OpenOF
Framework for Sparse Non-linear Least Squares Optimization on a GPU

Cornelius Wefelscheid and Olaf Hellwich
Computer Vision and Remote Sensing, Berlin University of Technology,
Sekr. MAR 6-5, Marchstrasse 23, D-10587, Berlin, Germany

Keywords: Least Squares Optimization, Bundle Adjustment, Levenberg Marquardt, GPU, Open Source.

Abstract: In the area of computer vision and robotics non-linear optimization methods have become an important tool. For instance, all structure from motion approaches apply optimizations such as bundle adjustment (BA). Most often, the structure of the problem is sparse regarding the functional relations of parameters and measurements. The sparsity of the system has to be modeled within the optimization in order to achieve good performance. With *OpenOF*, a framework is presented, which enables developers to design sparse optimizations regarding parameters and measurements and utilize the parallel power of a GPU. We demonstrate the universality of our framework using BA as example. The performance and accuracy is compared to published implementations for synthetic and real world data.

1 INTRODUCTION

Non-linear least squares optimization is used in many research fields where measurements with an underlying model are present. Most of the problems consist of hundreds of measurements and are represented by hundreds of parameters that need to be estimated. Minimizing the L2-norm between the actual measurement and the estimated value is established by e.g. solving a least squares system. Usually, as the underlying model is not linear, an iterative method for solving non-linear least squares needs to be applied. Such methods linearize the cost function in each iteration. The Levenberg-Marquardt (LM) algorithm has more or less become a standard. It combines the Gauss-Newton algorithm with the gradient descent approach. In contrast to other approaches, LM guarantees convergence. The computationally most intensive part within LM is solving $\mathbf{Ax} = \mathbf{b}$ in each iteration. Finding the solution of this normal equation for a dense matrix has a complexity of $O(n^3)$. This is not effective when dealing with 100k - 1M parameters on a standard computer. As most of the entries in $\mathbf{A}$ are zero, a sparse matrix representation is feasible. Different parametrization for sparse matrices have been used in the past. Most commonly a coordinate list (COO) is used, which stores for each entry row, column and value. A more compact storage is represented by the compressed sparse column (CSC) format. For each entry, the row and the value are stored in an array. Additionally, the starting index of each column is saved separately, requiring less space than COO. The reduction of space between COO and CSC is neglectable, while the implementation of efficient algorithms for matrix-vector multiplications is important. Since the operations on matrices can be implemented in a highly parallel manner, we believe that multiprocessors, such as GPUs, are most suitable for this task. The operations within LM are as well highly parallel, as each cost function can be evaluated independently. To our knowledge no library has been developed which can solve general sparse non-linear least squares optimizations on a GPU. Two libraries address this problem in general, *sparseLM* (Lourakis, 2010) and *g2o* (Kummerle et al., 2011), both operating on a CPU. For solving the normal equation, they implemented interfaces to various external sparse math libraries, which provide different algorithms for direct solving the normal equation. In most cases a Cholesky decomposition $\mathbf{A} = \mathbf{LDL}^T$ is applied. In contrast to *sparseLM*, our approach works on a GPU (presently only NVIDIA graphics cards are supported) and uses the high parallel power of the graphics card to solve the normal equation with a conjugate gradient (CG) approach, which can also be chosen as a solver in *g2o*. The potential of parallelizing CG is the main advantage over a Cholesky decomposition. Our framework is built upon three

Wefelscheid C. and Hellwich O..
OpenOF - Framework for Sparse Non-linear Least Squares Optimization on a GPU.
DOI: 10.5220/0004282702600267
In *Proceedings of the International Conference on Computer Vision Theory and Applications* (VISAPP-2013), pages 260-267
ISBN: 978-989-8565-48-8

major libraries, *Thrust*, *CUSP* and *SymPy*. *Thrust* enhances the productivity of developers by providing high level interfaces for massive parallel operations on multiprocessors. *Thrust* is integrated in the current CUDA version 4.2 and supported by NVIDIA. The framework presented in this paper can directly take advantage of any improvements. The library *CUSP* is based on *Thrust* and provides developers with routines for sparse linear algebra. The library *SymPy* is an open source library for symbolic mathematics in Python. The aim of our work is to give scientists a new tool to test new models in less amount of time. Using *SymPy* in our framework enables us to define least squares models within a high level scripting language. Starting from this, our framework automatically generates C++ code which is compiled, generating the *OpenOF* optimization library, which can either be called from C++ programmes for efficient use, or from Python for rapid prototyping. *OpenOF* will be published as open source library under the GNU GPL v3 license.

