MASTER

Monte Carlo simulation of photon transport in tissue verification and augmentation towards complex configurations

van Belkum, J.A.J.

Award date: 2008

Link to publication
Monte Carlo Simulation of Photon Transport in Tissue
verification and augmentation towards complex configurations

by
J.A.J. van Belkum

Supervisor:
prof. dr. P.A.J. Hilbers

Committee:
prof. dr. ir. J.J. van Wijk
prof. dr. M.G.J. van den Brand

Supervisors:
dr. drs. K.K. Thumma
ir. S.C.B.J Vrijaldenhoven

AUGUST 2008
Abstract

Skin is a complex structure composed of many elements, making light based skin diagnostics challenging. To better understand photon transport in tissue, simulations are performed using a physical/numerical Monte Carlo Model. The Monte Carlo Optical Model simulation software (MCOM) is analyzed in this thesis. Issues, resulting from agglomeration of different software versions, are addressed and corrected. A verification framework using Unit and Acceptance tests is designed and implemented, restoring confidence in the software. The corrected and verified software is taken as the base for enhancements towards complex configurations. A new method for specifying arbitrary placement of multiple source/detection fibers is designed and implemented. Results from testing shows issues dealing with reflection at boundaries, that are still under investigation. Furthermore, a parallel solution for complex simulations is researched, where a hybrid decomposed solution, consisting both of task farm and spatial decomposition, is most optimal for simulations with high memory and execution time requirements.
Acknowledgment

First of all, I thank Peter Hilbers for guiding me through this project, even after becoming dean of the department of Biomedical Engineering. I am grateful for the time and energy he invested. Many thanks also to my two supervisors at Philips Research, Kiran Thumma and Serge Vrijaldenhoven. Both were helpful in many ways, each in their own area of expertise. Their enthusiasm, and guidance kept me focused towards successful completion of the project. Also Golo von Basum, and Arie Kaizer invested much time in making this project a success. Furthermore both the Philips Research ICT ADICT and Philips Research Care & Health Application group members have supported me with information and constructive discussions.

Finally a special thanks goes to my Family and girlfriend. They have supported me greatly during the last five years, during which there were times when I struggled to continue this study. They have stood by me, urged me to move forward, knowing I would regret not completing the educational program.
# Contents

## Acknowledgment

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>iii</td>
</tr>
</tbody>
</table>

## Glossary

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>viii</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Problem Statements ......................................................... 2
   1.1.1 Software Confidence through Verification ............................. 2
   1.1.2 Complexity Enhancement for Elaborate Simulations ................... 2

## 2 Analysis

2.1 History ........................................................................... 5

2.2 MCOM Theory ................................................................. 5
   2.2.1 Photon Representation ................................................... 6
   2.2.2 Photon Launch ............................................................. 6
   2.2.3 Step size .................................................................. 6
   2.2.4 Absorption ................................................................ 8
   2.2.5 Scattering ................................................................ 8
   2.2.6 Roulette ................................................................ 9
   2.2.7 Boundary Conditions ................................................... 10
   2.2.8 Simulation Termination .................................................. 11

2.3 Random Number Generator ................................................. 12

2.4 Input .............................................................................. 12
   2.4.1 Batch Input File .......................................................... 12
   2.4.2 Wavelength Database File .............................................. 13

2.5 Implementation ................................................................. 13
   2.5.1 Simulation Process ....................................................... 13
   2.5.2 Tissue Representation .................................................... 16
   2.5.3 Classes ................................................................... 16

2.6 Output ........................................................................... 18
   2.6.1 Constant Simulation Information ...................................... 18
   2.6.2 Current Run Information ................................................ 18
   2.6.3 3D absorption matrix .................................................... 18
   2.6.4 2D reflection matrix and 2D transmission matrix ................. 19
   2.6.5 Photon Traces .............................................................. 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>Complexity</td>
<td>19</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Run Time Complexity</td>
<td>19</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Memory Complexity</td>
<td>21</td>
</tr>
<tr>
<td>2.7.3</td>
<td>Cyclomatic Complexity</td>
<td>22</td>
</tr>
<tr>
<td>2.8</td>
<td>Data Processing</td>
<td>22</td>
</tr>
<tr>
<td>2.9</td>
<td>Issues &amp; Improvements</td>
<td>25</td>
</tr>
<tr>
<td>2.9.1</td>
<td>Standard Libraries</td>
<td>25</td>
</tr>
<tr>
<td>2.9.2</td>
<td>Configuration Files &amp; Variables</td>
<td>25</td>
</tr>
<tr>
<td>2.9.3</td>
<td>Reduction of Complexity</td>
<td>26</td>
</tr>
<tr>
<td>2.9.4</td>
<td>Object Oriented Design</td>
<td>26</td>
</tr>
<tr>
<td>2.9.5</td>
<td>Conservation of Energy</td>
<td>29</td>
</tr>
<tr>
<td>2.9.6</td>
<td>RNG</td>
<td>29</td>
</tr>
<tr>
<td>2.9.7</td>
<td>Output Files</td>
<td>30</td>
</tr>
<tr>
<td>2.9.8</td>
<td>Documentation</td>
<td>31</td>
</tr>
<tr>
<td>2.9.9</td>
<td>Visualization</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>Verification</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Unit Tests</td>
<td>33</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Implementation Cycle</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>Acceptance Tests</td>
<td>34</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Optical Properties Tests</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Literature Tests</td>
<td>37</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Phantom Tests</td>
<td>39</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Conclusion</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>Complex Configuration Implementation</td>
<td>51</td>
</tr>
<tr>
<td>4.1</td>
<td>Matrix implementation</td>
<td>52</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Existing Software</td>
<td>52</td>
</tr>
<tr>
<td>4.1.2</td>
<td>User Requirements</td>
<td>53</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Design</td>
<td>53</td>
</tr>
<tr>
<td>4.1.4</td>
<td>UML</td>
<td>59</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Testing</td>
<td>61</td>
</tr>
<tr>
<td>4.1.6</td>
<td>Conclusion</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Parallel implementation</td>
<td>70</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Analysis</td>
<td>70</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Random Number Generator</td>
<td>74</td>
</tr>
<tr>
<td>4.3</td>
<td>Conclusion</td>
<td>75</td>
</tr>
<tr>
<td>5</td>
<td>Concluding Remarks</td>
<td>77</td>
</tr>
<tr>
<td>5.1</td>
<td>Future Enhancements</td>
<td>78</td>
</tr>
<tr>
<td>Project Reflection</td>
<td></td>
<td>81</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>83</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
<td>87</td>
</tr>
</tbody>
</table>
A  Current Implementation  
A.1  Input .................................................................................................................. 87
  A.1.1  Batch File Variables ..................................................................................... 87
  A.1.2  Layer Database Variables ............................................................................. 90
A.2  Output .................................................................................................................. 91
  A.2.1  Constant Simulation .................................................................................... 91
  A.2.2  Current Run .................................................................................................. 92
  A.2.3  3D Absorption .............................................................................................. 92
  A.2.4  2D Reflectance & Transmittance ................................................................ 92
  A.2.5  Photon Traces ............................................................................................... 92

B  Verification .............................................................................................................. 93
B.1  Unit Testing .......................................................................................................... 93
  B.1.1  Class CArea .................................................................................................. 93
B.2  Acceptance Tests .................................................................................................. 97

C  Matrix Implementation ............................................................................................ 103
C.1  Input .................................................................................................................... 103
C.2  Output .................................................................................................................. 105
  C.2.1  Constant Simulation .................................................................................... 105
  C.2.2  Current Run .................................................................................................. 106
  C.2.3  3D Absorption .............................................................................................. 107
  C.2.4  2D Reflectance & Transmittance ................................................................ 107
  C.2.5  Photon Traces ............................................................................................... 107
Glossary

The following list of terms are used throughout the document, and referenced by their abbreviations.

MC         Monte Carlo
MCOM       Monte Carlo Optical Model
$\mu_a$    Absorption coefficient
$\mu_s$    Scattering coefficient
$\mu_t$    Attenuation coefficient, is equal to $\mu_a + \mu_s$
MFP        Mean free path is defined as the distance a photon travels between interactions. Determined by taking the reciprocal of $\mu_t(\mu_t^{-1})$.
g         Anisotropy
n         Refractive index
NIR        Near Infra Red light spectrum
RNG        Random Number Generator
$\xi$      Uniform distributed random number in range $[0, 1)$
random seed RNG initial seed
packet     Weighted packet of statistical photons
grid       Cluster of multiple computers facilitating parallel execution of software.
Chapter 1

Introduction

Skin is a complex structure, composed of many components, each having its own characteristics (Fig. 1.1). Performing light based skin diagnostics is challenging because of the complex structure. In order to perform simulations of these diagnostics, a physical/numerical model is created. The Monte Carlo Optical Model (MCOM) implements a Monte Carlo algorithm, simulating trajectories of photons through layers of biological medium. This software package is developed and used within Philips Research Care & Health Applications (Philips Research C&HA), using the work by Wang & Jacques [1] as a base.

Fig. 1.1 – Two dimensional skin representation
The Monte Carlo method is widely used in the world for simulation of physical models, ranging from light dosimetry determination in radiation therapy [2], to ray tracing in optical canopy reflectance modeling [3].

Within Philips Research C&HA the initial version of the software was created in the programming language Pascal. Recently that version was converted into C++, using parts of code that originate from the original ANSI C code. Rewriting the original code into C++ was done by a student. The new implementation is not completely finished and requires adjustments. This thesis deals with finishing and adapting the existing code towards particular applications for Philips Research C&HA.

This document consists of the following parts:
- Chapter 1: Introduction and problem description
- Chapter 2: Monte Carlo Optical Model theory, description of software, analysis and corrections of issues found.
- Chapter 3: Description of verification framework, and executed tests.
- Chapter 4: Implementation of simulation matrix and parallelization investigation.
- Chapter 5: Concluding Remarks

1.1. Problem Statements

1.1.1 Software Confidence through Verification

During implementation of the previous code, no detailed verification of the software was implemented. This lack of testing leads to a decreased confidence in the results produced by the MCOM software. Changing the existing software, or implementing new features, requires a correct and stable software base.

An increase in confidence is achieved by implementing a verification framework, that verifies the correct functioning of the MCOM software. If problems are found using this framework, they are addressed and corrected.

Verification of the software is done in two steps. One part of this framework focuses on the correct implementation of the MCOM physics and theory. This is done using Unit and integration tests. The second part verifies the correct functioning of the code as a whole using acceptance tests, subdivided in three categories: Reasoned tests, Literature Comparison tests, and Phantom tests.

The results, produced by the verification framework, indicate the correct functioning of the software, only when all created tests are successfully completed.

1.1.2 Complexity Enhancement for Elaborate Simulations

Within the MCOM software it is possible to define one source of photons and (optionally) one photon detection fiber. This configuration does not give the user sufficient flexibility to run elaborate
simulations using more than one source or detector. Implementing such a configuration, where a more flexible approach for simulation is required, is requested by Philips Research C&HA.

Implementation of new features should not lead to a decrease in performance. Adding more complexity to the existing software usually leads to an increase in the overall execution time. To overcome this, the current implementation of the simulation is adapted to run efficiently on a grid.

The original implementation in C++ only allows execution on a single CPU, running simulations for each requested wavelength sequentially. The execution time of a complete simulation lasts as long as the sum of the execution times for each separate wavelength. Porting the code to the UNIX platform enables the simulation to run each separate wavelength in parallel. This change results in an improvement of execution time, depending on the wavelength requiring the most execution time (when the number of processes equals the number of wavelengths). This parallel solution is not the most optimal solution for achieving the fastest execution time of the MCOM software.

The wishes of Philips Research C&HA, and future enhancements, require further optimization of execution time. Currently bounded by the longest running wavelength, implementing parallelization should achieve an execution time no more than the average execution time of all wavelengths. Achieving this reduced execution time, requires some other scheme, splitting wavelengths over multiple processors.

The second part of the enhancement focuses on finding a suitable solution for increasing the performance of the MCOM software enabling the implementation of the requested enhancements. Analysis, requirements and improvements are discussed in Chapter 4.
Chapter 2

Analysis

Within Philips Research C&HA MCOM is used for many years, during which the software is updated to keep up with new standards in programming and hardware capabilities. This chapter gives an overview of the current implementation and its analysis. The final section of the chapter focuses on issues discovered and their solutions.

2.1. History

The current C++ implementation is largely based on two existing applications. The first application is a design by Lihong Wang and Steven L. Jacques. Their implementations in PASCAL and later in ANSI C are found on the Internet and are free for public use. The second application in Pascal, based on the Steven L. Jacques design, is further developed by Marleen Keijzer (TUDelft, Netherlands) and Gerald Lucassen (AMC, Netherlands). These two implementations are combined by Hugo Wilms (Fontys Hogeschool Eindhoven) into the current C++ software package.

This implementation is only executable on the Windows platform, where a simple simulation (40 wavelengths, 1 input fiber, 1 output fiber) requires close to three days of execution time. In order to decrease the execution time, the Philips Research Adaptive ICT (ADICT) group ported the software to UNIX. This new application facilitated the splitting of the simulation by wavelength, and executing each separately on a different processor. This improved the simple simulation execution time to about an hour.

The UNIX ported software is taken as the base for the rest of this document. The windows part is completely removed, and not taken into account anymore. The future of the software requires more than one computer for allowing the software to complete within an acceptable time-frame. This is not achievable on the current windows environment.

2.2. MCOM Theory

The Monte Carlo method uses random sampling from well-defined probability distributions to model the actions of a photon ([4],[5],[6]). An excellent description of the Monte Carlo method, the random sampling process, and the explicit physics involved in the simulation are found in the manual.
of the original ANSI C implementation [7].

The basis of this method is described by Prahl [8]. From these resources the important parts for photon propagation are explained in this section. A flowchart depicting the MCOM is shown in Fig. 2.1.

### 2.2.1 Photon Representation

MCOM does not treat the photon as a wave phenomenon. Although phase and polarization phenomena can be represented by Monte Carlo methods, it does not play an important role in the focus of this simulation: energy transport.

The method depicts photons as statistical packets with an initial weight of 1, representing multiple individual photons. This allows a more efficient simulation. Variance reduction techniques, such as weighting, are used to reduce the number of photons necessary to achieve the desired accuracy for a Monte Carlo calculation. Due to the probability of absorption, the photon packet loses some of its weight after each propagation step. Representing the photon as a packet increases the number of tissue interactions before the photon has no more energy left and loses its usefulness.

At any point in the simulation a photon packet has three properties associated with it: position, direction, and weight. The position of a photon is specified using three Cartesian coordinates: $x$, $y$, and $z$. The photon packet direction of propagation is specified using direction cosines $(U_x, U_y, U_z)$, and weight is specified by $W$. The representation is shown in Fig. 2.2.

Within the rest of this report both the term photon packet, and photon are used. Unless specified otherwise, both terms refer to the statistical photon packet as explained in this section.

### 2.2.2 Photon Launch

The initial position and direction of a photon are determined upon launch, using the chosen source type properties. Most of the sources apply a random sampling technique to distribute the photon launch locations equally over the source area. The direction is calculated using the properties defined for the source type, and a random sampling to distribute the directions equally.

After launch of the photon, if there is a mismatch in boundary at the tissue surface, specular reflection occurs. The amount of specular reflection is determined by Equation 2.1, where the refractive indexes of the tissue and the outside medium are $n_1$ and $n_2$. The weight of the photon packet is then determined by Equation 2.2

$$R_{sp} = \frac{(n_1 - n_2)^2}{(n_1 + n_2)^2}$$  

$$W = 1 - R_{sp}$$

### 2.2.3 Step size

The distance a single photon (not a packet) travels in the tissue between interaction points is chosen randomly according to the optical properties of the tissue. This method of determining the next
Fig. 2.1 – Monte Carlo method flowchart
Fig. 2.2 – Photon representation in Cartesian coordinates, and directional cosines

location, forces a single photon to either be absorbed or scattered at the new location. The actual step size $\Delta s$ is determined by:

$$\Delta s = -\frac{\ln(\xi)}{\mu_a + \mu_s} \quad (2.3)$$

The new location $(x', y', z')$ is then calculated using the previous coordinates, the directional cosines and $\Delta s$:

$$x' = x + U_x \Delta s$$
$$y' = y + U_y \Delta s$$
$$z' = z + U_z \Delta s \quad (2.4)$$

At each next interaction point $(x', y', z')$, a single photon is either absorbed or scattered. However, the photon packet representation used in the simulation causes both actions to occur at this point for the packet.

2.2.4 Absorption

Absorption of a photon is represented by the photon packet losing weight, and is dependent on the scattering and absorption coefficients of the interacting tissue. Calculation of the weight fraction lost, $\Delta W$, is done using the following formula:

$$\Delta W = W \cdot \frac{\mu_a}{\mu_a + \mu_s} \quad (2.5)$$

Which is then subtracted from the previous weight, $W$, to set the new weight of the packet $W'$:

$$W' = W - \Delta W \quad (2.6)$$

2.2.5 Scattering

After the photon has lost weight due to absorption, only when $\mu_a$ is not 0, the photon packet is given a new direction as the result of a scatter event (Fig. 2.3). This new direction, represented by the
scattering angle $\theta$, is determined using the Henyey-Greenstein phase function [9]:

$$p_{HG}(\cos \theta) = \frac{1 - g^2}{2 \left( 1 + g^2 - 2g \cos \theta \right)^{3/2}}$$

(2.7)

The phase function is sampled uniformly using a random value between zero and one, $\xi$, this leads to:

$$\cos \theta = \begin{cases} \frac{1}{2g} \left[ 1 + g^2 - \left( \frac{1 - g^2}{1 - g + 2g \xi} \right)^2 \right] & \text{if } g \neq 0 \\ 2\xi - 1 & \text{if } g = 0 \end{cases}$$

(2.8)

Where the calculation of $\theta$ is split into two cases: one where the anisotropy is equal to zero, and another for all other cases. The azimuthal angle, $\psi$, is uniformly distributed over the range $0$ to $2\pi$, sampled using:

$$\psi = 2\pi \xi$$

(2.9)

Using the calculated $\psi$ and $\theta$, the new directional cosines are calculated using the following equations:

$$U'_x = \frac{\sin \theta (U_y U_z \cos \psi - U_z \sin \psi)}{\sqrt{1 - U_x^2}} + U_x \cos \theta$$

$$U'_y = \frac{\sin \theta (U_x U_z \cos \psi - U_z \sin \psi)}{\sqrt{1 - U_y^2}} + U_y \cos \theta$$

$$U'_z = -\sin \theta \cos \psi \sqrt{1 - U_z^2} + U_z \cos \theta$$

(2.10)

### 2.2.6 Roulette

If photon weight falls below the, user defined, critical weight, the roulette function determines if the photon is allowed to continue or terminated. Termination is determined by a random variable $\xi$. If $\xi$ is less than the user defined survival chance, the photon weight is reduced to 0 and is terminated.

When a photon is not terminated, the weight of the photon is increased by setting the new weight of the photon to its current weight divided by the survival chance. This is done to compensate for the
loss of weight when a photon is terminated. Overall this addition of weight compensates for loss of weight in order to sustain the conservation of energy.

