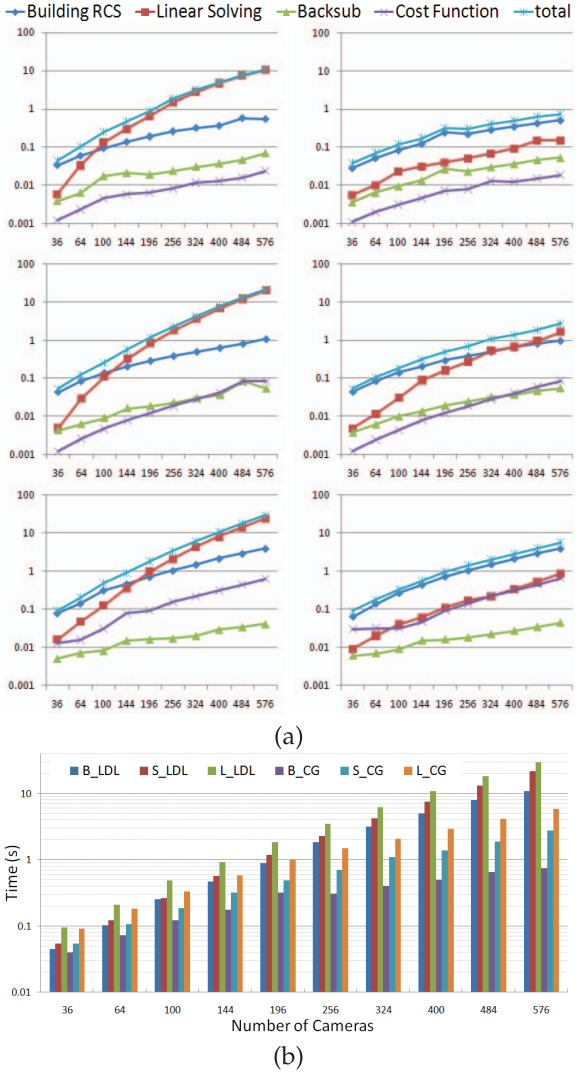


Fig. 6. Time (sec) of each step in one iteration for the synthetic data sets. (a) Block-based (top), scalar-based (middle), and conventional (bottom) bundlers with LDL (left) and CG (right) as the linear solvers. Minimum degree ordering was used for B_LDL and S_LDL. (b) Total time with respect to the number of cameras for six bundlers is shown.

better constrained and, by their nature, they settled down quickly. The core-EPI used the linearization that was computed with the updated cameras, whereas the back-substitution uses the linearization computed with the previous cameras. Therefore, we speculate that the proposed EPIs performed better because it made the points follow the recently updated cameras, which were more reliable than the previous points and were placed better than the previous cameras. Considering the fact that the bundlers with EPIs rarely failed to reach 1.0 pixel error (failures are counted and shown as the numbers in parentheses in Table 3), EPIs are helpful not only for saving iterations, but also for achieving better convergence.

The times required for one iteration of the five free data sets are also compared in Fig. 7a as they were in Fig. 6b for the synthetic data sets. Aside from the reduced number of iterations, the EPIs had low costs, and our block-based bundlers (both direct LDL and iterative CG) were faster than the scalar ones even with the EPIs.

For the linear solving, the number of cameras and the sparsity of the 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 increases as the number of cameras increases, and the dense Hessian matrix widens the gap between LDL and CG. A direct comparison for the time for linear solving in Fig. 7b may help the reader to understand this effect. **korpalace** and **cliffhouse** have similar complexities, but **korpalace** has a denser Hessian matrix than **cliffhouse** (refer to Table 1 and Fig. 4). **korpalace** causes a much larger gap between the "B_LDL" and "B_CG" than **cliffhouse** in Fig. 7. This difference means that the proposed block-based preconditioned CG becomes more crucial when problems not only get bigger but also become denser.

