SIMULATION OF OPTICAL AND THERMAL RESPONSES OF LASER IRRADIATED TISSUE

by

Bahar Kurt

B.S. in Physics Engineering, ITU, 2005

Submitted to the Institute of Biomedical Engineering
in partial fulfillment of the requirements
for the degree of
Master of Science
in
Biomedical Engineering

Boğaziçi University
2009

ACKNOWLEDGMENTS

I would like to thank my advisor Assoc. Prof. Dr. Murat Gülsoy for his help during my thesis project. His technical support and advice help me to develop this study.

I would like to express my gratitude to TUBITAK for supporting me financially during my graduate education.

I also want to thank my family and fiance for their patience and moral support. Also special thanks to Gökçen Karasu who gave me clues about software development.

ABSTRACT

SIMULATION OF OPTICAL AND THERMAL RESPONSES OF LASER IRRADIATED TISSUE

A Java Simulation Application was developed to simulate optical and thermal response of laser irradiated tissue by using Monte Carlo and Finite Difference Methods. For light propagation Monte Carlo was preferred according to its high accuracy because a number of photon packets can be launched to increase efficiency. Also Finite Difference Method was used for heat diffusion because of its simplicity.

The results taken from Monte Carlo Model are used as a source term in heat conduction equation. Thus, in small time steps by doing iterations of Finite Difference Method thermal gradients are measured inside a tissue. At 1064 nm fluence and thermal contours of brain and liver tissues are obtained with same laser parameters. Also at different wavelengths, 1064 nm and 630 nm, fluence and thermal contours of liver tissue are calculated in the same conditions. Moreover, fluence and thermal contours of liver tissue are measured for different power densities and exposure times. The thermal gradients of laser irradiated tissue were plotted with 2D colored surface plotting.

Keywords: Monte Carlo Model, Finite Difference Method, light propagation, heat diffusion, Java.

ÖZET

LASER IŞINLANAN DOKUNUN OPTİK VE TERMAL TEPKİSİNİN SİMULASYONU

Laser ışınlanan dokunun optik ve termal tepkisini simule etmek için Monte Carlo ve Sonlu Farklar Yöntemlerini kullanarak bir Java Simulasyon Uygulaması geliştirilmiştir. Işık dağılımı için doğruya en yakın neticeyi sağladığından Monte Carlo tercih edilmiştir çünkü etkinliğini artırmak için çok sayıda foton paketi kullanılabilmektedir. Aynı zamanda sıcaklık dağılımı için sadeliğinden dolayı da Sonlu Farklar Yöntemi kullanılmıştır.

Monte Carlo modelinden elde edilen sonuçlar ısı iletim denklemindeki kaynak terimi olarak kullanılmıştır. Böylece Sonlu Farklar yönteminin küçük zaman aralıklarında iterasyonu yapılarak doku içersindeki sıcaklık değişimleri hesaplanmıştır. Sonuçlar farklı doku tiplerine ve dalgaboylarına göre karşılaştırılmıştır. 1064 nm'de beyin ve karaciğer dokularının aynı laser parametreleri kullanılarak fluence ve sıcaklık eğrileri elde edilmiştir. Aynı zamanda karaciğer dokusunun aynı koşullarda farklı dalgaboylarındaki, 1064 nm ve 630 nm, fluence ve sıcaklık eğrileri hesaplanmıştır. Laser ışınlanan dokunun sıcaklık değişimleri 2D yüzey grafikleri ile çizdirilmiştir.

Anahtar Sözcükler: Monte Carlo Modeli, Sonlu Farklar Yöntemi, ışık dağılımı, sıcaklık dağılımı, Java.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

θ_r	reflection angle
θ_i	incident angle
θ_t	refraction angle
v	speed of light in medium
c	speed of light in vacuum
n	refractive index
I	intensity
μ_a	absorption coefficient
μ_s	scattering coefficient
g	anisotropy factor
p	probability function
dw	solid angle
$\mu_s^{'}$	transport scatter coefficient
$\mu_t^{'}$	total transport attenuation coefficient
c_h	thermal capacity
Q	heat
c	specific heat capacity
m	mass
ΔT	derivative of temperature
T	temperature
k	thermal conductivity
ρ	density

