MCmatlab: An open-source, user-friendly, MATLAB-integrated 3D Monte Carlo light transport solver with heat diffusion and tissue damage

Marti, Dominik; Aasbjerg, Rikke N.; Andersen, Peter E.; Hansen, Anders Kragh

Published in:
Proceedings of SPIE

Link to article, DOI:
10.1117/12.2507754

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
MCmatlab: an open-source user-friendly MATLAB-integrated 3D Monte Carlo light transport solver with heat diffusion and tissue damage

Dominik Marti, Rikke Aasbjerg, Peter E. Andersen, Anders K. Hansen

Event: SPIE BiOS, 2019, San Francisco, California, United States
MCmatlab: An open-source, user-friendly, MATLAB-integrated 3D Monte Carlo light transport solver with heat diffusion and tissue damage

Dominik Marti, Rikke Aasbjerg, Peter E. Andersen, Anders K. Hansen

Technical University of Denmark, Department of Photonics Engineering, Frederiksborgvej 399, Roskilde, Denmark, 4000

Abstract: While there exist many Monte Carlo (MC) programs for solving the radiative transfer equation (RTE) in biological tissues, we have identified a need for an open-source MC program that is sufficiently user-friendly for use in an education environment, in which detailed knowledge of compiling or UNIX command-line cannot be assumed. Therefore, we introduce MCmatlab, an open-source codebase thus far consisting of (a) a fast 3D Monte Carlo RTE solver and (b) a finite-element heat diffusion and Arrhenius-based thermal tissue damage simulator, both run in MATLAB. The kernel for both of these solvers are written in parallelized C and implemented as MATLAB MEX functions, combining the speed of C with the familiarity and versatility of MATLAB. We present example results generated by the RTE solver and the thermal model. MCmatlab is easy to install and use and can be used by students and experienced researchers alike for simulating tissue light propagation and, optionally, thermal damage.

Keywords: Monte Carlo methods, Photothermal effects, 3D modeling, Tissue optics, Scattering, Absorption.

*Anders K. Hansen, E-mail: ankrh@fotonik.dtu.dk

1. Introduction

When modelling light interaction with biological tissue, the first step is typically to calculate the distribution of light within the tissue given the optical properties for a given illumination. This light distribution is described by the solution to the radiative transfer equation (RTE), which is often solved with Monte Carlo (MC) methods, in which the light is simulated as photon packets which are gradually absorbed as they travel through the medium and will randomly undergo scattering at a rate dependent on the local optical properties of the medium.

Due to the conceptually straightforward nature of the Monte Carlo methods of solving the RTE, especially using rectangular cuboid voxel meshes, many implementations have been made with various strengths and weaknesses and using various meshing schemes. Some solvers assume certain geometrical symmetries (usually cylindrical) or make other numerical approximations to speed up the calculations, which may otherwise in some cases be prohibitively time-consuming. Many advanced methods have been demonstrated over the years, simulating effects such as anisotropic light propagation, light polarization and fluorescence. The simulation speed has been increased using CPU and GPU parallelization and polyhedral meshing of the geometry. Various RTE solvers, including Steven Jacques’ mcxyz, have been coupled to photoacoustics and other physical phenomena.

However, most of these programs were not written with user-friendliness in mind or are not free of charge or open-source, and some are outdated and can no longer be compiled or run on modern PCs. Some programs use MATLAB or Octave as the user interface of choice, but most were implemented in low-level programming languages like C or C++ to make them very fast, but this usually also makes them difficult to integrate into a larger framework such as batch execution with programmatical pre- and post-processing, especially for researchers, engineers or students who are not themselves experienced C or C++ programmers. Additionally, many im-
implementations are compilable and executable only on a UNIX system or through a UNIX terminal on a Windows or Mac system or rely on having a CUDA enabled GPU and the corresponding development libraries installed, which in turn requires specialized knowledge on the user side to set up and compile. From our own experience in teaching students on the bachelor and master level and in collaborating with other researchers not familiar with Monte Carlo methods or software compilation, we have identified a need for an easy to use Monte Carlo code that doesn’t need to be the fastest, most versatile or most compatible to other specialized software tools, but instead is easy to install, run, and use by non-experts in programming.