Fig. 8a shows the entire version of Table 2. In this case, lower is better, and the lines of "B_CG_P*" and "B_LDL_P*" are obviously the two lowest ones in all of the data sets. 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*" shows the best result over all data sets in terms of time and convergence speed.

We also show that using the exact minimum degree ordering and block-based Jacobi preconditioner for LDL and CG-based bundlers, respectively, is suitable. A comparison focused on variable ordering is depicted in Fig. 8b. It should be noted that the four bundlers that are compared in Fig. 8b followed one common convergence path and only differed in time because the ordering did not affect the result of the LDL solving. Therefore, their plotted lines converge. Because of the increasing complexity, the absolute gap between lines widens in the latter data sets.

TABLE 2
Elapsed Time (Sec) and Number of Iterations (in Parentheses) to Reduce RMS Reprojection Error to a Satisfactory Level (Mostly 1.0 Pixels)

set	Block-based LDL (B_LDL)			Block-based CG (B_CG)			S_LDL	S_CG
	BS only	BS&EPI (_P)	EPI only (_P*)	BS only	BS&EPI (_P)	EPI only (_P*)	BS only	BS only
1	0.48 (17)	0.39 (8)	**0.13 (2)**	0.53 (17)	0.41 (8)	0.14 (2)	0.56 (17)	0.57 (17)
2	**0.12 (1)**	0.20 (1)	0.18 (1)	0.15 (1)	0.20 (1)	0.18 (1)	0.13 (1)	0.13 (1)
3	17.23 (131)	**6.91 (36)**	11.65 (65)	18.69 (143)	7.43 (39)	N/A	23.49 (131)	18.65 (131)
4	2.89 (89)	0.65 (12)	**0.14 (2)**	3.44 (106)	0.64 (12)	0.15 (2)	7.82 (89)	4.49 (99)
5	0.24 (11)	0.15 (5)	**0.04 (1)**	0.23 (11)	0.16 (5)	0.05 (1)	0.85 (11)	0.42 (11)
6	0.17 (1)	0.30 (1)	0.28 (1)	**0.16 (1)**	0.30 (1)	0.28 (1)	0.23 (1)	0.23 (1)
7	1.57 (23)	1.82 (18)	0.15 (1)	1.33 (23)	1.61 (18)	**0.15 (1)**	3.06 (23)	1.81 (23)
8	0.32 (5)	0.13 (1)	0.13 (1)	0.30 (5)	0.14 (1)	**0.13 (1)**	0.73 (5)	0.51 (5)
9	2.89 (36)	0.67 (5)	**0.20 (1)**	2.77 (36)	0.67 (5)	0.21 (1)	6.14 (36)	3.58 (36)
10	0.31 (3)	0.25 (1)	**0.25 (1)**	0.52 (5)	0.26 (1)	0.26 (1)	0.81 (3)	0.55 (2)
11	1.53 (17)	1.18 (10)	**0.41 (3)**	1.53 (17)	1.16 (10)	0.44 (3)	4.31 (17)	2.67 (17)
12	0.16 (2)	0.18 (1)	0.18 (1)	**0.16 (2)**	0.18 (1)	0.18 (1)	0.49 (2)	0.32 (2)
13	1.81 (5)	1.21 (2)	0.69 (1)	1.78 (5)	1.20 (2)	**0.69 (1)**	3.19 (5)	2.19 (5)
14	4.64 (15)	0.99 (2)	0.55 (1)	4.26 (15)	0.95 (2)	**0.55 (1)**	9.05 (15)	53.38 (164)
