Journal of Biomedical Optics

SPIEDigitalLibrary.org/jbo

The Toast++ software suite for forward and inverse modeling in optical tomography

Martin Schweiger
Simon Arridge

SPIE

Journal of Biomedical Optics 19(4), 040801 (April 2014)

TUTORIAL

The Toast++ software suite for forward and inverse modeling in optical tomography

Martin Schweiger* **and Simon Arridge**
University College London, Department of Computer Science, Gower Street, London WC1E 6BT, United Kingdom

Abstract. We present the Toast++ open-source software environment for solving the forward and inverse problems in diffuse optical tomography (DOT). The software suite consists of a set of libraries to simulate near-infrared light propagation in highly scattering media with complex boundaries and heterogeneous internal parameter distribution, based on a finite-element solver. Steady-state, time- and frequency-domain data acquisition systems can be modeled. The forward solver is implemented in C++ and supports performance acceleration with parallelization for shared and distributed memory architectures, as well as graphics processing computation. Building on the numerical forward solver, Toast++ contains model-based iterative inverse solvers for reconstructing the volume distribution of absorption and scattering parameters from boundary measurements of light transmission. A range of regularization methods are provided, including the possibility of incorporating prior knowledge of internal structure. The user can link to the Toast++ libraries either directly to compile application programs for DOT, or make use of the included MATLAB and PYTHON bindings to generate script-based solutions. This approach allows rapid prototyping and provides a rich toolset in both environments for debugging, testing, and visualization. [DOI: 10.1117/1.JBO.19.4.040801]

Keywords: diffuse optical tomography; image reconstruction software; computer program; light transport simulation; Toast++.

Paper 130890TR received Dec. 17, 2013; revised manuscript received Mar. 4, 2014; accepted for publication Mar. 10, 2014; published online Apr. 29, 2014.

1 Introduction

Diffuse optical tomography (DOT)[1–4] is a medical imaging modality for measuring and visualizing the distribution of absorption and scattering properties in an organ such as the brain. The optical parameters are related to physiological markers, such as blood oxygenation and tissue metabolism, making DOT a functional imaging tool. Applications include brain activation studies[5] and breast tumor detection.[6]

Data acquisition systems consist of a light source that delivers near-infrared light to the body surface at different points or with different spatial patterns, and a detector system that measures the light transmitted through the tissue and emitted from the boundary. Light sources include steady-state systems, time-resolved and frequency-domain systems, providing measurements of intensity, temporal dispersion or phase shift, and modulation amplitude, respectively.

Biological tissues are highly scattering in the near-infrared wavelength range, and the detected photons have undergone multiple scattering events. Monte Carlo models of light transport in tissue build intensity distributions by computing individual photon paths as a random walk model, given the parameters of single-scattering events. By comparion, methods based on the Boltzmann equation treat light as a density wave propagating through the tissue. Computational methods solve the arising partial differential equations numerically by discretizing the domain and parameter distributions; for example, by the use of finite-difference (FDM), finite-element (FEM), or boundary element (BEM) methods.

*Address all correspondence to: Martin Schweiger, E-mail: M.Schweiger@ucl.ac.uk

In this paper, we describe the Toast++ software, an efficient open-source toolbox developed for modeling and reconstruction in DOT. Toast++ is a collection of libraries for sparse matrix algebra, finite-element analysis and nonlinear inverse problem solution that can be linked directly into the application programs. It allows researchers to rapidly construct analysis software without the need to develop the underlying low-level sparse matrix and finite-element subsystems. Toast++ contains parallel matrix assembly and solver tools that provide scalability for distributed and shared memory architectures[7] as well as graphics-processor platforms.[8]

The Toast++ software suite includes bindings for MATLAB and PYTHON, which exposes the functionality of the low-level Toast solver engine to both of these scripting environments. This method allows the rapid delvelopment of script-based application code, combining the high performance of the Toast libraries with the rich toolsets of MATLAB or PYTHON, including linear solvers and visualization tools. This paper contains a number of code and script examples for various aspects of the software and user interface.

While Toast++ was originally developed for modeling and reconstruction in DOT, its modular design and extensible programming interface allows it to be adapted to other medical imaging problems and applications in different fields, such as linear elasticity models for soft tissue deformation.[9]

2 Background

Toast was originally developed as a closed-source C++ finit-element-based light transport and inverse reconstruction software tool in time-resolved optical tomography. The software was later extended for frequency-domain problems. Scalable matrix assembly for shared and distributed memory architectures

using threads and message passing interface (MPI), respectively, was added later. Most recently, the matrix library was extended to include a graphics-accelerated version, using Compute Unified Device Architecture (CUDA) for execution on NVidia graphics hardware.

Toast was initially distributed as a set of command line utilities for light transport simulation and image reconstruction. A significant improvement in usability was provided by adding a MATLAB interface that exposed the Toast functionality as a toolbox containing a collection of MATLAB functions for mesh manipulation, matrix assembly, and solution. Scripts for forward and inverse solver examples for time- and frequency-domain problems were included. Recently, the MATLAB interface has been revised to make use of object-oriented MATLAB programming paradigms.

In addition, an alternative PYTHON module interface has been included in the Toast suite. It makes use of data structures and linear solvers provided by the `numpy` and `scipy` modules, and it also provides similar functionality to the MATLAB toolbox.

Toast has now been released as an open-source software distribution, allowing users to access and extend the low-level library code.

3 Computational Methods

Image reconstruction in optical tomography is an ill-posed nonlinear inverse problem. Toast uses a model-based optimization approach, which iteratively minimizes an objective function defined as a norm of the difference between model and measurement data. In the following sections, we describe the components of the forward and inverse solvers used by the Toast software.

3.1 *Forward Modeling*

The forward model in DOT describes the propagation of near-infrared light through biological tissues from a distribution of light sources illuminating the surface, to a collection of detectors measuring the light exiting the tissue. The sources can either consist of optical fibers delivering light to a small skin area, or of devices that illuminate a larger area of the surface possibly using structured light patterns.[10] Data acquisition systems may employ continuous sources (continuous wave imaging), periodic at MHz to GHz frequencies (frequency-domain imaging) or pulsed (time-domain imaging). At the wavelength used in DOT, tissue is highly scattering, and photon transport can often be described as a diffusion process. For the frequency domain case, the forward model is then given by

$$-\nabla\cdot\kappa(\mathbf{r})\nabla\phi(\mathbf{r},\omega)+\left[\mu_a(\mathbf{r})+\frac{i\omega}{c}\right]\phi(\mathbf{r},\omega)=0,\quad \mathbf{r}\in\Omega \tag{1}$$

with boundary condition[11]

$$\phi(\mathbf{m},\omega)+2\zeta(c)\kappa(\mathbf{m})\frac{\partial\phi(\mathbf{m},\omega)}{\partial\nu}=q(\mathbf{m},\omega),\quad \mathbf{m}\in\partial\Omega, \tag{2}$$

where q is a source distribution on boundary $\partial\Omega$ of domain Ω, modulated at frequency ω. Field ϕ represents the complex-valued photon density distribution. The model parameters are absorption coefficient μ_a, diffusion coefficient $\kappa=[3(\mu_a+\mu_s)]^{-1}$ with (reduced) scattering coefficient μ_s, and speed of light in the medium c. ζ is a term incorporating the refractive index mismatch at the boundary, $\partial\nu$ is the outward boundary normal, and i is the imaginary unit. The measurable quantity is the normal current across the boundary,

$$J_n(\mathbf{m},\omega)=-c\kappa(\mathbf{m})\frac{\partial\phi(\mathbf{m},\omega)}{\partial\nu}. \tag{3}$$

Analytic solutions of Eqs. (1) and (2) only exist for simple geometries. For practical applications, a numerical model must be employed that can incorporate inhomogeneous parameter distributions and complex boundaries. The Toast++ package uses a Galerkin FEM to implement the DOT forward problem.

3.2 *Time-Domain Modeling*

As an alternative to measuring a modulated signal, time-domain data acquisition systems use short pulses of light $Q(\mathbf{r},t)$ in the picosecond range for sources, and measure the temporal dispersion of the transilluminated signal at the detector sites. The light transport model in this case is the time-dependent diffusion equation,

$$-\nabla\cdot\kappa(\mathbf{r})\nabla\phi(\mathbf{r},t)+\left[\mu_a(\mathbf{r})+\frac{\partial}{\partial t}\right]\phi(\mathbf{r},t)=Q(\mathbf{r},t),\quad \mathbf{r}\in\Omega. \tag{4}$$

Numerically, the time derivative term can be approximated by a finite-difference scheme. The Toast++ software allows the simulation of the time-dependent measurement signals, using either implicit or explicit schemes for propagation in time. In addition, Toast++ allows the direct computation of the temporal moments, as well as the Laplace transform and its moments (the Mellin–Laplace transform) of the measurement signal, without the need for a full computation of the temporal profile.[12,13]

3.3 *Inverse Problem Solver*

The Toast++ suite contains the building blocks to construct an iterative inverse solver for reconstructing the distribution of model parameters $\mathbf{x}=\{\mu_a,\kappa\}$ or $\mathbf{x}=\{\mu_a,\mu_s\}$ from boundary measurements $\mathbf{y}=\langle M,J\rangle_{\partial\Omega}$ for some measurement profile M. The inverse problem is defined by a regularized least-squares approach, where an objective function Ψ is minimized,[1,14] given by

$$\Psi(\mathbf{x})=\frac{1}{2}\sum_i[y_i-f_i(\mathbf{x})]^2+\tau\mathcal{R}(\mathbf{x}) \tag{5}$$

with forward model f given by Eqs. (1)–(3), regularization functional $\mathcal{R}$ and hyperparameter τ.

Toast++ provides a range of regularization options, including Tikhonov and total variation (TV) methods, to enforce different smoothness conditions in the solution. In addition, spatially varying regularization parameters can be applied to take into account prior knowledge about the internal structure of the target.

