High-performance image reconstruction in fluorescence tomography on desktop computers and graphics hardware

Manuel Freiberger,[1,*] **Herbert Egger,**[2] **Manfred Liebmann,**[2] **and Hermann Scharfetter**[1]

[1]*Graz University of Technology, Institute of Medical Engineering, Kronesgasse 5/II, 8010 Graz, Austria*

[2]*University of Graz, Institute for Mathematics and Scientific Computing, Heinrichstr. 36/III, 8010 Graz, Austria*

**manuel.freiberger@tugraz.at*

Abstract: Image reconstruction in fluorescence optical tomography is a three-dimensional nonlinear ill-posed problem governed by a system of partial differential equations. In this paper we demonstrate that a combination of state of the art numerical algorithms and a careful hardware optimized implementation allows to solve this large-scale inverse problem in a few seconds on standard desktop PCs with modern graphics hardware. In particular, we present methods to solve not only the forward but also the non-linear inverse problem by massively parallel programming on graphics processors. A comparison of optimized CPU and GPU implementations shows that the reconstruction can be accelerated by factors of about 15 through the use of the graphics hardware without compromising the accuracy in the reconstructed images.

OCIS codes: (100.3190) Inverse problems; (100.3010) Image reconstruction techniques; (170.6960) Tomography; (170.7050) Turbid media; (300.6280) Spectroscopy, fluorescence and luminescence.

References and links

1. S. Mordon, V. Maunoury, J. M. Devoisselle, Y. Abbas, and D. Coustaud, “Characterization of tumorous and normal tissue using a pH-sensitive fluorescence indicator (5,6-carboxyfluorescein) in vivo,” J. Photochem. Photobiol. B **13**, 307–314 (1992).
2. I. Gannot, I. Ron, F. Hekmat, V. Chernomordik, and A. Gandjbakhche, “Functional optical detection based on pH dependent fluorescence lifetime,” Lasers Surg. Med. **35**, 342–348 (2004).
3. E. Shives, Y. Xu, and H. Jiang, “Fluorescence lifetime tomography of turbid media based on an oxygen-sensitive dye,” Opt. Express **10**, 1557–1562 (2002).
4. B. Zhang, X. Yang, F. Yang, X. Yang, C. Qin, D. Han, X. Ma, K. Liu, and J. Tian, “The CUBLAS and CULA based GPU acceleration of adaptive finite element framework for bioluminescence tomography,” Opt. Express **18**, 20201–20214 (2010).
5. J. Prakash, V. Chandrasekharan, V. Upendra, and P. K. Yalavarthy, “Accelerating frequency-domain diffuse optical tomographic image reconstruction using graphics processing units,” J. Biomed. Opt. **15**, 066009 (2010).
6. F. Knoll, M. Freiberger, K. Bredies, and R. Stollberger, “AGILE: An open source library for image reconstruction using graphics card hardware acceleration,” in “Proceedings of the 19th Scientific Meeting and Exhibition of ISMRM, Montreal, CA,” (2011), p. 2554.
7. S. R. Arridge, “Optical tomography in medical imaging,” Inv. Probl. **15**, R41–R93 (1999).
8. A. Joshi, W. Bangerth, and W. M. Sevick-Muraca, “Adaptive finite element based tomography for fluorescence optical imaging in tissue,” Opt. Express **12**, 5402–5417 (2004).

#149380 - $15.00 USD Received 16 Jun 2011; revised 4 Aug 2011; accepted 16 Aug 2011; published 28 Oct 2011

9. S. R. Arridge and M. Schweiger, “The use of multiple data types in time-resolved optical absorption and scattering tomography (TOAST),” in “Mathematical Methods in Medical Imaging II,” , D. C. Wilson and J. N. Wilson, eds. (1993), Proc. SPIE 2035, pp. 218–229.
10. E. M. Sevick and B. Chance, “Photon migration in a model of the head measured using time and frequency domain techniques: potentials of spectroscopy and imaging,” in “Time-Resolved Spectroscopy and Imaging of Tissues,” , B. Chance and A. Katzir, eds. (1991), Proc. SPIE 1431, pp. 84–96.
11. S. R. Arridge, “Photon-measurement density functions. part I: Analytical forms,” Appl. Opt. **34**, 7395–7409 (1995).
12. A. Bakushinsky, “The problem of the convergence of the iteratively regularized Gauss-Newton method,” Comput. Math. Math. Phys. **32**, 1353–1359 (1992).
13. M. Hanke, “Regularizing properties of a truncated Newton-CG algorithm for nonlinear inverse problems,” Numer. Func. Anal. Optim. **18**, 971–993 (1997).
14. D. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” SIAM J. Appl. Math. **11**, 431–441 (1963).
15. B. Kaltenbacher, “Some Newton-type methods for the regularization of nonlinear ill-posed problems,” Inv. Probl. **13**, 729–753 (1997).
16. H. Jiang, “Frequency-domain fluorescent diffusion tomography: A finite-element-based algorithm and simulations,” Appl. Opt. **37**, 5337–5343 (1998).
17. R. Roy and E. M. Sevick-Muraca, “Truncated Newtons optimization scheme for absorption and fluorescence optical tomography: Part I theory and formulation,” Opt. Express **4**, 353–371 (1999).
18. M. Schweiger, S. R. Arridge, and I. Nissilä, “Gauss–Newton method of image reconstruction in diffuse optical tomography,” Phys. Med. Biol. **50**, 2365–2386 (2005).
19. H. Egger and M. Schlottbom, “Efficient reliable image reconstruction schemes for diffuse optical tomography,” Inv. Probl. Sci. Eng. **19**, 155–180 (2011).
20. A. Godavarty, E. M. Sevick-Muraca, and M. J. Eppstein, “Three-dimensional fluorescence lifetime tomography,” Med. Phys. **32**, 992–1000 (2005).
21. M. Freiberger, H. Egger, and H. Scharfetter, “Nonlinear inversion schemes for fluorescence optical tomography,” IEEE Trans. Biomed. Eng. **57**, 2723–2729 (2010).
22. D. Göddeke, C. Becker, and S. Turek, “Integrating GPUs as fast co–processors into the parallel FE package FEAST,” in “19th Symposium Simulations Technique,” , M. Becker and H. Szczerbicka, eds., Frontiers in Simulation (SCS Publishing House, 2006), pp. 277–282.
23. M. M. Baskaran and R. Bordawekar, “Optimizing sparse matrix-vector multiplication on GPUs,” IBM Technical Report RC24704, IBM Ltd. (2009).
24. M. Liebmann, “Efficient PDE solvers on modern hardware with applications in medical and technical sciences,” Ph.D. thesis (University of Graz, 2009).
25. W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial* (SIAM, 2000), 2nd ed.
26. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide* (SIAM, 1999), 3rd ed.
27. B. Dogdas, D. Stout, A. F. Chatziioannou, and R. M. Leahy, “Digimouse: a 3D whole body mouse atlas from CT and cryosection data,” Phys. Med. Biol. **52**, 577–587 (2007).
28. G. Alexandrakis, F. R. Rannou, and A. F. Chatziioannou, “Tomographic bioluminescence imaging by use of a combined optical-PET (OPET) system: a computer simulation feasibility study,” Phys. Med. Biol. **50**, 4225–4241 (2005).
29. M. Keijzer, W. M. Star, and P. R. M. Storchi, “Optical diffusion in layered media,” Appl. Opt. **27**, 1820–1824 (1988).
30. A. Joshi, “Adaptive finite element methods for fluorescence enhanced optical tomography,” Ph.D. thesis (Texas A&M University, 2005).
31. M. L. Landsman, G. Kwant, G. A. Mook, and W. G. Zijlstra, “Light-absorbing properties, stability, and spectral stabilization of indocyanine green,” J. Appl. Physiol. **40**, 575–583 (1976).
32. J. Schöberl, “NETGEN - an advancing front 2D/3D-mesh generator based on abstract rules,” Comput. Vis. Sci. **1**, 41–52 (1997).
33. NVIDIA, *NVIDIA CUDA Programming Guide 2.0* (NVIDIA Cooperation, 2008).
34. N. Bell and M. Garland, “Implementing sparse matrix-vector multiplication on throughput-oriented processors,” in “SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis,” (ACM, 2009), pp. 1–11.
35. D. Göddeke, H. Wobker, R. Strzodka, J. Mohd-Yusof, P. S. McCormick, and S. Turek, “Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU,” Int. J. Comput. Sci. Eng. **4**, 254–269 (2009).
36. M. Köster, D. Göddeke, H. Wobker, and S. Turek, “How to gain speedups of 1000 on single processors with fast FEM solvers – benchmarking numerical and computational efficiency,” Tech. rep., Fakultät für Mathematik, TU Dortmund (2008). Ergebnisberichte des Instituts für Angewandte Mathematik, Nummer 382.
37. V. A. Morozov, “On the solution of functional equations by the method of regularization,” Soviet Math. Dokl. **7**, 414–417 (1966).