W	weight
R_{sp}	specular reflectance
ξ	random variable
s	step size
x, y, z	cartesian coordinates
ψ	azimuthal angle
R	internal reflectance
R_d	diffuse reflectance
T_d	diffuse transmittance
A	absorption density
Ω	solid angle
Δa	annular area
Φ	fluence rate
G	green function
S	source term
B	Bessel function
R	radius
P	power
V	volume
$\dot{q}$	heat generation rate
h_∞	convective heat transfer coefficient
t	time

1. INTRODUCTION

1.1 Motivation

Lasers have been widely used in medicine for treatment. Main problem in laser applications is to estimate the right dose specific to the target tissue. Choosing the wavelength, pulse width, pulse shape, beam profile, power density depend on the predosimetry estimations [1]. For photochemical applications, modeling light distribution is crucial, on the other hand, for photothermal applications estimating temperature distribution is essential in order to determine the extent of thermal changes such as coagulation and hyperthermia.

1.2 Objective

The aim of the study is to develop a multipurpose Java application which has the capability of estimating the optical (fluence distribution) and thermal response (temperature gradient) of a biological tissue after laser exposure by entering laser and tissue properties as an input parameters.

1.3 Background

Optical properties, thermal properties, laser-tissue interactions and numerical solutions are explained in this section to understand basic principles of the study. Optical and thermal properties should be expressed to apprehend their effects to laser irradiated tissue. Laser-tissue interaction mechanisms are briefly mentioned for distinguishing laser effects to tissue according to different laser exposure time and power densities. Numerical solutions are described to find out advantageous of Monte Carlo and Finite Difference Method.

1.3.1 Optical Properties

When light interacts with matter, light distribution can be affected by some physical phenomena:

1. Reflection
2. Absorption
3. Scattering

These phenomena are related to the type of medium and wavelength. The index of refraction, absorption and scattering coefficients can be defined with the wavelength of the light. The reflectivity of the medium is expressed with index of refraction. For laser applications these optical properties are essential to determine the effects of light exposed to target tissue [2].

1.3.1.1 Reflection. When light comes to an interface of two different media, it can either reflects or refracts [2-4]. The reflection angle, θ_r, equals to the incident angle θ_i and θ_t represents the angle of refraction as shown in the Figure 1.1.

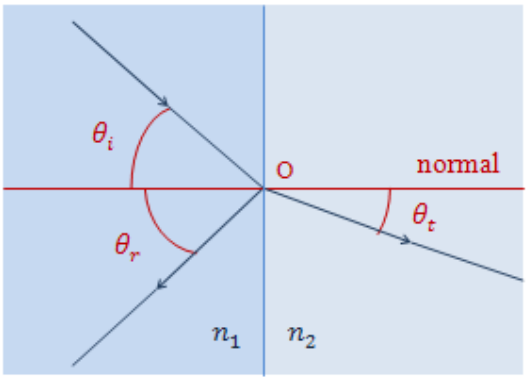


Figure 1.1 Reflection and refraction of light [2].

If surface region is smooth the reflection is called specular reflection and the angles of incident and reflection equal to each other. If not, then diffuse reflection occurs and due

to the irregularities of the surface the light rays reflects in many different directions which is illustrated in Figure 1.2 [5].

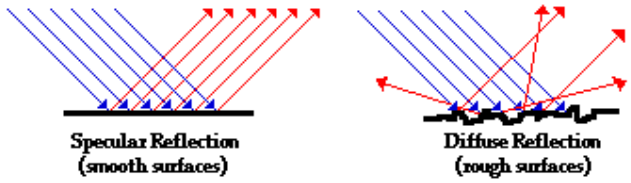


Figure 1.2 Specular and diffuse reflection [5].