Toast++ provides functions for constructing the direct and adjoint fields for a given set of source and detector locations, and it allows the solution of the adjoint problem either by explicitly constructing the Jacobian and Hessian matrices of the forward operator, or by an implicit, matrix-free approach. Using this functionality, gradient-based inverse solvers can be constructed by making use of first derivatives, such as nonlinear

conjugate gradients (NCG),[15] or second derivatives such as Gauss-Newton (GN) or Levenberg–Marquardt approaches.[14]

4 Program Description

Toast++ is designed as a hierarchical set of core libraries written in C++ for sparse matrix computation (`libmath`), finite-element computation (`libfe`), and iterative parameter reconstruction (`libtoast`). Sample command-line application programs for diffuse light transport modeling (`fwdfem`) and image reconstruction (`supertoast`) linking to the libraries are included, as well as tools for visualizing the simulated measurements and volume reconstructions.

In addition to the C++ interface, the libraries can also be accessed via bindings for the MATLAB and PYTHON scripting environments, allowing application code to be written as high-level scripts. Embedding Toast in MATLAB or PYTHON allows the use of pre-existing tools, such as preconditioned parallel solvers or visualization tools without additional effort. The ability for runtime debugging and inspecting intermediate results aids in rapid prototyping of new application code.

A schematic overview of the building blocks of the Toast suite is shown in Fig. 1. In the following sections, we describe the components in more detail.

4.1 *Matrix Library*

The Toast++ matrix library (`libmath`) contains a class hierarchy for sparse and dense matrix and vector types, as shown in Fig. 2. It exposes a common matrix interface via the abstract `Matrix` class, which hides the implementation details of specialized matrix subtypes such as dense, compressed sparse row (CSR), and coordinate-indexed matrix classes, from the application layer. The matrix library is templated and can be instantiated for real and complex-valued data types.

Public methods include element, row, and column access via iterators, matrix-vector products, as well as direct and iterative linear solvers including conjugate gradients (CG), generalized

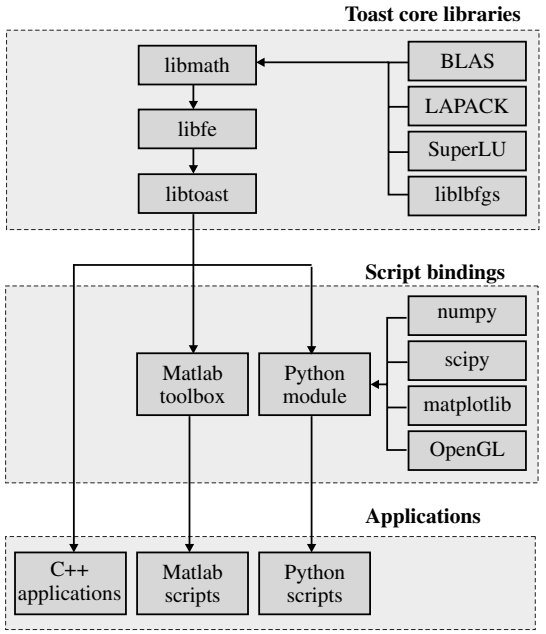

Fig. 1 Toast core library, script bindings, and application layer layout.

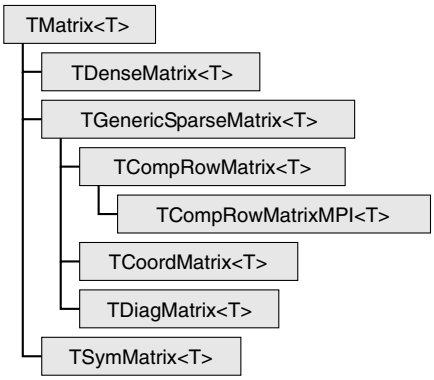

Fig. 2 Matrix class hierarchy in the Toast `libmath` library.

minimum residual methods, LU, and Cholesky decompositions. Many of the methods are implemented as wrapper functions to external libraries, such as the SuperLU library for LU solution or preconditioning of sparse matrix classes.

The MATHLIB library also contains a distributed sparse matrix class (`TCompRowMatrixMPI`) using the MPI protocol, which allows the matrix data to be distributed over multiple processor nodes, while maintaining the common matrix interface, and thus being transparent to the application programmer. This matrix class can be used to represent the linear system of the finite-element model in conjunction with a partitioned distributed mesh class, as discussed in the next section.

4.2 *Finite-Element Library*

Using an N-dimensional polynomial basis expansion with local support $u_i(\mathbf{r})$, $i = 1 \ldots N$ over domain Ω, a piecewise polynomial approximation ϕ^h of field ϕ can be expressed as

$$\phi^h(\mathbf{r},\omega) = \sum_i \Phi_i(\omega) u_i(\mathbf{r}), \tag{6}$$

which is defined by the vector $\Phi \in \mathbb{C}^N$ of nodal coefficients. Applying the weak formulation of Eq. (1) and integration by parts[16] leads to the linear system

$$[\mathsf{K}(\kappa^h) + \mathsf{C}(\mu_a^h) + \zeta\mathsf{A} + i\omega\mathsf{B}]\Phi(\omega) = Q(\omega), \tag{7}$$

where K, C, A and B are sparse symmetic positive definite system matrices. The assembly of each matrix is performed on a per-element basis, followed by a mapping from the local to global degree of freedom (DOF). The individual integrals over an element $\Omega_i^{(\mathrm{el})}$ are given by[14]

$$\begin{aligned}
\mathsf{K}_{ij}^{(\mathrm{el})} &= \sum_{k \in S[\Omega^{(el)}]} \kappa_k \int_{\Omega^{(el)}} u_k(\mathbf{r}) \nabla u_j(\mathbf{r}) \cdot \nabla u_i(\mathbf{r}) \mathrm{d}\mathbf{r} \\
\mathsf{C}_{ij}^{(\mathrm{el})} &= \sum_{k \in S(\Omega^{(\mathrm{el})})} (\mu_a)_k \int_{\Omega^{(el)}} u_k(\mathbf{r}) u_i(\mathbf{r}) u_j(\mathbf{r}) \mathrm{d}\mathbf{r} \\
\mathsf{B}_{ij}^{(\mathrm{el})} &= \frac{1}{c} \int_{\Omega^{(\mathrm{el})}} u_i(\mathbf{r}) u_j(\mathbf{r}) \mathrm{d}\mathbf{r} \\
\mathsf{A}_{ij}^{(\mathrm{el})} &= \int_{\partial\Omega^{(\mathrm{el})}} u_i(\mathbf{m}) u_j(\mathbf{m}) \mathrm{d}\mathbf{m} \\
\mathbf{Q}_i^{(\mathrm{el})} &= \int_{\Omega^{(\mathrm{el})}} u_i(\mathbf{r}) q_0(\mathbf{r},\omega) \mathrm{d}\mathbf{r}.
\end{aligned} \tag{8}$$

The realization of the integrals in Eq. (8) depends on the type of element and the shape functions used. In some cases, such as triangles or tetrahedra with polynomial basis functions, analytic integration formulae may be available. Otherwise, numerical quadrature rules must be employed.

The Toast++ finite-element subsystem (`libfe`) contains the mesh and element classes for defining a numerical representation of the forward problem. The `Mesh` class represents the discretization of the computation domain and it is a container for a list of node coordinates and a list of elements. It provides functions for returning the sparsity pattern of the system matrices for the linear FEM system, and for populating the matrices by performing the appropriate integrals over elements and mapping the results from the local element matrices to the global system matrices. Toast also contains methods for node reordering, e.g., for bandwidth minimization or minimizing the fill-in of the system matrix decomposition for direct solvers and preconditioners.

Toast++ provides support for different element types and shape functions in two and three dimensions. In 2-D, 3-, 6- and 10-noded subparametric and isoparametric triangles are available, providing linear, quadratic and cubic polynomial shape functions, respectively. In addition, Toast provides 4-noded rectangular elements for representing structured meshes. In 3-D, Toast offers 4- and 10-noded sub- and isoparametric tetrahedra, as well as wedge and regular voxel elements. The element class hierarchy is shown in Fig. 3. The element types are represented in a hierarchical class structure, and expose a common interface via the abstract `Element` base class. This includes methods for mapping from local to global coordinates, Jacobian computation, as well as integrals of shape functions, shape function derivatives and combinations thereof, over the element volume.

Elements use analytic integration rules where available, including subparametric triangle and tetrahedron elements, and resort to numerical quadrature rules for isoparametric elements with curved surfaces. The implementation details of the element integrals and other methods are hidden from the application layer by use of polymorphism. It is, therefore, possible to write applications independent of element type, and to define meshes with mixed elements.

Toast++ also contains a distributed mesh class (`MeshMPI`) which can be used to automatically partition a mesh and distribute it over multiple processor nodes. Toast provides the possibility to link to the Zoltan partitioning library[17] for partitioning and node migration. In combination with Toast's distributed sparse matrix support, the linear FEM system can be assembled and solved transparently to the application layer. This allows the deployment of Toast for large-scale problems on computer clusters without additional coding effort from the user.

Different options are available for the choice of source distributions Q. Toast supports point sources and extended source profiles with a Gaussian or cosine shape to simulate optical fiber-based light delivery systems.[18] In addition, Toast also allows modeling of illumination of large areas of the surface with shaped light patterns.[10] Source patterns are projected onto the surface of a mesh of arbitrary shape by using an instance of one of Toast's projector classes, which support either pinhole or orthographic projection. The projectors are implemented with the use of OpenGL routines via the Mesa-3-D library which can also make use of graphics hardware acceleration. MATLAB bindings for the projector functions are supported in the Toast toolbox.

