Biomedical Physics & Engineering Express

RECEIVED
6 April 2017

REVISED
21 June 2017

ACCEPTED FOR PUBLICATION
26 June 2017

PUBLISHED
6 July 2017

PAPER

GPU accelerated Monte Carlo simulation of light propagation in inhomogeneous fluorescent turbid media: application to whole field CW imaging

N A Carbone, D I Iriarte and J A Pomarico
Instituto de Física Arroyo Seco-IFAS (UNCPBA) and CIFICEN (UNCPBA-CICPBA-CONICET), Pinto 399, 7000 Tandil, Argentina
E-mail: ncarbone@exa.unicen.edu.ar

Keywords: Monte Carlo, NIR, GPGPU, fluorescence

Abstract
We present an implementation of a Monte Carlo simulation software for fluorescent turbid media, accelerated by a GPU (graphic processing unit). The code is based on previous work by Alerstam *et al* (2008 *J. Biomed. Opt.* **13** 060504) and Wang *et al* (1995 *Comput. Methods Programs Biomed.* **47** 131–46), with the addition of a voxelized medium without symmetries and with an inhomogeneous distribution of absorbers and fluorescent markers. Cartesian coordinates are used in place of the cylindrical ones used in previous versions. It is particularly aimed at the simulation of CW whole-field reflectance and transmittance images of fluorescence and absorption. Several tests and comparisons with numerical and theoretical techniques were performed in order to validate our approach.

1. Introduction

Light propagation through turbid media is a field of optics that has seen increasing research and development in recent years, mainly due to its potential as a medical image diagnosis tool. A number of biological tissues can be described as turbid media in relation to the way light behaves inside them.

Upon entering a turbid medium, light interacts with it mainly in two ways: absorption and scattering. Instead of following a straight path, light travels following a snake-like path until it is absorbed or it escapes the boundaries of the medium. Moreover, in a region of wavelengths between red and near infrared (NIR), called the optical window, scattering dominates over absorption, allowing light to penetrate deeply enough in the tissue to obtain valuable diagnostic information [1].

Given these two methods of interaction, turbid media can be optically characterized by two magnitudes: the absorption coefficient μ_a and the scattering coefficient μ_s, related to the mean free path between absorption and scattering events, respectively. Also the anisotropy factor, g, describing the average angle of the change in direction after an scattering event, and the refractive index, n, are needed for a complete optical description of turbid media. However, since these two last parameters are usually obtained from the literature [2] or are given, knowing the volumetric distribution of μ_a and μ_s is often enough information to describe the medium.

It is known [3] that the mentioned optical parameters are different depending on the tissues and their health state. For example, active tumors are highly vascularized and the increased concentration of haemoglobin translates into an elevated μ_a. On the other hand, water-filled cysts have decreased μ_a and μ_s. However, the intrinsic contrast provided by the different haemoglobin concentrations is often not sufficient to provide good localization and/or characterization of lesions. Thus, in order to improve detectability and/or specificity, in addition to intrinsic absorption, turbid media imaging can be complemented with the use of fluorescent markers like indocyanine green (ICG). The absorption and emission spectra of this molecule lie in the NIR and it is possible to differentiate excitation and emission radiation by proper filtering of the radiation being gathered by the imaging device.

Several novel techniques using fluorescent markers have been developed recently [4]. In particular, ICG, despite its low quantum efficency (4%), has been used in humans for a long time and is considered safe by the FDA. Besides, different studies show that the pharmacokinetics of ICG is different in healthy tissues than in malignant lesions, allowing the observation of lesions by temporal discrimination of fluorescence [5, 6].

Detection of stained inclusions in turbid media is relative simple if the surrounding medium contains no fluorescent agent. However, in real clinical cases, the host (healthy) medium around the target lesion(s) always contains some residual amounts of fluorophore. Because the volume of the healthy host is much larger than that of the lesion, even a small amount of fluorophore in the bulk may produce strong fluorescence signals which compete with the emission of the lesion and precludes its detection.

The study of the propagation of light, needed for the development of the aforementioned techniques, has a number of challenges. Mainly, the equation describing the problem, the radiative transfer equation, can only be solved numerically. Moreover, its diffusive approximation can only be analytically solved for a small number of simple geometries, even before considering the inclusion of fluorophores. Not having an analytic solution to work with impairs the research and development of actual laboratory systems and the analysis of the results obtained by them.

In light of this, a number of numerical solutions and simulations have been developed. In particular, Monte Carlo simulations are considered the 'gold-standard' against which other solutions (analytical or numerical) are compared. Monte Carlo allows for the simulation of the direct problem without any limitations in geometrical complexity and relying on very few assumptions [7]. However, the main limitation is computational time.

Monte Carlo simulations are statistical by definition, a problem that is exacerbated by the scattering nature of light inside turbid media. In order to have an adequate signal to noise ratio, a great number of photons need to be simulated (about 10^9), each of which suffers thousands of scattering events before being terminated.

As a way to deal with the computational problem, the simulation can be accelerated using graphics processing units (GPUs). These specialized processors are characterized by having a very high number of simple cores. These cores were originally constructed to compute illumination and transformation of pixels in 3D modeling software in real time, and in recent years they have seen increased usage in more general problems. General purpose graphics processing units (GPGPU) systems are particularly useful in problems that allow extensive parallel processing. Monte Carlo simulation of light propagation through turbid media is one of these problems, in which each photon is completely independent of the others, allowing them to be simulated in different threads and thus saving up to orders of magnitude in processing time compared to CPU computing.

1.1. Background and state-of-the art of Monte Carlo simulations

The implementation of a complete Monte Carlo simulation of light propagation on turbid media is not trivial. However, its advantages as a highly valued method of solving this problem mean that a lot of effort has been put into its research in recent years.

The final objective is the development of a complete Monte Carlo code where no assumptions of symmetry are made, an arbitrarily complex heterogeneous media can be modeled, and absorption, scattering and fluorescence phenomena are taken into account. Furthermore, for a number of applications time resolution is important. All of this in a reasonable run-time for individual simulations.

One of the first implementations is *MCML* (Monte Carlo Multi-Layer) by Wang *et al* [7]. This code considers a multilayered medium in which each layer is homogeneous. By assuming this geometry, it can take advantage of the cylindrical symmetry of the problem to speed up the run-time of the code.

With *MCML* as a starting point, Alerstam *et al* developed *CUDAMC* [8]. This code is a refit of *MCML* to be run in GPUs using Compute Unified Device Architecture (CUDA) as a framework. CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia® that allows the use of a CUDA-enabled GPU for general purpose processing. The reimplementation is far from trivial as a lot of the code needs to be rethought to address the different nature of GPUs regarding thread management and memory access. Nevertheless, it achieves speeds-ups of $100\times$ over *MCML* or even more, depending on the GPU/CPU combination.

MCML and *CUDAMC* are well tested, and are often used as reference software for the validation of other Monte Carlo algorithms. However, they are limited in the complexity of the media they can simulate and do not incorporate fluorescence phenomena. The first of these limitations has been worked on by a number of research teams. In fact, there are several implementations of 3D Monte Carlo simulations with voxel-described media [9–12] and mesh-described media [13–16]. In particular, both the code developed by Fang *et al* [12] (voxel-based) and the one by Shen *et al* [16] (mesh-based) are available as open source code [17, 18]. These codes are well tested and validated, but none of them considers a direct implementation of fluorescence simulations.

There are also several fluorescence Monte Carlo algorithms [19–22], but they assume homogeneous and/or layered media. They also usually implement time-resolved simulations in a single-point detection geometry. These codes are used with great success for the study of the time-dependent characteristics of ICG bolus infusions, useful to assess cerebral perfusion [23, 24]. Monte Carlo simulations are also being used for the study of photoacoustics and related phenomena [25].

