MASSIVELY PARALLEL SIMULATOR OF OPTICAL COHERENCE TOMOGRAPHY OF INHOMOGENEOUS MEDIA

by

Mauricio Rodrigo Escobar Ivanauskas

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

in partial fulfilment of the requirements of the degree of

Master of Science

Department of Electrical and Computer Engineering

University of Manitoba

Winnipeg

Abstract

Optical coherence tomography (OCT) imaging is used in an increasing number of biomedical and industrial applications. A massively parallel simulator of OCT of inhomogeneous turbid media, e.g., biological tissue, could be used as a practical tool to expedite and expand the study of the physical phenomena involving such imaging technique, as well as, to design OCT systems with enhanced performance. Our work presents the open-source implementation of this massively parallel simulator of OCT to satisfy the ever-increasing need for prompt computation of OCT signals with accuracy and flexibility. Our Monte Carlo-based simulator uses graphic processing units (GPUs) to accelerate the intensive computation of processing tens of millions of photon packets undergoing a random walk through a sample. It provides computation of both Class I diffusive reflectance due to ballistic and quasi-ballistic scattered photons and Class II diffusive reflectance due to multiple scattered photons. Our implementation was tested by comparing results with previously validated OCT simulators in multilayered and inhomogeneous (arbitrary spatial distributions) turbid media configurations. It models the objects as a tetrahedron-based mesh and implements and advanced importance sampling technique. Our massively parallel simulator of OCT speeds up the simulation of OCT signals by a factor of 40 times when compared to it central processing unit (CPU)-based sequential implementation.

Acknowledgements

It is a pleasure to thank those who made this thesis and my graduate studies experience possible. First and foremost, my deepest gratitude to my advisor, Professor Sherif S. Sherif, his support and guidance helped me in every step of my studies and writing of this thesis. He has helped me shape the work presented here with patience and thoughtful assistance when most needed.

I would also like to show my gratitude to the members of my committee, Professor Parimala Thulasiraman and Professor Pradeepa Yahampath for taking the time to review my thesis and for their useful comments. A special thanks to Professor Ivan T. Lima, Electrical and Computer Engineering Department, North Dakota State University, for his generous and prompt suggestions. Thanks to Professor Ian Jeffrey, Electrical and Computer Engineering Department, University of Manitoba, for allowing me to use the CUDA capable GPU Tesla K40c in the research lab under his supervision. I also thank the academic, technical, and administrative staff at the Department of Electrical and Computer Engineering, University of Manitoba and especially to Ms. Amy Dario M.Sc. the department's Graduate Advisor.

I dedicate this thesis to my treasured mother, family, Simon and Gabriel, for their constant support and unconditional love. Thanks to my friends in Canada and abroad. I love you all dearly.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANSI	American National Standards Institute
API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DRAM	Dynamic Random Access Memory
GPU	Graphics Processor Unit
GSL	GNU Scientific Library
IS	Importance Sampling
MCML	Monte Carlo Simulation of Light Transport in Multi-Layered Turbid Media
MWC	Multiply-with-Carry
OCT	Optical Coherence Tomography
OCT-MPS	Optical Coherence Tomography Massively Parallel Simulator
PRNG	Pseudo Random Number Generator
RAM	Random Access Memory
RTE	Radiative Transfer Equation
SD-OCT	Spectral Domain Optical Coherence Tomography
SIMT	Single Instruction, Multiple Thread
SNR	Signal to Noise Ratio
SS-OCT	Swept Source Optical Coherence Tomography
TD-OCT	Time Domain Optical Coherence Tomography
TIM-OS	Tetrahedron Based Inhomogeneous Monte Carlo Optical Simulator

Chapter 1

Introduction

Optical Coherence Tomography (OCT) is a well-established high resolution, non-invasive imaging technique with ample biomedical and industrial applications [1] [2] [3] [4] [5] [6] [7]. It relies on measurement of backscattered light from cross-sectional light beams to provide high resolution images of a sample's internal structure. It has penetration depth of 2 to 3 mm and axial spatial resolution of between 1 μm to 15 μm [8], which is a higher resolution than ultrasound imaging but lower penetration.

OCT was conceived as a diagnostic tool in ophthalmology. Since then, many more applications of OCT have emerged. These applications include cancer detection in a variety of tissues, monitoring of coronary diseases, gastrointestinal imaging, and caries detection in dentistry [9]. Aside from its application in biomedical imaging, OCT has an increasing number of industrial applications. It can be used for non-destructive testing of polymers, ceramics, and art work.

Further studies of the underlying physical phenomena and design of novel OCT systems could be carried analytically or through experiments. However, a practical computer simulator of such systems could enable and/or facilitate such efforts. Most available simulators of OCT are limited in functionality and/or unable to manage in practical time the computational burden of simulating millions of photon packets.

Therefore, the development of a massively parallel simulator of OCT of inhomogeneous media using a graphics processing unit (GPU) would be useful. Such massively parallel simulator of OCT would reach significantly lower computation time to accelerate the study of different OCT physical phenomena or to design novel OCT systems with improved performance.

1.1 Organization of this thesis

Implementing a massively parallel OCT simulator requires an understanding of the operation of OCT systems and the optical properties of tissues. Therefore, in Chapter 2, we describe the principles of OCT imaging and the different types of available OCT systems. In Chapter 3 we explain concepts for simulating photon transport in tissue. Additionally, we give a description of the different equations governing the physics of light propagation in turbid media as well as the optical coefficients of tissues.

In Chapter 4 we define Monte Carlo (MC) methods to present a background on MC-based simulators for light transport in tissue and earlier implementations of OCT simulators. We include the details of the serial MC-based OCT simulator on which our massively parallel simulator is based on.

In Chapter 5 we present an in-depth study of the different random number generators available and techniques currently in use to generate pseudo-random numbers on deterministic systems with emphasis in parallel generation of pseudo-random numbers.

In Chapter 6 we review concepts for parallel computing and details on the hardware characteristics of graphic processor units (GPUs).

In Chapter 7 we give the implementation details of our massively parallel simulator of OCT (OCT-MPS) of inhomogeneous media. In this chapter we include:

1. A description of all functions in the algorithm

2. The memory utilization of the program
3. The process of parallel random number generation
4. Portability across different GPUs of our implementation

In Chapter 8 we present examples of simulations run on OCT-MPS. OCT-MPS was validated by comparing its results to the previously validated serial OCT simulator on which our implementation is based on. The example used for validation is a four layer medium used by Yao and Wang in [10] and Malektaji *et al.* in [11]. Following the serial implementation, there are two examples of OCT imaging simulations from two non-layered objects, a sphere inside a slab and a slab containing an ellipsoid and two spheres.

In final Chapter 9 we present the conclusions and suggested future work.

1.2 Thesis research contributions

We developed a massively parallel simulator of OCT (OCT-MPS) of inhomogeneous turbid media 40 times faster than previous similar simulators. We implemented it in graphic processing units (GPUs) using the Compute Unified Device Architecture (CUDA) platform and programming model by NVIDIA® with extensions of the C language.

The features implemented with OCT-MPS are:

1. A tetrahedron mesh to model inhomogeneous media as proposed in [12]. Meshes of tetrahedron form can model any arbitrary shaped media with any desired accuracy

2. An advanced importance sampling method introduced by Lima *et al.* to reduce the variance of Class I and Class II estimates [13]

OCT-MPS has been validated by comparing its results to previously validated simulators. The practicality of our simulator lies on the speed-up of at least 40 times achieved by simulating simultaneously thousands of photon packets. We present simulation results of OCT signals from arbitrary-shaped media which would take a minimum of 15 days to months to obtain using previous OCT simulators.

1.2.1 Publications related to this thesis

M. R. Escobar I., S. Malektaji, I. T. Lima, and S. S. Sherif, "Massively parallel simulator of optical coherence tomography of inhomogeneous turbid media," Submitted to *Biomedical Optics Express*, 2015.

Once accepted for publication, OCT-MPS will be made available in the public domain.

M. R. Escobar I., S. Malektaji, I. T. Lima and S. S. Sherif, "Accelerated simulation of optical coherence tomography of objects with arbitrary spatial distributions," *Proc. SPIE Photonics North 2014*, Montreal, QC, pp.928818-928818. International Society for Optics and Photonics, 2014.

Chapter 2

Optical Coherence Tomography

Optical coherence tomography (OCT) is an imaging technique with numerous biomedical and industrial applications. It has established itself as an important diagnostic tool in ophthalmology, as the eyes' soft and transparent tissues are well suited for light propagation. Additionally, OCT has proven efficient for a wide range of biomedical applications on different types of tissues, including hard tissues as the tooth [9]. Some medical applications of OCT include the assessment of coronary diseases, optical biopsy and cancer detection, and caries detection in dentistry [14] [15] [16] [17]. Furthermore, OCT has been used in industry for non-destructive testing of polymers, ceramics, and art objects [7]. In addition to being used for structural imaging, OCT has been used for functional imaging. Polarization sensitive OCT, Oximetry, differential absorption OCT and Doppler OCT are examples of functional OCT systems [18].

2.1 Optical imaging techniques and OCT

Optical imaging is a method to reproduce the physical structure of an object by radiation of electromagnetic wave for later collection and measuring of the back-reflected wave by some sensor [19]. The shape and physical properties of the object are obtained from the measured field. Such physical properties include among others the absorption and scattering coefficients of the wave. Well-known biomedical imaging techniques include microscopy/confocal microscopy, X-Ray tomography, magnetic resonance imaging

(MRI), ultrasound, and optical coherence tomography (OCT). Each technique provides different imaging resolution and penetration depth due to the different wave characteristics in use for each system. OCT uses near infrared non-ionizing light and fills the gap between confocal microscopy and ultrasound imagine as shown in Figure 2-1.

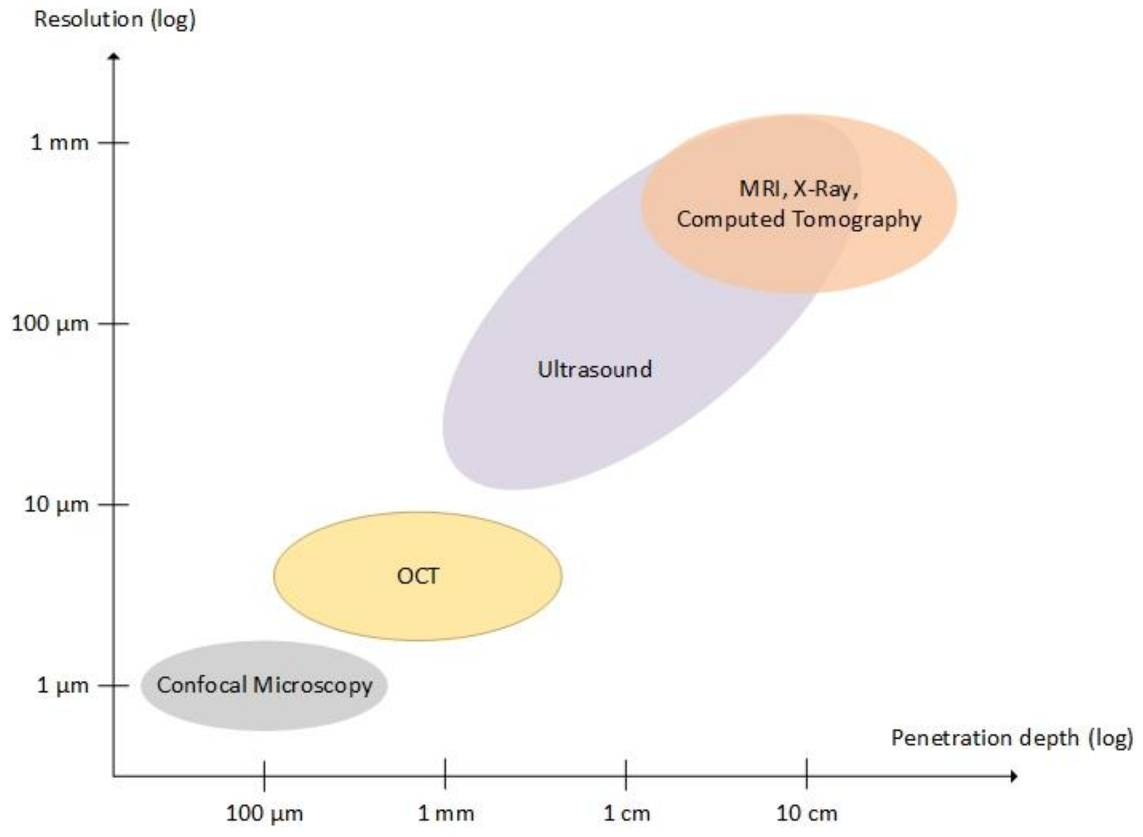

Figure 2-1 Resolution and penetration depth of different biomedical imaging techniques

OCT and ultrasound are often compared because ultrasound measures time delays in reflected echoes while OCT measures the intensity of light reflected from different depths of an imaged sample [20]. The axial resolution in OCT, between $1\ \mu m$ to $15\ \mu m$, is one to two orders of magnitude higher than ultrasound [8]. However, since light is strongly scattered and absorbed in most tissues, OCT imaging depth is restricted to $2 - 3\ mm,$ which is less than ultrasound imaging [16]. Contrasting OCT with confocal microscopy, the latter has a higher axial resolution which can reach $0.5\,\mu m$ [21]. However, confocal microscopy penetration depth is typically about $100\,\mu m$ which is considerably smaller than OCT [21].

2.2 Principles of OCT

Time delays of reflected light in OCT are in the order of the femtoseconds due to the high speed of light. This situation makes difficult to calculate these delays, making necessary the use of an optical interferometer. Additionally, the reflected optical signals from different depths are usually very weak. Interferometry is a central technique in all types of OCT systems. Typically, the interferometers in use are the Michelson or the Mach-Zhender, both in use with low coherence light source [22].

2.2.1 Coherence of light

The random nature of light is quantified primarily by its second order coherence function. Spatial and temporal coherence are two types of light coherence [23]. The temporal coherence is defined by the correlation between an optical field and a delayed version of itself. Spatial coherence describes the correlation between different points of an optical field in space. Typically, the light sources used in OCT systems are spatially coherent.

2.2.2 Low-coherence interferometry

Interferometry is a technique that makes possible measuring, with high sensitivity, the magnitude and echo time of back-scattered light from a sample. Back-scattered light from different depths of a sample interfere with the light that has travelled a known distance through a reference path in an interference pattern [23]. Interferometry measures the optical field instead of the light intensity.

2.3 Types of OCT systems

OCT imaging systems can be divided into Time Domain OCT (TD-OCT) and Frequency Domain OCT (FD-OCT). FD-OCT includes Spectral Domain (SD-OCT) and Swept Source (SS-OCT). Both TD-OCT and FD-OCT use interferometry to measure backscattered light from an object of sample.

2.3.1 Time Domain OCT

Time-domain OCT (TD-OCT) is a localized imaging technique based on low-coherence interferometry. The interference signal is obtained by mixing a wave that is scattered from an object or sample with a reference wave. Modifying the time delay in the reference wave path provides an axial scan, i.e., A-scan, direction and the sub-structure of a sample is scanned point-by-point. The system typically is enhanced with a movable sample stage for raster scanning in the transverse direction, i.e., B-scan, to obtain a cross-sectional or volumetric image of micro-structure of the sample.

A simplified diagram of a typical time-domain OCT system using a Michelson interferometer is shown in Figure 2-2. The incident light from the broadband light source is divided into a reference beam and a signal beam which travel different distances in the two arms of the system. The output is the summation of the reference field and the signal field, the detector measures the intensity of the output.

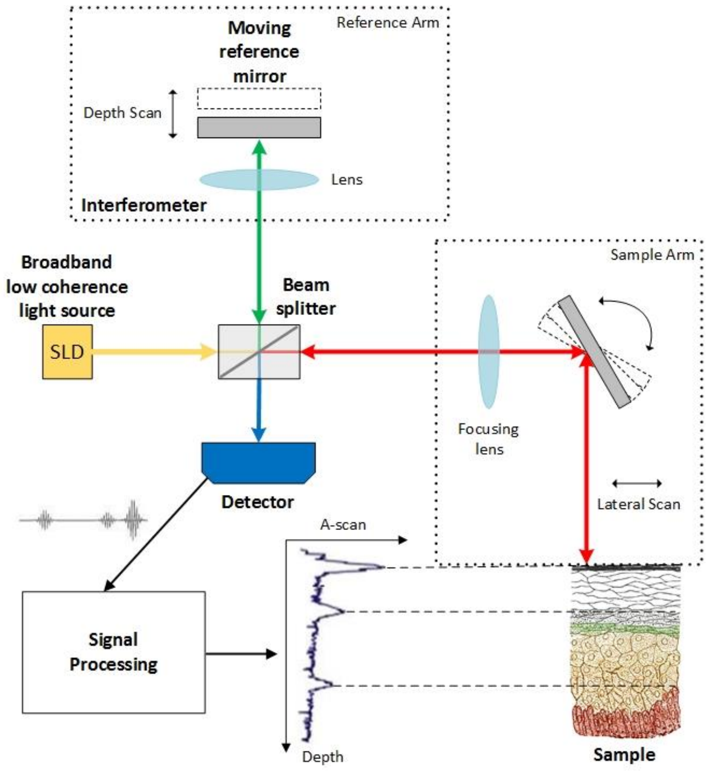

Figure 2-2 Diagram of a time-domain OCT system

2.3.2 Frequency Domain OCT

Frequency-domain or Fourier-domain OCT (FD-OCT) samples back-scattered light from all depth simultaneously. In FD-OCT the reference mirror is fixed so scanning a reference path length is not needed. This configuration makes scanning of larger samples faster. FD-OCT is subdivided in two types, i.e., spectral-domain OCT (SD-OCT) and swept-source OCT (SS-OCT).

2.3.2.1 Spectral Domain OCT

In Spectral-Domain OCT (SD-OCT), the photo-detector is an optical spectrometer consisting of several light collecting optical elements, i.e., grating and a CCD camera. The mirror in the reference arm is stationary and the light source is a broadband light similar to TD-OCT. A typical configuration of a SD-OCT is shown in Figure 2-3.

SD-OCT is a tomographic approach whereas TD-OCT is localized imaging method. Each A-scan in SD-OCT is obtained by detecting the spectrum of the summation of back-scattered light from the sample and the light reflected from the reference mirror by the CCD. Finally, the sample's depth structure is obtained by the inverse Fourier transform of the signal detected by a spectrometer.

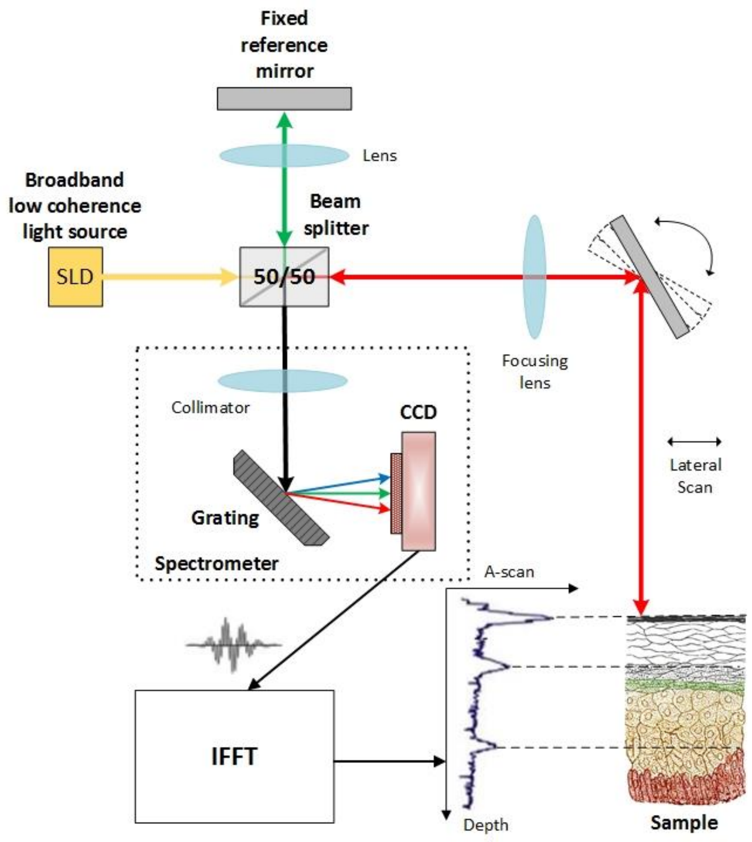

Figure 2-3 Diagram of a spectral-domain OCT

It is important to mention that in SD-OCT the axial resolution is proportional to the combination of the light source bandwidth, the spectrometer bandwidth, and the number of pixels in the CCD [24]. The design of SD-OCT systems presents many challenges. For example, the use of an array of detectors may produce phase washout caused by changes in the sample arm length during the pixel integration time. Additionally, it is difficult to implement balanced detection and polarization diversity.

2.3.2.2 Swept Source OCT

In Swept Source OCT (SS-OCT), also known as Optical Frequency Domain Imaging (OFDI) [25], a wavelength swept laser source and a photo-detectors are used as the light source and the detector respectively. SS-OCT has a simple mechanical design with a fixed reference mirror. As with previous OCT configurations, the light beam is divided by a beam splitter and then a fraction of the light is projected to the sample and the rest is guided to the reference arm. Finally, both signals from both arms interfere at the photo-detector. The axial reflectivity profile is obtained by computing the inverse Fourier transform of the signals sequentially acquired. In practice the detector output is digitized and sampled into a finite number of data points. A typical configuration of a SS-OCT is shown in Figure 2-4.

