Abstract

3D reconstruction with missing data has been a very challenging computer vision task since the late 90s. This paper proposes the GPU-powered version of our weak-perspective Structure from Motion algorithm published in BMVC2008. Although this method is iterative, it is very rapid since all substeps in each iteration minimize the parameters optimally with respect to the reprojection error. We demonstrate here in both synthetic and real tests that the use of the GPU significantly reduces the time demand of the algorithm. Real-time 3D reconstruction is possible if the parameters of the optimization algorithm are set properly.

1. Introduction

Developing Structure from Motion (SfM) algorithms [10] is a very challenging research area. There are many efficient methods to compute the camera parameters and the 3D coordinates of the points if they are tracked in 2D through the frames of a video sequence.

The classical factorization method for the full case – when the measurement matrix is factorized into 3D motion and structure matrices – was developed by Tomasi and Kanade [22] for the very simple orthographic projection model in 1992. The weak-perspective extension was published by Weinshall and Kanade [26]. The factorization was extended to the paraperspective case [16] as well as to the real perspective one [21].

These methods have two main drawbacks: (i) they often cannot cope with missing data, (ii) even if they can, their computational demand is very high.

The problem of missing data was initially addressed by Tomasi and Kanade [22]. They proposed a naive approach which transforms the missing data problem to full matrix factorization by estimating the missing entries.

The mainstream idea to the factorization with missing data is to decompose the rank 4 measurement matrix into affine structure and motion matrices which are of dimension 4. (Shum’s method [4, 19] also computes affine structure and motion matrices, but the dimension of those matrices is 3. The Shum method was originally developed for range images, but it can be applied to the SfM problem as we demonstrated in [15].) This matrix decomposition can be done by using the mathematical method called Principal Component Analysis with Missing Data (PCAMD). This problem has been addressed by mathematicians since the middle 70’s [17] and can be applied directly to the SfM problem [4]. Hartley and Schaffalitzky [9] proposed the PowerFactorization method which is based on the Power method, which in turn is an iteration to compute the dominant $n$-dimensional subspace of a given matrix. Buchanan and Fitzgibbon [5] handled the problem as an alternation consisting of two nonlinear iterations to be solved. They suggested using the Damped-Newton method with line search to compute the optimal structure and motion matrices. An interesting approach was proposed by Whang et al. [25]. Their so-called quasi-perspective reconstruction fills the gap between affine and perspective ones. We have also proposed a factorization method [8, 15] which assumes weak-perspective camera and can be used for the full factorization [8] as well as for the one with missing data [15].

These weak-perspective methods can be used for perspective reconstruction if the elements of the measurement matrix are multiplied by the corresponding projective depths as discussed by several authors [7, 11, 21]; the perspective camera parameters can be estimated from weak-perspective camera reconstruction. Finally, the results can be refined by the well-known bundle adjustment (BA) method [2].

The aim of this study is to show that SfM algorithms can be run rapidly by utilizing the extreme computational power of the Graphics Processing Unit (GPU). To our best knowledge, the only solution to the perspective SfM problem using the GPU is that of Choudhary et al. [6] published at ECCV CVGPU workshop in 2010 which is the GPU-
powered version of the well-known BA method [2]. There are other kinds of GPU-based 3D reconstruction methods such as [20, 24], however, we only concentrate on the SfM problem here.

Remark that there are also real-time applications using the CPU, but these methods concentrate only on the motion reconstruction [13, 14] or can cope with only tens of feature points [18], therefore they cannot be used for realistic 3D object reconstruction.

The key idea of this paper is that if the perspective camera model is replaced by a simpler one, the adjustment process becomes significantly faster, especially if the GPU is used. We have selected the weak-perspective projection model. The goal of this paper is to accelerate our weak-perspective SfM algorithm [15], which solves the 3D reconstruction using the least-squares alternation scheme and can also cope with missing data.

The drawback of the algorithm is the use of weak-perspective projection. It is a good approximation of the real perspective projection if the depth of the object to be reconstructed is significantly smaller than the distance between the camera center and the object. The advantage of our weak-perspective factorization method is that it consists of optimal substeps [12, 15].