2 RELATED WORK

Recently, non-linear optimization has gathered a lot of attention. It has been successfully applied in different areas, e.g. Simultaneous Localization and Mapping (SLAM) or BA. We will use BA for evaluating our framework, as it is a perfect example for a rather complex but sparse system. Several libraries can be used to solve BA. The *SBA* library (Lourakis and Argyros, 2009) which is used by several research groups takes advantage of the special structure of the Hessian matrix to apply the Schur complement for solving the linear system. Nevertheless it has several drawbacks. Integrating additional parameters which remain identical for all measurements (e.g. camera calibration) is not possible, as the structure would change such that the Schur complement could not be applied anymore. With *OpenOF*, we provide a solution for those problems.

sparseLM (Lourakis, 2010) provides a software package which is supposed to be general enough to handle the same kind of problems as *OpenOF* does. However, integrating new models within *sparseLM* is time consuming. The performance is slow for problems with many parameters, as no multithreading is integrated.

More recently another framework was developed for similar purposes in graph optimization (Kummerle et al., 2011). In *g2o*, the Jacobian is evaluated by numerical differentiation which is time consuming and also degrades the convergence rate. Developers can choose between direct (Davis, 2006) and iterative solver such as CG. Kummerle et al. (2011) claims that CG is most often slower than a direct solver. In our experience direct solvers are more precise. However utilizing the parallelism of multiprocessors, CG can be significantly faster. Applying a preconditioner to CG has a large impact on the speed of convergence. *g2o* uses a block Jacobi preconditioner. For simplicity we use a diagonal preconditioner which has been applied successfully for various tasks. A big advantage of the diagonal preconditioner is, that the least squares CG is computed directly from the Jacobian without explicitly calculating the Hessian matrix. Several other approaches (Kaess et al., 2011), which address only a subset of problems, have been presented previously for least squares optimization. But so far no library provides the speed, the ease of use and the generality to become a milestone within the communities of computer vision or robotics.

3 LEVENBERG-MARQUARDT

LM is regarded as standard for solving non-linear least squares optimization. It is an iterative approach which was first developed by Levenberg in 1944 and rediscovered by Marquardt in 1963. The algorithm determines the local minimum of the sum of least squares of a multivariant function. An overview of the family of non-linear least squares optimization techniques can be found in (Madsen et al., 2004). LM combines a Gauss-Newton method with a steepest descent approach. Given a parameter vector $\mathbf{x}$ we want to find a vector $\mathbf{x}_{min}$ that minimizes the function $g(\mathbf{x})$ which is defined as the sum of squares of the vector function $f(\mathbf{x})$.

$$\mathbf{x}_{min} = \arg\min_{\mathbf{x}} g(\mathbf{x}) = \arg\min_{\mathbf{x}} \frac{1}{2} \sum_{i=0}^{m} f_i(\mathbf{x})^2 \quad (1)$$

As the function is usually not convex, a suitable initial solution $\mathbf{x}_0$ needs to be provided. Otherwise, LM will not converge to the global optimum, but get stuck in a local minimum, which might be far away from the optimal solution. In each iteration i, $g(\mathbf{x})$ is approximated by a second order Tailor series expansion.

$$\tilde{g}(\mathbf{x}) = g(\mathbf{x}_i) + g(\mathbf{x}_i)'(\mathbf{x}-\mathbf{x}_i) + \frac{1}{2} g(\mathbf{x}_i)''(\mathbf{x}-\mathbf{x}_i)^2 \quad (2)$$

To find the minimum the first derivative of Equation 2 is set to zero, resulting in

$$\frac{\partial \tilde{g}(\mathbf{x})}{\partial \mathbf{x}} = g(\mathbf{x}_i)' + g(\mathbf{x}_i)''(\mathbf{x}-\mathbf{x}_i) = \mathbf{0}\,. \quad (3)$$

The first order derivative at the point $\mathbf{x}_i$ can be replaced by the transposed Jacobian $\mathbf{J}^T$ multiplied with

the residual. The second order derivative is approximated by the Hessian matrix given as $\mathbf{J}^T\mathbf{J}$. The parameter update $\mathbf{h} = \mathbf{x} - \mathbf{x}_i$ in each iteration is acquired by solving the normal equation.