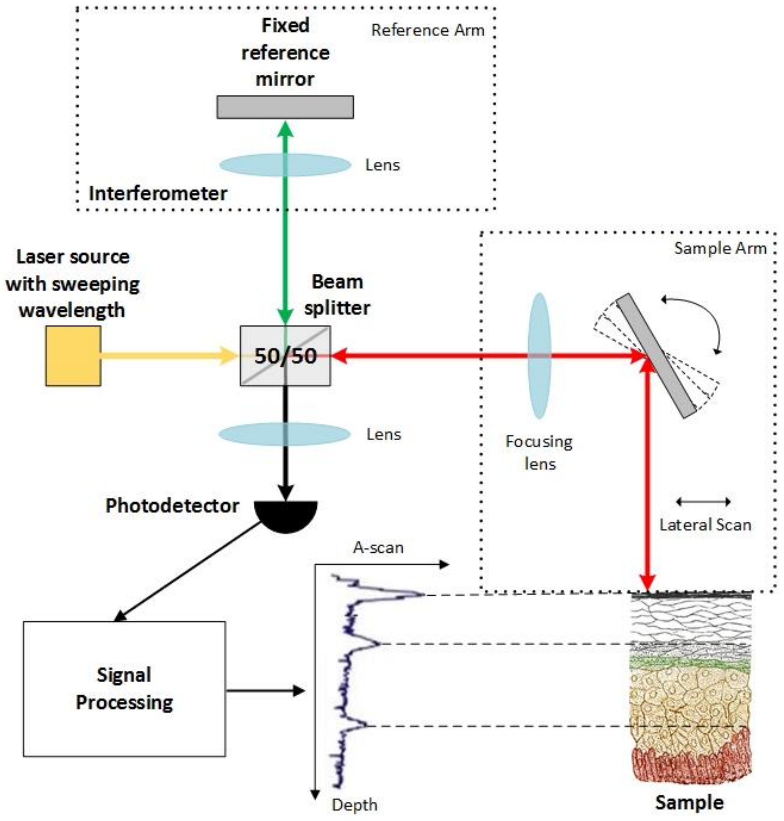

Figure 2-4 Diagram of a swept source OCT

2.4 Chapter summary

In this chapter we reviewed optical coherence tomography (OCT), its uses in biomedicine and industry, operation principles, and types of systems. The massively parallel simulator of OCT of inhomogeneous media presented in this thesis simulates time-domain OCT systems, i.e., TD-OCT.

Chapter 3

Mathematical description of photon transport in tissue

Biological samples influence the propagation of light through them in a well-studied fashion. Tissues have optical factors and coefficients that reflect this interaction. Scattering coefficient μ_s, absorption coefficient μ_a, refractive index n, and anisotropy factor g are the optical properties needed to simulate OCT signals from tissue.

Radiative Transport Equation (RTE), also known as Boltzmann's equation, is accepted as the standard model to describe analytically light transport in tissue. Another common approach is based on Maxwell's equations, which through proper approximations can derive to RTE [26]. Typically these integro-differential equations are only used for simple models due to their complexity. RTE could be solved by numerical techniques such as discrete-ordinate method [27], finite-difference method [28], or finite-elements method [29].

3.1 Radiative Transfer Equation

The Radiative Transfer Equation (RTE) is given by the following integro-differential expression [26] [30],

$$\frac{1}{C_0}\frac{\partial}{\partial t}I(r(t),\hat{s},t) + (\mu_s + \mu_a)I(r(t),\hat{s},t) + \frac{(\mu_s + \mu_a)}{4\pi}\int_{(4\pi)} p(\hat{s}.\hat{s}')I(r(t'),\hat{s}',t)d\Omega' + Q(r(t),\hat{s},t), \tag{3.1}$$

where $I(r(t), \hat{s}, t)$ is the intensity at position $r(t)$, expressing the power per unit area that flows in the direction $\hat{s}$ at the time t [26]. The scattering phase function is represented by $p(\hat{s}.\hat{s}')$ and $Q(r(t), \hat{s}, t)$ is a source term [30]. As mentioned before, solutions of RTE are only known for simple cases. Hence, approximations for RTE are obtained by the Diffusion Equation (DE).

3.1.1 Diffusion Equation

In the Diffusion Equation, the scattering directions are assumed to be isotropic and the scattering coefficient is assumed to be considerably higher than the absorption coefficient. The DE is given by the following partial differential equation (PDE) [30]:

$$\frac{1}{C}\frac{\partial}{\partial t}\Phi_d(r(t), t) + \mu_a\Phi_d(r(t), t) - D\nabla^2\Phi_d(r(t), t) = Q_c + Q_s, \tag{3.2}$$

where

$$D = \frac{C}{3(\mu_a + (1 - g)\mu_s)}. \tag{3.3}$$

D is a constant known as the diffusion parameter. C is the speed of light in the medium. $\Phi_d(r(t), t)$ is the average photon flux density (particles per second per unit area, also named fluence) [31] [30],

$$\Phi_d(r(t), t) = \int_{(4\pi)} I(r\,(t), \hat{s}, t)d\Omega. \tag{3.4}$$

Q_c and Q_s are the coherence field and the local source respectively. The DE equation is a PDE that is not always easy to solve. Here, the scattering can have any direction. Another model is the Kubelka-Munk, which considers only forward and backward scattering directions making it simpler.

3.1.2 Kubelka-Munk Equations

The Kubelka-Munk (K-M) equations describe the light propagation in a planar homogeneous medium which is illuminated from one side with a monochromatic light [30]. The K-M equations are given by [30],

$$-\frac{di}{dx} = -(S+K)i + Sj, \tag{3.5}$$

$$\frac{dj}{dx} = -(S+K)j + Si, \tag{3.6}$$

where x represents the medium depth, i the intensity of light propagating in the medium in forward direction, and j the intensity of light propagating in the medium in backward direction. S and K are respectively the scattering and absorption per unit thickness. A general solution of Eq. 3.5 and 3.6 is given by [30]

$$i = Asinh(bSx) - Bcosh(bSx), \tag{3.7}$$

$$i = (aA - bB)sinh(bSx) - (aB - bA)cosh(bSx), \tag{3.8}$$

where,

$$a = 1 + \frac{K}{S}, \tag{3.9}$$

$$b = (a^2 - 1)^{1/2}. \tag{3.10}$$

Constants A and B can be calculated according to the following conditions

$$i = I_0T;\ j = 0 \text{ for } x=0, \tag{3.11}$$

$$i = I_0;\ j = I_0R \text{ for } x=X, \tag{3.12}$$

where the transmittance T and reflectance R are given by

$$T = b(a \times \sinh(bSX) + b \times (\cosh(bSX))^{-1}, \tag{3.13}$$

$$R = \sinh(bSX)\,(a \times \sinh(bSX) + b \times (\cosh(bSX))^{-1}, \tag{3.14}$$

here, X is the full thickness of the medium. Note that the K-M model describes single and multiple backscattered radiation.

3.1.3 Beer-Lambert Equation

The simplest method to model light interaction in a single layer of tissue is the Beer-Lambert equation described by [32],

$$\mathrm{I(d)} = (1 - \mathrm{R_F})\mathrm{I_0}\exp(-\mu_t \mathrm{d}), \tag{3.15}$$

where $I(d)$ is the intensity of the light at depth d. $\mathrm{R_F}$ is the Fresnel reflection coefficient at the boundary between the tissue and the ambient medium. Finally, $\mu_t = \mu_s + \mu_a$ is the extinction coefficient obtained by adding the scattering and absorption coefficients, and $\mathrm{I_0}$ is the incident light intensity.

3.2 Optical properties of tissue

Most biological tissues are inhomogeneous with strong scattering and low absorption behavior in the presence of light. The inhomogeneity of tissues is due to the biological components of them such as cells, fiber structure, and organelles [33].

A couple of approaches have been proposed to model biological tissues. The first, describes a cell of tissue as a collection of microspheres and micro ellipsoids embedded in a finite background medium. This modeling approach is suited for solving with Maxwell's equations inside single cells [34]. The second approach models a larger volume of tissue with an ensemble of optically homogeneous particles uniformly suspended in an optically homogeneous background [35]. In this approach the particles can have any shapes or sizes. However, it has been shown that optical properties of randomly oriented particles can be modeled with spherical particles [36]. The choice of spheres for the shape of particles allows analytical calculation of optical coefficients by Mie theory for electromagnetic [37] or scalar wave scattering from spheres [38]. The following sections discuss the calculation of optical coefficients of tissues modeled with spherical particles inside a homogeneous background.

3.2.1 Refractive Index

The optical coefficients of tissue depend on the refractive-index of its homogeneous background, the refractive index of its particles and their sizes and concentrations. The refractive-index of the background $\bar{n}$ can be calculated by Gladstone-Dale equation [39]:

$$\bar{n} = \sum_{i=1}^{L} n_i f_i, \tag{3.16}$$

where L is the number of tissue components, and f_i is the volume fraction of a particular tissue component i, in which $\sum_{i=1}^{L} f_i = 1$. Therefore the refractive index of the background medium for soft tissue can be calculated as the weighted average of the refractive index of cytoplasm n_{cp} and the interstitial fluid n_{is},

$$\bar{n}_{bkg} = f_{cp} n_{cp} + (1 - f_{cp}) n_{is}, \tag{3.17}$$

where $\mathrm{f_{cp}}$ is the volume fraction of the fluid inside tissue cells [33]. Approximately 60% of the total fluid in a human soft tissue is contained in the intracellular compartment, i.e., cytoplasm, and 40% occupies the extracellular compartment, i.e., interstitial fluid [33].

The refractive index of cytoplasm $\mathrm{n_{cp}}$, is typically between 1.35–1.38 [40]. The refractive index of the interstitial fluid $\mathrm{n_{is}}$, is typically between 1.33-1.35 [41]. Assuming the values of $\mathrm{n_{cp}} = 1.37$, $\mathrm{n_{is}} = 1.35$, and $f_{cp} = 0.7$ refractive index of the background medium would be $\bar{\mathrm{n}}_{\mathrm{bkg}} \cong 1.36$ [33].

The refractive index of particles $\bar{n}_p$ can be modeled as the sum of the refractive index of the background $\bar{n}_{bkg}$ and the mean of the variation of the refractive indices of cell components $\overline{\Delta n}$ [33],

$$\bar{n}_p = \bar{n}_{bkg} + \overline{\Delta n}. \tag{3.18}$$

$\overline{\Delta n}$ can be calculated as the weighted average of differences between the refractive index of the cell components and their local background [33],

$$\overline{\Delta n} = f_f(n_f - n_{is}) + f_{nc}(n_{nc} - n_{cp}) + f_{0r}(n_{0r} - n_{cp}), \tag{3.19}$$

where $n_f, n_{is}, n_{nc}, n_{cp}$ and n_{or} are the refractive indices of cell fiber, interstitial fluid, nuclei, cytoplasm and organelles, respectively. f_f, f_{nc} and f_{or} are the volume fraction of fiber, nuclei and organelles in the solid portion of the tissue. The refractive index of nuclei is reported to be in the narrow range of 1.38–1.41. Assuming the approximation, $n_{or} = n_{nc} = 1.39$, Eq. (**3.19)** can be simplified into [33]

$$\overline{\Delta n} = f_f(n_f - n_{is}) + (1 - f_f)(n_{nc} - n_{cp}). \tag{3.20}$$

Fibrous-tissue fraction f_f depends on the type of tissue. The value of f_f is 0.7 in the dermis, 0.45 for the heart and between 0.02–0.03 for the non-muscular internal organs [35]. Therefore, f_f can vary between 0.02–0.7 for different type of tissues which will result in the variation range of 0.03–0.09 for the mean variation of the refractive index $\overline{\Delta n}$.

3.2.2 Probability distribution of spherical particles sizes

The probability distribution of the particle sizes is needed to calculate the optical coefficients. It has been shown that the size distribution of particles can be approximated by a skewed log-normal distribution with parameter m,

$$f(x) = C_m x^m \exp\left[-\frac{(\ln x - \ln x_m)^2}{2{\sigma_m}^2}\right], \tag{3.21}$$

where C_m, x_m and ${\sigma_m}^2$ are the normalizing constant, mean and variance of the distribution, respectively. The following not normalized distribution has been proposed for the volume fraction of particles with radius *r* by rewriting Eq. (**3.21)** [35] ,

$$\eta(r) = \frac{F_v}{C_m}(2r)^{3-D_f} \exp\left[-\frac{(\ln[2r] - \ln[2r_m])^2}{2{\sigma_m}^2}\right]. \tag{3.22}$$

The parameter D_f is called the fractal dimension. The normalizing factor C_m is given by,

$$C_m = \sigma_m \sqrt{2\pi}(2r_m)^{4-D_f} \exp\left[\frac{\left(4-D_f\right)^2 {\sigma_m}^2}{2}\right]. \tag{3.23}$$

Parameter F_v is the total volume fraction of particles,

$$F_v = \int_0^\infty \eta(r)\,\mathrm{d}r. \tag{3.24}$$

d_m is the mean of the particle size distribution. In the extreme case of infinite broad particle size distribution, this distribution turns into the ideal fractal distribution,

$$\lim_{\sigma_m \to \infty} \eta(r) \approx (2r)^{3-D_f}. \tag{3.25}$$

3.2.3 Optical coefficients of particle based tissue model

After calculating the refractive indices and particle size distribution, we can calculate the tissue's optical coefficients as described in the following subsections.

3.2.3.1 Absorption coefficient

The tissue's absorption coefficient can be due to absorption in a dispersive background media given by [36]

$$\mu_{a_{bkg}} = \frac{4\pi I(n_{bkg})}{\lambda}, \tag{3.26}$$

where λ is the wavelength of the light and $\mathrm{I}(\mathrm{n_{bkg}})$ is the imaginary part of the refractive index of the background medium. This absorption has non-zero value for any dispersive medium. In tissue modeling this kind of absorption in the background medium is usually neglected. Another cause of light absorption is due to the absorption in the particles. The absorption coefficient of tissue due to the particles with a continuous distribution of particle size is [36]

$$\mu_a = N \int_0^{\infty} \sigma_{abs}(r) f(r) dr, \tag{3.27}$$

where N and $f(r)$ are the particle concentration and the probability distribution of particles with radius r. $Nf(r)dr$ is the concentration of particles with radii in an interval $(r, r + dr)$ and $\sigma_{abs}(r)$ is the absorption cross-section of particle with radius r.

In the case of particles with M discrete sizes, the absorption coefficient is given by [35]

$$\mu_a = \sum_{i=1}^{M} \mu_a(r_i), \tag{3.28}$$

where $\mu_a(r_i)$ is the absorption coefficient due to particles with radius r_i

$$\mu_a(r_i) = \frac{\eta(r_i)}{v_i} \sigma_{abs}(r_i), \tag{3.29}$$

where $\sigma_{abs}(d_i)$ and v_i are the absorption cross-section and the volume of a particle with radius r_i and $\eta(d_i)$ is the volume fraction of particles with radius r.

3.2.3.2 Scattering coefficient

An important assumption in modeling the scattering of light waves in tissue is that scattered optical fields from different particles are independent of each other. The modeled tissue' scattering coefficient with continuous distribution of particle sizes is [36]

$$\mu_s = N\int_0^\infty \sigma_{sca}(r) f(r) dr, \tag{3.30}$$

where $\sigma_{sca}(r)$ is the scattering cross-section of a particle with radius r. In the case of particles with M discrete sizes, the scattering coefficient would be [35]

$$\mu_s = \sum_{i=1}^{M} \mu_s(r_i), \tag{3.31}$$

where $\mu_s(r_i)$ is the scattering coefficient of particles with radius r_i

$$\mu_s(r_i) = \frac{\eta(r_i)}{v_i}\sigma_{sca}(r_i), \tag{3.32}$$

where $\sigma_{\mathrm{sca}}(\mathrm{r_i})$ is the scattering cross-section of a particle with radius r_i.

3.2.3.3 Phase function

The phase function $P(\hat{S}\,,\hat{S}')$ is the distribution of the scattered light intensity in the direction $\hat{S}'$ if the incoming light has direction S. It is assumed that the tissue to be modeled is not birefringent, i.e., its refractive index does not depend on the direction of incoming light. Therefore the phase function of tissue, based on a spherical particles model, can be written as a function of an azimuthal angle, $0 \leq \phi < 2\pi$, and a longitudinal angle, $0 \leq \theta < \pi$,

$$P(\theta, \phi) = P_\Theta(\theta). P_\Phi(\phi), \tag{3.33}$$

where $P_\Theta(\theta)$ and $P_\Phi(\phi)$ are the longitudinal angle and azimuthal angle parts of the scattered intensity distribution. Tissue modeled with spherical particles $P_\Phi(\phi)$ has a uniform distribution in the interval $[0, 2\pi]$. The longitudinal part of the modelled tissue's phase function with particles' ensemble with continuous range of sizes is given by [36]

$$P_\Theta(\theta) = \frac{N\int_0^\infty p_\Theta(\theta,r)\sigma_{sca}(r)f(r)dr,}{N\int_0^\infty \sigma_{sca}(r)f(r)dr}, \tag{3.34}$$

where $p_\Theta(\theta, r)$ is the longitudinal scattering phase function of a particle with radius r. In the case of particles with M discrete sizes [36]

$$P_\Theta(\theta) = \frac{\sum_{i=1}^{M} \mu_s(r_i) p_\Theta(r_i,\theta)}{\sum_{i=1}^{M} \mu_s(r_i)}. \tag{3.35}$$

The anisotropy factor g is defined as the average of the longitudinal scattering angle. In the tissue modeled with discrete particle sizes, g is given by [35]

$$g = \frac{\sum_{i=1}^{M} \mu_s(r_i) g_i(r_i)}{\sum_{i=1}^{M} \mu_s(r_i)}, \tag{3.36}$$

where $g_i(r_i)$ is the anisotropy factor of particles with radius r_i,

$$g_i(r_i) = \int_0^\pi \theta \, P_\Theta(r_i, \theta) d\theta. \tag{3.37}$$

3.2.3.4 Henyey-Greenstein phase function

Due to the computational complexity for Monte Carlo simulations while using the phase function defined in Eq. (**3.34** and Eq. (**3.35**, Henyey-Greenstein phase function are proposed as an approximation of it. This phase function is commonly used in Monte Carlo simulation of light propagation in tissues [42]. The azimuthal part of the Henyey-Greenstein scattering phase function is uniform,

$$p_\Phi^{HG}(\phi) = \frac{1}{2\pi}, 0 \leq \phi < 2\pi. \tag{3.38}$$

The longitudinal part of Henyey-Greenstein phase function, as a function of direction cosine of longitudinal angle, is given by

$$\mathrm{p}_{\cos\Theta}^{\mathrm{HG}}(\cos\theta) = \frac{1-\mathrm{g}^2}{2(1+\mathrm{g}^2-2\mathrm{g}\cos\theta_{\mathrm{s}})^{3/2}}. \tag{3.39}$$

Where g is a given anisotropy factor which is typically about 0.9 for tissues.

3.2.3.5 Packing factor

All the above equations for optical coefficients are only valid under the assumption that scattered optical fields from different particles are independent of each other. Particles with high sizes or high total volume fraction can violate this assumption. To account for such cases, Twersky proposed a packing factor W [43] [35],

$$\mathrm{W} = \frac{(1-\mathrm{F_v})^4}{(1+2\mathrm{F_v})^2}. \tag{3.40}$$

Proposed by Bascom and Cobbold, a modification to this packing factor for particles with arbitrary shape and volume fraction η [44],

$$\mathrm{W_p}(\eta) = \frac{(1-\eta)^{\mathrm{p}+1}}{[1+\eta(\mathrm{p}-1)]^{\mathrm{p}-1}}, \tag{3.41}$$

where p is a shape dependent constant called the packing factor. For the spherical particles $p = 3$. To account for the inter-particles correlated scatterings, the modified volume fraction of the particles $\eta'(\mathrm{r})$ should be used instead of volume fraction $\eta(\mathrm{r})$ in calculation of optical coefficients,

$$\eta'(\mathrm{r}) = \frac{(1-\eta(\mathrm{r}))^{\mathrm{p}+1}}{[1+\mathrm{h}(\eta(\mathrm{r})-1)]^{\mathrm{p}-1}}. \tag{3.42}$$

3.2.4 Optical cross-sections and phase function of spherical particles

The scattering of the electromagnetic wave from a sphere is known as the Mie theory of scattering. In this research light is modeled as scalar waves. The relative refractive index of the sphere to the background is $\mathrm{n_r}$, i.e.,

$$\mathrm{n_p} = \mathrm{n_r n_{bkg}}. \tag{3.43}$$

It is shown in reference [38] that the scattering cross-section $\sigma_{sca}\ (r)$ of a sphere with radius *r* with relative refractive index n_r is

$$\sigma_{\mathrm{sca}}(\mathrm{r}) = \frac{\pi}{\mathrm{k}^2}\sum_{\mathrm{l}=0}^{\infty}(2\mathrm{l}+1)[1-2\eta_\mathrm{l}\cos(2\delta_\mathrm{l})+\eta_\mathrm{l}{}^2], \tag{3.44}$$

where k is the wavenumber of the planar incident wave. Constants η_l and δ_l are given by

$$\mathrm{S_l} = \eta_\mathrm{l}\mathrm{e}_\mathrm{l}^{\mathrm{i}2\delta_\mathrm{l}}, 0 \le \eta_\mathrm{l} \le 1, \tag{3.45}$$

where $\mathrm{S_l}$ is the complex phase shifts given by,

$$S_l \equiv \frac{\mathrm{A_l}}{\mathrm{B_l}} = -\frac{\mathrm{h}_\mathrm{l}^{(2)}(\beta)}{\mathrm{h}_\mathrm{l}^{(1)}(\beta)}\frac{\ln'\mathrm{h}_\mathrm{l}^{(2)}(\beta)-n\ln' j_l(\alpha)}{\ln'\mathrm{h}_\mathrm{l}^{(1)}(\beta)-n\ln' j_l(\alpha)}. \tag{3.46}$$

$\mathrm{h}_\mathrm{l}^{(1)}$, $\mathrm{h}_\mathrm{l}^{(2)}$ and j_l are spherical Hankel functions of first and second kinds and the spherical Bessel function of first kind of order l. Constant β is known as the size parameter and is given by

$$\beta = kr. \tag{3.47}$$

Constant α is given by

$$\alpha = n_r\beta. \tag{3.48}$$

The absorption cross-section of such sphere is given by

$$\sigma_{abs}(\mathrm{r}) = \frac{\pi}{k^2}\sum_{l=0}^{\infty}(2l+1)[1-\eta_\mathrm{l}{}^2]. \tag{3.49}$$

Due to the symmetry in spheres the azimuthal part of the scattering phase function is uniform,

$$p_{\Phi}(\phi, r) = \frac{1}{2\pi}, 0 \leq \phi < 2\pi. \tag{3.50}$$

The longitudinal angle part of the scattering phase function is given by

$$p_{\Theta}(\theta, r) = \frac{|f(k,\theta,\beta)|^2}{\sigma_{abs}(r)}, 0 \leq \theta \leq \pi, \tag{3.51}$$

where $f(k, \theta, \beta)$ is the scattering amplitude of sphere with size parameter β at far field and given by

$$f(k, \theta, \beta) = \frac{1}{2ik} \sum_{l=0}^{\infty} (2l + 1)[S_l(\beta) - 1] P_l(\cos\theta), \tag{3.52}$$

where P_l are the Legendre polynomials of order l.