The rival BA algorithm [2] is significantly slower since it requires much more iterations because the error function optimized by BA is approximated using its first-order Taylor series. This approximation is very rough, and for this reason more iterations are required. In contrast, our method minimizes the total error function.

The main contribution of this paper is to demonstrate that real-time 3D reconstruction is possible by proposing an incremental extension to Pernek’s SfM algorithm, and providing a parallel, hardware-accelerated implementation. A very important advantage of our approach is that it enables realistic 3D object reconstruction in real time. The object itself may consist of thousands of feature points.

The structure of this study is as follows. In Section 2, we describe the algorithm itself. In Section 3, we discuss the techniques of our rapid implementation. Sections 4 and 5 are devoted to test results on synthetic and real data which validate the real-time nature of the algorithm. We conclude our paper in Section 6.

2. Basic Algorithm

Given \( P \) tracked feature points through \( F \) frames of an object, the aim of SfM algorithms is to compute the 3D coordinates of the points and the parameters of the \( F \) cameras. In this paper, weak-perspective projection is assumed. The projection of the point \( S_i = [X_i, Y_i, Z_i]^T \) in frame \( f \) is written under weak-perspective as follows:

\[
[u_{i}^f, v_{i}^f]^T = q^f R^f S_i + t^f \tag{1}
\]

where \( R^f, q^f \), and \( t^f \) are the first two rows of the rotation matrix, scale factor, and 2D offset vector of frame \( f \), respectively, while the projected coordinates are denoted by \( u_{i}^f \) and \( v_{i}^f \). The points of the object can be occluded as the camera moves throughout the sequence. Therefore, points are allowed to appear and disappear. The aim of our algorithm [15] is to minimize the error function:

\[
\left\| H \odot \left( W - [M]_t \left[ \begin{array}{c} S_1 \\ 1 \end{array} \cdots S_P \\ 1 \end{array} \right] \right) \right\|_F \tag{2}
\]

where \( \| \cdot \|_F \) denotes the Frobenius norm of the error matrix, \( A \odot B \) denotes the Hadamard product of matrices \( A \) and \( B \). \( H \) is the mask matrix: if \( h_{2i-1,j} = h_{2i,j} = 1 \), then the \( j \)th feature point is visible in the \( i \)th frame, otherwise \( h_{2i-1,j} = h_{2i,j} = 0 \). \( M = [q^1 R^1 \ldots q^F R^F]^T \) is the motion matrix. The so-called measurement matrix \( W \) contains the measured coordinates as

\[
W = \left[ \begin{array}{ccc} u_{1}^1 & \ldots & u_{1}^P \\ v_{1}^1 & \ldots & v_{1}^P \\ \vdots & \ddots & \vdots \\ u_{1}^F & \ldots & u_{1}^P \\ v_{1}^F & \ldots & v_{1}^P \end{array} \right]. \tag{3}
\]

The basic idea of our original paper [15] is that the minimization of Eq. 2 can be done using alternating least squares if the measured 2D vectors are completed to 3D ones. Before the alternation procedure, the motion and structure matrices have to be initialized. (We do not consider the initialization problem here, because the time demand of the initialization is significantly smaller than that of the iterative minimization.) For carrying out the completion of the measured values, the rotation matrices \( R^f \) and the offset vectors \( t^f \) should be completed first. The latter is the easier: the third coordinate of the offset vector is simply set to zero. The third row of the rotational matrix is given by the cross product of the first two rows. After the matrix \( R^f \) and offset vector \( t^f \) are completed, the third coordinates of the measured points are calculated by substituting the completed elements into Eq. 1.

The alternation method consists of three main substeps:

S-step. It estimates the 3D coordinates of the points if the camera motion is known. The estimation of each point can be optimally carried out in the least squares sense independently using the Moore-Penrose pseudoinverse: \( \hat{S}_i = M^+(W_i - \hat{t}) \), where \( W_i \) denotes the corresponding (measured) 3D coordinates in the image frames. The sign \( \hat{\cdot} \) denotes the corresponding vector/matrix containing only the motion information of the frames in which the processed point is visible. \( \hat{\cdot} \) stands for the pseudoinverse.