2.2.7 Boundary Conditions

The photon moves freely through tissue until change in optical properties occurs. That point is a boundary between two different media (Fig. 2.4). This boundary can be between two media in the simulation area, or between the surrounding medium and a medium in the simulation area. When a step size $\Delta s$ is large enough to cross a boundary between media, it is split up in two parts.

$$\Delta s = \Delta s_{\text{to boundary}} + \Delta s_{\text{remaining}} \quad (2.11)$$

The new position $(x', y', z')$ of the photon is determined using:

$$x' = x + U_x \Delta s_{\text{to boundary}}$$

$$y' = y + U_y \Delta s_{\text{to boundary}}$$

$$z' = z + U_z \Delta s_{\text{to boundary}} \quad (2.12)$$

After translation to the boundary, the probability of the photon being reflected internally is calculated. Calculation depends on the angle of incidence, $a_i$, onto the boundary. The angle is calculated from the directional cosine $U_z$:

$$a_i = \cos^{-1}(|U_z|) \quad (2.13)$$

The critical angle, $a_c$, for the boundary determines if the photon is completely internally reflected. The critical angle is calculated by:

$$a_c = \sin^{-1}(n_t/n_i) \quad (2.14)$$

Where $n_t, n_i$ are the refractive indexes of the transmission medium and incidence medium respectively. If a fraction of weight is transmitted to the new medium, the new angle, $a_t$, is determined by Snell’s law:

$$n_t \sin a_i = n_i \sin a_t \quad (2.15)$$

The reflected part of the photon, $R(a_i)$, is determined by Fresnels’s laws:

$$R(a_i) = \begin{cases} 
\left( \frac{n_i - n_t}{n_i + n_t} \right) & \text{for } a_i = 0, \\
\frac{1}{2} \sin^2(a_i - a_t) + \tan^2(a_i - a_t) & \text{for } 0 < a_i < a_c, \\
1 & \text{for } a_c \leq a_i.
\end{cases} \quad (2.16)$$

The next step of the photon is determined by the type of boundary. It is positioned at either an outer boundary, or an internal boundary. The distinction between these is below:

- When the photon is positioned at the outer boundary, a part of the photon, $T(a_i)$, is transmitted into the surrounding medium:

$$T(a_i) = 1 - R(a_i) \quad (2.17)$$

The new weight of the photon, $W'$, is updated to the reflected part of the photon’s weight, $W$:

$$W' = W \times R(a_i) \quad (2.18)$$

The new direction is chosen by reversing the directional cosine $U_z$. 

10 CONFIDENTIAL
The new path and weight of a photon at an inner boundary is determined by a random variable $\xi$. If $\xi \leq R(a_i)$, the photon is reflected internally and the new weight is determined by Equation 2.18.

Otherwise, the photon is transmitted to the transmission medium. The new weight after transmission is determined by:

$$W' = W \times (1 - R(a_i))$$

The new step size is determined using the attenuation coefficient of the transmission medium $\mu_t$:

$$\Delta s = \frac{\Delta S_{\text{remaining}}}{\mu_t}$$

The new directional cosines, $U'_{x, y, z}$, are determined by:

$$U'_{x} = U_x \frac{\sin a_t}{\sqrt{1 - U^2_z}}$$

$$U'_{y} = U_y \frac{\sin a_t}{\sqrt{1 - U^2_z}}$$

$$U'_{z} = \cos a_t \frac{z_{\text{boundary}} - z}{|z_{\text{boundary}} - z|}$$

Fig. 2.4 – Boundary conditions

### 2.2.8 Simulation Termination

Launching of photons is done in groups, called seeds, with a user defined size. The maximum number of seeds processed is dependent on the use of a detector.

If no detector is used, only one seed is processed, and the simulation terminates when the number of photons in that seed is launched.

When a detector is defined and used during simulation, the maximum number of seeds processed is 10000 (fixed in software). This hard limit is created to prevent infinite run time. In order to obtain trustworthy results in less time, the simulation tracks the ratio of detected photons per seed. The mean of all completed seeds is determined, as well as the standard deviation (STD) from this mean after the
completion of each seed. The corrected STD is calculated using the Student’s t-distribution (\([10],[11]\)), with a 95% confidence level. The obtained corrected STD is compared to the user defined lower limit for the STD. When the calculated STD is less than the defined STD, the simulation terminated before the maximum number of photons is launched.

### 2.3. Random Number Generator

Currently, the subtractive lagged Fibonacci ran3 algorithm (based on Knuth’s subtractive method [12]), is used for the generation of random numbers. The specific implementation is taken from Numerical Recipes in C++ [13]. This algorithm has a sufficient large period of \(2^{55}\) for current simulations. Launching a photon requires at most 6 random numbers, while photon propagation requires at most 4. Maximally a single simulation run requires:

\[
N_w \times \max(N_p)(6 + \max(N_s \times 4))
\]

(2.22)

where \(N_w\) is the number of wavelengths simulated, \(N_p\) is the number of photons launched, and \(N_s\) is the number of steps taken during one photon trajectory. As an example, average steps: \(N_s = 150\), max photons: \(N_p = 50000000\), wavelengths: \(N_w = 40\) leads to a total of \(1.212 \times 10^{12}\) random numbers required.

### 2.4. Input

The MCOM software is broken down into three main parts:

- Input
- Simulation
- Output

The interaction between these three components is shown in Fig. 2.5. This section deals with the first part: Input.

Input to the software is given by files, using a comma separated format. These files are either generated in a text editor, spreadsheet software or by a program written in Matlab. There are two different types of files. One is the batch input file, and the second is a wavelength database file. This second file is optional, and the parameters used can be specified in the batch input file as well.

#### 2.4.1 Batch Input File

The batch input file consists of multiple columns. The first, mandatory, column contains the variable name. This variable is connected to value(s) defined in the following column(s). Each of the following columns defines parameters for a single simulation run. It is possible to define multiple simulation runs by adding new columns in the file. There is no limit on the number of columns.

All the possible variables and their properties are listed in Appendix A.1.1.
2.4.2 Wavelength Database File

The wavelength database file consists of multiple rows. All fields in a row are separated by a comma. The first, mandatory, row in the file contains the variable names, the rest of the rows contain data.

Each row specifies tissue properties of the associated skin layer from the batch file. These rows of data can also be specified in the Batch Input File (Section 2.4.1). An explanation of the variables is in Appendix A.1.2.

2.5. Implementation

The second part in the MCOM software is the actual simulation, as depicted in Fig. 2.5. This section introduces the code and program flow that make up the simulation process.

2.5.1 Simulation Process

The MCOM simulation process is depicted in the activity graph Fig. 2.6. Simulation runs, wavelengths, and photon steps are all processed sequentially. Within the activity graph the simulation activity is traceable by following the edges in the graph. Each guard states the precondition that needs to hold in order to allow traversal of the edge in the direction of the arrow.

The photon propagation node in the graph is the part of the simulation where most work takes place. The breakdown of this node is shown in Fig. 2.7. The life cycle of a photon starts with the launch, where the initial direction and position are determined according to the properties of the source. A distinction is made between photon propagation in glass and not in glass. In glass there is no absorption or scattering. Photons traverse the complete thickness of glass in one step, placing them at the next boundary. At this point refraction and transmission are determined.
When a photon traverses through tissue, an explicit call to 'HitBoundary' (Fig. 2.7 determines if the step size causes the photon to cross a boundary. If this is the case, the step is shortened to the boundary and the refraction and transmission are determined within 'CrossOrNot'. Otherwise the photon takes the complete step, loses weight in 'Drop', and the new direction is determined in 'Spin'. If there is a detector defined, the photon’s position and direction determine if a photon is detected after the 'Hop' in 'isDetected'.

The last step in propagation is making the decision if the photon is allowed to continue to the next 'HopDropSpin' cycle, or if the propagation is terminated. Termination occurs either if the photon is 'dead', or when the photon’s weight falls below the user defined critical weight and does not survive the 'Roulette'.
Fig. 2.7 – Activity graph for the photon propagation.
2.5.2 Tissue Representation

Tissue is represented as finite homogeneous layers where boundaries between layers are parallel to each other (Fig. 2.8(a)). Photon packets are launched into the top layer, and propagate through the layers until the photon is terminated. A photon packet only leaves the tissue either at the top or bottom boundary. When a photon crosses the side boundaries, the weight is reduced to 0, and the photon is terminated. The lost weight is not recorded in the output.

The simulation area boundaries are determined by the user, but the center of the simulation area is fixed at half the x and y size (Fig. 2.8(b)). For storing absorption, reflection, and transmission the area is divided into boxes, with a shifted center to remove negative indexes.

A fixed maximum of seven layers is used in the simulation. The first and last layer are implicit and only have a refractive index. These layers are not used for photon propagation, but only for launch, termination, and storage of reflection/transmission.

Boundaries are considered part of the tissue where the next step is taken. Any photon packet leaving a layer is always placed at a boundary, where the new direction is determined according to the boundary conditions (Section 2.2.7).

![2D layered tissue representation, including a photon trace](image)

![3D simulation area, including storage boxes](image)

Fig. 2.8 – Tissue representation

2.5.3 Classes

The classes used in the software and their relationship are depicted in Fig. 2.9. Functions for each class are listed in Tab. 2.1.
Fig. 2.9 – UML Class diagram for complete simulation and relationships, not showing attributes and operations.

<table>
<thead>
<tr>
<th>ClassName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CArea</td>
<td>3D &amp; 2D storage areas</td>
</tr>
<tr>
<td>CPhoton</td>
<td>Photon properties</td>
</tr>
<tr>
<td>CTissue</td>
<td>Base class for all tissue types</td>
</tr>
<tr>
<td>CSOURCE</td>
<td>Derived from Cphoton gives the photon its initial properties</td>
</tr>
<tr>
<td>CLayer</td>
<td>Representing actual layer in simulation</td>
</tr>
<tr>
<td>CDetectionFiber</td>
<td>Fiber to detect reflected photons</td>
</tr>
<tr>
<td>CSimulation</td>
<td>The simulation class containing all objects like photon, source and layers. This class has all the Monte Carlo functions</td>
</tr>
<tr>
<td>CSimInputParams</td>
<td>Storage of all input variables. Input is stored (and checked) in this class. The initialization of the simulation copies all data out of this class into the simulation classes</td>
</tr>
<tr>
<td>CRandomizer</td>
<td>Class containing the RNG</td>
</tr>
<tr>
<td>CDataFile</td>
<td>Class handling the CSV files</td>
</tr>
<tr>
<td>List</td>
<td>Template class; Linked List Used for temporary storage in memory and eventually writing the list to a file</td>
</tr>
<tr>
<td>ListNode</td>
<td>Template class; Node in Linked List</td>
</tr>
<tr>
<td>NDXVector</td>
<td>Double Vector class for positions and directions</td>
</tr>
<tr>
<td>NDXVector_float</td>
<td>Float Vector class for positions and directions for storage</td>
</tr>
</tbody>
</table>

Tab. 2.1 – Existing classes and their functions
2.6. Output

The MCOM software generates output files, containing the details of each simulation run. Three of these are always generated, and five other file formats are generated on request. All files are generated in binary format (Layout is found in Appendix A.2 and Chapter 2.3 of the MCOM GUIDE [14]). All of the files are generated in an output directory, containing all files for one simulation specified in the batch input file (see: Section 2.4.1).

Following sections briefly discuss the output files listed in Tab. 2.2.

<table>
<thead>
<tr>
<th>extension</th>
<th>Description</th>
<th>Created?</th>
</tr>
</thead>
<tbody>
<tr>
<td>.mcom.sim(.txt)</td>
<td>Constant simulation information</td>
<td>Always</td>
</tr>
<tr>
<td>.atr</td>
<td>Current Run Information</td>
<td>Always</td>
</tr>
<tr>
<td>.abs</td>
<td>3D absorption matrix</td>
<td>On Request</td>
</tr>
<tr>
<td>.ref</td>
<td>2D reflection matrix</td>
<td>On Request</td>
</tr>
<tr>
<td>.tra</td>
<td>2D transmission matrix</td>
<td>On Request</td>
</tr>
<tr>
<td>.pht</td>
<td>Detected photon history</td>
<td>On Request</td>
</tr>
<tr>
<td>.pha</td>
<td>Complete photon history</td>
<td>On Request</td>
</tr>
</tbody>
</table>

**Tab. 2.2 – List of all simulation output files.**

2.6.1 Constant Simulation Information

The files ending in `.mcom.sim` and `.mcom.sim.txt` contain variables passed into the software, that remain unchanged during simulation. Information is gathered straight from the input files, and files are created as one binary and one text file at the start of simulation.

2.6.2 Current Run Information

The absorption, transmission, and reflection file (`.atr`) is a standard file, storing variable information for each wavelength run. Data is stored in two sections. One section contains run information, and the other contains layer information.

2.6.3 3D absorption matrix

The 3D absorption matrix stores all the weight absorbed in the tissue. This is not done by storing each individual data point, resulting in a very large data file. Instead the absorption is stored into bins, using a resolution specified in the batch file. The storage coordinates \((x, y, z)\), are calculated from the Cartesian coordinates \((x_c, y_c, z_c)\), the resolution, \((X_b, Y_b, Z_b)\), and the dimensions of the area, \((X_s, Y_s, Z_s)\):

\[
\begin{align*}
x &= \left\lfloor \frac{X_b}{X_s} \left( x_c - \frac{X_s}{2} \right) + X_b \right\rfloor \\
y &= \left\lfloor \frac{Y_b}{Y_s} \left( y_c - \frac{Y_s}{2} \right) + Y_b \right\rfloor \\
z &= \left\lfloor \frac{Z_b}{Z_s} \left( z_c - Z_s \right) + Z_b \right\rfloor
\end{align*}
\]

(2.23)
Boundary conditions are solved by:

\[ x = x - 1 \quad \text{if} \quad x = X_b \]
\[ y = y - 1 \quad \text{if} \quad y = Y_b \]
\[ z = z - 1 \quad \text{if} \quad z = Z_b \]  

(2.24)

For collecting and storing this data, the three dimensional array is converted to a single dimension, where the new array index is retrieved using the following formula:

\[ i = x + (y \times X_b) + (z \times X_b \times Y_b) \]  

(2.25)

### 2.6.4 2D reflection matrix and 2D transmission matrix

The 2D reflection matrix, and 2D transmission matrix store the reflection, and transmission leaving the simulation area. Data is stored in bins in a two dimensional array, using the same resolution used for the 3D absorption matrix. The new coordinates are calculated using Equation 2.23, Equation 2.24. And similar to the absorption matrix, the two dimensional array is converted to a one dimensional array using the following formula:

\[ i = x + (y \times X_b) \]  

(2.26)

### 2.6.5 Photon Traces

Simulated photon trajectories are collected and stored for all detected photons (if a detection fiber is used), and/or all photons launched in the simulation area. The method of storing is the same for both files.

During execution of the code, each interaction point is stored in a linked list. The data stored are the coordinates, and the weight of the packet. The only relation between individual data points is that they are stored sequentially.

### 2.7. Complexity

Measuring complexity of software is done by multiple techniques. The MCOM software is analyzed on three levels:

- run time complexity using computational complexity and profiling;
- memory requirement complexity;
- code complexity using cyclomatic complexity of files and classes.

#### 2.7.1 Run Time Complexity

Determining the run time of the MCOM software is important for deciding where to optimize the current implementation. Using properties of the MCOM simulation, the model is analyzed for run time complexity.

Analysis focuses on simulating a single wavelength, broken down into: initialization, photon launch,
Algorithm 2.1 Single Wavelength Simulation

1: Initialize Simulation
2: while Photons Available do { N photons are launched }
3:   Launch Photon
4:   while Photon Alive do { M photon propagation steps }
5:     Photon Propagation Step
6:   end while
7: end while
8: Terminate Simulation

 photon propagation, and termination. This structure is given in Algorithm 2.1.

The complete run time of this algorithm is given by $T_r$, defined by Equation 2.27

$$T_r = T_1 + N \times T_2 + N \times T_3 + M \times N \times T_4 + M \times N \times T_5 + T_8$$  \hspace{1cm} (2.27)

, where $T_n$ stands for the running time of the $n^{th}$ line of code, $N$ is the number of photons launched, and $M$ the number of steps a photon takes after it is launched in the tissue.

Within photon propagation, no loops are defined. This means that the complexity of the photon propagation is $O(\max(T_5))$. For the current implementation $T_5 = O(1)$. The complete complexity of the system is hence defined as $O(NM)$.

The number of steps taken by a photon ($M$) depends on the optical properties ($\mu_a$, $\mu_s$, $g$, $n$), and the dimensions of the simulation area. In order to compute the complexity, the minimum ($M_{\text{min}}$), maximum ($M_{\text{max}}$) and average ($M_{\text{ave}}$) complexity is analyzed.

There is one situation that is singled out from determining run time complexity. When $\mu_a = 0$ and $\mu_s = 0$, the step size is infinitely large, and no absorption occurs. This means that the photon only takes one step to the next boundary, where it leaves the simulation area (depending on the angle of incidence). This case is omitted in the analysis.

The minimum number of steps taken is trivial to determine. It mainly depends on the anisotropy of the tissue, since this determines the photon direction. From entry in the tissue, the shortest path to the reachable boundary is taken, divided by the step size in the tissue to determine the minimum number of steps. This leads to two steps in the most optimal case (one step in tissue, one step out).

Finding the maximum number or steps is done by simplifying the problem by choosing the simulation area of infinite dimensions and having only one tissue type. The number of steps taken by a photon is related to its weight. At each step in the propagation the photon loses weight, until the weight of the photon is too small for the photon to continue and it dies. The weight loss rate is determined by Equation 2.5 and Equation 2.6. The minimum critical weight, $W_c$ for a photon to continue propagation is determined by the user. If the weight falls below this value, the Roulette function determines if the photon continues its path. Algorithm 2.2, for finding the maximum number of steps, disregards the Roulette function. During Roulette (Section 2.2.6), there is a small probability that weight is added to photons, increasing the number of steps taken by a photon. Since this is only a small fraction, it is assumed that the photon dies when the weight falls below $W_c$.

This algorithm shows that when the tissue has no absorption or scattering, the simulation never
Algorithm 2.2 Worst Case Photon Step Count

1: \( n \leftarrow 0 \)
2: \textbf{if} \( \mu_a = 0.0 \land \mu_t = 0.0 \) \textbf{then}
3: \( n \leftarrow 1 \)
4: \textbf{else}
5: \textbf{while} \( \text{Photon Weight at step } n, \text{ larger than } W_c \) \textbf{do}
6: \( W_{n+1} \leftarrow W_n - W_n \times \frac{\mu_a}{\mu_t} \)
7: \( n \leftarrow n + 1 \)
8: \textbf{end while}
9: \textbf{end if}
10: \textbf{return} \( n \)

terminates, and an infinite number of steps is taken. In the simulation this is solved by choosing a step size large enough to position the photon at the next tissue boundary. When \( \mu_a \) approaches 0 the number of steps needed to lose all weight increases, since \( \mu_t \) decreases and the amount of weight lost at each step approaches 0 (assuming a fixed \( \mu_s \)).

Calculation of the average number of steps taken requires an approximation of the mean path length. This is the mean length of a photon trajectory from launch until termination. In a finite simulation area, this is the point where the photon leaves the simulation area. The determination of this path depends on the optical properties of the tissue, the light source, and the geometry of the tissue sample. Migration of a photon may be described using radiative transfer theory, but leads to a complicated equation, too analytical for practical use. In most cases an approximation is made using either the diffusion approximation [15], or a Monte Carlo implementation. Another method calculating the path length of a photon trajectory is the path integral formalism [16][17][18]). This method does not find the most likely paths, but tries to identify them directly. The implementation and analysis of these methods is not required for this project, but is useful in the analysis of run time. For determining the mean path length a number of MCOM simulations is executed.

Once the mean path length is found, the number of steps taken is deduced by dividing the mean path length by the step size through tissue. For multilayer tissues, with changing optical properties, the number of steps is determined by finding the mean path length for each layer, using the predicted trajectory, and the layer’s step size to find the number of steps. The sum of all steps from all layers is taken as the average number of steps.

The maximum of \( N \) is defined by the user. The minimum is either the maximum set by the user, or the number of photons needed to reach the user defined standard deviation for detected photons.

2.7.2 Memory Complexity

The MCOM software memory requirements consist of a default, and an optional part. Memory requirement is only analyzed for one wavelength, since at any one time only one wavelength is kept in memory.