3.3 Chapter summary

This chapter described the radiative transfer equation (RTE), a mathematical description of transport of light in tissue. Optical parameters of biological tissues, expressed in the RTE are the scattering coefficient, μ_s, the absorption coefficient, μ_a, the refractive index, n, and the anisotropy factor g. These optical parameters are obtained through modelling tissues as particles inside a background medium, as described in Section 3.2.3.

Chapter 4

Monte Carlo based simulators of light transport and of OCT

4.1 Monte Carlo methods

Monte Carlos (MC) methods are a family of algorithms used to solve, through repeated random sampling, many mathematical problems including integral equations [45]. MC methods are also used to simulate many physical phenomena.

MC methods are not the only technique to solve integral equations. Other techniques, e.g., method of moments and finite element method, require discretization of the problem in some form of grid or mesh. Such grids can easily grow to impractical sizes when dealing with optical wavelengths, which makes them unfeasible for optical problems.

The MC methods are a computationally efficient technique used as a numerical quadrature to estimate the value of a definite integral, or as a method to simulate and obtain values of a physical phenomenon. Both, Monte Carlo as quadrature [46] [47], and Monte Carlo as simulation [48] have been used for solving the RTE.

4.1.1 Accuracy of Monte Carlo methods

MC methods are considered the best algorithm for evaluation of high-dimension multidimensional integrals [49]. The standard deviation of the error of MC methods is reduced proportional to $1/\sqrt{N}$ regardless of dimension, where N is the number of points being evaluated. For example, to reduce the standard deviation of an MC algorithm by half, four times the number of samples is required [50]. Most quadrature rules have convergence rates reduced by a factor of $1/d$ as the dimension d increases [51]. Table 4-1 shows a comparison between MC and different quadrature rules to evaluate integrals.

Table 4-1 Convergence rate of Monte Carlo vs. quadrature rules

Convergence as function of n	
Monte Carlo	$\sqrt{n}$
Trapezoidal rule	$n^{-2/d}$
Simpson's rule	$n^{-4/d}$
Gauss' rule	$n^{-(2m-1)/d}$

Since MC convergence rate is independent of the dimension, there is a value of d above which it will converge to the solution much faster than other methods. The memory storage required for MC algorithms is minimal, as most times only the summation of estimates has to be stored.

The flexibility of MC algorithms allows this technique to be adaptable to solving a particular point in the domain as well as finding solutions on the whole domain. Additionally, MC algorithms are naturally suited to run in parallel.

4.2 Modeling arbitrary shaped media in MC simulation of light transportation in tissues

Introduced by Wang *et al.* in 1995, Monte Carlo simulation of light transport in multilayered turbid media (MCML) is restricted to multilayer media. Most OCT simulators based on MCML present the same limitation, which confines their applicability to multilayered media.

The first approach to model arbitrary shaped media was using cubic voxelization by Pferer *et al.* [52]. Here the medium is divided into a number of voxels with different optical properties. This modeling method has been used to simulate the propagation of light in different media such as human brain, skin, and trabecular bone [53] [54] [55].

Another technique to model turbid media, presented by Li *et al.* [56], uses standard geometrical building blocks, e.g., ellipsoids, cylinders, and polyhedrons. This method was used for simulation of bioluminescent light transport in a model mouse which consists of several segmented regions extended from a number of building blocks.

Cote and Vitkin [57], Margallo-Balbas and French [58], and Ren *et al.* [59] used triangular surface-mesh to model media for simulation of light transport. In this approach, the surface of homogeneous regions inside a turbid media is modeled by a surface mesh. Triangular surface-mesh allows approximation of complex tissue structures with any desired accuracy.

A high computational cost is involved in locating the path intersection of a photon with its enclosing surface mesh when using triangular surface-meshes. To overcome this, Fang proposed a Plücker coordinate system mesh-based, Monte Carlo method (MMCM) for fast computation of the location of such intersections [60]. In mesh-based approaches, to locate the intersection of the photon packets travelling path with the enclosing surface-mesh, the intersections with all planes comprising the surface-mesh is needed. If the enclosing surface-mesh is convex, the intersection with the minimum distance from the photon's location among these intersections would be the actual intersection. Tetrahedrons are convex volumes with minimum number of planes, therefore using them as a building block would minimize the computational cost of simulation.

In 2010 Shen and Wang [12] developed a tetrahedron-based inhomogeneous Monte-Carlo optical simulator (TIM-OS) where the media is modeled with tetrahedron mesh. The OCT simulator presented in this thesis, as in Reference [11], uses the tetrahedron mesh to model the arbitrary media as used in TIM-OS.

A broad review of methods to simulate light transport in turbid media can be found in the review paper by Zhu and Liu [61].

4.3 Earlier implementation of MC-based OCT simulators

In 1998 the first Monte Carlo (MC) based TD-OCT simulator was presented by Smithies *et al.* [62]. The authors of this paper simulated TD-OCT signals from a homogeneous slab media, e.g., an intralipid and blood model. In this paper, TD-OCT signals were shown to follow the extinction-single-backscatter model at shallow optical depths, where single scattered photons are dominant [62]. The extinction-single-backscatter models the imaging process of TD-OCT as an incoming beam getting continuously and exponentially attenuated according to the total extinction coefficient of the medium [62] [63]. Smithies *et al.* also shows that at deeper depth, where multiple scattered photons are dominant, the localization of backscattered photons is lost.

Smithies *et al.* studied the multiple scattering in terms of the spreading of the point spread function (PSF). In an imaging system, PSF is typically defined as the response of it to an illumination by a point source. They also studied the simulation of OCT signals using Gaussian beam source. To perform finite beam simulation, the initial transverse positions of photons are sampled randomly according to the intensity distribution of the source.

Next, the photons are directed towards the beam's geometrical focus. This method is known as *geometric-focus* [64]. The *geometric-focus* method is also used in Monte Carlo simulation of laser beam propagation in inhomogeneous tissues [65].

Spot-focus is another method which has been used for MC simulation of Gaussian beam light interaction with inhomogeneous media [64]. As in *geometric-focus*, in *spot-focus* the initial position of photons are sampled randomly. However, the photons' direction are chosen toward a random point within an area in the focal plane of the beam [64].

The third method used for MC simulation of finite beam light propagation in turbid media is using convolution as proposed by Wang *et al.* [66]. In this method the simulation results for a delta beam source are convolved with the finite size input beam to obtain the simulation result. This method is only applicable if the problem is shift invariant, i.e., the simulation results are the same for different position of the source such as simulation of light propagation or OCT inside a planar multilayered medium.

In Reference [62], the authors assumed a detection using two lenses as illustrated in Figure 4-1. Here, the probe has an angle α with the medium to be able to measure Doppler shifts. They implemented their detection system with a rejection process. In the rejection process the collected photon must satisfy a number of criteria on the position and their acceptance angle to contribute to OCT signals.

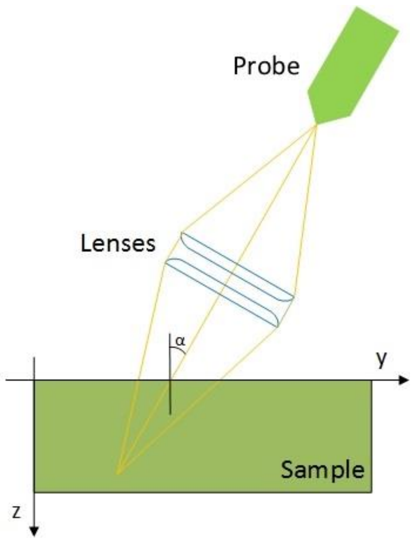

Figure 4-1 Illustration of the detection system used by Smithies *et al.* [62]

Tycho *et al.* used a 4F detection system as illustrated in Figure 4-2 [64]. In their 4F detection system the two lenses have equal focal lengths. The probe and the sample are placed in the focal length of their lenses. The distance between two lenses is twice the focal length.

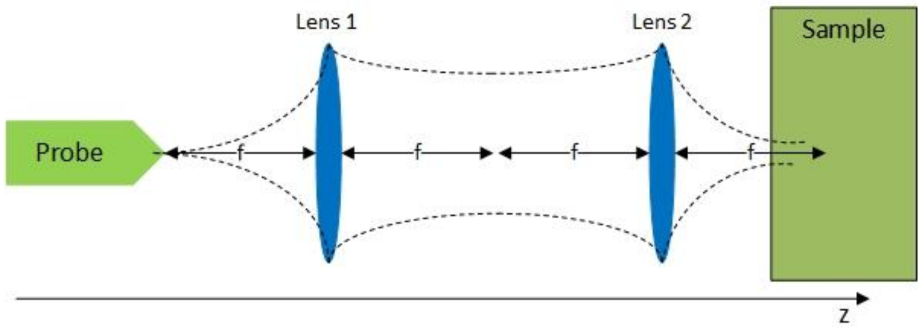

Figure 4-2 The 4F optical system used for detection by Tycho *et al*. [64]

In 1999, Yao and Wang developed a TD-OCT simulator for multilayered media [10]. They introduced Class I and Class II photons as photons backscattered from the target layer in the medium and photons backscattered from the rest of medium. They showed that these two classes of signal have different spatial and angular distributions [10].

In Reference [10], an importance sampling scheme is proposed. The authors biased the photons travelling away from the probe, towards the inverse of their travelling direction. The detection approach used by Yao and Wang differs from the detection systems in the previously mentioned OCT simulators. The authors placed a fiber probe in contact with the media for detection.

Later on, Lima *et al.* showed that the importance sampling method by Yao and Wang generated a systematic bias resulting in underestimation of TD-OCT signals [67]. They introduced an importance sampling scheme which generates unbiased results and reduces the computation time of calculating Class II diffusive reflectance in TD-OCT by up to three orders of magnitude, comparing with standard Monte Carlo simulation of TD-OCT. Later in 2012, they introduced a simulator with an importance sampling method which reduced the calculation of both Class I and Class II by two orders of magnitudes [13].

The main drawback of these simulators is their restriction to multilayered media. Kirillin *et al.* proposed an OCT simulator of non-planar multilayered objects in which they used different mathematical functions, as opposed to planes, for the boundaries of layers [68]. In Reference [69], Kirillin *et al.* used sinusoidal functions to define the upper and lower boundaries of layers in a model paper. This method has also been used to simulate the OCT images of the human enamel [9]. Here, the boundaries are modeled as non-parallel planes. In Reference [9], Geometric-focus is used to simulate the OCT signals assuming a Gaussian beam source. This method for modeling non-planar layers has also been used to model human skin in simulation of polarization-sensitive OCT [70].

All of the above OCT simulators are based on a MC simulator of light propagation in multilayered media.

4.4 MC simulation of OCT of inhomogeneous media

This section gives and in-depth description of the processes involved in the simulation of OCT signals, these processes are later on implemented in the massively parallel simulator of OCT presented in Chapter 7. The first step to simulate OCT signals of a medium is to model the propagation of light through that medium. In an OCT simulator, the light intensity is represented by photons which perform a random walk through the medium. Figure 4-3 shows this random walk (cut lines) in a tetrahedron-based medium.

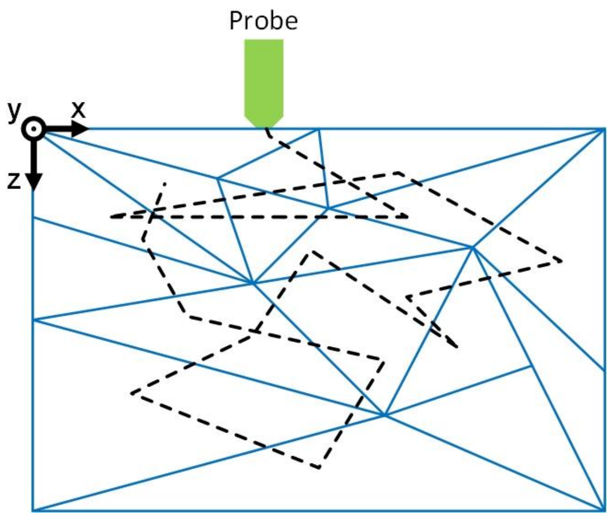

Figure 4-3 Photon packets random walk on a tetrahedron-based medium

The medium, consisting of a homogenous background slab and arbitrary shaped homogeneous regions, is divided into a number of tetrahedrons. The regions are defined by their optical parameters: scattering coefficient μ_s, absorption coefficient μ_a, refractive index n, and anisotropy factor g.

To start a simulation a large number of photons are successively launched into the medium. Each photon has a position and travelling direction defined with respect to a rectangular coordinate system placed of the top of the medium with its *z*-axis normal to the surface of medium as shown in Figure 4-3.

While the simulation is performed these photons packets undergo random walks in the medium. The photon packets position change as they propagate inside the medium. As well as their travelling directions due to scatterings, specular reflections, or refractions at the boundaries of regions inside the medium. The change of the travelling direction of photons, known as specular refraction, happens when they enter a region with different refractive index. Specular reflection is the photons' reflection at the boundary of regions.

The general flowchart of the photon packets random walk can be seen in Figure 4-4. The different stages a photon packet undergoes is explained in the following subsections.

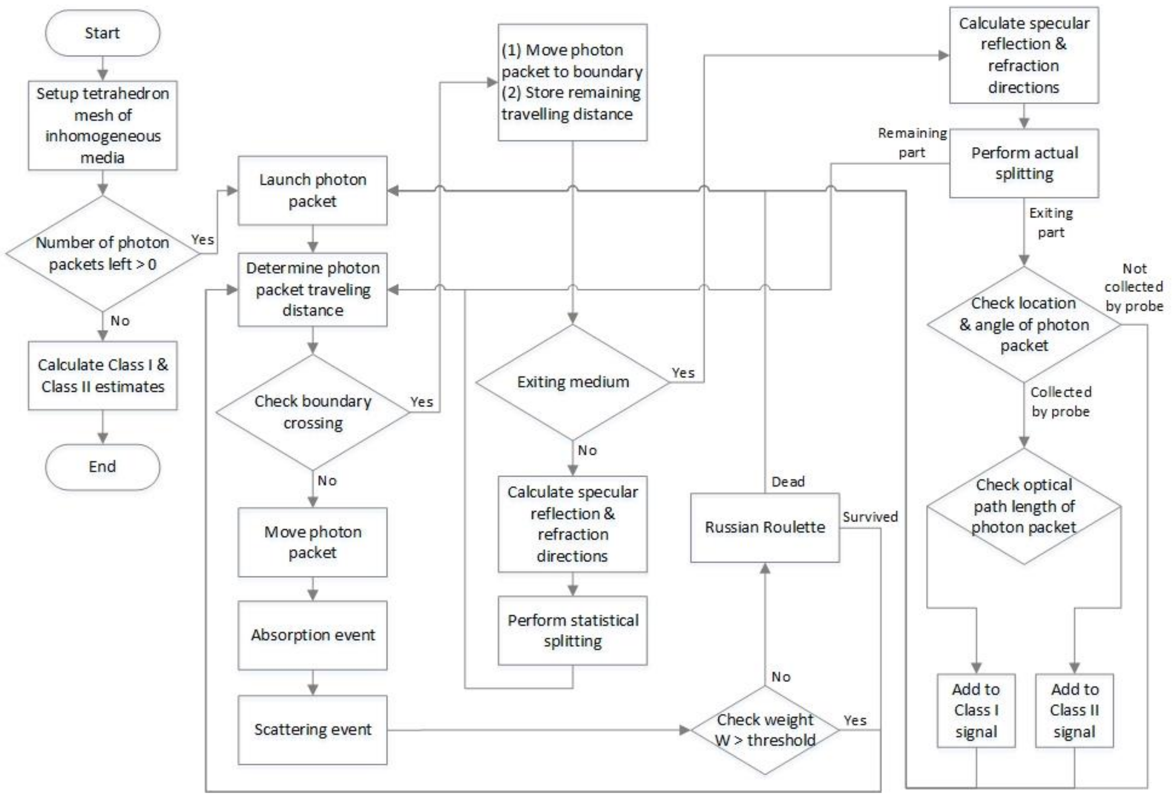
Start
Setup tetrahedron mesh of inhomogeneous media
Number of photon packets left > 0
Yes
No
Calculate Class I & Class II estimates
End
Launch photon packet
Determine photon packet traveling distance
Check boundary crossing
Yes
No
Move photon packet
Absorption event
Scattering event
(1) Move photon packet to boundary (2) Store remaining travelling distance
Exiting medium
Yes
No
Calculate specular reflection & refraction directions
Perform statistical splitting
Russian Roulette
Dead
Survived
Check weight W > threshold
Yes
No
Calculate specular reflection & refraction directions
Remaining part
Perform actual splitting
Exiting part
Check location & angle of photon packet
Not collected by probe
Collected by probe
Check optical path length of photon packet
Add to Class I signal
Add to Class II signal

Figure 4-4 Flowchart of photon packet tracing of an OCT simulator

4.4.1 Launch of a photon packet

All photons at the beginning have the same position equal to the light source position used to represent the delta beam source. The initial travelling directions are normal to the surface of the medium. Simulations can be performed by tracing one photon at the time. To reduce simulation time multiple photons are grouped in a photon packet which has a weight W factor assigned, $0 \leq W \leq 1$. This method is proposed by Kahn and Harris to reduce the variance of estimations and computation time in Monte Carlo simulation of particle transmission [71]. The initial weights of all photon packets are unity $\mathrm{W} = 1$. Absorption reduces the photon packets weight.

4.4.2 Determining the photon packet travelling distance

Following the launch, a photon packet travels a free-path after where it undergoes absorption and scattering. This free-path is the distance that photon packets travel between consequent interactions. The locations where absorption and scattering events happen is known as interaction sites.

The cumulative distribution function (CDF), of the free-path S, is given by

$$P\{S < s\} = 1 - \exp(-\mu_t s), \tag{4.1}$$

where μ_t is the total extinction coefficient of the regions that photon packets are travelling in.

Following the free-path sampling, the photon packets should travel the distance equals to this free-path in its travelling direction. However, if the photon packet hits the boundary of the enclosing region during travelling its free-path, it will undergo specular reflection and refraction.

4.4.3 Photon hitting a boundary of a tetrahedron

Specular reflection and refraction on a photon packet happen when they enter a region with a different refractive index. At each step it has to be checked if the photon packet's path and the enclosing tetrahedron intersect.

Let the photon packet position be $\hat{\mathrm{p}}$, its direction $\hat{\mathrm{u}}$, and its free-path length s. Therefore any point $\hat{\mathrm{p}}'$ on the photon's path satisfies $\hat{p}' = \hat{p} + \hat{u}t$ where $\mathrm{t} \in [0, \mathrm{s}]$. Let $\hat{n}.\hat{u} + d = 0$ be the equation of any plane of the enclosing tetrahedron, where $\hat{n}$ is any point on this plane, $\hat{u}$ is its inward normal unit vector, and d is its minimum distance from the origin. The distance from the photon's current position to this facet in the direction of photon's travelling direction is given by

$$\mathrm{t} = -\frac{\hat{\mathrm{n}}.\hat{\mathrm{p}} + d_b}{\hat{\mathrm{n}}.\hat{\mathrm{u}}}. \tag{4.2}$$

Let d_b be the minimum positive valued t among four tetrahedron facets. The photon's path and the enclosing tetrahedron intersect if $d_b > s$. If $d_b < s$ the photon remains in the same tetrahedron in which it moves to the interaction site and undergoes absorption and scattering.

If the photon hit the boundary of two tetrahedrons it may undergo specular reflection and refraction if the refractive indices of them differ. Prior to performing the specular reflection and refraction, the photon is moved to the boundary of the two tetrahedrons, i.e.,

$$\hat{p} = \hat{p} + d_b\hat{u}, \tag{4.3}$$

and the remaining of its sampled free photon path, s, is stored in a variable s_{left}, as follows

$$s_{left} = (s - d_b) \times \mu_t. \tag{4.4}$$

Next the specular reflection and refraction is performed.

