CoLi-BA: Compact Linearization based Solver for Bundle Adjustment

Zhichao Ye, Guanglin Li, Haomin Liu, Zhaopeng Cui, *Member, IEEE*, Hujun Bao, *Member, IEEE*, and Guofeng Zhang, *Member, IEEE*

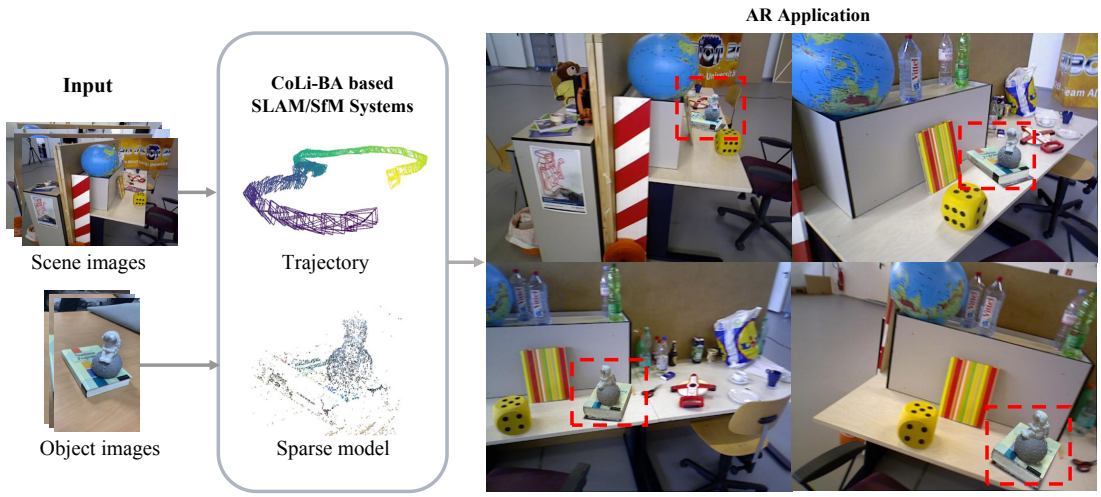

Fig. 1. We propose CoLi-BA, an efficient BA solver to accelerate the optimization process in SLAM and SfM. Employing CoLi-BA in a single thread, SLAM takes only 20ms in local BA, and reconstructing an object by 100 frames only consumes 6s. Based on the recovered camera trajectory, we show an AR demo by placing a reconstructed object on a user-selected position of the desk. The virtual object is highlighted with red boxes.

Abstract— Bundle adjustment (BA) is widely used in SLAM and SfM, which are key technologies in Augmented Reality. For real-time SLAM and large-scale SfM, the efficiency of BA is of great importance. This paper proposes CoLi-BA, a novel and efficient BA solver that significantly improves the optimization speed by compact linearization and reordering. Specifically, for each reprojection function, the redundant matrix representation of Jacobian is replaced with a tiny 3D vector, by which the computational complexity, memory storage, and cache missing for Hessian matrix construction and Schur complement are significantly reduced. Besides, we also propose a novel reordering strategy to improve the cache efficiency for Schur complement. Experiments on diverse datasets show that the speed of the proposed CoLi-BA is five times that of Ceres and two times that of g2o without sacrificing accuracy. We further verify the effectiveness by porting CoLi-BA to the open-source SLAM and SfM systems. Even when running the proposed solver in a single thread, the local BA of SLAM only takes about 20ms on a desktop PC, and the reconstruction of SfM with seven thousand photos only takes half an hour. The source code is available on the webpage: `https://github.com/zju3dv/CoLi-BA`.

Index Terms—Bundle adjustment, Compact linearization, Schur complement, SLAM, Structure-from-Motion

1 Introduction

Augmented Reality (AR) requires real-time motion tracking of mobile devices for rendering 3D objects at the correct position, and also massive virtual resources to enrich user experiences in various applications. The former is typically realized by Simultaneous Localization and Mapping (SLAM) [7, 21, 26, 33]. The latter can be generated at a low cost by capturing objects or scenes from photos with Structure-from-Motion (SfM) [1, 35, 43]. The key technique behind SLAM and SfM is Bundle Adjustment (BA) [39], which jointly optimizes the 3D structure and camera poses such that the rays from 3D points to cameras can be coincident with observations on images. Due to the massive dimension of parameter space, the efficiency of BA is the main bottleneck for both real-time SLAM and large-scale SfM. Therefore, how accelerating BA is a crucial issue for AR applications.

Essentially, BA can be regarded as a nonlinear least square problem, which is solved iteratively by linearizing the cost function. In each iteration, a linear system related to the Hessian matrix is constructed and solved, which takes up most of the calculations. Thanks to the sparsity of problem that each reprojection function only relates to one camera and one point, the Hessian matrix has a particular sparsity pattern and can be constructed efficiently. Besides, the special linear system can be solved efficiently with the Schur complement trick [46]. In the direction of utilizing the sparsity pattern, [14, 22, 39] have done in-depth

- *Z. Ye, G. Li, Z. Cui, H. Bao and G. Zhang are with the State Key Lab of CAD&CG, Zhejiang University. E-mail: {yezhichao_cad, liguanglin, zhpcui, baohujun, zhangguofeng}@zju.edu.cn.*
- *H. Liu is with SenseTime Research. E-mail: liuhaomin@sensetime.com.*
- *Corresponding Author: Guofeng Zhang.*

Manuscript received 11 March 2022; revised 11 June 2022; accepted 2 July 2022. Date of publication 01 September 2022; date of current version 03 October 2022. Digital Object Identifier no. 10.1109/TVCG.2022.3203119

research. To further improve the efficiency, [18, 21, 34] leverage the incremental nature of SLAM and propose to update Schur complement incrementally. Other researchers tend to solve BA in parallel by multi-core CPU/GPU [30, 44], or in distributed manners [12, 47], typically for the large-scale SfM.

However, the past works do not make full use of the geometric meaning of BA that the Euclidean distance between the observation of 2D feature points and the projection of 3D map points in the image. Different from previous methods, we provide a new perspective to accelerate BA. The observation is that the 2×9 matrix representation for the Jacobian of reprojection error is redundant and can be expressed by 6 degrees of freedom (DoF). Geometrically, in the 6DoF representation of Jacobian, 3 degrees of freedom correspond to rotation, and 3 degrees of freedom relate to the ray from the camera to the point. Inspired by this, we propose a 3D compact representation for linearization by slightly modifying the reprojection error to fully exploit the geometry properties of bundle adjustment. For each reprojection function, there are simple formulas that the relevant Hessian blocks can be represented by the 3D compact vectors completely. Therefore, the amount of calculation and storage space is significantly reduced.

The other observation is that for large-scale problems, there are severe cache misses caused by large sparse matrix multiplication, which significantly affect the efficiency of BA. Although the memory storage and cache misses have been reduced by the compact representation, we further find that the ordering of parameters also greatly impacts the cache efficiency during the construction of Schur complement. We describe the calculation process of Schur complement in detail, and quantify the impact of parameter permutation on the calculation efficiency, then formulate and solve the optimal problem to further improve the efficiency.

Based on the proposed compact linearization and reordering strategy, we propose an efficient BA solver, called CoLi-BA. The proposed solver can be easily integrated into the framework of SLAM and SfM. Using the SLAM system and SfM system with CoLi-BA, the scene camera trajectory and the sparse model of objects can be obtained quickly. We show an AR demo that places a toy model on the desk of another scene in Fig. 1.

To sum up, our major contributions are as follows:

- We propose a novel 3D compact representation of Jacobian matrix and Hessian matrix for each reprojection error, by which the computational complexity, memory storage, and cache misses for linear system construction are greatly reduced.
- We propose a novel reordering strategy that can find a better parameter permutation to accelerate the memory access speed in Schur complement. The proposed method improves the Schur complement speed by 25% without sacrificing accuracy.
- Based on compact linearization and reordering strategy, we propose an efficient BA solver, CoLi-BA. Experiments on the sequential images and Internet photo collection verify the effectiveness of the proposed solver, whose speed is five times that of Ceres and two times that of g2o without sacrificing accuracy.
- We replace the optimizers on open-source systems (e.g., ORB-SLAM2, COLMAP) with CoLi-BA. Extensive experiments show that the proposed method is efficient and robust, which can be directly applied to SLAM and SfM.