M-step. This estimates optimally the motion parameters of all the frames. Each frame can be processed independently. After the measurement matrix is completed,
the problem becomes a 3D registration one. To solve this, the (visible) 3D structure coordinates and the corresponding completed 3D coordinates of the measured values are assembled and registered to each other. The optimal offset vector \( \mathbf{t} \) is the offset between the centers of gravity of the two point sets. If the corresponding (offset) points are denoted by \( p_i \) and \( q_i \), the optimal motion is given by the singular value decomposition of the \( 3 \times 3 \) matrix \( \sum_i p_i^T q_i \). The scale itself is simply given as a fraction as shown in [15].

**Algorithm 1** Skeleton of weak-perspective SfM method

\[
M^{(0)}, t^{(0)}, S^{(0)} \leftarrow \text{Parameter Initialization}
\]

\[
W^{(0)}, M^{(0)}, t^{(0)} \leftarrow \text{Complete}(W, M^{(0)}, t^{(0)}, S^{(0)})
\]

\( k \leftarrow 0 \)

\( \text{repeat} \)

\( k \leftarrow k + 1 \)

\( S^{(k)} \leftarrow S\text{-Step}(H, W^{(k-1)}, M^{(k-1)}) \)

\( W^{(k)} \leftarrow \text{Update}(H, M^{(k-1)}, S^{(k)}, t^{(k-1)}) \)

\( M^{(k)}, t^{(k)} \leftarrow \text{M\text{-Step}(H, W^{(k)}, S^{(k)}, t^{(k-1)})} \)

\( \text{until } \| H \odot \left[ W^{(k)} - [M^{(k)}]_t^{(k)} \right] \|_F^2 \) converges.

**Algorithm 2** Skeleton of Weak-perspective Camera Calibration

\[
\text{repeat}
\]

\( W \leftarrow \text{Update}(M, S) \)

\( M \leftarrow \text{M-step}(S, W) \)

\( \text{until convergence.} \)

**2.1. Weak-perspective Camera Calibration**

A special version of the weak-perspective reconstruction is when the structure matrix \( S \) is known and only the parameters of the camera have to be estimated. This problem is called camera calibration. The (global) optimal solution for this problem is obtained by repeating the M-step and the Update as it is written in Algorithm 2. Note that the optimality of this algorithm has been proven in [12].

**Algorithm 3** Incremental reconstruction

\( (M_5, S_5) \leftarrow \text{Reconstruction}(W_5) \)

\( t \leftarrow 5 \)

\( \text{repeat} \)

\( \text{for } i = 1 \rightarrow RP - 1 \text{ do} \)

\( i \leftarrow i + 1 \)

\( t \leftarrow t + 1 \)

\( M_t \leftarrow \text{Camera Calibration}(W_t, S_{t-1}) \)

\( S_t \leftarrow S\text{-step}(W_t, M_t) \)

\( W_t \leftarrow \text{Update}(M_t, S_t) \)

\( \text{end for} \)

\( (M_t, S_t) \leftarrow \text{Refinement by Alg. 1. } (M_t, S_t, W_t) \)

\( \text{until there are no new frames.} \)

**2.2. Incremental Reconstruction**

If one wants to use the reconstruction algorithm in a real-time system, then it should be modified. At each time \( t \) the motion matrix \( M_t \) and the structure matrix \( S_t \) should be computed where the \( t \) index shows that the factorization is obtained using the feature points in frames 1 to \( t \). The corresponding measurement matrix is denoted by \( W_t \). If the reconstruction results of the previous timestamps (matrices \( M_{t-1} \) and \( S_{t-1} \)) are known, the motion matrix corresponding to the new timestamp \( t \) can be computed: it is a simple weak-perspective camera reconstruction. The input 3D coordinates are placed in \( S_{t-1} \), the corresponding 2D coordinates are the tracked feature points in the new frame at time \( t \). If a point in \( S_{t-1} \) is not visible in the new frame, it must not be considered when the camera calibration is carried out. After calibration, the matrix \( S_t \) has to be calculated. The coordinates in matrices \( S_t \) and \( S_{t-1} \) are the same except for the new points. A point is new at timestamp \( t \) if it is visible in the new frame and in two other (previous) frames since a feature point can be reconstructed only if it is visible in at least three frames. The computation of the 3D coordinates of the new point is done using the pseudoinverse since the coordinate estimation problem is linear as described in the S-step of the reconstruction algorithm. Then an Update step is executed to estimate the third coordinates of the new measured points.