The default memory requirement consists of two parts:
1. The configuration files, including the simulation tissue. This part is negligible, especially since the tissue model is not complicated.

2. The storage area for storage of absorption, reflection and transmission.

The latter depends on the accuracy requested by the user for division of the simulation area into bins. The complete description of the storage area is found in Section 2.6. The actual storage is made up of three parts, reflection $S_r$, absorption $S_a$, transmission $S_t$. So the minimal memory $S_{\text{min}}$ required is:

$$S_{\text{min}} = S_r + S_a + S_t$$  \hfill (2.28)

The complete storage environment is created in memory when the simulation starts. Storage is calculated in terms of the number $X,Y,Z$ bins $(X_b, Y_b, Z_b)$, and the size of the storage type, $D$. The new equation becomes:

$$S_{\text{min}} = X_b Y_b D + X_b Y_b Z_b D + X_b Y_b D = X_b Y_b D(Z_b + 2)$$  \hfill (2.29)

Optional storage consists of the complete history of steps taken by each photon. The maximum memory requirement occurs when both the complete history and detected history are required. These two data sets are recorded separately and require $4 \times D$ per step of storage for the position and weight. The precise amount of memory required depends on the number of photons launched per wavelength, $N_p$, and the number of steps per photon $N_s$. This memory is not preallocated but grows during simulation. The complete maximum memory requirement, $S_{\text{max}}$ is defined by

$$S_{\text{max}} = S_{\text{min}} + 4D \max(n_p) \max(n_s) = D(X_b Y_b(Z_b + 2) + 4(\max(n_p n_s)))$$  \hfill (2.30)

An example of storage requirements for a single wavelength with resolution $(X_b, Y_b, Z_b) = (200, 200, 200)$, $N_p = 5 \times 10^7$, $N_s = 150$, and $D = 8$ bytes, where only the detected photons are recorded (taken here as 0.10% of the launched photons), has a total memory requirement of 290MB. When all photons are tracked, the memory requirement becomes 223GB.

### 2.7.3 Cyclomatic Complexity

Complexity measurement for software is done using McCabe’s cyclomatic complexity [19]. This method involves measurement of linearly independent paths through a program’s source code. The complexity of the code is determined and assigned in one of four categories from Tab. 2.3.

The most complex methods are shown in Tab. 2.4. When refactoring the software, these methods are the first candidates for reducing overall complexity.

### 2.8. Data Processing

All processing of output files is done using Matlab. Some specific libraries are created to read the output files, and analyze the results. Visualization is also done using Matlab. Some of the images retrieved from the Matlab applications are shown in Fig. 2.10.
## Analysis

### Cyclomatic Complexity

<table>
<thead>
<tr>
<th>Cyclomatic Complexity</th>
<th>Complexity/Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 10</td>
<td>simple code, not much risk</td>
</tr>
<tr>
<td>11 - 20</td>
<td>more complex, moderate risk</td>
</tr>
<tr>
<td>21 - 50</td>
<td>complex, high risk</td>
</tr>
<tr>
<td>&gt; 50</td>
<td>very complex, very high risk</td>
</tr>
</tbody>
</table>

*Tab. 2.3 – Cyclomatic complexity and risk evaluation table [20]*

### Method Cyclomatic Complexity >= 10

<table>
<thead>
<tr>
<th>Class :: Method</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSimInputParams::ReadSimParamsFromBatch()</td>
<td>130</td>
</tr>
<tr>
<td>CSimulation::Start_MC_Sim()</td>
<td>36</td>
</tr>
<tr>
<td>CSimulation::Hop()</td>
<td>18</td>
</tr>
<tr>
<td>CSimulation::WriteConstSimParams_AS_TXT()</td>
<td>14</td>
</tr>
<tr>
<td>CRandomizer::ran3()</td>
<td>12</td>
</tr>
<tr>
<td>CSimulation::WriteConstSimParams</td>
<td>12</td>
</tr>
</tbody>
</table>

*Tab. 2.4 – Method Cyclomatic complexity >= 10*
2 Analysis

(a) compressed total absorption histogram  (b) 3D photon traces

(c) Transmission spectra of multiple wavelengths

**Fig. 2.10 – Example Matlab data analysis output**
2.9. Issues & Improvements

Analysis of the software turned up issues that are the result of rewriting and using logic from other, not object oriented, implementations. The best improvement to this software is a complete redesign from scratch, implementing only the logic from other implementations. This is not within the scope of this project, but is a recommendation for future developers. This section deals with some of the encountered issues, and changes made to correct them.

2.9.1 Standard Libraries

Reusing software made by others is a common practice in software engineering. This allows the developer to profit from others who have solved common programming problems, and created freely available standardized libraries. Two of the most well known standardized libraries for C++ are the Standard Template Library (STL) [21] and Boost [22]. These libraries add functionality to the software, preventing common programming errors, and reduce development time. The following list discusses some changes implemented using standard libraries.

- The storage list for tracked photons is changed from generic linked list implementation to a STL vector. There is no need for a linked list, and the generic implementation caused overhead in the software. A comparison of the run time before the implementation of the vector, and after the implementation shows a decrease in list run time of 15% for list actions (Tab. 2.5).

<table>
<thead>
<tr>
<th>List Actions</th>
<th>HopDropSpin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListT</td>
<td>25%</td>
</tr>
<tr>
<td>Vector</td>
<td>10%</td>
</tr>
</tbody>
</table>

Tab. 2.5 – Performance of list actions, comparing old ListT implementation versus new Vector implementation.

- The storage for absorption, reflection, and transmission during simulation is changed to a vector, implementing more efficiently the dynamic storage. It also allows creation of a multidimensional array for storage, when required.

- In order to decrease the risk of leaking memory, shared pointers and shared pointer lists are introduced from the Boost library. These template types take care of automatically destructing objects when no references exist to the object. This simplifies the memory model for the software, and prevents memory leaks. The previous implementation has memory leaks in its implementation, causing the risk of running out of memory or causing a simulation crash.

2.9.2 Configuration Files & Variables

There is little input validation of the configuration files, enhancing the risk of incorrect simulation output. All possible variables and input parameters are identified, and are given their logical limits. This is added to the list of variables in Appendix A.1.

As shown in Section 2.7.3 the SimInputParams class is too complex. Keeping in mind the enhancements to come, this class is discarded and a new method of reading in variables from the configuration
files is created. A more standardized method of configuring (such as XML) is preferred, but not implemented to keep the configuration of the software similar to the current implementation.

Each class requiring a configuration file is responsible for validation of its own variables. This decentralized method allows simplified addition of new classes using configuration files without changing the central method of reading in variables. Reading in configuration files is further explained in Section 4.1.3.

2.9.3 Reduction of Complexity

Reducing complexity is done by rearranging logic, and moving components to other files. The ‘CSimulation::Start_MC_Sim’ method is removed, and most of its content is moved to the main program (method ‘main’). The reduction in complexity is shown in Tab. 2.6, where there is no method left with a complexity greater than 14.

<table>
<thead>
<tr>
<th>Class::Method</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSource::createFromConfig</td>
<td>14</td>
</tr>
<tr>
<td>CSimulation::HopDropSpin</td>
<td>14</td>
</tr>
<tr>
<td>MismatchRefraction</td>
<td>11</td>
</tr>
<tr>
<td>CLayeredTissue::setNextLambda</td>
<td>10</td>
</tr>
</tbody>
</table>

**Tab. 2.6** – *Method Cyclomatic complexity >= 10, after implemented improvements.*

Redesign of Photon Propagation

The current implementation of photon propagation (Fig. 2.7), shows multiple entry points for the states ‘CrossOrNot’, ‘Hop’, and ’isDetected’. Simplification of photon propagation is done by combining paths, and reducing the entry points of the nodes to one. This is shown in Fig. 2.11. Also the detection of a photon is only done when its positioned at a boundary, but ’isDetected’ is also accessed when a photon is not at a boundary. The new photon propagation corrects this, reducing calls to ’isDetected’.

2.9.4 Object Oriented Design

Implementing the MCOM software in C++ was done to introduce a more object oriented approach. This goal is not reached since most of the software is taken literally from previous implementations. The sources, detectors and tissue are corrected in an object oriented manner, to make verification, adaptation, and extension of the software a more simple task.

Sources

The current implementation has the CSource class as a subclass of the CPhoton class. The different types of sources are all implemented in a large ‘switch’ statement in the CSimulation method launchPhoton. Implementing the source more object oriented is done with one abstract class CSource, and eleven subclasses that extend CSource. The abstract class CSource now defines a method launch photon, while this was first a method of CSimulation. The abstract methods setLocation and setDirection are overridden by each subclass.
Detectors

As with the sources, the definition of detectors and their implementation is not gathered under a single class, but implemented in different methods of CSimulation. Determining if a photon is detected is incorrectly based on the source type instead of a detector type. The new CDetection abstract class is similar to the CSource class mentioned above, and allows easy extension using subclasses. Each of these detector types is implemented in its separate class, extending the abstract base class.

Fig. 2.13 shows the CDetection class and its two subclasses. The abstract method inNA determines if a photon is detected by the detector.

Tissue

The current CTissue class is not used in simulation. The actual depiction of tissue is done using a fixed array of CLayer objects (see Fig. 2.9). This is changed to a situation where the CTissue class is used as an abstract class, extended by the CLayeredTissue class (see Fig. 2.14). This allows the implementation of new tissue types without changing the rest of the simulation.
2 Analysis

J.A.J. van Belkum

Fig. 2.12 – UML class representation of CSource and subclasses. No attributes and operations are shown. CSource shows two operations.

Fig. 2.13 – UML class representation of CDetection and subclasses. No attributes and operations are shown. CDetection shows the abstract inNA method.

Fig. 2.14 – CTissue UML class diagram, and subclasses. The SLambda is a structure containing the optical properties for the current layer and wavelength.
2.9.5 Conservation of Energy

During simulation energy, depicted by weight, is transferred from the photon packets to the surrounding tissue (absorption), and passed on to the surrounding environment (transmission, reflection) (Fig. 2.15).

The law of conservation of energy states that the amount of energy in a closed system remains constant. This implies for the simulation

\[
W_T = W_{abs} + W_{ref} + W_{tra}
\]

Fig. 2.15 – Conservation of energy depicted as sum of three parts.

, where \(W_T\) is the total weight of the photon packets, and \(W_{abs}, W_{ref}, W_{tra}\) are the weights of absorption, reflection, and transmission.

In the MCOM software this is not the case. Due to weight being dropped at the side boundaries, and the loss/addition of weight during Roulette (Section 2.2.6), the recorded weight always differs from the total weight put into the system.

If desired, the weight lost and added during simulation may be stored. This is not implemented, since this is a known and accepted side effect of the Monte Carlo method.

2.9.6 RNG

A new solution for the current ran3 is studied, to allow enhancements of the software, and implement a standard RNG solution. An algorithm with a sufficiently large period \((2^{19937} - 1)\), and a good reputation is the Mersenne Twister algorithm (MT19937) [23]. This RNG is well known for its quality and speed. The SIMD-oriented Fast Mersenne Twister (SFMT19937) [24], is an improved version of MT19937, implementing more hardware aware improvements. Both are implemented in standard libraries, that simplify integration in the current code, and reduce the chance of introducing bugs.

The current RNG (ran3) is compared with two implementations of the MT19937 algorithm, and one implementation of the SFMT19937 algorithm.
The first MT19937 and the SFMT19937 algorithms are implemented by Charles Karney [25]. The second MT19937 implementation is from the Boost libraries. Generation of \(10^8\) Reals in the interval \([0,1)\) resulted in the timings listed in Tab. 2.7.

<table>
<thead>
<tr>
<th>implementation</th>
<th>execution time (sec)</th>
<th>% Karney SFMT</th>
<th>% Karney MT</th>
<th>% Boost MT</th>
<th>% ran3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karney SFMT</td>
<td>6.51</td>
<td>100.00</td>
<td>113.21</td>
<td>312.29</td>
<td>377.88</td>
</tr>
<tr>
<td>Karney MT</td>
<td>7.37</td>
<td>88.33</td>
<td>100.00</td>
<td>275.85</td>
<td>333.79</td>
</tr>
<tr>
<td>Boost MT</td>
<td>20.33</td>
<td>32.02</td>
<td>36.25</td>
<td>100.00</td>
<td>121.00</td>
</tr>
<tr>
<td>ran3</td>
<td>24.60</td>
<td>26.46</td>
<td>29.96</td>
<td>82.64</td>
<td>100.00</td>
</tr>
</tbody>
</table>

Tab. 2.7 – RNG comparisons for MT19937, SFMT19937 and RAN3.

The SFMT19937 Karney implementation is by far the fastest implementation of the MT19937 algorithm, being almost four times as fast as ran3 and three times faster than the Boost implementation. It also outperforms the Karney MT19937 implementation by 13 percent.

Furthermore the Karney library has an extensive set of methods, providing interfaces for ease of parallelization, and random selection. The Karney SFMT19937 implementation is therefore chosen as the new RNG for the MCOM software.

For each wavelength run in the current implementation, the RNG is restarted using a new random seed, but if a fixed random seed is given in the configuration, this seed is used for all wavelength runs. This implies that each wavelength simulation uses the same sequence of numbers all over again. The problem concerning the RNG is discussed further, along with solutions, in the parallel implementation Section 4.2.2.

2.9.7 Output Files

The current method of output relies on a generic format created solely for the MCOM software. This creates several problems:

**Portability**: Since a non standard format for output is chosen, each method of analysis is forced to write a specialized parser to retrieve the data. This can cause inconsistencies or incorrect reading of data. Also the reading of data is architecture dependent and requires addressing in data creation.

**Maintainability**: Every change in the software, altering the output of the simulation, forces the adaptation of the software that analyzes the results.

**Preserving data**: Results obtained with old versions loose their meaning and make parsing difficult when software and data format evolve, but the old data remains unchanged.

The best method to solve this problem is to write all output data in a standardized format that has the three properties mentioned above. Examples of these are HDF5 [26], and CDF [27]. The research and implementation of such a format is not within the scope of this project but is left as a recommendation for further development.
Some issues are addressed now to provide a better interface to the output. These issues are discussed per output file.

**2D and 3D storage matrices**

The division of the simulation area into storage boxes is not done according to a specified method. Unclear defined boundary conditions lead to inaccuracies in the output files. The implementation is corrected such that all boundary conditions are treated similar.

**Photon History files**

The photon history files store information in blocks of four floats, specifying the location and weight of a photon at a certain point in time. The relationship between data points is not available, resulting in a loss of information. In order to have more knowledge about the path of each photon packet, the sequence number of each photon is recorded. Since the individual photons are recorded sequentially, there is no need to also record the sequence number for each individual step.

**2.9.8 Documentation**

Documentation of software requires clarity and structure for quick comprehension by new developers. To achieve this, code commenting is changed to allow generation of documentation from the source code. The documenting package Doxygen [28] provides a method of creating nicely formatted source documentation. Existing code comments are adjusted to fit the new documentation framework, and new code comments are implemented using this format.

**2.9.9 Visualization**

Visualizing the data created by the MCOM software is important in understanding the simulation process, and the results produced. Currently visualization is done in Matlab by custom created scripts. This method of visualization is not suitable for detailed analysis of large data sets, since the memory requirements greatly exceed the available resources. Visualization is now done using reduced data sets, where detail is lost. A different method to analyze and visualize the results also involves the adaptation of the output created by the software. Using standardized methods of storing data provides a starting point for visualization, since many tools are capable of reading these standard data formats.

To make a start with visualization of the data produced, the Visualization Toolkit (VTK)[29] provides a good starting point for implementing better visualization. A tool is created to convert existing "*.pht" files into VTK format. The resulting files may be viewed using applications supporting the VTK format (such as Paraview [30]). An example is shown in Fig. 2.16. Most of these tools allow rendering of data on more than one computer using a grid, increasing detail of visualization.
Fig. 2.16 – Single tilted input fiber, with one detection fiber. Color indicates weight of photon packet, where red is the highest and blue is the lowest weight. Each detected photon packet trajectory is shown.
Chapter 3

Verification

Creating confidence in the existing code is the single prime pre-condition for making changes to the current implementation. To get reliable simulation results, high confidence levels are required. This confidence is attained through the process of code verification. Verification is done using two methods: Unit testing, and Acceptance testing. Unit testing involves verification of each logical section of the code (the classes and methods), and in particular the code sections involving MCOM logic. Acceptance testing focuses on execution of simulations and verification of the simulation results with theory, literature and experiments.

3.1. Unit Tests

During development, and improvement of the code, each newly created section requires correct implementation. To verify this, the complete software package can be build and executed. If there are any errors, they might appear during execution, but this is not guaranteed. A better method is to subject the newly build program to acceptance tests (see Section 3.2). This method reveals more errors in the software, since they are designed to test if the code is behaving as designed. Acceptance testing is not meant to catch all possible errors, but only to verify functionality.

Instead of rerunning all of the acceptance tests after each change in the code, unit tests are used to verify the workings of the newly added code sections. These test cases are sections of code, that test each possible outcome of a small section of implemented logic. All of the unit tests combined, create a framework that verifies the correct implementation of the MCOM logic. This style of development is called 'Test Driven Development' [31], where tests are leading in development.

Unit tests are not only useful for verifying new code, but are also used to test legacy code [32]. This is an important step in verifying the MCOM software. The previously implemented MCOM logic has no unit testing. Unit tests are created to verify existing logic. Problems are detected early in development using this method, and are corrected before continuing with enhancement creation.

Unit tests are implemented using the CxxTest framework [33], a C++ test framework consisting of only header files and a Perl script. For each implemented class or functionality, a C++ header file is created to implement the tests that verify the correct functioning. Using the provided Perl script a test
program file is generated. This file is then compiled into a program that is executed. All created tests are compiled and executed during a test run, and the result of the tests is printed to the screen.

### 3.1.1 Implementation Cycle

For newly implemented features in the code, unit tests are created and executed at the beginning of the development cycle. The most important and crucial parts of the existing code base are also tested using unit tests, but this is only done while adapting them for the required changes. The actual verification of physical properties of the code is performed by the Philips Research C&HA group members during implementation of new features.

The following steps are taken in software development using unit tests:

1. create a new test case,
2. run the test suite to see the test fail,
3. create the code,
4. run the test suite to see the test pass,
5. refactor code if needed, and verify using the test suite.

This method of testing forces the developer to thoroughly research the problem before implementing changes. Creating tests upfront that validate logic saves time during integration, since it is verified that the implemented components are correct. Errors found during integration are either a result of integration issues, or incorrect logic.

When all created tests pass for the complete code, each of the created code sections is trusted, and implemented correctly. At this point the code is ready to undergo Acceptance Tests, verifying correct implementation, and integration.

### 3.2. Acceptance Tests

Acceptance tests are performed to check if the functional requirements are correctly implemented in the code. All of the mentioned tests are simulated by launching 50000 photon packets into a homogeneous turbid medium, having optical properties defined for the test set. The simulation is executed 10 times using a different random seed each time to create different result sets. From each simulation the reflection, transmission and absorption are recorded, and the mean is calculated to obtain an accurate result. Also the standard error is determined in order to determine how accurate individual simulation results are.

Initial tests are executed using the simulation software, after the suggested changes from Chapter 2 are implemented. Whenever changes to the software are made, or when new features are implemented, these tests are executed to verify if the code still functions as designed.

The following sections contain the methods of the acceptance tests, their results after execution, and analysis of the results. In Appendix B.2 a tabular overview of all acceptance tests is given, including the labels used for each of the tests.
3.2.1 Optical Properties Tests

Methods

The first set of tests verify if the MCOM software correctly implements the optical properties of the model by executing four simulations. In these simulations a single optical property is varied, and the other two are kept constant. The properties examined are: absorption, scattering, and anisotropy. Below four tests are split in two sections, absorption and scattering verification.