2 Related Work

Bundle adjustment is critical and widely used in many systems, especially in SLAM [7, 8, 11, 21, 26, 33, 40, 41] and SfM [1, 20, 24, 35, 43]. In SLAM systems, local and global BA are often used to reduce accumulated drifts. [26] performs frequent local BA to optimize keyframes and keypoints, thus improving the accuracy of SLAM. Meanwhile, the efficiency of BA has become the bottleneck of this kind of SLAM system. Large-scale reconstructions methods [35] often take BA as the last step to refine camera poses and points. Recent proposed SfM methods based on deep features [20] still use BA to refine cameras and points and improve the accuracy of reconstruction. The process of large-scale reconstructions often takes hours, and BA accounts for a large part of the time.

Bundle adjustment can be solved as a nonlinear least-squares problem by iteratively computing and updating the linear approximation at the current state. Thanks to the sparsity of the problem, Schur complement can construct a reduced linear system, reducing linear solution time, which is a standard step in many BA solver [22, 39, 48]. Besides, the sparse structures also contribute to matrix operations, g2o [14] use graph structure to represent the sparse relation between residuals and parameters that use graph optimization to solve problems efficiently. Some large-scale matrix solving algorithms with pure numerical optimization, such as PCG, is introduced to accelerate further the speed of BA solver [3, 5]. The speed of optimization is highly related to the number of parameters. Facing large scenes, the convergence becomes extremely slow, resulting in many offline reconstruction systems time-costing [3, 35, 45]. For real-time applications, it is unrealistic to carry out global optimization frequently, so only the cameras and 3D points related to the current frame will be added to a local optimization, which ensures accuracy. Even for the optimization in such a small window, the existing methods are difficult to achieve real-time operation on the lightweight computing platform.

To meet the real-time challenge of application, the incremental strategy [19] is proposed, which updates the square root factor directly, so that avoid unnecessary calculations. To further enhance incremental update, specially designed data structures are proposed, such as Bayes tree [18]. [34] presents an incremental implementation of the Powell's Dog-Leg [32] numerical optimization method, which is robust to objective function nonlinearity and numerical ill-conditioning. [17] solves the fast covariance recovery with an incremental framework. [21] leverages the sparseness and banded matrix structure for the optimization of sliding window and global optimization and proposes an efficient VIO system. Due to fixing the linearization of some parameters, there is often a slight sacrifice in accuracy for incremental strategy. Another method [6] uses multiple camera poses to represent points, so as to build a pose-only problem to improve the speed. In essence, it is an approximation of the original cost function, and the accuracy will also be reduced.

With the development of hardware such as multi-core CPU and GPU, many works realized distributed or parallel implementations to accelerate BA. [27] speed up the BA consider solving in a distributed manner and [44] restructured and parallelized the matrix production used in the PCG iterations. Considering the characteristics of block matrices, [31] takes advantage of the block matrix manipulation and the efficiency of the cache. [30] accelerated the sparse matrix multiplication on a GPU. Besides, deep learning methods based on differentiable and end-to-end training [37, 38] provide another approach for improving the robustness and efficiency of BA.

Unlike the previous methods, this paper reviews the linearization from the perspective of geometry, and proposes a compact linearization method. Besides, the reordering strategy is proposed to alleviate the memory access efficiency problem in large sparse matrix multiplication.

3 Method

3.1 Framework of Bundle Adjustment

Before going into details of the proposed method, we introduce the general implementation of BA. As shown in Fig. 2, the general framework of BA solver is mainly composed of four parts. In the Jacobian evaluation module, the solver computes the residuals and Jacobians at the current state of parameters for each reprojection function. Then, the solver constructs a linear system which is a linear approximation of the nonlinear cost function. Schur complement is used here to reduce the linear system size, also called point marginalization. After the linear system is constructed, there are many methods for solving the linear system, such as Cholesky factorization [9], QR factorization [15] and PCG [10]. The linear solution consists in potential parameter increments.Before parameters are actually updated by the increments, it is also necessary to check whether the linear solution is reliable and the total cost is decreased. The linear solving and parameter update is

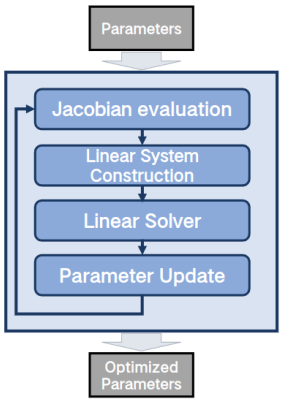

Fig. 2. The framework of a general BA solver.

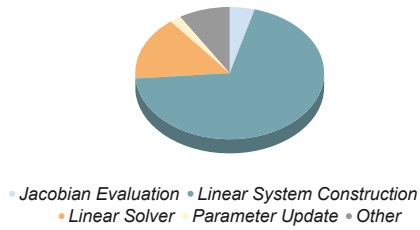

Fig. 3. Time proportion of each part of BA in Ceres Solver. The problem has $4,541$ cameras and $646,971$ points, and linear system construction dominates the overall calculation time.

performed iteratively, until the problem convergence or the maximum number of iterations. In the process described above, some details vary depending on the minimization strategy being used, such as Gauss-Newton (GN) [4], Levenberg–Marquardt(LM) [23], and Dogleg [28]. The famous open-source frameworks Ceres and g2o follow this framework. According to our test, the calculation time is mainly spent on the process of constructing and solving the linear system, as shown in Fig. 3. The proposed method mainly focuses on accelerating the construction, which occupies the most time.

3.2 Compact Linearization

Bundle adjustment aims to optimize camera poses and 3D points by minimizing the reprojection error between the 2D observation of a 3D point and its projection onto the image plane by the camera pose. Since each reprojection function relates only to one camera and one point, the problem has a special sparse structure. Previous methods mainly focused on the sparse structure, while few researchers considered its geometric properties. In fact, from the perspective of geometry, problems often have more concise expressions and great acceleration potentials. For example, the rotation matrix has nine elements but only 3 degrees of freedom. For the interpolation or multiplication operation of two rotations, using quaternion expression instead of rotation matrix can simplify the computation. The 3D Lie algebra captures the essence of rotation multiplication, and can effectively deal with the infinitesimal transformation and iterative optimization problems of rotation.

In previous works, the linearization of reprojection function means to compute a 2×9 Jacobian matrix, where 2 and 9 are the dimensions of the error and parameter, respectively. The observation is that the Jacobian only depends on the rotation and the ray from the camera center to the point, corresponding to 6 degrees of freedom. Therefore, the matrix representation is redundant. In this section, we propose a compact representation for linearization to speed up.

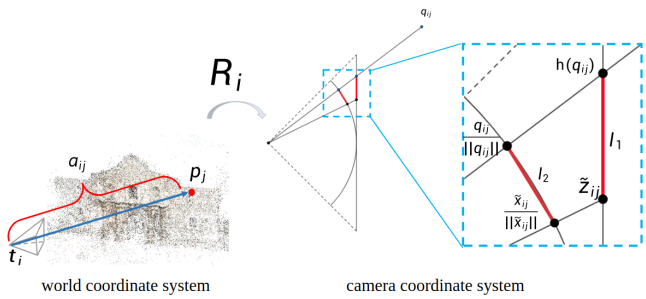

Fig. 4. Visualization of the reprojection error. The point p_j is transformed from the world coordinate system to the camera coordinate system q_{ij}, then projected to the normalized plane by function $h(q_{ij})$. We enlarged the part in the blue dotted box and marked red on the geometric vectors l_1 and l_2 corresponding to the reprojection errors in the normalized plane and the unit sphere. The ray in the world coordinate system is denoted by a_{ij} which represents the vector from t_i to p_j.

3.2.1 Modified reprojection error

Given camera pose set $\mathbf{C}=\{c_i|i=1...n_c\}$ and point set $\mathbf{P}=\{p_i|i=1...n_p\}$, the cost function of BA problem can be formulated as

$$f(\mathbf{C},\mathbf{P})=\frac{1}{2}\sum_k||e_k(c_i,p_j)||^2, \tag{1}$$

where $e_k(c_i,p_j)$ is the reprojection error function which projects p_j onto the image normalized plane of c_i and computes the distance between the projection z_{ij} and the 2D observation $\tilde{z}_{ij}$. The reprojection error can be defined as

$$\begin{aligned} e_k(c_i,p_j) =& z_{ij}-\tilde{z}_{ij}=h(q_{ij})-\tilde{z}_{ij},\\ q_{ij} =& R_i a_{ij},\\ a_{ij} =& p_j-t_i,\\ h([x,y,z]^T) =& [x/z,y/z]^T, \end{aligned} \tag{2}$$

where (R_i,t_i) is the rotation and translation of camera c_i, a_{ij} is the ray from camera center t_i to the point p_j. The reprojection function first rotates the ray a_{ij} from world to camera coordinate to obtain q_{ij}, then projects q_{ij} to the normalized plane by the homogeneous normalization function $h(\cdot)$. The process is illustrated in Fig. 4.