According to a recent review of Monte Carlo modelling of light transport in tissues by Zhu *et al* [26] and Fujita *et al* [27], and to the best of the authors' knowledge, there is no Monte Carlo simulation software, accelerated by GPU, of a voxelized medium without symmetries and with an inhomogeneous distribution of absorbers and fluorescent markers. The present contribution aims to present such a software. In particular, its main intended use is the simulation of CW reflectance and transmittance wide field images of both absorption and fluorescence phenomena. The code discussed here is available as a constantly updated and improved open source in the GitHub webpage [28] and as an static snapshot of the code used while writing the present paper in our group webpage [29].

2. CUDAMCFL implementantion

The software here presented, hereafter called *CUDAMCFL*, is based on the code of Alerstam *et al* [8], which is an reimplementation in CUDA of the software developed by Wang *et al* [7].

The Alerstam code, hereafter referred as *CUDAMC*, contains several important assumptions: an homogeneous, non-fluorescent medium; cylindrical symmetry for absorption deposition; and an infinitely narrow beam as the photon source. These latter two assumptions need to be lifted for the simulation of an inhomogeneous fluorescent media. Broadly speaking, the code developed in the present contribution adds over *CUDAMC*:

(i) A complete Cartesian geometry, without assumptions of symmetry.

(ii) Inhomogeneities embedded in the medium, defined by an input matrix.

(iii) A voxelization of the bulk media, for inhomogeneity description and for photon hitting density storage.

(iv) The possibility of an isotropic source located anywhere in the medium.

(v) Fluorescent sources simulation.

(iv) Output of (x, y) 2D images of transmission and reflection over a user-selectable area of interest.

2.1. Overview

Figure 1 shows a simplified flux diagram of *CUDAMCFL*. The software outputs transmittance and reflectance images for both absorption and fluorescence. It takes as input a configuration file (the 'MCI' file) describing the optical parameters of both the bulk and the inclusion and a 3D matrix describing the distribution of the inhomogeneities. The latter file encodes an integer (implemented as a `short int`) for each voxel of the simulated media called 'bulk *descriptor*'. Each possible value correlates to a set of optical properties specified in a user-provided line of the MCI file.

Very roughly speaking, the fluorescence simulation proceeds in two steps. First, a standard absorption simulation is performed. Using a narrow beam source, a high number of photons are propagated through the medium, adding the current weight of each photon to a voxel-specific 'bucket' as it passes through said voxel. This is called the *photon hitting density* matrix.

After this, each voxel becomes an isotropic source for a second set of simulations. A fixed number of photons is launched and the absorption and/or reflection image produced by them is scaled by the photon hitting density and fluorescence characteristics of the source voxel and added to the final image.

Both kernel simulations are essentially the same, the main differences being that the absorption simulation needs to accumulate the photon hitting density and that in the fluorescence simulations the source is located inside the medium and is isotropic. In the present contribution, we will not enter into much detail about the bulk of the transport process as it is very detailed explained in the papers of *CUDAMC* [8]. Instead, we will focus on the modifications done for *CUDAMCFL*.

2.2. Cartesian coordinates

The first distinction between *CUDAMCFL* and the original *CUDAMC* is the utilization of Cartesian coordinates in every step of the simulation. As we want to simulate inhomogeneities, the cylindrical symmetry is lost, forcing one to abandon the cylindrical coordinates previously used.

Doing this implies a clear degradation in performance as much larger matrices and slightly more complicated calculations are required for photon deposition.

2.3. Voxelization of bulk properties

In order to be able to simulate an arbitrary distribution of inhomogeneities, a voxelized medium description has been adopted. This approach takes two input files: one describing the optical parameters and the fluorophore characteristics of each possible component of the simulated problem, and a simple 3D matrix with the spatial distribution of these components (called *descriptors*).

A single input in which each voxel in the 3D matrix contains all the parameters necessary for the simulation, thus allowing the possibility of as many different types of tissue as voxels, was considered. However, it proved too cumbersome to implement and the required input file would have been too large and complicated.

The size and definition of the 3D matrix used to define optical properties and to store the photon hitting density is defined by a parameter called *finesse*. This parameter is user-defined and determines the number of divisions of the 3D matrices per centimeter in each direction. So, a finesse parameter of 1 implies that each voxel is 1 cm × 1 cm × 1 cm, a finesse parameter of 2 generates voxels of 0.5 cm × 0.5 cm × 0.5 cm, and so on.

The open source code presented with this contribution includes an example Python script that allows the creation of slabs with several spherical or cylindrical inclusions. More complex geometries should be created by different means, but its implementation is trivial.

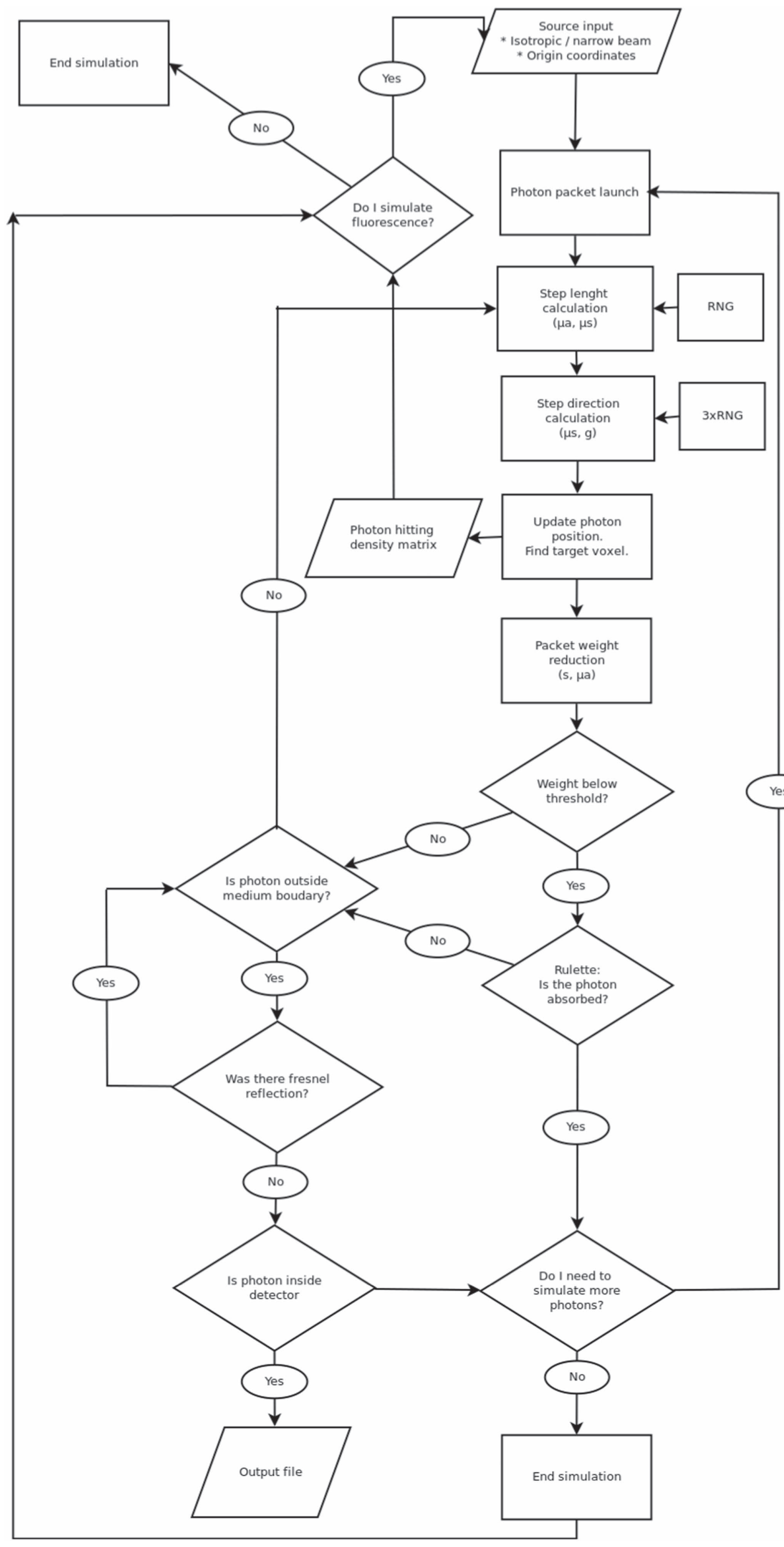
Source input
* Isotropic / narrow beam
* Origin coordinates
End simulation
Yes
No
Do I simulate fluorescence?
Photon packet launch
Step lenght calculation (μa, μs)
RNG
Step direction calculation (μs, g)
3xRNG
Photon hitting density matrix
Update photon position. Find target voxel.
No
Packet weight reduction (s, μa)
Weight below threshold?
Yes
No
Is photon outside medium boudary?
Yes
No
Yes
Rulette: Is the photon absorbed?
Yes
Was there fresnel reflection?
Yes
No
Is photon inside detector
Do I need to simulate more photons?
Yes
No
Output file
End simulation