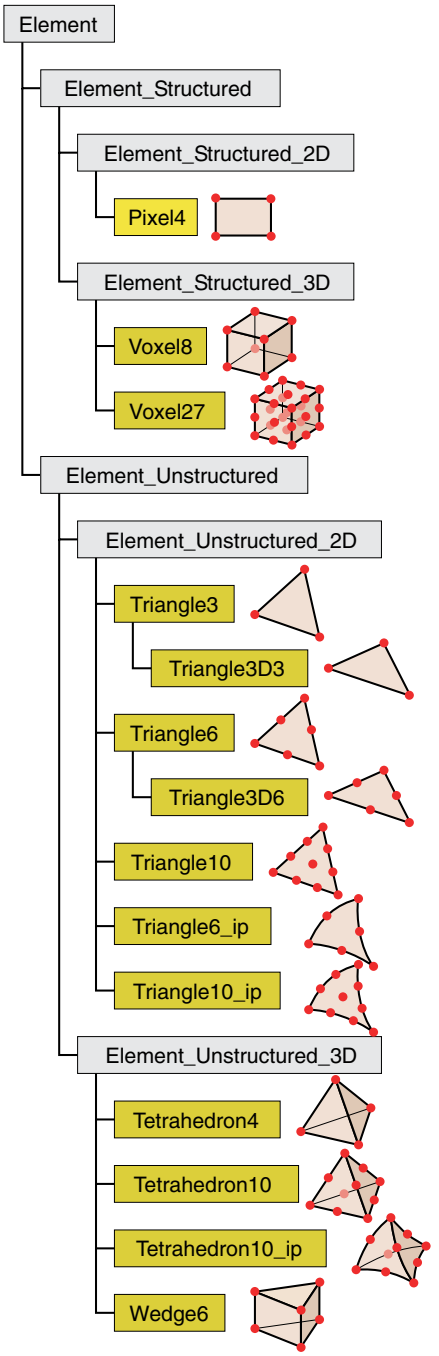

Fig. 3 Toast element class hierarchy. Abstract classes are shown in gray, nonabstract classes in yellow together with graphical shape representations.

The same projectors can be used to map a distribution of transilluminated light from the surface to an imaging system such as a charge coupled device (CCD) camera. Figure 4 shows an example of a source pattern projected onto a cylindrical surface, and the resulting surface exitance on the opposite side of the cylinder mantle projected back onto a CCD camera. The use of shaped light illumination and CCD detectors can reduce measurement times significantly.

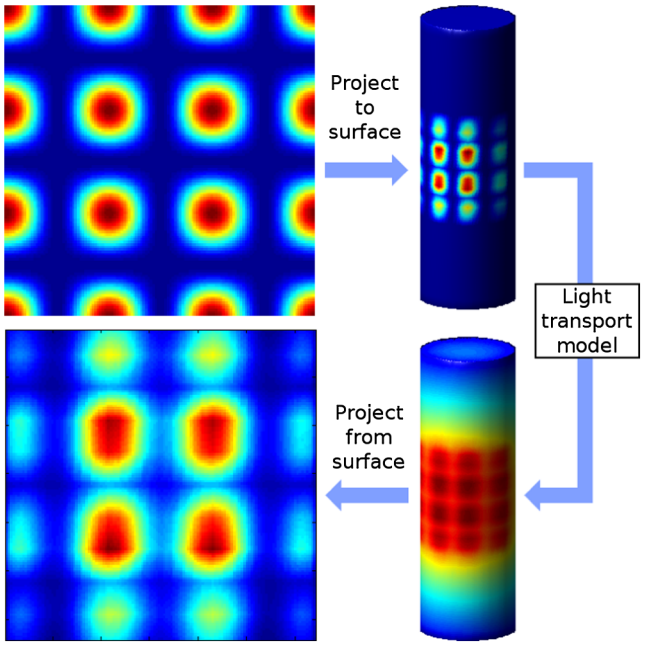

Fig. 4 Example of a source pattern (top left) projected on a cylinder with Toast's pinhole projector operator (top right), and resulting surface exitance (bottom right) projected back into a CCD detector array positioned opposite the source projector (bottom left).

4.3 *Alternative Numerical Modeling Schemes*

In addition to the FEM scheme described above, Toast also supports alternative numerical subsystems for simulating light propagation in scattering media. The software package includes a discontinuous Galerkin (DG) discretization scheme,[19] which can be used to accurately simulate discontinuities in the internal optical parameter distributions, including the refractive index. In addition, meshes for DG computation can be refined more easily than those used in standard FEM methods. DG, therefore, supports efficient adaptive refinement methods, e.g., as a function of the gradient of the evolving parameter distribution during a reconstruction.

Toast also contains a modeling module using the boundary element method (BEM),[20] which allows efficient simulation of piecewise constant parameter distributions. This scheme has been applied to shape reconstruction of internal boundaries of regions with piecewise constant optical parameter values. BEM can also be used to reduce the complexity of reconstruction by restricting the solution on interface layers, such as the cortical surface.[21] Toast has been used to combine FEM and BEM models for DOT reconstruction in layered media, where superficial tissue layers are modeled with BEM, while the region of interest is reconstructed with a spatially resolved FEM approach.[22]

4.4 *DOT Inverse Solver Library*

Low-level support for building an inverse solver for parameter estimation in DOT is provided by the `libstoast` library. The library contains a class hierarchy of basis mapping operators that allow a field or parameter distribution on Ω to be mapped from the mesh basis to an independent basis used for the reconstruction. Given the mesh basis $\{u_i(\mathbf{r})\}$, $i = 1 \ldots N$, and inverse solution basis $\{v_j(\mathbf{r})\}$, $j = 1 \ldots M$, the mapping

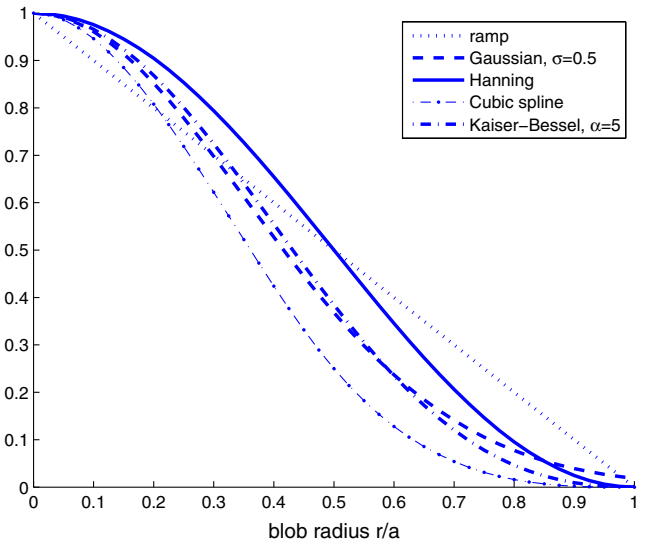

Fig. 5 Radial profiles of blob basis types supported by Toast.

from nodal coefficients $\Phi_i^{(u)}$ to inverse basis coefficients $\Phi_j^{(v)}$ is given by[23]

$$\Phi_j^{(v)} = \sum_{i=1}^{N} \Phi_i^{(u)} \int_\Omega u_i(\mathbf{r}) v_j(\mathbf{r}) \mathrm{d}\mathbf{r}. \tag{9}$$

Toast supports a range of inverse basis functions v, including linear and cubic voxel bases, as well as blob bases with radially symmetric basis functions of different shape, such as truncated Gaussian and Kaiser–Bessel window functions.[23] Figure 5 shows examples of the radial profiles of the supported blob basis functions.

The separation of forward and inverse basis allows the reconstruction to be tailored to the required performance and convergence criteria, while conserving the fidelity of the forward solver. For example, it is then possible to adaptively refine the mesh without also affecting the reconstruction component.

`libstoast` contains support for defining the parameter estimation problem by minimizing the regularized least-squares cost functional Eq. (5). Methods for computing direct and adjoint fields are provided by linking to the forward solver module. Computation of the Jacobian matrix of the forward operator for gradient-based methods is supported as well as matrix-free methods. MATLAB and PYTHON bindings for the inverse solver utilities are provided, allowing the iterative optimization loop to be defined in application space directly by the user.

4.5 *Support for Parallel Computation*

Toast++ can be used on different parallel architectures for performance improvement. It contains suppport for shared memory systems by using threaded solutions for the available Krylov solvers, using a master-worker structure to distribute multiple right-hand sides of the linear FEM problem over multiple threads.[7] This high-level parallelization strategy is easy to implement and has little computational overhead, resulting in good scalability, in particular where the FEM problem is to be solved iteratively for a large number of right-hand sides, as is typically the case in DOT. When compiled with thread support, parallel versions of time-critical routines, such as forward solution and gradient computation, are also available from the

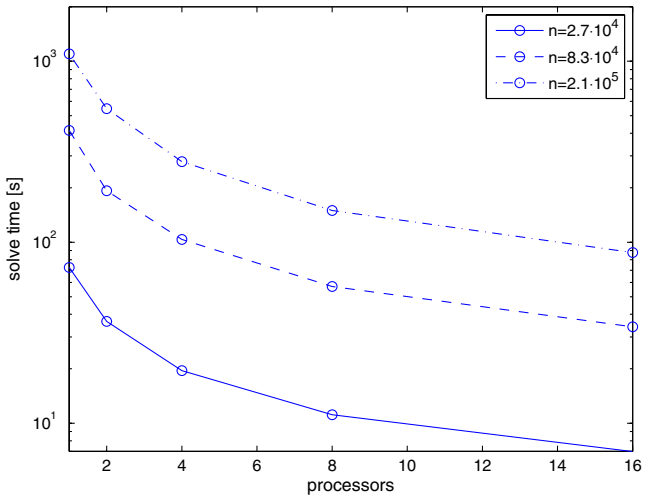

Fig. 6 Timing comparsions for FEM forward computations of 64 source positions and three different mesh resolutions using a BiCGSTAB iterative solver with thread support.