- The first test is designed to verify the correct working of absorption in tissue. Scattering and anisotropy are set to 0 to make sure that no reflection is observed, but only transmission and absorption. A law describing the relationship between light absorption in an absorbing medium, \( \mu_a \), and sample thickness is known as Lambert-Beer:

\[
\frac{dI}{I} = \mu_a dl
\]  

(3.1)

In this equation \( I \) is the intensity falling on a medium, \( \frac{dI}{I} \) is a change in light intensity through a layer of thickness \( dl \). For an initial intensity \( I_0 \) the remaining intensity after traversing a medium of thickness \( l \) is

\[
I = I_0 e^{-\mu_a l}
\]  

(3.2)

This formula is used to predict the remaining intensity, the transmission, for the simulation. Conservation of energy is mandatory for all simulations, meaning that the recorded amount of reflection, absorption, and transmission add up to one. Since for this test the reflection measured is 0, absorption are complementary to the results for transmission.

This test is executed using the following optical properties: \( \mu_s = 0 \text{ cm}^{-1}, g = 0 \text{, and } \mu_a = [0 : 20 : 200] \text{ cm}^{-1} \) (absorption is increased in steps of 20, starting from 0, up to 200).

**Hypothesis 1 (Absorption verification)**

*Simulation of transmission and absorption in a non-scattering medium should match calculations using the Lambert-Beer equation, and the sum of transmission, reflection, and absorption should add up to 1.*

- Tests two to four are designed to observe the influence of \( \mu_s \) and \( g \), without the influence of \( \mu_a \). These simulations all test \( \mu_s \) in 10 steps from 0 to 200 cm\(^{-1}\), but where in each simulation the \( g \) is either, 0, 0.5, 0.9. Unlike absorption, the ratio of reflection versus transmission is more difficult to predict in a scattering medium, due to the influence of anisotropy. After each scattering event, the photon has an unpredictable direction, that makes the determination of transmission and reflection difficult.

Verification using Prahl’s adding doubling method [34] (explained further in Section 3.2.2) is included in the graphs to give an indication of the quality of the software.

During each of these tests the thickness, chosen to show the most clear effect, of the layers is set to 0.02 cm. The refractive index \( n \) is set equal to the medium surrounding the simulated tissue. In this case it is 1.00, that matches the refractive index of air.
Results

- Fig. 3.1 shows the results from simulation, and the line created using the Lambert-Beer equation: Equation 3.2. The first thing to notice is that no reflection is measured, concurring with a \( \mu_s \) of 0.

The figure further shows that the transmission collected from simulation matches the Lambert-Beer line. The sum of transmission, reflection, and absorption add up to 1. The maximum standard error for transmission is 0.0007.

![Simulation results with Lambert-Beer line](image)

**Fig. 3.1 – Photon absorption verification: step \( \mu_a \) with: \( \mu_s = 0 \), \( n = 1 \), and \( g = 0 \)**

- Fig. 3.2 shows the results for the tests with varying \( \mu_s \) and three different \( g \) values. For all three graphs the sum of transmission, absorption, and reflection is 1, concurring with conservation of energy. All three graphs also show that the absorption is 0, as expected with a \( \mu_s \) of 0. The influence of the increase of \( g \) is visible with an increase in the transmission measured. This is expected with an increasing \( g \); the scattering action is more forward directed. The maximum standard error for transmission and reflection in all data sets: 0.0010. The verification data, calculated using Prahl’s adding doubling method, matches the results obtained from simulation closely.

Conclusion

Testing the extreme conditions of \( \mu_a \) and \( \mu_s \) is the first step in the verification process. These simulations should always give correct results, without deviation in order to continue testing. Since the performed simulations give the expected results, the code is good enough to continue testing using literature tests.
3.2.2 Literature Tests

Methods

Comparing the results obtained during simulation against known data from literature give a good indication of simulation correctness. There are two sources of data for comparison. Their methods and results are discussed below.

1. Validation of simulation results for the original MCOM software was done by Wang and Jacques by comparing transmission and reflection obtained against other simulation and calculation results [1]. Two tests are taken from their article and simulated using the MCOM software.

Hypothesis 2 (Wang & Jaques verification)

*Average Reflection and Average Transmission results from simulation should match results listed in Wang & Jacques [1] section 6.1 up to three decimal places, with comparable standard error.*

Three decimal places of precision is defined as the level of accuracy accurate for use within.
2. The following 30 tests are taken from a book by Prahl [34]. In this book Prahl describes the adding-doubling method for finding accurate estimates of light distributions in any biological tissue. The goal of this method is to provide a golden standard whereby other models and simulations can be verified.

From the Prahl book, two types of tests are taken to verify the MCOM software. These tests are chosen from the available tests to verify the correct implementation of the boundary conditions under influence of the refractive index. The parameters for the tests are:

**Matched boundary** Setting $n = 1$, gives the single layered tissue the same refractive index as the surrounding area. This allows accurate measurement of transmission and refraction without being influenced by refraction.

**Unmatched boundary** Sandwiching the medium between two glass layers, that have a different refractive index than the medium. This allows measurements of transmission and refraction for photon paths under the influence of refraction at tissue boundaries.

Each of these two parts is split in three sections where $g$ is either 0, 0.5, or 0.875. Within these different sections, simulations are done through five different thicknesses: 0.01, 0.02, 0.04, 0.08, and 0.16 cm. For all tests $\mu_a$ is fixed at 10, and $\mu_s$ is fixed at 90. These values are taken from calculated tables in the Prahl book [34]. The chosen thicknesses are most representative of thicknesses used in the MC simulations within Philips Research C&HA, and have the most variation in calculation results for reflection and transmission.

**Hypothesis 3 (Prahl Adding-Doubling verification)**

*Average reflection and average transmission results obtained from simulation, using parameters mentioned in Prahl[34], should match the calculated results within 97% accuracy.*

The expected minimal accuracy of the Adding-Doubling method for determining transmission and reflection is three percent as mentioned by Prahl [34], forcing the simulation to be 97% accurate.

**Results**

1. The results from the Wang and Jacques test are summed up in two tables in Tab. 3.1, together with the results obtained from simulation. The simulated results in both tables match the other data sets within three decimal places, taking in account the standard error measured. For the second test, only the average reflection is recorded, since the thickness is semi-infinite to exclude transmission. These results show that the current version is equally accurate when compared to other implementations.

2. The Prahl comparison simulation produces five result sets. Each of these test sets is compared to the data calculated by Prahl [34].

Tab. 3.2 shows the simulation results, and the values from the Prahl document. The standard error for $T_s$ and $R_s$ is very small, indicating that each simulation is accurate in producing similar
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>van de Hulst, 1980</td>
<td>0.09739</td>
<td></td>
<td>0.66096</td>
<td></td>
</tr>
<tr>
<td>Prahl et al., 1989</td>
<td>0.09711</td>
<td>0.00033</td>
<td>0.66159</td>
<td>0.00049</td>
</tr>
<tr>
<td>Wang et al., 1995</td>
<td>0.09734</td>
<td>0.00035</td>
<td>0.66096</td>
<td>0.00020</td>
</tr>
<tr>
<td>Test L1, 2008</td>
<td>0.09742</td>
<td>0.00021</td>
<td>0.66140</td>
<td>0.00023</td>
</tr>
</tbody>
</table>

(a) thickness = 0.02 cm, $\mu_a = 10$, $\mu_s = 90$, $g = 0.75$, $n = 1$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Giovanelli, 1955</td>
<td>0.2600</td>
<td></td>
</tr>
<tr>
<td>Prahl et al., 1989</td>
<td>0.26079</td>
<td>0.00079</td>
</tr>
<tr>
<td>Wang et al., 1995</td>
<td>0.25907</td>
<td>0.00170</td>
</tr>
<tr>
<td>Test L2, 2008</td>
<td>0.26069</td>
<td>0.00059</td>
</tr>
</tbody>
</table>

(b) thickness = $\infty$, $\mu_a = 10$, $\mu_s = 90$, $g = 0$, $n = 1.5$

Tab. 3.1 – Wang & Jacques Literature Tests.

results. Both for transmission and reflection, the difference between the calculated data, and the simulation results is less than the maximum 3% (last two columns).

**Conclusion**

The MCOM software produces accurate results when comparing transmission and reflection to results obtained from other implementations of MC photon transport software as seen in the Wang & Jacques tests. The software also performs as expected when comparing the transmission and reflection to calculated data using the adding-doubling method described by Prahl. This positive result from the Prahl test also verifies the correct functioning of the scattering optical properties test from Section 3.2.1.

Overall the verification results in this section give good confidence in the correct implementation of MCOM. The last step is now to compare simulation results with data obtained from phantom experiments.

### 3.2.3 Phantom Tests

**Methods**

Within Philips Research C&HA phantom tests were executed to test the previous version of the MCOM software. The results from these tests are still available, and are used to compare against results obtained from the current implementation. Three phantom tests are executed and compared to the existing data sets from experiments.

For comparison of simulation against experiments using phantom tissues, the optical properties of the phantom tissues need to be known. The phantom tissues used in these experiments are created by Philips Research C&HA, using the methods described in the technical note from Uzunbajakava [35]. With an integrated sphere setup, transmission was measured for each different phantom tissue used. The values obtained were used to find the $\mu_a$ and $\mu_s$ for the NIR spectra used in the experiments. For test 1 and 2, the NIR range determined is from 1000nm to 2000nm (in steps of 1nm), and for test 3 it is from 1100nm to 1800nm (in steps of 1nm). The refractive index was also determined using an
<table>
<thead>
<tr>
<th>Test</th>
<th>$T_s$</th>
<th>$T_t$ err.</th>
<th>$R_s$</th>
<th>$R_t$ err.</th>
<th>$T_p$</th>
<th>$R_p$</th>
<th>$T_d$</th>
<th>$R_d$</th>
<th>$%$ $T_d/T_p$</th>
<th>$%$ $R_d/R_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.3.0</td>
<td>0.5910</td>
<td>0.0006</td>
<td>0.2674</td>
<td>0.0004</td>
<td>0.5916</td>
<td>0.2674</td>
<td>0.0006</td>
<td>0.0000</td>
<td>0.1690</td>
<td>0.0000</td>
</tr>
<tr>
<td>L.3.1</td>
<td>0.3546</td>
<td>0.0004</td>
<td>0.3634</td>
<td>0.0004</td>
<td>0.3655</td>
<td>0.3616</td>
<td>0.0019</td>
<td>0.0017</td>
<td>0.5610</td>
<td>0.5531</td>
</tr>
<tr>
<td>L.3.2</td>
<td>0.1267</td>
<td>0.0002</td>
<td>0.4097</td>
<td>0.0005</td>
<td>0.1285</td>
<td>0.4081</td>
<td>0.0018</td>
<td>0.0016</td>
<td>1.5564</td>
<td>0.4901</td>
</tr>
<tr>
<td>L.3.3</td>
<td>0.0159</td>
<td>0.0001</td>
<td>0.4166</td>
<td>0.0004</td>
<td>0.0160</td>
<td>0.4148</td>
<td>0.0001</td>
<td>0.0018</td>
<td>0.0000</td>
<td>0.4822</td>
</tr>
<tr>
<td>L.3.4</td>
<td>0.0003</td>
<td>0.0000</td>
<td>0.4156</td>
<td>0.0004</td>
<td>0.0002</td>
<td>0.4149</td>
<td>0.0001</td>
<td>0.0007</td>
<td>0.0000</td>
<td>0.2410</td>
</tr>
<tr>
<td>L.4.0</td>
<td>0.7371</td>
<td>0.0005</td>
<td>0.1310</td>
<td>0.0004</td>
<td>0.7391</td>
<td>0.1298</td>
<td>0.0020</td>
<td>0.0012</td>
<td>0.2706</td>
<td>0.7704</td>
</tr>
<tr>
<td>L.4.1</td>
<td>0.5205</td>
<td>0.0005</td>
<td>0.2062</td>
<td>0.0004</td>
<td>0.5233</td>
<td>0.2045</td>
<td>0.0028</td>
<td>0.0017</td>
<td>0.5733</td>
<td>0.9780</td>
</tr>
<tr>
<td>L.4.2</td>
<td>0.2486</td>
<td>0.0004</td>
<td>0.2627</td>
<td>0.0004</td>
<td>0.2505</td>
<td>0.2612</td>
<td>0.0019</td>
<td>0.0015</td>
<td>0.7984</td>
<td>0.3828</td>
</tr>
<tr>
<td>L.4.3</td>
<td>0.0540</td>
<td>0.0002</td>
<td>0.2784</td>
<td>0.0005</td>
<td>0.0544</td>
<td>0.2770</td>
<td>0.0004</td>
<td>0.0014</td>
<td>0.0000</td>
<td>0.3610</td>
</tr>
<tr>
<td>L.4.4</td>
<td>0.0025</td>
<td>0.0000</td>
<td>0.2792</td>
<td>0.0004</td>
<td>0.0025</td>
<td>0.2778</td>
<td>0.0000</td>
<td>0.0014</td>
<td>0.0000</td>
<td>0.3600</td>
</tr>
<tr>
<td>L.5.0</td>
<td>0.8690</td>
<td>0.0003</td>
<td>0.0242</td>
<td>0.0002</td>
<td>0.8702</td>
<td>0.0238</td>
<td>0.0012</td>
<td>0.0004</td>
<td>0.1149</td>
<td>0.0000</td>
</tr>
<tr>
<td>L.5.1</td>
<td>0.7407</td>
<td>0.0003</td>
<td>0.0431</td>
<td>0.0002</td>
<td>0.7432</td>
<td>0.0422</td>
<td>0.0025</td>
<td>0.0009</td>
<td>0.2691</td>
<td>2.3697</td>
</tr>
<tr>
<td>L.5.2</td>
<td>0.5183</td>
<td>0.0004</td>
<td>0.0661</td>
<td>0.0002</td>
<td>0.5212</td>
<td>0.0657</td>
<td>0.0029</td>
<td>0.0004</td>
<td>0.5756</td>
<td>0.0000</td>
</tr>
<tr>
<td>L.5.3</td>
<td>0.2302</td>
<td>0.0003</td>
<td>0.0843</td>
<td>0.0002</td>
<td>0.2335</td>
<td>0.0826</td>
<td>0.0033</td>
<td>0.0017</td>
<td>1.2848</td>
<td>2.4213</td>
</tr>
<tr>
<td>L.5.4</td>
<td>0.0407</td>
<td>0.0001</td>
<td>0.0869</td>
<td>0.0003</td>
<td>0.0410</td>
<td>0.0864</td>
<td>0.0003</td>
<td>0.0005</td>
<td>0.0000</td>
<td>1.1574</td>
</tr>
</tbody>
</table>

(a) Matched Boundary

<table>
<thead>
<tr>
<th>Test</th>
<th>$T_s$</th>
<th>$T_t$ err.</th>
<th>$R_s$</th>
<th>$R_t$ err.</th>
<th>$T_p$</th>
<th>$R_p$</th>
<th>$T_d$</th>
<th>$R_d$</th>
<th>$%$ $T_d/T_p$</th>
<th>$%$ $R_d/R_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.6.0</td>
<td>0.5140</td>
<td>0.0006</td>
<td>0.2361</td>
<td>0.0004</td>
<td>0.5149</td>
<td>0.2348</td>
<td>0.0009</td>
<td>0.0013</td>
<td>0.1942</td>
<td>0.4259</td>
</tr>
<tr>
<td>L.6.1</td>
<td>0.2977</td>
<td>0.0003</td>
<td>0.2736</td>
<td>0.0004</td>
<td>0.2988</td>
<td>0.2721</td>
<td>0.0011</td>
<td>0.0015</td>
<td>0.3347</td>
<td>0.3675</td>
</tr>
<tr>
<td>L.6.2</td>
<td>0.1036</td>
<td>0.0002</td>
<td>0.2858</td>
<td>0.0003</td>
<td>0.1038</td>
<td>0.2835</td>
<td>0.0002</td>
<td>0.0023</td>
<td>0.0000</td>
<td>0.7055</td>
</tr>
<tr>
<td>L.6.3</td>
<td>0.0129</td>
<td>0.0001</td>
<td>0.2868</td>
<td>0.0004</td>
<td>0.0127</td>
<td>0.2845</td>
<td>0.0002</td>
<td>0.0023</td>
<td>0.0000</td>
<td>0.7030</td>
</tr>
<tr>
<td>L.6.4</td>
<td>0.0002</td>
<td>0.0000</td>
<td>0.2864</td>
<td>0.0002</td>
<td>0.0002</td>
<td>0.2845</td>
<td>0.0000</td>
<td>0.0019</td>
<td>0.0000</td>
<td>0.7030</td>
</tr>
<tr>
<td>L.7.0</td>
<td>0.6034</td>
<td>0.0003</td>
<td>0.1672</td>
<td>0.0003</td>
<td>0.6046</td>
<td>0.1659</td>
<td>0.0012</td>
<td>0.0013</td>
<td>0.1654</td>
<td>0.6028</td>
</tr>
<tr>
<td>L.7.1</td>
<td>0.3985</td>
<td>0.0006</td>
<td>0.1897</td>
<td>0.0003</td>
<td>0.3999</td>
<td>0.1883</td>
<td>0.0015</td>
<td>0.0014</td>
<td>0.2501</td>
<td>0.5311</td>
</tr>
<tr>
<td>L.7.2</td>
<td>0.1801</td>
<td>0.0004</td>
<td>0.1920</td>
<td>0.0004</td>
<td>0.1805</td>
<td>0.1905</td>
<td>0.0004</td>
<td>0.0015</td>
<td>0.0000</td>
<td>1.0499</td>
</tr>
<tr>
<td>L.7.3</td>
<td>0.0382</td>
<td>0.0001</td>
<td>0.1906</td>
<td>0.0003</td>
<td>0.0382</td>
<td>0.1886</td>
<td>0.0000</td>
<td>0.0020</td>
<td>0.0000</td>
<td>1.0604</td>
</tr>
<tr>
<td>L.7.4</td>
<td>0.0018</td>
<td>0.0000</td>
<td>0.1904</td>
<td>0.0003</td>
<td>0.0017</td>
<td>0.1885</td>
<td>0.0001</td>
<td>0.0019</td>
<td>0.0000</td>
<td>1.0610</td>
</tr>
<tr>
<td>L.8.0</td>
<td>0.7564</td>
<td>0.0004</td>
<td>0.0918</td>
<td>0.0001</td>
<td>0.7590</td>
<td>0.0902</td>
<td>0.0026</td>
<td>0.0016</td>
<td>0.3953</td>
<td>2.2173</td>
</tr>
<tr>
<td>L.8.1</td>
<td>0.6111</td>
<td>0.0004</td>
<td>0.0960</td>
<td>0.0001</td>
<td>0.6139</td>
<td>0.0944</td>
<td>0.0028</td>
<td>0.0016</td>
<td>0.4887</td>
<td>2.1186</td>
</tr>
<tr>
<td>L.8.2</td>
<td>0.3890</td>
<td>0.0005</td>
<td>0.0935</td>
<td>0.0002</td>
<td>0.3918</td>
<td>0.0920</td>
<td>0.0028</td>
<td>0.0015</td>
<td>0.7657</td>
<td>1.0870</td>
</tr>
<tr>
<td>L.8.3</td>
<td>0.1560</td>
<td>0.0003</td>
<td>0.0834</td>
<td>0.0001</td>
<td>0.1568</td>
<td>0.0823</td>
<td>0.0008</td>
<td>0.0011</td>
<td>0.6378</td>
<td>1.2151</td>
</tr>
<tr>
<td>L.8.4</td>
<td>0.0258</td>
<td>0.0001</td>
<td>0.0798</td>
<td>0.0002</td>
<td>0.0259</td>
<td>0.0786</td>
<td>0.0001</td>
<td>0.0012</td>
<td>0.0000</td>
<td>1.2723</td>
</tr>
</tbody>
</table>

(b) Mismatched Boundary

Tab. 3.2 – Transmission and Reflectance from simulation ($T_s$, $R_s$) and error. Transmission and Reflectance from Prahlo ($T_p$, $R_p$), Delta between Prahlo and simulation ($T_d$, $R_d$), and the percentage difference.
ellipsometry-based method for all wavelengths. This resulted in a $n$ ranging from 1.57 to 1.54, but since the values were close together a constant value of 1.55 was used. The $\mu_a$, $\mu_s$, and $n$ determined using this process are used as input for the simulations. The specifics of each separate simulation is listed below.