The Jacobian of Eq. (2) is composed of three components $J_{ij}=[J_{ij}^{\varphi},J_{ij}^{t},J_{ij}^{p}]$ with respect to rotation, translation and point respectively, which can be calculated with the chain rule:

$$\begin{aligned} J_{ij}^{\varphi}&=\frac{\partial e_k}{\partial \varphi_i}=\frac{\partial e_k}{\partial q_{ij}}\frac{\partial q_{ij}}{\partial \varphi_i},\\ J_{ij}^{t}&=\frac{\partial e_k}{\partial t_i}=\frac{\partial e_k}{\partial q_{ij}}\frac{\partial q_{ij}}{\partial t_i},\\ J_{ij}^{p}&=\frac{\partial e_k}{\partial p_i}=\frac{\partial e_k}{\partial q_{ij}}\frac{\partial q_{ij}}{\partial p_j}, \end{aligned} \tag{3}$$

where φ_i is the right perturbations of rotation. The first parts in Eq. (3) correspond to the homogeneous normalization which are common for the three components, and are computed as:

$$\frac{\partial e_k}{\partial q_{ij}}=\begin{bmatrix}\frac{1}{z} & 0 & -\frac{x}{z^2}\\ 0 & \frac{1}{z} & -\frac{y}{z^2}\end{bmatrix}, \tag{4}$$

where (x,y,z) are three elements of q_{ij}. The second parts correspond

to rigid transformation, calculated as

$$\begin{aligned}\frac{\partial q_{ij}}{\partial \varphi_i} &= -R_i[a_{ij}]_\times,\\ \frac{\partial q_{ij}}{\partial t_i} &= -R_i,\\ \frac{\partial q_{ij}}{\partial p_j} &= R_i.\end{aligned} \tag{5}$$

The rigid transformation part has 6 degrees of freedom, 3 for rotation R_i and 3 for the ray a_{ij}. The normalization part also depends on this 6DoF, since $q_{ij} = R_i a_{ij}$. The Jacobian will be used to compute Hessian by multiplication among the three components. Although the Jacobian has only 6 degrees of freedom, due to the troublesome calculation of normalization part $\frac{\partial e_k}{\partial q_{ij}}$, it is more efficient to use the redundant matrix representation for the later multiplication operations.

In order to improve the efficiency, we use spherical reprojection error, which is applied in VINS-Mono [33] and SfM for fisheye cameras [16, 29]. Previous work [29] has shown that using the modified reprojection is adequate and leads to better results than a reprojection on the normalized plane when using panoramic cameras. However, the potential improvements on efficiency are not exploited in previous works. Compared with the traditional reprojection error, the spherical reprojection error modify the normalization part by projecting q_{ij} onto a unit sphere instead of the normalized plane:

$$e_k = \frac{q_{ij}}{||q_{ij}||} - \frac{\tilde{x}_{ij}}{||\tilde{x}_{ij}||} = R_i \frac{a_{ij}}{||a_{ij}||} - \frac{\tilde{x}_{ij}}{||\tilde{x}_{ij}||}, \tag{6}$$

where $\tilde{x}_{ij} = [\tilde{z}_{ij}, 1]^T$ is the homogeneous coordinate of $\tilde{z}_{ij}$. As shown in Fig. 4, both errors measure the distance between ray a_{ij} and observation $\tilde{x}_{ij}$, but in different domains.

After modification, the Jacobian becomes

$$\begin{aligned}J_{ij}^{\varphi} &= -R_i[\bar{a}_{ij}]_\times,\\ J_{ij}^{t} &= -s_{ij}R_i(I - \bar{a}_{ij}\bar{a}_{ij}^T),\\ J_{ij}^{p} &= s_{ij}R_i(I - \bar{a}_{ij}\bar{a}_{ij}^T),\end{aligned} \tag{7}$$

where $s_{ij} = \frac{1}{||a_{ij}||}$ and $\bar{a}_{ij} = \frac{a_{ij}}{||a_{ij}||}$. Compared with the original cost function, the modified Jacobian has the same 6DoF, but consists of three matrices having strong geometric properties:

- rotation matrix R_i: can be transposed to get its inverse, as an orthogonal matrix.
- antisymmetric matrix $[\bar{a}_{ij}]_\times$: can be represented by cross-production.
- projection matrix $(I - \bar{a}_{ij}\bar{a}_{ij}^T)$: can project vectors to the vertical plane of $\bar{a}_{ij}$.

Leveraging these geometric properties will significantly accelerate the construction of the Hessian matrix and Schur complement.

3.2.2 3D compact representation

As shown in Fig. 5 ,the Hessian matrix of BA is a square matrix of dimension $(N_c + N_p)$, that $N_c = 6n_c, N_p = 3n_p$. Note, the top-left part is a diagonal block matrix $A = \text{diag}(H_1^{cc} ... H_{n_c}^{cc})$ with i-th block H_i^{cc} corresponding to the i-th camera. Similarly the bottom-right part is also a diagonal block matrix $B = \text{diag}(H_1^{pp} ... H_{n_p}^{pp})$ with j-th block H_j^{pp} corresponding to the j-th point. The top-right block matrix D is also a sparse matrix, with non-zero block (i, j) corresponding to the reprojection function of i-th camera observing j-th point, denoted as

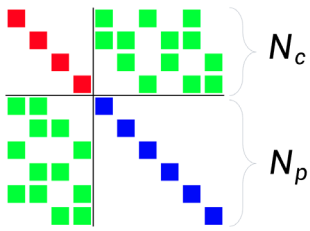

Fig. 5. An example of the Hessian matrix of BA. Small colorful squares represent the blocks of the Hessian matrix that red square is 6×6 matrix, the blue square is 3×3 matrix, and the green square of the upper right corner is 6×3 matrix.

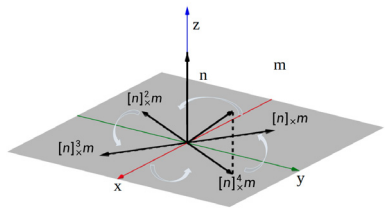

Fig. 6. The cyclic group of cross production in the vertical plane of vectors. Suppose n is a unit vector, m is an arbitrary vector, and i is a positive integer. Vector $[n]_\times^i m$ is always in the vertical plane of n, vectors $[n]_\times^i m$ and $[n]_\times^{i+1} m$ are always orthogonal. Besides, the projection of m is equivalent to the four consecutive cross production.

H_{ij}^{cp}. The blocks of the Hessian matrix can be computed by

$$\begin{aligned}H_i^{cc} &= \sum_j {J_{ij}^c}^T J_{ij}^c,\\ H_j^{pp} &= \sum_i {J_{ij}^p}^T J_{ij}^p,\\ H_{ij}^{cp} &= {J_{ij}^c}^T J_{ij}^p,\end{aligned} \tag{8}$$

where $J_{ij}^c = [J_{ij}^R, J_{ij}^t]$.

Because the rotations of Jacobians are on the left side of the formula in the Eq. (7), it will be eliminated in the Hessian matrix. Take H_j^{pp} as an example:

$$\begin{aligned}H_j^{pp} &= \sum_i (s_{ij}R_i(I - \bar{a}_{ij}\bar{a}_{ij}^T))^T (s_{ij}R_i(I - \bar{a}_{ij}\bar{a}_{ij}^T))\\ &= \sum_i s_{ij}^2 (I - \bar{a}_{ij}\bar{a}_{ij}^T)^T R_i^T R_i (I - \bar{a}_{ij}\bar{a}_{ij}^T)\\ &= \sum_i s_{ij}^2 (I - \bar{a}_{ij}\bar{a}_{ij}^T)^T (I - \bar{a}_{ij}\bar{a}_{ij}^T).\end{aligned} \tag{9}$$

By constructing $e_k' = R_i e_k$, the rotation of ${J_{ij}^c}^T e$ and ${J_{ij}^p}^T e$ can also be eliminated. Then we can compute the Hessian matrix merely with antisymmetric matrix and projection matrix. Since the antisymmetric matrix and projection matrix can be represented by a_{ij}, it means that we can replace the 2×9 Jacobian matrix in linearization by merely a 3D vector which we call compact representation.