Finally, the matrices should be refined using Algorithm 1. For faster execution, the refinement is not performed after every single frame but only for each RP-th. We call this parameter the Refinement Period in the algorithm. The whole incremental method is overviewed in Alg. 3.

**3. Rapid Structure from Motion**

The high speed of the proposed rapid implementation comes from two improvements: the used linear algorithm methods can be accelerated, and the substeps can be parallelized.

**3.1. Faster Matrix Computations**

General linear algebra methods in common mathematical software packages operate on arbitrary matrices. By
taking advantage of the fact that the dimensions of the matrices in our SfM algorithm are known, the general matrix operations can be accelerated as follows.

**Pseudoinverse.** In our original implementation, the SVD algorithm applied in the M-step is performed using Java Matrix Package (JAMA)\(^2\). The pseudoinverse is also calculated using SVD. However, this is not the fastest way. In the S-step, the pseudoinverse of a \(3F \times 3\) matrix \(M\) is computed, where \(F\) is the number of frames in which the processed point is visible. It is well known [3] that the pseudoinverse can be written as

\[
\tilde{M}^\dagger = (\tilde{M}^T \tilde{M})^{-1} \tilde{M}^T.
\]

The size of matrix \(\tilde{M}^T \tilde{M}\) is \(3 \times 3\). Its inverse can be written with the help of the adjoint matrix and the determinant. Therefore, we have implemented a special pseudoinverse algorithm which computes the inverse of matrices with 3 columns. This simplification reduces the computational load of the method. According to the tests we executed, our special pseudoinverse implementation is 15–25 times faster than the original one.

**SVD.** The SVD implementations in linear algebra software libraries such as JAMA contain iterative solutions [3], because the size of the matrix to be processed is arbitrary. In this SfM algorithm, the SVD is required only for solving the registration problem, and the matrix to be decomposed is always a \(3 \times 3\) one. The SVD itself has three subproblems: (i) For the calculation of the singular values, the eigenvalues of a \(3 \times 3\) matrix are required. This calculation is equivalent to finding the three real roots of a 3-degree polynomial. (ii) The calculation of the left and/or right singular vectors are given by determining the null-vectors corresponding to the singular values. In order to solve this homogenous linear problem, we have used the well-known Gauss-Newton elimination. (iii) If the left/right singular vectors are known, the vectors on the other side are obtained using simple matrix multiplications with normalization. The computational time of our \(3 \times 3\) SVD implementation is approximately half of that of the original one.

### 3.2. GPU implementation

The architecture of modern GPUs offers very high computing power to certain algorithms. The main requirement is that the execution be split into branches which are independent and can be executed concurrently. Although our algorithm is iterative and each step must be fully completed before starting the next one, the individual steps themselves are suitable for parallelization and are complex enough for the implementation to benefit from the GPU architecture.

We used OpenCL, a standard for general purpose programming on various devices; it is supported by the modern video card brands of both AMD and NVIDIA. It defines a C-like language for creating so called kernels, functions which are to be executed on the device, and an API for data transfer and execution control. The host code uses this API to access the device, send the input data to the device memory, initiate execution of kernels, and finally read the result data back to the main memory.

After the parameter initialization and completion steps, the GPU implementation of the algorithm transfers all four matrices, \(M, t, W,\) and \(S\) to the GPU memory. The M-step, the S-step, the Update, and the error computation kernels are executed appropriately, as required by the algorithm. As all four components run on the device, we can avoid most data transfer during the iteration itself. The result is read back to the main memory only when the algorithm terminates.

**Parallelism.** The key idea in our GPU-based implementation is that most parts of the algorithms can be run in parallel.