1. The first test is a scattering test, using the mentioned 1000 wavelengths, specified as different $\mu_a/\mu_s$ combinations. The simulation is executed for 5 different thicknesses of the homogeneous medium, that has a fixed $g$ of 0.7 and $n$ of 1.55.

2. The second is a transparent test, using the 1000 wavelengths, specified as different $\mu_a$ values. The simulation is executed for 5 different thicknesses of the homogeneous medium using a fixed $g$ of 0.7, $n$ of 1.55, and $\mu_s$ close to 0. The absence of $\mu_s$ makes the tissue transparent, no scattering causes photons to travel in a straight line.

3. The last test is one that implements a two fiber setup. This is the setup that is used most in the experiments by Philips Research C&HA. There is one source fiber, fixed in its position. The detector fiber is placed at nine different distances from the source fiber, and records the detected reflection. This test verifies that the spectral signature of the simulation tissue is comparable to the signature observed in an experimental setup.

Several experimental considerations are taken into account when comparing the simulation results against the experimental results. Determining the optical properties using an integrated sphere setup has issues that cause inaccuracies in measurements:

(for a detailed description see Uzunbajakava [35])

- light is lost at edges of tissue samples,
- accuracy of sample thickness estimations,
- reliability of collimated transmission measurement,
- surface condition of tissue,
- accuracy of distance between source/detector and tissue.

The accuracy for determining the $\mu_a$ using this method and setup is between 1% and 10%, with an average of 2.1% [35]. Also the difference in spectral resolution for the experimental setup and the integrating sphere setup causes a shift in accuracy. The spectral resolution of the setup to measure $\mu_a$, and $\mu_s$ is 5nm per data point, while the spectral sampling of the experimental setup is 0.94nm per pixel. The resolution of the imaging system of the spectrometer used in the experimental setup is 2.3nm, meaning that the experimental setup is twice more accurate than the integrating sphere setup.

**Hypothesis 4 (Phantom verification)**

Reflectance and/or transmission measured in the three simulations should match the experimental data within ten percent.

For these tests, ten percent is a reasonable measurement of accuracy, since both the determination of tissue optical properties and experimental measurement are accurate with a five percent margin. This means that a difference of ten percent between experiment and simulation is expected.
Results

1. The scattering test was executed using the properties listed above. Figures 3.3 through 3.7 show the comparison of the simulations versus experiments. All graphs show that the simulation results are within ten percent of the experimental data. The simulations have a maximum standard error of 0.00111 for transmission, and 0.00110 for reflection.

![Graph showing comparison between simulation and experiment](image)

**Fig. 3.3 – Scattering phantom experiment vs. simulation, thickness=830μm**
**Fig. 3.4** – *Scattering phantom experiment vs. simulation, thickness=1080µm*

**Fig. 3.5** – *Scattering phantom experiment vs. simulation, thickness=1100µm*
Fig. 3.6 – Scattering phantom experiment vs. simulation, thickness=1210 µm

Fig. 3.7 – Scattering phantom experiment vs. simulation, thickness=1510 µm
2. The results from the transparent phantom simulations are shown in Figures 3.8 through 3.12. The graphs show that the simulation results are within ten percent of the experimental data. The simulations have a maximum standard error of 0.00106 for transmission, and 0.00005 for reflection.

![Graph showing simulation and experimental data comparison.](image)

**Fig. 3.8** – *Transparent phantom experiment vs. simulation, thickness=1510µm*
Fig. 3.9 – *Transparent phantom experiment vs. simulation, thickness=2970µm*

Fig. 3.10 – *Transparent phantom experiment vs. simulation, thickness=3450µm*
Fig. 3.11 – *Transparent phantom experiment vs. simulation, thickness=10030µm*

Fig. 3.12 – *Transparent phantom experiment vs. simulation, thickness=14990µm*
3. Testing a two fiber setup, where there is one source of photons, and one detector of reflected photons, verifies the most used experimental setup. The results from experiment and simulation are shown in the graphs in Fig. 3.13. In the graph set the results from the experiments and simulations are normalized to values between 0 and 1, using the following equation

\[ Y_n = Y_i \times \frac{1}{\max(Y_1 \ldots Y_N)} \]  

(3.3)

, for \(1 \leq i \leq N\), where \(N\) is the number of data points. Normalization is done since the results produced in the experiments and simulations never yield the exact same value. The amount of photons detected is expressed in a percentage over the amount of photons launched. This percentage should be the same for experiment and simulation. Normalization causes misalignment of graphs in the direction. It is done using the maximum value in the data set, found mostly at the lower wavelengths. Jitter in this region, results in a downward shift. The first observation from the graph set shows that the experimental data has more defined ('sharp') peaks as opposed to the simulation data. This property is explained by the fact that the detection of light in the experimental setup is twice more sensitive than the setup that determines the optical properties of the tissue. This results in a more 'smooth' result set from the simulation as opposed to the sharp peaks in the experimental data set.

It is also worth noting that the percentages of detected photons versus launched photons is very small. For the simulation the maximum detected percentage is shown in Tab. 3.3. Even with this small number of detected photons in the simulation, the spectral properties of the tissue are comparable to those measured in the experiment.

At fiber separation distances larger than 1660 µm, the simulation is outside the allowed boundaries, but this is still considered a good fit when taking into account the amount of detected photons. The reason that the two data sets don’t match perfectly could be the result of normalization, since this is based on maximum values. Also comparison using a 10% relative difference is not accurate enough.

<table>
<thead>
<tr>
<th>Fiber Separation (µm)</th>
<th>Max. Detected (%)</th>
<th>Std. Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>0.0028</td>
<td>7.2297e-04</td>
</tr>
<tr>
<td>810</td>
<td>0.0027</td>
<td>7.1445e-04</td>
</tr>
<tr>
<td>860</td>
<td>0.0025</td>
<td>6.7250e-04</td>
</tr>
<tr>
<td>960</td>
<td>0.0023</td>
<td>6.0259e-04</td>
</tr>
<tr>
<td>1160</td>
<td>0.0018</td>
<td>4.9438e-04</td>
</tr>
<tr>
<td>1660</td>
<td>0.0012</td>
<td>3.2557e-04</td>
</tr>
<tr>
<td>2660</td>
<td>0.0006</td>
<td>1.6445e-04</td>
</tr>
<tr>
<td>4660</td>
<td>0.0002</td>
<td>5.6501e-05</td>
</tr>
<tr>
<td>8660</td>
<td>0.0001</td>
<td>1.1734e-05</td>
</tr>
</tbody>
</table>

Tab. 3.3 – *Maximum percentage and standard error for detected photons vs. launched photons in simulation.*
Fig. 3.13 – Two fiber phantom test for wavelengths 1100 - 1800 with increasing distance between fibers. Y axis contains normalized ratio detected/launched photons.
Conclusion

Verification of the software using phantom experiments is valuable in determining the quality of the results produced by the simulation. Most of the results of the simulation versus the experiments from this chapter match within the allowed limits. Even detecting small amounts of photons using a detection fiber give an accurate approximation of the experimental data. It is advisable to perform extra experiments in more complicated configurations to test every configurable aspect of the software. These experiments can than be added to the set of phantom tests.

3.2.4 Conclusion

From the tests performed it is clear that the current, corrected, code base is stable enough for further enhancements. The set of acceptance tests forms only a base, verifying the correct functioning of the basic simulation functionality. The more intricate functions of the simulation, such as tilting fibers, should also be added to the acceptance framework as soon as results from experiments are available. The implemented logic is verified by the unit tests, but it should also be compared to experimental results.

The verification framework is easily extended to incorporate testing of new features. When new experimental results are available, these can also be incorporated to provide even more confidence in the code produced.
Chapter 4

Complex Configuration Implementation

The verification framework provides a stable base for implementing new features. This is a prerequisite for the proposed changes in this chapter. First the software is adapted to allow arbitrary placement of multiple source and detection fibers throughout the simulation area. Within Philips Research C&HA simulation requirements are expanding, due to new ideas for fiber arrangements. Avoiding tissue components, such as hair follicles, is done by using more than one detection and source fiber. The first step is to allow simulations using arbitrary fiber arrangements, matching experimental configurations. The next step, not addressed in this project, is to create a heterogeneous tissue model. The current implementation is adapted to allow multiple source and detection fibers.

The second part investigates a possible parallel solution for reducing execution time, and memory requirements. The parallel solution is designed for use with future complex tissue models.
4.1. Matrix implementation

The MCOM software is currently capable to simulate one single run (or a batch of single runs) for a setup consisting of one source and optionally one detector of photons (Fig. 4.1(a)). This setup suffices for most simulations, but a more flexible setup is required, where more sources and detectors are used (Fig. 4.1(b)).

![Simulation area seen from above, depicting arrangement of sources and detectors.](image)

4.1.1 Existing Software

Simulations are now created using a single source of photon packets, where the source can be of a certain type. This source launches photon packets into the specified layers of tissue, and their path through these layers is then simulated. The simulation ends when either a specific number of photon packets is simulated, or (if a fiber is specified) a required standard deviation is reached. Within one software execution run, a multiple of these simulations (with different configurations) can be performed. The new implementation is based on the current software, and extends it using the new requirements. The requirements listed in the following section are created under the following assumptions:

- The original functionality of the simulation remains intact, and can still be performed using the new implementation,
- One simulation is specified as one run from a batch file, for the specified number of wavelengths,
- Each Source and Detector used is rotational symmetric,
- Photons do not interact with each other in the simulation.
4.1.2 User Requirements

Requirements for the enhancement are gathered from the users of the software within Philips Research C&HA. The gathered list is broken up in two parts: high priority requirements (uh), and low priority requirements (ul). Both of these are listed below:

**High Priority User Requirements 1**

- **uh-1** A simulation has one or more sources
- **uh-2** A simulation has zero or more detectors
- **uh-3** The position of source/detector is specified with a coordinate consisting of 3 floats.
- **uh-4** The coordinate is of significance to the implementation of the source/detector.
- **uh-5** A coordinate may contain zero or more sources/detectors.
- **uh-6** Photons are emitted from all sources.
- **uh-7** Photons are launched randomly distributed over all specified sources.
- **uh-8** Photons are detected by all detectors.
- **uh-9** The simulation always has a maximum number of photons defined for launch.
- **uh-10** The simulation may have a standard deviation defined.
- **uh-11** The simulation always ends if the maximum number of photons are launched.
- **uh-12** If the simulation has a standard deviation defined, the simulation stops when the standard deviation reached is lower or equal to the defined standard deviation.
- **uh-13** The standard deviation is calculated over all photons that are detected by one or more of the defined detectors.
- **uh-14** Each detector keeps track of the number of detected photons.
- **uh-15** Input files remain comma separated files, but the layout may change to accommodate changes.
- **uh-16** Separate input files are used to specify the sources and detectors.
- **uh-17** Sources and detectors are specified using a source/detector type and position.
- **uh-18** Output is done using the existing format (.atr, .pht, .sim, .ref, .abs, .tra files), but may change to accommodate changes

**Low Priority User Requirements 1**

- **ul-1** Intensity of photon launches can be specified per source.

4.1.3 Design

From the user requirements, the following design points are created. All listed design points include references to the user requirements that it implements. The cross link between user requirements and design are shown in Tab. 4.1.
The batch input file, and related files change to accommodate the matrix. The old format of input files does not suffice for the new simulation input. The new comma separated files are defined using Backus Naur Form (BNF) [36] (uh-15):

File ::= Variables (Record)* 'EOF'
Variables ::= VarStringList ('\n' | 'EOF')
Record ::= StringList ('\n' | 'EOF')
VarStringList ::= simpleField [',' simpleField]
StringList ::= rawField [',' StringList]
rawField ::= simpleField | quotedField
simpleField ::= (any char except \n, EOF, \t, space, comma or double quote)+
quotedField ::= ''' escapedField '''
escapedField ::= (any char except \n, EOF, \t, space, or double quote)+

The new simulation has four different types of input files, that adhere to the previously mentioned rules. These files are the following:

**Batch File** A CSV file containing 1 or more simulations, each row containing one simulation. This batch file references three files: a detector file, a source file, and a layer file.

**Detector File** A CSV file containing 0 or more detectors (uh-1), each row containing one detector (uh-16)

**Source File** A CSV file containing 1 or more sources (uh-2), each row containing one source (uh-16)
Layer File A CSV file containing 3 or more layers, each row containing one layer. This file references 3 or more wavelength files. The first and last layer are special layers describing the top and bottom layers, that are not part of the simulating tissue, but are used only for specifying the refractive index.

Wavelength File A CSV file containing 1 or more wavelengths, each row containing one wavelength.

Fig. 4.2 shows the relation between configuration files. The description of all variables in each of the mentioned files is available in Appendix C.1.

The source and detector file have the same variables, specifying the position using three variables (posX,posY,posZ) (uh-3), and the type (type) (uh-17). The subclasses of the abstract source and detector classes implement the launching according to the location from the configuration file (uh-4). No restriction is placed on the the specification of more than one source/detector on one location (uh-5), meaning that overlap is possible. This implementation does not represent physical possibilities, but allows flexibility in the definition of a simulation.
DI-2 *The method of reading in the input files changes to adapt to the new input file format.*

A general purpose configuration file reader class is created to accommodate the loading of the configuration files (uh-15). This class replaces the current method of reading in configuration files, using the SimInputParams class. Validation of input variables is the task of the class reading in the variables using the CConfig class.

DI-3 *Each simulation has a list of detectors and sources.*

At the start of the simulation all available sources and detectors are read into the simulation. At least one source is defined in the source file (uh-1). The detector file needs to exist, but can be empty (except for the first line that contains the variable names)(uh-2). Each of the sources and detectors is stored in a list that remains unchanged during a single simulation run.

DI-4 *Before launching a photon, a specific source is selected.*

Fig. 4.3 shows the new simulation activity graph, where the source selection is added before launching a photon. The method of source selection is changed to allow launching of a photon from more than one source (uh-6). In the current simulation, photons are launched from one source, and the launch position is distributed randomly over the launch area of the source.

Implementing multiple sources is comparable to using a large source with an irregular shape.

![Activity graph for matrix MCOM simulation](image)

**Fig. 4.3 – Activity graph for matrix MCOM simulation**

All sources are treated as a part of one large source, using one light source, allowing the random distribution of launching photons (uh-7) (Fig. 4.4).
This method of choosing a source implies that the ratio of photons launched per surface area decreases when the surface area increases. This distribution does not represent reality, and requirement (ul-1) asks for a better solution. However, this feature is not implemented since an accurate algorithm is not available at the time of design. The design of the source selector facilitates the future implementation of (ul-1), by having a weighted selection process, that currently has an equal weight assigned to each source. This method uses an implementation by Knuth [12] of the Walker algorithm [37] for selecting from a finite set of elements. This algorithm is implemented in the Karney library.

All photons launched are randomly distributed from all sources. This is achieved by assigning a weight of 1.0 to each source.

**Fig. 4.4** – *Example of conversion from a single shape source to a more realistic multiple fiber source setup.*
DI-5 Detection of a photon is done by all of the specified detectors.

Fig. 4.5 shows the implementation of the detection loop in the photon propagation cycle. When detectors are configured for a simulation, the detection is performed by all defined (uh-8). Similar to the multiple sources implementation, the detectors are treated as one large implementation.

A photon is counted if it is detected by a minimum of 1 detector. The photon is then added to a list, and stored in the '.pht' file upon request. The photon is added to all detector counts that have detected the photon (uh-8, uh-14).

![Diagram](image)

Fig. 4.5 – Single photon propagation lifetime activity graph

DI-6 Termination of a simulation changes.

To accommodate the new detectors, termination requirements are changed. A simulation now terminates according to the following rules:

- If detectors are defined:
  - If a standard deviation is requested (uh-10), the simulation terminates if the calculated standard deviation is equal or lower than the requested standard deviation (uh-12).
  - If a standard deviation is requested, but the calculated standard deviation is not
reached, the simulation stops when the maximum amount of photons is launched (uh-11).

- If no standard deviation is requested, the simulation stops when the maximum amount of photons is launched (uh-11).
- If no detectors are defined, the simulation stops when the maximum amount of photons is launched. (ul-11).

The minimum and maximum number of photons launched by a simulation is defined using three variables: \textit{minSeeds}, \textit{maxSeeds}, and \textit{seedSize} (uh-9). The \textit{seedSize} specifies the size of a seed used for calculation of the standard deviation. The \textit{minSeeds} specifies the lower boundary of how many seeds of \textit{seedSize} are processed, and \textit{maxSeeds} defines the maximum seeds needed.

The standard deviation is still calculated using the method described in Section 2.2.8, using the photons that are detected for each seed of size \textit{seedSize} (uh-13).

\textbf{DI-7 \ Recording of information is attached to a simulation, and specific to that simulation.}

Two output data files are changed to store the extra information generated by the simulation (uh-18). The changed output files are the constant simulation information (Section 2.6.1), and the current run information (Section 2.6.2). The optional files remain unchanged, since these files are not influenced by the changes made.

The new layout of the two files is specified in Appendix C.2

\textbf{4.1.4 UML}

The completed UML diagram for the matrix implementation is shown in Fig. 4.6. Configuration files are initially loaded by the CSimulation class, but parsed and validated by the corresponding classes. An example of this is shown in the sequence diagram Fig. 4.7. The CSimulation class initializes the CConfig object, using the configuration file. From the file, the number of CDetection objects is determined, and created using the existing CConfig object. All specific variables are loaded from file, and verified within the CDetection class. This method of loading, and verifying the configuration files is implemented analogously for the CTissue, CLayer, and CSource classes.
Fig. 4.6 – UML Class diagram overview. No attributes/operations shown.

Fig. 4.7 – UML sequence diagram depicting initialization of detectors, including loading/validation of variables from configuration file.
4.1.5 Testing

Testing during development is done using Unit tests (Section 3.1). At the moment of writing, the software is not completed yet, but preliminary acceptance testing is executed. The following are the results of this testing.

Optical Properties Tests

Fig. 4.8 shows the result of optical properties verification. Both the step absorption and step scattering show good results. The simulated results match the expected data, meaning that the basic optical properties are implemented correctly.

(a) Absorption: \( \mu_a, \mu_s = 0, n = 1, g = 0 \)

(b) Scattering: \( \mu_s, \mu_a = 0, n = 1.0, g = 0.0 \)

(c) Scattering: \( \mu_s, \mu_a = 0, n = 1.0, g = 0.5 \)

(d) Scattering: \( \mu_s, \mu_a = 0, n = 1.0, g = 1.0 \)

**Fig. 4.8 – Optical Properties verification**
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>van de Hulst, 1980</td>
<td>0.09739</td>
<td></td>
<td>0.66096</td>
<td></td>
</tr>
<tr>
<td>Prahl et al., 1989</td>
<td>0.09711</td>
<td>0.00033</td>
<td>0.66159</td>
<td>0.00049</td>
</tr>
<tr>
<td>Wang et al., 1995</td>
<td>0.09734</td>
<td>0.00035</td>
<td>0.66096</td>
<td>0.00020</td>
</tr>
<tr>
<td>Test L1, 2008</td>
<td>0.09760</td>
<td>0.00035</td>
<td>0.66040</td>
<td>0.00035</td>
</tr>
</tbody>
</table>

(a) thickness = 0.02 cm, $\mu_a = 10, \mu_s = 90, g = 0.75, n = 1$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Giovanelli, 1955</td>
<td>0.2600</td>
<td></td>
</tr>
<tr>
<td>Prahl et al., 1989</td>
<td>0.26079</td>
<td>0.00079</td>
</tr>
<tr>
<td>Wang et al., 1995</td>
<td>0.25907</td>
<td>0.00170</td>
</tr>
<tr>
<td>Test L2, 2008</td>
<td>0.22913</td>
<td>0.00021</td>
</tr>
</tbody>
</table>

(b) thickness = $\infty, \mu_a = 10, \mu_s = 90, g = 0, n = 1.5$

Tab. 4.2 – Wang & Jacques Literature Tests for matrix implementation.