3.2.3 Cyclic group

There are a lot of multiplications about antisymmetric matrix and projection matrix if we substitute the Eq. (7) into the Eq. (8). Since the direct calculation is very time-consuming, we introduce the geometric properties of cross-production and projection to simplify the form and reduce the amount of calculation. Given a 3D unit vectors $n =$

(n_1,n_2,n_3), the matrix $[n]_\times$ satisfies

$$\begin{aligned}[n]_\times^2 &= \begin{bmatrix} -n_2^2-n_3^2 & n_1n_2 & n_1n_3 \\ n_2n_1 & -n_1^2-n_3^2 & n_2n_3 \\ n_3n_1 & n_3n_2 & -n_1^2-n_2^2 \end{bmatrix} = nn^T - I \\ [n]_\times^3 &= [n]_\times(nn^T-I) = [n]_\times nn^T - [n]_\times = -[n]_\times \\ [n]_\times^4 &= [n]_\times[n]_\times^3 = [n]_\times(-[n]_\times) = I - nn^T \end{aligned} \tag{10}$$

and thus the multiplications about $[n]_\times$ and $I-nn^T$ can be expressed in terms of $[n]_\times^i$. In fact, for arbitrarily positive integer i, we have

$$\begin{aligned} k &= i \pmod 4, t = (i-k)/4, \\ [n]_\times^i &= [n]_\times^{4t}[n]_\times^k = (I-nn^T)^t[n]_\times^k = [n]_\times^k. \end{aligned} \tag{11}$$

Thus there is a cyclic group $G = \{[n]_\times^k \mid k = 0,1,...,3\}$ with $[n]_\times^0 = I - nn^T$. To understand the cyclic group better, we also illustrate it in Fig. 6. All vectors $[n]_\times^k m$ are located on the vertical plane of n and the vector $[n]_\times^{k+1}m$ can be obtained by rotating vector $[n]_\times^k m$ by 90 degrees in the plane.

With $\hat{a}_{ij} = s_{ij}\bar{a}_{ij}$, Eq. (8) can be simplified as:

$$\begin{aligned} H_i^{cc} &= \sum_j \begin{bmatrix} -[\bar{a}_{ij}]_\times^2 & -[\hat{a}_{ij}]_\times \\ [\hat{a}_{ij}]_\times & -[\hat{a}_{ij}]_\times^2 \end{bmatrix} \\ H_j^{pp} &= \sum_i -[\hat{a}_{ij}]_\times^2 \\ H_{ij}^{cp} &= \begin{bmatrix} [\hat{a}_{ij}]_\times \\ [\hat{a}_{ij}]_\times^2 \end{bmatrix} \end{aligned} \tag{12}$$

Compared with direct matrix multiplication, 3D compact representation requires less computation, but does not sacrifice accuracy. In addition, the block H_{ij}^{cp} can also be represented by the tiny 3D vector $\hat{a}_{ij}$, and does not need to be explicitly calculated or stored.

3.2.4 Schur complement

Schur complement is a classic method for constructing a reduced linear system, which has become a standard step in the BA problem. The reduced system decreases the time of solving but adds the computational overhead for the construction. For large problems, the computing time of Schur complement is dominated by the construction of the linear system.

Consider Schur complement in the Hessian matrix:

$$\begin{pmatrix} A & D \\ D^T & B \end{pmatrix} \begin{pmatrix} \delta a \\ \delta b \end{pmatrix} = \begin{pmatrix} \varepsilon_a \\ \varepsilon_b \end{pmatrix}, \tag{13}$$

$$\begin{aligned} S\delta a &= \varepsilon_a - D'\varepsilon_b, \\ S &= A - D'D^T, \\ D' &= DB^{-1}. \end{aligned} \tag{14}$$

The above formula presents a reduced matrix equation which can greatly decrease the time of solving. However, the calculation process of $A-D'D^T$ is quite time-consuming. Since A and B^{-1} is diagonal block matrix, the calculation of $A-X$ and XB^{-1} is relatively fast, and the main time-consuming part is the multiplication of two large sparse matrices in $D'D^T$. Denoting the blocks of $D'D^T$ by d_{ij},

$$d_{i_1i_2} = \sum_j^{j\in T_{i_1i_2}} H_{i_1j}^{cp} H_j^{pp-1} H_{i_2j}^{cpT}, \tag{15}$$

where $T_{i_1i_2}$ is the set of common points between camera (i_1,i_2). In each item of Eq. (15), the number of floating-point multiplication is 162 that 54 of the first matrix multiplication $H_{i_1j}^{cp} \times H_j^{pp-1}$ and 108 of the second matrix multiplication $(H_{i_1j}^{cp}H_j^{pp-1}) \times H_{i_2j}^{cpT}$. With our compact representation, multiplication can be simplified. Given a 3D vector x,

$$H_{ij}^{cp}x = \begin{pmatrix} \hat{a}_{ij} \times x \\ \hat{a}_{ij} \times (\hat{a}_{ij} \times x) \end{pmatrix} \tag{16}$$

so the calculation of $H_{ij}^{cp}x$ in the compact representation only needs two cross productions. Compared with the original 3×6 matrix, the amount of floating-point multiplication of $H_{ij}^{cp}x$ is reduced from 18 to 12. The amount of floating-point multiplication in Eq. (15) is reduced by one-third, from 162 to 108 for each point.

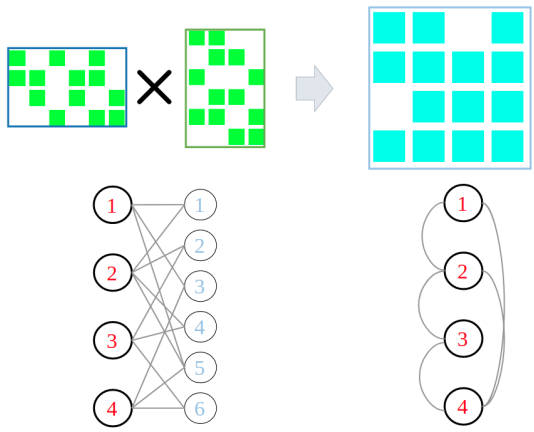

Fig. 7. The block structure of matrices D corresponds to the visibility between cameras and points, which a bipartite graph on the left can represent. Red/blue nodes are cameras/points, and the edge represents the point is detected in the camera. The graph on the right represents the structure of S. Nodes are cameras, and an edge represents the two cameras that have common points.

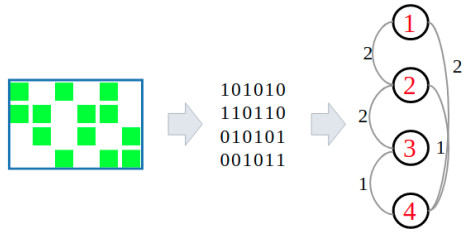

Fig. 8. We can use four binary codes to represent the row structure of matrix D. The optimal permutation problem can be transformed into a graph, and the weights of edges are the similarity of different rows. In this example, 3214 is one of the best permutations of rows.

3.3 Reordering Strategy

In Schur complement, memory access is one of the bottlenecks to improving computing speed. Even if adopting sparse matrix storage, there still be severe cache misses for large-scale problems. In order to solve this problem, we propose a reordering strategy to make the data arrangement more suitable for efficient memory access. Different from previous works of reordering parameters [39] aiming to reduce the calculation of solving the linear system, our goal is to speed up the construction by improving the memory access efficiency. In this section, we first explain why the matrix permutation can speed up memory access, then formulate the optimal permutation problem, give a heuristic solution based on graph theory, and finally discuss some details of implementation.