Figure 1. Simplified flux diagram of *CUDAMCFL*.

Once the input file is read, the simulation itself needs to be modified from *CUDAMC*. In each step, the program needs to know from which descriptor it should obtain the optical parameters for the propagation. The pseudocode is available in appendix A.1.

Also, a function called `Reflect` is called when the photon steps into a voxel that has a descriptor different than the previous one and it checks for reflection at the interface between voxels. It is inherited from *CUDAMC* with a small modification to use a change descriptor instead of a layer number.

2.4. Simulation of inhomogeneities

An important objective of the present reimplementation is the simulation of inhomogeneities embedded in the medium. These objects can have different optical properties and different concentrations or characteristics of fluorophores.

Initial 'alpha' versions of *CUDAMCFL* simulated a multilayer medium (inherited from *CUDAMC*) with the possibility of a spherical inclusion embedded in a given layer. Such an inhomogeneity is described by its position, radius, optical parameters, and fluorophore characteristics. This method is useful for comparison with theoretical models and with simple experiments but is not very versatile. Nevertheless, the program retains the possibility of using this approach as an option.

In both cases, the software needs to check at every step of the photon propagation if the photon is located inside an inclusion and use the appropriate parameters for propagation. In this part of the photon propagation is where the simulation of multiple inclusions becomes computationally too inefficient when using spherical inclusions defined by their radius. Since testing if the photon is inside the inclusion means computing the distance between the current position and the center of the sphere in Cartesian coordinates, it implies power and/or square-root calculations. Doing this test multiple times, for multiple inclusions in each step of each photon, is computationally prohibitive.

In the voxelized description, in each step we only need to calculate its correspondent voxel and, with it, the adequate descriptor. It is a simpler computation and is independent of the number of inclusions. As a drawback, the inclusion geometry only has the resolution of the voxelized bulk and aliasing artifacts may be present.

In the present version of the code, Fresnel reflections are not calculated at the boundary of the inclusions. Given that inclusions are defined as voxels with different optical properties and it is not trivial to distinguish between a defined inclusion to more diffuse or random inhomogeneities that can also be simulated, Fresnel calculations need to be done at the boundary of each different voxel. Also, algorithms that are non voxel-local are needed if we want to compute Fresnel reflections based on the general shape of the inclusion (an sphere with its curved surface, for example) and not in the always perpendicular faces of the cubical voxel. All of this is cumbersome and highly computationally intensive.

Fortunately, in the vast majority of practical situations, the refraction index difference between inclusions and bulk are so small (if any) that this limitation is not significant [30, 31]. Because of this, we decided to postpone the implementation of this characteristic for a future work.

2.5. Photon launch

CUDAMC assumes a single narrow beam located at the coordinates origin as a source. This allows a simple implementation given the cylindrical symmetry of the simulated problem. For *CUDAMCFL* we had to extend the possible sources.

Narrow beam sources are still required and utilized since they are the actual sources used in the laboratory, but now they can be located in any place of the entry face using a setting in the input file. Also, given the voxelization of, particularly, the photon hitting density accumulation, a delta-like source was problematic in the not-so-improbable case of the source being exactly at the boundary of two or more voxels. To address this, and as a slight increase of realism, the narrow beam is modeled as a gauss-like source with a set width.

Another, more extensive, modification is the addition of isotropic sources. These can be located anywhere in the bulk and the photons are launched from a random position within the source voxel. Spreading the photons through a voxel instead of launching all of them from its center (or any other point) allows for better masking of the discretization problems that arise when simulating the spatially extended fluorescence source.

In the second pass of the simulation, where the fluorescence itself is simulated, the main routine calls the CUDA kernel one time per voxel, launching a reduced number of photons from an isotropic source located in said voxel.

The simplified pseudocode of the `LaunchPhoton` function is available in appendix A.2.

2.6. Photon hitting density

As mentioned in section 2.1, the simulation of the fluorescent signal is done in two steps. The first is very similar to a purely absorption simulation, where the main difference is that the photon hitting density is accumulated for each voxel. After each step of each photon, we need to calculate the voxel in which it is located and add to it the current weight of the photon. As before, a simplified pseudocode of the section of the program that performs the calculation is available in appendix A.3.

Even if the implementation seems simple, the need to check for the current position in every step and add to a thread-shared big variable (done by an `atomicAdd` CUDA function) imposes a heavy computational burden. However, as is the case with the passage to Cartesian coordinates, advances in GPU computational capabilities give rise to a usable time, unlike the GPUs available when *CUDAMC* was implemented. Also, it is important to note that the PHD accumulation is done in the same step as the calculation of the current voxel done in section 2.3.

3. Validation and results

3.1. Comparison with *CUDAMC*

The present contribution differs in many key areas from the original *CUDAMC*, which precludes many meaningful comparisons. However, in the limit of an homogeneous, non-fluorescent, cylindrically symmetrical medium, both approaches are expected to coincide.

As a first validation test we simulated light propagation in an homogeneous absorbent slab using both codes. It consisted of a 5 cm thick turbid medium with an absorption coefficient of 0.04 cm^{-1}, scattering

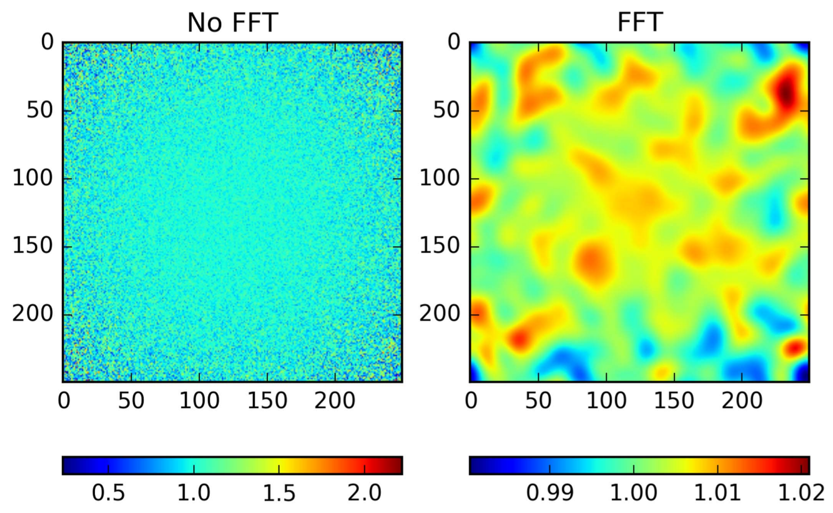

Figure 2. Division of images generated by CUDAMCFL and CUDAMC. Transmittance case.

coefficient of 48 cm^{-1}, anisotropy factor of 0.8 and refractive index of 1.33. The source was located at the center of the entry face and both reflection and transmission images were recorded. These optical properties were chosen to mimic healthy mammary tissue [2].