38. H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of Inverse Problems* (Kluwer, 1996).
39. P. C. Hansen, *Rank-Defficient and Discrete Ill-Posed Problems* (SIAM, 1998).
40. G. Wahba, *Spline Models for Observational Data* (SIAM, 1990).

1. Introduction

In this paper, we study an indirect measurement technology, namely fluorescence optical tomography, which relies on the ability of fluorescent dyes (fluorophores) to absorb and re-emit light at different wavelengths. The aim of fluorescence tomography is to obtain the distribution of a fluorophore inside an object (an only indirectly accessible quantity) from optical measurements at the boundary (an easily obtainable information). The strength of this imaging modality is the availability of a large number of fluorophores which change their optical properties in dependence of its biological surrounding. Thus, they reflect physiologically interesting parameters like the pH value [1, 2] or tissue oxygenation [3].

Unfortunately, photons are massively scattered in biological tissues. Therefore, fluorescence tomography requires the accurate simulation of light propagation in highly scattering media and the solution of ill-posed nonlinear inverse problems, which are computationally intensive tasks. Fast and reliable numerical algorithms will crucially determine the success of this new imaging modality in biomedical sciences as well as in engineering applications.

A recent trend is the application of graphics processing units (GPUs) in scientific computations. In the field of optical tomographic imaging, Zhang et al [4] have reported GPU accelerated bioluminescence reconstructions and make use the proprietary package CULA by EM Photonics for the inversion of the finite element system matrix. Prakash et al [5] also use CULA from within the Matlab-framework NIRFAST to speed up the reconstruction algorithm for diffuse optical tomography. In their work, six distinct tasks are GPU-accelerated such as the solution of a linear equation system and matrix-matrix multiplications.

The previously mentioned works use the GPU as co-processor mainly, i.e. data is transferred to the graphics card where certain operations are performed in parallel. The result is copied back to the CPU then. This submission is different in the sense that we aim to perform nearly all computations on the graphics hardware without the need to transfer intermediate data between the CPU and the graphics card. Thus, the GPU is not only used for trivial multiplication tasks but also for the assembly of the forward and adjoint system matrices and the sensitivity matrix, for example. This reduces the comparatively slow data transfer between the computer and the graphics hardware to a minimum and results in much faster image reconstruction. Furthermore, all algorithms are iterative which circumvents the need for matrix inversion algorithms. Our implementation is based on the open-source library *AGILE* [6] which is available at `http://www.imt.tugraz.at/research/agile-gpu-image-reconstruction-library/`.

The outline of this paper is as follows: Section 2 provides background material about fluorescence tomography and describes the iterative regularization method that is used for the stable solution of the inverse problem. Section 3 deals with the discretization of the problem by finite element methods and gives details for an efficient implementation on graphics hardware. In Section 4, we define a test problem and derive estimates for the complexity and required memory of the reconstruction algorithm. The theoretical results are confirmed by numerical tests, in which we compare the performance of our algorithms on CPU and GPU hardware. Our tests illustrate that the reconstruction times can be reduced significantly (by a factor of about 15) through the use of the graphics hardware.

2. Methods

2.1. Mathematical model

The propagation of light in highly scattering media, e.g. in biological tissue, is typically modeled by the diffusion approximation [7], which serves as a first order approximation to the radiative transport equation. Since light of two different wavelengths is employed in fluorescence tomography, the mathematical model consists of a coupled system of partial differential equations (PDE) [8], namely

$$-\mathrm{div}(\kappa_x \nabla \varphi_x) + \mu_x \varphi_x = q, \qquad \text{in } \Omega, \tag{1}$$

$$-\mathrm{div}(\kappa_m \nabla \varphi_m) + \mu_m \varphi_m = \gamma c \varphi_x, \qquad \text{in } \Omega. \tag{2}$$

Here, φ_i, $i = x, m$ are the photon densities for the excitation (x) and the emission (m) light in the domain Ω covered by tissue, and the source term q models a collimated light source. The parameters κ_i and μ_i are the diffusion and absorption coefficients of the tissue, and γ is a measure incorporating the fluorophore's extinction coefficient and its quantum yield. We assume that these parameters depend in a known form on the concentration c of fluorophore in the tissue, and we write $\kappa_i(c)$, $\mu_i(c)$ or $\gamma(c)$ if this dependence needs to be emphasized.