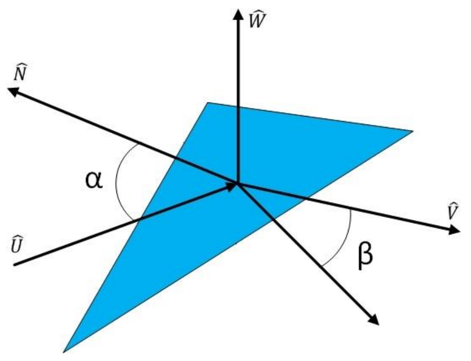

Figure 4-5 Geometrical representation of vectors and angles involved in reflection and refraction

4.4.3.1 Specular reflection and refraction

Figure 4-5 shows the geometrical presentation for reflection and refraction inside a tetrahedron. Let the photon packet's direction be $\hat{u}$ and the inward normal vector of the tetrahedron facet which photons hit be $\hat{n}$. The cosine of the incident angle α is given by

$$\cos\alpha = |\hat{n}.\hat{u}|. \tag{4.5}$$

The reflection direction $\hat{u}_{reflected}$ is given by

$$\hat{u}_{reflected} = 2\cos(\alpha)\,\hat{n} + \hat{u}. \tag{4.6}$$

The refraction direction $\hat{u}_{refraction}$ is given by

$$\hat{u}_{refracted} = \frac{\sin(\beta)}{\sin(\alpha)}(\cos(\alpha)\,\hat{n} + \hat{u}) - \cos(\beta), \tag{4.7}$$

where β is the refraction angle given by the Snell's law

$$\sin(\beta) = \frac{n_1}{n_2}\sin(\alpha), \tag{4.8}$$

where n_1 is the refractive index of the enclosing tetrahedron and n_2 is the refractive index of the next tetrahedron that the photon packet enters. If α is bigger than the critical angle arcsin $\left(\frac{n_2}{n_1}\right)$, total internal reflection happens. For total internal reflection the photon packet is completely reflected from the boundary without refraction. Otherwise portion of photon packet will be reflected with the direction $\hat{u}_{reflected}$ and the rest will be transmitted to the next tetrahedron with the refracted direction $\hat{u}_{refraction}$. A statistical splitting is performed, i.e., splitting the photon packet will be reflected with the probability R, or will be refracted with the probability $1 - R$. R is the Frensel's coefficient given by,

$$\mathrm{R} = \frac{1}{2}(\sin(\alpha)\cos(\beta) - \cos(\alpha)\sin(\beta))^2 \times$$

$$\left[\frac{(\cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta))^2 + \left(\cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)\right)^2}{\left(\sin(\alpha)\mathrm{cosb}(\beta) + \cos(\alpha)\sin(\beta)\right)^2\left(\cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta)\right)^2}\right]. \tag{4.9}$$

Whenever a photon packet enters the ambient medium, tracing of the refracted part is not needed. Therefore actual splitting can be used. In this case the photon will be reflected with the new weight $W = (1 - R)W$ and the refracted part that exits the medium will be stored to check if it contributes to OCT signals.

After performing specular reflection and refraction since the photon has not finished the sampled step size, the remaining part stored in s_{left} calculated according to Eq. (4.4**)** should be continued. The new step size will be calculated using s_{left} as follows

$$\mathrm{s} = \frac{\mathrm{s_{left}}}{\mu_\mathrm{t}}, \tag{4.10}$$

where μ_t is the extinction coefficient of the enclosing tetrahedron. It can be shown [48] that the CDF of the total step size of the photon through different regions will be

$$\mathrm{P}\{\mathrm{S} \leq \mathrm{s_{sum}}\} = \exp\left(\sum\nolimits_\mathrm{i} \ \mathrm{s_i}\, \mu_\mathrm{t}^\mathrm{i}\right), \tag{4.11}$$

where $\mathrm{s_{sum}} = \sum_{\mathrm{i}} \mathrm{s_i}$ is the total step size and $\mathrm{s_i}$ is the step size in the region i with total extinction coefficient $\mu_{\mathrm{t}}^{\mathrm{i}}$.

4.4.3.2 Absorption and scattering of photons

As mentioned before, whenever a photon packet reaches interaction sites it undergoes absorption and scattering. The portion of the photon which will be absorb ΔW is given by

$$\Delta W = W \times \frac{\mu_a}{\mu_t}, \tag{4.12}$$

where μ_a and μ_t are the absorption and extinction coefficient of the enclosing tetrahedron. Therefore the new weight of the photon will be $W = W - \Delta W$.

In case of scattering events, the direction of the photon will change randomly according to the Henyey-Greenstein scattering phase function described in Section 3.2.3.4.

4.4.3.3 Photon termination with Russian roulette

The tracing of a photon ends whenever a photon packet exits the medium or gets collected by the probe. However, as it propagates in the media, its weight decreases as well as its effect on the estimates. Therefore, tracing of it can be stopped if its weight falls below a certain threshold. This threshold is usually set to 10^{-4}. To maintain the unbiased estimation *Russian roulette* method is used to stop photon packet's tracing. In *Russian roulette*, a photon packet would be given a chance, with probability $\frac{1}{m}$ to survive or its tracing will be ended with probability $1 - \frac{1}{m}$. If the photon survives, its weight will be increase from W to mW. The constant m is typically 10 in OCT simulations.

4.4.4 Estimation of Class I and Class II OCT signals

All photon packets exiting the media to the ambient medium contribute to OCT signals only if they are collected by the probe. Three types of photons collected by the probe constitute TD-OCT signals; i.e., ballistic photons, quasi-ballistic photons, and multiple scattered photons. Ballistic photons have undergone only one scattering event. Quasi-ballistic photons have undergone multiple scattering events within the coherence length of the optical source. Multiple scattered photons have undergone scatterings beyond the coherence length of the optical source.

Class I diffusive reflectance is represented by ballistic and quasi-ballistic photons. The multiple scattered photons are grouped as Class II diffusive reflectance. Class II diffusive reflectance is the main source of error in OCT imaging and the fundamental limit of OCT imaging depth [63] [72].

A fiber probe is typically used as detector. As in reference [10], it has a radius $\mathrm{d_{max}}$ and acceptance angle θ_{max}. Spatial-temporal indicator functions are used to classify the photons into Class I and Class II photons. Class I diffuse reflectance spatial-temporal indicator function is,

$$I_1(z,i) = \begin{cases} 1, & l_c > |\Delta s_i - 2z_{max}|, r_i < d_{max}, \theta_{z,i} < \theta_{max}, |\Delta s_i - 2z| < l_c \\ 0, & otherwise \end{cases}, \tag{4.13}$$

where $\mathrm{l_c}$ is the coherence length of the source, $\mathrm{r_i}$ is the distance of the $\mathrm{i^{th}}$- reflected photon packet from the probe, $\Delta\mathbf{s_i}$ is the optical path of $\mathrm{i^{th}}$ photon packet, $\theta_{\mathrm{z,i}}$ is the angle of the photon packet direction with the z-axis, and $\mathrm{z_{max}}$ is the maximum depth reached by the photon packet. Similarly Class II diffuse reflectance $\mathrm{I_2}$ is,

$$I_2(z,i) = \begin{cases} 1, & l_c < |\Delta s_i - 2z_{max}|, r_i < d_{max}, \theta_{z,i} < \theta_{max}, |\Delta s_i - 2z| < l_c \\ 0, & otherwise \end{cases}. \tag{4.14}$$

Class I diffusive reflectance $R_1(z)$, and Class II diffusive reflectance $R_2(z)$, at depth z, is the weighted mean of these indicator functions,

$$R_{1,2}(z) = \frac{1}{N}\sum_{i=1}^{N} I_{1,2}(z,i)L(i)W(i), \tag{4.15}$$

where N is the number of simulated photon packets, L(i) is a likelihood ratio used in the importance sampling scheme, and $W(i)$ is the weight of the i^{th} photon. An estimate of the variance of these estimations is as follows,

$$\sigma^2{}_{1,2}(z) = \frac{1}{(N-1)}\sum_{i=1}^{N}[I_{1,2}(z,i)L(i)W(i) - R_{1,2}(z)]^2. \tag{4.16}$$

4.4.5 Importance sampling for reducing computational time of OCT signals

The group of techniques used to reduce the variance of Monte Carlo estimations using the same number of statistical samples is known as importance sampling. It can be used to reduce the computational cost of a result with a required accuracy. Importance sample can be used in both Monte Carlo as quadrature and Monte Carlo simulations. However, it has to be tailored to each particular estimation.

Turbid tissue has a high anisotropy factor g. Hence, light most likely propagates in the forward direction inside it. Due to this, the probability of backscattering is very low which will result in the necessity of a very large number of photon to estimate OCT signals. The probability of these back scattering events could be increased by properly biasing the scattering direction. The importance sampling technique used in this simulator is similar to the one described in [13].

If the travelling direction of the photon is away from the probe ($u_z > 0$), to increase the probability of its detection, it will be biased towards the position of the probe $\hat{v}$. The geometry of the importance sampling, vectors and angles involved in it can be seen in Figure 4-6.

The following probability density function (PDF) is used to sample the biased longitudinal scattering angle θ_B in the first biased scattering

$$p^{B}_{\cos(\Theta)}(\cos\theta_B) = \begin{cases} \left(1 - \frac{1-a}{\sqrt{a^2+1}}\right)^{-1} \frac{a(1-a)}{(1+a^2-2a\cos\theta_B)^{3/2}}, & \cos\theta_B < [0,1] \\ 0, & otherwise \end{cases}, \tag{4.17}$$

where a is the given bias coefficient in the range $[0,1]$.

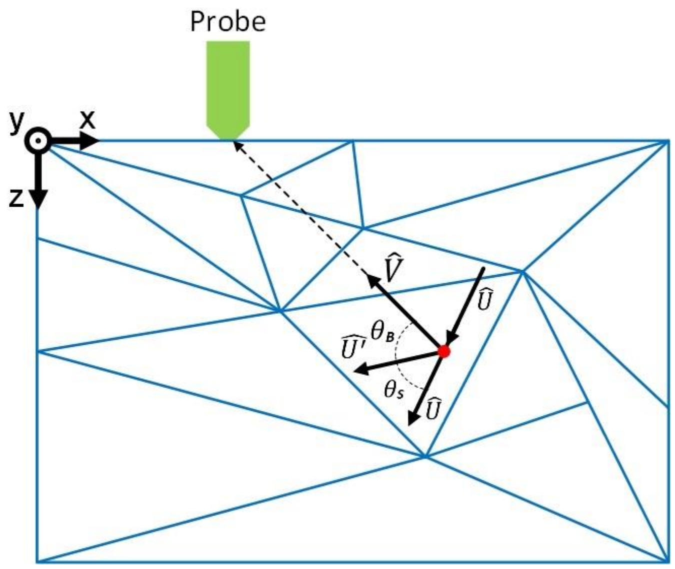

Figure 4-6 Geometry of Importance Sampling method used to reduce the variance of OCT signals estimations

The process described will create biased estimates. To keep the diffusive reflectance estimates unbiased, a compensating likelihood value is assigned to this biased scattering event,

$$\mathrm{L}(\cos\theta_B) = \frac{p^{HG}_{\cos(\Theta)}(\cos\theta_s)}{P^{B}_{\cos(\Theta)}(\cos\theta_B)} \tag{4.18}$$

$$= \frac{1-g^2}{2a(1-a)}\left(1-\frac{1-a}{\sqrt{a^2+1}}\right)\left(\frac{1+a^2-2a\cos\theta_B}{1+g^2-2g\cos\theta_s}\right)^{3/2}.$$

After the first biased scattering the photon packet can undergo unbiased scatterings with probability p or other biased scattering with probability $1-p$. In the case of biased scattering the longitudinal scattering angle is sampled from a Henyey-Greenstein phase function that is oriented towards the actual position of the probe with anisotropy factor a. The following likelihood function is assigned for both biased and unbiased scattering events,

$$\mathrm{L}(\cos\theta_B) = \frac{p^{HG}_{\cos(\Theta)}(\cos\theta_s)}{p \times P^{B}_{\cos(\Theta)}(\cos\theta_B) + (1-p) \times p^{HG}_{\cos(\Theta)}(\cos\theta_s)}. \tag{4.19}$$

When a photon packet is collected by the probe, exits from the medium or it is absorbed by that medium, the tracing of that photon packet ends and the total likelihood of the photon packet is calculated. This total likelihood L is the product of all likelihood values assigned to all scattering events the photon packet undergone. If $\mathrm{L} < 1$, another packet with initial likelihood value of $\mathrm{L}' = 1 - \mathrm{L}$ will be launched from the previous first scattering event position with a direction sampled from the unbiased Henyey-Greenstein phase function [8].

4.5 Chapter summary

In this chapter we provided a brief description of Monte Carlo (MC) methods before getting into an extensive literature review of many simulators of light transport in inhomogeneous media, in addition to simulators of OCT of layered media. Section 4.4

described in depth the MC-based simulator of OCT of inhomogeneous media that our massively parallel simulator, main contribution of this thesis, is based upon.

Chapter 5

Parallel pseudo-random number generators

Since random numbers play a very important role in Monte Carlo methods, i.e., they represent the samples in the problems addressed by MC, in this chapter we present a review of pseudo-random number generators with an emphasis in parallel generation of pseudo-random numbers.

A truly random number generator, i.e., one obtained from stochastic physical sources as atomic decay or atmospheric phenomena, is complicated and usually costly to build using deterministic systems such as today's digital computers [73]. Nevertheless, these computers are the main tool used for solving MC experiments.

Several techniques were developed to produce high quality pseudo-random numbers which satisfy all statistical tests and, within certain constrains, appear to be a sequence of numbers with no coherence between each other.

A group of independent random numbers are those uncorrelated and without linear pattern within that sequence of numbers. Thus, in a sequence of random numbers, there is no way to predict what the next number is and the occurrence of a pair of same numbers next to each other is highly unlikely. Lack of coherence and uniform distribution are desirable characteristics of most pseudo-random number generators.

A long *period* is another characteristic of high quality pseudo-random generators. The *period* is defined as how long the generator runs without failing on any of the previous requisites, i.e., no-coherence and distribution. This characteristic is especially important in big dimension problems which consume large number of random numbers, i.e., typically ten thousands and above random numbers. MC methods are typically used to solve these type of high dimension problems.

Every random number generator has a certain first state, known as *seed state*, from which the rest of the sequence is obtained. There are "good" seed states which will generate statistically good sequences of random numbers, and there are "weak" seed states which will produce statistically poor sequences of random numbers.

There are many additional desired characteristics for high quality generators, e.g., portability and repeatability/predictability. It is important for cross-validation that experimental results are reproducible many times with comparable outputs and known distribution, i.e., repeatability, and in as many different systems as desired, i.e., portability. Performance of the generator is very important to consider since most systems are limited by the generation of random numbers speed throughput.

Besides pseudo-random numbers, quasi-random numbers are another type of random numbers. Quasi-random numbers have very low discrepancy, they are evenly distributed with certain specific patterns in their outputs and are generally suitable for cryptography applications. Quasi-random numbers fall outside the scope of this work.

5.1 Types of pseudo-random number generators

There are several techniques to generate pseudo-random numbers in a deterministic fashion. Nevertheless, all generators are built on the same concept of producing a random number based on a subset of previous numbers by recurrence.

5.1.1 Sequential pseudo-random number generators

The common standard characterization methods for generating pseudo-numbers are grouped in two types; these are congruential and feedback shift register methods. Both methods make use of modular arithmetic reduction in congruential relationships, also known as clock arithmetic, i.e., the pseudo-random numbers "wrap-around" upon reaching a value [73]. Each of these methods present different variations and combinations that are described in this subsection.

5.1.1.1 Linear congruential generators

Linear Congruential Generators (LCG) are built following the form,

$$x_n = ax_{n-1} + c \pmod{m}$$

where a is known as the *multiplier*, c the *increment*, and m is known as the *modulus* of the generator. This type of generator is also known as "mixed congruential generator". Whenever the increment equals zero, the generator is known as "multiplicative congruential generator".

Due to its simplicity is often used as the based generator to understand and built better generators upon. The main reason LCG is not used in production settings is because of its short period, which is determined by the seed x_{n-1} and the value of the modulus m. Note that seed values equal zero is not allowed and therefore the maximum period is $m - 1$.

It has been proven that using prime moduli, usually Mersenne primes, gives statistically better results with longer periods if, additionally, the multipliers a are primitive roots of those moduli. Common modulus is $2^{31}-1$ with multiplier 7^5, occasionally the modulus $2^{61}-1$ is used.

Another problem in LCG generators is their lattice structure output, where it can be observed an evenly distribution of numbers. Meaning that the points can lie along some small amount of lines with similar slope not filling entirely the space. This problem can be present in higher dimension lags as well. Nevertheless, some shuffling techniques can be applied to the output stream in order to minimize this condition.

A desired characteristic worth analyzing with this basic generator is the creation of sub-streams (particularly useful in parallel generation). It is important that these streams do not overlap and there are no correlations among them. Common techniques used for creations of sub-streams are *skipping ahead* and *leapfrogging*. This techniques are review in Section 5.1.2 and Section 5.1.3, respectively.

LCG generators usually are implemented in 48-bits and 64-bits versions. The 48-bits version has the following form,

$$x_n = ax_{n-1} + p \ (\text{mod } 2^{48}) : p \text{ is a prime},$$

will generate 2^{19} streams (each generated with a different prime number) and a total period of 2^{48} per stream.

In the 64-bits version,

$$x_n = ax_{n-1} + p \ (\text{mod } 2^{64}) : p \text{ is a prime},$$

there are 2^{24} streams available, each with a total period of 2^{64}.

5.1.1.2 Multiple recursive generators

Multiple recursive generators are an extension of the multiplicative congruential generator. For this type of generator, the multiples of the previous k values are used to generate the next one. The form of this generator is,

$$x_n \equiv (a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_k x_{n-k}) (\text{mod}\, m)$$

The number of previous numbers used k is known as the "order" if the generator.

The period of a multiple recursive generator is often much longer that a multiplicative generator. With the right initial parameters it can go to $m^k - 1$. A common moduli is $2^{31} - 1$ and $k = 2, 3, 4$.

5.1.1.3 Combined multiple recursive generator

The combined multiple recursive generator is one of the most commonly used. Here, the based multiple recursive generator is combined through various mathematical operators with another type of generator. For example,

$$z_n = x_n + y_n \times 2^{32} \text{ mod } 2^{64}$$

where x_n has the same form described with the multiple recursive generator, but y_n is a linear congruential generator with the following values

$$y_n = 107374182 y_{n-1} + 104480 y - 5 \text{ mod } 2147483647$$

in a 64-bits setting this generator can produce 2^{24} distinct streams with period of 2^{219}.

5.1.1.4 Prime modulus linear congruential generator

This generator has the following form

$$x_n = a x_{n-1} \text{ mod } 2^{61} - 1$$

Which is the same form as LCG with zero increment and moduli in primes. This generator can produce 2^{58} different distinct streams using different multipliers $\boldsymbol{a}$ per stream. The period is expected to be in the order of $2^{61} - 2$ per stream.

5.1.1.5 Other congruential generators

The lagged-Fibonacci congruential generators are in the same family of the linear congruential generator (LCG).

The different types of lagged-Fibonacci generators are:

- Generalized lagged-Fibonacci generator

  $$x_n = x_{n-k} \circ x_{n-1} \text{ mod } 2^b : l > k$$

  where $\circ \in \{\oplus, +, -, \times\}$.

- Modified lagged-Fibonacci generator (one of the most popular) has the form

  $$z_n = x_n \oplus y_n$$

  which does not require a mod operation, making it faster. Both operands, x and y are additive lagged-Fibonacci, and $\oplus$ is an exclusive OR operation. Using a 32-bits additive lagged-Fibonacci generators with the forms $x_n = x_{n-k} + x_{n-l}\, mod\ 2^{32}$ and $y_n = y_{n-k} + y_{n-l} \text{ mod } 2^{32}$ there are $2^{31(l-1)}$ streams available with periods of $2^{31(2^l-1)}$ each. When the default lag $l = 1279$ is used, these values translate in 2^{39618} streams with 2^{1310} period each.

- Matrix congruential

  $$x_n \equiv (Ax_{i-1} + c) \text{ mod } m$$

  where x_i, x_{i-1}, and c are vectors of length d, and A is a $d \times d$ matrix. The elements of the vectors and matrices are integers between 1 and $m - 1$.

- Add-with-Carry, Subtract-with-Borrow, and Multiply-with-Carry generators shared the same basic form. For example, the add-with-carry takes the form

$$x_i \equiv (x_{i-2} + x_{i-1} + c_i) \bmod m$$

  where $c_1 = 0$, and $c_{i+1} = 0$ if $x_{i-2} + x_{i-r} + c_i < m$ and $c_{i+1} = 1$ otherwise. Note that c in the value of carry.

5.1.1.6 Multiply-with-Carry generator

The Multiply-with-Carry (MWC) is the pseudo random number generator chosen to be used in our simulator because its computational complexity and memory usage are very low and the initialization process very simple. MWC as implemented in our simulator uses two bitwise operators to obtain the multiplier a and the carry c from the same prime number, one multiplication of the multiplier times the previous pseudo random number x_{i-1}, and one addition. MWC form is

$$x_i = a x_{i-1} + c$$

where the multiplier has the following characteristic, $m = 2^{32}a - 1$ and $m - 1/2$ are prime. The lower 32 bits of a prime number forms x_{i-1} and the upper 32 bits the carry c.

5.1.1.7 Feedback shift register generators and Mersenne Twister

These type of generators make use of binary logic and is based on sequences of 0s and 1s in a vector space. The bits are shifted, say, to the left one bit at the time. The bit shifted out is then combined, usually with and exclusive-OR operator, with other bits in the register to form the rightmost bit. This specific generator is known as XOR shift generator,

$$y_n = y_{n-s} + y_{n-r} \ (\mathrm{mod}\ 2) : r > s$$

The two sources of bits to shift into the right-hand side of the register are called *taps* and can be easily extended to more taps. The period of this generator is $2^{251} - 1$ and each tap can produce and independent stream of random numbers.

Mersenne Twister

The Mersenne Twister (MT) generator is a modification of the generalized feedback shift register generator with the use of Mersenne primes, i.e., primes of the form $M_n = 2^n - 1$ (resulting in prime numbers of 48 digits in total as of January 2013).

In the MT generator the recurrence is obtained by twisting the bit pattern in x_{i-p+q}. This is achieved by formatting the x values in l-vectors of zeros and ones and multiplying x_{i-p+q} by a matrix A with dimension $l \times l$. The form is

$$x_i = x_{i-p} \oplus Ax_{i-p+q}$$

The period for MT generator is $2^{19937} - 1$.

5.1.1.8 Combinational generators

Often times, as already seen, more than one type of generator is combined in order to obtain statistically improved random number streams. Recall the shuffling technique in the LCG generator, which avoids evenly distributed outputs by combining the output of the LCG generator with another type of generator. The basic form is

$$w_n = y_n + z_n \pmod{p}$$

Each generator can be seen as an independent stream which later is combined with another stream. A consideration to make is that both streams should not suffer from similar irregularities since the combination of these most likely will not overcome the problems [73].