Refraction occurs when indices of refraction of the two sides of reflecting surface are different and is defined by the Snell's law that is given below,

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{v_1}{v_2} \tag{1.1}$$

where v_1 and v_2 are the speed of light in the left side and right side medium, respectively. Index of refraction equals to c/v where c and v is the speed of light in vacuum and in medium, respectively. So Eq. 1.1 becomes,

$$n_1 \sin\theta_i = n_2 \sin\theta_t . \tag{1.2}$$

If light travels from a medium with higher refractive index to the medium with lower one, it will be refracted at 90°. Incident angle is called critical angle when this condition occurs. If light comes to the reflecting surface with angle greater than the critical angle then it doesn't pass into the second medium and totally reflected back to the same medium and this is called total internal reflection [2-4].

1.3.1.2 Absorption. The intensity of incident electromagnetic radiation is attenuated when light goes through the medium by the rule of energy conservation and the radiant energy is converted into the heat motion or molecular vibrations of the absorbing medium.

The several factors such as the electronic and nuclear constitution of the atoms and molecules of a medium, the wavelength of the light, the thickness of the irradiated medium, and the variables which determine the state of the medium like temperature and the concentration of the absorbing agent determine the cabability of the medium to absorb light [2-4].

When light passes through the medium with intensity I, the absorption coefficient, μ_a, can be expressed as below,

$$dI = -\mu_a I dx \tag{1.3}$$

where dI is the differential change of intensity of the light beam. Attenuation of light intensity is shown in Figure 1.3. If it is integrated over the distance x, Eq. 1.3 becomes,

$$I = I_0 \exp(-\mu_a x)\,. \tag{1.4}$$

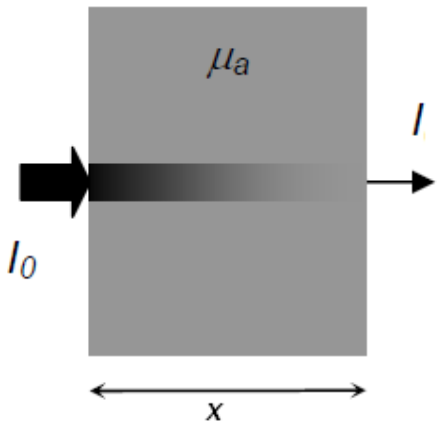


Figure 1.3 Attenuation of light through a non-scattering medium [3].

The tissue compounds which absorb light in the spectral region are called chromophores. Absorption spectra of biological tissues are given in Figure 1.4.