- **S-step.** Computation of the 3D coordinates are performed for each point independently, so the S-step translates to the GPU architecture in a straightforward manner. Since the number of points is generally high, and this step requires non-trivial calculations (including a pseudoinverse), we achieved great GPU utilization and performance improvement compared to the traditional implementation.

- **M-step.** The M-step can be executed for each frame concurrently. The number of frames is usually lower than the number of points, but 3D registration is even more computationally intensive than the 3D point estimation of the S-step. For optimal parallel performance, we avoided allocations of large temporary data structures and handled missing data in-place.

- **Update step.** As Eq. 1 shows, every single projection can be calculated independently.

- **Computation of reprojection error.** Although the error of each measured point can be computed independently, summation does not translate to a parallel architecture as it is. In our implementation of this step we first calculate the entire error matrix, then execute a recursive parallel reduction algorithm to produce the sum of more and more elements until the entire matrix is processed. The main benefit of such an implementation is to avoid reading back the matrices to the CPU for the error calculation, as the cost of data transfer is very high.

### 4. Tests on synthetic data

Several experiments with synthetic data have been carried out to study the properties of the reconstruction methods. We focus on the time demand of our following three

\(^{2}\)http://math.nist.gov/javanumerics/jama/
implementations. CPU: Straightforward single-threaded implementation running on an Intel Core4Quad 2.33 GHz CPU with 4 Gbyte memory. CPU4TH: The multi-threaded version of the same code using the same CPU. GPU256TH: The OpenCL port of the multi-threaded version running on the NVIDIA GTX 285 GPU with 256 parallel threads.

All three implementations include the faster matrix operations described in Sec. 3.1.

We have tested both the offline and the incremental version of the SfM algorithm.

For the synthetic tests, we have generated the input measurement matrices as follows. (i) A random 3D point cloud has been generated. (ii) Then these points have been rotated randomly and projected into the hypothetic image plane using weak-perspective projection. (iii) Random Gaussian noise has been added to the projections. (iv) Finally, the coordinates have been scaled to the interval [0, 1000].

4.1. Offline SfM

![Figure 1](image)

Figure 1. Time demand and iteration numbers of implementations w.r.t. number of points (top), number of frames (middle), missing data ratio (bottom).

The plots in Fig. 1 show the execution times of the three implementations as a function of the number of points, the number of frames, and the missing data ratio. For each graph, one of these test parameters was incremented, the other two were set to a fix value. This value was 1000 for the number of points, 50 for the number of frames, and 50% for the missing data ratio. The termination threshold δ was set to $10^{-5}$ pixel.

The first plot shows that the execution time of the algorithm is approximately linear in the number of points. When the number of frames increases (middle chart), the higher computational demand of the steps themselves is offset by the lower number of iterations required. This can be explained by the initial parameter estimates being of better quality. The bottom graph shows that the missing data ratio has little effect on execution time until about 60%. Then the algorithm starts requiring an increasing number of iterations, and this results in a sharp rise of execution time, even though the less points are visible, the faster each step becomes, since they only use the visible points.

We have performed an additional test with 20000 points, and 500 frames to compare our method to the GPU-powered BA algorithm. The total execution time was 3.8 sec. This is less than the time demand of a single iteration of the GPU-based BA. The total execution time of BA for this huge amount of data is approximately one day as it is analyzed in [6].

4.2. Incremental SfM

For brevity, and since the incremental variant is aimed for real-time conditions, we only present the results for the GPU version. (The general relationship between the performance of the three implementations is the same as in the offline case.)

In our tests, we used the following parameters by default: 1000 points, 50 frames, 50% missing data ratio, 2.5% noise, an error threshold of $δ = 10^{-3}$ pixel, and refinement every fifth frame (refinement period of 5).

Fig. 2 displays the performance of the algorithm in frames per second (FPS) and the final reprojection error with respect to noise level. The latter is the root mean square (RMS) of the error values calculated by Eq. 2. We see that the algorithm becomes faster as the noise increases because less iterations are required altogether. We cannot explain this behavior, but we can clearly state that the reconstruction is real-time. As expected, the error increases with noise.