Matrix permutation. Eq. (15) involves $H_{i_1j}^{cp}$ and $H_{i_2j}^{cp}$ that are in the i_1-th row and the i_2-th row of block matrix D. Denote T_i as the set

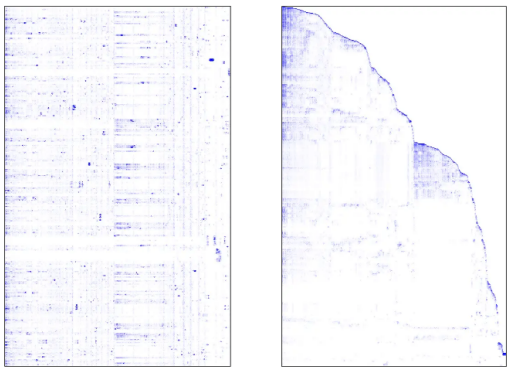

Fig. 9. The structure of D before and after reordering in real data Alamo. The matrix which has $792 \times 50,000$ blocks, is represented by a image with resolution of 792×500, and each pixel corresponds to 100 blocks. The pixel color indicates the number of non-zero blocks. The white pixel means there is no non-zero block. The darker the blue color, the more non-zero blocks it contains. After reordering, the structure becomes more compact.

including column indexes of non-zero blocks of i-th row of D. There is

$$T_{i_1 i_2} = T_{i_1} \cap T_{i_2}. \tag{17}$$

Even if T_{i_1} and T_{i_2} have thousands of elements, most $T_{i_1 i_2}$ have no or few elements. To compute the few elements in $T_{i_1 i_2}$, we have to read data from thousands of elements in T_{i_1} and T_{i_2}. If the involved blocks are stored compactly, we only need to access a small area to read them. Unfortunately, these elements are generally scattered throughout the row, resulting in inefficient memory access. By changing the column permutation of block matrix D, we can make the non-zero elements of indexes in $T_{i_1 i_2}$ close to each other to alleviate this problem. The row permutation is similar. Since each element H_{ij}^{cp} of D corresponds to a constraint between cameras and points, reordering the rows or columns of D is equivalent to changing the index of cameras or points. Fig. 7 shows the structure of D corresponds to a bipartite graph that consists of cameras and points.

Problem definition. How to reorder to accelerate $D'D$ is a fascinating topic. Due to the inconsistent memory access rules on different hardware, it is not easy to use one model to quantitatively estimate the time under different permutations to select a valid optimal permutation. Therefore, we propose a qualitative optimization of the data permutations. The plain idea is that two rows/columns with similar structures should be adjacent. We define the problem as minimizing the structure difference between adjacent rows/columns according to this idea. We can use a string of binary b_i^r to represent the data structure of i-th row in the block matrix D that **1** indicates a non-zero element and **0** indicates an empty block. A simple example of the binary representation is given in Fig. 8. With the binary strings, the problem is formulated as

$$\min_{\mathbf{O}} \sum b_{o_i}^r \wedge b_{o_{i+1}}^r, \tag{18}$$

where $\mathbf{O} = o_1 \cdots o_n$, which is a permutations of $1 \cdots n$.

Graph representation. It is not easy to solve the original problem directly, so we convert it to a weighted graph. Graph G is defined that each node is the structure of a row, and the weight of the edge between the two nodes b_i^r, b_j^r is set to $b_i^r \wedge b_j^r$. An example of this graph representation is also illustrated in Fig. 8. The problem is to find a path P that passes through all nodes and has the highest weight, that is, the maximum Hamiltonian path on graph G. Finding the maximum Hamiltonian path is a variant of the symmetric traveling salesman problem (TSP), which is an NP-hard problem. Finding the exact solution to the problem is unrealistic, leading to the reordering time greater than the multiplication of the sparse matrix. Although there are many solutions for TSP, we finally choose the nearest neighbor heuristic algorithm that treats the nearest neighbor as the next node to avoid the excessive time of calculating the optimal permutation. Denote i-th row of D by D_i. We select an arbitrary row as the first node D_{o_1}. Then we recursively set the node in the remaining rows that has the most similar structure with D_{o_i} as the next node $D_{o_{i+1}}$.

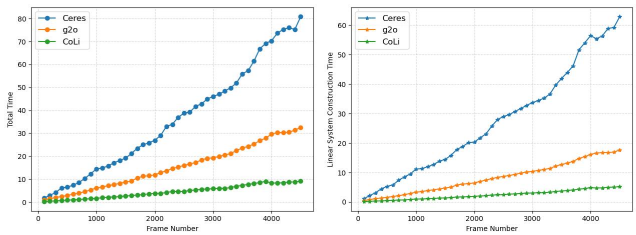

Fig. 10. Time comparison under different problem sizes (Left). We intercept $100 \sim 4,500$ frames on Seq.00 to construct BA problems of different sizes. As the problem size increases, our calculation time increases more smoothly than other methods. Linear construction time comparison under different problem sizes (Right). It can be found that the trend of linear construction time and total time is similar. The time difference between different methods is mainly caused by the linear construction time.

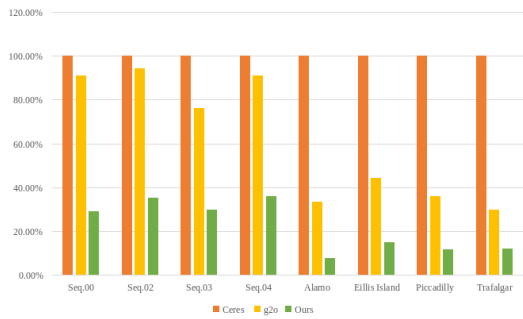

Fig. 11. The statistics of cache misses for different datasets. We count the number of cache misses, and use the result of Ceres for normalization such that Ceres is always 100%.

Implementation details. For camera reordering, directly computing the hamming distance between binary strings is time-consuming because the size of the string equals the total number of points in the whole scene. In the implementation, we evaluate the similarity by counting the number of common points between two cameras. For point reordering, finding the most similar next point for each point is also time-consuming because there are too many candidates. We propose to use the bucket sorting algorithm to reorder points directly according to the minimal index of the camera connecting to the point. In order to better show the effect of reordering on data arrangement, we illustrated a real example in Fig. 9, visualizing the change of D matrix before and after reordering. After sorting, the original messy data becomes more compact and regular.

4 Experiment

We evaluate the proposed CoLi-BA on real datasets, including sequential images (KITTI dataset [13]) and unordered images (1DSfM dataset [42]). Firstly, the performance is compared to the open-source optimizer Ceres [2] and g2o [14]. Then, the ablation experiments of the compact linearization and the reordering strategy are carried out respectively. Finally, we test the performance of the proposed method in the SLAM/SfM systems. The experiments are conducted on a desktop PC with an Intel i7-9700K 3.6GHz CPU, 64GB of memory.

Table 1. Performance comparison of BA solvers.

	#cameras	#points	#constraints	#Memory[MB]			#Time[s]			#Avg. Reproj. Error [px]		
				ceres	*g2o*	*CoLi*	*ceres*	*g2o*	*CoLi*	*ceres*	*g2o*	*CoLi*
Seq.00	4541	646971	5149157	3288	4145	1380	81.43	32.44	12.61	0.54	0.52	0.52
Seq.01	1101	204186	1083097	854	948	354	19.14	6.07	3.91	2.71	7.23	1.46
Seq.02	4661	1319942	7328965	4636	6169	1458	77.80	32.23	13.53	0.77	0.78	0.41
Seq.03	801	145589	1361058	815	1059	412	22.94	6.87	3.28	0.45	8.08	0.76
Seq.04	271	61210	390669	238	315	90	4.41	1.68	0.83	0.40	0.41	0.40
Seq.05	2761	347937	3086266	2038	2469	918	56.74	17.50	11.42	0.50	0.49	0.48
Seq.06	1101	310668	1913646	1267	1577	449	26.80	9.71	4.60	0.67	0.50	0.53
Seq.07	1101	242300	1566925	893	1247	324	20.32	6.89	2.61	0.88	4.45	0.53
Seq.08	4071	1095471	6237268	3942	5240	1264	60.94	27.17	10.42	1.66	7.63	0.47
Seq.09	1591	466468	2306450	1469	1984	473	19.74	9.61	3.50	0.92	0.72	0.46
Seq.10	1201	274900	2052222	1142	1546	383	25.24	9.70	3.97	1.01	2.08	0.44
Alamo	797	50030	559346	460	449	308	38.42	8.78	4.27	1.02	1.01	1.02
Ellis Island	394	39350	254437	159	191	76	6.79	1.70	0.75	1.16	1.16	1.63
Gendarmenmarkt	949	76820	499985	313	366	145	12.96	3.42	1.81	1.28	1.37	1.18
Madrid Metropolis	405	28236	170431	108	121	47	3.22	1.01	0.48	1.36	1.78	1.33
Montreal Notre Dame	556	56591	502472	325	385	198	21.93	4.90	2.26	1.19	1.19	1.18
NYC Library	509	37859	275573	157	185	67	6.43	2.17	0.87	1.30	1.91	1.02
Piazza del Popolo	895	47980	379416	231	256	132	14.82	3.35	2.64	1.05	1.06	1.04
Piccadilly	2874	141532	1248286	2213	1419	1097	139.97	30.85	17.02	1.14	1.21	1.10
Roman Forum	1407	104613	800409	559	617	254	23.92	9.69	3.04	2.09	1.27	1.14
Tower of London	584	47106	371717	225	277	98	7.67	2.66	1.31	1.03	1.02	1.00
Trafalgar	7082	264540	2667791	6129	3408	3005	436.34	102.82	49.80	1.19	1.19	1.22
Union Square	750	25050	213156	98	105	31	4.56	1.38	0.84	1.64	1.76	0.94
Vienna Cathedral	1049	96549	887375	612	691	387	52.06	11.95	5.20	1.08	1.08	1.07
Yorkminster	967	93883	648636	376	463	146	14.06	4.92	2.24	1.07	1.07	1.06