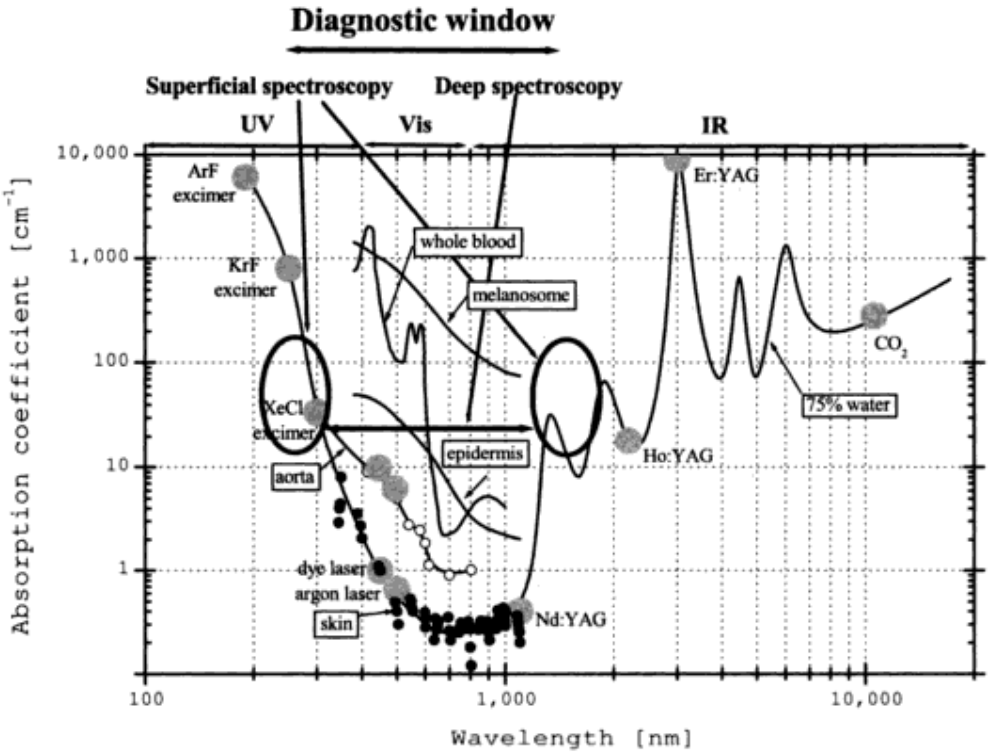


Figure 1.4 Absorption spectra of biological tissues [6].

1.3.1.3 Scattering. In biological medium scattering occurs at microscopic boundaries such as organelles and cell membranes, if their refractive indices are different. Attenuation of light through scattering medium is shown in Figure 1.5. The scattering coefficient, μ_s, can be expressed as,

$$I = I_0 \exp(-\mu_s x)\,. \tag{1.5}$$

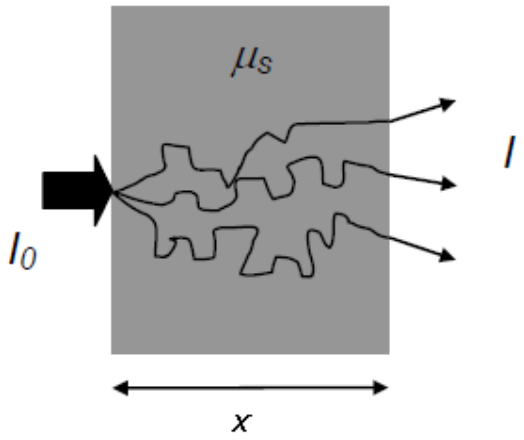


Figure 1.5 Attenuation of light through scattering medium [3].

There are two types of scattering; elastic and inelastic. For elastic scattering energies of the scattered photons is not altered. Rayleigh scattering is an example of elastic scattering and strongly depends on the wavelength. Mie scattering is another type of scattering which has lower dependence on wavelength. The scattering in any tissue is described by both Rayleigh and Mie Scattering because in most tissues forward scattering is dominant and it is explained by Mie scattering. Also strong wavelength dependence is explained by Rayleigh scattering [2, 7, 8]. A probability function $p(\theta)$ is defined to determine scattering condition. Anisotropic factor is expressed as,

$$g = \frac{\int_{4\pi} p(\theta)\cos\theta dw}{\int_{4\pi} p(\theta) dw} \tag{1.6}$$

where dw is the differential change of solid angle. When anisotropy factor g=0, g=1 and g=-1, it defines isotropic, forward and backward scattering, respectively.

Henvey Greenstein scattering phase function is one of the analytical expression for probability function $p(\theta)$ [9] which is given by,

$$p(\theta) = \frac{1-g^2}{(1+g^2-2g\cos\theta)^{3/2}}. \tag{1.7}$$

In random walk the transport scattering coefficient, μ_s', is defined by scattering coefficient μ_s and anisotropy factor g,

$$\mu_s' = \mu_s(1-g). \tag{1.8}$$

Total transport attenuation coefficient can be defined as,

$$\mu_t' = \mu_s' + \mu_a = \mu_s(1-g) + \mu_a. \tag{1.9}$$

1.3.2 Thermal Properties

For understanding the heat transfer inside a biological tissue, some definitions are needed to be given such as thermal capacity, specific heat capacity and thermal conductivity.

1.3.2.1 Thermal Capacity. Thermal capacity, c_h, of a substance is defined as the amount of heat required for one degree temperature increase and is given by,

$$c_h = \frac{Q}{\Delta T} \tag{1.10}$$

where Q is heat and ΔT is the change in temperature.

1.3.2.2 Specific Heat Capacity. Specific heat capacity, c, is described by the amount of heat required for one degree temperature increase in specified mass and can be expressed as,

$$c = \frac{Q}{m\Delta T} \tag{1.11}$$

where Q is heat, m is the mass of material and ΔT is the change in temperature.