**Literature Tests**

The first Wang & Jacques comparison in Tab. 4.2(a), with a matched boundary, show good results. The transmission shows a slight deviation from the other values, where the other results are all close to 0.661, the simulation results stay below this value. This might indicate a problem, although the deviation is small.

The second table, Tab. 4.2(b), shows a bigger problem. With a mismatched boundary not enough reflection is measured from the simulation. The accuracy of the ten measurements is good, with a standard error of 0.00021, but the reflection of 0.22913 is too low. This indicates that there is a problem with the boundary conditions, causing less detected reflection.

The results and problems detected by the previous literature tests are also be visible in the comparison with the Prahl adding-doubling method. The matched boundary verification, Tab. 4.3(a), shows promising results. None of the tests differ more than 2.5% from the expected value. This is well within the allowed 3% difference.

However, the mismatched boundary verification, Tab. 4.3(b), shows big problems. None of the values for reflection, and only five values for transmission match the expected results. The reflection deviates greatly from the expected values, indicating that the issue lies with the crossing of boundaries.

**Phantom Tests**

The first two phantom verification tests (Figures 4.9 to 4.13, and Figures 4.14 to 4.18) show the same results, where both transmission and reflection deviate from the accepted limits. Since all of the phantom tests have a mismatched boundary with the surrounding environment, it becomes clear that this is the problem.
<table>
<thead>
<tr>
<th>Test</th>
<th>$T_s$</th>
<th>$T_i$ err.</th>
<th>$R_s$</th>
<th>$R_i$ err.</th>
<th>$T_p$</th>
<th>$R_p$</th>
<th>$T_d$</th>
<th>$R_d$</th>
<th>$% T_d/T_p$</th>
<th>$% R_d/R_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.3.0</td>
<td>0.5902</td>
<td>0.0007</td>
<td>0.2687</td>
<td>0.0007</td>
<td>0.5916</td>
<td>0.2674</td>
<td>0.0014</td>
<td>0.0012</td>
<td>0.1690</td>
<td>0.3740</td>
</tr>
<tr>
<td>L.3.1</td>
<td>0.3538</td>
<td>0.0007</td>
<td>0.3637</td>
<td>0.0004</td>
<td>0.3656</td>
<td>0.3616</td>
<td>0.0027</td>
<td>0.0021</td>
<td>0.8415</td>
<td>0.5531</td>
</tr>
<tr>
<td>L.3.2</td>
<td>0.1270</td>
<td>0.0003</td>
<td>0.4094</td>
<td>0.0006</td>
<td>0.1285</td>
<td>0.4081</td>
<td>0.0015</td>
<td>0.0013</td>
<td>1.5564</td>
<td>0.2450</td>
</tr>
<tr>
<td>L.3.3</td>
<td>0.0158</td>
<td>0.0001</td>
<td>0.4154</td>
<td>0.0004</td>
<td>0.0160</td>
<td>0.4148</td>
<td>0.0002</td>
<td>0.0006</td>
<td>0.0000</td>
<td>0.2411</td>
</tr>
<tr>
<td>L.3.4</td>
<td>0.0002</td>
<td>0.0000</td>
<td>0.4171</td>
<td>0.0002</td>
<td>0.0002</td>
<td>0.4149</td>
<td>0.0000</td>
<td>0.0021</td>
<td>0.0000</td>
<td>0.4820</td>
</tr>
<tr>
<td>L.4.0</td>
<td>0.7377</td>
<td>0.0005</td>
<td>0.1304</td>
<td>0.0004</td>
<td>0.7391</td>
<td>0.1298</td>
<td>0.0014</td>
<td>0.0006</td>
<td>0.1353</td>
<td>0.7704</td>
</tr>
<tr>
<td>L.4.1</td>
<td>0.5199</td>
<td>0.0003</td>
<td>0.2655</td>
<td>0.0005</td>
<td>0.5233</td>
<td>0.2045</td>
<td>0.0034</td>
<td>0.0020</td>
<td>0.5733</td>
<td>0.9780</td>
</tr>
<tr>
<td>L.4.2</td>
<td>0.2487</td>
<td>0.0004</td>
<td>0.2627</td>
<td>0.0004</td>
<td>0.2505</td>
<td>0.2612</td>
<td>0.0018</td>
<td>0.0015</td>
<td>0.7984</td>
<td>0.7657</td>
</tr>
<tr>
<td>L.4.3</td>
<td>0.0541</td>
<td>0.0002</td>
<td>0.2790</td>
<td>0.0005</td>
<td>0.0544</td>
<td>0.2770</td>
<td>0.0003</td>
<td>0.0020</td>
<td>0.0000</td>
<td>0.7220</td>
</tr>
<tr>
<td>L.4.4</td>
<td>0.0025</td>
<td>0.0000</td>
<td>0.2798</td>
<td>0.0004</td>
<td>0.0025</td>
<td>0.2778</td>
<td>0.0000</td>
<td>0.0020</td>
<td>0.0000</td>
<td>0.7199</td>
</tr>
<tr>
<td>L.5.0</td>
<td>0.8689</td>
<td>0.0003</td>
<td>0.0242</td>
<td>0.0002</td>
<td>0.8702</td>
<td>0.0238</td>
<td>0.0013</td>
<td>0.0004</td>
<td>0.1149</td>
<td>0.0000</td>
</tr>
<tr>
<td>L.5.1</td>
<td>0.7408</td>
<td>0.0004</td>
<td>0.0431</td>
<td>0.0002</td>
<td>0.7432</td>
<td>0.0422</td>
<td>0.0024</td>
<td>0.0009</td>
<td>0.2691</td>
<td>2.3697</td>
</tr>
<tr>
<td>L.5.2</td>
<td>0.5173</td>
<td>0.0004</td>
<td>0.0670</td>
<td>0.0003</td>
<td>0.5212</td>
<td>0.0657</td>
<td>0.0039</td>
<td>0.0013</td>
<td>0.7675</td>
<td>1.5221</td>
</tr>
<tr>
<td>L.5.3</td>
<td>0.2303</td>
<td>0.0002</td>
<td>0.0835</td>
<td>0.0002</td>
<td>0.2335</td>
<td>0.0826</td>
<td>0.0032</td>
<td>0.0009</td>
<td>1.2848</td>
<td>1.2107</td>
</tr>
<tr>
<td>L.5.4</td>
<td>0.0404</td>
<td>0.0001</td>
<td>0.0874</td>
<td>0.0003</td>
<td>0.0410</td>
<td>0.0864</td>
<td>0.0006</td>
<td>0.0010</td>
<td>2.4390</td>
<td>1.1574</td>
</tr>
</tbody>
</table>

(a) Matched Boundary

<table>
<thead>
<tr>
<th>Test</th>
<th>$T_s$</th>
<th>$T_i$ err.</th>
<th>$R_s$</th>
<th>$R_i$ err.</th>
<th>$T_p$</th>
<th>$R_p$</th>
<th>$T_d$</th>
<th>$R_d$</th>
<th>$% T_d/T_p$</th>
<th>$% R_d/R_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.6.0</td>
<td>0.4899</td>
<td>0.0006</td>
<td>0.1524</td>
<td>0.0004</td>
<td>0.5149</td>
<td>0.2348</td>
<td>0.0250</td>
<td>0.0824</td>
<td>4.8553</td>
<td>34.9233</td>
</tr>
<tr>
<td>L.6.1</td>
<td>0.2748</td>
<td>0.0004</td>
<td>0.1933</td>
<td>0.0002</td>
<td>0.2988</td>
<td>0.2721</td>
<td>0.0240</td>
<td>0.0788</td>
<td>8.0321</td>
<td>29.0334</td>
</tr>
<tr>
<td>L.6.2</td>
<td>0.0929</td>
<td>0.0004</td>
<td>0.2080</td>
<td>0.0002</td>
<td>0.1038</td>
<td>0.2835</td>
<td>0.0109</td>
<td>0.0755</td>
<td>10.5973</td>
<td>26.4550</td>
</tr>
<tr>
<td>L.6.3</td>
<td>0.0112</td>
<td>0.0001</td>
<td>0.2085</td>
<td>0.0002</td>
<td>0.0127</td>
<td>0.2845</td>
<td>0.0015</td>
<td>0.0760</td>
<td>7.8740</td>
<td>26.7135</td>
</tr>
<tr>
<td>L.6.4</td>
<td>0.0002</td>
<td>0.0000</td>
<td>0.2092</td>
<td>0.0004</td>
<td>0.0002</td>
<td>0.2845</td>
<td>0.0000</td>
<td>0.0753</td>
<td>0.0000</td>
<td>26.3620</td>
</tr>
<tr>
<td>L.7.0</td>
<td>0.5834</td>
<td>0.0008</td>
<td>0.0954</td>
<td>0.0006</td>
<td>0.6046</td>
<td>0.1659</td>
<td>0.0212</td>
<td>0.0706</td>
<td>3.4734</td>
<td>42.7969</td>
</tr>
<tr>
<td>L.7.1</td>
<td>0.3751</td>
<td>0.0005</td>
<td>0.1219</td>
<td>0.0003</td>
<td>0.3999</td>
<td>0.1883</td>
<td>0.0248</td>
<td>0.0664</td>
<td>6.2516</td>
<td>35.0505</td>
</tr>
<tr>
<td>L.7.2</td>
<td>0.1643</td>
<td>0.0004</td>
<td>0.1290</td>
<td>0.0003</td>
<td>0.1805</td>
<td>0.1905</td>
<td>0.0162</td>
<td>0.0615</td>
<td>8.8643</td>
<td>32.0210</td>
</tr>
<tr>
<td>L.7.3</td>
<td>0.0343</td>
<td>0.0001</td>
<td>0.1287</td>
<td>0.0003</td>
<td>0.0382</td>
<td>0.1886</td>
<td>0.0040</td>
<td>0.0599</td>
<td>10.4712</td>
<td>31.8134</td>
</tr>
<tr>
<td>L.7.4</td>
<td>0.0015</td>
<td>0.0000</td>
<td>0.1288</td>
<td>0.0003</td>
<td>0.0017</td>
<td>0.1885</td>
<td>0.0002</td>
<td>0.0597</td>
<td>0.0000</td>
<td>31.8302</td>
</tr>
<tr>
<td>L.8.0</td>
<td>0.7722</td>
<td>0.0004</td>
<td>0.0412</td>
<td>0.0001</td>
<td>0.7590</td>
<td>0.0902</td>
<td>0.0132</td>
<td>0.0490</td>
<td>1.7128</td>
<td>54.3237</td>
</tr>
<tr>
<td>L.8.1</td>
<td>0.6163</td>
<td>0.0004</td>
<td>0.0451</td>
<td>0.0001</td>
<td>0.6139</td>
<td>0.0944</td>
<td>0.0024</td>
<td>0.0493</td>
<td>0.3258</td>
<td>51.9068</td>
</tr>
<tr>
<td>L.8.2</td>
<td>0.3845</td>
<td>0.0005</td>
<td>0.0446</td>
<td>0.0001</td>
<td>0.3918</td>
<td>0.0920</td>
<td>0.0073</td>
<td>0.0474</td>
<td>1.7866</td>
<td>51.0870</td>
</tr>
<tr>
<td>L.8.3</td>
<td>0.1484</td>
<td>0.0002</td>
<td>0.0375</td>
<td>0.0001</td>
<td>0.1568</td>
<td>0.0823</td>
<td>0.0084</td>
<td>0.0448</td>
<td>5.1020</td>
<td>54.6780</td>
</tr>
<tr>
<td>L.8.4</td>
<td>0.0241</td>
<td>0.0001</td>
<td>0.0340</td>
<td>0.0002</td>
<td>0.0259</td>
<td>0.0786</td>
<td>0.0018</td>
<td>0.0446</td>
<td>7.7220</td>
<td>57.2519</td>
</tr>
</tbody>
</table>

(b) Mismatched Boundary

**Tab. 4.3** – Transmission and Reflectance from matrix simulation ($T_s$, $R_s$) and error, Transmission and Reflectance from Pahl ($T_p$, $R_p$), Delta between Pahl and simulation ($T_d$, $R_d$), and the percentage difference.
4 Complex Configuration Implementation

Fig. 4.9 – Scattering phantom experiment vs. simulation, thickness=830µm

Fig. 4.10 – Scattering phantom experiment vs. simulation, thickness=1080µm
Fig. 4.11 – Scattering phantom experiment vs. simulation, thickness=1100µm

Fig. 4.12 – Scattering phantom experiment vs. simulation, thickness=1210µm
Fig. 4.13 – Scattering phantom experiment vs. simulation, thickness=1510µm

Fig. 4.14 – Transparent phantom experiment vs. simulation, thickness=1510µm
Fig. 4.15 – Transparent phantom experiment vs. simulation, thickness=2970\,\mu m

Fig. 4.16 – Transparent phantom experiment vs. simulation, thickness=3450\,\mu m
Fig. 4.17 – *Transparent phantom experiment vs. simulation, thickness=10030µm*

Fig. 4.18 – *Transparent phantom experiment vs. simulation, thickness=14990µm*
4.1.6 Conclusion

The new version of the software, implementing the arbitrary placement of source and detection fibers, was partly completed. Initial execution of acceptance tests shows a deviation from expected results. Since these deviations occur with a mismatch in refractive indexes between tissue layers, this is investigated further.
4.2. Parallel implementation

Changes to the MCOM software increase the overall computational and memory complexity. The detection loop adds extra calculation steps, while the extra sources add more launches to the system. Within this chapter a solution is researched, for reducing the overall running time of more complex simulations.

4.2.1 Analysis

The inherent structure of the Monte Carlo simulation is perfectly suited for parallel execution. Fig. 4.19 shows how a complete MC simulation can be broken down into independent sections.

Each single photon trajectory is a single Markov chain, where the next interaction or movement is always determined by the current state of the photon. Such a chain of events is the smallest independent component of a simulation. These chains can each be executed on a different processor without interaction between chains. This gives an almost linear decrease in execution time, with the added bonus that there is no communication needed between executing processes, meaning low overhead. This embarrassingly parallel solution has a downside. Each process has knowledge of the entire simulation environment, and has the complete storage area in memory. This causes all processes to have the same memory complexity, while this is not needed.

Monte Carlo simulations can be globally broken down into two different methods of approach: Event based, or History Based processing. The difference is shown in Fig. 4.20 and explained below.

**Event Based Simulation** processes simulation events, such as 'hop', 'drop', or 'spin', for multiple photons in a tight loop. This method of calculation is especially useful when executed on processors optimized for vector processing ([39], [38]). Since this hardware is not used, and a complete rewrite of the logic is needed, this method is not investigated further.

**History Based Simulation** takes a complete event-chain for a single photon, and simulates this sequentially from start to finish. This method of simulation is used in the current MCOM software, and is used as starting point for parallelization.

Parallelization of History Based processing of photons involves decomposition of the complete simulation into smaller pieces, simulated on different processors. There are different types of decomposition [40].

---

1 This is an interesting subject using modern graphics cards, designed for doing vector calculations in parallel.
Fig. 4.20 – Comparison of history based parallel processing and event-based vector processing [38], where $\{e_1 \ldots e_n\}$ are sequential events for one photon event-chain, and $\{p_1 \ldots p_m\}$ are photon event-chains.

**Task Farm Parallelism** is used to spread out all available photon histories over the available processors, where each processor completely processes the event-chain for each photon. The global solution is then obtained by a reduction step at the end of the simulation (Fig. 4.21). All processors operate independently of each other without the need for communication. This method gives almost a linear increase in performance, over the sequential simulation method. Each processor in the environment has knowledge of the complete simulation area.

**Geometrical Decomposition** is the solution for parallelization of photon transport for complex tissue configurations. To decrease the memory requirement of the processes, the simulation is broken down into geometric areas. Each separate processor only simulates a small portion of the photon event-chain, that occurs in the part of the simulation area assigned to it. When a photon leaves the processor’s area, the photon is passed on to a neighbouring processor (Fig. 4.22). Depending on the size of each area, communication is a large part of the complete simulation process. Since communication between processors is more time consuming than computation, this method of parallelization is less efficient in reducing execution time when compared to task farm decomposition.

**Hybrid Decomposition** is a combination of the fore mentioned methods. The simulation area is split up into smaller parts, but the number of processors greatly outnumbers the created parts. The simulation areas are equally distributed over the available processors, ensuring that at least each area is assigned to at least one processor (Fig. 4.23). This method has the advantage that memory requirements for each processor are decreased, while the communication is kept to a minimum.
Fig. 4.21 – Task farm parallelism, where each processor computes the complete history for each photon assigned to it.

Fig. 4.22 – Geometrical decomposition parallelization. Each processor only computes that part of a photon event-chain that occurs in the area known by the processor.

The hybrid method enables the simulation process to assign more processors to areas having the largest load. During simulation not all areas of simulation have the same number of interactions, a good distribution of processors to areas is a must.

Assigning areas to processors is not a trivial task, since photon interaction points are not uniformly spread out through the computational mesh. This means that more time is spent in some areas, than in others. Load balancing of the computations over all available processors is imperative for obtaining good results. The distribution of workload is not known in advance, so it requires estimation. The most logical method is to use mean free path (MFP) estimation for determining the amount of work performed in a calculation area [40]. This method assumes that in an area with a short MFP more photon interaction events take place (Fig. 4.24), and areas far away from the source tend to see fewer
Fig. 4.23 – Hybrid Decomposition. Small sections of the simulation area are assigned to multiple processors, therefore reducing communication, and decreasing memory requirements per processor.

Fig. 4.24 – MFP work estimate, where area with short MFP has more interactions than an area with a long MFP.
particles than zones close to a source. A work estimate is generated:

$$w_i = \frac{1}{\lambda_i} \sum_{j=1}^{N_s} \frac{1}{r_{i,j}}$$  \hspace{1cm} (4.1)

Where $\lambda_i$ is the MFP to scatter in zone $i$, determined by Equation 2.3. $N_s$ is the number of photon sources, and $r_{i,j}$ is the distance between source $j$ and the center of zone $i$.

The simulation area is split up into $A$ smaller pieces, where $A$ is smaller than the number of available processors. For each area $A$ the total weight is determined, by taking the sum of the calculated weight of the smaller sub parts of $A$. According to the ratio of the sum of weights a number of processors is assigned to each area.

As mentioned by Alme [40], the hybrid solution is usable if memory requirements grow large. For now the MCOM simulation memory requirements have not grown enough to use such a decomposition. Future requirements, where a more realistic model of the tissue is used, benefit from the hybrid solution.

### 4.2.2 Random Number Generator

For the implementation of a correct parallel solution for the MCOM software, the RNG needs adjustment to fit the needs of a parallel RNG implementation. A parallel RNG generator should most optimally conform to the following points (Hellekalek [41]):

1. The period of the algorithm needs to be large enough,
2. Results obtained should be reproducible,
3. The numbers generated should be uncorrelated,
4. The inner structure of the algorithm should be analyzable,
5. Should be usable for any number of processors,
6. Parallel streams of the generator should be uncorrelated,
7. Numbers should be generated independently.

Implementing a parallel simulation setup requires the simulation to produce the same results, independently of the platform and number of processor it is run on. It is not difficult to give each processors it’s own independent stream of random numbers, but this does not produce similar results in simulations using a different number of processors.