The MATLAB computing environment, although not itself free of charge, is nevertheless available to and familiar to many students and researchers worldwide and offers a wealth of functions and interfaces for setting up, visualizing and analyzing data. Therefore, we have started the open-source project MCmatlab, a codebase that thus far consists of a Monte Carlo RTE solver inspired by mcxyz and a finite-element thermal diffusion and thermal damage solver, both implemented as MATLAB MEX functions. MEX functions are written in C, C++ or Fortran, and, once compiled, can be called in MATLAB as ordinary functions that combine the high speed of those low-level languages with the versatility and data-interfacing of MATLAB. MCmatlab comes with pre-compiled MEX functions verified to work with recent versions of MATLAB. The code has been written in C in such a way that, if recompiling should prove necessary, it can be done completely within MATLAB in only two steps.

In section 2, we describe MCmatlab’s capabilities for modelling the fluence distribution of incident light (2.1), fluorescence light generation and fluence distribution simulation (2.2.2), and its imaging simulation (2.2.3). Finally, we describe how MCmatlab can be used to simulate heat generation, diffusion and ensuing damage (3).

Parts of sections 1, 2.1, 2.2.1 and 3 of this proceeding have previously been published in the JBO special section in honour of Steve Jacques, on whose program mcxyz our software is based.²⁰

MCmatlab is available at https://gitlab.gbar.dtu.dk/biophotonics/MCmatlab.

2. MCmatlab’s Monte Carlo solver for radiative transfer

2.1. MCmatlab’s RTE solver

The distribution of light within the tissue is found by solving the radiative transfer equation. MCmatlab’s RTE solver is based on and still follows at its core the method of the program mcxyz⁴, developed by Steven Jacques, Ting Li and Scott Prahl at the Oregon Medical Laser Center.

MCmatlab, like mcxyz, operates in Cartesian coordinates and makes no assumptions as to symmetry in the simulated volume. Its boundary is a rectangular cuboid and its internal volume is uniformly divided into voxels, which are themselves also rectangular cuboids. The user assigns to each voxel a medium or tissue type, described by its absorption coefficient $\mu_a$, its scattering coefficient $\mu_s$ and its Henyey-Greenstein scattering anisotropy factor $g$ at the applied wavelength.

An input light beam is simulated by launching photon packets and calculating their paths in the simulated volume, using pseudo-randomly generated numbers to determine initial photon packet position and trajectory as well as path lengths between scattering events and scattering angles. As photon packets propagate from one voxel to the next, they deposit some of their en-
ergy ("weight") into the voxel depending on the voxel’s absorption coefficient \( \mu_a \). This deposited energy is numerically accumulated in a 3D matrix, which upon conclusion of the simulation is normalized to the input power, providing the local absorption rate per watt of incident light, in units of W/cm^2/W.incident. Dividing this by the voxels’ absorption coefficients yields \( F(x, y, z) \), the local fluence rate (irradiance or intensity) per watt of incident light, in units of W/cm^2/W.incident.

The RTE solver itself is written in C code to be compilable and callable in Windows or on Mac from MATLAB as a MEX function, combining the speed of C with the flexibility and ease-of-use of MATLAB. This means that one can perform the entire simulation initialization, execution, postprocessing, visualization and further interfacing in MATLAB. As such, there is on Windows or Mac no need for, e.g., a separate UNIX-emulating console shell to run a compiler and the Monte Carlo executable and furthermore no need for any intermediate input/output files, since all data is passed in and out of the MEX function through memory. The simulation inputs and the fluence rate matrix output (the solution to the radiative transport equation) are saved as MATLAB .mat files, a compressed binary file format allowing for flexible and easy saving and loading and ensuring smaller file sizes compared to uncompressed binary or plain-text file formats.

The full selection of beam types is currently pencil beams; isotropically emitting points; infinite plane waves; Gaussian focus, Gaussian far field beams; Gaussian focus, top-hat far field beams; top-hat focus, Gaussian far field beams; top-hat focus, top-hat far field beams and Laguerre-Gaussian LG01 beams.

The focal plane of the beams is defined as the plane containing the user-specified focus point with a normal vector given by the user-specified beam center trajectory. For the beams with Gaussian and/or top-hat beam profiles, the initial position and trajectory of each photon packet is calculated by first randomly sampling, according to the chosen focus beam profile, Gaussian or top-hat, the point in the focal plane that the photon packet would hit in the absence of scattering and then randomly sampling the initial trajectory according to the chosen far field beam profile. The initial position of the photon packet is then defined as the intersection between the \( z = 0 \) plane and the line going through the focal plane target point along the photon packet’s initial trajectory. For the Laguerre-Gaussian LG01 beam, the sampling is slightly different; for a focal plane target point at a position \( \mathbf{r} \) relative to the focus point, the trajectory is chosen only among those directions that are orthogonal to \( \mathbf{r} \). This ensures that the intensity is zero everywhere along the center axis of the beam, as expected of an LG01 beam. The focus and far-field widths can be freely specified and could, for instance, be chosen to match that of a diffraction limited beam.