Typical combinations of generators

The XOR shift or generalized feedback shift register generator is combined with the Weyl generator to create the popular XORWOW generator available on many commercial libraries. The Weyl generator takes the following form

$$y_n = y_{n-1} + 362437 \text{ mod } 2^{32}$$

with period of 2^{32}, generating a sequence of 32-bit integers $y_0, y_1, \cdots, y_n$. The combined with the XOR shift (x_n)

$$z_n = x_n + y_n \text{ mod } 2^{32}$$

This combined generator produces streams of period $(2^{160} - 1)2^{32}$.

5.1.1.9 Non-linear congruential generators

These type of generators are complex and usually require more computational power than previously described generators. One random number takes $O(\log m)$ to generate [74] [75]. The lattice structure obtained in non-trivial with a strong non-linearity property and a good serial correlation.

The two typical forms of non-linear congruential generators are:

- Implicit inversive congruential

$$x_n = a\overline{x_{n-1}} + c \text{ (mod } p)$$

- Explicit inversive congruential

$$x_n = a\overline{n} + c \text{ (mod } p)$$

In both cases the apparent randomness indicates better results than other congruential generators. The inversive generator does not perform well on tests based on spacings of the lattices. In general and based on current evidence, it is not recommended the use of inversive generators or the modified explicit inversive [73].

5.1.1.10 MATLAB® library of generators and other commercial libraries

MATLAB® uses the Mersenne Twister (MT) generator by default. The command to generate a random number is *rng*. With the latest version of MATLAB® it is possible to define which generator to use between the following options; MT with the command *twister*, combined multiple recursive with the command *combRecursive*, or the multiplicative lagged-Fibonacci with the command *multFibonacci*. Additionally, MATLAB® provides backward compatibility with three additional generators; multiplicative congruential generator (*mcg16807*), shift-register generator summed with linear congruential generator (*shr3cong*), and modified subtract with borrow generator (*swb2721*). Only two of these generators offer the option to create multiple sub-streams, those are the multiplicative lagged-Fibonacci and the combined multiple recursive generators. Table 4-1 lists all the random number generators available in the MATLAB® library with their respective periods.

Table 5-1 MATLAB® Random number library summary

Keyword	**Generator**	**Sub-streams**	**Period**
`mt19937ar`	Mersenne twister (default)	No	$2^{19937} - 1$
`mcg16807`	Multiplicative congruential	No	$2^{31} - 1$
`mlfg6331_64`	Multiplicative lagged-Fibonacci	Yes	2^{124}
`mrg32k3a`	Combined multiple recursive	Yes	2^{127}
`shr3cong`	Shift-register summed with LCG	No	2^{64}
`swb2712`	Modified subtract with borrow	No	2^{1492}

Other commonly used libraries for generation of random numbers are:

1. Scalable parallel random number generator (SPRNG) library [77]. Which includes the following generators:

    a. Combined multiple recursive generator
        - Period: 2^{219}
        - Streams: 10^8

b. 48-bit Linear congruential generator with Prime addend
   - Period: 2^{48}
   - Streams: 2^{19}

c. 64-bit Linear congruential generator with Prime addend
   - Period: 2^{64}
   - Streams: 10^{8}

d. Modified lagged-Fibonacci generator
   - Period: 2^{1310}, with default $l = 1279$
   - Streams: 2^{39648}

e. Multiplicative lagged-Fibonacci generator
   - Period: 2^{81}, with default $l = 17$
   - Streams: 2^{1008}

f. Prime modulus LCG
   - Period: $2^{61} - 1$, with moduli $2^{61} - 1$
   - Streams: 2^{58}

2. GNU Scientific library [78], with the following generators:

a. Mersenne twister
   - Period: $2^{19937} - 1$ or about 10^{6000}

b. Ranlux: subtract-with-borrow and different levels of statistical quality
   - Period: 10^{171}

c. Combined multiple recursive generator
   - Period: 2^{185} or about 10^{56}

d. Fifth-order multiple recursive generator
   - Period: 10^{46}

e. Shift register generator
   - Period: 2^{88} or about 10^{26}

f. Lagged-Fibonacci generator
   - Period: 10^{2917}

5.1.2 Skipping ahead technique for streams of pseudo-random numbers

In the skipping ahead technique, as the name implies, it is skipped a known distance k ahead in the stream to start a subsequence with no overlapping output. The generator takes the form,

$$x_{i+k} = a^k x_i \pmod{m}$$

For example, processor number one could traverse the sub-stream in the sequence $x_s, x_{s+1}, x_{s+2}, \cdots$, while another processor will traverse the next non-overlapping sequence $x_{s+k}, x_{s+k+1}, x_{s+k+2}, \cdots$, as so on.

5.1.3 Leapfrogging technique for streams of pseudo-random numbers

In the leapfrogging technique, the sub-sequence would be generated as $x_s, x_{s+k}, x_{s+2k}, \cdots$, leapfrogging or jumping by a factor k, taking careful consideration that the distances are relatively prime. For example, the first process will start at x_1 and the second at x_{1+k}, then the first will jump over the second to position at x_2 and the second process over this to x_{s+k} and so on.

5.1.4 Parallel pseudo-random number generators

Several techniques are used to generate random numbers in a parallel setting. The basic idea is to produce independent non-overlapping streams on each thread, taking into account the computational resources available. Usually, these streams are obtained by splitting a serial pseudo-random number generator into statistically independent streams.

One possible option to generate different streams is to use the same type of generator with changed initial parameters for each processor or thread. The different parameters could be chosen randomly. But, the results have high chances of being poor statistically speaking. If the seeds are chosen randomly the method is called *seeding*. If there are parameters taken from a predefined values, typically known to give better results, the method is called *parametrization* [76].

Again, it is possible to use the same generator with a very large period and divide the output stream using the methods mentioned earlier. An option could be the *block splitting* technique, i.e., skipping ahead. Here, the period is divided in known sizes and each division distributed among different streams. Another option is the *leapfrogging* technique where the period is divided to different streams in a leapfrog manners.

Both previously mentioned options make use of the same output stream of random numbers subdivided for each thread. Nevertheless, a third option is to use a totally different type of generator for each thread.

Leaping ahead a fixed length L costs no more than generating $O(\log_2(L))$ numbers [77]. The drawback with this technique is the lack of scalability, which could be circumvented by leapfrogging. Still, when splitting, the costs are not constant and some generic correlations are present, i.e., cross-correlation between sequences or random numbers and inter-correlations between threads or processors.

Leapfrogging produces $l = \left\lfloor \frac{\mathrm{Period}(x_i)}{L} \right\rfloor$ [77]. So, it can be obtained a first block such as $\{x_0, x_l, x_{2l}, \cdots, x_{(L-1)l}\}$, a second block such as $\{x_1, x_{1+l}, x_{1+2l}, \cdots, x_{1+(L-1)l}\}$, to the i^{th} block $\{x_i, x_{i+l}, x_{i+2l}, \cdots, x_{i+(L-1)l}\}$.

Another issue to be addressed is the memory usage with such big amount of random numbers generated. Typically, it is preferred to generate the random numbers as are needed. Also, the initial states need to be small enough since in a parallel setting the number of initial parameters can be large. Note the trade-off, the larger states is possible to have larger periods which generally translates in better statistical properties.

5.1.4.1 Parallel libraries of pseudo-random generators

Commercially available libraries relieve the user from the need of testing for statistical correctness. Nevertheless, these libraries tend to be developed for specific environments and usually general use. They can easily become a computational burden themselves, especially in heavy use.

The list of the most known libraries is presented her:

1. cuRAND by NVIDIA© [78]. It uses the technique of skipping ahead to generate the different streams. The available generators in this library are:
   a. Mersenne twister
   b. Combined multiple recursive generator
   c. XOR WOW shift register
2. Thrust::random with the following generators [79]
   a. Linear congruential generator
   b. Linear feedback shift register
   c. Subtract-with-carry
3. Scalable parallel random number generator (SPRNG) library. It uses parametrization [80]
4. GPU accelerated scalable parallel random number generator (GASPRNG), based on the previous generator but optimized for massively parallel environments [81]. It includes the following generators:
   a. Lagged-Fibonacci generator
   b. Linear congruential generator (LCG)
   c. 64-bit LCG

    d. Multiplicative lagged-Fibonacci generator
    e. Combined multiple recursive generator
    f. Prime modulus LCG
5. Tina’s random number generator library [76]. It employs block splitting and leapfrog sequence strategies
6. RngStream library [76]. It uses block splitting strategy and developed for OpenMP and MPI
    a. Combined multiple recursive generator
7. ShoveRand used as a framework to define common rules [82]

5.1.4.2 Stand-alone implementations of pseudo-random generators

Outside the scope of previously listed libraries there are stand-alone random number generators that are available for a specific type of generator and architecture.

1. Tiny Mersenne twister for CUDA [83]: It passes all statistical tests. Nevertheless, its initialization process and generation speeds are slower than those of the Mersenne twister implemented in cuRAND. The authors include an initialization routine which generates the initial parameters a priori
2. WarpStandard [84]: Based on XOR and shift registers operations. It passes all statistical tests and runs faster than Tiny Mersenne twister for CUDA. There is no enough documentation available on this implementation

And extensive and updated comparison of all different random number generators can be found in Reference [85].

5.2 Chapter summary

In this chapter we provided a detailed review of pseudo random number generators, including their different parallel implementations. The pseudo number random generator used in our massively parallel simulator of OCT of inhomogeneous media is the Multiply-With-Carry (MWC). It was chosen because its computational complexity and memory requirement are very low. The MWC that is implemented in our simulator uses two bitwise operators, one multiplication and one addition.

Chapter 6

Massively parallel computing concepts

Parallel computation is a technique by which programs can be accelerated through the use of parallel hardware and capable of executing program instructions in parallel, i.e., simultaneously. There are several conceptual and programming challenges to address when working in a parallel environment. Parallelism is a familiar concept in everyday life. For example, humans are capable of performing certain tasks simultaneously.

This thesis deals with massively parallel computing which typically involves thousands of small scale processors or many-core processors (to contrast with multi-core processors, see Section 6.1.1). In massively parallel computing the focus is on the execution throughput of parallel applications. Graphics processing units (GPUs) are built with as many cores as possible since each core processes output for a typically small number of pixels in a display or are used for general purpose computing.

6.1 Parallel hardware

There are several conceptual differences between the diverse hardware in use for exploiting parallelism. The focus of this thesis is on general purpose computing on graphic processing unit (GPGPU). In the following sections there are reviews for multicore and many-core processors with a comparison in hardware between this two hardware architectures.

6.1.1 Multicore processors

Multicore processors incorporate multiple cores on the same integrated circuit chip (die). These cores are used collectively and there is a linear relation between the amount of cores and overall performance. Most central processing units (CPUs) in today's electronic devices like cellphones, personal computers, and servers make use of multicore processors. Unfortunately, there are boundaries on the packaging of more cores into a single die. Typical constrains are heat generation, power consumption, miniaturization limits, and increased overhead in control logic systems. It is no surprise that clock-speeds have not being dramatically improved since the 2000's and only the increasing number of cores has facilitated the growth of overall CPU performance.

The advantage of multicore processors is that most parallel paradigms are hidden and automatically implemented by the embedded control logic systems. The control system is in charge of making use of instruction level parallelism at execution time. The idea behind this is to maintain and speed-up execution of sequential programs through multiple cores.

Nevertheless, full advantage of parallelism with parallel programs can be implemented as well in multicore systems. Supercomputers with more than one CPU, clusters across networks, servers in distributed settings, and grid computing are some of the typical configurations of multicore processors in systems for parallel computing. These configurations are outside the scope of this thesis.

6.1.2 Many-core processors

Many-core processors typically have a larger number of cores than multicore processors. The implementation of more cores in a die is feasible due to the fact that these cores are smaller in size and typically use basic control systems. As mentioned before, the focus of many-core processors is execution throughput of parallel applications. These applications have to be designed following parallel paradigms to make full use of the advantages of parallelism. GPUs are examples of many-cores processors. GPUs excel in floating-point performance comparing to CPUs due to their fundamentally different design philosophy.

6.1.3 CPU vs. GPU design characteristics

The hardware design of a CPU is optimized for sequential code execution. This optimization is possible through the use of sophisticated control logic which allows sequential instructions to be executed in parallel or out of their sequential order while maintaining sequential behavior. Additionally, large cache memory are available to reduce instructions and data access latencies. It is important to mention that neither control logic nor cache memories have effect on peak calculation speeds.

Since its conception, GPU hardware design has been optimized for throughput of massive number of floating-point calculations per video frame. The hardware takes advantage of the larger number of cores for thread execution. This translates in a large number of execution threads that keeps the cores busy, so the latency of memory access is hidden and the control logic required per execution thread is reduced. It is clear that GPUs are designed as numeric computing engines, even more apparent with the higher number of arithmetic-logic units (ALUs). The fundamentally different design differences that make possible the higher throughput of floating-point calculation of GPUs against CPUs are shown in Figure 6-1.

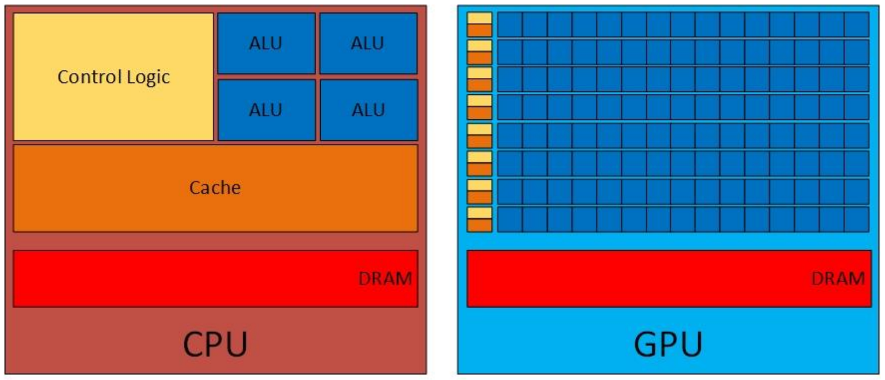

Figure 6-1 CPU vs. GPU general hardware design philosophies

Another important difference between GPUs and CPUs is the memory bandwidth to dynamic random access memory (DRAM). GPUs have been operating at approximately 10 times the bandwidth of contemporaneously available CPUs. This is because of the relaxed memory model that GPUs implement to satisfy the frame buffer requirements. CPUs are most times constrain by requirements from legacy operative systems, applications, and I/O devices which make memory bandwidth more difficult to increase.

6.2 Compute Unified Device Architecture (CUDA)

GPU performance is highly parallel. The capability to use GPU hardware for parallel programming was greatly simplified with the release of the Compute Unified Device Architecture (CUDA). CUDA is a platform and programming model developed by NVIDIA for GPUs. The Application Programming Interface (API) included with CUDA offers extensions for many industry standard programming languages, as the C language, with CUDA's own accelerated libraries and compiler directives [78]. The CUDA environment allows the programmer to simultaneously develop code intended for both central processing unit (CPU) and GPU.

6.2.1 Logical hierarchy in CUDA

The basic unit of execution in a CUDA program is a thread which runs independently and concurrently with a big number of other similar threads. A group of 32 threads forms a Warp, which runs on the same instruction counter. A thread has the finest granularity and is identified by a number or index. A group of threads forms a block, which has associated to it an identification number or index and a dimension. A set of blocks forms a grid with a dimension associated to it. Figure 6-2 shows the relation diagram between threads and blocks.

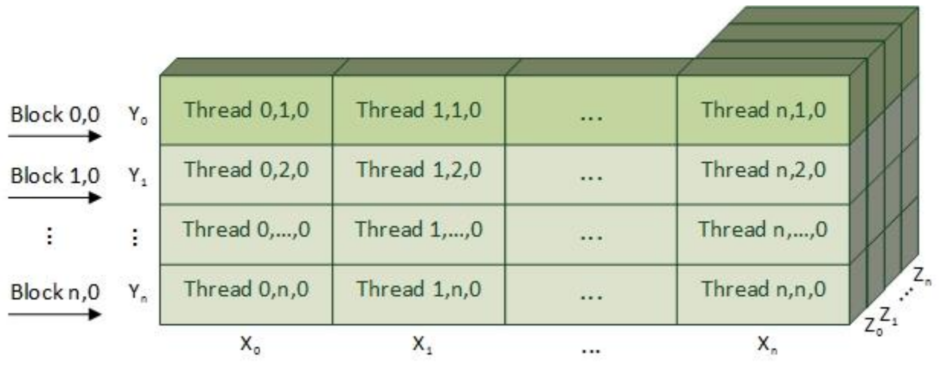

Figure 6-2 Threads and blocks diagram

CUDA uses an execution model known as Single Instruction, Multiple Thread (SIMT) which allows independence between threads. SIMT allows each thread to have different execution path, separate registers, and possibly execution divergence that would result in different values in instruction address counters [86]. The most common execution divergence occurs with control flow statements (e.g., if-then-else, switch), where threads not meeting a logical condition are stopped until all other threads fulfilling this condition finish executing this condition [87] [88].

To minimize performance penalties CUDA caches data from stopped threads for fast access. However, avoiding execution divergence and minimizing execution times of all logical conditions are preferable not always simple [89] [90].

6.2.2 Definition of *device* and *host*

In the CUDA environment, the GPU hardware is identified as *device* and the CPU hardware is known as *host*. The code executed on a GPU is called a kernel, where it consists of a typically large number of identical threads.

Whenever the *host* code encounters a kernel function, the later will be executed by the GPU's processors in an asynchronous manner. Once this kernel is launched, control is immediately relinquished to the *host* and further *host* code will be executed by the CPU alongside the GPU execution. Any function called and executed by the GPU is known as *device* function.

Data needed by a kernel has to be moved back and forth from the host's memory to the *device's* memory. Note that CPU and GPU memory spaces are completely separate and have no direct access to each other.

6.2.3 CUDA memory hierarchy

A GPU has different memory types that are configured in a hierarchical structure, see Figure 6-3. The global memory is available to all threads but has slow access, i.e., ~400 to 600 clock cycles. The register is a relatively small memory space assigned to each thread with very fast, i.e., 1 clock cycle, and exclusive access. Constant memory is available to all threads and is read-only. Local memory is only available for groups of 32 threads, called a warp, and is where all variables actively used by this warp are allocated. The compiler sets this local memory space according to the kernel's requirement at execution time. Constant and local memories are part of the global memory's physical space but are cached to speed up their access times. Figure 6-3 shows the memory hierarchy and access scope.

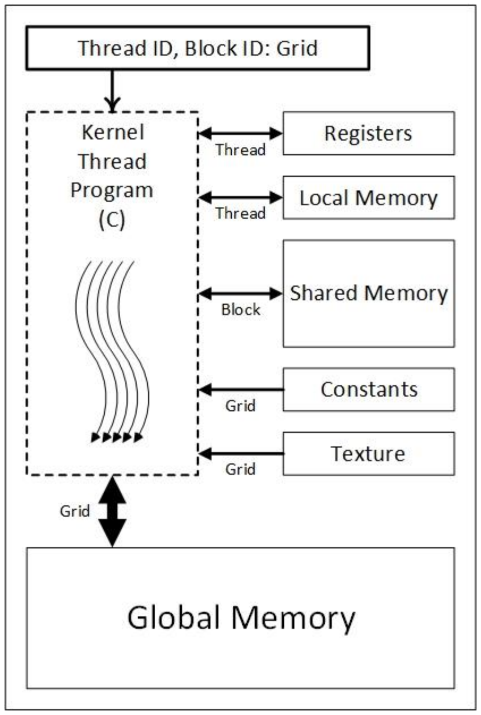

Figure 6-3 Memory hierarchy with access scope

6.2.4 CUDA kernel function

A kernel function, which consists of a typically large number of identical threads, is executed with a given kernel configuration that sets up the GPU's memory space and its computational resources needed during execution. The kernel configuration also defines a grid of thread blocks that are identified by a number of thread blocks. Every thread block could possibly include more than one warp.

6.2.5 Atomics in CUDA

To avoid race conditions [91], where more than one thread need to write to the same memory location, CUDA offers special operators called atomic operators. These atomic operators have the same precision as their counterpart CPU based operators.

6.2.6 Definition of CUDA *compute capability*

Compute capability is NVIDIA's classification of CUDA capable GPU hardware. It is an incremental number to group different hardware with similar computational capabilities, i.e., types of arithmetic operators and their precision. *Compute capability* is typically backward compatible and does not refer to the hardware's computational speed.

NVIDIA's family of GPUs for scientific computing is named Tesla and to the date of this writing they are sold up to *compute capability* 3.7. Table 4-1 shows a comparison between the different *compute capability* upper limit configurations for threads, grid, and block sizes.

Table 6-1 Tesla GPU *compute capability* comparison

Compute capability:		1.x	2.x	3.x
Total threads per block:		512	1024	1024
Grid size	dGrid.x	65535	65535	$2^{31}-1$
	dGrid.y	65535	65535	65535
	dGrid.z	1	65535	65535
Block size	dBlock.x	512	1024	1024
	dBlock.y	512	1024	1024
	dBlock.z	64	64	64

6.3 Chapter summary

This chapter provided all the necessary concepts to understand why a massively parallel implementation of our simulator is well suited for implementation on Graphics Processing Units (GPU). Also most CUDA parallel terminology and concepts used in the implementation of our massively parallel simulator of OCT were explained in detail.

Chapter 7

Implementation of a massively parallel simulator of OCT of arbitrary shaped inhomogeneous media

The implementation of an OCT simulator in a massively parallel environment addresses the high computational load associated with most Monte Carlo methods. That is, processing a typically large number of samples required for a suitable accuracy.

In this simulator a large number of samples, i.e., photon packets, are launched simultaneously and traced independently. Thus, it should be expected a considerable reduction in computation time due to parallel implementation. In this chapter, a description of the design and implementation of this simulator is given, including photon tracing, random number generation, and memory utilization. Also, a discussion about portability across different GPUs is presented.