For this analysis, where a single simulation run is taken as the largest part, is broken down as shown in Fig. 4.19. Within a single run, a sequentially ordered set of $J$ wavelengths is simulated. Each wavelength in the matrix implementation has $K$ sources from where photons are launched. To guarantee the distribution of random numbers independently from the number of processors, runs, wavelengths, and sources, the following scheme is chosen.

As depicted in Fig. 4.25, each simulation starts with a random seed. The state of the RNG is represented by 19937 bits. The starting state, random seed, of the algorithm is any arrangement of these 19937 bits. From the chosen state all $2^{19937}$ possible states are visited. The number of bits is too large for practical use as a seed, so Karney’s implementation uses a vector of arbitrary size, containing 32-bit integers. This vector is used to generate the initial state of 19937 bits. The function used to
generate the starting state takes care to minimize the risk of overlap between two sequences with different random seeds. The streams generated from the seed vectors are therefore seen as independent streams. When no random seed (a single 32-bit integer, or a vector containing 32-bit integers) is given by the user, a random seed is generated from various system properties, that generate a random seed in the range $[0, 2^{32})$.

The vector representation is used to distribute the random numbers, to create independent streams for each run, wavelength, and source. This ensures that independent of the number of processors, runs, wavelengths, and sources, each has an independent stream of random numbers. The next step is to ensure that each photon launched receives the same random numbers, even if it is passed on between processes.

The distribution of photons from a launch fiber is done in blocks, where each photon receives its own block of random numbers. The block size is $\max(N) \times \max(P)$, where $N$ is the number of steps taken by a photon in propagation, and $P$ is the maximum of random numbers needed per propagation step. During propagation, the photon uses the random numbers as needed, and if the photon moves to a different process, the random numbers are also passed on. This new method of random number distribution is implemented in the matrix design.

![Fig. 4.25 – Distribution of random seed](image)

### 4.3. Conclusion

The matrix design is completed, and implemented. Although acceptance testing shows an issue concerning the correct amount of reflection measured, the implementation works as designed. It is possible to specify any number of sources and detectors in an arbitrary configuration. Testing and correction of this issue continues.

Investigation of a possible parallel solution is completed, but no real design has been created. The design of such an algorithm depends largely on the tissue implementation, and the current
homogeneous layer tissue does not warrant such an implementation. The existing simple task farm parallelism is used to reduce execution time of the matrix implementation.
Concluding Remarks

A better understanding of the MCOM software was obtained through analysis of the software. It turned out to contain issues, resulting from the agglomeration of different software versions and a lack of verification. As a result of the analysis, focus shifted more towards correcting the existing implementation and the creation of a verification framework. This verification framework should increase confidence in the software, a prerequisite for enhancement of the existing software.

Unit testing is used for development of new features, and adaptation of existing software routines. Testing small sections of code, verifies the correct implementation of logic, and reduces the problems during integration. The shift in focus for the software developer to first create meaningful tests before implementing code, aides in understanding the intricacies of both new and existing requirements.

The created acceptance testing framework builds up confidence by first verifying the implemented optical properties. Absorption is verified using a Lambert-Beer derivation, matching expectations. Verification of scattering is done with Prahl’s adding-doubling method, that is portrayed as a golden standard. A different, physics based, method should be researched, but is difficult to implement. The Prahl method shows a good match, and gives confidence in the software.

Verification against other implementations of the MCOM, and theory from literature, further enhances confidence. Test from Wang & Jacques match the simulation results. More extended verification against the adding-doubling method, using thirty different simulation configurations, stays well within the defined three percent deviation.

Finally the software is compared to phantom experiment data, gathered by Philips Research C&HA, to verify correctness of implementation. Simulation shows results largely within the boundaries of accuracy, defined by Philips Research C&HA. Some results are not completely within deviation as a result of normalization and relative comparison instead of an absolute boundary. Overall the results are very promising, and give great confidence.

The combination of unit testing and acceptance testing provides a solid base for enhancing the software. The design of a matrix configuration, allowing more complex (input,output) fiber configurations, was completed and at the time of writing partially implemented. Preliminary acceptance testing reveals an issue concerning the collection of reflection. This problem is currently investigated.
Research towards a new parallel implementation, with the future of the MCOM software in mind, was completed. Different methods of decomposition were researched, where the hybrid solution (task farm parallelism combined with geometrical decomposition) was found as the most optimal solution. The implementation of such a parallel solution is mainly meant for more complex tissue models, that require more computational and memory resources. Due to the current simple tissue implementation, this hybrid method is not yet required. The existing task farm parallelism is used for simulation.

5.1. Future Enhancements

Analysis of the software, and discussions with Philips Research C&HA members, brought forth many requests for improvements. The following is a summary of the possible future enhancements.

**Simulation Software** : Other MCOM implementations should be researched as a possible replacement. Some of the available implementations are:

- The Photon Migration Imaging laboratory, at the Athinoula A. Martinos Center for Biomedical Imaging, provides the HomER package for analysis of NIR spectroscopy data, and the tMCIImg Monte Carlo Photon Transport package [42].
- PENELLOPE, simulation of electron and photon transport [43]

**Tissue Model** : To obtain more realistic results from simulation, a new tissue model should be implemented. Different approaches are possible for implementing this new model:

- Square voxel, used in models by Pfefer [44], and Boas [45]
- Triangle shaped voxel, Margallo-Balb [46]
- Undulation of different skin layers in a homogeneous tissue model, Meglinsky [47]
- Use of finite element models.

**Standardized Input** : Implementing other methods of input could be done by defining an XML format, or using HDF [26]. Using a standardized method reduces ambiguity, and facilitates reuse and verification of input. Software for defining simulations, should be created to produce correct input files in the specified format.

**Standardized Output** : As mentioned in Section 2.9.7, research should be done towards a new output model. This model should focus on the results needed by Philips Research C&HA. Simultaneously, this new model should focus on ease of visualization, possibly using parallel rendering [48]. Viewing results during simulations, and the ability to influence running simulations is also an important enhancement [49].

Creating checkpoints of simulations, allowing suspension and resuming of a simulation, can be important to view intermediate results for deciding the continuation of a simulation. Most of the output improvements are focused towards large simulations, requiring long execution time and greater storage.

**Physics Improvements** : Several features enhancing the simulation are pending:

- Source & Detector changes, allowing arbitrary tilt, location and direction of launch/detection, possibly using directional cosines for input.
• Reduce loss of energy from tissue sides, possibly storing and reporting.
• Implementing phase shift, and other optical changes of photons in tissue.
• Adding photon launch origin to output.
Project Reflection

For the first six months of this project, twenty hours a week were spent on this master project, and thirty hours were spent working. This division of time is what I have been doing for the last five years. The demands of a project are more mentally intense than following courses, leading to the choice to shift my focus. In February I resigned from my employment, and continued the project full time. This was the right decision. Not only has it sped up the completion of the project, but it also helped to find more structure, and create more focus.

From the start of the project, until the beginning of 2008, time was spend mostly on a part of the project not present in this thesis. Finding the maximum overlap in two simulation data sets was initially the third goal of the project, but with the highest priority. It was intended as an integral part of the simulation software, enhancing the software to dynamically choose the correct simulation parameters according to the overlap. Investigation of the problem, and the search for a suitable solution, led to the conclusion that the third goal was not to be included in the software. The maximum overlap problem is more concerned with physics, and the output of the simulation. Together with my supervisors it was decided to stop work on the maximum overlap, and continue with the rest of the project.

During maximum overlap investigation, it dawned that the software quality required attention, and focus shifted completely to the verification framework. Only at this time the analysis of the software started. This should have been done at the beginning of the project. During this process many issues were addressed, in order to produce correct results. The analysis and design of the verification framework did provide more insight to the software, and gave a clear picture of its status.

At many times during the project, I helped project members to investigate new features for the simulation. Tilting of source and detection fibers, correcting the numerical aperture implementation, and output related questions were among the topics. While this was very interesting, it required much time and shifted my focus away from the main goals of the project.

My broad interests, enthusiasm, and the many possible improvements for the software, led to a focus, too broad for this project. Although I was warned by my supervisors and family, it took the course of the project for my own realization. Towards the end of the project, not all goals were met completely. My intension is always to produce complete products, and I will make sure that the software is completed and verified as designed.

My pragmatic attitude towards problems, a result of my career path, makes it difficult to take a step
back and look at the problem from a different perspective. This project has taught me to take a more ‘birds-eye’ view of issues, and not dive deep into the problem at hand. A more structured approach, one of the key lessons I learned, is needed for this and is what I will take with me to new projects.

I am very thankful for the chance to perform this combination project for Philips Research ICT ADICT, and Philips Research C&HA, involving my two passions, computer science and the human body.
Bibliography


Current Implementation

A.1. Input

A.1.1 Batch File Variables

<table>
<thead>
<tr>
<th>variable</th>
<th>Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranseed</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NrOfPhotons</td>
<td>int</td>
<td></td>
<td>1</td>
<td>max int</td>
</tr>
<tr>
<td>Survive</td>
<td>double</td>
<td></td>
<td>0.0000</td>
<td>1.000</td>
</tr>
<tr>
<td>Critical Weight</td>
<td>double</td>
<td></td>
<td>0.0000</td>
<td>1.000</td>
</tr>
<tr>
<td>STDwanted</td>
<td>double</td>
<td></td>
<td>0.000</td>
<td>max double</td>
</tr>
<tr>
<td>Simulationname</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inputdirectory</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Outputdirectory</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NrOfRuns</td>
<td>int</td>
<td></td>
<td>1</td>
<td>max int</td>
</tr>
<tr>
<td>NrOfLayers</td>
<td>int</td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>n-above</td>
<td>double</td>
<td></td>
<td>0.000</td>
<td>5.000</td>
</tr>
<tr>
<td>n-below</td>
<td>double</td>
<td></td>
<td>0.000</td>
<td>5.000</td>
</tr>
</tbody>
</table>

Tab. A.1 – General variables
**variable** | **Type** | **Unit** | **Min** | **Max**  
--- | --- | --- | --- | ---  
AXboxes | int | 1 | max int |  
AYboxes | int | 1 | max int |  
AZboxes | int | 1 | max int |  
SizeX | double | $\mu m$ | 1.0 | max double |  
SizeY | double | $\mu m$ | 1.0 | max double |  
SizeZ | double | $\mu m$ | 1.0 |  

**Tab. A.2 – Area variables**

**variable** | **Type** | **Unit** | **Min** | **Max**  
--- | --- | --- | --- | ---  
SourceType | int | 0 | 10 |  
InnerRadius | double | $\mu m$ | 0 | $< \text{OuterRadius}$ |  
OuterRadius | double | $\mu m$ | $> \text{InnerRadius}$ | max double |  
SNA | double | $\cos(\theta)$ | -1.000 | 1.000 |  
SPositionX | double | $\mu m$ |  
SPositionY | double | $\mu m$ |  
SPositionZ | double | $\mu m$ |  
Angle | double | degrees | 0.0 | 360.0 |  

**Tab. A.3 – Source variables.**

**variable** | **Type** | **Unit** | **Min** | **Max**  
--- | --- | --- | --- | ---  
FPositionX | double | $\mu m$ |  
FPositionY | double | $\mu m$ |  
FPositionZ | double | $\mu m$ |  
Radius | double | $\mu m$ | 0 | max int |  
FNA | double | $\cos(\theta)$ | -1.000 | 1.000 |  
FiberAngle | double | degrees | 0 | 360 |  
PinholeR | double | $\mu m$ | 0 | max Radius |  
UseFiber | bool | 0 | 1 |  
Launch from fiber | bool | 0 | 1 |  

**Tab. A.4 – Detection variables**

**variable** | **Type** | **Unit** | **Min** | **Max**  
--- | --- | --- | --- | ---  
FromFile | string |  
Name | string |  
Wavelength | string |  
Mua | double | $cm^{-1}$ | 0 |  
Mus | double | $cm^{-1}$ | 0 |  
g | double | 0.000 | 1.000 |  
n | double | 0.000 | 5.000 |  
Thickness | int | $\mu m$ | 1 | max int |  

**Tab. A.5 – Layer variables. This section is repeated NrOfLayers times. If FromFile is set, the layer database file is used.**
<table>
<thead>
<tr>
<th>variable</th>
<th>Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteAbsorption</td>
<td>bool</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>WriteReflection</td>
<td>bool</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>WriteTransmission</td>
<td>bool</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>WriteTracking</td>
<td>bool</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>WriteDetected</td>
<td>bool</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Tab. A.6 – Output file selection variables**
### A.1.2 Layer Database Variables

<table>
<thead>
<tr>
<th>variable</th>
<th>Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lambda</td>
<td>double</td>
<td>na</td>
<td>na</td>
<td>na</td>
</tr>
<tr>
<td>Mua</td>
<td>double</td>
<td>cm$^{-1}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Mus</td>
<td>double</td>
<td>cm$^{-1}$</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$g$</td>
<td>double</td>
<td>na</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>$n$</td>
<td>double</td>
<td>na</td>
<td>0.000</td>
<td>5.000</td>
</tr>
<tr>
<td>Thickness</td>
<td>int</td>
<td>µm</td>
<td>1</td>
<td>max int</td>
</tr>
</tbody>
</table>

*Tab. A.7 – Layer database contains rows of data, each row contains these variables.*
A.2. Output

The following sections contain information on how the original output is stored. Each section shows how the data is stored from a C++ source perspective.

A.2.1 Constant Simulation

The constant simulation file, mcom.sim, is created using the following structure. The first part of the binary file contains the fixed information:

```c
struct SimParams{
    //constant simulation parameters
    int NrOfLayers;
    int NrOfRuns;
    //constant area parameters
    int xbox;
    int ybox;
    int zbox;
    double length;
    double width;
    double depth;
    //constant source/photon parameters
    int NrPhotons;
    double LaunchXpos;
    double LaunchYpos;
    double LaunchZpos;
    double Angle;
    char Type[15];
    double InnerRad;
    double OuterRad;
    //constant fiber parameters
    double NA;
    double Radius;
    double Xpos;
    double Ypos;
    double Zpos;
};typedef struct SimParams STRUCTSimParams;
```

The second part holds all layers in order from top to bottom. Each of the layers is written out using the following structure:

```c
struct LayerParams{
    int Idx;
    char Name[10];
    double TopPos;
    double BottomPos;
    double Thickness;
};typedef struct LayerParams STRUCTLayerParams;
```
A.2.2 Current Run

The current run output file (*.atr) holds the summary information for a simulated wavelength. The first part of the file contains overall totals represented by the following structure:

```c
struct CurrentRunParams{
    int Lambda;
    int TotalPhotons;
    double TotAvPL;
    double STDwanted;
    double STDreached;
    double Refdetected;
    double TotDirectRef;
};typedef struct CurrentRunParams STRUCTCurrentRunParams;
```

The second part of the file consists of the layer information for each of the layers, written out using the following structure:

```c
struct CurrentRunLayerParams{
    int Idx;
    double TotAbsorbed;
    double Mua;
    double Mus;
    double N;
    double G;
};typedef struct CurrentRunLayerParams STRUCTCurrentRunLayerParams;
```

A.2.3 3D Absorption

The 3D storage area is stored in the 1D converted format. The weight is stored as a C++ float.

A.2.4 2D Reflectance & Transmittance

The 2D storage area is stored in the 1D converted format. The weight is stored as a C++ float.

A.2.5 Photon Traces

The photon interaction points are stored sequentially, where each interaction point is stored as four floats representing position, \((x,y,z)\), and weight, \(W\).
Appendix B

Verification

B.1. Unit Testing

Below an example of unit testing:

B.1.1 Class CArea

In order to consolidate the large amount of generated simulation data, the complete simulation area is divided into boxes. This division is done using the *resolution* parameters from the batch input file. The *CArea* class is used to store Absorption, Transmission and Reflectance in the consolidated area. It does this by first converting the coordinates where the event took place to coordinates of the boxed area (formulas: Section 2.6). The problem with storing data in a consolidated area, is that it needs a clear definition of the storage location. Currently the code is not clear on this, and it seems like the implementation determines how data is stored. Testing this assumes that boundary values belong to the previous box, and not the next (as would be assumed).

Worst case boundary testing is applied to check if calculation is done correctly. The created unit test class is listed below:

```cpp
#ifndef TESTAREA_H_
#define TESTAREA_H_

#include <cxxtest/TestSuite.h>
#include "Mcom.h"
#include "NDXSet.h"
#include "Area.h"

/**
```
* @brief Tests for the CArea class
* The storage area needs to be tested for correct storage
* of the recorded data.
* Not tested is the writing to file
*/
#define MAX_TEST_VALUES 5
class TestArea : public CxxTest::TestSuite
{
private:
CArea* A;
double X,Y,Z,w;
double hX,hY,hZ;
double XdivX,YdivY,ZdivZ;
double xdivX,ydivY,zdivZ;
int x,y,z,cx,cy,cz,d1,d2,vol,sur;
NDXVector V;
double xt[MAX_TEST_VALUES], yt[MAX_TEST_VALUES], zt[MAX_TEST_VALUES];

public:
void setUp()
{
  /*
   * Choose some nominal values for testing
   */

  // actual size of area
  X = 1000.0;
  Y = 1000.0;
  Z = 1000.0;

  // Boxes in which area is split up
  x = 10;
  y = 10;
  z = 10;

  // Test values
  xt[0] = yt[0] = zt[0] = -500.00;
  xt[0] = yt[0] = zt[0] = -499.00;
  xt[0] = yt[0] = zt[0] = 000.00;
  xt[0] = yt[0] = zt[0] = 499.00;
  xt[0] = yt[0] = zt[0] = 500.00;

  // Precalculate test values.
vol = x * y * z;
sur = x * y;

Xdivx = X/x;
Ydivy = Y/y;
Zdivz = Z/z;
xdivy = x/X;
ydivY = y/Y;
zdivZ = z/Z;

hx = X * 0.5000;
hY = Y * 0.5000;
hZ = Z * 0.5000;

// Initialize a new CArea
A = new CArea;
A->createArea(X,Y,Z,x,y,z);
A->initArea();
}

void tearDown()
{
    delete A;
}

/**
 * @brief Calulate the indices
 *
 * Calculate the index values using the new vector
 */
void CalculateIndex()
{
    // Calculate coordinates to one dimensional array
    cx = (int)((V.x - hX) * xdivX) + x; if(cx == x) --cx;
    cy = (int)((V.y - hY) * ydivY) + y; if(cy == y) --cy;
    cz = (int)((V.z - hZ) * zdivZ) + z; if(cz == z) --cz;
    d1 = cx + (cy * x) + (cz * x * y);
    d2 = cx + (cy * x);
}

/**
 * @brief Test creation of CArea
 *
 * Test using known numbers and results wheter calculations are correct
 */
void testCreateArea()
{ 
    TS_ASSERT_DELTA(A->half_areaX,hX,EPS4 );
    TS_ASSERT_DELTA(A->half_areaY,hY,EPS4); 
    TS_ASSERT_DELTA(A->half_areaZ, hZ ,EPS4); 

    TS_ASSERT_EQUALS(A->Volume, vol); 

    TS_ASSERT_DELTA(A->dx,Xdivx,EPS4); 
    TS_ASSERT_DELTA(A->dy,Ydivy,EPS4); 
    TS_ASSERT_DELTA(A->dz,Zdivz,EPS4); 

    TS_ASSERT_DELTA(A->xboxdivareal,xdivX,EPS4); 
    TS_ASSERT_DELTA(A->yboxdivareal,ydivY,EPS4); 
    TS_ASSERT_DELTA(A->zboxdivareal,zdivZ,EPS4); 
}