MATLAB and PYTHON interfaces. Figure 6 shows timings for FEM forward solution with a BiCGSTAB iterative linear solver (tolerance 10^{-14}) on cylindrical meshes of different resolution, using a threaded implementation with 1, 2, 4, 8, and 16 threads on a Xeon E5-2665 workstation with 16 2.4 GHz processors. When using the MATLAB and PYTHON interfaces, no additional setup code is required to make use of the multithreaded Toast function support. The number of processors can be adjusted manually, e.g., in MATLAB with

```
toastThreadCount(n);
```

A graphics processor hardware accelerated version of the FEM forward solver for Toast has been developed using the CUDA framework for NVidia graphics processors, and the CUSP library[24] for sparse linear system computation on the graphics processor. The Toast CUDA implementation moves the solution of the linear system to the graphics hardware after the sparse system matrices and right-hand sides have been assembled. This approach is particularly efficient for the iterative solution of the time-domain problem, because the stiffness and mass matrices in Eq. (8) remain unchanged and can be reused for each iteration after having been copied once to the device memory. We have reported speed improvements of a factor of up to 8 for a frequency-domain solution, and up to 13 for a time-domain solution on an NVidia GTX 285 GPU, compared to a single-threaded CPU solution on an Intel Xeon processor clocked at 2.0 GHz with 4 MB cache and 12 GB main memory.[8]

4.6 *MATLAB Bindings*

The functionality of the Toast core libraries is exposed to the MATLAB scripting environment as a toolbox containing a collection of compiled mex files and utility scripts that are available to MATLAB as callable functions. The MATLAB interface to Toast makes use of an object-oriented approach, by representing meshes, elements, solvers, etc., as MATLAB objects. Each object has properties and methods that closely resemble the class definitions of the underlying C++ code. As an example, a subset of methods of the `toastMesh` class is shown in Fig. 7. The interface consists of a single mex file that handles the communication between the MATLAB Toast toolbox and

ToastMesh methods	
Make()	% create a mesh from geometry data
Read()	% read mesh definition from file
Write()	% write a mesh definition to file
Element()	% returns a mesh element
NodeCount()	% returns the number of nodes
ElementCount()	% returns the number of elements
Dimension()	% returns the mesh dimension
BoundingBox()	% returns the mesh bounding box
Size()	% returns the mesh volume
Data()	% returns mesh geometry data
SurfaceData()	% returns the mesh surface geometry
ElementSize()	% returns a list of element volumes
FindElement()	% identify element containing a point
Elmat()	% assemble an element matrix
SysmatComponent()	% assemble a term of the global system matrix
Massmat()	% assemble a mass matrix
SetQM()	% define a source/detector set
ReadQM()	% read a source/detector set from file
WriteQM()	% write a source/detector set to file
DataLinkList()	% returns permutation index for a data vector
Qpos()	% returns a list of source positions
Mpos()	% returns a list of detector positions
Qvec()	% returns an array of right-hand sides
Mvec()	% array of boundary projection operators
Display()	% display a mesh and optional surface field

Fig. 7 Methods in the ToastMesh class of the Toast MATLAB toolbox.

the C++ libraries. For example, the ToastMesh class contains a handle to a C++ mesh object, and a series of methods for computing mesh geometry and transformations, matrix assembly, extracting element data, surface integrals, and so on. Each of these methods is implemented by calling the mex file to pass a request to the C++ Toast libraries. The libraries process the request, and pass the results back to the MATLAB method. In this way, the efficiency of the Toast library is maintained, while allowing the user the convenience of the MATLAB script interface.

4.7 *PYTHON Bindings*

Similar to the MATLAB toolbox, the PYTHON script interface is provided by compiled PYTHON modules that can be imported by a PYTHON script. The functionality of the MATLAB and PYTHON interfaces is similar. Sharing of the underlying core libraries minimizes code duplication and improves stability and simplifies the process of code generation. Toast functionality is available in PYTHON via a plugin module that links to the Toast core libraries. Toast makes use of the `numpy` and `scipy` modules for the sparse matrix data structures and linear solvers. The interface is similar to the MATLAB Toast toolbox, with methods provided for constructing or reading meshes, defining the FEM problem, assembling, and solving the linear system. An additional advantage of the MATLAB and PYTHON interfaces is the availability of additional functionalities, such as sparse matrix solvers and visualization tools.

The Toast toolbox functions in both interfaces can be used as building blocks for creating scripts for solving both the forward and inverse problems. Section 5 contains script examples for different types of problems.

4.8 *Data Structures and Efficiency*

Both MATLAB and PYTHON support their own data structures to represent matrix objects which are then exposed to external modules via a programming interface. To avoid replication and

copying of data between MATLAB or PYTHON and the Toast core libraries, Toast allows the construction of matrix and vector objects that link to external data buffers. This allows the efficient direct manipulation of MATLAB and PYTHON objects while maintaining a consistent interface in the Toast matrix and vector class structure. The following code example from a PYTHON module shows the construction of a Toast vector that directly links to a PYTHON vector:

```
double *vec_data=(double*)PyArray_
DATA(py_vec);
npy_intp *vdims=PyArray_DIMS(py_vec);
TVector<double> vec(vec_data,vdims[0],
SHALLOW_COPY);
```

Subsequently, `vec` behaves like an ordinary Toast vector, but shares its data with the `py_vec` PYTHON object. Dense matrices are wrapped in a similar way. For sparse matrices, Toast uses a CSR format which is also supported by PYTHON with the `scipy` module. `Scipy` does not have a C interface; therefore, the data and index arrays are extracted from the PYTHON object at script level, and passed separately to the interface module

```
# a function that copies a CSR matrix mat to
# a toast Module function
def function1(mat):
  toast.Function1(mat.data, mat.indptr,
  mat.indices)
# a function that returns a CSR matrix from
# a Toast module function
def function2(prm):
  data,rp,ci,m,n=toast.Function2(prm)
  return scipy.sparse.csr_matrix((data,
  ci,rp),
    shape = (m,n))
```

Where functions 1 and 2 are the corresponding function in the toast PYTHON module. The CSR matrix construction in their implementation would then be

```
int *rowptr = (int*)PyArrayDATA(py_mat_rp);
int *colidx = (int*)PyArrayDATA(py_mat_ci);
T *data = (T*)PyArrayDATA(py_mat_data);
TCompRowMatrix<T> mat(m, n, rowptr, colidx,
data, SHALLOW_COPY);
```

Where `py_mat_data`, `py_mat_rp` and `py_mat_ci` are the representations of the CSR matrix arrays passed to the function. Again, this construct avoids the duplication and copying of data between matrix representations.

MATLAB only supports the compressed sparse column matrix format. While this could be accommodated by a transposition flag in the Toast sparse matrix class, MATLAB in addition uses a storage format for complex data types that is incompatible with the Toast format, necessitating the conversion of data structures at the Toast-MATLAB interface, which makes the Toast-MATLAB interface slightly less efficient than PYTHON.

4.9 *Visualization Tools*

Both MATLAB and PYTHON provide visualization tools that can be utilized in connection with Toast functionality to display mesh geometry and results. MATLAB provides a `toastShowMesh` function to display 2-D and 3-D meshes and nodal funtions interpolated over the mesh area or surface. In addition, Toast can write out mesh geometries and nodal solutions in visualization toolkit (VTK) format, which can be visualized in a variety of standard tools. Figure 8 shows the surface of a head mesh exported from Toast and displayed in the MayaVi2 Data Visualizer.

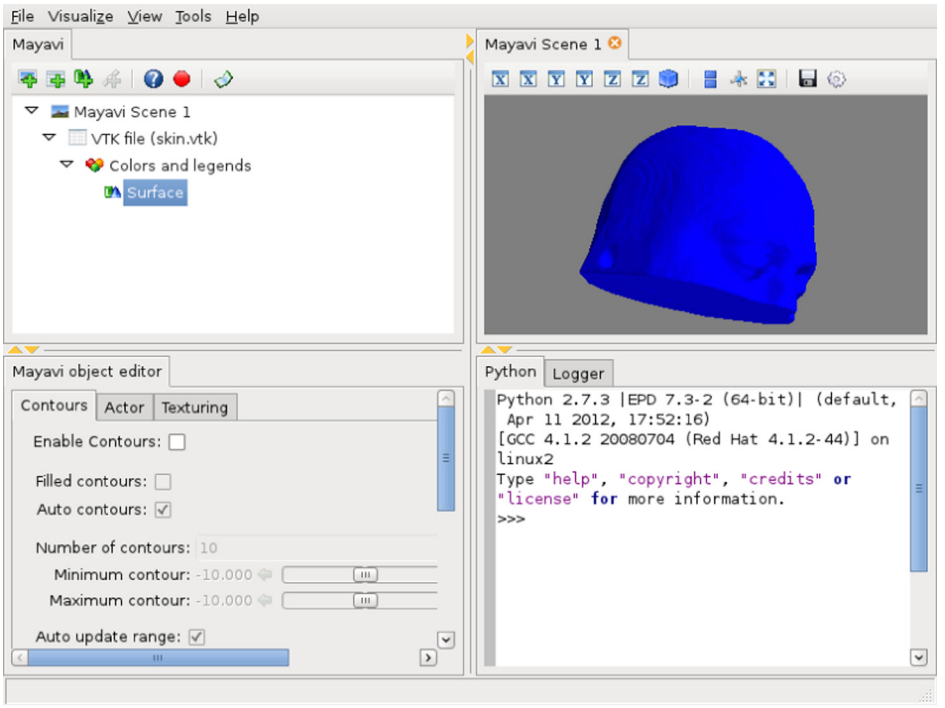

Fig. 8 Head mesh exported to VTK format and displayed in MayaVi2 Data Visualizer.

5 Usage Examples

In this section, we demonstrate the functionality of the Toast suite by presenting script examples for forward and inverse DOT solvers in MATLAB and PYTHON.

5.1 *DOT Forward Solver*

We consider the problem of computing the complex photon density field $\phi(\mathbf{r},\omega)$ in Eq. (1) in a complex head mesh using the Toast-MATLAB toolbox. A schematic of the data components comprising the forward problem is shown in Fig. 9, including the definition of the mesh geometry from node coordinates and element connectivity graph, the definition of source and boundary projection operators and the nodal parameter distribution. The forward solver uses this information to assemble the system matrices of the FEM problem, the right-hand sides and boundary operators. Using an appropriate preconditioned linear solver, the resulting fields and boundary data can be computed.