15	2.07 (13)	1.15 (5)	0.30 (1)	1.46 (13)	0.91 (5)	**0.26 (1)**	6.43 (13)	2.63 (13)
16	1.74 (11)	0.62 (2)	0.38 (1)	1.56 (11)	0.59 (2)	**0.37 (1)**	6.21 (11)	2.86 (11)
17	3.22 (42)	**0.27 (2)**	0.39 (3)	2.52 (42)	0.65 (6)	0.39 (3)	21.53 (42)	4.52 (29)
18	N/A	**34.80 (101)**	N/A	N/A	59.67 (186)	57.80 (192)	N/A	N/A
19	1.48 (8)	0.48 (1)	**0.43 (1)**	1.49 (9)	0.69 (2)	0.43 (1)	6.80 (8)	2.75 (8)
20	14.40 (26)	5.07 (6)	1.16 (1)	13.63 (26)	4.29 (6)	**1.01 (1)**	37.46 (26)	16.15 (26)
21	11.76 (69)	4.61 (20)	**0.84 (3)**	9.56 (69)	4.08 (20)	0.97 (4)	79.51 (69)	19.39 (69)
22	44.75 (62)	2.15 (2)	2.27 (2)	37.63 (62)	2.86 (3)	**2.15 (2)**	107.67 (62)	45.43 (65)
23	5.64 (8)	1.86 (2)	1.05 (1)	2.27 (7)	0.74 (1)	**0.73 (1)**	13.59 (8)	4.34 (8)
24	11.19 (17)	4.25 (5)	1.01 (1)	9.33 (17)	3.73 (5)	**0.95 (1)**	33.36 (17)	12.97 (17)
25	2.71 (3)	2.10 (2)	2.09 (2)	**0.85 (3)**	0.86 (2)	0.87 (2)	5.45 (3)	3.52 (3)
26	43.74 (125)	6.09 (12)	4.43 (8)	36.32 (126)	5.46 (12)	**4.04 (8)**	241.79 (125)	64.70 (125)
27	26.00 (75)	3.14 (6)	0.70 (1)	20.71 (72)	2.93 (6)	**0.68 (1)**	155.14 (75)	38.99 (75)
28	21.72 (94)	2.19 (6)	1.21 (3)	17.06 (94)	1.97 (6)	**1.17 (3)**	185.99 (94)	37.44 (94)
29	41.67 (78)	2.86 (3)	**1.25 (1)**	40.37 (78)	5.95 (7)	1.27 (1)	328.21 (78)	66.12 (78)
30	19.35 (6)	3.90 (1)	3.74 (1)	6.09 (3)	2.66 (1)	**2.50 (1)**	44.07 (6)	21.70 (12)
31	4.15 (6)	2.92 (3)	1.18 (1)	2.28 (6)	2.13 (3)	**0.96 (1)**	34.56 (6)	8.53 (6)
32	N/A	12.11 (7)	2.27 (1)	N/A	9.85 (7)	**1.99 (1)**	N/A	N/A
33	N/A	44.58 (53)	28.92 (35)	N/A	41.70 (60)	**22.56 (33)**	N/A	N/A
34	38.27 (18)	37.14 (16)	2.59 (1)	12.98 (18)	9.35 (10)	**1.47 (1)**	124.93 (18)	22.52 (18)
35	N/A	115.89 (84)	4.54 (3)	N/A	20.70 (27)	**3.28 (3)**	N/T	N/T
36	186.63 (45)	88.90 (21)	4.36 (1)	20.34 (45)	12.78 (21)	**1.12 (1)**	N/T	N/T
37	N/A	25.95 (48)	31.43 (61)	N/A	**20.04 (57)**	22.38 (67)	N/T	N/T
38	1326.2 (149)	146.00 (15)	10.95 (1)	345.21 (149)	50.00 (15)	**4.81 (1)**	N/T	N/T
39	N/A	44.61 (28)	4.09 (2)	N/A	33.18 (28)	**3.72 (2)**	N/T	N/T
40	N/A	1701.23 (94)	649.50 (36)	N/A	186.96 (94)	**81.67 (46)**	N/T	N/T

Bold font indicates the minimum time for each row, and N/A stands for the case in which the desired pixel error is not achieved within 200 iterations. N/T means "not tested."