Water content of the tissue affects the heat capacity linearly as below [2],

$$c = 4190(0.37 + 0.63W_c) \tag{1.12}$$

where W_c is water content.

1.3.2.3 Thermal Conductivity. Thermal conductivity represents the capability of material to conduct or transmit heat and given by,

$$k = \rho c \alpha \tag{1.13}$$

where ρ is density of the tissue and α is thermal diffusivity of the tissue. Thermal conductivity of the tissue changes according to water content and it is defined as [2],

$$k = 0.054 + 0.573W_c . \tag{1.14}$$

1.3.3 Laser-Tissue Interactions

The optical and thermal responses of laser irradiated tissue are related to laser source parameters. In laser therapeutic applications power, irradiation time and spot size are needed for estimation. Irradiance which is a function of power and spot size is the most significant parameter. Low and high irradiance can damage tissues in separate ways. Lasers with different wavelengths affect tissue in different manner due to the change of optical parameters of tissues depend on the various wavelengths [10]. There are four main laser interactions mechanisms with tissue: Photochemical, Thermal, Photoablation, Electromechanical.

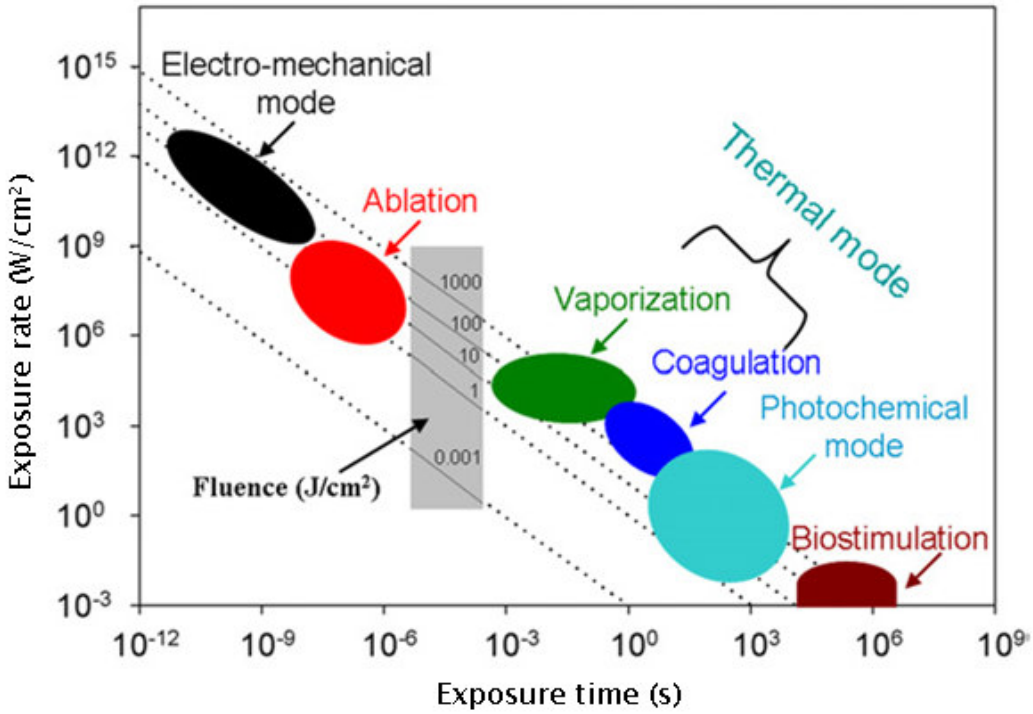


Figure 1.6 Laser-tissue interaction mechanisms [8].

1.3.3.1 Photochemical. In photochemical interactions tissues are affected chemically by laser exposure with low power densities. There are two applications for photochemical interactions: Photodynamic therapy (PDT) and Biostimulation. PDT used photosensitizers which take place in the lesions and are activated by the light at low power densities. After absorbing energy from light, photosensitizer produce highly reactive oxygen species by transfering energy without heat production. And this causes irreversible damage to the lesions and destroy them [2, 4, 8].