It can be challenging to properly visualize data in a three-dimensional volume, especially if the volume contains fine-structured heterogeneities in multiple layers. Therefore, MCmatlab uses interactive 3D slice plotting, in which the color scale can be switched between linear and logarithmic by checking a box.

MCmatlab can also optionally simulate refraction and Fresnel reflection at interfaces between media with different refractive index, provided that the geometry is oriented so that the interface is parallel to the xy-plane. Since MCmatlab does not track polarization, the reflection probabilities are calculated assuming that the light is unpolarized.
2.2. Example and comparison of results

2.2.1. RTE solver for incident light

To test whether the results of our Monte Carlo RTE solver agrees with prior work, we compared MCmatlab with mcxyz using the example shown on the mcxyz website, a $200 \times 200 \times 200$ voxel model of a cylindrical blood vessel of radius 100 µm embedded 400 µm deep in dermis, with a 60 µm layer of epidermis on the surface. Figure 1 illustrates the geometry, visualized using the interactive 3D volumetric slice plotting used for many different plots in MCmatlab. Table 1 shows the parameters used for the demonstration. For comparison purposes, all $\mu_a$, $\mu_s$ and $g$ values were set equal to those used in the mcxyz demonstration example. The thermal properties are used only for the thermal simulations, shown in section 3. Note that the values are not necessarily accurate for real tissue, but are for demonstration purposes only. The illumination beam was a collimated top-hat beam of radius 300 µm.

![Geometry illustration](image)

**Figure 1:** Screenshot of MCmatlab’s illustration of the simulation geometry definition with slices shown at $x = 0.05$ cm, $x = 0$ cm, $y = 0.05$ cm, $y = 0.01$ cm, and $z = 0.1$ cm.

<table>
<thead>
<tr>
<th>Tissue</th>
<th>$\mu_a$ (cm$^{-1}$)</th>
<th>$\mu_s$ (cm$^{-1}$)</th>
<th>$g$</th>
<th>$c_v$ (J cm$^3$ K$^{-1}$)</th>
<th>$k$ (W cm$^{-1}$ K$^{-1}$)</th>
<th>$E_a$ (W mol$^{-1}$)</th>
<th>$A$ (s$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water</td>
<td>0.00036</td>
<td>10.0</td>
<td>1.00</td>
<td>4.19</td>
<td>0.0058</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Epidermis</td>
<td>16.6</td>
<td>376</td>
<td>0.90</td>
<td>3.76</td>
<td>0.0037</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Dermis</td>
<td>0.459</td>
<td>357</td>
<td>0.90</td>
<td>3.76</td>
<td>0.0037</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Blood</td>
<td>231</td>
<td>94.0</td>
<td>0.90</td>
<td>3.80</td>
<td>0.0052</td>
<td>$4.23 \cdot 10^5$</td>
<td>$7.6 \cdot 10^6$</td>
</tr>
</tbody>
</table>

$6.36 \cdot 10^7$ photon packets were launched in both programs. In Figure 2, the results of the MCmatlab simulations are shown in terms of the fluence rate and the absorbed power per unit volume. Figure 3 shows a contour-plot comparison to the results of mcxyz in the $y = 0$ and $x = 0$ planes. It can be seen from Figure 3 that there is good agreement between the results of mcxyz and MCmatlab. There is a bug in mcxyz that affects fluence rates in voxels bordering some of the cuboid boundaries, but this does not affect all inner voxels.
Running in a UNIX-emulating Cygwin terminal on a mid-range laptop PC from 2012 (Dell Latitude E6530) with 64-bit Windows, mcxyz launched $6.29 \times 10^4$ photon packets per minute. When run in MATLAB on the same PC, MCmatlab launched $1.09 \times 10^6$ photon packets per minute (a factor of 17 more). MCmatlab’s multithreading to the laptop’s 8 CPU processors accounts for roughly a factor of 4.5 of the speed increase compared to mcxyz, while the remaining factor of 4 is due partly to switching to the fast Mersenne Twister pseudo-random number generator and partly to many small optimizations throughout the code base.