The block-based LDL with the exact minimum degree ordering was clearly better than the one using reverse Cuthill-McKee ordering (REV) and was the fastest.

Another comparison that was focused on preconditioning is depicted in Fig. 8c. As shown previously in Fig. 5, the bundlers using block-based CG with various preconditioners showed similar performances. This proves that using the block-based Jacobi preconditioner is also a good choice for real data sets.

In addition, we plotted convergence curves (time versus cost curves) of the eight bundlers in Table 2 for all real data sets to verify that using CG and EPIs is safe in terms of the final convergence. The plots for the five free data sets are shown in Fig. 9. We verified that bundlers using EPIs ("P" and "P*") achieved lower minima than bundlers using back-substitution alone over all data sets. During this process, we observed an interesting case. For a few data sets including **sanmacro** and **korpalace**, the proposed bundlers using EPIs

TABLE 3
Group-Wise Counting of Data Sets for Which Each Bundler Is the Fastest or Fails to Get Close to the Minimum Error, i.e., "N/A in the Table 2" (in Parentheses)

group	Block-based LDL (B_LDL)			Block-based CG (B_CG)			S_LDL	S_CG
	BS only	BS&EPI (_P)	EPI only (_P*)	BS only	BS&EPI (_P)	EPI only (_P*)	BS only	BS only
group 1 & 2	1 (0)	2 (0)	6 (0)	2 (0)	0 (0)	6 (1)	0 (0)	0 (0)
group 3 & 4	0 (3)	1 (0)	3 (1)	1 (3)	0 (0)	7 (0)	0 (3)	0 (3)
group 5 & 6	0 (4)	0 (0)	0 (0)	0 (4)	1 (0)	10 (0)	0 (N/A)	0 (N/A)
total	1 (7)	3 (0)	9 (1)	3 (7)	1 (0)	23 (1)	0 (N/A)	0 (N/A)

N/A in this table means that counting is not possible.

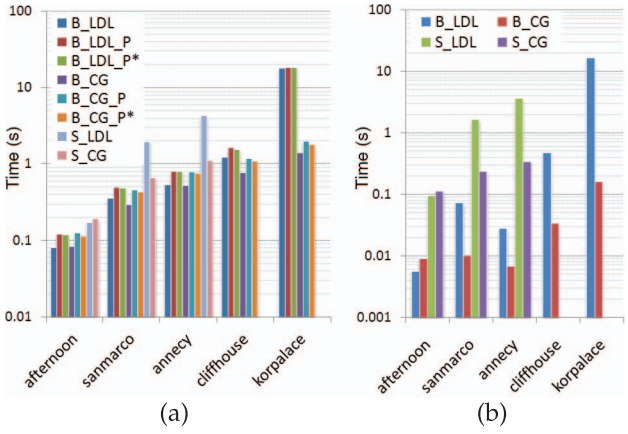


Fig. 7. Time (sec) for (a) one bundle iteration and (b) one linear solving of the five free data sets. As explained, the scalar solver could not be performed on the two data sets on the right.