1.3.3.2 Photothermal. Photothermal interaction causes temperature change to the tissue. The light energy is converted into heat in photothermal applications. For axample, coagulation, vaporization, carbonization and melting may occur inside the tissue after laser exposure [8]. Photothermal effects listed in Table 1.1.

Table 1.1
Photothermal effects [8].

Temperature	Molecular and tissue reactions
42-45 °C	Hyperthermia leading to protein structural changes, hydrogen bond breaking, retraction
45-50 °C	More drastical conformational changes, enzyme inactivation, changes in membrane permeabilization, oedema
50-60 °C	Coagulation, protein denaturation
~80 °C	Collagen denaturation
80-100 °C	Dehydration
>100 °C	Boiling, steaming
100-300 °C	Vaporization, tissue ablation
>300 °C	Carbonization

The energy deposition inside the tissue is related to both optical properties of tissue and laser parameters. Also thermal tissue parameters provide heat storage inside the tissue after laser irradiation. A flowchart can show all the steps of heat generation and transferring in Figure 1.7.

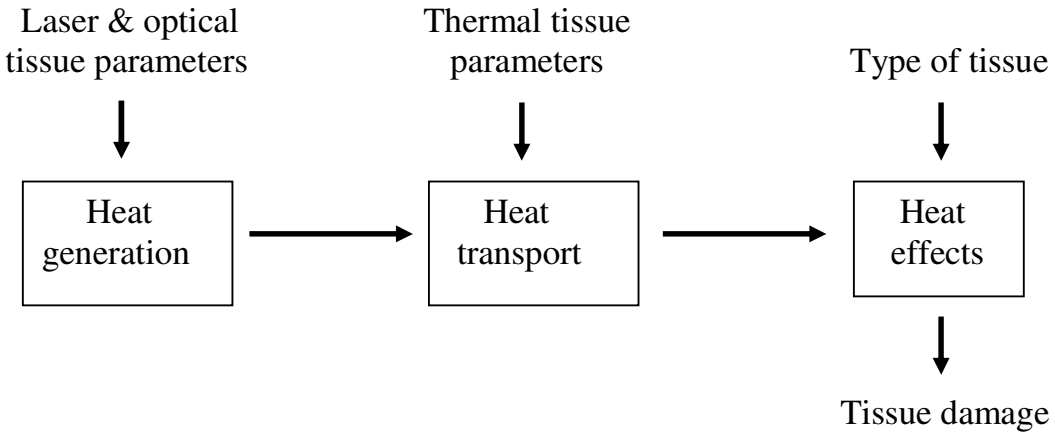


Figure 1.7 The steps of thermal mechanism [2].

1.3.3.3 Photoablation. In photoablation, with high energy the molecular bonds in tissues are broken and pure ablation occurs without any heat generation at the boundaries. Photoablation process has two steps:

Excitation: $AB + hv \rightarrow (AB)^*$

Dissociation: $(AB)^* \rightarrow A + B + E$

where A and B are atoms and E is kinetic energy. Excimer lasers are used for photoablation because high energy UV photons are needed [2, 4, 8].

1.3.3.4 Electromechanical. Photomechanical occurs at very high fluence rates causing optical breakdown. Plasma formation, shock wave generation, jet formation and cavitation are the types of electromechanical events [2, 4, 8]. The summary of the electromechanical interaction is given in Figure 1.8.