$$\mathbf{J}^T\mathbf{J}\mathbf{h} = -\mathbf{J}^T f(\mathbf{x}_i) \tag{4}$$

To include the gradient descent approach, Equation 4 is extended by a damping factor μ. For $\mu \to 0$ LM applies the Gauss-Newton method and for $\mu \to \infty$ a gradient descent step is executed.

$$(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\mathbf{h} = -\mathbf{J}^T f(\mathbf{x}_i) \tag{5}$$

If the update step does not result in a smaller error the damping factor is increased pushing the parameters further towards the steepest descent. Else, the update is performed and the damping factor is reduced. In cases with a known covariance of the measurements, Equation 5 is extended to

$$(\mathbf{J}^T\Sigma^{-1}\mathbf{J} + \mu\mathbf{I})\mathbf{h} = -\mathbf{J}^T\Sigma^{-1} f(\mathbf{x}_i)\,. \tag{6}$$

Instead of the Euclidean norm the Mahalanobis distance is minimized. Solving the normal equation is the most demanding part of the algorithm. If the normal matrix has a certain structure, approaches such as the Schur complement can be applied. Generally, the normal equation can be solved with a Cholesky decomposition. Another alternative is an iterative approach such as CG, which is incorporated in *OpenOF*. In the next section, the CG approach will be described in more detail.

4 CONJUGATE GRADIENT

The CG method is a numerical algorithm for solving a linear equation system $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $\mathbf{A}$ being symmetric positive definite and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. As CG is an iterative algorithm, there exists a residual vector $\mathbf{r}_k$ in each step k defined as:

$$\mathbf{r}_k = \mathbf{A}\mathbf{x}_k - \mathbf{b}\,. \tag{7}$$

CG belongs to the Krylov subspace methods. Krylov subspace is described by

$$\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) = span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\}\,. \tag{8}$$

It is the basis for many iterative algorithms. There is proof that those methods terminate in at most n steps. CG is a highly economical method which does not require much memory or time demanding matrix-matrix multiplication. While applying CG, a set of conjugate vectors $(\mathbf{p}_0, \mathbf{p}_1, .., \mathbf{p}_{k-1}, \mathbf{p}_k)$ is computed belonging to the Krylov subspace. In each step the algorithm computes a vector $\mathbf{p}_k$ being conjugate to all previous vectors. A great advantage is that only the previous vector $\mathbf{p}_{k-1}$ is needed. Thus, the other vectors can be omitted, thereby saving memory. The algorithm is based on the principle that $\mathbf{p}_k$ minimizes the residual over one spanning direction in every iteration. CG is initialized with $\mathbf{r}_0 = \mathbf{A}\mathbf{x}_0 - \mathbf{b}, \mathbf{p}_0 = -\mathbf{r}_0, k = 0$. In each iteration a step length α is computed, which minimizes the residual over the current spanning vector $\mathbf{p}_k$.

$$\alpha_k = -\frac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{p}_k^T\mathbf{A}\mathbf{p}_k} \tag{9}$$

The update of the current approximate solution is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k \tag{10}$$

which results in a new residual

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k\mathbf{A}\mathbf{p}_k\,. \tag{11}$$

The new conjugate direction is computed from Equations 12 and 13.

$$\beta_{k+1} = -\frac{\mathbf{r}_{k+1}^T\mathbf{r}_{k+1}}{\mathbf{r}_k^T\mathbf{r}_k} \tag{12}$$

$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k \tag{13}$$

Finally k is increased. This procedure is repeated as long as $||\mathbf{r}_k|| > \varepsilon$ and $k < k_{max}$. In most cases, only a few iterations are necessary as the CG method is used within the iterative LM algorithm. For prove of convergence and full derivation we refer to (Nocedal and Wright, 1999).

For practical considerations the explicit computation of the matrix $\mathbf{A} = \mathbf{J}^T\mathbf{J}$ can be omitted. Concatenating matrix-vector multiplications for $\mathbf{J}^T$ and $\mathbf{J}$ is faster and the numerical precision is higher. Such an approach is known as least squares CG.