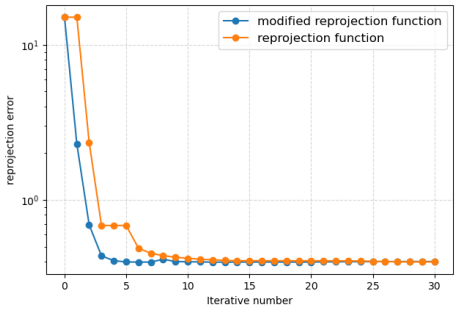

Fig. 12. The different convergence speed of reprojection function and modified reprojection function on KITTI Seq.02.

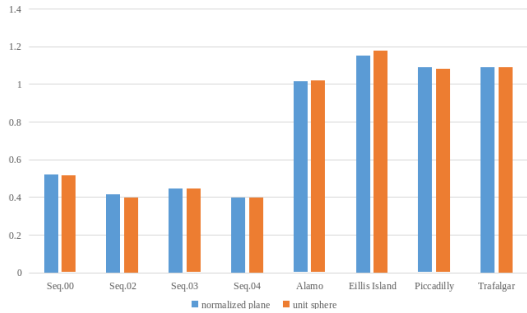

Fig. 13. Accuracy comparison for different cost functions. The modified cost function has comparable accuracy to the original reprojection function.

4.1 Performance Comparison

We compare the performance of the proposed CoLi-BA with two state-of-the-art methods (i.e., g2o and Ceres) on various datasets. All optimizers iterate five times with the convergence condition off. The adopted strategy is LM, and the linear solution method is PCG. The input to different solvers is the reconstruction result generated by COLMAP containing camera poses, 3D points, and 2D observations. Furthermore, we add Gaussian noise to the initial camera poses and 3D points to make the average reprojection error about 10 pixels. Ceres and g2o use the classic reprojection error, while we use the modified reprojection error defined on the sphere. As shown in Table 1, we record the problem size, the memory size for optimization, the overall optimization time, and the final reprojection error (the classic reprojection for a fair comparison) to analyze the differences in efficiency and accuracy. The memory size is computed by the total program memory subtracting the size of the input map. Thanks to compact linearization, our memory consumption is significantly reduced. Meanwhile, the speed of the proposed method is improved about two times than g2o and five times than Ceres. The impact of the problem size on speed is further discussed in Fig. 10. In all scale problems, the proposed method has a significant speed improvement, and this improvement mainly comes from the high efficiency of linear system construction. In terms of accuracy, CoLi-BA achieves the best performance on most of the datasets when considering limited iteration numbers of optimization, which is mainly due to the different convergence speeds of reprojection function and modified reprojection function. The error-iteration curves under different formulations of reprojection error are reported in Fig. 12. The loss of the spherical error decreases faster and tends to converge after 4 iterations while the baseline formulation spends 12 iterations achieving competitive accuracy.

In addition to time and accuracy, we quantitatively measure the number of caching misses of different methods in optimization using the performance analysis tool Perf. As shown in Fig. 11, the proposed method has a good memory access efficiency which is significantly superior to state-of-the-arts. The number of cache misses is reduced by four times compared to Ceres and twice compared to g2o.

4.2 Ablation Study

We perform the ablation study to investigate the effect of the proposed compact linearization and reordering strategy on accuracy and efficiency. We select 8 representative data according to problem size. Sequential data contains two long sequences (Seq.00, Seq.02) and two short sequences (Seq.03, Seq.04). Similarly, unordered data contains two small scenes (Alamo, Ellis Island) and two large-scale scenes (Piccadilly, Trafalgar).

4.2.1 Compact Representation

The proposed compact representation requires to slightly modify the reprojection function. To investigate the effect on reconstruction accuracy, we use the original reprojection error as the indicator of accuracy,

Table 2. Time Comparison for Compact Linearization.

	#Evaluation[s]		#Construction[s]		#Total Time[s]	
	baseline	*compact*	*baseline*	*compact*	*baseline*	*compact*
Seq.00	0.79	0.54	9.27	5.63	25.22	19.63
Seq.02	1.13	0.76	8.69	5.85	22.94	18.66
Seq.03	0.21	0.14	2.98	1.67	6.38	4.72
Seq.04	0.06	0.04	0.51	0.34	1.13	0.90
Alamo	0.87	0.06	4.73	1.72	7.74	4.71
Ellis Island	0.04	0.03	0.80	0.34	1.29	0.86
Piccadilly	0.21	0.15	14.16	5.99	47.71	40.40
Trafalgar	0.69	0.32	40.32	18.49	85.64	62.85

Table 3. Time Comparison for Different Reordering Strategies.

	#Linear System Construction Time[s]			
	baseline	*RC*	*RP*	*RC* + *RP*
Seq.00	9.27	9.07	7.72	7.59
Seq.02	8.69	8.27	7.35	7.45
Seq.03	2.98	2.94	2.36	2.30
Seq.04	0.51	0.49	0.43	0.42
Alamo	4.73	3.80	3.89	3.11
Ellis Island	0.80	0.63	0.67	0.56
Piccadilly	14.16	11.89	11.09	9.28
Trafalgar	40.32	33.47	33.08	26.70

Table 4. Performance comparison with the incremental method.

	#Time[s]		#Error[px]	
	inc. [17]	*ours*	*inc.* [17]	*ours*
Cathedral	66.88	26.49	7.05	2.12
Venice	3078.52	3250.59	10.07	2.47

and compare the accuracy with and without modification. To ensure convergence, we set the number of iterations to 50. As shown in Fig. 13, slightly modifying the cost function does not affect the final results.

To investigate the effect on efficiency, we compare the runtime with and without compact representation in Table 2. Both methods use the modified reprojection function. *baseline* denotes traditional matrix representation and *compact* denotes the proposed compact representation. Here the reordering is disabled to emphasize the effect of compact representation. Table 2 compares the Jacobian evaluation time (**Evaluation**), linear system construction time (**Construction**), and total optimization time (**Total Time**) between the two methods. The Jacobian evaluation occupies a small fraction of computation. The efficiency gain is mainly reflected in the linear system construction. We also find that the compact linearization on the unordered images has a more significant improvement compared to the sequential sequences. This is because the D matrix of Hessian for unordered data is relatively dispersed in memory. Using 3D vectors to represent the blocks of D brings more significant benefits.

4.2.2 Reordering Strategy

To analyze the efficiency of the reordering strategy, we disable the compact linearization, and compare the time of linear system construction under different strategies. As shown in Table 3, we compare four cases: no reordering *baseline*, only camera reordering *CR* (also called row reodering), only point reordering *PR* (also called column reodering), and both reordering $CR+PR$. It can be found that point reordering is very effective for all data sets. However, for camera reordering, the performance in unordered images is undeniable, but the linear system construction time of sequential sequence has little change. In sequence images, due to the large overlapping between adjacent frames, the original camera order is good enough, which makes camera reordering not significant. To better illustrate this point, we demonstrate the reordering difference of sequential images and unordered images in Fig. 14.

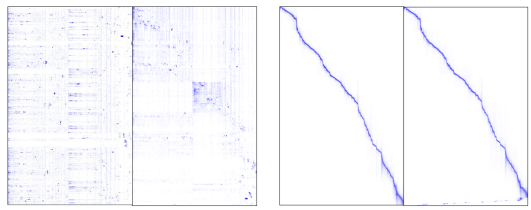

Fig. 14. The camera reordering results in unordered images and sequential images. The left two pictures represent the structure of matrix D before and after reordering rows in Alamo. The right two picture is the results in Seq.03. Compared with unordered images, camera reordering has a minor impact on sequential images.