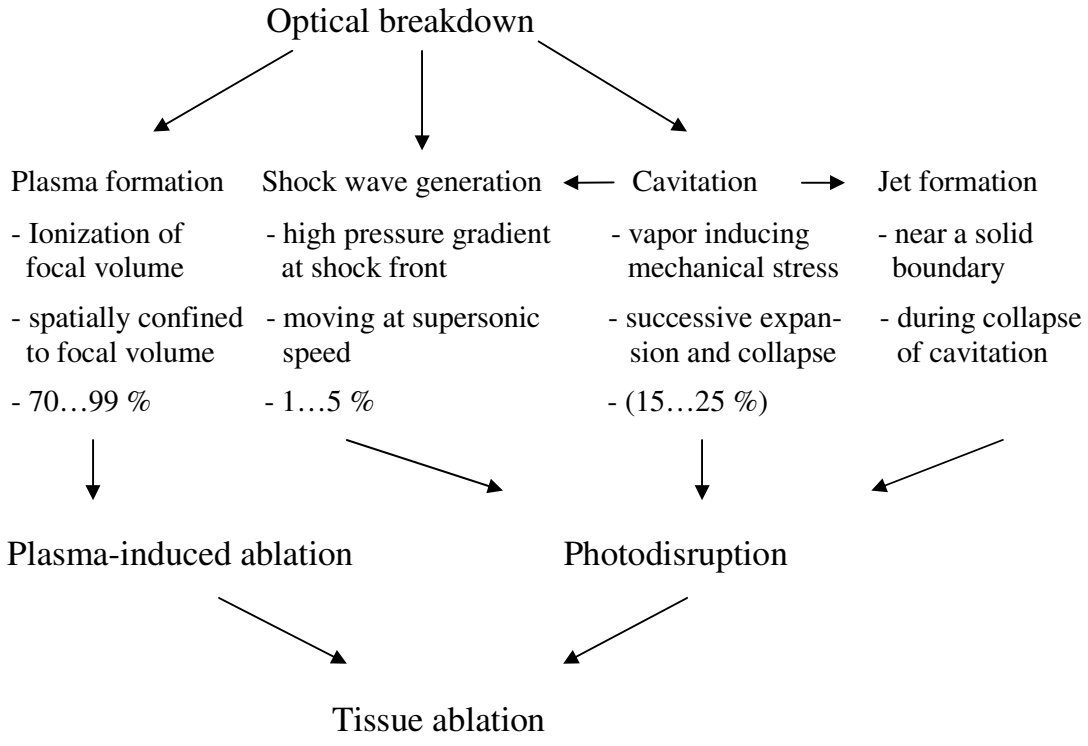


Figure 1.8 The summary of the electromechanical interaction [2].

1.3.4 Laser parameters:

Some definitions of laser parameters such as radiance, fluence rate and beam size are given in this part to understand terms used for light propagation [10-11].

Radiance (L(r, s)): is the quantity which describes the power of light propagation;
Fluence Rate ($\Phi(r)$): is defined as energy applied to an area of target tissue;
Beam Size or radius (r): distance from the laser beam center to the edge.

1.3.5 Numerical Solutions

The thermal response of laser irradiated tissue relies on both optical and thermal properties of the tissue. Various numerical models have been used for light propagation in turbid media such as discrete ordinate methods, diffusion theory and various flux models [12]. For alternative to all these numerical models, Monte Carlo Method has been frequently preferred for simulating light propagation in tissue. It can effectively simulate absorption and scattering of the light in the tissue, hence providing the spatial distribution of photon absorption. Moreover, light absorption can be modeled easily for multilayered tissues with different optical properties [10-14].

Numerical methods such as finite difference or finite element methods can be used to solve heat conduction equation for the calculation of temperature diffusion. Finite difference method is the most common numerical scheme to solve partial differential equations. The major advantage of the finite element method is its ability to deal with irregular boundaries and complex geometries [15-18]. On the other hand, the finite difference method is a more straightforward representation than the other. For solving time dependent heat conduction equation different schemes are available in finite difference technique such as explicit method, implicit method, Crack Nicholson method and alternating direction method [19-20].

2. METHODS

In this study, to control laser mechanism a simulation was designed by using the most reliable method Monte Carlo for estimating light distribution and Finite Difference Method for estimating thermal distribution. The simulation can be used for multilayered tissues and this provides wide usage for various laser applications. The program consists of three parts: Monte Carlo, Convolution and Finite Difference Method.

2.1 Monte Carlo for Photon Propagation

Photon propagation is simulated with perpendicularly incident photon beam on multilayered tissues by using Monte Carlo Method. Cartesian and cylindirical coordinate systems with the same origin are used for the program. Each layer is predicted to be infinitely wide because the layer length is much wider than the spatial extent of the photon distribution. Thickness, refractive index, absorption coefficient, scattering coefficient and anisotropy factor should be entered for each layer as an initial values. Refractive indices of ambient medium above and below the tissue is essential as well if they exist.