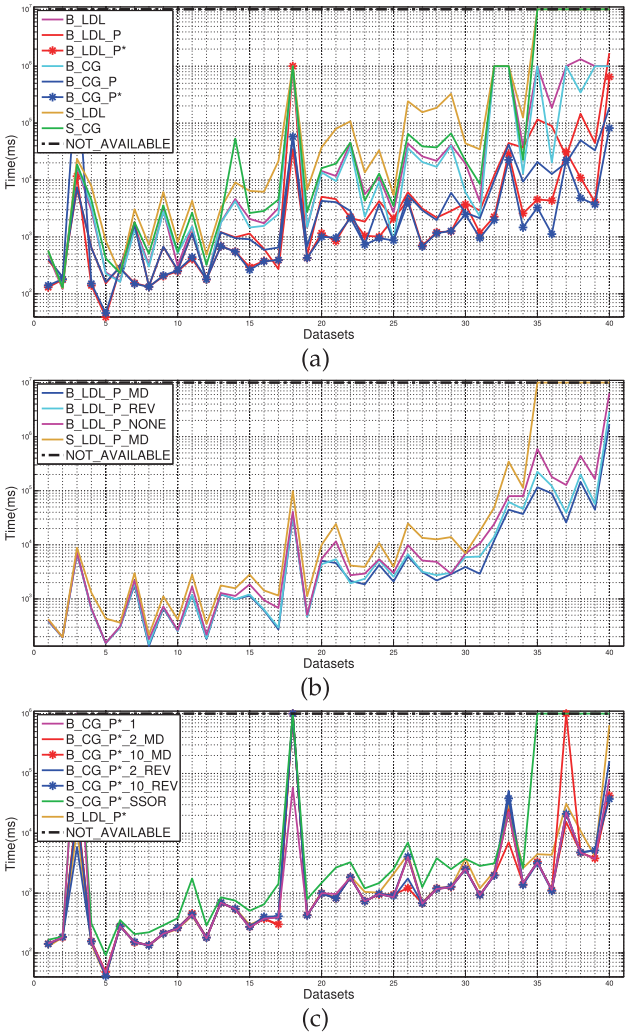


Fig. 8. Time to reach 1.0 pixel RMS reprojection error. The plotted lines meet "NOT_AVAILABLE" when the corresponding setting of the bundler cannot achieve 1.0 pixel error for the data sets or was not tested. (a) Eight representative bundlers, (b) bundlers using LDL with different reorderings, and (c) bundlers using CG with different preconditioners are compared.

with back-substitution ("P") reached a lower level of error than bundlers using EPIs ("P*") alone even though the "P*"-bundlers reduced errors more rapidly during the early