CUDAMC, given its use of cylindrical symmetry, returns reflection and transmission images as functions of the radius from the center, while *CUDAMCFL* uses a Cartesian coordinate system. For the sake of the present comparison, a helper Python script was used to convert the *CUDAMC* images to Cartesian coordinates.

The resulting images are 250 $\times$ 250 pixels, covering an area of 20 $\times$ 20 cm. Figure 2 shows the result of dividing the transmittance image of *CUDAMCFL* by the image generated by *CUDAMC*. Because of the statistical nature of Monte Carlo simulations the raw result is dominated by pixel to pixel noise and some filtering is required in order to appreciate any difference in the signal itself. Thus we also show the result of the division after a low pass Fourier filter is applied.

As can be seen, both algorithms produce essentially the same image beyond noise-driven fluctuations, which are always below 2%.

Analogously to the previous comparison, figure 3 shows the result of dividing the reflectance image of *CUDAMCFL* by the image generated by *CUDAMC* with and without a low-pass FFT filter applied. This filter was done in Python, using the built-in functions of NumPy. The image was converted to the frequency domain using `rfftn`, filtered with a gaussian mask using `ndimage.fouriergaussian` with a kernel size of 3 and converted back to the space domain using `irfftn`. Finally, the resulting image was normalized to be compared with the unfiltered one.

Before the FFT filtering some artifacts can be seen. The angular-dependent structures are the result of the conversion between polar and Cartesian coordinates done on the images produced by *CUDAMC* and are not related to any difference between the codes. The difference in the source point is however a result of the differences in how the sources are modeled in both codes. While in *CUDAMC* the source is strictly point-like, in *CUDAMCFL* it has a Gaussian shape, explaining the difference in the immediate neighborhood of the source point.

The filtered image shows a very good concordance, with the images only differing because of their noisy nature.

3.2. Comparison with theory

As the community-proclaimed 'gold-standard' in photon transport, it is difficult to compare results from Monte Carlo codes with an independent model. This is particularly true for fluorescent inhomogeneous media images like the one our code attempts to simulate.

However, an analytical theoretical model for a semi-infinite fluorescent medium with or without inclusions is available [32, 33]. This approach is extensively discussed in a previous work [34] and the explicit expression for B_m when the inclusion is a sphere can be found in the work by Zhu and coworkers [35].

Aiming to do a more realistic and complete comparison, instead of directly comparing the homogeneous and/or inhomogeneous outputs, we compare the results of normalizing the inhomogeneous image by the homogeneous one. As shown in previous works [34, 36], due to the exponential decrease of the diffusely reflected or transmitted light intensity with increasing distance from the source, subtle variations of intensity introduced by inhomogeneities are difficult to detect. The normalization procedure tries to overcome this problem.

Figure 4 shows the comparison of a theoretical model based on the work of Li *et al* [37] and *CUDAMCFL* for a 5 cm slab ($\mu_a = 0.04$ cm^{-1}, $\mu_s = 48$ cm^{-1}, ICG concentration = 10 nM) with a single spherical inclusion ($\mu_a = 0.08$ cm^{-1}, $\mu_s = 48$ cm^{-1}, ICG concentration = 50 nM, $d = 1$ cm). The profile along the line

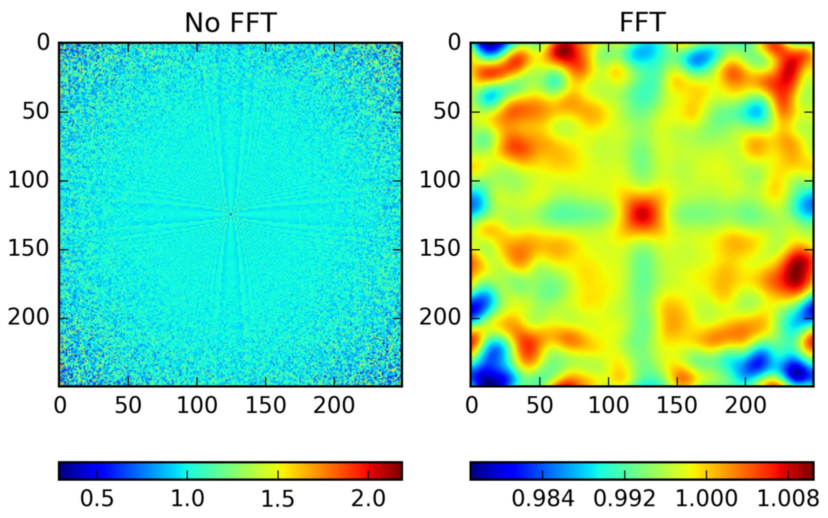

Figure 3. Division of images generated by CUDAMCFL and CUDAMC. Reflectance case.

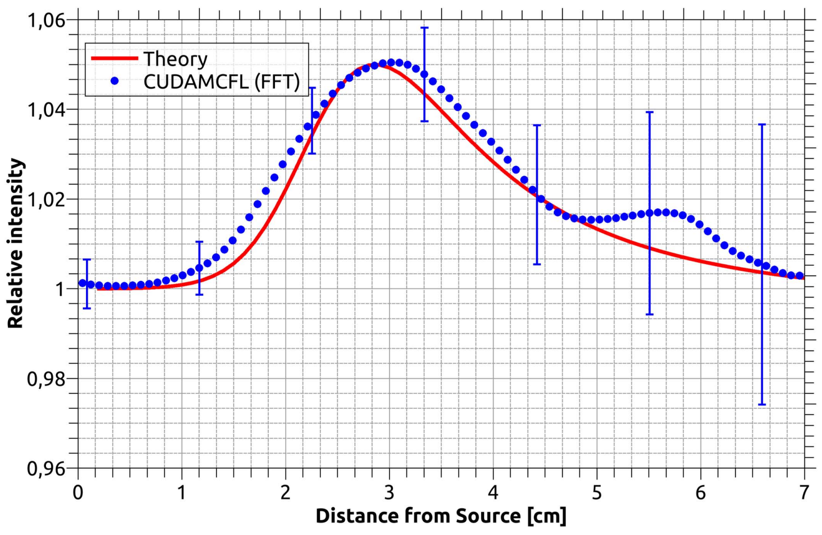

Figure 4. Comparison between theory and *CUDAMCFL* for a single spherical inclusion. Profile of the line joining the source and the center of the inclusion. The error bars of the simulation are also shown.

joining the source and the center of the inclusion is shown. A low-pass FFT filter, as implemented in QtiPlot, was used to smooth the Monte Carlo profile. Given the previously mentioned normalization, a value of 1 implies that there is no difference between the homogeneous reference image and the heterogeneous one.