5 FRAMEWORK

OpenOF aims at making development cycles faster. It saves the time of the developer searching for errors by automatic code generation, but not at the expense of computational efficiency. As we use fast high level libraries on a GPU, the generated executable gains performance. The design of the underlying model assumes different types of objects containing groups of parameters.These objects are linked by measurements. The objects can be combined in arbitrary ways (see Fig. 1). Usually, the computation of the Jacobi matrix is either done by hand, or with the help of a symbolic toolbox (*SymPy*, Matlab). Both cases are error prone. The mistakes easily occure when either transferring the code into an optimization library

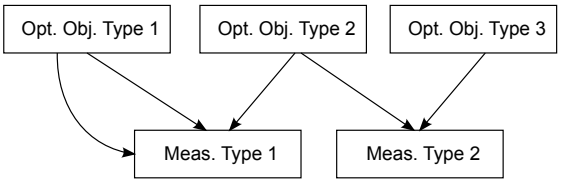


Figure 1: Optimization objects interact with different measurement types.

such as *sparseLM* (Lourakis, 2010) or from generating the derivatives by hand. This step needs to be done each time small changes are made in the underlying model. As the cost function usually can be described in various ways, exploring which modeling of e.g. relations between unknowns and measurements works best is time consuming. Additionally the convergence rate highly depends on the chosen model e.g. a convex cost function is most preferable. In *OpenOF* the model can be adjusted without further low level programming steps, as the necessary code is generated automatically. The main optimization is designed in a modular way, enabling developers to easily modify the optimization algorithm or add new algorithms while still having the advantage of the high performance of a GPU.

An optimization task often comprises thousands of parameters. Usually several entities have the same meaning, e.g. (x,y) refer to the coordinate of a point. Furthermore, these parameters can be grouped to objects. Within *OpenOF* all parameters are assigned to an object. When designing optimization objects, we distinguish between two different kinds of parameters:

- Variable parameters
- Fixed parameters

The variable parameters will be adjusted within the optimization. The fixed parameters remain constant but are necessary to evaluate the cost function. Most often we end up with a small number of different types of objects. These objects are linked by so called measurement types. A measurement represents a function on which the least squares optimization is performed. The user defines the function representing relations between the different objects. Usually, not all types of objects are necessary to evaluate a certain measurement (see Figure 1). Sometimes we have a direct but noisy observation $\mathbf{X}_{obs}$, e.g. the position given from a GPS sensor. In this case we define a measurement $f(\mathbf{X}_{est},\mathbf{X}_{obs}) = \mathbf{X}_{est} - \mathbf{X}_{obs}$. It is clear that the observation $\mathbf{X}_{obs}$ is not allowed to change. This function can be modeled in two ways in *OpenOF*. For the first option two different object types are defined, where the first object type has only variable parameters and the second has only fixed parameters. In this case there are two almost identical object types which only differ regarding which parameters are allowed to be changed. It would be more intuitive to have only one object type which defines a position with variable parameters. To still be able to fix the parameters of the observation $\mathbf{X}_{obs}$, it is possible in *OpenOF* to also fix complete object types within a measurement function. Getting back to the example, one would define for the function f, $\mathbf{X}_{est}$ to be variable and $\mathbf{X}_{obs}$ to be fixed. This enables researchers to incorporate constraints such that certain parameters stay close to the initial values without the need to generate a new type of object with identical parameters. Being able to define an object as constant in one measurement and variable in another measurement gives much flexibility for defining an optimization.

Each measurement forms an equation, which is transformed to a cost function. This cost function is designed with *SymPy*. The Jacobian of the cost function is computed automatically and transformed to C++ code. Different measurement types as well as object types combined, result in a graph similar to the one presented in (Kummerle et al., 2011) (see Figure 1).

By now only the structure of the optimization has been defined. Next, the optimization needs to be filled with actual data. Therefore, a list containing the data of each type of object is created. A measurement is defined by the type of measurement as well as the index pointing to the corresponding data within the data list.

6 EXAMPLES

We present two examples, first the classic BA and second an alternative BA parametrization inspired by (Civera et al., 2008) using inverse distances. BA contains four kinds of optimization objects. The first object is a 3D point parametrized by a name and its coordinates (`pt3:[X,Y,Z]`). The second object type defines the external camera parameters (`cam:[CX,CY,CZ,q1,q2,q3,q4]`) with the center point and a quaternion representing the rotation of the camera. Since the same camera is repeatedly used within one reconstruction, an extra object for the internal camera parameters (`cam_in:[fx,fy,u0,v0,s,k1,k2,k3]`) is defined. This enables us to constrain the optimization. Several cameras then share the same internal calibration (focal length, principle point, skew and distortion parameters). At last a constant object which contains the pixel measurements (`pt2:[x,y]`) is specified. These parameters remain unchanged within the optimization