The problem geometry is represented in a `Mesh` class which acts as a container for element and node coordinate lists, it maintains the mapping from individual element to global DOF, and provides methods for querying the sparsity structure of the global system matrices and their assembly. Toast only contains limited support for mesh generation, but can import meshes generated from external meshing tools, which can then be saved in Toast format. Access to the mesh geometry and integration methods in the MATLAB-Toast framework is provided via the `toastMesh` class. A mesh object may be loaded from a file, or constructed on the fly, and then can be queried with the `Data` method to extract node coordinates and element connectivity graph:

```
mesh = toastMesh('head.msh');
% read mesh from file
[nd elidx eltp] = mesh.Data;
% get mesh geometry
nnd = size(nd,1);
% node count
nel = size(elidx,1);
% element count
mesh.Display;
% display the mesh
```

The `Data` method retrieves the node (`nd`) and element arrays (`elidx`), together with an element type list (`eltp`). The `Display` command shows the mesh geometry in a 3-D surface plot. An example mesh view for a head mesh with internal structure is shown in Fig. 10.

To perform the system matrix assembly for the DOT forward problem in Eq. (7), we now define an empty sparse matrix to hold the DOT stiffness matrix, and loop over all mesh elements to compute the element integrals in Eq. (8), before mapping them to the global DOF:

```
smat = sparse(nnd,nnd);
for i=1:nel
  el = mesh.Elmat(i);     % extract element
    object
  idx = el.Dof;           % local->global
    DOF mapping
  Kel = el.Mat('PDD',kap);      % diffusion
    term
  Cel = el.Mat('PFF',mua);      % absorption
    term
  Bel = omega*el.Mat('FF');     % frequency
    term
  Ael = hmesh.Elmat(el,'BndPFF',zeta);  % bnd
term
  % assemble into global stiffness matrix
  smat(idx,idx) = smat(idx,idx) +...
    Kel + Cel + Ael + 1i*Bel;
end
```

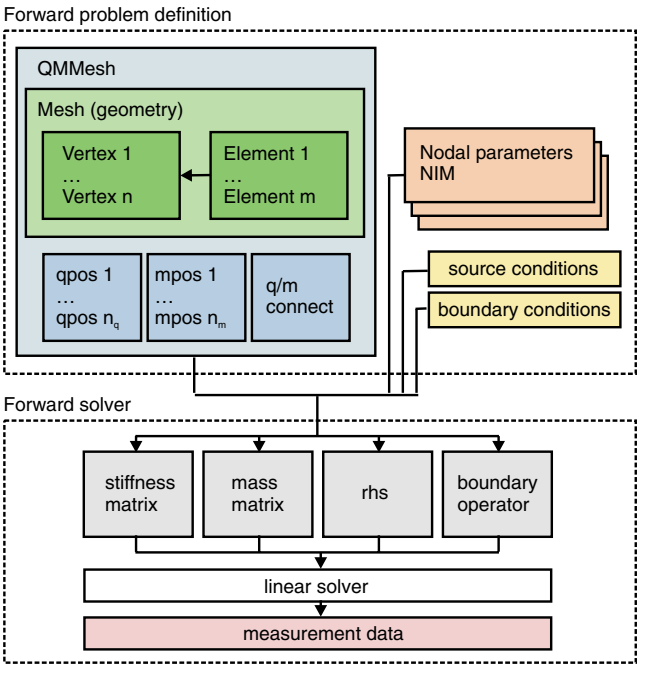

Fig. 9 Layout of the Toast forward solver. The mesh object, together with parameter distributions provided as nodal coefficients, source and boundary conditions is used to construct the FEM linear system.

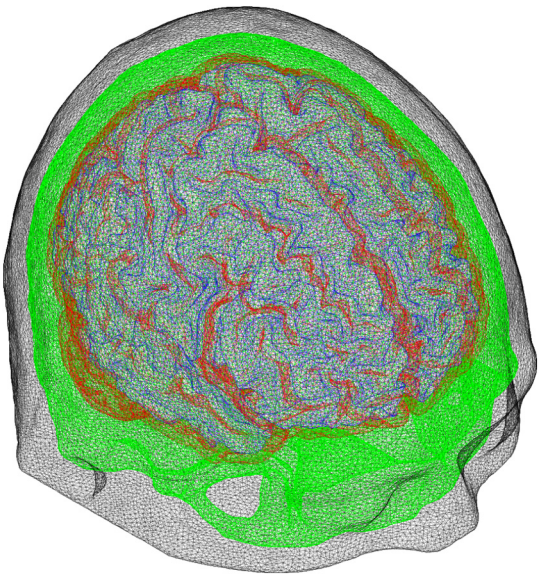

Fig. 10 MATLAB mesh viewer: Visualization of region interfaces for skin, skull, gray and white matter of a volume mesh of the head.

Where `mua`, `kap`, and `zeta` are the nodal coefficient vectors of the parameter distributions of the DOT forward problem, and `omega` is the modulation frequency. Instead of computing the integrals per element and assembling them manually into the global stiffness matrix, Toast also allows to assemble each global matrix component in a single step with the `SysmatComponent` method, performing the mapping from local to global DOF in the library core:

```
K = hmesh.SysmatComponent ('PDD', kap);
C = hmesh.SysmatComponent ('PFF', mua);
B = omega*hmesh.SysmatComponent ('FF');
A = hmesh.SysmatComponent ('BndPFF', zeta);
smat = K + C + A + 1i*B;
```

As an additional level of abstraction, Toast also provides a convenience function, `dotSysmat` that simplifies the matrix assembly specifically for the DOT problem to a single function call:

```
smat = dotSysmat(hmesh,mua,mus,ref,omega);
```

where `mua`, `mus`, and `ref` are the nodal absoption, scattering, and refractive index coefficient vectors.

The right-hand sides are assembled from a predefined list of source positions read from a data file:

```
mesh.ReadQM('head.qm');
qvec = hmesh.Qvec();
```

where `qvec` is a sparse matrix representing multiple boundary source distributions as column vectors. The complex fields for all sources can now be computed with a MATLAB linear solver, such as GMRES (generalized minimum residuals):

```
nq = size(qvec,2);    % number of sources
phi = zeros(n,nq);
for q=1:nq
  qq = qvec(:,q);
  phi(:,q) = gmres(smat,qq,20,1e-10,1000);
end
```

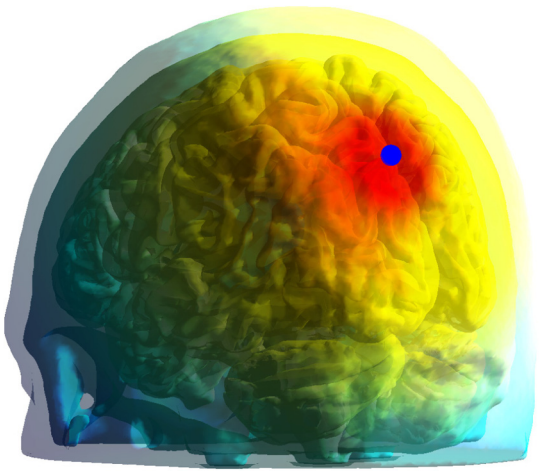

Fig. 11 Visualization of logarithmic amplitude on the skin surface and the interfaces between skin/skull, CSF/gray matter and gray matter/white matter arising from a point source at the outer surface (indicated by a blue dot).

As an example, the logarithmic amplitude on the interfaces of a four-layer head model, arising from a modulated point source on the skin surface, is shown in Fig. 11. The fields can now be projected to measurement data of boundary exitance by applying the measurement operators, which can then be displayed as source-detector maps, as shown in Fig. 12:

```
mvec = mesh.Mvec();
meas = mvec.' * phi;
% sinogram for log amplitude:
figure; imagesc(real(log(meas)));
% sinogram for phase shift:
figure; imagesc(imag(log(meas)));
```

To demonstrate the use of the Toast-PYTHON module, the equivalent PYTHON script is shown in the following listing. Both script interfaces link to the same Toast core libraries, and therefore produce equivalent results.

```
# PYTHON DOT forward solver
import numpy as np
from scipy.sparse import linalg
import matplotlib.pyplot as plt
from toast import mesh
#load mesh
hmesh = mesh.Read('headmesh.msh')
n = mesh.NodeCount(hmesh)
# load source-detector definitions
mesh.ReadQM(hmesh,'circle.qm')
qvec = mesh.Qvec(hmesh).real
mvec = mesh.Mvec(hmesh).real
# construct the system matrix
mua = np.ones(n)*0.01
mus = np.ones(n)*1
ref = np.ones(n)*1.4
smat = mesh.Sysmat(hmesh,mua,mus,ref,freq)
# compute the fields for all sources
phi = np.empty(qvec.shape)
for q in range(qvec.shape[1]):
  q = qvec[:,q].todense()
  res = linalg.bicgstab(smat,qq,tol=1e-10)
  phi[:,q] = res[0]
# map fields to boundary measurement
meas = mvec.transpose() * phi
# display data as sinogram
plt.imshow(np.real(np.log(meas)),
interpolation='none')
plt.show()
plt.imshow(np.imag(np.log(meas)),
interpolation='none')
plt.show()
```

For ease of use, the MATLAB toolbox also provides a graphical interface for providing the input parameters to the DOT forward solver, as shown in Fig. 13.