Homogeneous Robin boundary conditions [7]

$$\rho \varphi_i + \kappa_i \frac{\partial \varphi_i}{\partial n} = 0, \qquad \text{on } \partial\Omega, \qquad i = x, m. \tag{3}$$

are used to model that no light enters the domain from the outside apart from the light injected through the internal source q.

The simplest type of boundary measurement is the photon flux at the emission wavelength which leaves the domain which is modeled by

$$m(\varphi_m) = \int_{\partial\Omega} d(s) \varphi_m(s) ds, \tag{4}$$

where the function d allows to model the aperture and additional (linear) transfer characteristics of the detector. Other types of measurement functions are possible as well and have been used in e.g. [9–11].

The tomographic measurement process finally consists of illuminating the object with a sequence of light sources q_j, $j = 1, \ldots, ns$, and taking measurements of the emission light with an array of detectors d_i, $i = 1, \ldots, nd$ at the boundary. This gives rise to a measurement matrix $\mathscr{M}$ of dimension $nd \times ns$, which obviously depends on the distribution c of the fluorophore, and we write $\mathscr{M}(c)$ to express this dependence.

2.2. Image reconstruction

The aim of fluorescence tomography is to determine the inaccessible fluorophore distribution c which gave rise to the measurements $\mathscr{M}^\delta$—which are perturbed by measurement noise and model errors—of the emitted light. A Tikhonov-type regularization term is incorporated in the objective functional for stability reasons. Then the reconstruction problem reads

$$\mathrm{argmin}_c \left(\left\| \mathscr{M}(c) - \mathscr{M}^\delta \right\|^2 + \frac{\alpha}{2} \|c - c_0\|^2 \right). \tag{5}$$

To solve this problem in a fast and stable manner, we utilize the iteratively regularized Gauß-Newton method (IRGN) [12], which leads to the iterative algorithm

$$(S_k^* S_k + \alpha_k I)(c_{k+1} - c_k) = S_k^*(\mathscr{M}^\delta - \mathscr{M}(c_k)) + \alpha_k (c_0 - c_k) \qquad k = 0, 1, \ldots. \tag{6}$$

Here, $S_k := \frac{\partial \mathscr{M}}{\partial c}(c_k)$ denotes the sensitivity operator (i.e. the Jacobian), which measures the effect of a change in the fluorophore distribution c onto the output $\mathscr{M}(c)$ and S_k^* is its adjoint. The regularization parameter α_k stabilizes the solution and is usually decreased in every iteration.

For the sake of completeness, we would like to mention other possibilities for the regularizing of the Newton iteration, e.g. the truncated Newton-CG algorithm [13], the Levenberg-Marquardt method [14], or the Newton-Landweber method [15]. Several of these approaches have been demonstrated to work well for optical tomography applications [8, 16–19]. Non-quadratic penalty terms have been studied e.g. in [20, 21].

3. Implementation

In the following, we discuss the the main steps of the reconstruction algorithm in more detail and provide information for an efficient implementation. In order to quantify the expected computational workload, we also derive complexity estimates for the individual steps and provide upper bounds for the memory requirements.

3.1. *Discretization*

For the discretization of the governing partial differential equations, we consider a standard finite element method with piecewise linear basis functions for the photon fields. In our computations, we use an unstructured mesh with N_V vertices consisting of N_T tetrahedral elements. For computational efficiency but also simplicity, the optical parameters and the fluorophore concentration c are discretized by piecewise constant functions over the same mesh. The discretized version of the system (1)–(2) reads

$$[K_x(c) + M_x(c) + R_x]V_x = Q \tag{7}$$

$$[K_m(c) + M_m(c) + R_m]V_m = G(c)V_x. \tag{8}$$

In this linear system, K_i, M_i and R_i, $i \in \{x, m\}$, denote the stiffness-, mass- and boundary mass-matrices at the respective wavelength. Each column of the $N_V \times ns$ matrix Q corresponds to a different excitation q_j, $j = 1, \ldots, ns$, and the ns columns of the solution matrices V_x and V_m store the vector representations of the corresponding photon density fields φ_x and φ_m. The measurement data are given by $\mathscr{M}(c) = D^\top V_m$, where each of the nd columns of the detector matrix D corresponds to one of the detectors d_i, $i = 1, \ldots, nd$, in (4).

The discrete sensitivity $S = S(c)$ is an $nd \times ns \times N_T$ tensor (respectively an $(nd \cdot ns) \times N_T$ matrix). Its action onto a parameter perturbation $h \in \mathbb{R}^{N_T}$ is given by

$$\begin{aligned} Sh := &-W_m^\top K_m(h)V_m - W_m^\top M_m(h)V_m \\ &- W_x^\top K_x(h)V_x - W_x^\top M_x(h)V_x + W_m^\top G(h)V_x, \end{aligned} \tag{9}$$

where h is an arbitrary element from the N_T dimensional parameter space, and W_m, W_x denote the solutions of the adjoint system

$$[K_m(c) + M_m(c) + R_m]W_m = D, \tag{10}$$

$$[K_x(c) + M_x(c) + R_x]W_x = G(c)W_m. \tag{11}$$

The fully discrete equivalent of the iteratively regularized Gauß-Newton method (6) has the form listed in Algorithm 1.

Before going into details of the implementation and giving complexity estimates for the individual step as well as upper bounds for the memory requirements, let us make some general remarks:

Algorithm 1 Iteratively regularized Gauß-Newton algorithm

```
1: initialize
2: $c \leftarrow 0$
3: for $k = 1$ to maxit do
4:     $[A_x, A_m, G] \leftarrow \texttt{assembleSystemMatrices}(c_k)$
5:     $V_x \leftarrow \texttt{PBCG}(A_x, Q)$
6:     $V_m \leftarrow \texttt{PBCG}(A_m, GV_x)$
7:     $M \leftarrow D^\top V_m$
8:     $W_m \leftarrow \texttt{PBCG}(A_m, D)$
9:     $W_x \leftarrow \texttt{PBCG}(A_x, GW_m)$
10:    $S_k \leftarrow \texttt{assembleSensitivity}(c_k, V_x, V_m, W_x, W_m)$
11:    $dc \leftarrow \left(S_k^\top S_k + \alpha_k I\right)^{-1} S_k^\top (M^\delta - M)$
12:    $c \leftarrow c + dc$
13:    $\alpha_{k+1} \leftarrow q\alpha_k$
14: end for
```

(i) The presented algorithms are independent of the architecture used for the actual computations, i.e. apart from some implementation details, the same algorithms are used for CPU and GPU implementations. In our numerical tests, the high-level algorithms are always controlled by the CPU, while the actual computations (e.g. matrix-vector multiplications) are executed on the CPU or GPU, respectively, inside a few *kernels*. Only these few kernels have to be adapted to the specific hardware. The advantage of such a “minimally invasive” framework is that algorithms can be migrated gradually from one hardware to another while at the same time the high-level abstraction of the algorithms can be preserved. Such an approach seems very natural, and has been employed previously, e.g. for the simulation of fluid dynamics on CPU-GPU clusters [22].