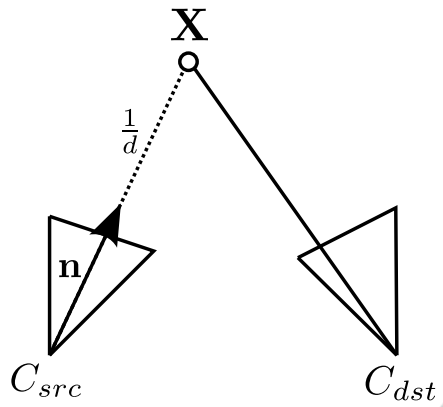


Figure 2: Inverse point parametrization with source camera and destination camera.

but are generated as the same type of structure. Next, different measurement types are described. We create a measurement (`quat:[cam],[0]`) where the object is a quaternion enforced to have unit length. The indices written in the second pair of square brackets specify the indices of the objects to be optimized. The next measurement represents the error of the reprojection in the image with fixed intrinsic parameters (`projMetric:[pt2,pt3,cam,cam_in],[1,2]`). Only the objects `pt3` and `cam` are part of the optimization. Adjusting the measurement in a way that the intrinsic parameters are optimized as well, the last list is extended by the fourth object (`projRadial:[pt2,pt3,cam,cam_in],[1,2,3]`). The user provides the system with information, closely related to the bracket notation above. Using *OpenOF*, it is easy and fast to test different kinds of parametrizations.

We compare a new BA parametrization with the one previously described. This new method is called inverse bundle adjustment (IBA) due to the inverse depth parametrization (Civera et al., 2008). A 3D point is no longer described by three parameters, but with a source camera, a unit vector and an inverse depth. So the point is linked to the camera of first appearance (see Fig. 2). For the parametrization we end up with the following list:

- `pt3inv:[d][nx,ny,nz]`
- `cam:[CX,CY,CZ,q1,q2,q3,q4]`
- `cam_in:[fx,fy,u0,v0,s,k1,k2,k3]`
- `pt2:[x,y]`

In this example we distinguish between fix and optimizable parameters. For `pt3inv`, `d` is part of the optimization, but `nx,ny,nz` remain constant. The coordinates (X,Y,Z) of a 3D point are computed according to

$$\tilde{\mathbf{X}} = \begin{pmatrix} 0 \\ X \\ Y \\ Z \end{pmatrix} = \frac{1}{d}\mathbf{q}^{-1}\tilde{\mathbf{n}}\mathbf{q} + \tilde{\mathbf{C}} \tag{14}$$

with $\mathbf{q}$ being the quaternion corresponding to the source camera and $\mathbf{C}$ is the camera center. The tilde above the variable denotes a quaternion representation of a 3D vector. The described parametrization has several advantages. The degrees of freedom of the complete optimization are reduced by $2n$ with n representing the number of points. The part of the state vector which holds the parameters of the cameras has the same size, since a camera can be a source and a destination camera at the same time. At the same time the amount of measurements is reduced by n. The measurement in the source camera is neglected. This might be a disadvantage as more measurements make the optimization more stable. Regarding the optimization, more important is the ratio between parameters and measurements. Assuming that each point is seen by each camera and $n > 7, m > 3$, with m being the number of cameras, the ratio of parameters to measurements is always smaller for IBA than for BA.

$$\frac{7m+n}{2n(m-1)} < \frac{7m+3n}{2nm} \tag{15}$$

For Equation 15 we assumed that the intrinsic camera parameters are fixed. We investigated a depth parametrization as well but found the inverse parametrization to be superior due to a more linear behavior.

7 RESULTS

For evaluation we compare IBA to BA both computed with *OpenOF*. Both methods are evaluated against simulated data as well as a real world dataset. The performance is compared against two open source BA implementations, Simple Sparse Bundle Adjustment (*SSBA*) (Zach, 2012) and *PBA* (Wu et al., 2011). The latter stands for Parallel Bundle Adjustment implemented for the GPU and CPU. In contrast to *SSBA*, *PBA* uses CG as presented in our approach. We show that the performance of the specially optimized *PBA* is comparable to *OpenOF*. All evaluations are performed on an Intel i7 with 3.07 GHz, 24 GB RAM and an NVIDIA GTX 570 graphics card with 2.5 GB memory. For both versions of *PBA* as well as for *OpenOF* we use single precision. In contrast *SSBA* runs with double precision. For evaluation we use synthetic data shown in Fig. 3. The 3D points are randomly generated and placed on the walls of a cube. The points are only projected into cameras for