Overall, the position and width of the fluorescence signal in the Monte Carlo simulation agree with the theory within the statistical error. The more evident difference is the shape of the ascending and descending slope. These differences can be explained by the difference in the modeling of the source (perfect point-like source for the theory, Gaussian beam for *CUDAMCLFL*) and of the inclusion (voxelized sphere in *CUDAMCFL* and perfect mathematical sphere in the theory), and by the diffusion approximation used by the theory.

In addition, the noisy nature of the Monte Carlo simulation, even after filtering, is responsible for some of the differences. As a FFT filter is used to remove the high-frequency noise, some artifacts are visible in the form of a low-frequency oscillation, particularly when the source–detector separation is large. The error bars represent one standard error inside a window of 25 pixels in the non filtered profile. It is evident how the noise increases for larger source–detector separations as the number of detected photons decreases.

3.3. Effects of grid finesse

The voxelization of the simulation gives the presently contributed Monte Carlo code a lot of flexibility and versatility but also constitutes one of its main limitations. By definition, using discrete voxels to define the optical properties of an heterogeneous medium and to store the calculations and final results implies reducing the resolution of the simulation.

In order to explore this effect, we did the same simulation changing the *finesse* parameter, as defined in section 2.3, to 2, 4, 6 and 8. We then subtracted the higher definition simulation to each of the three remaining ones. The simulated medium was a 5 cm slab ($\mu_a = 0.04$ cm^{-1}, $\mu_s = 48$ cm^{-1}, ICG concentration = 10 nM) with a single spherical inclusion ($\mu_a = 0.08$ cm^{-1}, $\mu_s = 48$ cm^{-1}, ICG concentration = 50 nM, $d = 1$ cm) located at a depth of 1.35 cm and at $r = (-2, 0, 0)$ cm. The source was located at $r = (-4, 0, 0)$ cm. In both cases we defined the center of the imaged area as (0, 0, 0) cm.

We aimed to simulate a typical clinical condition of a vascularized tumor embedded in otherwise healthy mammary tissue, using ICG concentrations well below the achievable concentrations *in vivo* in order to stress-test the code [38].

In all cases cases transmission and reflection images for both fluorescence and absorption were generated. A total of 5×10^9 photons were simulated for the construction of the photon hitting density and the absorption simulation while 1×10^{10} photons were simulated for the final fluorescence simulation. Figure 5 shows the difference between simulations using a finesse parameter of 2, 4 and 6 and the reference using a finesse parameter of 8, calculated as `diff` $(x, y) = [(I_f(x, y) - I_{f=8}(x, y))/I_{f=8}(x, y)] \times 100\%$. A low-pass FFT filter was applied for better visualization, implemented in the same way as was done in section 3.1.

Looking at the absorption results, we see that the difference is small. Even in the case with finesse = 2, the difference is below 6% in the region of interest and it comes from where the spherical inclusion is. The reason for this is that as the grid density decreases the error in volume and shape between a real sphere and a voxelized one increases. This effects is much lower for finesse = 4 and practically non-existent for finesse = 6. The outer region of the absorption images are rather noisy which explains the apparent high deviation in lower and upper right corners.

The interpretation of fluorescence result is less straightforward. In both the transmission and the reflectance case there is a high difference around the source position. This difference originates in the lower precision of the photon hitting density matrix and in the lower number of discrete fluorescence sources. The lack of resolution is evident in areas of high intensity gradient, such as close to the source. This error, however, decreases quickly and is already well controlled at finesse = 4 for the transmittance case.

Given the nature of the reflectance geometry, there will always be a precision error close to the source, but for this case this area is already small at finesse = 4. Special care should be taken if there is a need to simulate the conditions close to the source, though.

Generally speaking, the diffusive nature of turbid media allows us to use a rather coarse 3D matrix for both the photon hitting density and the fluorescence simulation sources. Nevertheless, if needed, the density of these matrices can be increased greatly. In the present version, *CUDAMCFL* uses an `unsigned int` for the index of the array containing the 3D matrix, limiting the number of voxels to 4 294 967 295 on 64-bit systems. However, the main limitation in choosing a high finesse parameter is the computational time required. As we are dealing with a 3D space, there is a cubic relation between the finesse parameter and the number of voxels and, with it, the number of discrete sources for the fluorescence and the number of different 'buckets' for the photon hitting density.

In order to analyse the effect on performance of the finesse parameter, figure 6 shows the run time for each of the previous simulations, divided iton the absorption simulation and the fluorescence simulation. Superimposed there is an exponential fit of the total run-time showing a good match.

It may seem that the performance penalty for having a large finesse parameter is prohibitive. However, two things must be taken into account. On one side, the performance of the simulation not only depends on hardware but also on the optical parameters. For example, in a medium with high scattering coefficient each simulated photon will suffer more collisions and, thus, move in small steps and require more iterations to be fully simulated. Conversely, a medium with low scattering requires less computation time. The latter media are also the kind of simulations that usually require a more dense grid and spatial resolution, and some of the increase in computation time can be compensated.

Also, these simulations were carried out in a modern but low-end GPU: a Nvidia GT750Ti. Using a more powerful GPU will greatly reduce run-times. Moreover, given the linearity of performance improvements with the number of cores of the GPU for highly parallel problems like Monte Carlo, the performance on more powerful GPUs can be easily extrapolated.

3.4. Sample simulation: multiple cylindrical inhomogeneities

As an example of a kind of simulation that is not possible with previous Monte Carlo codes, we show the case of three finite cylinders (1 cm diameter, 4 cm length) embedded in an otherwise homogeneous medium (slab, 5 cm thickness). The medium simulated a typical breast tissue with low but extant plasmatic ICG concentration ($\mu_a = 0.04$ cm^{-1}, $\mu_s = 48$ cm^{-1}, ICG concentration = 10 nM). The inclusions were more absorbent ($\mu_a = 0.12$ cm^{-1}) and had higher ICG concentration (150 nM) simulating a vascularized and previous tissue [6].

The group of three cylinders was located at four different depths: 1 cm, 2 cm, 3 cm and 4 cm from the source position. The geometry is shown in the first column of figure 7. In all cases transmission and reflection images for both fluorescence and absorption were generated.