(ii) All compute intensive steps of the reconstruction algorithm have a high degree of locality: A major part of the operations, e.g. the assembly of the system matrices or the sensitivity, can be performed in an element-by-element fashion. This means that similar computations can executed for individual elements independently from each other. Global operations, e.g. the solution of the linear systems, rely on sparse matrix algebra. Note that each row (or column) of the sparse matrices corresponds to a vertex of the mesh, and the non-zero entries of this row stem from degrees of freedom (vertices) belonging to the patch of elements surrounding this vertex. Therefore, also sparse-matrix operations share a high degree of locality, and consequently, the overall reconstruction algorithm is very well-suited for parallelization.

3.2. Initialization

The first task is to set up a multilevel mesh hierarchy: The vertex coordinates of the finest level are loaded, and the transfer operators between adjacent levels l and $l+1$ are stored as sparse prolongation matrices P_l^{l+1}. This allows to lift a function from level l to the next finer level $l+1$, i.e. $u_{l+1} = P_l^{l+1} u_l$ denotes the linear prolongation of a piecewise linear finite element function u_l to the finer mesh where it is represented by u_{l+1}. The mesh hierarchy is only required for the initialization step. The memory requirement for storing the prolongation matrices P_l^{l+1} is $O(N_V)$.

The element vertex numbers are stored in an $N_T \times 4$ table containing the global vertex numbers for the N_T individual elements. The 4-element array g^T then contains the numbers of the vertices spanning the tetrahedron T. During the setup step, also the vertex-to-element connectivity is required, which is simply given by $L_V^E = (L_E^V)^\top$. The memory requirement for these

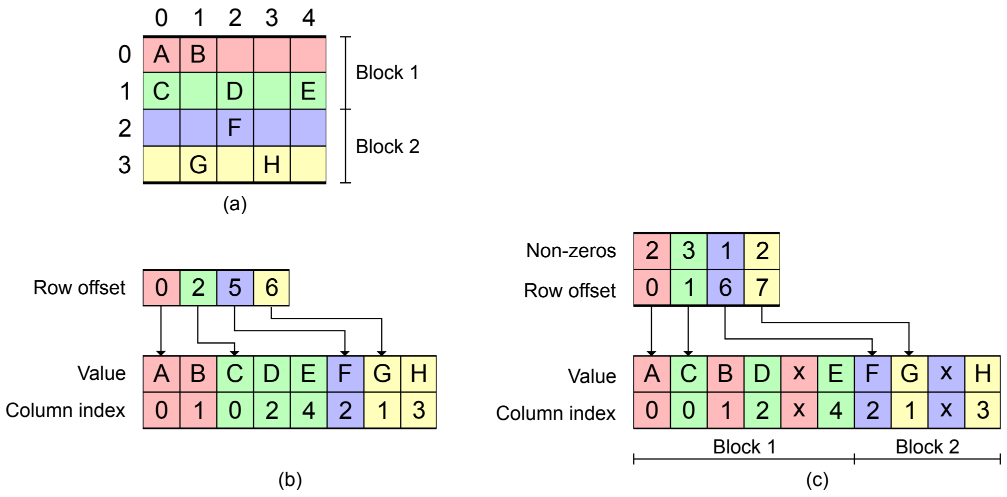


Fig. 1. (a) Sparse matrix which is to be stored in compressed row-storage (CRS) format. (b) The conventional memory layout on the CPU consists of the linearized data elements together with their column indices and a vector holding the offset of each row in the data vector. The number of elements in a row is given implicitly by the difference in the row offsets. (c) The adapted layout for the GPU stores the rows in an interleaved manner which requires memory padding marked with $\times$ and an additional vector holding the number of non-zero entries per row.

look-up tables is $O(N_V + N_T)$.

One additional look-up table is needed for the efficient assembly of the sparse system matrix. This matrix is stored in compressed row-storage (CRS) format. Its sparsity pattern can be created from the mesh hierarchy and the mapping (local index $\rightarrow$ global index $\rightarrow$ CRS data index) can be readily computed. We store the inverse of this mapping, which is a one-to-many table and requires $O(N_T)$ memory.

Finally, we pre-compute the mass-, boundary-mass- and stiffness-matrices, M^T, R^T and K^T, for unit coefficients:

$$K_{ij}^T := \int_T \nabla\psi_{g^T(i)} \nabla\psi_{g^T(j)} dx, \qquad M_{ij}^T := \int_T \psi_{g^T(i)} \psi_{g^T(j)} dx$$

$$\text{and} \quad R_{ij}^T := \int_{\partial T \cap \partial\Omega} \psi_{g^T(i)} \psi_{g^T(j)} dx \qquad 1 \le i, j \le 4.$$

Each expression results in N_T matrices of size 4×4, which are copied to the GPU memory.

3.3. *Sparse matrix format*

The solution of linear equation systems heavily relies on fast sparse matrix-vector products. For comparison of matrix-vector kernel performance of different sparse storage layouts we refer to [23]. The results presented in this paper are based on a blocked interleaved compressed row-storage (CRS) format sketched in Fig. 1. In contrast to the conventional CRS layout, the adapted storage scheme stores the elements of the rows interleaved which makes access from different threads more efficient. As not all of the rows have the same number of non-zero entries, the shorter rows have to be padded with dummy elements. In order not to waste too much memory, this padding is done in blocks of 32 rows (a block-size of two is used in Fig. 1 to illustrate the technique). Further details about this format can be found in [24].

3.4. Assembly of the system matrices

When assembling of the system matrix, the individual element contributions have to be summed. In a straightforward implementation this would result in non-local random write-access on the output matrix which is costly on graphics hardware. However, due to the piece-wise constant discretization of the coefficients κ, μ, ρ and c, the contribution of an element T to the excitation and emission system is given as

$$A_x^T = \kappa_x(c_k^T)K^T + \mu_x(c_k^T)M^T + \rho_x R^T, \tag{12}$$

$$A_m^T = \kappa_m(c_k^T)K^T + \mu_m(c_k^T)M^T + \rho_m R^T. \tag{13}$$

Therefore, we split the assembly into two steps: First, the pre-computed element matrices K^T, M^T and R^T are scaled with the element's optical parameters and the output is stored in a vector H with length $(4 \cdot 4 \cdot N_T)$. In the second step, we sum for every position in the CRS matrix the corresponding elements from H. In principle, this summation can be done by multiplying a suitable sparse matrix with H. However, as all non-zero elements in this matrix are 1, we implemented a specialized GPU kernel for this task, which simply sums the elements whose indices are stored in a look-up table. By using such a two-step scheme, we avoid non-local write-access and need non-local read-access only for the second summation step, which is implemented using the GPU's texture cache. Figure 2 illustrates this procedure. As the construction of the local element matrices does only require local memory access, it can be done fully in parallel which provides almost optimal performance on both, CPU and GPU processors. The total complexity of this step is $O(N_T)$.