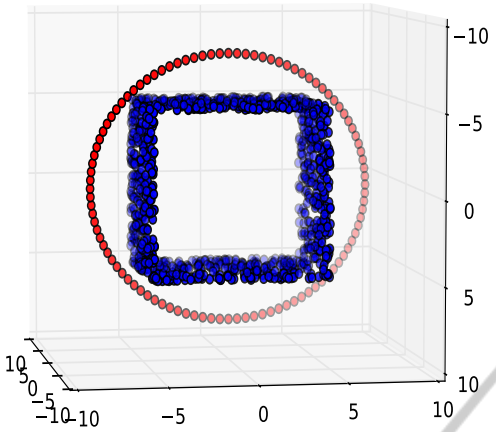
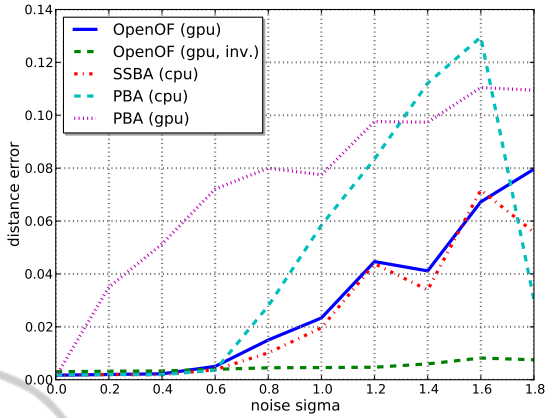
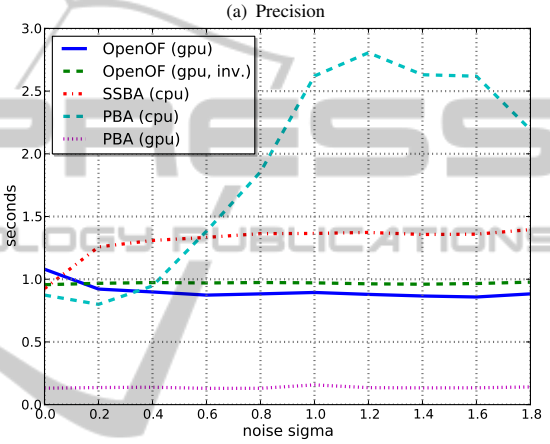


Figure 3: Setup of points (blue) and cams (red) for the simulated data.

which the points are directly visible in reality, assuming the cube is not transparent. The cube has a size of 10x10x10. The cameras are positioned on a circle with a radius of 8, facing the center of the cube. The points are projected into a virtual camera with an image size of 640x480 and a focal length of 1000 measured in pixels. The principle point is assumed to lie in the center of the image. To simulate structure from motion approaches, where the initial camera position is rather a rough estimate, we apply noise to the camera position. Normally distributed noise of 0.5 pixels is applied to the measurement. Furthermore, we triangulate new points from the virtual projections instead of using the original 3D point, resulting in a more realistic setup. With known positions of the cameras, we investigate the convergence of the five BA implementations stated earlier by continuously increasing the noise of the camera position. Within this simulation we use 100 cameras and 1000 points. The overall estimate of each method is transformed to the original cameras with a Helmert transformation. The mean distance error between the estimated and true camera position is shown in Fig. 4(a) for different amounts of noise. For each noise level the mean of several iterations is plotted. Within our test the GPU implementation of *PBA* did not converge in most cases, which is shown in Fig. 4(a). The reason for the failure could not be identified but we can exclude a wrong usage as there is only a single line of code which toggles *PBA* from a CPU version to a GPU one. Every other method shows similar precision for $\sigma < 0.6$. For $\sigma > 0.6$, IBA shows a better convergence than the other methods due to less degrees of freedom. Especially the CPU version of *PBA* most often did not converge for noise $\sigma > 1.2$. The convergence problem of *PBA* causes also an increasing time (Fig. 4(b)), while the other approaches stay con-