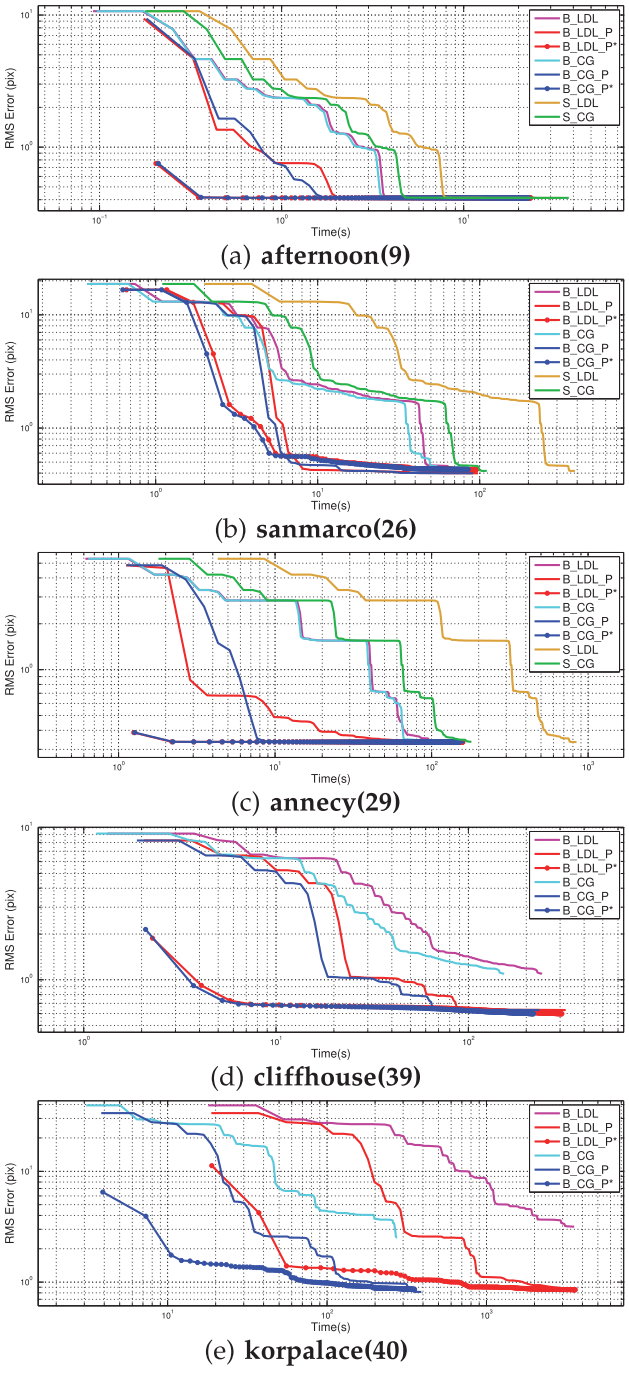


Fig. 9. Convergence plots for the five free data sets. All plots show the time and the error on the x and y-axes, respectively, using a log scale. The final errors of bundlers with EPIs ("P" and "P*") are normally equivalent to each other and lower than those of other bundlers. In terms of convergence speed, bundlers with EPIs only ("P*") are the fastest.

iterations. This indicates that solving and updating the cameras and points together (back-substitution) might help to find a narrow pathway to the lower minimum. Therefore, a hybrid use of "P*" and "P"- bundlers could be considered to accomplish the fastest convergence speed accompanied with the lowest error if a proper switching scheme is found. From another perspective, the EPIs' fast movement during the early iterations could have led the bundler to an unpromising minimum. In this case, greater damping of the EPIs could also be considered to slow the movement.

7 Conclusion and Future Work

This paper provides three main contributions, namely, adapting a 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, of which the effectiveness and the stability are proven by our experiments. Concurrently, 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 substantial improvement, particularly when the complexity of a problem is high. 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 should be addressed in a future study. First, the effect of the λ control strategy has not been fully investigated. A comparison between the suggestion given in [3] and LMs with various strategies would also be interesting. Another task is parallelizing the proposed method. All of our substeps except for linear solving can be computed in parallel in a straightforward manner with the 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. Finally, a method to mix the back-substitution with the core-EPI could obtain a faster cost decrease and a lower minimum simultaneously.

Acknowledgments

This work was partially done while Yekeun Jeong was an intern at Microsoft Research. This work was supported by the MKE, Korea, under the Human Resources Development Program for Convergence Robot Specialists support program supervised by the NIPA (NIPA-2011-C7000-1001-0007) and a National Research Foundation grant funded by the Korean government (MEST) (No. 2011-0018250).

References

[1] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle Adjustment—A Modern Synthesis," *Proc. Int'l Workshop Vision Algorithms: Theory and Practice,* B. Triggs, A. Zisserman, and R. Szeliski, eds.,, pp. 298-372, 2000.

[2] 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, Inst. of Computer Science-FORTH, Heraklion, Crete, Greece, Aug. 2004.

[3] M. Lourakis and A. Argyros, "Is Levenberg-Marquardt the Most Efficient Optimization Algorithm for Implementing Bundle Adjustment?" *Proc. IEEE Int'l Conf. Computer Vision,* pp. 1526-1531, 2005.

[4] K. Ni, D. Steedly, and F. Dellaert, "Out-of-Core Bundle Adjustment for Large-Scale 3D Reconstruction," *Proc. IEEE Int'l Conf. Computer Vision,* 2007.

[5] M. Byröd and K. Åström, "Bundle Adjustment Using Conjugate Gradients with Multiscale Preconditiong," *Proc. British Machine Vision Conf.,* 2009.

[6] S. Agarwal, N. Snavely, I. Simon, S.M. Seitz, and R. Szeliski, "Building Rome in a Day," *Proc. IEEE Int'l Conf. Computer Vision,* 2009.