3.5. Solution of the linear systems

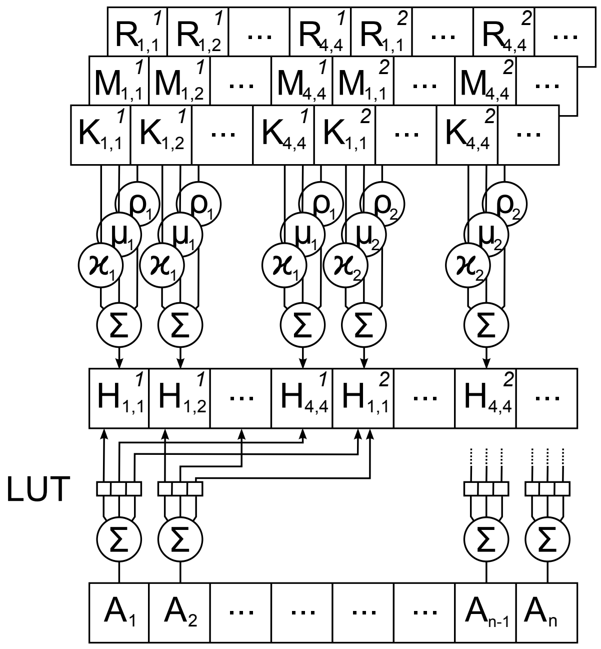
For the solution of the linear systems $A_xV_x = Q$ and $A_mV_m = G(c)V_x$ for the forward problem, respectively, $A_mW_m = D$ and $A_xW_x = G(c)W_m$ for the adjoint problem, we employ a preconditioned blocked conjugate gradient method (`PBCG`): instead of solving for one right hand side (i.e. a column in Q) after another, we solve simultaneously for all right hand sides (blocked). The algorithm uses the PCG framework provided by the *AGILE* library. Thus, only a new kernel had to be coded for the blocked matrix-vector-product.

To accelerate the convergence, a single V-cycle of a geometric multigrid preconditioner [25] is used. On the coarsest mesh level, an exact solve is performed by a blocked conjugate gradient method (`BCG`), which is similar to `PBCG` but without further preconditioning. A single `BCG` iteration is also used in the smoothing step. The application of one multigride V-cycle has linear complexity. If the right-hand side has ns columns, one application of the multigrid preconditioner requires $O(ns \cdot N_V)$ algebraic operations.

Note that all basic operations in the conjugate gradient method and the multigrid preconditioner (e.g. the matrix-vector products or simpler vector operations) can easily be executed on CPUs or GPUs, respectively. Due to the preconditioning, the typical iteration numbers required for convergence of the `PBCG` algorithm are 10–15. Therefore, the total complexity of the solution of the forward and adjoint problems is $O(ns \cdot N_V)$ and $O(nd \cdot N_V)$, respectively.

3.6. Assembly of the sensitivity

For the assembly of the sensitivity matrix, we use the representation (9). As h is a piecewise constant function, the mass- $M(h)$ and stiffness-matrices $K(h)$ are just the element matrices M^T and K^T for the T-th finite element scaled by the derivatives of the optical parameters. Since the computations for the individual elements are independent from each other, this loop can be executed in parallel.



$R^1_{1,1}$ $R^1_{1,2}$ ··· $R^1_{4,4}$ $R^2_{1,1}$ ··· $R^2_{4,4}$ ···
$M^1_{1,1}$ $M^1_{1,2}$ ··· $M^1_{4,4}$ $M^2_{1,1}$ ··· $M^2_{4,4}$ ···
$K^1_{1,1}$ $K^1_{1,2}$ ··· $K^1_{4,4}$ $K^2_{1,1}$ ··· $K^2_{4,4}$ ···
ρ_1 ρ_1 ρ_1 ρ_2 ρ_2
μ_1 μ_1 μ_1 μ_2 μ_2
$\varkappa_1$ $\varkappa_1$ $\varkappa_1$ $\varkappa_2$ $\varkappa_2$
Σ
$H^1_{1,1}$ $H^1_{1,2}$ ··· $H^1_{4,4}$ $H^2_{1,1}$ ··· $H^2_{4,4}$ ···
LUT
A_1 A_2 ··· ··· ··· ··· A_{n-1} A_n

Fig. 2. Assembly of the system matrix: The 4×4 stiffness-, mass- and boundary mass-matrices K, M, and R of every element are scaled with the optical parameters κ, μ and ρ, summed and stored temporarily into a vector T. In a second step, these elements are compressed into a sparse matrix A by summing the elements specified by the vertex-to-element connectivity which is given as look-up table (LUT).

The pseudo-code for the GPU kernel used to assemble the sensitivity matrix is listed in Algorithm 2. The threads are started in blocks of dimension $(\mathbf{x},\mathbf{y},\mathbf{z}) = (4 \times 4 \times 32)$. The size of (4×4) is due to the size of the element-wise mass- and stiffness-matrices on a tetrahedral mesh. The last dimension is the number of elements which are processed in parallel and has been set to 32 to maximize the number of threads per block.

For readability, all data structures which exist once for every element being processed carry a superscript $\mathbf{z}$ instead of writing them as an array. Thus, $n^{\mathbf{z}}$ is actually an array with 32 entries, $d^{\mathbf{z}}[\mathbf{x}]$ is a 2D array with size (32×4) and $k^{\mathbf{z}}(\mathbf{x},\mathbf{y})$ is a 3D array with size $(32 \times 4 \times 4)$, for example.

The number of the element which is processed by a thread can be inferred from the index of the thread block as shown in line 1. In line 2, g^T denotes the element-to-vertex connectivity of the T-th finite element, which is cached into the vector d. The algorithm caches the element's mass- and stiffness-matrices (lines 4–6) and the forward and ajoint solutions (lines 9–10 and 12–13). Line 15 is an abbreviation for the computation of the sensitivity contribution according to Eq. (9). These contributions are summed in parallel (line 16) and the result is stored in S.