7.1 OCT massively parallel simulator (OCT-MPS) design

This OCT massively parallel simulator (OCT-MPS) of inhomogeneous media is implemented in NVIDIA’s CUDA environment using extensions of the C language. Most GPU configuration and code design are similar to the GPU-MCML simulator [92]. Computational intensive functions, i.e., photon packet tracing, are implemented in a kernel function, while less computational intensive ones, i.e., I/O data handling, are executed in the host. Three configuration text files are read at program initialization: *object.txt* which stores the optical parameters of the object, *mesh.txt* which stores coordinates of the tetrahedron mesh elements, and *sim_param.txt* which stores user defined simulator parameters, e.g., coherence length of the optical source, fiber probe's collection angle, etc. Figure 7-1 shows a flowchart representing the design of OCT-MPS.

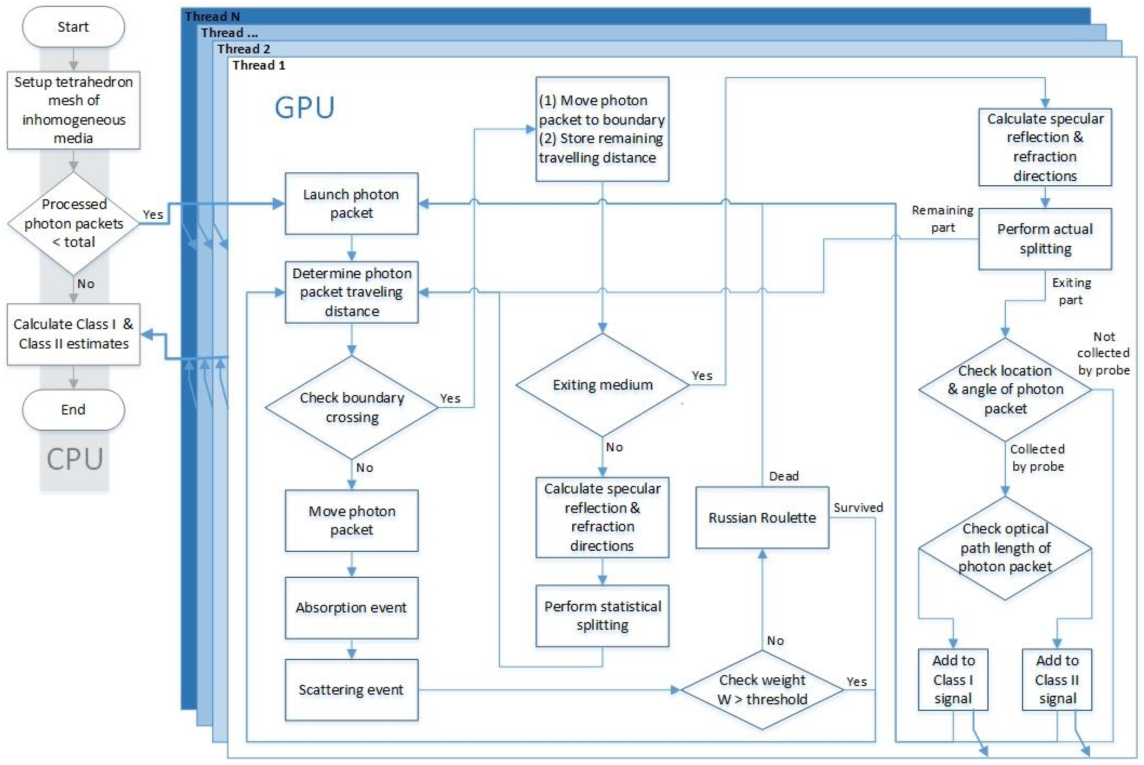
Start
Setup tetrahedron mesh of inhomogeneous media
Processed photon packets < total
Yes
No
Calculate Class I & Class II estimates
End
CPU
Thread N
Thread ...
Thread 2
Thread 1
GPU
Launch photon packet
Determine photon packet traveling distance
Check boundary crossing
Yes
No
Move photon packet
Absorption event
Scattering event
(1) Move photon packet to boundary (2) Store remaining travelling distance
Exiting medium
Yes
No
Calculate specular reflection & refraction directions
Perform statistical splitting
Russian Roulette
Dead
Survived
Check weight W > threshold
No
Yes
Calculate specular reflection & refraction directions
Remaining part
Perform actual splitting
Exiting part
Check location & angle of photon packet
Not collected by probe
Collected by probe
Check optical path length of photon packet
Add to Class I signal
Add to Class II signal

Figure 7-1 Flowchart representing the design of OCT-MPS

7.2 Tracing photon packets and recording OCT signals

As shown in Figure 7-1, each thread traces the complete path of a photon packet from its launch into the object until the end of its life, or when it exits the object. When a photon packet dies and if more photon packets need to be traced, another one will be launched using the same thread. Therefore the number of photon packets traced per thread depends on the total number of photons to be simulated and the maximum number of simultaneous threads that could run on the GPU. This number of simultaneous threads depends on CUDA’s *compute capability*, and it is set as an initial configuration of the kernel for optimized hardware utilization.

7.2.1 Description of OCT-MPS code functions

All functions that trace a photon packet in a thread are implemented as device functions, and are called by the CPU-launched kernel: *OCT-MPSKernel*. The names of these functions and their variables are self-descriptive and use the following convention: all function names and names of variables common between CPU and GPU use the “CamelCase” naming convention [93], while names of variables used only in GPU only are all in lower case. The code listing 7.1 is the kernel *OCT-MPSKernel*.

```
__global__ void OCT-MPSKernel(SimState *d_state, GPUThreadStates *tstates)
{
  // photon structures
  PhotonStructGPU photon, photon_cont;

  // random number seeds
  UINT64 rnd_x, rnd_xR, rnd_xS;
  UINT32 rnd_a, rnd_aR, rnd_aS;

  // Flag to indicate if this thread is active and region number
  UINT32 is_active, region;

  // Restore the thread state from global memory
  RestoreThreadState(d_state, tstates, &photon, &photon_cont, &rnd_x, &rnd_a,
                     &rnd_xR, &rnd_aR, &rnd_xS, &rnd_aS, &is_active);

  for (int iIndex = 0; iIndex < NUM_STEPS; ++iIndex) {
    // Only process photon if the thread is active
    if (is_active) {
      region = photon.layer = photon.tetrahedron[photon.rootIdx].region + 1;

      //>>>>>>>>> StepSizeInTissue() in Serial-OCT
      ComputeStepSize(&photon, &rnd_x, &rnd_a);

     //>>>>>>>>> HitBoundary() in Serial-OCT
     photon.hit = HitBoundary(&photon);
```

```
        Hop(&photon);
        if (photon.hit) {
           //>>>>>>>>> CrossOrNot() in Serial-OCT
           FastReflectTransmit(&photon, d_state, &rnd_xR, &rnd_aR);
           photon.region = photon.tetrahedron[photon.rootIdx].region + 1;
        } else {
           Drop (ignoreAdetection, &photon);
           SpinBias(d_layerspecs[region].g, &photon, &photon_cont,
                    &rnd_xS, &rnd_aS);
        } // End else-if(!photon.hit)

        /*************************************************************************
        *  >>>>>>>>> Roulette() in OCT-Serial
        *  If the photon weight is small, the photon packet tries to survive a Russian roulette
        *************************************************************************/
        if (photon.w < WEIGHT) { // && !photon.dead
           FLOAT rand = rand_MWC_co(&rnd_x, &rnd_a);

           // This photon survives the roulette
           if (photon.w != ZERO_FP && rand < CHANCE) {
              photon.w *= (FLOAT) FAST_DIV(FP_ONE, CHANCE);
           }
        /*************************************************************************
           // This photon is terminated if not Back Reflection event
           else if (photon.FstBackReflectionFlag) {
              // Do not subtract if there was Back Reflection event
              FLOAT LikelihoodRatioTmp = photon.LikelihoodRatioAfterFstBias;
              CopyPhotonStruct(&photon_cont, &photon);
              Spin(d_layerspecs[photon.layer].g, &photon, &rnd_xS, &rnd_aS);
              if (LikelihoodRatioTmp < 1)
                 photon.LikelihoodRatio = 1 - LikelihoodRatioTmp;
              else photon.LikelihoodRatio = 1;
           }

           // This photon is terminated
           else if (atomicSub(d_state->n_photons_left, 1) > NUM_THREADS) {
              LaunchPhoton(&photon); // Launch a new photon
           }
           // No need to process any more photons
           else
              is_active = 0;
        } // End if (Roulette)
      } // End if (is_active)
   } // end of the main for-loop (NUM_STEPS = 50000)
   __syncthreads();

   // Save the thread state to the global memory
   SaveThreadState(d_state, tstates, &photon, &photon_cont, rnd_x, rnd_a,
                   rnd_xR, rnd_aR, rnd_xS, rnd_aS, is_active);
}
```

Listing 7.1 Main global kernel *OCT-MPSKernel*

7.2.1.1 OCT-MPS kernel

The main device kernel function, tagged with the special reserved keyword __global__, is called *OCT-MPSKernel* and controls the flow of photon packets life. It initializes a data structure, *tstates*, to store parameters of the traced photon packet, e.g., its weight, location and direction, in addition to parameters of the tetrahedron where it is located. It also initializes an identical data structure to store parameters of a split photon packet that could arise from a potential reflection and transmission at a medium boundary.

OCT-MPSKernel creates and manages the following variables:

1. *is_active*: a flag to indicate if a thread is active which occurs when the number of packets left to be simulated is more than the maximum simultaneous number of threads available
2. Seeds variables for the random number generators, copied from the prime numbers list calculated a priori and provided to the algorithm as input

This function also uses the structure *DeviceMem* to keep track of simulation results, generated random numbers, and the number of traced photon packets. A Russian roulette routine, as described in Section 4.4.3.3, is also implemented in this function for photon packets with very low weights. It determines whether to terminate, or increase, the weight of such photon packets. Also if the number of photon packets left to simulate is less than the total number of available threads, it sets the *is_active* flag corresponding to these idle threads to false. This would prevent these idle threads from utilizing GPU resources.

A conceptual difference between the serial and parallel implementations is that in the former each photon packet is subtracted from the total number of photon packets, at the end of its life cycle. The total number of photons packets is established at the beginning of the simulation through the input file. The serial implementation adds each run, i.e., photon packet life complete life cycle, until the number of photon packets is reached.

As some photon packet traces will finish before others, a mechanism is implemented to minimize thread stalls due to long-lived traces. Each thread keeps track of the total number of steps that a photon packet underwent since its incidence into the medium. It then compares this number to a user defined constant. When the maximum allowed number of steps, MAX_NUM_STEPS, is reached, *OCT-MPSKernel* saves the states of these long-lived active threads to CPU memory. Then the program launches a new kernel with the maximum number of possible threads, including the saved long-lived threads, to ensure higher utilization of the GPU's hardware at any given time.

After the required number of photon packets are simulated, results are copied to CPU memory. The CPU then evaluates both Class I and Class II A-scans and saves them to a file. The above process of simulating an A-scan is repeated at different locations until complete Class I and Class II B-scans are obtained.

The following functions are device function tagged with the reserved word __device__, and can only be called from within __global__ device functions.

7.2.1.2 Restore thread state

The device function *RestoreThreadState* restores the state of each thread, e.g., optical parameters of the photon packet being traced and its random number streams. This function does not change the state of the photon packet and does not have a counterpart developed in the serial implementation of this simulator.

The first time this function is called, the simulator gets the default initial optical values and the first set of safe prime numbers for the random number generation. In consecutive calls the simulation gets the optical parameters that were saved before exiting the previous run, including the parameters for those scattered photons and the last random numbers used from the designated stream.

7.2.1.3 Compute step size

The device function *ComputeStepSize* determines the random travelling distance of a photon packet. It also updates the photon packet parameter's structure, *tstates*.

7.2.1.4 Hop

The device function *Hop* device function moves the photon packet according with the travelling distance computed in *ComputeStepSize*.

The placement in the logic flow of this function marks another difference with the serial implementation. The logic flow kept since Wang *et al.* [48] first implemented the MCML algorithm on which most OCT simulators are based on, including the serial OCT simulator implementation by Malektaji *et al.* [11] calls a "hop" function twice; (1) for packets not crossing the boundary and, then again, (2) for those crossing the boundary. Since the photon packet will move or hop with the same distance in both cases OCT-MPS implementation avoid divergence calling this function only once right after the step size is computed.

7.2.1.5 Hit boundary

The device function *HitBoundary* checks if a photon packet crossed the boundary of the tetrahedron where it is located. The result of this function determines one of two logical branches in the code corresponding to a photon packet crossing, or not crossing, its tetrahedron's boundary.

Whether the photon packet hits or not the boundary the algorithm faces the first unavoidable divergence. If the photon packet hits the boundary one branch will calculate if the packet gets reflected towards the probe or transmitted to an adjacent tetrahedron. The other branch will drop the weight of the photon packet and spin it.

7.2.1.6 Reflect and transmit

If a photon packet crosses its tetrahedron boundary, *FastReflectTransmit* device function checks if the neighboring tetrahedron belongs to a different region in the medium. If true, it calculates the reflectance and transmittance at this boundary. It is the most computationally expensive device function, and it inevitably creates another logical branch in the code. It calculates the number of photons to be reflected towards the probe and the number of photons to be transmitted to the adjacent tetrahedron. This splitting of the photon packet involves updating its weight and directional cosines. This function performs a read from the GPU's global memory, in addition to atomic operations to record Class I and Class II signals.

7.2.1.7 Drop photon weight

If a photon packet did not cross its tetrahedron boundary, *Drop* device function updates its weight.

7.2.1.8 Spin with bias

The device function *SpinBias* updates the directional cosines of a photon packet, by introducing a bias to deflect it towards the optical detector, as in Reference [67]. This is the second most expensive device function. The spinning process of photon packets involves the utilization of random numbers.

7.2.1.9 Copy photon structures

The device function *CopyPhotonStruct* stores optical parameters of the back-scattered (deflected towards optical detector) portion of an original photon packet that was split at a boundary. These stored optical parameters allow simulation of this back-scattered portion after tracing the transmitted portion of the original packet.

7.2.1.10 Spin without bias

The device function *Spin* updates the directional cosines of a photon packet without applying any bias. This process involves the use of random numbers.

7.2.1.11 Launch photon packet

The device function *LaunchPhoton* launches a new photon packet and resets its structure to the default values.

7.2.1.12 Save thread states

The device function *SaveThreadState* saves parameters related to the active thread, e.g., random number streams, and the parameters of currently traced photon packet, e.g., weight, location and direction, to the *tstates* structure.

7.2.2 OCT-MPS memory utilization

A kernel, *InitThreadState*, allocates and initializes all data structures to be used by *OCT-MPSKernel*, using *cudaMalloc* and *cudaMemset* CUDA commands. The coordinates of individual tetrahedrons that comprise the tetrahedron mesh are stored as a linked list in *global memory*. Access to this linked list results in a considerable implementation delay. Individual tetrahedrons are identified by an ID, stored in the structure *tstates* located in local memory, to track the tetrahedron where a traced photon packet is located. The optical parameters of every distinct region of the inhomogeneous medium are stored in *constant memory*. To reduce the number of slow accesses to *global memory*, the optical parameters of a region are cached in *tstates*, the structure associated with the currently traced photon packet.

7.2.3 Random number generator in OCT-MPS

The random number generation in a parallel environment is not a trivial matter. We need to avoid any type of inter thread correlations, the generator has to have a long period, and has to be relatively light in computation. The random number generator used is the Multiply-With-Carry (MWC) by Marsaglia [94] due to its light computation weight and hardware resources requirement.

OCT-MPS uses an improved implementation of MWC by Alerstam *et al.* [95], the modifications made guarantee avoiding correlations. The implementation by Alerstam *et al.* uses one stream of random numbers for all functions involved in a trace. OCT-MPS implementation creates an independent stream of random numbers for the spinning functions, another for the reflection and refraction functions, and a third one for the rest of functions where a random number is needed.

The seeds for the different random number streams previously described can be obtained from the system's clock or set manually assuming that the three are different. In the device each thread will have a unique seed safe prime number and unique multiplier to obtain uncorrelated streams of periods of ~260 each. A list of safe prime numbers was created previously and is available as a text file with our implementation. OCT-MPS implementation of the MWC uses 9 registers per thread.

7.2.4 OCT-MPS portability across GPUs

OCT-MPS is ready to work with many hardware configurations, with support of a wide range of *compute capability* (versions 1.3 to 3.5), as well as with single and multiple GPUs. The different pre-configured parameters guarantee the maximum utilization of the available GPU resources with the specific data while overlapping as much as possible the latency produced by data transfers. These configurations can be easily exchanged or modified for future releases or further experiment bound uses and optimizations.

Many macros are defined to use optimized operators when possible or more expensive operators when greater precision is required. From *compute capability* 1.3 and up CUDA supports double precision floating-point numbers, our simulator makes use of this at a cost of more expensive computation. Some optimized operators are only available in higher *compute capability* versions.

Due to expensive performance using double precision addition operations, CUDA has not implemented an optimized operator for atomic addition of double precision numbers in *global memory*. To overcome this problem an atomic addition function was implemented using the available atomic operator for compare-and-swap method (*atomicCAS*) following the technical details described in reference [78].

7.3 Computational time of the simulator

The profiling of the computation time of our simulator was done using the visual profiler included in Nsight integrated development environment (IDE) which is part of the CUDA software development kit (SDK). Detailed times of all device functions were obtained using our own algorithm implemented with the *clock()* function from the C language standard library. Figure 2-1 shows the factions of the total time spent to obtain a typical A-scan.

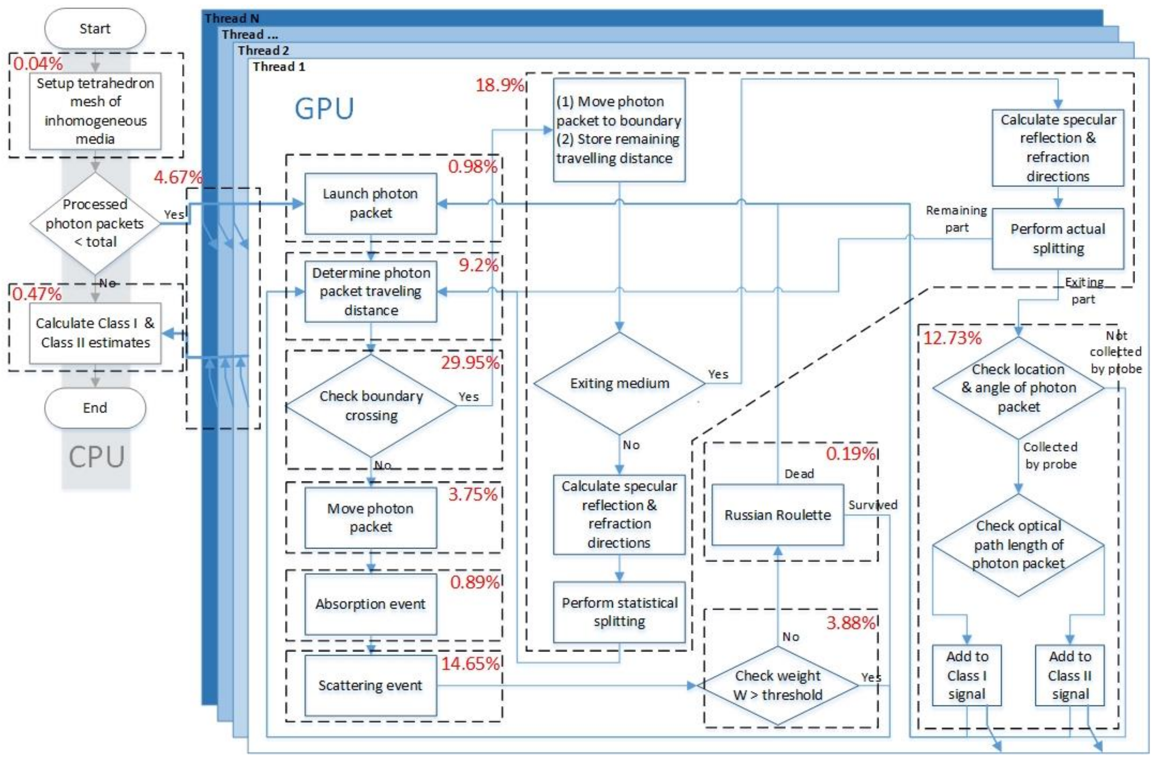
Start
0.04%
Setup tetrahedron mesh of inhomogeneous media
4.67%
Processed photon packets < total
Yes
No
0.47%
Calculate Class I & Class II estimates
End
CPU
Thread N
Thread ...
Thread 2
Thread 1
GPU
0.98%
Launch photon packet
9.2%
Determine photon packet traveling distance
29.95%
Check boundary crossing
Yes
No
3.75%
Move photon packet
0.89%
Absorption event
14.65%
Scattering event
18.9%
(1) Move photon packet to boundary
(2) Store remaining travelling distance
Exiting medium
Yes
No
Calculate specular reflection & refraction directions
Perform statistical splitting
0.19%
Dead
Russian Roulette
Survived
3.88%
No
Check weight W > threshold
Yes
Calculate specular reflection & refraction directions
Remaining part
Perform actual splitting
Exiting part
12.73%
Check location & angle of photon packet
Not collected by probe
Collected by probe
Check optical path length of photon packet
Add to Class I signal
Add to Class II signal

Figure 7-2 Time fractions of the computation time for a typical A-scan

The most computationally demanding module of our massively parallel simulator of OCT of inhomogeneous media is the checking of intersection of a photon packet's path with its enclosing tetrahedron, taking close to 30% of the total time. This behavior is expected since the access to global memory has high latency. Also all modules where access to global memory is involved we observe higher computational times. We note that transfer of the data structures prior to an actual simulation is included in Figure 7-2, comprising 4.67% of the total simulation time.

Chapter 8

Simulation results and computational times

We implemented our optical coherence tomography massively parallel simulator (OCT-MPS) in CUDA with extensions of the C language. To validate OCT-MPS all simulation results we compared them with those previously validated results obtained by Malektaji *et al.* [11].