### 2.2.2. RTE solver for fluorescence

After simulating the fluence distribution of incident, or excitation, light in the voxel model, MCmatlab can simulate fluorescence. MCmatlab uses the absorbed power per voxel obtained...
from the first Monte Carlo run as a 3D source distribution in a second Monte Carlo run, where each new photon packet is emitted from a voxel that is randomly chosen with a probability proportional to the absorbed excitation power in that voxel multiplied by the fluorescence yield of the medium in that voxel. The photon packet is emitted from a random location within the voxel and with a random initial direction, sampled from an isotropic probability distribution. All the further steps of photon packet propagation, scattering and absorption are as described in section 2.1 above. Figure 4 shows the geometry for such a model for fluorescence light, where we used a test fluorescer as an example for a fluorophore. The optical properties for the fluorescing solution used here are $\mu_s = 100 \text{ cm}^{-1}$ for both the excitation and emission wavelength (488 nm and 518 nm, respectively), and $\mu_a = 100 \text{ cm}^{-1}$ and $\mu_a = 1 \text{ cm}^{-1}$ for the excitation and emission wavelength, respectively. The fluorescence yield in this example is arbitrarily assumed to be 0.5. Figure 5 shows the fluence rate and absorbed power per unit volume for the excitation light at 488 nm. Figure 6 shows the resulting fluorescence fluence distribution at 518 nm; only the fluorescer is assumed to fluoresce in this example, while blood will absorb both the excitation and the fluorescent light, as can be seen in Figure 5 (left) and Figure 6 (right).

![Geometry illustration](image)

Figure 4: Geometry for a fluorescent sample: three absorbing tubes containing blood above three fluorescing tubes containing a test fluorescer, embedded in a phantom that has the optical properties of dermis.

![Normalized fluence rate and absorbed power](image)

Figure 5: (left) The fluence rate and (right) the absorbed power per unit volume for the excitation light. Slices are shown at the same positions as in Figure 4.
2.2.3. Image formation in turbid media

For both the normal and the fluorescence Monte Carlo run, the user can define an imaging system that will capture the photons if they hit a user-defined lens aperture. For each photon hitting this aperture, its apparent origin position in the focal plane is calculated based on the photon’s trajectory: this corresponds to where the photon will end up in the image plane for magnification 1x. With this addon, the user can simulate optical imaging in scattering media. Figure 7 shows the geometry of an imaging lens with a focal length of 0.05 cm, a collector aperture of 0.2 cm in diameter and a field of view of 0.08 cm in diameter. Figure 8 shows both the bright-field and fluorescence image detected with an imaging lens as shown in Figure 7 (left), when focusing the lens on the blood-filled tubes, visible as shadows in both of the images. Figure 9 shows the same two images when focusing the system on the fluorescer filled tubes (Figure 7 (right)): here, the blood-filled and thus absorbing tubes cannot be resolved (Figure 9 (left)), while the fluorescing tubes can be seen (Figure 9 (right)), weakly shadowed vertically by the crossing absorbing blood filled tubes.

Figure 7: Light collector geometry when (left) focusing on the blood-filled tubes and when (right) focusing on the fluorescer filled tubes.
The MCmatlab heat deposition, diffusion and Arrhenius-based thermal damage solver

As another optional add-on to the Monte Carlo RTE solver described in section 2, MCmatlab includes a 3D finite-element time-domain heat deposition and diffusion solver, operating on the same voxel grid as the MC RTE solver. During the simulation, Arrhenius-based thermal chemical changes such as tissue coagulation can also be calculated. Like the Monte Carlo solver, the thermal solver is also implemented in C as a MATLAB-compilable and -callable MEX function, and thus also benefits from combining the speed of C with the flexibility of MATLAB.

3.1. Theory

The thermal solver takes as input the fluence rate output $F$ and the geometry definition of the Monte Carlo simulations, this time with additional material parameters; the media’s physical thermal properties $c_v$ and $k$ and, optionally, chemical thermal properties $E_a$ and $A$ of the considered chemical reaction. Although the physical thermal parameters must be specified for all involved media, the chemical properties may be specified for all, some or none of them.