Assembling the sensitivity matrix requires $O(ns \cdot nd \cdot N_T)$ operations and the same amount of memory. Hence, this is the most time consuming and memory intensive step in the current algorithm. We would also like to note that the assembly of the sensitivity matrices S_k can be avoided completely if it is applied on-the-fly when computing the matrix-vector products; cf [19] and the remarks in section 4.6.

Algorithm 2 GPU kernel for assembling the sensitivity matrix

```
1:  $n^{\mathbf{z}} \leftarrow 32 \cdot \texttt{blockIdx} + \mathbf{z}$                  % compute element index
2:  $d^{\mathbf{z}}[\mathbf{x}] \leftarrow g^{n^{\mathbf{z}}}(\mathbf{x})$                   % cache connectivity
3:  synchronize
4:  $k^{\mathbf{z}}(\mathbf{x},\mathbf{y}) \leftarrow K^{n^{\mathbf{z}}}(\mathbf{x},\mathbf{y})$          % cache element matrices
5:  $m^{\mathbf{z}}(\mathbf{x},\mathbf{y}) \leftarrow M^{n^{\mathbf{z}}}(\mathbf{x},\mathbf{y})$
6:  $r^{\mathbf{z}}(\mathbf{x},\mathbf{y}) \leftarrow R^{n^{\mathbf{z}}}(\mathbf{x},\mathbf{y})$
7:  $p \leftarrow 0$
8:  for $k = 0$ to $ns - 1$ do
9:      $v_x^{\mathbf{z}}[\mathbf{x}] \leftarrow V_x(d^{\mathbf{z}}[\mathbf{x}], k)$             % cache forward fields
10:     $v_m^{\mathbf{z}}[\mathbf{x}] \leftarrow V_m(d^{\mathbf{z}}[\mathbf{x}], k)$
11:     for $l = 0$ to $nd - 1$ do
12:         $w_x^{\mathbf{z}}[\mathbf{y}] \leftarrow W_x(d^{\mathbf{z}}[\mathbf{y}], l)$         % cache adjoint fields
13:         $w_m^{\mathbf{z}}[\mathbf{y}] \leftarrow W_m(d^{\mathbf{z}}[\mathbf{y}], l)$
14:         synchronize
15:         $H^{\mathbf{z}}(\mathbf{x},\mathbf{y}) = f(k^{\mathbf{z}}(\mathbf{x},\mathbf{y}), m^{\mathbf{z}}(\mathbf{x},\mathbf{y}), r^{\mathbf{z}}(\mathbf{x},\mathbf{y}), v_x^{\mathbf{z}}[\mathbf{x}], v_m^{\mathbf{z}}[\mathbf{x}], w_x^{\mathbf{z}}[\mathbf{y}], w_m^{\mathbf{z}}[\mathbf{y}])$
16:         $t^{\mathbf{z}} \leftarrow \sum_{\mathbf{xy}} H^{\mathbf{z}}(\mathbf{x},\mathbf{y})$          % parallel reduce
17:         $S(p, n^{\mathbf{z}}) \leftarrow t^{\mathbf{z}}$                  % store element's sensitivity
18:         $p \leftarrow p + 1$
19:     end for
20: end for
```

3.7. *Solution of the Gauß-Newton system*

Formally, the Gauß-Newton matrix $S_k^* S_k + \alpha_k I$ is a dense $N_T \times N_T$ matrix, which would be expensive to compute and almost impossible to store. Since by construction $S_k^* S_k + \alpha_k I$ is symmetric and positive definite, the Gauß-Newton system can be solved efficiently by the method of conjugate gradients, which requires only the multiplication with S_k and S_k^*. In this manner, the complexity of multiplication with the Gauß-Newton matrix is given by $O(ns \cdot nd \cdot N_T)$ algebraic operations, whereas a multiplication with the pre-multiplied matrix $S_k^* S_k + \alpha_k I$ would be $O(N_T{}^2)$.

For the iterative solution of the Gauß-Newton systems we utilize hardware optimized dense matrix-vector kernels, i.e. *LAPACK-BLAS* [26] for the CPU and a custom implementation based on *AGILE* for the GPU version.

4. Results

4.1. *Model problem*

For our numerical tests, we considered a mouse phantom based on the Digimouse model [27]. The measurement setup consisted of $ns = 24$ sources and $nd = 24$ detectors, which were arranged on three rings around the homogeneous torso. Two spherical fluorophore inclusions with a diameter of 5 mm were placed at the height of the middle optode ring inside the body, and the aim of our test was to reconstruct these inclusions. The setup is depicted in Fig. 3.

For our simulations we used realistic optical tissue properties which were gathered from literature [28–31], cf Table 1. These parameters were kept constant during simulation and reconstruction. In practice, the estimation of those background parameters would require additional a-priori knowledge or measurements.

The data used in our numerical tests were generated on a finer mesh with 200 835 elements and additionally perturbed by 1 % Gaussian noise relative to the largest measurement datum.

Table 1. Optical Parameters Used for the Simulations

	μ_s' mm^{-1}	$\mu_{a,i}$ mm^{-1}	ε mm^{-1} M^{-1}	ρ
excitation	0.275	0.036	$8.35 \cdot 10^3$	0.2
emission	0.235	0.029	$2.81 \cdot 10^3$	0.2

For the finite element discretization, we used a hierarchy of three nested tetrahedral meshes, which were generated with the open source mesh generator NETGEN [32]. The three meshes consisted of 2720, 21 760, and 174 080 elements and 853, 5103, and 34 677 vertices, respectively.

4.2. *Choice of parameters*

The regularization parameters α_k and the stopping index `maxit` in the Gauß-Newton algorithm were chosen as follows: We conducted a series of numerical tests for ten similar phantoms, and determined a set of parameters α_k that worked reasonably well for all samples. These parameters were then used for the actual reconstruction. For the results presented here, a geometric sequence $\alpha_k = q\alpha_0$ with $\alpha_0 = 1$ and $q = 0.2$ was used. The Gauß-Newton iteration was stopped when the regularization parameter reached the pre-defined threshold $\alpha_{min} = 1 \times 10^{-5}$, which corresponds to `maxit=8` Newton iterations.

4.3. *Numerical results*

All computations were executed on an AMD Phenom 9950 processor. Both, the CPU code and the GPU host code (i.e. not the code running on the graphics hardware ifself) are single-threaded. The compute intensive operations (e.g. the matrix-vector multiplications and the assembly of the sensitivity matrix) were realized as compute kernels, which were implemented for both, CPU and GPU hardware. For our GPU tests, we used an NVIDIA GTX 480 graphics card and NVIDIA's CUDA toolkit [33] for programming. A typical result of the image reconstruction is shown in Fig. 4.