In accordance with our expectations, the algorithm becomes slower as the number of points is increased (Fig. 3). The final RMS error is steadily between 1 and 1.3 pixels.

As we increased the total number of images (Fig. 4), the speed of processing increased at first, then it levelled off at around 65 FPS (with error values staying in the 1-1.3 pixel range). This promising result indicates that, although the iterative refinement struggles at the start when few images are available, it can sustain high performance in the long-term.

The missing data chart (Fig. 5) confirms the idea that the more data are available, the less iterations are required to
achieve the error threshold. We can even observe a similar jump at 60% missing data as we did in Fig. 1. Also note that for this test sequence we set the number of points to 2500 instead of the default 1000 in order to accommodate to higher ratios of missing data. The error plot shows that missing data ratio has little effect on the final error.

Finally, in Figure 6, we demonstrate that we can reduce the frequency of iterative refinement during the algorithm to achieve higher throughput without sacrificing accuracy. The RMS reprojection error with respect to the frame number is visualized in Fig. 7. It is clear that the refinement significantly reduces the final error especially when the reconstruction is computed from less frames.

5. Tests on real objects

5.1. Offline SfM

We have tested the implementations on two real sequences. They have produced identical results, and as expected, the parallel ones were significantly faster than the serial one.

For these tests we used the termination condition $\delta = 10^{-3}$.

'Dino' sequence. The test data were downloaded from the web page of the Oxford University\footnote{http://www.robots.ox.ac.uk/~amb/}. The sequence consists of 36 frames and 319 tracked points. 77% of the measurement matrix is missing. The input images and the reconstructed 3D points are visualized in Fig. 8. The time demands of the CPU, CPU4TH, and GPU256TH implementations were 5.4, 2.4, and 1.3 sec, respectively.

'Cat' sequence. We have tested the implementations on our ‘Cat’ sequence pictured in Fig. 9. The cat statuette was rotated on a table and 92 photos were taken using a commercial digital camera. Feature points were detected using the widely used KLT algorithm \cite{23}, and the points were
tracked by a correlation-based template matching method. The measurement matrix of the sequence consists of 2,290 points and 92 frames. The missing data ratio is 82%. The time demands of the CPU, CPU4TH, and GPU256TH implementations were 14.3, 4.6, and 1.2 sec, respectively. The decrease of the RMS reprojection error is pictured in Fig. 10. It is monotonic, as expected (since each substep of the algorithm is optimal).

5.2. Incremental vs. offline SfM

Table 1 compares the performance of the incremental and the offline variants of our reconstruction algorithm for the two real sequences. The incremental variant did not reach real-time speed, due to the high missing data ratio; this result is in accordance with the synthetic tests (see Figure 5). By increasing the termination threshold from $10^{-3}$ to $10^{-2}$ (Table 2), both sequences could be processed at over 15 FPS, the speed of conventional web cameras. The cost of this change is quite small with regards to the final reprojection error (for the ‘Cat’ sequence, the error increased from 3 to 3.5 pixels; and from 1.9 to 2.4 pixels for the ‘Dino’ sequence).

Still, the offline algorithm, having more data available to it, performs significantly better. These results suggest that the incremental version should only be used when it is important to obtain partial reconstruction results during the process.
6. Conclusion

In this paper, we have discussed that the GPU-powered version of our weak-perspective SfM algorithm presented in [15] is very rapid due to two reasons: (i) the substeps of the iterative algorithm consist of optimal estimations in the least squares sense and (ii) the substeps of the iterative algorithm can be parallelized efficiently. We have developed two variants of the method for offline and online 3D reconstruction. We have shown that the online version can be operated in real-time for realistic test sequences. The time demand of our implementations has been demonstrated for synthetic and real test data. The source code with GPU support is available on the web at [1].

Acknowledgement. The authors would like to thank Prof. Dmitry Chetverikov for his council. This research has been supported by the NKTH-OTKA grant CK 78409.

References

[23] Tomasi, C. and Shi, J. Good Features to Track. In IEEE Conf. Computer Vision and Pattern Recognition, pages 593–600, 1994. 6