8.1 Hardware setup used to run OCT-MPS

The simulations were tested on two different hardware setups and compute capabilities:

The first configuration consisting of three Tesla C1060 GPU cards with *compute capability* 1.3. Each one of these three Tesla C1060 cards has: 240 scalar processors running at 602 MHz, total global memory of 4 GB (DDR3), register memory per thread of 16384 bytes, same as available constant memory. The toolkit version used for the parallel simulations on these cards is 6.0 and the driver version is 331.62 on a Linux Ubuntu 12.04.4 LTS server.

The second hardware configuration has one Tesla K40c GPU card with *compute capability* 3.5. The K40c card has 2880 scalar processors arranged in 15 streaming multiprocessors with 192 CUDA cores each, running at 745 MHz, total global memory of 12 GB (DDR5), total number of registers per block of 65536 bytes, and 2 copy engines for concurrent execution. The toolkit version used for the parallel simulations on this GPU card is 6.5 and the driver version 340.32 on a Linux CentOS 6.5 server.

8.2 Validation of OCT-MPS using a four-layer medium

The medium used is a four-layer object with 9600 tetrahedrons and 2205 vertices. The optical properties are shown in Table 8-1 and are exactly as used in [48] [67] [13]. In all simulations the ambient medium is air with refractive index $n = 1$.

The optical fiber probe has radius 1 μm, acceptance angle 5° and an optical source with coherence length $\mathrm{l_c} = 15$ μm. The biasing coefficient $\mathrm{a} = 0.925$ and additional bias probability $\mathrm{p} = 0.5$, as in Reference [11]. The number of photon packets used is 10^7.

Table 8-1 Optical parameters of the four-layer medium used to validate OCT-MPS

Layers	Height (cm)	Scattering Coefficient $\mu_s\ (\mathrm{cm}^{-1})$	Absorption Coefficient $\mu_a\ (\mathrm{cm}^{-1})$	Anisotropy Factor g	Refractive Index N
Layer 1	0.0200	60	1.5	0.9	1.0
Layer 2	0.0015	120	3	0.9	1.0
Layer 3	0.0345	60	1.5	0.9	1.0
Layer 4	0.0440	120	3	0.9	1.0

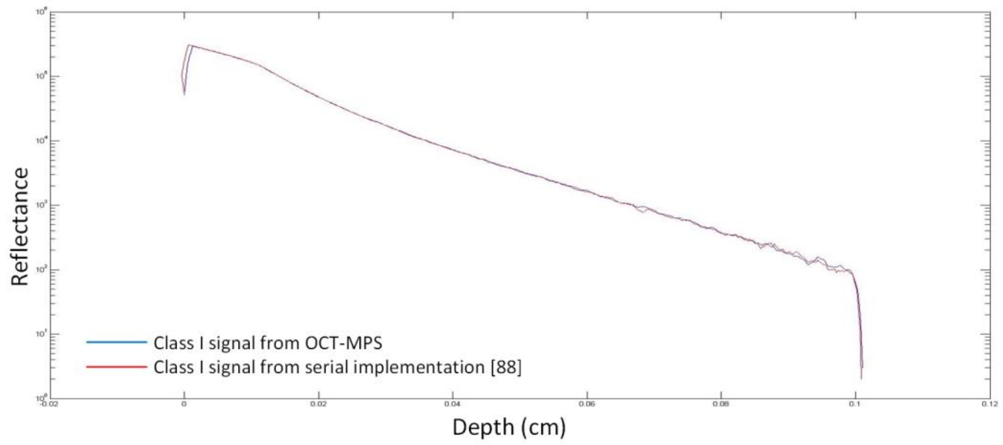

Figure 8-1 A-scans representing Class I diffusive reflectance OCT signals from the four-layer medium obtained by OCT-MPS (in red) and previously verified serial simulator (in blue) **[11]**

It can be seen on Figure 8-1 and Figure 8-2 that results of the two simulators are in excellent agreement with each other and compatible with what is expected from a physical OCT system.

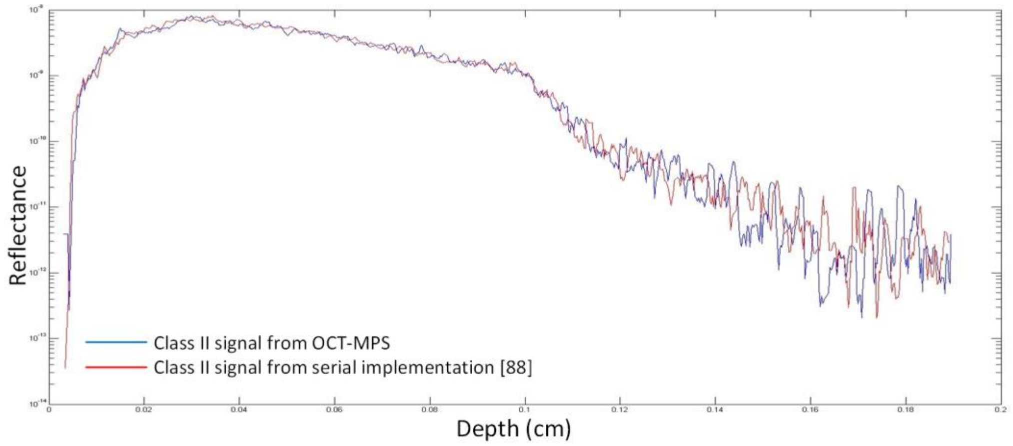

Figure 8-2 A-scans representing Class II diffusive reflectance OCT signals from the four-layer medium obtained by OCT-MPS (in red) and previously verified serial simulator (in blue) **[11]**

The computation of OCT signals by the serial simulator takes 43 minutes on a 2 GHz Intel Core i7 CPU with 4 GB of RAM. OCT-MPT produces the same results in 2.9

minutes on the older Tesla C1060 GPU cards, this shows around 15 times speed increase in the simulation. However, in the newer card K40c card takes only 1.7 minutes, which shows above 25 times speed increase.

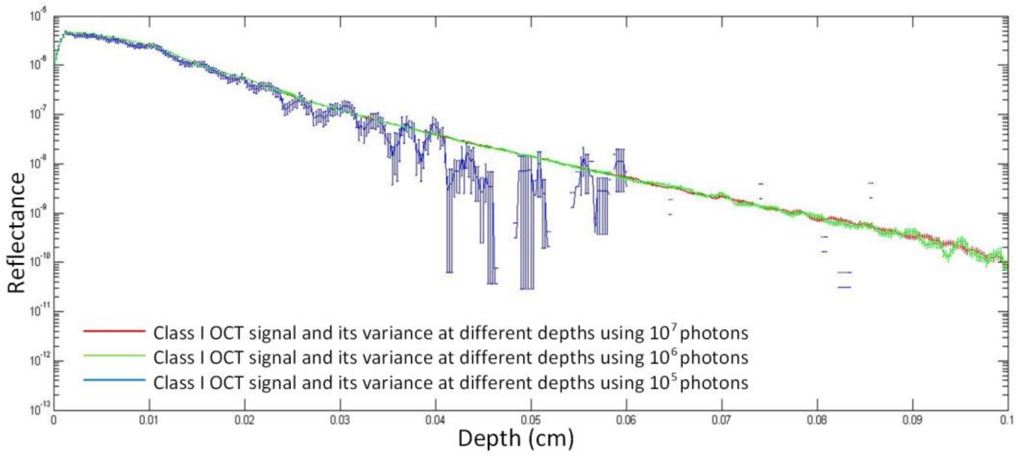

Figure 8-3 Class I OCT signals estimates and their confidence intervals using 10^7 (in red), 10^6 (in green), and 10^5 (in blue) photon packets

The accuracy of Monte Carlo simulations is generally quantified by the variance of obtained results. In Figure 8-3 we show estimates of Class I signals of an A-scan of our four layer object, in addition to confidence intervals (CIs) of these estimates, using different number of photons. The CI of Class I and Class II signals estimates, $\mathrm{CI}_{1,2}(z)$, are defined as

$$\mathrm{CI}_{1,2}(z) \equiv \left[R_{1,2}(z) - \sigma_{1,2}(z), R_{1,2}(z) + \sigma_{1,2}(z)\right].$$

We note from Figure 8-3 that as the imaging depth increases, the CIs increase relative to the signal values. Therefore, as reported by Yao and Wang [10], the accuracy of our signal estimates, relative to signal values, decrease with depth.

8.3 Simulations of OCT signal from a non-layered objects

OCT-MPS implements the ability to simulate signals from non-layered objects, the simulated OCT imaging B-Scans of two non-layered media are presented. First, the medium is a sphere inside a homogeneous slab and second, the medium is a homogeneous slab containing an ellipsoid and two spheres.

8.3.1 Simulated OCT signals from a sphere inside a slab

For this simulation, a sphere of radius 0.1 mm at depth 0.2 mm is placed inside a slab. The slab has 3×3 mm lateral dimensions and 1 mm axial dimension. As shown in Figure 8-4(a). The optical properties of this medium can be found in Table 8-2.

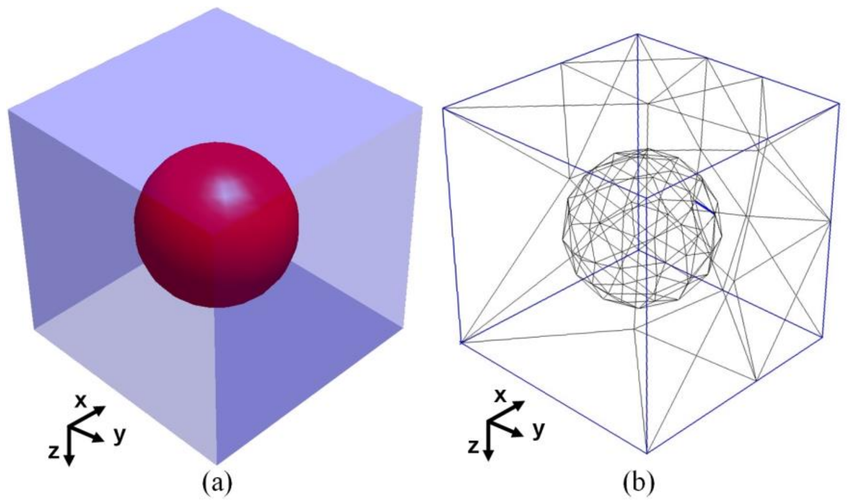

Figure 8-4 Sphere inside a slab medium: (a) Spatial structure, (b) Tetrahedron mesh

Table 8-2 Optical parameters of a non-layered medium with a sphere inside a slab

Medium	Absorption Coefficient μ_a (cm^{-1})	Scattering Coefficient μ_s (cm^{-1})	Anisotropy Factor g	Refractive Index n
Slab	1.5	60	0.9	1
Sphere	3	120	0.9	1

The mesh generator NETGEN [96] was used to generate the tetrahedron mesh with 4437 tetrahedrons and 800 vertices to represent the medium shown in Figure 8-4(b). It was simulated as 512 equidistant A-scans along the x-axis, starting from $x = -0.15$ mm to $x = 0.15$ mm, to obtain the B-scan image.

Note in Figure 8-5(b) the Class II diffusive reflectance proportional intensity increase with depth due to the multiple scattering photon. However, eventually it decreases due to absorption in the medium. The Class II signals are particularly stronger inside the sphere, since its scattering coefficient is higher.

These observations are in accordance to what is expected from physical OCT imaging.

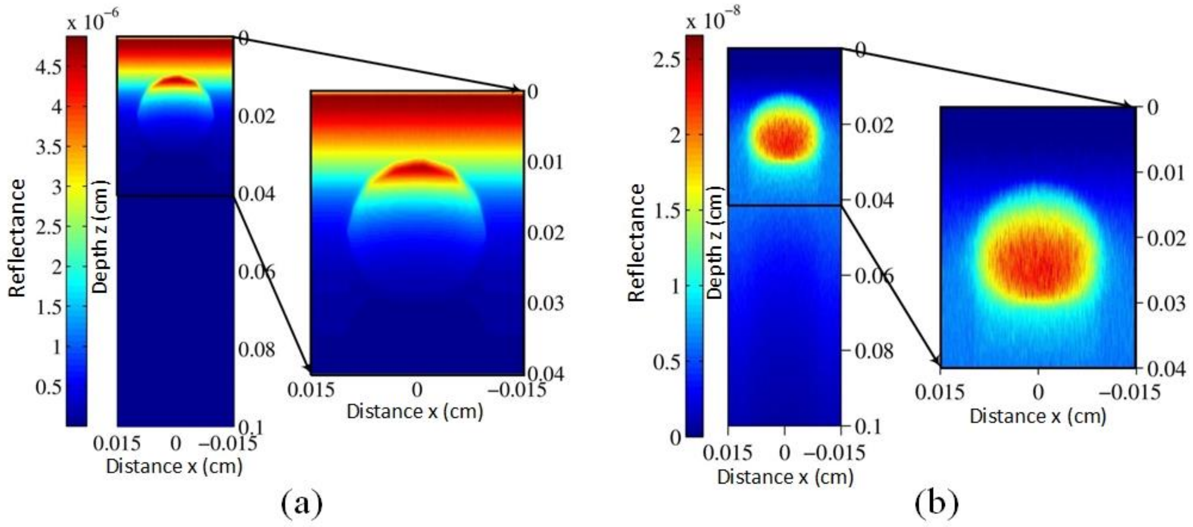

Figure 8-5 Simulated B-scan OCT images of a sphere inside a slab: (a) Class I and (b) Class II reflectance

8.3.1.1 Computational time for simulating one sphere inside a slab

Figure 8-5 shows the simulated B-scan OCT images of this medium for Class I and Class II diffusive reflectance. The serial implementation takes 360 hours to complete [11]. The parallel OCT-MPS implementation takes 19 hours to complete the simulated B-scan on the Tesla C1060 cards but only 11 hours on the K40c card.

8.3.2 Simulated OCT signals from ellipsoid and spheres inside a slab

Another simulated OCT image of a media consisting of a slab with two spheres and an ellipsoid inside it is presented here. The slab is extended for $3\,mm$ in each lateral dimension (x and y-axis) and $1\,mm$ thick in the axial direction.

The ellipsoid is placed at the position $(0.0, 0.0, 0.02)$. The three vectors defining this ellipsoid are $v_1 = [0.006sin(60°),\ 0, -0.006cos(60°)]^T$, $v_2 = [0, 0.1, 0]^T$, and $v_3 = [0.0170sin(30°), 0, 0.0170cos(30°)]^T$.

The two spheres have radius $0.005\,mm$ and are placed at positions $(0.008, 0.0, 0.01)$ and $(-0.008, 0.0, 0.03)$. The abstract view of the medium can be seen in Figure 8-6.

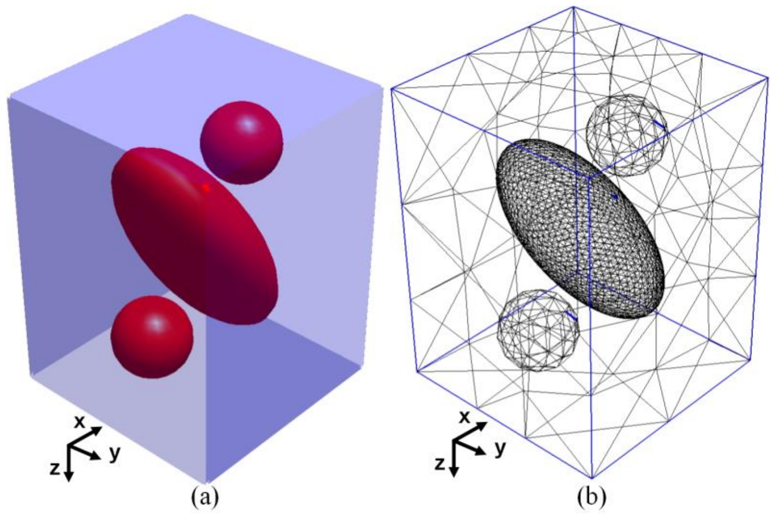

Figure 8-6 Ellipsoid and two spheres inside a slab medium: (a) Spatial structure, (b) Tetrahedron mesh

The optical parameters of this medium can be seen in Table 8-3. NETGEN [96] has been used to generate the tetrahedron mesh of this medium resulting in 5248 vertices and 29697 tetrahedrons. The simulation result of the Class I and Class II OCT B-Scans can be seen in Figure 8-7.

Table 8-3 Optical parameters of the medium consisting of a slab with an ellipsoid and two spheres inside of it

Medium	Absorption Coefficient μ_a (cm^{-1})	Scattering Coefficient μ_s (cm^{-1})	Anisotropy Factor g	Refractive Index n
Slab	1.5	60	0.9	1
Spheres	3	120	0.9	1
Ellipsoid	3	120	0.9	1

Similar to the previous example, it can be seen in Figure 8-7(a) the Class I signal decreases with the depth. The ellipsoid and the sphere placed in the lower depth are visible in the Class I B-Scan, however the sphere at the deeper depth is not observable. Nevertheless, it can be seen in Figure 8-7(b) the boundaries of the regions are mixed but the presence of a second lower sphere is noticeable in the Class II B-scan.

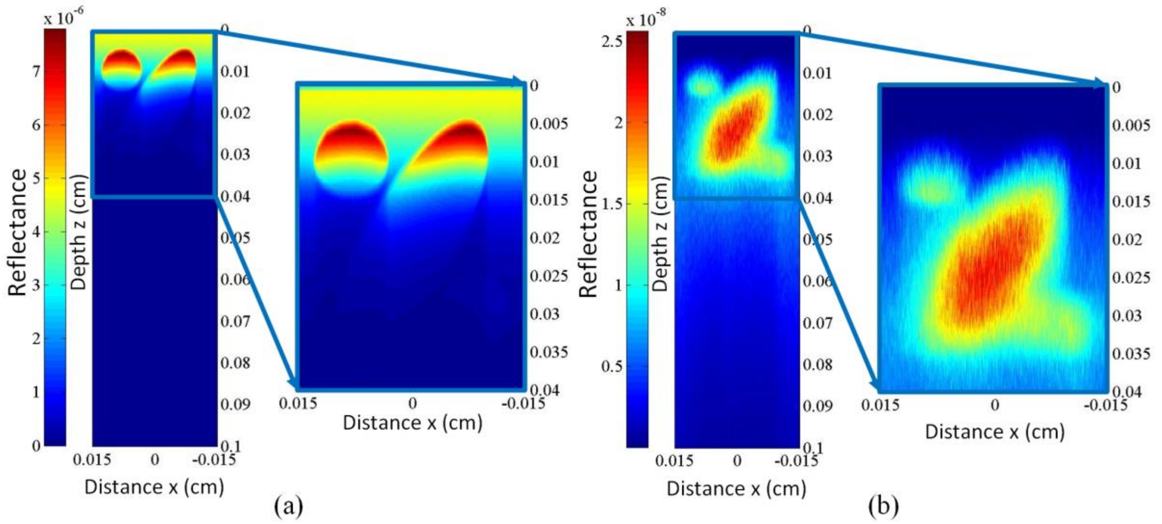

Figure 8-7 Simulated B-scan OCT images of an ellipsoid and two spheres inside a slab: (a) Class I and (b) Class II reflectance

8.3.2.1 Computational time for simulating an ellipsoid and two spheres inside a slab

OCT-MPS completed the simulation of the ellipsoid and two spheres inside a slab in 32.176 hours on the K40c card versus 1358.3 hours taken by the serial implementation.

The computation time per A-scan depends on the optical parameters of the media. For example, in objects with higher scattering coefficient μ_s the simulation will perform an increased number of refraction and reflection calculations which will result in longer processing time. For the region with the lowest scattering coefficient the computation time for an A-scan takes 3.2 minutes in the parallel implementation. The same A-scan takes 115 minutes in the serial implementation. The longest computation time per A-scan performed by OCT-MPS takes 4.4 minutes while in the serial implementation takes above 163 minutes. The average computation time per A-scan is 3.86 minutes on the parallel OCT-MPS implementation.

8.4 Computational time reduction due to our parallel implementation

The computation time gain depends on the complexity of the media to simulate. As seen in our previous parallel implementation results [97], the simulation of a single sphere inside a slab results in only 13 times speed increase for a single A-scan, i.e., from 43 minutes in the serial implementation to 3.3 minutes in the parallel simulator.

The massively parallel OCT-MPS implementation is above 40 times faster than its serial counterpart [11]. This behavior is expected from a parallel implementation since long lived photon packets will be running among newly launched packets and will not affect the overall performance of the simulation.

Chapter 9

Conclusions and suggested future work

9.1 Conclusions

We developed a massively parallel simulator of OCT of inhomogeneous turbid media that we named OCT-MPS. This simulator was implemented on GPUs using NVIDIA's CUDA architecture and extensions of the C language. We exploited the intrinsic parallelism of the latest implementation of MC methods to trace thousands of photon packets simultaneously. We also implemented a known advanced importance sampling technique to further reduce the computation time. We used a tetrahedron-based mesh to simulate arbitrary shaped objects and we implemented a computationally efficient parallel pseudo random number generator.

We showed that an acceleration of up to 40 times can be achieved compared to the available serial OCT simulator for arbitrary shaped objects. The accuracy of our simulation results were verified against previously validated serial simulations.

9.2 Suggested future Work

Our OCT-MPS assumes an infinitely thin incident optical beam. The implementation of an OCT simulator with a beam of finite width will make the simulations even closer in design to implemented OCT systems. Also, light polarization in a simulator will bring it closer to real OCT systems.

The computation time of our OCT massively parallel simulator was greatly improved from its serial counterpart. Nevertheless, even more speed can be achieved by exploiting parallelism within threads themselves. Additionally, the use of other memory spaces available in GPU hardware, e.g., shared memory and texture memory, will allow faster access to data structures. For example, shared memory can be useful for pre-calculation of averages of Class I and Class II signals in the recording block of OCT-MPS eliminating substantially the necessity of access to global memory. Finally, a domain decomposition of the problem could be performed to apply parallel processing on smaller domains incrementing the throughput of simulations.

OCT-MPS currently simulates time domain OCT (TD-OCT). But, it could be extended for swept source OCT (SS-OCT) simulation. SS-OCT simulators require the optical coefficients of the sample at different wavelength.

Bibliography

[1] W. Drexler and J. G. Fujimoto, Optical coherence tomography: technology and applications, New York: Springer, 2008.