(a) Precision

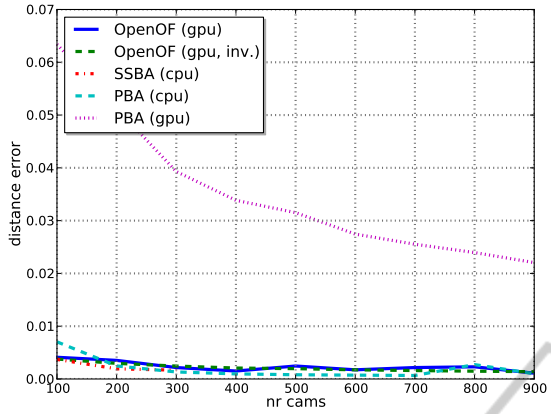


(b) Performance

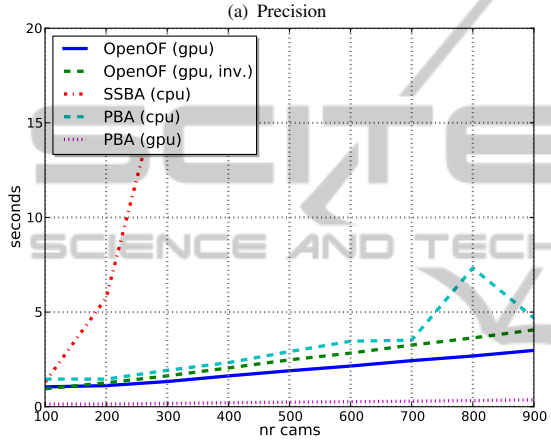
Figure 4: Applying normal distributed noise on the camera positions (100 cameras).

stant in time. To evaluate the speed of the algorithms, we run each method with a constant amount of noise while concomitantly increasing the number of cameras. This reduces the distance between two cameras and increases the number of projections of one 3D point. As expected, *SSBA*, using a direct solver implemented on a CPU, shows the lowest performance. Regarding this example, the CG approach (*OpenOF* and *PBA*) operates much faster than a direct solver (see Figs. 5(a) and 5(b)). The GPU version of *PBA* cannot be taken into account because it did not converge.

The real world data evaluation is performed on a dataset with 390 images taken with a Canon EOS 7D camera (see Fig. 6(b)). The pictures were taken in the American Museum of Natural History within the scope of a project aiming at the reconstruction of mammoths. Fig. 6(a) shows the final reconstruction computed with PMVS2 (Furukawa and Ponce, 2010). The path computed with *OpenOF* is illustrated in Fig. 6(c). We compare the convergence rates between *SSBA* and *OpenOF*, respectively. *PBA* is not

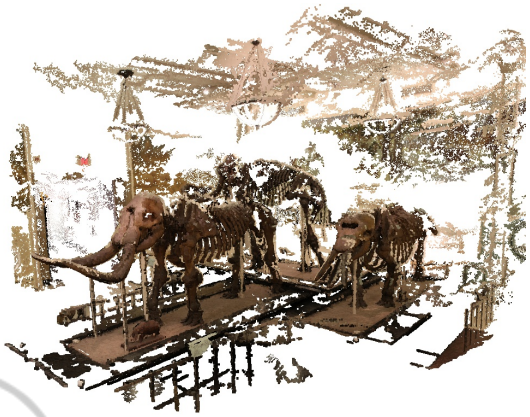


(a) Precision



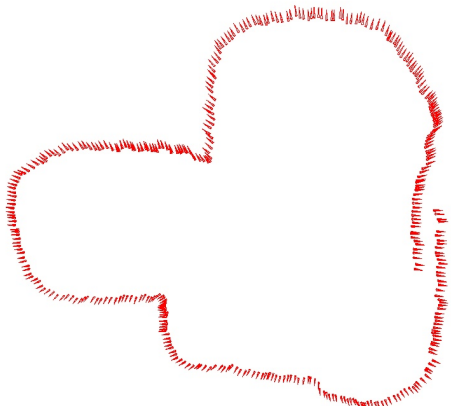
(b) Performance

Figure 5: Changing the number of cameras with constant amount of noise on the camera position ($\sigma = 0.5$).



(a) Quasi-dense point cloud

(b) Sample image



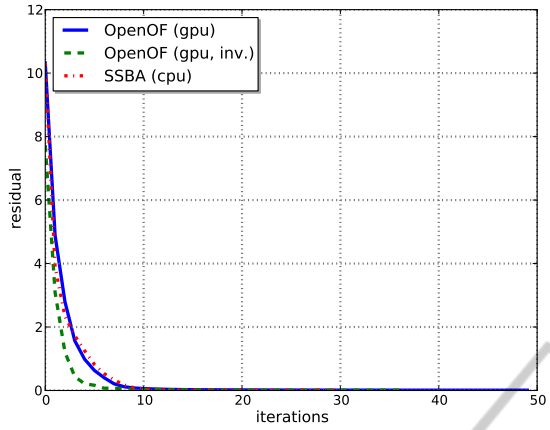
(c) Camera path

Figure 6: Three mammoths acquired at the American Museum of Natural History.