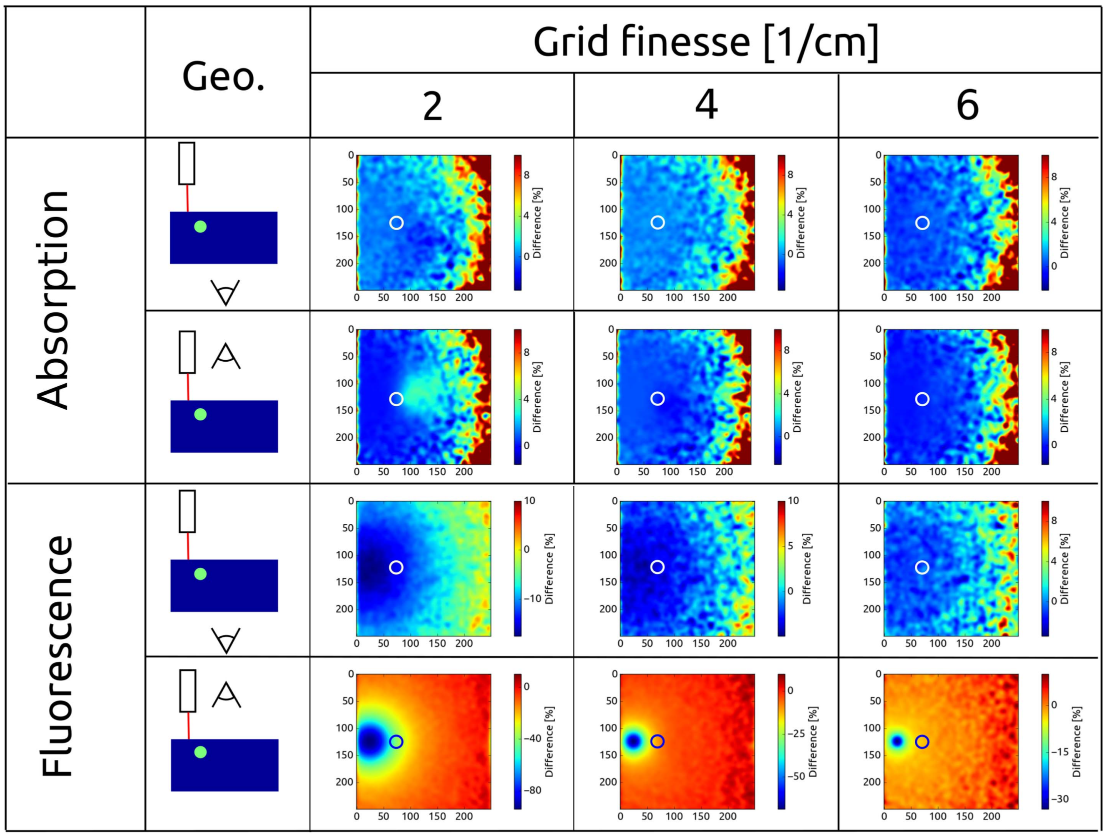

Figure 5. Simulations of a medium with a single spherical inclusion, percentage difference with a simulation using finesse = 8. Transmission and reflection. The images have a size of 250 × 250 pixels, covering an area of 10 × 10 cm. The circles represent the position of the inclusion.

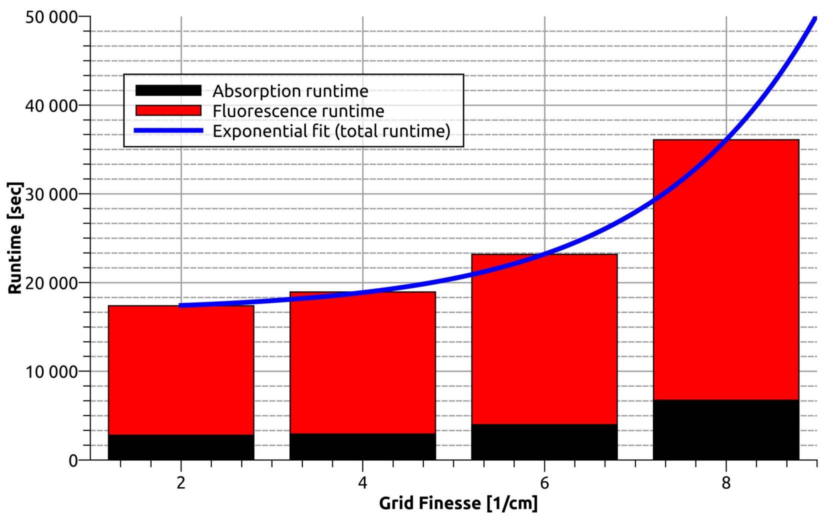

Figure 6. Simulation runtime as a function of the selected finesse parameter.

5×10^9 photons were simulated for the construction of the photon hitting density and the absorption simulation while 1×10^{10} photons were simulated for the final fluorescence simulation, using a finesse parameter of 5.

We also simulated the medium without the inclusions. As in section 3.2, we use the homogeneous image to normalize the inhomogeneous one and make the presence of the inclusion more evident [34, 36].

Figure 7 shows the results of the simulations after the mentioned normalization procedure was done and with a FFT low-pass filter applied in the same way as section 3.1. Each column represents one of the four different inclusion depths simulated. The first two rows are the absorption images for transmittance and reflectance geometries, while the last two are the fluorescence images for transmittance and reflectance geometries respectively.

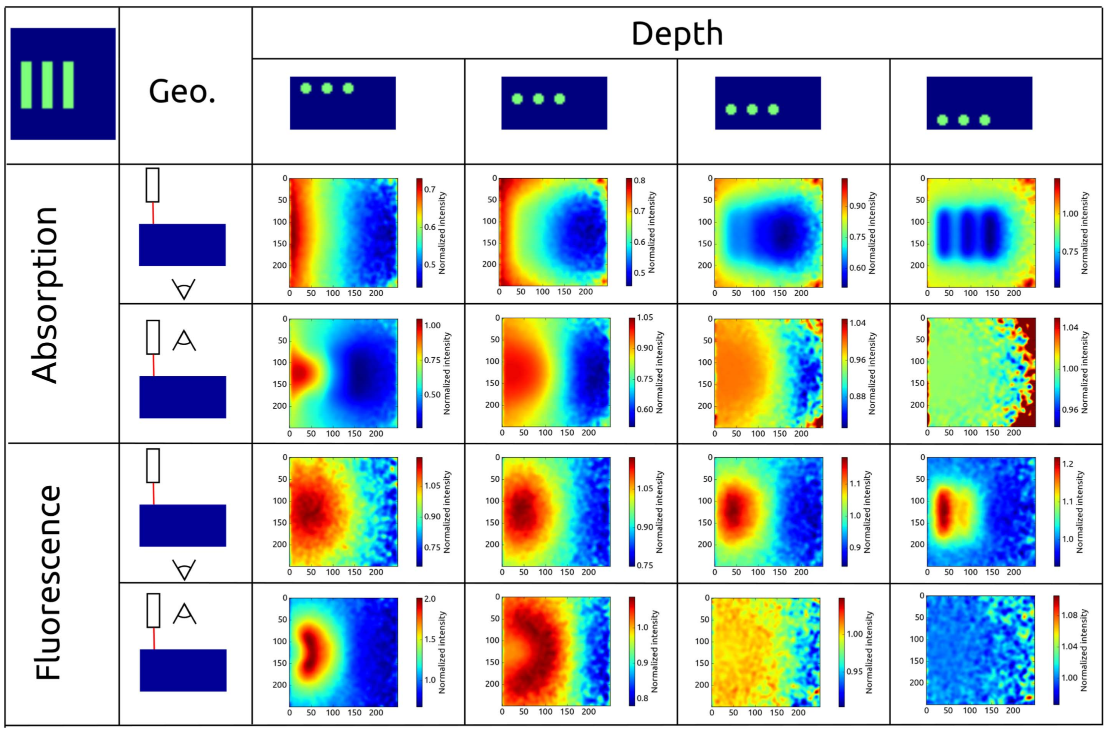

Figure 7. Simulations of three fluorescent and absorbent cylinders at different depths. Transmission and reflection. The images have a size of 250 × 250 pixels, covering an area of 10 × 10 cm.

The transmittance images show a clear increase in resolution when the inclusions get closer to the camera. It is noteworthy that in the case of fluorescence, the presence of inhomogeneities and its approximate location is evident in every position of the inclusion, despite the lack of resolution. Another interesting phenomena is that, for absorption, the location of the dip of image produced by the inhomogeneity shifts away from the source and inclusion when the latter moves towards the source.

The simulations also show how the reflectance images need more interpretation. In both the fluorescence and absorption cases the inhomogeneities are detected up to, and including, depth = 3 cm. However their shape and separation is not preserved. This is mainly an effect of the small region where useful data can be retrieved: too close to the source and its brightness destroys any modulation, too far away and noise starts to govern. This suggests that a scanning scheme may be needed for techniques that use fluorescence in real experiments. Also, it is important to take into account that the presence of fluorescence distributed in the bulk strongly masks the fluorescence generated by the inhomogeneity.