[2] D. P. Popescu, C. Flueraru, Y. Mao, S. Chang, J. Disano, S. Sherif and M. G. Sowa, "Optical coherence tomography: fundamental principles, instrumental designs and biomedical applications," *Biophysical Reviews,* vol. 33, pp. 155-169, 2011.

[3] J. Welzel, "Optical coherence tomography in dermatology: a review," *Skin Research and Technology,* vol. 7, no. 1, pp. 1-9, 2001.

[4] M. C. Pierce, J. Strasswimmer, B. H. Park, B. Cense and J. F. de Boer, "Advances in optical coherence tomography imaging for dermatology," *J. Invest. Dermatol.,* vol. 123, no. 3, pp. 458-463, 2004.

[5] W. Drexler, U. Morgner, R. K. Ghanta, F. X. Kartner, J. S. Schuman and J. G. Fujimoto, "Ultrahigh-resolution ophthalmic optical coherence tomography," *Nature medicine,* vol. 7, no. 4, pp. 502-507, 2001.

[6] A. Fercher, C. Hitzenberger, W. Drexler, G. Kamp and H. Sattmann, "In vivo optical coherence tomography," *American journal of ophthalmology,* vol. 116, no. 1, pp. 113-114, 1993.

[7] D. Stifter, "Beyond biomedicine: a review of alternative applications and developments for optical coherence tomography," *Applied Physics B,* vol. 88, no. 3, pp. 337-357, 2007.

[8] G. J. Fujimoto, "Optical coherence tomography for ultrahigh resolution in vivo imaging," *Nature Biotechnology,* vol. 21, no. 11, pp. 1361 - 1367, 2003.

[9] B. Shi, Z. Meng, L. Wang and T. Liu, "Monte Carlo modeling of human tooth optical coherence tomography imaging.," *Journal of Optics,* vol. 15, no. 7, p. 075304, 2013.

[10] G. Yao and L. V. Wang, "Monte Carlo simulation of an optical coherence tomography signal in homogeneous turbid media," *Physics in medicine and biology,* vol. 44, no. 9, pp. 2307-2320, 1999.

[11] S. Malektaji, I. T. Lima Jr. and S. S. Sherif, "Monte Carlo simulation of optical coherence tomography for turbid media with arbitrary spatial distributions," *Journal of Biomedical*

Optics, vol. 19, no. 4, pp. 046001-046001, 2014.

[12] H. Shen and G. Wang, "A tetrahedron-based inhomogeneous Monte Carlo optical simulator," *Physics in medicine and biology*, vol. 55, no. 4, pp. 947-962, 2010.

[13] I. T. Lima, A. Kalra, H. E. Hernandez-Figueroa and S. S. Sherif, "Fast calculation of multipath diffusive reflectance in optical coherence tomography," *Biomedical Optics Express*, vol. 2, no. 5, pp. 692-700, 2012.

[14] L. Meng, B. Lv, S. Zhang and B. Yv, "In vivo optical coherence tomography of experimental thrombosis in a rabbit carotid model.," *Heart (British Cardiac Society)*, vol. 94, no. 6, pp. 777-80, June 2008.

[15] T. Kubo, Y. Ino, T. Tanimoto, H. Kitabata, A. Tanaka and T. Akasaka, "Optical coherence tomography imaging in acute coronary syndromes.," *Cardiology research and practice*, vol. 2011, p. 312978, January 2011.

[16] J. G. Fujimoto, M. E. Brezinski, G. J. Tearney, S. A. Boppart, B. Bouma, M. R. Hee, J. F. Southern and E. A. Swanson., "Optical biopsy and imaging using optical coherence tomography," *Nature medicine*, vol. 1, no. 9, pp. 970-972, 1995.

[17] B. Colston, U. Sathyam, L. DaSilva, M. Everett, P. Stroeve and L. Otis, "Dental OCT.," *Optics Express*, vol. 3, no. 6, pp. 230-8, September 1998.

[18] A. G. Podoleanu, "Optical coherence tomography," *The British Journal of Radiology*, vol. 78, no. 935, pp. 976-988, 2005.

[19] B. Saleh, Ed., Introduction to subsurface imaging, Cambridge: Cambridge University Press, 2011.

[20] B. E. Bouma and G. J. Tearney, Eds., Handbook of Optical Coherence Tomography, Marcel Dekker, 2002.

[21] J. Pawley, Handbook of biological confocal microscopy, Springer, 2010.

[22] S. S. Sherif, C. C. Rosa, C. Flueraru, S. Chang, Y. Mao and A. G. Podoleanu, "Statistics of the depth-scan photocurrent in time-domain optical coherence tomography.," *JOSA A*, pp. 16-20, 2008.

[23] B. E. A. Saleh and M. C. Teich, Fundamentals of Photonics, vol. 45, Wiley-Interscience, 2007, p. 1177.

[24] P. Targowski, "Complex spectral OCT in human eye imaging in vivo," *Optics Communications*, vol. 229, no. 1-6, pp. 79-84, 2004.

[25] B. E. Bouma, G. J. Tearney, B. J. Vakoc and S. H. Yun, Optical Frequency Domain Imaging, W. Drexler and J. G. Fujimoto, Eds., Springer, 2008, pp. 209-237.

[26] J. Ripoll, "Derivation of the scalar radiative transfer equation from energy conservation of Maxwell’s equations in the far field.," *JOSA A,* vol. 28, no. 8, pp. 1765-1775, 2011.

[27] Z. Guo and S. Kumar, "Discrete-ordinates solution of short-pulsed laser transport in two-dimensional turbid media," *Applied Optics,* vol. 40, no. 19, pp. 3156-3163, 2001.

[28] A. H. Hielscher, R. E. Alcouffe and R. L. Barbour, "Comparison of finite-difference transport and diffusion calculations for photon migration in homogeneous and heterogeneous tissues," *Physics in Medicine and Biology,* vol. 43, no. 5, pp. 1285-1302, 1998.

[29] S. Arridge, M. Schweiger, M. Hiraoka and D. Delpy, "A finite element approach for modeling photon transport in tissue," *Medical physics,* vol. 20, no. 2, pp. 299-309, 1993.

[30] J. Mobley and T. Vo-Dinh, "Optical Properties of Tissue," in *Biomedical Photonics Handbook*, CRC press Boca Raton, FL, 2003.

[31] B. E. Salah, Introduction to subsurface imaging, Cambridge University Press Cambridge, 2011.

[32] V. V. Tuchin, "Light-Tissue Interactions," in *Biomedical Photonics Handbook,* CRC press Boca Raton, FL, 2003.

[33] V. V. Tuchin, L. V. Wang and D. A. Zimnyakov, "Tissue Structure and Optical Models," in *Optical Polarizationin Biomedical Applications*, Springer, 2006, pp. 7-28.

[34] A. Dunn and R. Richards-Kortum, "Three-dimensional computation of light scattering from cells," *IEEE Journal of Selected Topics in Quantum Electronics,* vol. 2, no. 4, pp. 898-905, 1996.

[35] J. Schmitt and A. Knuttel, "Model of optical coherence tomography of heterogeneous tissue," *JOSA A,* vol. 14, no. 6, pp. 1231--1242, 1997.

[36] F. Martelli, S. D. Bianco, A. Ismaelli and G. Zaccanti, Light propagation through biological tissue and other diffusive media: Theory, solutions, and software., SPIE Press, 2010.

[37] C. F. Bohren and D. R. Huffman, Absorption and scattering of light by small particles, John Wiley & Sons, 2008.

[38] W. T. Grandy Jr and W. T. Grandy, Scattering of waves from large spheres, Cambridge University Press, 2005.

[39] R. Barer and S. Joseph, "Refractometry of living cells part I. Basic principles," *Quarterly Journal of Microscopical Science,* vol. 3, no. 32, 1954.

[40] T. Vo-Dinh, Biomedical photonics handbook, CRC press, 2010.

[41] E. A. Genina, A. N. Bashkatov and V. V. Tuchin, "Tissue optical immersion clearing," *Expert review of medical devices,* vol. 7, no. 6, pp. 825-842, 2010.

[42] L. G. Henyey and J. L. Greenstein, "Diffuse radiation in the galaxy.," *The Astrophysical Journal.,* vol. 93, pp. 70-83, 1941.

[43] V. Twersky, "Acoustic bulk parameters in distributions of pair-correlated scatterers," *The Journal of the Acoustical Society of America,* vol. 64, no. 6, pp. 1710-1719, 1978.

[44] P. A. Bascom and R. S. Cobbold, "On a fractal packing approach for understanding ultrasonic backscattering from blood," *The Journal of the Acoustical Society of America,* vol. 98, no. 6, pp. 3040-3049, 1995.

[45] H. L. Anderson, "Metropolis, Monte Carlo, and the MANIAC," *Los Alamos Science,* vol. 14, pp. 96-108, 1986.

[46] D. O'Brien, "Accelerated quasi Monte Carlo integration of the radiative transfer equation," *Journal of Quantitative Spectroscopy and Radiative Transfer,* vol. 48, no. 1, pp. 41-59, 1992.

[47] J. Fleck Jr and J. Cummings Jr, "An implicit Monte Carlo scheme for calculating time and frequency dependent nonlinear radiation transport," *Journal of Computational Physics,* vol. 8, no. 3, pp. 313-342, 1971.

[48] L. Wang, S. L. Jacques and L. Zheng, "MCML—Monte Carlo modeling of light transport in multi-layered tissues," *Elsevier,* vol. 47, no. 2, pp. 131-146, 1995.

[49] I. T. Dimov and S. McKee, Monte Carlo methods for applied scientists, London: World Scientific, 2008.

[50] W. Dunn and J. K. Shultis, Exploring Monte Carlo methods, Elsevier, 2011.

[51] F. James, "Monte Carlo theory and practice," *Reports on Progress in Physics,* vol. 43, no. 9, p. 1145, 1980.

[52] T. J. Pfefer, J. Kehlet Barton, E. K. Chan, M. G. Ducros, B. S. Sorg, T. E. Milner, J. S. Nelson and A. J. Welch, "A three-dimensional modular adaptable grid numerical model for light propagation during laser irradiation of skin tissue," *IEEE Journal of Selected Topics in Quantum Electronics,* vol. 2, no. 4, pp. 934-942, 1996.

[53] T. Binzoni, T. Leung, R. Giust, D. Rufenacht and A. Gandjbakhche, "Light transport in tissue by 3D Monte Carlo: influence of boundary voxelization," *Computer methods and programs in biomedicine,* vol. 89, no. 1, pp. 14-23, 2008.

[54] D. Boas, J. Culver, J. Stott and A. Dunn, "Three dimensional Monte Carlo code for photon migration through complex heterogeneous media including the adult human head," *Optics express,* vol. 10, no. 3, pp. 159-170, 2002.

[55] T. Li, H. Gong and Q. Luo, "MCVM: Monte Carlo modeling of photon migration in voxelized media," *Journal of Innovative Optical Health Sciences,* vol. 3, no. 2, pp. 91-102, 2010.

[56] H. Li, J. Tian, F. Zhu, W. Cong, L. V. Wang, E. A. Hoffman and G. Wang, "A mouse optical simulation environment (MOSE) to investigate bioluminescent phenomena in the living mouse with the monte carlo method," *Academic Radiology,* vol. 11, no. 9, pp. 1029-1038, 2004.

[57] D. Cote and I. A. Vitkin, "Robust concentration determination of optically active molecules in turbid media with validated three-dimensional polarization sensitive Monte Carlo calculations," *Optics express,* vol. 3, no. 1, pp. 148-163, 2005.

[58] E. Margallo-Balbs and P. J. French, "Shape based Monte Carlo code for light transport in complex heterogeneous tissues," *Optics express,* pp. 13086-14098, 2007.

[59] N. Ren, J. Liang, X. Qu, J. Li, B. Lu and J. Tian, "GPU-based Monte Carlo simulation for light propagation in complex heterogeneous tissues.," *Optics express,* vol. 18, no. 7, pp. 6811--6823, 2010.

[60] Q. Fang, "Mesh-based Monte Carlo method using fast ray-tracing in Pl{\"u}cker coordinates," *Biomedical optics express,* vol. 1, no. 1, pp. 165-175, 2010.

[61] C. Zhu and Q. Liu, "Review of Monte Carlo modeling of light transport in tissues," *Journal of biomedical optics,* vol. 18, no. 5, pp. 050902-050902, 2013.

[62] D. J. Smithies, T. Lindmo, Z. Chen, J. S. Nelson and T. E. Milner, "Signal attenuation and localization in optical coherence tomography studied by Monte Carlo simulation," *Physics in Medicine and Biology,* vol. 43, no. 10, pp. 3025-3044, 1998.

[63] M. Yadlowsky, J. Schmitt and R. Bonner, "Multiple scattering in optical coherence microscopy," *Applied optics,* vol. 34, no. 25, pp. 5699-5707, 1995.

[64] A. Tycho, T. M. Jorgensen, H. T. Yura and P. E. Andersen, "Derivation of a Monte Carlo

method for modeling heterodyne detection in optical coherence tomography systems," *Applied optics,* vol. 41, no. 31, pp. 6676-6691, 2002.

[65] Z. Song, K. Dong, X. H. Hu and J. Q. Lu, "Monte Carlo simulation of converging laser beams propagating in biological materials," *Applied optics,* vol. 38, no. 13, pp. 2944-2949, 1999.

[66] L. Wang, S. L. Jacques and L. Zheng, "CONV—convolution for responses to a finite diameter photon beam incident on multi-layered tissues," *Computer methods and programs in biomedicine,* vol. 54, no. 3, pp. 141-150, 1997.

[67] I. T. Lima, A. Kalra and S. S. Sherif, "Improved importance sampling for Monte Carlo simulation of time-domain optical coherence tomography," *Biomedical Optics Express,* vol. 2, no. 5, pp. 1069-1081, 2011.

[68] M. Y. Kirillin, A. V. Priezzhev, J. Hast and R. Myllyla, "Monte Carlo simulation of optical clearing of paper in optical coherence tomography," *Quantum Electron,* vol. 36, no. 2, pp. 174-180, 2006.

[69] M. Y. Kirillin, E. Alarousu, T. Fabritius, R. Myllyla and A. V. Priezzhev, "Visualization of paper structure by optical coherence tomography: Monte Carlo simulations and experimental study," *Journal of the European Optical Society-Rapid publications,* vol. 2, p. 07031, 2007.

[70] I. Meglinski, M. Kirillin, V. Kuzmin and R. Myllyla, "Simulation of polarization-sensitive optical coherence tomography images by a Monte Carlo method," *Optics letters,* vol. 33, no. 14, pp. 1581--1583, 2008.

[71] H. Kahn and T. E. Harris, "Estimation of particle transmission by random sampling.," *National Bureau of Standards applied mathematics series,* vol. 12, pp. 27-30, 1951.

[72] M. Y. Kirillin, A. V. Priezzhev and R. A. Myllyla, "Role of multiple scattering in formation of OCT skin images," *Quantum Electronics,* vol. 38, no. 6, pp. 570-575, 2008.

[73] J. J. Mijares-Chan, B. Sharma, J. Lv, G. Thomas, R. Thulasiram and P. Thulasiraman, "True random number generator using GPUs and Histogram equalization techniques," in *IEEE International Conference on High Performance Computing and Communications*, Banff, AB, 2011.

[74] J. E. Gentle, Random number generation and Monte Carlo methods, New York: Springer, 2003.

[75] H. Niederreiter, "New developments in uniform pseudorandom number and vector generation," *Monte Carlo and quasi-Monte Carlo methods in scientific computing,* pp. 87-120, 1995.

[76] J. Eichenauer-Herrmann and H. Niederreiter, "Inversive congruential pseudorandom numbers: distribution of triples," *Mathematics of Computation of the American Mathematical Society,* vol. 66, no. 220, pp. 1629-1644, 1997.

[77] M. Mascagni, D. Ceperley and A. Srinivasan, "SPRNG: A Scalable Library for Pseudorandom Number Generation," in *Parallel Processing for Scientific Computing*, San Antonio, TX, 1999.

[78] GNUOS, "GNU Scientific Library," Free Software Foundation, [Online]. Available: http://www.gnu.org/software/gsl/. [Accessed 1 1 2014].

[79] M. Mascagni and L.-Y. Hin, "Parallel pseudo-random number generators: A derivative pricing perspective with the Heston stochastic volatility model," *Monte Carlo Methods and Applications,* vol. 19, no. 2, pp. 77-105, 2013.

[80] P. L'Ecuyer, Random number generation, Berlin Heidelberg: Springer, 2012.

[81] NVIDIA Corporation, CUDA Programming Guide 6.5, 2014.

[82] J. Hoberock and N. Bell, "Thrust: A parallel template library," 2010.

[83] S. Gao and G. D. Peterson, "GASPRNG: GPU accelerated scalable parallel random number generator library," *Computer Physics Communications,* vol. 184, no. 4, pp. 1241-1249, 2013.

[84] J. Passerat-Palmbach, C. Mazel, B. Bachelet and D. R. C. Hill, "ShoveRand: A model-driven framework to easily generate random numbers on GP-GPU," *Simulations (HPCS),* pp. 41-48, 2011.

[85] D. o. Mathematics, "TinyMT," Hiroshima University, 16 9 2011. [Online]. Available: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/CUDA/index.html. [Accessed 01 01 2014].

[86] D. B. Thomas, "The Warp Generator : A Uniform Random Number Generator for GPUs," Imperial College, [Online]. Available: http://cas.ee.ic.ac.uk/people/dt10/research/rngs-gpu-warp_generator.html. [Accessed 1 1 2014].

[87] M. Saito and M. Marsumo, "Variants of Mersenne twister suitable for graphic processors," *Transactions on Mathematical Software,* vol. 39, no. 2, 2013.

[88] J. Cheng, M. Grossman and T. McKercher, Professional CUDA C Programming, John Wiley & Sons, 2014.

[89] D. Kirk and W. Hwu, Programming Massively Parallel Processors, Morgan Kaufmann, 2010.

[90] J. Sanders and E. Kandrot, CUDA by Example, Addison-Wesley, 2011.

[91] S. Cook, CUDA Programming, Morgan Kaufmann, 2013.

[92] N. Wilt, The CUDA Handbook: A comprehensive guide to GPU programming, Pearson Education, 2013.

[93] M. J. Quinn, Parallel Computing: Theory and Practice, McGraw-Hill, 1994.

[94] E. Alerstam, W. C. Y. Lo, T. D. Han, J. Rose, S. Andersson-Engels and L. Lilge, "Next-generation acceleration and code optimization for light transport in turbid media using GPUs," *Biomedical Optics Express,* vol. 1, no. 2, pp. 658-675, 2010.

[95] T. Misfeldt, The Elements of C++ Style, Cambridge University, 2004.

[96] G. Marsaglia, "Random number generators," *J. Mod. Appl. Stat. Meth.,* vol. 2, no. 1, pp. 2-13, 2003.

[97] E. Alerstam, T. Svensson and S. Andersson-Engels, "Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration," *Journal of biomedical optics,* vol. 13, no. 6, p. 060504, 2008.

[98] J. Schoberl, "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules," *Computing and visualization in science,* vol. 1, no. 1, pp. 41-52, 1997.

[99] M. R. Escobar I., S. Malektaji, I. T. Lima and S. S. Sherif, "Accelerated simulation of optical coherence tomography of objects with arbitrary spatial distributions," in *Proc. SPIE 9288, Photonics North 2014,* Montreal, 2014.

[100] J. A. Izatt and M. A. Choma, "Theory of optical coherence tomography," in *Optical Coherence Tomography,* Springer Berlin Heidelberg, 2008, pp. 47-72.

[101] W.-F. Cheong and S. A. W. A. J. Prahl, "A review of the optical properties of biological tissues," *IEEE journal of quantum electronics,* pp. 2166--2185, 1990.

[102] R. K. Wang, "Signal degradation by multiple scattering in optical coherence tomography of dense tissue: a Monte Carlo study towards optical clearing of biotissues," *Physics in medicine and biology,* vol. 47, no. 13, p. 2281–2299, 2002.

[103] P. F. Pereira and S. S. Sherif, "Design of an optimum ultrasound pattern to minimize

multiple-scattered light reflected from inhomogeneous tissue," in *Proc. SPIE*, 2012.

[104] M. Kirillin, I. Meglinski, V. Kuzmin, E. Sergeeva and R. Myllyla, "Simulation of optical coherence tomography images by Monte Carlo modeling based on polarization vector approach," *Optics express,* vol. 18, no. 21, pp. 21714--21724, 2010.

[105] V. Tuchin, "Tissue optics," *SPIE Tutorial Texts in Optical Engineering light Scattering Methods and Instrumentation for Medical Diagnosis, TT38,* 2000.

[106] A. Kalra, I. T. Lima and S. S. Sherif, "Almost instantaneous Monte Carlo calculation of optical coherence tomography signal using graphic processing unit," in *IEEE Photonics Conference (IPC)*, 2013.

[107] Q. Fang and D. A. Boas, "Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units.," *Optics express,* vol. 17, no. 22, pp. 20178-20190, 2009.

[108] Q. Fang and D. A. Boas, "Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units," *Optics express,* vol. 17, no. 22, pp. 20178-20190, 2009.

[109] T. S. Leung and S. Powell, "Fast Monte Carlo simulations of ultrasound-modulated light using a graphics processing unit," *Journal of biomedical optics,* vol. 15, no. 5, p. 055007, 2010.

[110] M. Choma, M. Sarunic, C. Yang and J. Izatt, "Choma, Michael and Sarunic, Marinko and Yang, Changhuei and Izatt, Joseph," *Optics Express*, vol. 11, no. 18, pp. 2183-2189, 2003.

[111] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS),* vol. 8, no. 1, pp. 3-30, 1998.

[112] L. L. Carter and E. D. Cashwell, "Particle-transport simulation with the Monte Carlo method," *TID-26607, US Enery Research and Development Administration,* 1975.

[113] M. Galassi, J. Davies, J. Theiler, B. Gough and G. Jungman, “Gnu Scientific Library: Reference Manual.,” Network Theory Ltd., 2003.