5.2 *Iterative Parameter Reconstruction*

The inverse problem for the DOT problem consists of recovering the spatial distributions of the optical parameters $\mu_a(\mathbf{r})$ and $\kappa(\mathbf{r})$ [or equivalently, $\mu_a(\mathbf{r})$ and $\mu_s(\mathbf{r})$] of the model Eq. (1) in domain Ω from boundary measurements of light transmission at the surface $\partial\Omega$. The iterative approach provided by the Toast suite solves

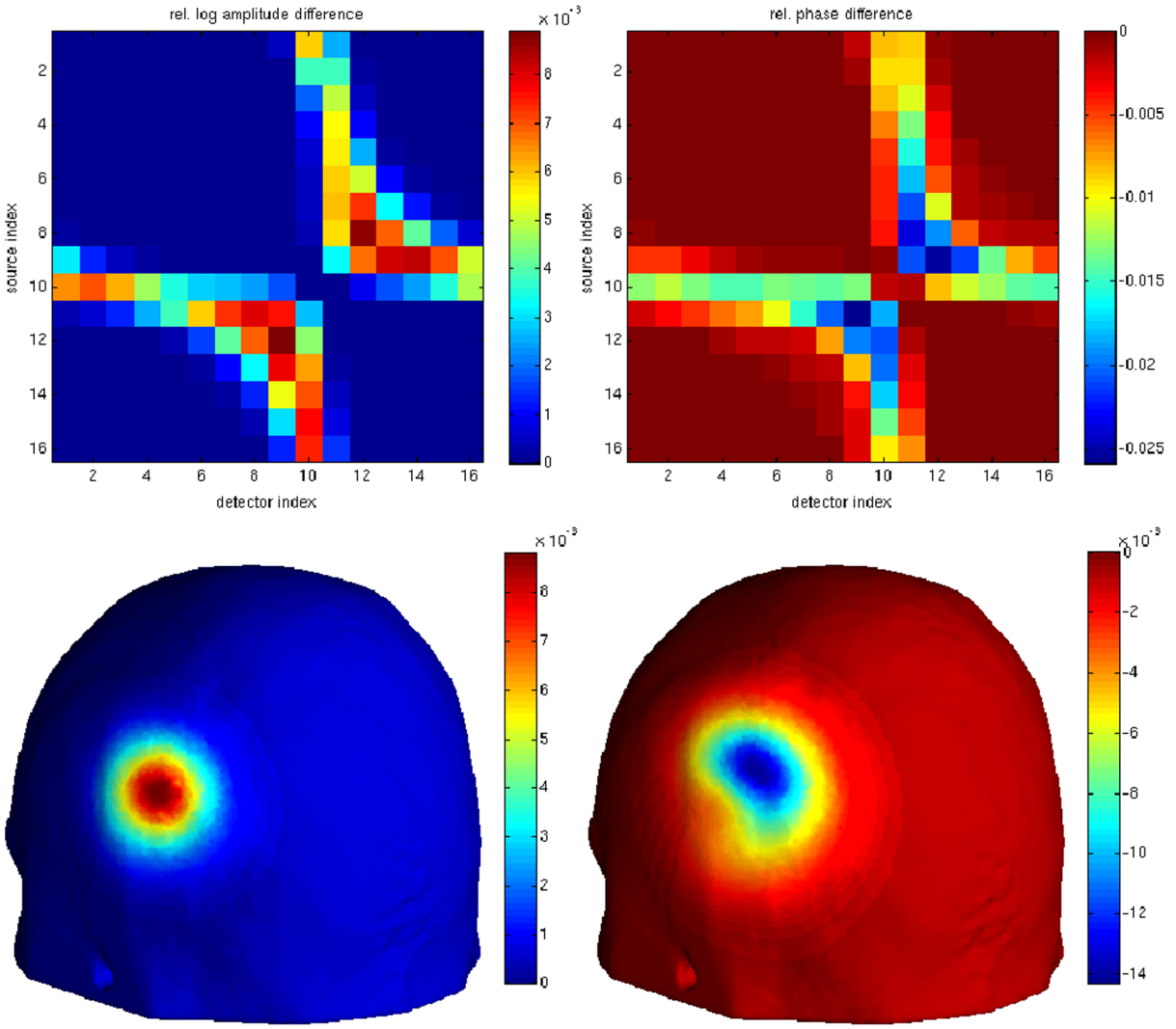

Fig. 12 Top: sinograms of relative logarithmic amplitude (left) and phase boundary data differences (right) between background parameters and an inclusion of increased absorption ("haemorrhage") in the cortical layer. The images in the bottom row show the projections onto the surface of the field differences from a single source opposite the inclusion. (An animation showing the projected inclusion from all sources is available online.)

this nonlinear problem by finding the minimizer $\{\mu_a, \kappa\} = \hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \Psi(\mathbf{x})$ to the objective function defined in Eq. (5).

Toast provides the building blocks to minimize Ψ with a choice of iterative optimization schemes, such as NCGs, or a GN approach.[14] This includes a method for the computation of the direct and adjoint fields (`toastFields`), the gradient of the objective function Ψ with respect to the model parameters x (`toastGradient`), the Jacobian of the forward model (`toastJacobian`), and an inexact line search for obtaining a step length in a given search direction (`toastLineSearch`).

Further, Toast provides a selection of regularization schemes, such as Tikhonov and TV, for implementing the regularization term $\mathcal{R}$ in Eq. (5). The Toast-MATLAB and PYTHON bindings provide a convenient way to implement the inverse solver, although it is also possible to write a solver that directly links to the C++ interface. A simple example for a nonlinear Polak-Ribiére conjugate gradient scheme with line search follows:

```
while (itr <= itrmax) && (err > tol)
  r    =    -toastGradient(hmesh,hbasis,...
qvec,mvec,mua,mus,ref,freq);
  r = r. * x;     % parameter scaling
  if itr > 1
    delta_old = delta_new;
    delta_mid = r' * s;
  s = r;
  if itr == 1
    d = s;
    delta_new =r' * d;
  else
    delta_new = r' * s;
    beta = (delta_new - delta_mid) / delta_old;
    d = s + d*beta;
  end
  step = toastLineSearch(x,d,step,...
  err,@objective);
  x = x + d*step;
```

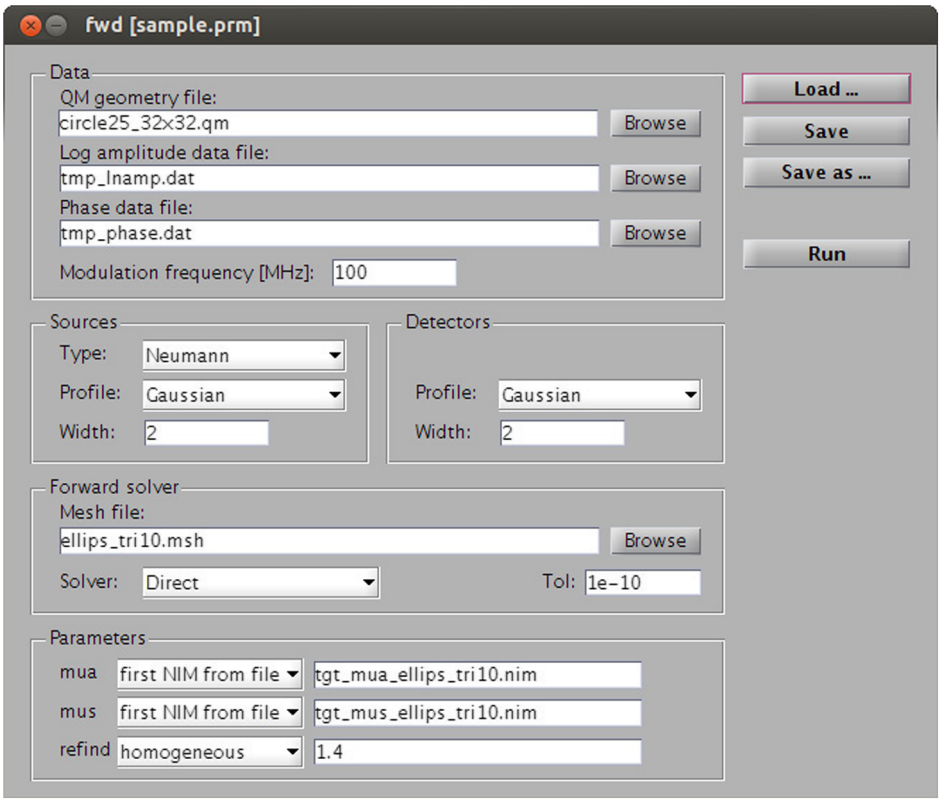

Fig. 13 Example for a MATLAB graphical interface to the DOT forward solver.

```
  proj = toastProject(hmesh,x,ref,...
   freq,qvec,mvec);
  err = toastObjective(proj,data,sed,...
   hreg,x);
end
```

As a further example, the following MATLAB code fragment implements a damped GN scheme with line search, given by the update rule

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(x)} + \alpha[\mathbf{J}^T(x)\mathbf{J}(\mathbf{x}) + \tau\mathcal{R}''(\mathbf{x})]^{-1}\{\mathbf{J}^T(\mathbf{x})[\mathbf{y} - f(\mathbf{x})] - \tau\mathcal{R}'(\mathbf{x})\}, \quad (10)$$

```
% Parameter reconstruction: GN iteration
mua = mua_estimate;
mus = mus_estimate;
x = [mua;mus];    % parameter vector
logx = log(x);    % reconstruct for log
while (itr <= itrmax) && (err > tol)
  % Jacobian at current estimate
  J = toastJacobian (hmesh,hbasis,...
   qvec,mvec,mua,mus,ref,freq;
  m = size(J,1);      % data space dimension
  p = size(J,2);    % parameter space dimension
  J = spdiags(1./sd,0,m,m) * J;
  % data normalisation
  J = J * spdiags(x,0,p,p);
  % parameter normalisation
  J = J * spdiags(M,0,p,p);
  % Hessian normalisation
  g = J' * ((data-proj)./sd) - ...
   hreg.Gradient(x);
  % Gradient of cost function; hreg is
  % a handle for a regularisation object
  dx = toastKrylov(x,J,g,M,hreg,1e-2);
  % Obtain an update with Toast's
  % implicit Krylov solver: toastKrylov
  % computes dx = (J' * J) \ r without
  % explicitly forming the Hessian
  % H = J' * J.
  step = toastLineSearch (logx,dx,...
   step,err,@objective);
  % Line search for step size
  % 'objective' is callback function for
  % computing objective function for
  % given parameter values
  logx = logx + dx*step;
  % add update to estimate
  x = exp(logx);    % map to linear parameters
  mua = x[1:p];       % extract absorption
  kap = x[p+1:end];   % extract diffusion
end
```

Where `toastJacobian` returns the Jacobian matrix $\mathbf{J}$ for the DOT problem for a given set of absorption and diffusion parameter coefficients, `toastKrylov` solves $(\mathbf{J}^T\mathbf{J} + \tau\mathcal{R}'')\mathbf{x} = \mathbf{g}$, `toastLineSearch` implements an inexact 1-D line search for step length α, and `hreg` is a regularization object that supports the evaluation of value, gradient $\mathcal{R}'$ and Hessian $\mathcal{R}''$ of the regularization term $\mathcal{R}$. In this example, the parameter vector $\mathbf{x}$ is transformed to logarithmic values, ensuring a restriction to positive values.