While a detailed analysis of diffuse fluorescence is beyond the scope of the present work, these simulations show the versatility of *CUDAMCFL*.

4. Additional notes about performance

The computing performance of GPUs has been in constant, exponential growth since their inception as general purpose computation devices. Following Moore's law, a high-end commercial GPU in 2016 has ~20 times more theoretical single-precision processing power than a GPU from 2007 [39]. This increase in computational power is what allows us to move from the limited and simplified *CUDAMC* code from Alerstam *et al* to a much more general and versatile one.

Nevertheless, given its simplicity, when running both codes in modern hardware *CUDAMC* is much faster than the present contribution. As an example, when running the same simulation for the comparison done in section 3.1, *CUDAMC* is close to 10 times faster, making it still the preferred choice when an homogeneous and non-fluorescent simulation is required.

All the simulations presented in the present contribution were carried out on a Nvidia GTX 750Ti. As briefly shown in section 3.3, a typical simulation of 5×10^9 absorption photons and 1×10^{10} fluorescent photons takes around 10 hours in such a system. However, this GPU is considered to be a low-end one with a theoretical single-precision power of only around 1 300 Gflops [40]. For reference, a higher-end, newer GPU like the GTX 1080 has 8500 Gflops [41]. Using a such a GPU would result in much reduced run-times and would open the possibility of more complex simulations with, for example, denser grids.

To investigate the performance characteristics of *CUDAMCFL* and to seek opportunities of future improvement we ran a short run (5×10^7 absorption photons, 1×10^8 fluorescent photons, finesse = 1) of *CUDAMCFL* through the `nvprof` profiler [42]. This utility allows developers to explore in which areas of their code the computing and memory resources are spent. The output is as follows.

```
                ==1363==NVPROF is profiling process 1363, command: ./cuda_fl
                            inclusion2-v2-3d-gs1-fast.mci
           ==1363== Profiling application: ./cuda_fl inclusion2-v2-3d-gs1-fast.mci
                               ==1363== Profiling result:
Time(%)      Time      Calls      Avg        Min        Max           Name
99.58%    71.0496s     2217    32.048ms   6.3998ms   148.63ms    MCd3D(MemStruct)
0.26%     185.72ms    12438    14.931us   1.2480us   63.620us   [CUDA memcpy DtoH]
0.09%     64.028ms     6003    10.666us   2.4000us   21.121us     [CUDA memset]
0.05%     35.759ms    42021     850ns      512ns     8.1280us   [CUDA memcpy HtoD]
0.02%     16.837ms     2001    8.4140us   6.6240us   9.4410us
                           LaunchPhoton_Global(MemStruct)
                                 ==1363== API calls:
Time(%)      Time      Calls      Avg        Min        Max           Name
61.78%    2.8e+03s     4218    661.22ms   3.4690us   26.7338s   cudaThreadSynchronize
36.48%    1.6e+03s    22443    73.386ms   3.8060us   26.6063s        cudaMemcpy
1.18%     53.2249s    32016    1.6624ms   2.9460us   26.5718s    cudaMemcpyToSymbol
0.54%     24.4294s    18009    1.3565ms   4.2110us   22.6391s        cudaMalloc
0.02%     900.31ms    18009    49.992us   3.9560us   419.35us         cudaFree
0.00%     75.886ms     6003    12.641us   5.2570us   111.10us        cudaMemset
0.00%     39.358ms     4218    9.3300us   4.8350us   38.731us        cudaLaunch
0.00%     2.0379ms     4218     483ns      186ns     9.1130us     cudaGetLastError
0.00%     1.8967ms     4218     449ns      158ns     191.31us    cudaConfigureCall
0.00%     1.3732ms     4218     325ns      151ns     8.8000us    cudaSetupArgument
0.00%     166.88us       91    1.8330us    124ns     68.760us   cuDeviceGetAttribute
0.00%     65.881us        1    65.881us   65.881us   65.881us    cuDeviceTotalMem
0.00%     22.471us        1    22.471us   22.471us   22.471us     cuDeviceGetName
0.00%     1.9430us        3     647ns      148ns     1.0530us    cuDeviceGetCount
0.00%      815ns          3     271ns      198ns      362ns         cuDeviceGet
```

It can be seen that, as expected, the overwhelming majority of computing time is spent in `MCd3D`, the kernel function that does the photon transport. However, it is noteworthy that `cudaThreadSynchronize()` is, by far, the most called API. This CUDA function is issued whenever we want to ensure that all threads are in sync and in *CUDAMCFL* it is used at the end of each transport simulation loop in order to 'wait' for all photons to be terminated. However, its significance is overstated as even while some thread may be waiting, most of them are running in a given time.

As the amount of scattering events that each photon will suffer is unknown and random, the runtime of each thread can differ quite significantly. In the present version of *CUDAMCFL* each fluorescence source (each voxel of the photon hitting density matrix) is treated as a new simulation and a call to `cudaThreadSynchronize()` is issued each time. In this manner some computing power is wasted while the faster running threads wait for the slower ones to finish. We expect to address this issue in future versions, but an important refactoring of the code is needed and it is not clear that the performance improvements would be very high.

5. Conclusions

We presented a novel software, *CUDAMCFL*, for the Monte Carlo simulation of heterogeneous and fluorescent turbid media. Even though it is based on previous developments, the modifications are widespread.

It has been shown that, within the limits of each respective reference, *CUDAMCFL* agrees well with the previously established *CUDAMC* code and the theory based on the diffusive approximation. Its dependence on the density of the matrices used for the computations and storage results has been studied and the performance implications analyzed. We also show, as an example of its versatility, simulation of a turbid medium with multiple inclusions of different absorption and fluorescence characteristics.

CUDAMCFL is offered as open source in the hope that it will be useful for researchers, and will allow its modification and improvement for everyone following the GPLv3 license.

Acknowledgments

Authors are grateful for financial support from CONICET, PIP 2013-2015, N 301 and CICPBA Grant FCCIC16.

Appendix. Pseudocodes

As a quick reference, in this appendix we present the main modifications done the transport routines in

CUDAMCFL over previous Monte Carlo algorithms in the form of language-agnostic pseudocode.

The actual implementation is available at the GitHub page [28].

A.1. Calculation of current voxel

```
// Calculation of current voxel
if (is inside PHD space) {
  // Inside space of 3D matrix
  // Index of the current voxel as a function of
   position
  index = index_of_current_voxel(p.x,p.y,p.z);

  // Store the new bulk descriptor
  new_bulk = bulk_info[index];
}
else {
  // Outside space of 3D matrix, assume inside
   homogeneous medium
  new_bulk = 0;
}

if (new_bulk ! = photon.bulkpos) {
    // If changing descriptor (different optical
   properties)
    reflected = Reflect(photon,new_bulk,2);
}
```

A.2. LaunchPhoton function

```
function LaunchPhoton()
{
    if (source is isotropic) {
      // Isotropic source, random position in
   voxel size

      // Random position around x, y, z, inside the
   voxel
      photon.x = input_x + random(voxel_size);
      photon.y = input_y + random(voxel_size);
      photon.z = input_z + random(voxel_size);

      // Random direction
      photon.dx = random();
      photon.dy = random();
      photon.dz = random();

      // Set weight to max unsigned int
      photon.weight = MAX_UINT;

      // If the bulk info is voxelized, we need to
   know where the photon is
      if (is using voxelize bulk description) {
        if (is inside space of Photon Hitting Den-
   sity) {
          index = index_of_current_voxel(p.x,p.y,
   p.z);
          photon.bulkpos = bulk_info[index];
          }
        else photon.bulkpos = 0;
      }
```