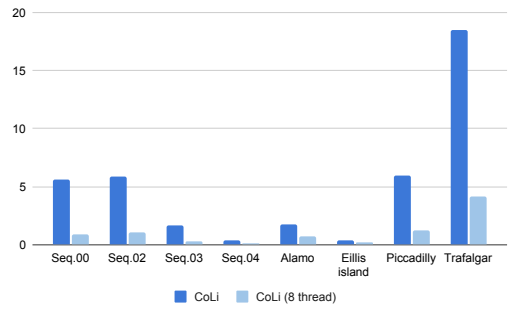

Fig. 15. Time comparison for multithreaded version. We recorded the linear construction time for the original CoLi-BA and 8 threads version.

4.3 Comparison of acceleration approaches

In addition to the comparison with the standard solver, we also analyze the advantages of the proposed method with other acceleration approaches.

Incremental methods. We compare with the incremental method [17] in two datasets Cathedral (92 frames) and Vince (871 frames), since these are the only datasets with the incremental version provided by its open-source warehouse. We run implementation of [17] with the incremental strategy and Schur complement activated. Note that without an incremental strategy, CoLi-BA always construct the linear equation from scratch when each frame is added to the problem. For a fair comparison, the maximum iteration number for each optimization of both solvers is set to 5. As shown in Table 4, the accuracy of the incremental method is inevitably affected by its approximation strategies, while our method consistently achieves better accuracy. Meanwhile, thanks to the proposed compact linearization, our method also achieves a decent speed even without an incremental strategy, and both our accuracy and speed exceed the results reported in [17]'s paper (i.e., Cathedral: 705.738s, 2.644px).

Parallelization methods. The parallelization methods accelerate BA with multithreading techniques (e.g., CPU based approaches such as MKL, OpenMP, and GPU based approaches such as CUDA and OpenCL), which can be easily applied to our method. Here we implement a multi-threaded CPU version CoLi-BA based on OpenMP. As shown in Fig. 15, our method gains significant performance improvement (with 467 % on average), which demonstrates the great potential of CoLi-BA when deploying on a parallelized framework.

Distributed methods. The distributed method divides the original BA problem into several subproblems to solve. For example, [47] build the subproblems with the ADMM approximation algorithm and use Ceres optimization to solve each subproblem. Nevertheless, minimizing the reprojection error is still a must for each subproblem. Therefore, the performance of these distributed methods can still benefit from our compact linearization without sacrificing accuracy.

4.4 Application for SLAM/SfM

Table 5. Evaluation Reconstruction Results for SfM Systems.

	#Size	#Registered		#Time[minutes]		#Avg. Reproj. Error [px]	
		COLMAP	SfM*	COLMAP	SfM*	COLMAP	SfM*
Alamo	2,915	803	841	41.55	4.30	1.01	1.04
Ellis Island	2,587	385	401	10.89	1.61	1.13	1.18
Gendarmenmarkt	1,463	956	961	17.70	2.76	1.07	1.07
Madrid Metropolis	1,344	405	404	9.17	1.00	1.25	1.25
Montreal Notre Dame	2,298	558	559	25.59	2.42	1.16	1.21
NYC Library	2,550	527	527	17.08	1.36	1.01	1.01
Piazza del Popolo	2,251	883	907	12.48	1.95	1.04	1.22
Piccadilly	7,351	2956	2893	64.00	11.77	1.09	1.17
Roman Forum	2,364	1416	1436	38.12	3.66	1.06	1.14
Tower of London	1,576	586	611	16.14	1.52	0.97	1.01
Trafalgar	15,685	6968	7392	179.46	30.47	1.07	1.18
Union Square	5,961	767	768	13.31	1.36	0.99	1.12
Vienna Cathedral	6,288	1115	1119	53.17	5.93	1.10	1.08
Yorkminster	3,368	984	981	38.94	2.46	1.05	1.09

Table 6. Translation RMSE and Time Comparison on KITTI Dataset.

	#RMSE[m]		#LBA Time[ms]		#GBA Time[s]	
	ORB-SLAM2	SLAM*	ORB-SLAM2	SLAM*	ORB-SLAM2	SLAM*
Seq.00	6.63	6.27	45	22	22.77	12.49
Seq.02	23.02	24.58	32	17	19.97	9.62
Seq.03	1.64	1.38	79	37	0.0	0.0
Seq.04	0.98	0.78	42	23	0.0	0.0
Seq.05	6.51	4.63	51	25	15.19	6.75
Seq.06	15.72	12.92	39	18	3.66	1.85
Seq.07	2.21	1.50	37	23	3.09	1.22
Seq.08	17.46	24.9	34	17	0.0	0.0
Seq.9	7.15	7.23	27	16	4.12	1.78
Seq.10	7.38	8.01	29	16	0.0	0.0

In order to further prove the robustness and efficiency of the proposed solver, we also carry out experiments in real SLAM and SfM systems. We replace the optimizers of ORB-SLAM2 and COLMAP with the proposed solver. The systems with the proposed optimizer are called SLAM* and SfM*. Except for the replacement of the optimizer, other modules and parameters are completely consistent. For the SLAM systems, we test ten sequences of the well-known KITTI dataset. Due to the tracking failure of ORB-SLAM2, the sequence Seq.01 is skipped. As shown in Table 6, we evaluate the **RMSE** of trajectory, the average local BA time (**LBA Time**), and the sum of global optimal time (**GBA Time**). The global optimization in ORB-SLAM2 is used in the loop closing stage, so the timing may be zero in sequences without loops. *SLAM*$*$ has comparable accuracy performance and takes only half the optimization time. The optimization time of local bundle adjustment is about 20ms, which can be carried out at 50 Hz theoretically. Since most SLAM systems only perform local BA on keyframes, this speed can fully meet the requirements of real-time applications.

For the SfM systems, we evaluate the 1DSfM dataset which has 14 diverse Internet photo collections. The intrinsic parameters of cameras are provided by optimized maps and fixed in this experiment. COLMAP uses Ceres in local and global BA, and we replace Ceres solver with CoLi-BA to achieve SfM*. In order to avoid the optimization time too long, PCG will be used for global optimization when the number of cameras is greater than 100 in both COLMAP and SfM*. As shown in Table 5, **Size** is the number of total images, **Registered** is the number of registered frames, **Time** denotes the total reconstruction time, **Avg. Reproj Error** is the average reprojection error. Compared with COLMAP, SfM* requires only $10 \sim 20\%$ reconstruction time to obtain comparable number of registered frames and reconstruction accuracy.

Using the optimized SLAM and SfM, we demonstrate an AR example, as shown in Fig. 16. The scene images come from the RGBD dataset TUM [36], and the object pictures are taken by ourselves with a mobile phone. Specifically, we obtain the trajectory of the scene images through SLAM* and reconstruct the sparse point cloud model of the object and the camera poses through SfM*. With this information, we superimpose the virtual object rendered on the scene image by a real-time NeRF implementation [25]. More AR results are shown in Fig. 17.

5 Discussion

With very fast solution speed, CoLi-BA also maintains good accuracy. To improve the speed losslessly, our method has some requirements for the problem, but it is still applicable to most scenarios. First, when a complex covariance matrix is used to weight the reprojection error, the rotation cannot be eliminated, and the compact representation becomes invalid. However, in most cases, the covariance matrix is a simple scalar matrix, so it would not affect the generalization of our method. Second, current implementation does not support the optimization of camera intrinsic, as it cannot be incorporated with the compact representation. However, in most real-time localization and reconstruction tasks , camera intrinsic parameters is readily available with a pre-stage calibration. To the best of our knowledge, the proposed method is widely applicable to camera trajectory recovery in AR applications.

Fig. 16. The AR demo putting the toy on the desk. The top-left is the original scene image, and the top-right is a picture of the object. The NeRF model of the object is rendered in the scene by the camera poses, and shown in the lower left. We highlighted the virtual object on the desk with a red box, and zoom-in it on the lower right.

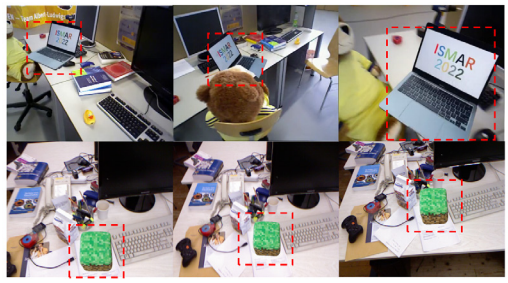

Fig. 17. More AR results. Virtual objects are highlighted with red boxes.