Where the computation of the Jacobian matrix is impractical due to its size, the solver can be modified to use a matrix-free approach, where the Frechet derivative $\mathbf{Jx}$ and adjoint Frechet

derivative $\mathbf{J}^T\mathbf{y}$ are evaluated implicitly by function calls rather than using an explicit matrix representation. This allows the application of the GN solver to large-scale problems with a combination of high-grid resolutions and a large number of measurements.

A graphical user interface (GUI) MATLAB application for the DOT inverse problem included in the Toast++ package is shown in Fig. 14.

An example of a reconstruction of the spatial distribution of the two coefficients in a cylindrical domain is shown in Fig. 15. In this example, frequency-domain forward data were generated on a mesh with 83,142 nodes and 444,278 tetrahedral elements, for all combinations of 80 sources and 80 detectors, arranged in five rings around the cylinder mantle. The forward mesh geometry, as well as source (red) and detector positions (blue) are shown in Fig. 15(a).

The data were then contaminated with 1.0% additive random Gaussian noise and used as an input for a GN reconstruction with TV regularization into a regular $48 \times 48 \times 48$ grid of trilinear basis functions. The mesh used by the inverse solver was coarser with 27,084 nodes and 141,702 tetrahedra elements. Figures 15(b) and 15(c) show cross sections of the target and reconstructed parameter distributions after 10 GN iterations. Figure 15(a) shows isosurfaces of the target and recovered inclusions at value 0.012 for μ_a and 1.2 for μ_s.

5.3 *Regularization and Structural Priors*

The Toast library contains regularization classes for augmenting the objective function of the inverse problem with a regularization term. Supported methods include Tikhonov, TV, and Markov random field (MRF) regularizers. In addition, structural information, such as internal boundaries with step changes in parameters can be incorporated by spatially varying regularization. Boundary information can be provided as a raster image, which can for example be obtained from a segmented magnetic resonance imaging or computed tomography image.

Using the MATLAB bindings, a TV regularization object can be created with

```
hreg = toastRegul('TV', hbasis, x, tau,
'Beta', beta);
```

Where `'TV'` denotes the regularization method (here a TV scheme), `hbasis` is the handle of a basis mapper object, `x` is the initial parameter vector, and `tau` is the regularization hyperparameter. In addition to these fundamental parameters required by all regularization constructors, individual regularization schemes may require addtional arguments. In this case, the TV object is supplied with a threshold parameter `beta`. After construction, the regularization object can be queried to return a value, gradient, second derivative, or its diagonal, given a parameter vector $\mathbf{x}$:

```
v = hreg.Value(x);
g = hreg.Gradient(x);
H = hreg.Hess(x);
Hdiag = hreg.HDiag(x);
```

In addition to the TV prior, Toast supports a range of other regularization strategies, including Tikhonov, Huber, or Perona-Malik methods. With each regularization method, Toast also allows the inclusion of structural prior information about internal boundaries. This information is provided to the regularization constructor as an image of a diffusivity field. This allows application of the regularization term in a spatially varying

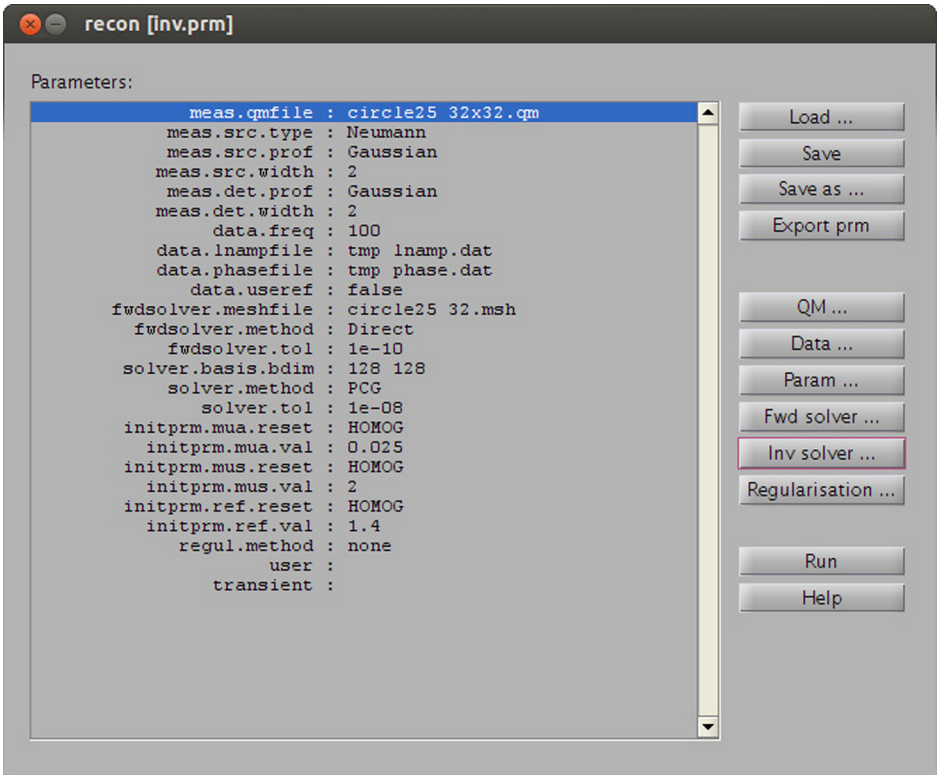

Fig. 14 GUI sample application for the DOT inverse solver included in the Toast-MATLAB toolbox.

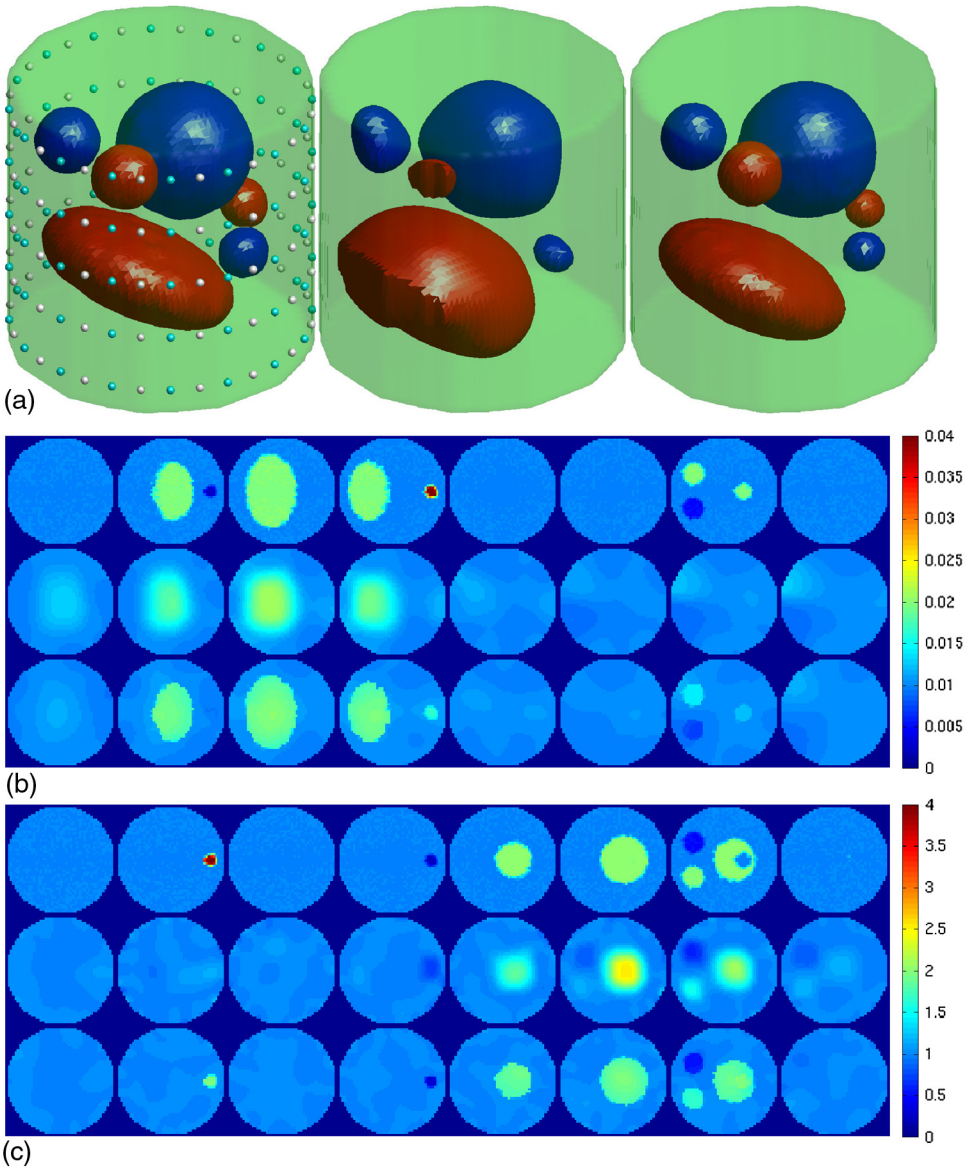

Fig. 15 Reconstruction of absorption and scattering features in a cylindrical object. (a) Left: target with absorption (red) and scattering features (blue). Source and detector positions on the surface are marked with white and cyan points, respectively. Middle: isosurfaces of features reconstructed with a generic TV regularizer. Right: reconstruction results with structural MRF prior. (b) Cross sections of absorption distributions. Rows from top to bottom: target, generic TV reconstruction, structural MRF prior reconstruction. (c) Cross sections of scattering distributions. Rows from top to bottom: target, generic TV reconstruction, and structural MRF prior reconstruction.

manner. For the definition of a TV prior, the constructor would then be specified as

```
hreg = toastRegul('TV', hbasis, x,...
  tau, 'Beta', beta, 'KapRefImage', img, ...
  'KapRefScale', scale,...
  'KapRefPMThreshold', th);
```

where `scale` is a global scaling factor for the diffusivity image, and `th` is the Perona-Malik[25] threshold as a fraction of the maximum image value.