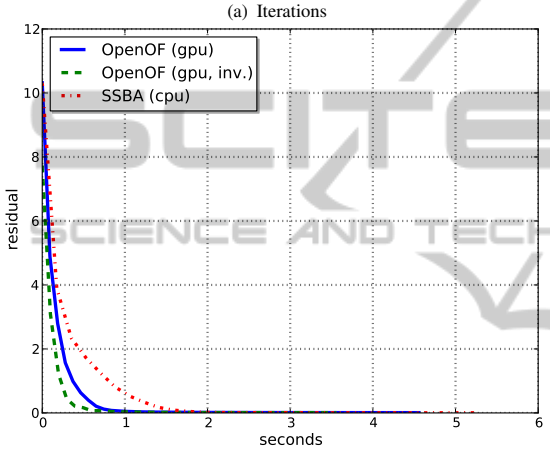
included in this part of the evaluation. It has an internally different error metric for the residuals so that *SSBA* and *PBA* can not be compared at the same time. As shown in Fig. 7(a), the convergence of *SSBA* and *OpenOF* is similar for BA with respect to the number of iterations. Regarding the number of iterations relative to the overall time, the CG approach outperforms the direct solver. This success is even larger for datasets where the amount of overlapping images is higher, as shown in the synthetic evaluation. The convergence rate of IBA is better than of BA. This might not hold in any case, but developers are encouraged to try out different parametrizations to determine the optimal solution. Additionally a different parametrization for the rotation is feasible, e.g. Euler angles.

8 CONCLUSIONS

We present *OpenOF*, an open source framework for sparse non-linear least squares optimization. The cost functions to be optimized are described in a high level scripting language. Reducing the implementation time enables developers to comfortably try different modelings of functions. The estimates are optimized on a GPU considering the sparse structure of the model. We showed that the generality of *OpenOF* is not accomplished at the expense of speed. *OpenOF* can compete with implementations that are specifically designed for certain applications.



(a) Iterations



(b) Time

Figure 7: Convergence of the mammoth dataset.

OpenOF works successful on BA. A fair comparison to the GPU version of *PBA* was not possible since *PBA* did not provide reliable results. We will further investigate why *PBA* shows just a poor convergence behavior on our data. With IBA a new parametrization for BA has been introduced which often converge for examples in which BA gets stuck in a local minimum. As in IBA the reprojection error of a point in the source camera is zero, IBA probably does not find the overall best solution. In practice we experienced that IBA still converges for multiple datasets where BA found a poor solution. In this case the result of IBA can be further optimized with an additional BA step.

The main aim of *OpenOF* is not to be another BA implementation but to solve much more complex least squares optimizations problems. Recently a feature has been added to *OpenOF* which allows the cost function to include numerical parts. This is useful for minimizing pixel differences in multiview stereo approaches. A set of various robust cost functions such as Huber or a truncated L2 norm is available as well.

For future releases we want to integrate a CPU version especially for developers which does not need the speed, but want to have the flexibility in their design of cost functions.

REFERENCES

Civera, J., a.J. Davison, and Montiel, J. (2008). Inverse Depth Parametrization for Monocular SLAM. *IEEE Transactions on Robotics*, 24(5):932–945.

Davis, T. A. (2006). *Direct Methods for Sparse Linear Systems*, volume 75 of *Fundamentals of Algorithms*. SIAM.

Furukawa, Y. and Ponce, J. (2010). Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–76.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2011). iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3281–3288. IEEE.

Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE.

Lourakis, M. (2010). Sparse non-linear least squares optimization for geometric vision. *Computer Vision - ECCV 2010*, pages 43–56.

Lourakis, M. I. A. and Argyros, A. A. (2009). SBA: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software*, 36(1):1–30.

Madsen, K., Nielsen, H. B., and Tingleff, O. (2004). Methods for Non-Linear Least Squares Problems (2nd ed.).

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*, volume 43 of *Springer Series in Operations Research*. Springer.

Wu, C., Agarwal, S., Curless, B., and Seitz, S. (2011). Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, number x, pages 3057–3064. IEEE.

Zach, C. (2012). Sources. http://www.inf.ethz.ch/personal/chzach/ opensource.