4.4. *Performance analysis*

In Table 2, we provide a detailed performance analysis of the reconstruction algorithm and compare the computation times of the CPU and GPU implementations. For both architectures we also compared the reconstruction with and without multigrid preconditioning. For all relevant tasks, the execution on the GPU results in a significant speed-up. An exception is the assembly of the system matrices on the coarsest level, where the number of elements is so small that the acceleration achieved on the graphics card is spoiled by the latency for starting the kernels on the GPU. However, the time spent for this part of the computation is negligible. The observed speed-up by factors of about 10 is in good agreement to the accelerations of sparse matrix-vector products [34]. (Note that the factor 10 is mainly due to the approximately 10 times larger memory bandwidth provided by GPUs).

4.5. *Accuracy*

Only single-precision arithmetic executes very fast on current commodity graphics hardware, i.e. use of double-precision operations results in increased memory requirements and a tremendous performance drop. In order to obtain insight into the errors introduced by using single-

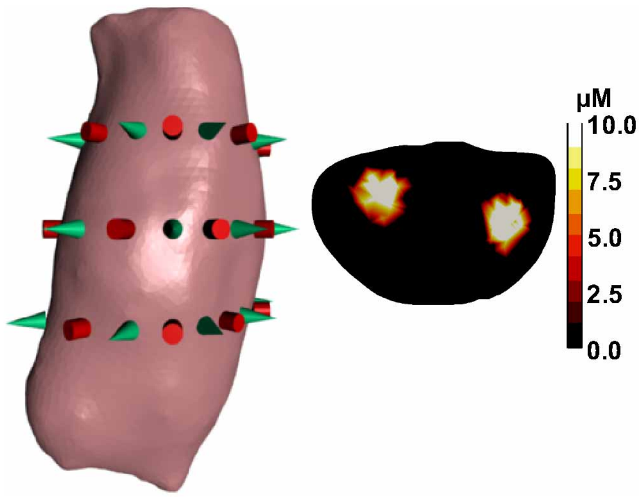


Fig. 3. Simulation setup showing the mouse phantom, the excitation sources (cylinders) and the photon-detectors (cones). The cross-section at the height of the central optode ring shows the assumed concentration distribution which consists of two 5 mm spheres with a fluorophore concentration of 10 μM.

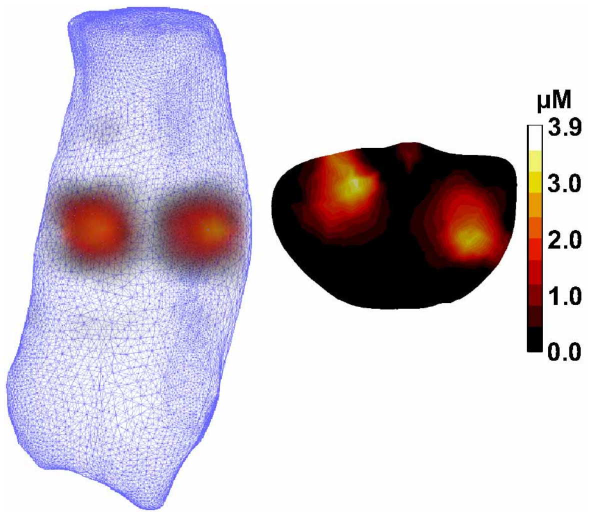


Fig. 4. Reconstruction of the two fluorescent inclusions shown in Fig. 3 performed with graphics hardware acceleration. The cross-section is again taken at the height of the middle optode ring. 1 % noise relative to the largest measurement datum was added for to the reconstruction.

Table 2. Comparison of Single-Threaded CPU and Parallelized GPU Reconstruction Times for the Digimouse Phantom with 174 080 Elements

Task	CPU		GPU		Speed-up
Assembly of the system matrices					
Compute element contributions					
Mesh level 1	n.a.[1]		0.04	ms	
Mesh level 2	n.a.[1]		0.10	ms	
Mesh level 3	n.a.[1]		0.66	ms	
Convert to CRS format					
Mesh level 1	n.a.[1]		0.72	ms	
Mesh level 2	n.a.[1]		2.06	ms	
Mesh level 3	n.a.[1]		13.17	ms	
Total					
Mesh level 1	0.63	ms	0.76	ms	0.83
Mesh level 2	6.37	ms	2.16	ms	2.95
Mesh level 3	121.97	ms	13.83	ms	8.82
Solution of the linear systems					
`PBCG` without multigrid					
Forward solution (V_x, V_m)	10.12	s	736.64	ms	13.73
Adjoint solution (W_x, W_m)	8.84	s	633.92	ms	13.94
`PBCG` with multigrid					
Forward solution (V_x, V_m)	5.45	s	461.60	ms	11.81
Adjoint solution (W_x, W_m)	6.02	s	508.29	ms	11.85
Computation of measurements					
$D^\top V_m$	496.40	ms	6.53	ms	76.01
Assembly of the sensitivity matrix					
`assembleSensitivity`	16.68	s	1.03	s	16.23
Solution of the Gauß-Newton system					
$(S_k^* S_k + \alpha_k I)^{-1}$	10.87	s	331.92	ms	32.75
Total reconstruction time					
Without multigrid	6.39	min	27.39	s	14.00
With multigrid	5.41	min	24.94	s	13.01

[1] On the CPU the system matrices are assembled in one step

precision operations, we conducted the tests in single-precision on CPU and GPU hardware, and also in double-precision on the CPU.

First, the outcome of the forward operator (i.e. the simulated measurement data) is compared for the true fluorescence distribution on the mesh with with 200 835 elements. The errors are related to the largest measurement datum such that the numerical error can be compared to the data noise easily. Given the single- and double-precision CPU and GPU measurement data $\mathscr{M}_{CS}$, $\mathscr{M}_{CD}$ and $\mathscr{M}_{GS}$ the discrepancies are

$$\frac{\|\mathscr{M}_{CD} - \mathscr{M}_{CS}\|}{\max(\mathscr{M}_{CD})} = 4.71 \times 10^{-5}$$

and

$$\frac{\|\mathscr{M}_{CD} - \mathscr{M}_{GS}\|}{\max(\mathscr{M}_{CD})} = 7.63 \times 10^{-7}.$$

As can be seen, the numerical error is far beyond the measurement noise of 1 % of the largest measurement datum. For practical applications, the "noise" will be determined mainly by the measurement setup (e.g. inaccuracies in the optode locations), model errors and the detector noise, whereas the additional numerical noise is negligible. As a consequence, the same regularization parameter (which has to be chosen depending on the noise level) can be used for both the CPU and GPU implementation.