(Continued.)

```
      else photon.bulkpos = 0;

     }
    else {
      // Colimated source

      // Random position around x, y, z, gaussian beam
      sample_rad = input_fibre_diameter*random();
      sample_phi = 2*PI*random();

      photon.x = input_x + sample_rad*cos
   (sample_phi);
      photon.y = input_y + sample_rad*sin
   (sample_phi);
      photon.z = input_z;

      // Photon pointing inwards
      photon.dx = 0;
      photon.dy = 0;
      photon.dz = 1;

      // Set weight to max unsigned int minus loss
   due to reflection entering the bulk
      photon.weight = MAX_UINT − reflection_loss;

      // If the bulk info is voxelized, we need to
   know where the photon is
      if (is using voxelize bulk description) {
        if (is inside space of Photon Hitting Den-
   sity) {
          index = index_of_current_voxel(p.x,p.y,
   p.z);
          photon.bulkpos = bulk_info[index];
        }
        else photon.bulkpos = 0;
       }
        else photon.bulkpos = 0;
     }

      // If the bulk is multilayer, we need to know
   which layer the photon is
      if (is multilayer bulk) {
          // Found photon start layer
          photon.layer = current_layer;
      }
      else photon.layer = 0;
    }
```

A.3. Accumulate photon hitting density and retrieve bulk position

```
// Accumulate photon hitting density and
  retrieve bulk position
if (is inside PHD space) {
    // Inside space of 3D matrix
    // Index of the current voxel as a function of
  position
    index = integer_index_of_voxe(p.x,p.y,p.z)

    // Check for overflow and add
    if (fhd[index] + photon.weight < LLONG_MAX)
      fhd[index] + = photon.weight;
}
```

ORCID

N A Carbone https://orcid.org/0000-0002-5452-1165

References

[1] Tromberg B J, Pogue B W, Paulsen K D, Yodh A G, Boas D A and Cerussi A E 2008 *Med. Phys.* **35** 2443–51
[2] Vo-Dinh T 2002 *Biomedical Photonics Handbook* (Boca Raton, FL: CRC Press)
[3] Cerussi A, Shah N, Hsiang D, Durkin A, Butler J and Tromberg B J 2006 *J. Biomed. Opt.* **11** 044005
[4] Grosenick D, Hagen A, Steinkellner O, Poellinger A, Burock S, Schlag P, Rinneberg H and Macdonald R 2011 *Rev. Sci. Instrum.* **82** 1024302
[5] Poellinger A, Burock S, Grosenick D, Hagen A, Lüdemann L, Diekmann F, Engelken F, Macdonald R and H Rinneberg P M S 2011 *Radiology* **258** 409–16
[6] Intes X, Ripoll J, Chen Y, Nioka S, Yodh A G and Chance B 2003 *Med. Phys.* **30** 1039–47
[7] Wang L H, Jacques S L and Zheng L Q 1995 *Comput. Methods Programs Biomed.* **47** 131–46
[8] Alerstam E, Svensson T and Andersson-Engels S 2008 *J. Biomed. Opt.* **3** 060504
[9] Pfefer T J, Barton J K, Chan E K, Ducros M G, Sorg B S, Milner T E, Nelson J S and Welch A J 1996 *IEEE J. Sel. Top. Quantum Electron* **2** 934–42
[10] Pfefer T J, Schomacker K T, Ediger M N and Nishioka N S 2001 *IEEE J. Sel. Topi. Quantum Electron.* **7** 1004–12
[11] Boas D A, Culver J P, Stott J J and Dunn A K 2002 *Opt. Express* **10** 159–70
[12] Fang Q and Boas D A 2009 *Opt. Express* **17** 20178–90
[13] Margallo-Balbás E and French P J 2007 *Opt. Express* **15** 14086–98
[14] Fang Q 2010 *Biomed. Opt. Express* **1** 165–75
[15] Ren N, Liang J, Qu X, Li J, Lu B and Tian J 2010 *Opt. Express* **18** 6811–23
[16] Shen H and Wang G 2010 *Phys. Med. Biol.* **55** 947–62
[17] MCX/MMC http://mcx.sourceforge.net/cgi-bin/index.cgi
[18] TIM-OS https://sites.google.com/a/imaging.sbes.vt.edu/tim-os/ accessed: 2017-03-29
[19] Crilly R J, Cheong W F, Wilson B and Spears J R 1997 *Appl. Opt.* **36** 6513–9
[20] Liebert A, Wabnitz H, Żołek N and Macdonald R 2008 *Opt. Express* **16** 13188–202
[21] Péry E, Blondel W C P M and Thomas C 2009 *J. Biomed. Opt* **14** 024048
[22] Pfefer T J, Wang Q and Drezek R A 2011 *Comput. Methods Programs Biomed.* **104** 161–7
[23] Gerega A *et al* 2012 *J. Biomed. Opt* **17** 087001
[24] Milej D, Gerega A, Wabnitz H and Liebert A 2014 *Phys. Med. Biol.* **59** 1407–24
[25] Hochuli R, Powell S, Arridge S and Cox B 2016 *J. Biomed. Opt* **21** 126004
[26] Zhu C and Liu Q 2013 *J. Biomed. Opt* **18** 050902
[27] Fujita H, Ali M, Selamat A, Sasaki J and Kurematsu M (ed) 2016 *Trends in Applied Knowledge-Based Systems and Data Science* (Berlin: Springer)
[28] CUDAMCFL Github page https://nicocarbone.github.io/CUDAMCFL/ accessed: 2016-11-21
[29] Group's web page https://sites.google.com/site/grupoopticabiomedicauncpba/trabajos/codigos-fuente accessed: 2016-12-01
[30] Peters V G, Wymant D R, Patterson M S and Frank G L 1990 *Phys. Med. Biol.* **35** 1317–34
[31] Steinbrink J, Wabnitz H, Obrig H, Villringer A and Rinneberg H 2001 *Phys. Med. Biol.* **46** 879–96
[32] Rocco H O D, Iriarte D I, Lester M, Pomarico J A and Sandoval H F R 2011 *Int. J. Light Electron Opt* **122** 577–81
[33] Ishimaru A 1997 *Wave Propagation and Scattering in Random Media* (Oxford: Oxford University Press)
[34] Carbone N A, Baez G R, García H A, Serra M V W, Rocco H O D, Iriarte D I, Pomarico J A, Grosenick D and Macdonald R 2014 *Biomed. Opt. Express* **5** 1336–54
[35] Zhu X D, po Wei S, Feng S C and Chance B 1996 *J. Opt. Soc. Am.* A **23** 494–9
[36] Carbone N A, Rocco H O D, Iriarte D I and Pomarico J A 2010 *J. Biomed. Opt.* **15** 035002
[37] Li X D, O'Leary M A, Boas D A, Chance B and Yodh A G 1996 *Appl. Opt.* **21** 3746–58
[38] Milej D *et al* 2012 *Phys. Med. Biol.* **57** 6725–42
[39] GeForce 8800 GT specifications http://geforce.com/hardware/desktop-gpus/geforce-8800-gt/specifications accessed: 2016-11-21
[40] GeForce GTX 750 Ti specifications http://geforce.com/hardware/desktop-gpus/geforce-gtx-750-ti/specifications accessed: 2016-11-21
[41] GeForce GTX 1080 specifications http://geforce.com/hardware/10series/geforce-gtx-1080 accessed: 2016-11-21
[42] Profiler user guide http://docs.nvidia.com/cuda/profiler-users-guide/ accessed: 2016-11-21