The temperature evolution can be simulated for continuous light exposure or for single- or multi-pulse light exposure with user-specified duty cycle and period. At the end of the simulation, a plot is shown that illustrates the spatial distribution of chemically changed media in the simulation volume. A video can also be automatically generated showing the temperature evolution.
Denoting the local volumetric heat capacity by \( c_V \) and thermal conductivity by \( k \), the heat equation in continuous form is given by
\[
\frac{\partial T}{\partial t} = \frac{q + \nabla \cdot (k \nabla T)}{c_V},
\]
where \( q \) is the local rate of heat deposition, which is calculated from multiplying the voxel’s fluence rate with its absorption coefficient. The equation is solved with an explicit finite-element method, where the temperature change \( \Delta T \) after a small time step \( \Delta t \) for the voxel at spatial position indices \( (i_x, i_y, i_z) \) is calculated as
\[
\Delta T = \frac{\Delta t}{c_V} (P \mu_a F \ dx \ dy \ dz + \\
(2k k_{x^-} \ dy \ dz) \frac{d}{dx} (T_{x^-} - T) + (2k k_{x^+} \ dy \ dz) \frac{d}{dx} (T_{x^+} - T) + \\
(2k k_{y^-} \ dx \ dz) \frac{d}{dy} (T_{y^-} - T) + (2k k_{y^+} \ dx \ dz) \frac{d}{dy} (T_{y^+} - T) + \\
(2k k_{z^-} \ dx \ dy) \frac{d}{dz} (T_{z^-} - T) + (2k k_{z^+} \ dx \ dy) \frac{d}{dz} (T_{z^+} - T)),
\]
where \( P \) is the incident power, \( (dx, dy, dz) \) are the voxel side lengths, \( k \) is the thermal conductivity of a voxel, \( T \) its temperature, and the subscripts designate neighboring voxels in the negative and positive \( x \), \( y \), and \( z \) directions. The user can specify whether the solution should be calculated by assuming \( \Delta T = 0 \) on all boundaries, corresponding to constant-temperature (heat-sinked) boundaries, or by simply omitting the above terms that contain temperatures outside the volume, corresponding to insulated boundaries. In the current version of MCmatlab, effects such as cooling by blood perfusion are neglected, as well as any temperature dependence of the optical and thermal properties.

The thermal chemical change is quantified for each voxel in terms of the dimensionless Arrhenius \( \Omega \) value, which starts at zero and increases over time as the voxel is exposed to elevated temperatures \( T \), depending on the Arrhenius activation energy \( E_a \) and Arrhenius pre-exponential factor \( A \):
\[
\Omega(x, y, z) = \ln \left( \frac{c_0(x, y, z)}{c_T(x, y, z)} \right) = A \int_0^\tau \exp \left( \frac{-E_a}{R \cdot T(x, y, z, t)} \right) dt,
\]
where \( c_0 \) is the concentration of undamaged molecules at the simulation start, \( c_T \) is the concentration of undamaged molecules at the simulation end, \( R \) is the gas constant, \( t \) is simulation time and \( \tau \) is the total simulated time duration. Those voxels that have \( \Omega > 1 \) at the end of the simulation are considered to be “damaged”, that is, more than \( 1 - \frac{1}{e} = 63\% \) of the molecules having undergone a chemical reaction such as coagulation.

### 3.2. Example

In this demonstration, the blood vessel defined in Sec. 2 was used as an input to the thermal solver, setting the input power to 4 W, running for 5 light pulses of 1 ms on-time and 4 ms off-
time each. A still frame of the video (supplementary material) is shown in Figure 10, and the resulting illustration of damaged tissue is shown in Figure 11.

![Temperature evolution, t = 0.021 s](image1)

Figure 10: Video 1 still image showing the temperature evolution during 5 pulses of light incident on the tissue defined in Section 2.2.1.

![Thermal damage illustration](image2)

Figure 11: Screenshot of MCmatlab's thermal damage illustration. The volume of coagulated blood is shown in green.

4. Conclusions

Open-source Monte Carlo programs such as mcxyz are invaluable as tools for researchers and students to model light-tissue interaction and to help understand some of the underlying physics. With MCmatlab, we have taken the core concepts of mcxyz and integrated them into open-source MATLAB MEX functions, making it easier for researchers and students who are not experienced C programmers or users of UNIX systems to join in and use these tools.

The MCmatlab codebase currently consists of the RTE solver for finding the light distribution in complex turbid media for incident light and/or fluorescence light, simulation of image formation and a thermal solver, useful for simulating, e.g., photocoagulation.

We expect that students and experienced researchers alike will benefit from this suite of software for both research and teaching activities, and we openly share the code with everyone, encouraging others to add to and modify the code as they wish.

Disclosures

The authors have no relevant financial interests in the article and no other potential conflicts of interest to disclose.
References