To give evidence for the similarity of the resultant images, the relative errors based on the L^2 norm $\|c\|_{L^2}^2 := \int_\Omega c(x)^2 dx$ of the reconstructed concentrations are compared. Denoting the reconstructed fluorophore concentration from the single- and double-precision CPU and GPU implementation with c_{CS}, c_{CD} and c_{GS}, these relative errors are

$$\|c_{CD} - c_{CS}\| / \|c_{CD}\| = 6.94 \times 10^{-3} \qquad \text{and}$$
$$\|c_{CD} - c_{GS}\| / \|c_{CD}\| = 2.39 \times 10^{-3}.$$

The resultant images differ by less than 1 % which we consider to be sufficient for practical applications.

4.6. Memory requirements

In Table 3, we list the memory requirements for the essential data structures on the finest finite element mesh. The additional memory required for the coarse level matrices required for the multigrid preconditioner is negligible.

Table 3. Estimated and True Memory Required (in Mega Bytes; MB) for Selected Data Structures of the Reconstruction Algorithm with Single-Precision Floating-Point Numbers

Data structure	Estimate	MB
Local element matrices K, M, R	$3 \cdot (4 \cdot 4) \times N_T$	31.88
Assembled matrices A_x, A_m, G	depending on connectivity	15.31
Source vectors Q	$ns \times N_V$	3.17
Detector vectors D	$nd \times N_V$	3.17
Forward solutions V_x, V_m	$2 \cdot ns \times N_V$	6.35
Adjoint solutions W_x, W_m	$2 \cdot nd \times N_V$	6.35
Sensitivity S	$(nd \cdot ns) \times N_T$	382.50
Total		448.73

In our implementation, the sensitivity matrix requires most of the GPU memory. To avoid excessive memory consumption, the product of the sensitivity matrix with a vector can also be computed on-the-fly, without the need to store the whole sensitivity. Also the complexity of the multiplication with the sensitivity (now $O(ns \cdot nd \cdot N_T)$) can be reduced, i.e. it has been demonstrated in [19] that the application of the sensitivity can be realized in $O(ns \cdot N_V)$ operations. Such an approach, however, requires the solution of the adjoint problems for every multiplication with the sensitivity matrix. Comparing with the computation times in Table 2, this would drastically slow down the computations in practice, at least for the values of ns, nd, and N_T, N_V used in our test study. If the number of sources or detectors becomes very large (e.g. if a camera is used as detector), such an alternative might become favorable.

5. Discussion

In this article, we considered the image reconstruction in fluorescence tomography by iterative regularization methods of Newton-type and a discretization by finite element methods. We demonstrated that a combination of state of the art numerical methods, the utilization of modern hardware, and a hardware optimized implementation allows to solve this large-scale nonlinear inverse problem within a few seconds on standard desktop PCs, without compromising accuracy. In particular, the extensive use of the GPU's computing power resulted in a significant speed-up of the reconstruction process by factors of approximately 10–15. Such a degree of acceleration is in good agreement to other work [34, 35].

The use of unstructured meshes for our finite element simulations provides high flexibility with respect to geometry and facilitates the mesh generation process. A further acceleration of the computations could be achieved, however, by utilizing structured grids, which lead to structured sparsity patterns of the system matrices and allow faster access of global memory, cf [36].

In contrast to previous work [4, 5], we have presented algorithms which are fully iterative in nature and do not require matrix inversion libraries such as CULA, for example. Additionally, we ported the whole reconstruction algorithm to the graphics card and use the CPU only as controller which avoids expensive memory transfer between the host and the GPU.

The pre-computation of the mass- and stiffness-matrices of the individual elements and the creation of prolongation and restriction matrices for the mesh hierarchy are the only tasks which are still executed on the CPU. However, these setup steps have to be computed only once and could in principle also be stored along with the mesh data.

To achieve efficient code on graphics hardware, care has to be taken when accessing global memory. Memory access is most efficient, if a sequence of threads reads or writes to a sequential memory area (*coalesced* memory access) [33]. Therefore, the photon field matrices V_x, V_m and W_x, W_m, but also the associated source and detector matrices Q and D are stored in row-major order. As one column of these matrices belongs to one photon field, this means that the elements of a given photon field are interleaved. In combination with the blocked conjugate gradient method, internal performance measurements have shown that this memory layout provides speed-ups by a factor of roughly 2–3 on CPUs and up to 10 on GPUs.

The lack of fast double-precision floating point operations on current graphics hardware limits the accuracy of the reconstructions to some extent. In our numerical examples, the relative errors in the forward simulation due to the single-precision arithmetic was less then 1% for both the CPU and GPU reconstructions. Thus, the numerical errors can be expected to be dominated by measurement noise and model errors and single-precision arithmetic may well suffice for this kind of (severely ill-posed) inverse problems.

The choice of an appropriate sequence of regularization parameter α_k crucially determines the performance of the iterative reconstruction algorithm. Following the analysis of the iteratively regularized Gauß-Newton method [12,15], α_k should decay exponentially, e.g. $\alpha_k = \alpha_0 q^k$ for some $0 < q < 1$. The choice of α_0 and q, however, depends on the problem. While in principle, appropriate values can be found by a careful analysis of the problem under investigation, a more practical way to tune these parameters is to perform a sequence of reconstructions for known solutions and select regularization parameter that work well for all cases in this training set. The same set of parameters can then be used for similar problems.

The choice of the stopping index `maxit` is another crucial ingredient for the numerical algorithm. If α_0 and q have been fixed, and $\alpha_k = \alpha_0 q^k$, then this amounts to choosing a minimal regularization parameter $\alpha_{min} := \alpha_{\texttt{maxit}}$. A rigorous way to choose α_{min} (respectively `maxit`) is provided by Morozov's discrepancy principle [37, 38], i.e. the reconstruction algorithm is stopped, as soon as $\|\mathscr{M}(c_k) - \mathscr{M}^{\delta}\| \sim \delta$, where δ is a measure for the measurement and model errors. Another rigorous way would be to choose $\alpha \sim \delta$. Both choices however require the knowledge of the level δ of the perturbations, which is usually not known in practice. A heuristic way to choose α_{min} without knowledge of δ is provided by the L-curve method [38,39] or generalized cross-validation [40]. If a series of similar problems is considered, then `maxit` (or α_{min}) can again be obtained by "training" the algorithm on a small data set with known solutions. This strategy is probably the most practical one and also the method we utilize in our experiments.

In our opinion, the fast and reliable image reconstruction is a core requirement for the acceptance of new imaging modalities like fluorescence tomography in pre-clinical and also engineering applications. GPU-accelerated algorithms can result in reconstruction times in the order of a few seconds, which allows the inspection of the results on-the-fly and, thus, certainly facilitates the adoption of this technique in a non-research environment.

Acknowledgments

This work is supported by the project F3207-N18 granted by the Austrian Science Fund (FWF).