[7] S. Agarwal, N. Snavely, S.M. Seitz, and R. Szeliski, "Bundle Adjustment in the Large," *Proc. European Conf. Computer Vision,* vol. 6312, pp. 29-42, 2010.

[8] M. Byröd and K. Åström, "Conjugate Gradient Bundle Adjustment," *Proc. European Conf. Computer Vision, Part II,* vol. 6312, pp. 114-127, 2010.

[9] K. Levenberg, "A Method for the Solution of Certain Non-Linear Problems in Least Squares," *The Quarterly of Applied Math.,* vol. 2, pp. 164-168, 1944.

[10] H.-Y. Shum, Z. Zhang, and Q. Ke, "Efficient Bundle Adjustment with Virtual Key Frames: A Hierarchical Approach to Multi-Frame Structure from Motion," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* pp. 2538-2543, 1999.

[11] D. Steedly and I.A. Essa, "Propagation of Innovative Information in Non-Linear Least-Squares Structure from Moion," *Proc. IEEE Int'l Conf. Computer Vision,* pp. 223-229, 2001.

[12] D. Steedly, I.A. Essa, and F. Dellaert, "Spectral Partitioning for Structure from Motion," *Proc. IEEE Int'l Conf. Computer Vision,* pp. 996-1003, 2003.

[13] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real Time Localization and 3D Reconstruction," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* pp. 363-370, 2006.

[14] C. Engels, H. Stewénius, and D. Nistér, "Bundle Adjustment Rules," *Proc. Conf. Photogrammetric Computer Vision,* Sept. 2006.

[15] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," *Proc. Int'l Symp. Mixed and Augmented Reality,* 2007.

[16] S. Holmes, G. Sibely, G. Klein, and D.W. Murray, "A Relative Frame Representation for Fixed-Time Bundle Adjustment in SfM," *Proc. IEEE Int'l Conf. Robotics and Automation,* 2009.

[17] A. Eudes and M. Lhuillier, "Error Propagations for Local Bundle Adjustment," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* 2009.

[18] "Photosynth," http://photosynth.net, 2012.

[19] Y. Jeong, D. Nistér, D. Steedly, R. Szeliski, and I.-S. Kweon, "Pushing the Envelope of Modern Methods for Bundle Adjustment," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* 2010.

[20] R.I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision,* second ed. Cambridge Univ. Press, 2004.

[21] D.C. Brown, "The Bundle Adjustment—Progress and Prospects," *Int'l Archives of Photogrammetry,* vol. 21, no. 3, pp. 3-33, 1976.

[22] F. Dellaert and M. Kaess, "Square Root Sam: Simultaneous Location and Mapping via Square Root Information Smoothing," *Int'l J. Robotics Research,* vol. 25, pp. 1181-1203, 2006.

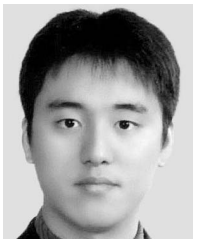
[23] G. Golub and C.V. Loan, *Matrix Computations,* Johns Hopkins Studies in Math. Sciences, third ed. JHU Press, 1996.

[24] N. Snavely, S. Seitz, and R. Szeliski, "Photo Tourism: Exploring Photo Collections in 3D," *ACM Trans. Graphics,* vol. 25, no. 3, pp. 835-846, 2006.

[25] T. Davis, *Direct Methods for Sparse Linear Systems.* SIAM, 2006.

[26] Y. Saad, *Iterative Methods for Sparse Linear Systems,* second ed. SIAM, 2000.

[27] M.J. Black and A. Rangarajan, "The Outlier Process: Unifying Line Processes and Robust Statistics," *Proc. IEEE CS Conf. Computer Vision and Pattern Recognition,* 1994.