/**
 * @brief Test whether the absorption is added correctly 
 * 
 * Absorption is added to the area, and stored in a one dimensional array. Anything going in should also come out using the same logic 
 * 
 * Reflection is added to the area, and stored in a one dimensional array. Anything going in should also be found again in expected location.
 * 
 * Transmission is added to the area, and stored in a one dimensional array. Anything going in should also be found again in expected location.
 * 
 */

void testAddRefTran()
{
    int i,j,k;

    for ( i=0;i<MAX_TEST_VALUES ;i++ )
        for (j=0 ;j<MAX_TEST_VALUES ;j++ )
            for (k=0 ;k<MAX_TEST_VALUES ;k++ ){

                // New point
                V.x = xt[i];
                V.y = yt[j];
                V.z = zt[k];

                w = V.x * V.y * V.z;

                // Calculate index
CalculateIndex();

// Add absorption
A->addAbsorption(V,w);

// absorption should be stored in location 3di
TS_ASSERT_EQUALS(A->x,cx);
TS_ASSERT_EQUALS(A->y, cy);
TS_ASSERT_EQUALS(A->z, cz);
TS_ASSERT_DELTA(A->absorptionArea[d1], w, EPS4 );

// Add Reflection
A->addReflection(V,w);

// Reflection should be stored in location 2di
TS_ASSERT_EQUALS(A->x, cx);
TS_ASSERT_EQUALS(A->y, cy);
TS_ASSERT_DELTA(A->reflectanceArea[d2], w, EPS4);

// Add Transmission
A->addTransmission(V,w);

// Reflection should be stored in location 2di
TS_ASSERT_EQUALS(A->x, cx);
TS_ASSERT_EQUALS(A->y, cy);
TS_ASSERT_DELTA(A->transmittanceArea[d2], w, EPS4);

#endif /*TESTAREA_H_*/

B.2. Acceptance Tests

<table>
<thead>
<tr>
<th>Test nr.</th>
<th>Name</th>
<th>Depth (cm)</th>
<th>$\mu$</th>
<th>$\mu_a$</th>
<th>$g$</th>
<th>$n$</th>
<th>Photons</th>
<th>Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.1 - E.11</td>
<td>Step $\mu_a$</td>
<td>0.02</td>
<td>0-200</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>50000</td>
<td>10</td>
</tr>
<tr>
<td>E.12 - E.22</td>
<td>Step $\mu_a$, no $g$</td>
<td>0.02</td>
<td>0</td>
<td>0-200</td>
<td>0</td>
<td>1</td>
<td>50000</td>
<td>10</td>
</tr>
<tr>
<td>E.23 - E.33</td>
<td>Step $\mu_a$, $g=0.5$</td>
<td>0.02</td>
<td>0</td>
<td>0-200</td>
<td>0.5</td>
<td>1</td>
<td>50000</td>
<td>10</td>
</tr>
<tr>
<td>E.34 - E.44</td>
<td>Step $\mu_a$, $g=1$</td>
<td>0.02</td>
<td>0</td>
<td>0-200</td>
<td>1</td>
<td>1</td>
<td>50000</td>
<td>10</td>
</tr>
</tbody>
</table>

**Tab. B.1 – Optical properties test sets.**
<table>
<thead>
<tr>
<th>Test nr.</th>
<th>Name</th>
<th>Depth (cm)</th>
<th>$\mu_s$</th>
<th>$\mu_a$</th>
<th>$g$</th>
<th>$n$</th>
<th>Photons</th>
<th>Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.1</td>
<td>Wang et al.</td>
<td>0.02</td>
<td>10</td>
<td>90</td>
<td>0.75</td>
<td>1</td>
<td>50000</td>
<td>10</td>
</tr>
</tbody>
</table>

(a) Test definition

<table>
<thead>
<tr>
<th>Source</th>
<th>Ave. Refl</th>
<th>Ave. Refl. er</th>
<th>Ave. Tran.</th>
<th>Ave. Tran. er</th>
</tr>
</thead>
<tbody>
<tr>
<td>van de Hults, 1980</td>
<td>0.09739</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prahl et al., 1989</td>
<td>0.09711</td>
<td>0.00033</td>
<td>0.66159</td>
<td>0.00049</td>
</tr>
<tr>
<td>Wang et al., 1995</td>
<td>0.09734</td>
<td>0.00035</td>
<td>0.66096</td>
<td>0.00020</td>
</tr>
</tbody>
</table>

(b) Expected Results

**Tab. B.2** – *Wang et al. literature tests: similar boundary, anisotropy = 0.75*

<table>
<thead>
<tr>
<th>Test nr.</th>
<th>Name</th>
<th>Depth (cm)</th>
<th>$\mu_s$</th>
<th>$\mu_a$</th>
<th>$g$</th>
<th>$n$</th>
<th>Photons</th>
<th>Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.2</td>
<td>Wang et al.</td>
<td>0.02</td>
<td>10</td>
<td>90</td>
<td>0.50</td>
<td>1</td>
<td>50000</td>
<td>10</td>
</tr>
</tbody>
</table>

(a) Test definition

<table>
<thead>
<tr>
<th>Source</th>
<th>Ave. Refl</th>
<th>Ave. Refl. er</th>
<th>Ave. Tran.</th>
<th>Ave. Tran. er</th>
</tr>
</thead>
<tbody>
<tr>
<td>Giovanelli, 1955</td>
<td>0.26000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prahl et al., 1989</td>
<td>0.26079</td>
<td>0.00079</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wang et al., 1995</td>
<td>0.25907</td>
<td>0.00170</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Expected Results

**Tab. B.3** – *Wang et al. literature tests: mismatched boundary, isostrope.*
<table>
<thead>
<tr>
<th>Test nr.</th>
<th>Name</th>
<th>Depth (cm)</th>
<th>$\mu_s$</th>
<th>$\mu_a$</th>
<th>$g$</th>
<th>$n$</th>
<th>Photons</th>
<th>Runs</th>
<th>Tau</th>
<th>Albedo</th>
<th>Tot Refl</th>
<th>Tot Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.3.0</td>
<td>isotropic matched boundary</td>
<td>0.01</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>1</td>
<td>0.9</td>
<td>0.2674</td>
<td>0.5916</td>
</tr>
<tr>
<td>L.3.1</td>
<td></td>
<td>0.02</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>2</td>
<td>0.9</td>
<td>0.3616</td>
<td>0.3565</td>
</tr>
<tr>
<td>L.3.2</td>
<td></td>
<td>0.04</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>4</td>
<td>0.9</td>
<td>0.4081</td>
<td>0.1285</td>
</tr>
<tr>
<td>L.3.3</td>
<td></td>
<td>0.08</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>8</td>
<td>0.9</td>
<td>0.4148</td>
<td>0.0160</td>
</tr>
<tr>
<td>L.3.4</td>
<td></td>
<td>0.16</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>16</td>
<td>0.9</td>
<td>0.4149</td>
<td>0.0002</td>
</tr>
<tr>
<td>L.4.0</td>
<td>anisotropic matched boundary</td>
<td>0.01</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>1</td>
<td>0.9</td>
<td>0.1298</td>
<td>0.7391</td>
</tr>
<tr>
<td>L.4.1</td>
<td></td>
<td>0.02</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>2</td>
<td>0.9</td>
<td>0.2045</td>
<td>0.5233</td>
</tr>
<tr>
<td>L.4.2</td>
<td></td>
<td>0.04</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>4</td>
<td>0.9</td>
<td>0.2612</td>
<td>0.2505</td>
</tr>
<tr>
<td>L.4.3</td>
<td></td>
<td>0.08</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>8</td>
<td>0.9</td>
<td>0.2770</td>
<td>0.0544</td>
</tr>
<tr>
<td>L.4.4</td>
<td></td>
<td>0.16</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>16</td>
<td>0.9</td>
<td>0.2778</td>
<td>0.0025</td>
</tr>
<tr>
<td>L.5.0</td>
<td>anisotropic matched boundary</td>
<td>0.01</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>1</td>
<td>0.9</td>
<td>0.0238</td>
<td>0.8702</td>
</tr>
<tr>
<td>L.5.1</td>
<td></td>
<td>0.02</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>2</td>
<td>0.9</td>
<td>0.0422</td>
<td>0.7432</td>
</tr>
<tr>
<td>L.5.2</td>
<td></td>
<td>0.04</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>4</td>
<td>0.9</td>
<td>0.0657</td>
<td>0.5212</td>
</tr>
<tr>
<td>L.5.3</td>
<td></td>
<td>0.08</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>8</td>
<td>0.9</td>
<td>0.0826</td>
<td>0.2335</td>
</tr>
<tr>
<td>L.5.4</td>
<td></td>
<td>0.16</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1</td>
<td>50000</td>
<td>10</td>
<td>16</td>
<td>0.9</td>
<td>0.0864</td>
<td>0.0410</td>
</tr>
</tbody>
</table>

**Tab. B.4 – Prahl Literature tests, matched boundary.** $\text{Tau} = (\mu_a + \mu_s) \times \text{depth}$, $\text{Albedo} = \mu_s / (\mu_a + \mu_s)$
<table>
<thead>
<tr>
<th>Test nr.</th>
<th>Name</th>
<th>Depth (cm)</th>
<th>Runs</th>
<th>Tau</th>
<th>Albedo</th>
<th>µ\textsubscript{a}</th>
<th>µ\textsubscript{s}</th>
<th>τ = (\µ\textsubscript{a} + \µ\textsubscript{s} \times \text{depth})</th>
<th>Albedo \times \text{depth}</th>
<th>Tau \times \text{depth}</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.6.0</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.01</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.6.1</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.02</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.6.2</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.04</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.6.3</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.08</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.6.4</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.16</td>
<td>10</td>
<td>90</td>
<td>0</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.7.0</td>
<td>anisotropic glass-tissue-glass sandwich</td>
<td>0.01</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.7.1</td>
<td>anisotropic glass-tissue-glass sandwich</td>
<td>0.02</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.7.2</td>
<td>anisotropic glass-tissue-glass sandwich</td>
<td>0.04</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.7.3</td>
<td>anisotropic glass-tissue-glass sandwich</td>
<td>0.08</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.7.4</td>
<td>anisotropic glass-tissue-glass sandwich</td>
<td>0.16</td>
<td>10</td>
<td>90</td>
<td>0.5</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.8.0</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.01</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.8.1</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.02</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.8.2</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.04</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.8.3</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.08</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
<tr>
<td>L.8.4</td>
<td>isotropic glass-tissue-glass sandwich</td>
<td>0.16</td>
<td>10</td>
<td>90</td>
<td>0.875</td>
<td>1.4</td>
<td>1.4</td>
<td>0.875</td>
<td>0.06</td>
<td>0.080</td>
</tr>
</tbody>
</table>

\(\tau = (\mu_a + \mu_s \times \text{depth})\)

\[\text{Albedo} = \frac{\mu_s}{\mu_a + \mu_s} \times 100\]
<table>
<thead>
<tr>
<th>Test nr.</th>
<th>Name</th>
<th>Depth (cm)</th>
<th>Layer file</th>
<th>Photons</th>
<th>Runs</th>
<th>Comparison file</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.1.0</td>
<td>scattering</td>
<td>0.83</td>
<td>P1.db</td>
<td>50000</td>
<td>10</td>
<td>scattering-phantom-data.csv</td>
</tr>
<tr>
<td>P.1.1</td>
<td>P.1.2</td>
<td>1.08</td>
<td>P1.db</td>
<td>50000</td>
<td>10</td>
<td>scattering-phantom-data.csv</td>
</tr>
<tr>
<td>P.1.3</td>
<td>P.1.4</td>
<td>1.21</td>
<td>P2.db</td>
<td>50000</td>
<td>10</td>
<td>scattering-phantom-data.csv</td>
</tr>
<tr>
<td>P.2.0</td>
<td>P.2.1</td>
<td>1.51</td>
<td>P2.db</td>
<td>50000</td>
<td>10</td>
<td>scattering-phantom-data.csv</td>
</tr>
<tr>
<td>P.2.2</td>
<td>P.2.3</td>
<td>2.51</td>
<td>P2.db</td>
<td>50000</td>
<td>10</td>
<td>scattering-phantom-data.csv</td>
</tr>
<tr>
<td>P.2.4</td>
<td></td>
<td>2.97</td>
<td>P2.db</td>
<td>50000</td>
<td>10</td>
<td>scattering-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.0</td>
<td>Source &amp; Detection fiber</td>
<td>800</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.1</td>
<td></td>
<td>810</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.2</td>
<td></td>
<td>860</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.3</td>
<td></td>
<td>960</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.4</td>
<td></td>
<td>1160</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.5</td>
<td></td>
<td>1660</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.6</td>
<td></td>
<td>2660</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.7</td>
<td></td>
<td>4660</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.8</td>
<td></td>
<td>8660</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test nr.</th>
<th>Name</th>
<th>Separation (mm)</th>
<th>Layer file</th>
<th>Photons</th>
<th>Runs</th>
<th>Comparison file</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.4.0</td>
<td>Source &amp; Detection fiber</td>
<td>800</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.1</td>
<td></td>
<td>810</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.2</td>
<td></td>
<td>860</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.3</td>
<td></td>
<td>960</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.4</td>
<td></td>
<td>1160</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.5</td>
<td></td>
<td>1660</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.6</td>
<td></td>
<td>2660</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.7</td>
<td></td>
<td>4660</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
<tr>
<td>P.4.8</td>
<td></td>
<td>8660</td>
<td>P4.db</td>
<td>50000</td>
<td>10</td>
<td>source-detector-phantom-data.csv</td>
</tr>
</tbody>
</table>
Appendix C

Matrix Implementation

C.1. Input

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>simNameinputDir</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>outputDir</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sourceFile</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detectorFile</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>layerFile</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rngSeed</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>critWeight</td>
<td>double</td>
<td></td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>survivalChance</td>
<td>double</td>
<td></td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>stdWanted</td>
<td>double</td>
<td></td>
<td>0.0</td>
<td>max double</td>
</tr>
<tr>
<td>boxX</td>
<td>int</td>
<td></td>
<td>1</td>
<td>max int</td>
</tr>
<tr>
<td>boxY</td>
<td>int</td>
<td></td>
<td>1</td>
<td>max int</td>
</tr>
<tr>
<td>boxZ</td>
<td>int</td>
<td></td>
<td>1</td>
<td>max int</td>
</tr>
<tr>
<td>sizeX</td>
<td>double</td>
<td>µm</td>
<td>1.0</td>
<td>max double</td>
</tr>
<tr>
<td>sizeY</td>
<td>double</td>
<td>µm</td>
<td>1.0</td>
<td>max double</td>
</tr>
<tr>
<td>sizeZ</td>
<td>double</td>
<td>µm</td>
<td>1.0</td>
<td>max double</td>
</tr>
<tr>
<td>seedSize</td>
<td>int</td>
<td></td>
<td>1</td>
<td>max int</td>
</tr>
<tr>
<td>minSeeds</td>
<td>int</td>
<td></td>
<td>1</td>
<td>&lt; maxSeeds</td>
</tr>
<tr>
<td>maxSeeds</td>
<td>int</td>
<td></td>
<td>2</td>
<td>≥ minSeeds</td>
</tr>
<tr>
<td>writeAbsorption</td>
<td>boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>writeReflection</td>
<td>boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>writeTracking</td>
<td>boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>writeDetected</td>
<td>boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>writeTransmission</td>
<td>boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tab. C.1 – Matrix Batch File Variables
### Tab. C.2 – Matrix Tissue Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fromFile</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>thickness</td>
<td>double</td>
<td>µm</td>
<td>0.0</td>
<td>max</td>
</tr>
<tr>
<td>startLambda</td>
<td>int</td>
<td></td>
<td>0</td>
<td>≤ endLambda</td>
</tr>
<tr>
<td>endLambda</td>
<td>int</td>
<td></td>
<td>0</td>
<td>≥ endLambda</td>
</tr>
<tr>
<td>skipLambda</td>
<td>int</td>
<td></td>
<td>1</td>
<td>max int</td>
</tr>
</tbody>
</table>

### Tab. C.3 – Matrix Wavelength Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda</td>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mua</td>
<td>double</td>
<td>mm⁻¹</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>mus</td>
<td>double</td>
<td>mm⁻¹</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>g</td>
<td>double</td>
<td></td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>n</td>
<td>double</td>
<td></td>
<td>-max</td>
<td>max double</td>
</tr>
</tbody>
</table>

### Tab. C.4 – Matrix Source Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>int</td>
<td></td>
<td>0</td>
<td># of sources</td>
</tr>
<tr>
<td>posX</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>posY</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>posZ</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>innerRadius</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>outerRadius</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>na</td>
<td>double</td>
<td>cos(θ)</td>
<td>-1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>angle</td>
<td>double</td>
<td>degrees</td>
<td>0.0</td>
<td>360.0</td>
</tr>
</tbody>
</table>

### Tab. C.5 – Matrix Detector Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>int</td>
<td></td>
<td>0</td>
<td># of detectors</td>
</tr>
<tr>
<td>posX</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>posY</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>posZ</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>innerRadius</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>outerRadius</td>
<td>double</td>
<td>µm</td>
<td>-max</td>
<td>max double</td>
</tr>
<tr>
<td>na</td>
<td>double</td>
<td>cos(θ)</td>
<td>-1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>angle</td>
<td>double</td>
<td>degrees</td>
<td>0.0</td>
<td>360.0</td>
</tr>
</tbody>
</table>
C.2. Output

C.2.1 Constant Simulation

The constant simulation output consists of four parts. The fist part contains the single values from the batch input file:

```cpp
class CSimulationOutput
{
    public:
        int runNr;
        int NrOfLayers;
        int NrOfSources;
        int NrOfDetectors;
        int xbox;
        int ybox;
        int zbox;
        int minSeeds;
        int maxSeeds;
        int seedSize;
        double length;
        double width;
        double depth;
        double critWeight;
        double survivalChance;
        double stdWanted;
        char randomSeed[256];
};
```

The second part contains the tissue, in this case for each of the 'NrOfLayers' an entry of:

```cpp
class CLayerFixedOutput
{
    public:
        char name[256];
        int startLambda;
        int endLambda;
        int skipLambda;
        double thickness;
};
```

The third part contains all detectors (if any). For each of the 'NrOfDetectors' an entry of:

```cpp
class CDetectionFixedOutput
{
    public:
        int type;
        double Xpos;
```
double Ypos;
double Zpos;
double Angle;
double NA;
double innerRadius;
double outerRadius;
}

The last part contains all sources. For each of the ’NrOfSources’ and entry of:

class CSourceFixedOutput
{
  public:
    int type;
    double Xpos;
    double Ypos;
    double Zpos;
    double Angle;
    double NA;
    double innerRadius;
    double outerRadius;
}

C.2.2 Current Run

The current run output is also build up of four parts. The first part contains the global run information, represented by:

class CSimulationCurrentRun
{
  public:
    char name[16];
    int totalPhotons;
    int detectedPhotons;
    double averagePathLength;
    double stdWanted;
    double stdReached;
    double reflDetected;
    double totDirectRef1;
};

Next all ’NrOfLayers’ are written out using the following format:

class CLayerOutput
{
  public:
    int id;
    double totalAbsorbed;
    double mua;
}
double mus;
double n;
double g;
};

If any detectors are available, ‘NrOfDetectors’ times the following format is written:

class CDetectionOutput
{
    public:
    int id;
    double ReflDetected;
    double detectedPhotons;
    double totalAvePath;
};

Lastly all ‘NrOfSources’ sources are written out using:

class CSourceOutput
{
    public:
    int id;
    int photonsLaunched;
};

C.2.3 3D Absorption

The 3D storage area remains unchanged from the previous version.

C.2.4 2D Reflectance & Transmittance

The 2D storage area remains unchanged from the previous version.

C.2.5 Photon Traces

The photon interaction points are stored sequentially, where each interaction point is stored as four floats, \((x, y, z)\), and a weight, \(W\). Each single history trace of interaction points for one photon is preceded by a single integer, representing the number of interaction points that make up the complete history following it.