The right image of Fig. 15(a) and the bottom rows of Figs. 15(b) and 15(c) show the results of a GN reconstruction of the absorption and scattering distributions in a cylinder, using a MRF regularization scheme that is based on the neighborhood graph of the reconstruction basis. Structural prior information was included, where the shapes and locations of the internal features were assumed known. Consequently, the results are significantly better than the generic TV reconstruction shown in the same image. All objects are recovered with correct shape and locations. Recovery of feature contrasts is also very good, except, in the case with no prior, for the most challenging features consisting of the small high-contrast absorption feature, and the absorption feature embedded in a highly scattering sphere.

6 Further Applications

In addition to the examples of light propagation and optical parameter reconstruction from frequency-domain measurement data presented in this paper, Toast++ can be applied to different

problems, such as chromophore reconstruction from multiwavelength measurements,[26] or fluorescence imaging.[27] While Toast ++ has been developed primarily for applications in DOT, the core finite-element libraries can be adopted in other fields of research, such as linear elasticity for tissue deformation modeling.[9]

Instead of reconstructing parameters in a regular basis, Toast is also able to recover piecewise constant region parameters, based on the known internal structure of the target volume. This approach can be combined with a shape-based reconstruction approach, where the boundaries of regions with distinct piecewise constant parameter values are recovered as well as their contrast. Shape-based reconstruction problems applied to Toast include explicit methods such as spherical harmonics,[28,29] as well as implicit methods such as level sets.[30]

7 Discussion

We have presented the Toast++ software suite of libraries and programs for light transport modeling and image reconstruction in DOT. The library supports simulation of diffuse light transport in heterogeneous highly scattering media by using a numerical finite-element approach. Parallel implementations of matrix assembly and solution are supported for shared and distributed memory architectures. In addition, graphics processor hardware acceleration provides scalable performance for large-scale problems.

For reconstruction of the optical parameter distributions, the software provides the building blocks for defining the inverse problem as an iterative minimization approach of the least-squares problem, taking into account the ill-posedness of the problem, and providing a range of regularization approaches. Examples for solvers based on NCGs and GN schemes are provided. Other types of solvers such as Markov chain Monte Carlo approaches can be easily constructed with the functions supported by the Toast interface.

Toast combines the efficiency of the compiled and parallelized library code for sparse matrix computation and finite-element analysis with the ease of use of script languages by providing function bindings for the MATLAB and PYTHON script environments. This allows the rapid prototyping of new code, convenient testing and debugging options, and the use of available toolsets including preconditioned linear solvers and advanced visualization routines. Toast++, thus, allows researchers in the field efficient code development and adaptation to specific problems without the need for coding of low-level numerics code, while maintaining high performance across a range of hardware platforms.

Future development of the Toast++ suite will include support for additional numerical modeling strategies, reconstruction, and regularization methods. We also plan to open up Toast for other medical imaging applications, such as electrical impedence tomography by providing appropriate application codes and sample scripts.

8 Toast++ Library Availability and Development

The Toast software suite, including the core C++ libraries, MATLAB and PYTHON interfaces, and example application codes, is published as an open-source package under a GNU general public license. The Toast home site is located at http://www.toastplusplus.org. It contains links to the subversion repository with the current build, as well as binary packages for Linux and Windows, together with tutorials, demos, sample scripts, and documentation. Source-level documentation is provided via doxygen and can be compiled by the user.

Acknowledgments

The work in this paper was supported by EPSRC (Grant Nos. EP/I030042/1, EP/K03829X/1, EP/J021318/1 and EP/E034950/1) and the Wellcome Trust. We wish to acknowledge the long and fruitful collaborations with colleagues at University College London, Kuopio University, Politecnico di Milano and others, in particular Teresa Correia, Jari Kaipio, Ville Kolehmainen, Tanja Tarvainen, Ilkka Nissilä, Jorge Ripoll, Athanasios Zacharopoulos, Nicolas Ducros, Andrea Bassi, Cosimo DAndrea, Oliver Dorn, and Arjun Yodh.

References

1. S. R. Arridge, "Optical tomography in medical imaging," *Inverse Probl.* **15**(2), R41–R93 (1999).
2. D. A. Boas et al., "Imaging the body with diffuse optical tomography," *IEEE Sig. Proc. Mag.* **18**(6), 57–75 (2001).
3. B. W. Pogue et al., "Calibration of near-infrared frequency-domain tissue spectroscopy for absolute absorption coefficient quantitation in neonatal head-simulating phantoms," *J. Biomed. Opt.* **5**(2), 185–193 (2000).
4. M. Cope and D. T. Delpy, "System for long-term measurement of cerebral blood and tissue oxygenation on newborn infants by near infra-red transillumination," *Med. Biol. Eng. Comput.* **26**(3), 289–294 (1988).
5. D. K. Joseph et al., "Diffuse optical tomography system to image brain activation with improved spatial resolution and validation with functional magnetic resonance imaging," *Appl. Opt.* **45**(31), 8141–8151 (2006).
6. D. R. Bush et al., "Toward noninvasive characterization of breast cancer and cancer metabolism with diffuse optics," *PET Clin.* **8**(3), 345–365 (2013).
7. M. Schweiger and S. R. Arridge, "Fast 3-D image reconstruction in optical tomography using a coarse-grain parallelization strategy," *Proc. SPIE* **4250**, 93–100 (2001).
8. M. Schweiger, "GPU-accelerated finite element method for modelling light transport in diffuse optical tomography," *Int. J. Biomed. Imag.* **2011**, Article ID 403892 (2011).
9. M. Schweiger et al., "An inverse problem approach to the estimation of volume change," in *Proc. of MICCAI*, J. S. Duncan and G. Gerig, Eds., pp. 616–623, Lecture Notes in Computer Science, Springer, Heidelberg (2005).
10. N. Ducros et al., "A virtual source pattern method for fluorescence tomography with structured light," *Phys. Med. Biol.* **57**(12), 3811–3832 (2012).
11. R. C. Haskell et al., "Boundary conditions for the diffusion equation in radiative transfer," *J. Opt. Soc. Am. A* **11**(10), 2727–2741 (1994).
12. S. R. Arridge and M. Schweiger, "Direct calculation of the moments of the distribution of photon time of flight in tissue with a finite-element method," *Appl. Opt.* **34**(15), 2683–2687 (1995).
13. M. Schweiger and S. R. Arridge, "Direct calculation with a finite-element method of the Laplace transform of the distribution of photon time of flight in tissue," *Appl. Opt.* **36**(34), 9042–9049 (1997).
14. M. Schweiger, S. R. Arridge, and I. Nissilä, "Gauss-Newton method for image reconstruction in diffuse optical tomography," *Phys. Med. Biol.* **50**(10), 2365–2386 (2005).
15. S. R. Arridge and M. Schweiger, "A gradient-based optimisation scheme for optical tomography," *Opt. Express* **2**(6), 213–226 (1998).
16. S. R. Arridge et al., "A finite element approach for modeling photon transport in tissue," *Med. Phys.* **20**(2), 299–309 (1993).
17. K. Devine et al., "Zoltan data management services for parallel dynamic applications," *Comput. Sci. Eng.* **4**(2), 90–97 (2002).
18. M. Schweiger et al., "The finite element model for the propagation of light in scattering media: boundary and source conditions," *Med. Phys.* **22**(11), 1779–1792 (1995).

19. P. S. Mohan, V. Y. Soloviev, and S. R. Arridge, "Discontinuous Galerkin method for the forward modelling in optical diffusion tomography," *Int. J. Numer. Meth. Eng.* **85**(5), 562–574 (2011).
20. J. Sikora et al., "Diffuse photon propagation in multilayered geometries," *Phys. Med. Biol.* **51**(3), 497–516 (2006).
21. J. Elisee, A. Gibson, and S. R. Arridge, "Diffuse optical cortical mapping using the boundary element method," *Opt. Express* **2**(3), 568–578 (2011).
22. J. P. Elisee, A. Gibson, and S. R. Arridge, "Combination of boundary element method and finite element method in diffuse optical tomography," *IEEE Trans. Biomed. Eng.* **57**(11), 2737–2745 (2010).
23. M. Schweiger and S. R. Arridge, "Image reconstruction in optical tomography using local basis functions," *J. Electron. Imaging* **12**(4), 583–593 (2003).
24. N. Bell and M. Garland, "Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations version: 0.3.0," http://cusp-library.googlecode.com (2012).
25. A. Douiri et al., "Anisotropic diffusion regularization methods for diffuse optical tomography using edge prior information," *Meas. Sci. Technol.* **18**(1), 87–95 (2007).
26. A. Corlu et al., "Diffuse optical tomography with spectral constraints and wavelength optimization," *Appl. Opt.* **44**(11), 2082–2093 (2005).
27. T. Correia et al., "Split operator method for fluorescence diffuse optical tomography using anisotropic diffusion regularisation with prior anatomical information," *Biomed. Opt. Express* **2**(9), 2632–2648 (2011).
28. A. Zacharopoulos et al., "3D shape based reconstruction of experimental data in diffuse optical tomography," *Opt. Express* **17**(21), 18940–18956 (2009).
29. S. R. Arridge et al., "Reconstruction of subdomain boundaries of piecewise constant coefficients of the radiative transfer equation from optical tomography data," *Inverse Probl.* **22**(6), 2175–2196 (2006).
30. M. Schweiger et al., "Reconstructing absorption and diffusion shape profiles in optical tomography by a level set technique," *Opt. Lett.* **31**(4), 471–473 (2006).

Martin Schweiger is a senior research fellow in the Department of Computer Science, University College London, United Kingdom, and a member of the Center for Medical Image Computing. His research interests are in numerical modeling, inverse problems, and medical image reconstruction. He has numerous publications in the field of diffuse optical tomography. He is the principal author of the Toast modeling and reconstruction software package.

Simon Arridge is professor of image processing in the Department of Computer Science, University College London, United Kingdom, and an honorary professor in the Department of Mathematics, and the director of the University College London Center for Inverse Problems. He has worked in medical imaging for more than 20 years and has approximately 150 publications in this area. His work has been pivotal in the development of diffuse optical tomography (DOT), for which he has developed techniques for solving both the forward and (nonlinear) inverse problems.