Yekeun Jeong received the BS degree in electrical engineering and computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2006, where he is currently working toward the PhD degree in the field of computer vision and robotics. His research interests include structure from motion, bundle adjustment, and their applications to large-scale 3D modeling with multiple sensors and mobile robots. He is a recipient of the Samsung Human-tech Thesis Prize Bronze Award and the National Research Foundation of Korea scholarship. He is a student member of the IEEE.

David Nistér received the PhD degree in computer vision, numerical analysis, and computing science from the Royal Institute of Technology (KTH), Stockholm, Sweden, with the thesis "Automatic Dense Reconstruction from Uncalibrated Video Sequences," in 2001. He is a principal scientist and head of the Augmented Reality group at Bing Mobile, Microsoft. His research interests include computer vision, computer graphics, structure from motion, multiple view geometry, Bayesian formulations, tracking, recognition, image and video compression. Before joining Microsoft, he was an assistant professor at the Center for Visualization and Virtual Environments and the Computer Science Department at the University of Kentucky. He is a recipient of the US National Science Foundation (NSF) CAREER Award, and was a principal investigator on the US Defense Advanced Research Projects Agency (DARPA) project Urbanscape. He was a researcher in the Vision Technologies Laboratory, Sarnoff Corporation, Princeton, New Jersey. He worked at Visual Technology, Ericsson Research, Stockholm, Sweden, at Prosolvia Clarus, Gothenburg, Sweden, specializing in virtual reality, and at Compression Lab, Ericsson Telecom. He is a member of the IEEE and American Mensa.

Drew Steedly received the MSc degree in electrical engineering in 1996 and the PhD degree in computer science in 2004, both from the Georgia Institute of Technology. He has worked as a scientist and software architect at Microsoft since 2004. At Microsoft, he has worked on multiple image stitching projects, led the development of the 3D reconstruction software in Photosynth, and been the photogrammetry architect for the Bing maps aerial and streetside mapping projects. He has published numerous papers in computer vision, 3D reconstruction from images, and image stitching. From 1996 to 2004, he worked at Integrated Device Technology as both a CMOS design engineer and design automation engineer. He is a member of the IEEE.

Richard Szeliski received the PhD degree in computer science from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 1988. He leads the Interactive Visual Media Group at Microsoft Research, which does research in digital and computational photography, video scene analysis, 3D computer vision, and image-based rendering. He joined Microsoft Research in 1995. Prior to Microsoft, he worked at Bell-Northern Research, Schlumberger Palo Alto Research, the Artificial Intelligence Center of SRI International, and the Cambridge Research Lab of Digital Equipment Corp. He has published more than 150 research papers in computer vision, computer graphics, medical imaging, and neural nets, as well as the books *Computer Vision: Algorithms and Applications* and *Bayesian Modeling of Uncertainty in Low-Level Vision*. He was a program committee chair for the IEEE International Conference on Computer Vision (ICCV '01) and the 1999 Vision Algorithms Workshop. He served as an associate editor of the *IEEE Transactions on Pattern Analysis and Machine Intelligence* and on the editorial board of the *International Journal of Computer Vision* and is also a founding editor of *Foundations and Trends in Computer Graphics and Vision*. He is a fellow of the ACM and the IEEE.

In-So Kweon received the BS and MS degrees in mechanical design and production engineering from Seoul National University, Seoul, Korea, in 1981 and 1983, respectively, and the PhD degree in robotics from the Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, in 1990. He worked for the Toshiba R&D Center, Japan, and joined the Department of Automation and Design Engineering, KAIST, Seoul, Korea, in 1992, where he is now a professor with the Department of Electrical Engineering. He is a recipient of the best student paper runner-up award at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '09). His research interests are in camera and 3D sensor fusion, color modeling and analysis, visual tracking, and visual SLAM. He was the program cochair for the Asian Conference on Computer Vision (ACCV '07) and is the general chair for ACCV '12. He is also on the editorial board of the *International Journal of Computer Vision*. He is a member of the IEEE and the KROS.