6 Conclusions

In this paper, we propose a compact linearization method to accelerate BA. By fully exploiting the geometry property, the proposed compact representation significantly reduces calculation, memory storage, and cache misses. Moreover, we also propose a rearrangement strategy to further improve the cache efficiency. Experiments show that the proposed method has a substantial speed advantage compared with the state-of-the-arts, and can be directly applied to the existing SLAM/SfM systems to improve efficiency. By superimposing incremental strategies, it is promising to further accelerate the optimization speed to meet low power consumption requirements and real-time requirements of the framework in future work.

Acknowledgments

This work was partially supported by the National Key Research and Development Program of China under Grant 2020YFF0304300, the National Natural Science Foundation of China (No. 61932003), and ZJU-SenseTime Joint Lab of 3D Vision.

REFERENCES

[1] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski. Building Rome in a Day. *Communications of the ACM*, 54(10):105–112, 2011.

[2] S. Agarwal, K. Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[3] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle Adjustment in the Large. In *European Conference on Computer Vision*, pp. 29–42. Springer, 2010.

[4] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[5] M. Byröd and K. Åström. Conjugate Gradient Bundle Adjustment. In *European Conference on Computer Vision*, pp. 114–127. Springer, 2010.

[6] Q. Cai, L. Zhang, Y. Wu, W. Yu, and D. Hu. A Pose-only Solution to Visual Reconstruction and Navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. doi: 10.1109/TPAMI.2021.3139681

[7] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[8] C. Campos, J. M. Montiel, and J. D. Tardós. Inertial-Only Optimization for Visual-Inertial Initialization. In *IEEE International Conference on Robotics and Automation*, pp. 51–57. IEEE, 2020.

[9] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Transactions on Mathematical Software*, 35(3):1–14, 2008.

[10] S. C. Eisenstat. Efficient Implementation of a Class of Preconditioned Conjugate Gradient Methods. *SIAM Journal on Scientific and Statistical Computing*, 2(1):1–4, 1981.

[11] R. Elvira, J. D. Tardós, and J. M. Montiel. ORBSLAM-Atlas: A Robust and Accurate Multi-Map System. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 6253–6259. IEEE, 2019.

[12] A. Eriksson, J. Bastian, T.-J. Chin, and M. Isaksson. A Consensus-Based Framework for Distributed Bundle Adjustment. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1754–1762, 2016.

[13] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision Meets Robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[14] G. Grisetti, R. Kümmerle, H. Strasdat, and K. Konolige. g2o: A General Framework for Graph Optimization. In *IEEE International Conference on Robotics and Automation*, pp. 9–13, 2011.

[15] M. Gu and S. C. Eisenstat. Efficient Algorithms for Computing a Strong Rank-Revealing QR Factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.

[16] H. Guan and W. A. Smith. Structure-from-Motion in Spherical Video Using the Von Mises-fisher Distribution. *IEEE Transactions on Image Processing*, 26(2):711–723, 2016.

[17] V. Ila, L. Polok, M. Solony, and K. Istenic. Fast Incremental Bundle Adjustment with Covariance Recovery. In *International Conference on 3D Vision*, pp. 175–184. IEEE, 2017.

[18] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert. iSAM2: Incremental Smoothing and Mapping using the Bayes Tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[19] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental Smoothing and Mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[20] P. Lindenberger, P.-E. Sarlin, V. Larsson, and M. Pollefeys. Pixel-Perfect Structure-from-Motion with Featuremetric Refinement. In *IEEE International Conference on Computer Vision*, 2021.

[21] H. Liu, M. Chen, G. Zhang, H. Bao, and Y. Bao. ICE-BA: Incremental, Consistent and Efficient Bundle Adjustment for Visual-Inertial SLAM. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1974–1982, 2018.

[22] M. I. Lourakis and A. A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Transactions on Mathematical Software*, 36(1):1–30, 2009.

[23] J. J. Moré. The Levenberg-Marquardt Algorithm: Implementation and Theory. In *Numerical analysis*, pp. 105–116. Springer, 1978.

[24] P. Moulon, P. Monasse, R. Perrot, and R. Marlet. OpenMVG: Open Multiple View Geometry. In *International Workshop on Reproducible Research in Pattern Recognition*, pp. 60–74. Springer, 2016.

[25] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):102:1–102:15, July 2022. doi: 10.1145/3528223.3530127

[26] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[27] K. Ni, D. Steedly, and F. Dellaert. Out-of-Core Bundle Adjustment for Large-Scale 3D Reconstruction. In *IEEE 11th International Conference on Computer Vision*, pp. 1–8. IEEE, 2007.

[28] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

[29] A. Pagani and D. Stricker. Structure from Motion Using Full Spherical Panoramic Cameras. In *IEEE International Conference on Computer Vision Workshops*, pp. 375–382. IEEE, 2011.

[30] L. Polok, V. Ila, and P. Smrz. Fast Sparse Matrix Multiplication on GPU. In *SpringSim (HPS)*, pp. 33–40, 2015.

[31] L. Polok, M. Solony, V. Ila, P. Smrz, and P. Zemcik. Efficient Implementation for Block Matrix Operations for Nonlinear Least Squares Problems in Robotic Applications. In *IEEE International Conference on Robotics and Automation*, pp. 2263–2269. IEEE, 2013.

[32] M. J. Powell. A Hybrid Method for Nonlinear Equations. *Numerical Methods for Nonlinear Algebraic Equations*, 1970.

[33] T. Qin, P. Li, and S. Shen. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[34] D. M. Rosen, M. Kaess, and J. J. Leonard. An Incremental Trust-region Method for Robust Online Sparse Least-Squares Estimation. In *IEEE International Conference on Robotics and Automation*, pp. 1262–1269. IEEE, 2012.

[35] J. L. Schonberger and J.-M. Frahm. Structure-from-Motion Revisited. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4104–4113, 2016.

[36] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *IEEE/RSJ International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[37] T. Tanaka, Y. Sasagawa, and T. Okatani. Learning To Bundle-Adjust: A Graph Network Approach to Faster Optimization of Bundle Adjustment for Vehicular SLAM. In *IEEE/CVF International Conference on Computer Vision*, pp. 6250–6259, 2021.

[38] C. Tang and P. Tan. BA-Net: Dense Bundle Adjustment Network. In *7th International Conference on Learning Representations*, 2019.

[39] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle Adjustment—A Modern Synthesis. In *International Workshop on Vision Algorithms*, pp. 298–372. Springer, 1999.

[40] A. Tyagi, Y. Liang, S. Wang, and D. Bai. DVIO: Depth-Aided Visual Inertial Odometry for RGBD Sensors. In *IEEE International Symposium on Mixed and Augmented Reality*, pp. 193–201. IEEE, 2021.

[41] J. Ventura, C. Arth, G. Reitmayr, and D. Schmalstieg. Global Localization from Monocular SLAM on a Mobile Phone. *IEEE Transactions on Visualization and Computer Graphics*, 20(4):531–539, 2014.

[42] K. Wilson and N. Snavely. Robust Global Translations with 1DSfM. In *European Conference on Computer Vision*, pp. 61–75, 2014.

[43] C. Wu. Towards Linear-Time Incremental Structure from Motion. In *International Conference on 3D Vision*, pp. 127–134. IEEE, 2013.

[44] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Multicore Bundle Adjustment. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3057–3064. IEEE, 2011.

[45] X. Yang, L. Zhou, H. Jiang, Z. Tang, Y. Wang, H. Bao, and G. Zhang. Mobile3DRecon: Real-time Monocular 3D Reconstruction on a Mobile Phone. *IEEE Transactions on Visualization and Computer Graphics*, 26(12):3446–3456, 2020.

[46] F. Zhang. *The Schur Complement and Its Applications*, vol. 4. Springer Science & Business Media, 2006.

[47] R. Zhang, S. Zhu, T. Fang, and L. Quan. Distributed Very Large Scale Bundle Adjustment by Global Camera Consensus. In *IEEE International Conference on Computer Vision*, pp. 29–38, 2017.

[48] L. Zhou, D. Koppel, H. Ju, F. Steinbruecker, and M. Kaess. An Efficient Planar Bundle Adjustment Algorithm. In *IEEE International Symposium on Mixed and Augmented Reality*, pp. 136–145. IEEE, 2020.