In the Monte Carlo Simulation two dimensional grid system in the r and z directions is essential to construct quantities of photon absorption. Each grid dimensions are selected as Δr and Δz in the r and z directions, respectively. The total number of grid elements are Nr and Nz in the r and z directions, respectively. A two dimensional grid system in the r and α directions is used to write diffuse reflactance and tranmittance. The range of α is $0 \leq \alpha \leq \pi/2$ and the total number of grid elements in α direction is Nα. So the grid separation for α is $\Delta\alpha = \pi/(2N\alpha)$.

Photon absorption, reflectance, transmittance and fluence are found in the Monte Carlo Simulation. The photon propagation is simulated in three dimensions. Firstly, photon deposition is recorded and then the probability of a photon absorption per unit volume is calculated after tracing multiple photons. Finally, the fluence is measured by dividing photon absorption over mua. This simulation also records the reflectance and transmittance

which are defined as the escape of photons at the top and bottom surface of the tissue. By integrating physical quantities such as A(r,z) over r, physical quantitities with lower dimensions (A(z)) can be calculated in the program [21-30].

2.1.1 Photon Initialization

Each photon has initial weight, W, equal to 1. The photon is started to pass perpendicularly into the tissue at the origin. The initial photon position is (0,0,0) and the directional cosines μ_x, μ_y, μ_z are (0,0,1), respectively [26-30].

After launching a photon, if refractive indices of the tissue and the ambient medium are different, specular reflectance, R_{sp}, will come into existence.

$$R_{sp} = \frac{(n_1 - n_2)^2}{(n_1 + n_2)^2}. \tag{2.1}$$

$n_1 \equiv$ refractive index of ambient medium

$n_2 \equiv$ refractive index of tissue

The photon weight changes due to the specular reflectance as below,

$$W = 1 - R_{sp}. \tag{2.2}$$

2.1.2 Generating Step Size

Random sampling of the probability density function is used to calculate photon's stepsize, s. The probability function over (a, b) is,

$$\int_a^b p(\chi)d\chi = \xi \tag{2.3}$$

where ξ is random variable in (0,1) range [26-30].

For the interval $(s',\ s'+ds')$ the probability of interaction is defined to calculate step size of the photon,

$$\mu_t = \frac{-dP\{s \geq s'\}}{P\{s \geq s'\}ds'}\ , \tag{2.4}$$

$$d(\ln(P\{s \geq s'\})) = -\mu_t ds'. \tag{2.5}$$

Eq. 2.5 is integrated in the range $(0, s_1)$, by using $P\{s \geq 0\} = 1$ and Eq. 2.5 becomes,

$$P\{s \geq s_1\} = \exp(-\mu_t s_1). \tag{2.6}$$

Eq. 2.6 can be rewritten as,

$$P\{s < s_1\} = 1 - \exp(-\mu_t s_1), \tag{2.7}$$

$$p(s_1) = \frac{dP\{s < s_1\}}{ds_1} = \mu_t \exp(-\mu_t s_1). \tag{2.8}$$

Eq. 2.8 is substituted into Eq. 2.3 and step size is found as below,

$$s_1 = -\ln(1-\xi)/\mu_t. \tag{2.9}$$

Due to symmetry, it becomes,

$$s_1 = -\ln(\xi)/\mu_t. \tag{2.10}$$

For multilayered media, Eq. 2.6 is rearranged,

$$P\{s \geq s_{sum}\} = \exp(-\sum_i \mu_{ti} s_i) \tag{2.11}$$

where i is index for layers. The total step size is written below,

$$s_{sum} = \sum_i s_i \,, \tag{2.12}$$

$$\sum_i \mu_{ti} s_i = -\ln(\xi) \,. \tag{2.13}$$

Eq. 2.13 is used as a step size for multilayered tissues.

2.1.3 Moving Photon

When the step size is calculated, the photon is moved in the biological medium. Then the new position of the photon is calculated,

$$\begin{aligned} x' &= x + \mu_x s_i \\ y' &= y + \mu_y s_i \\ z' &= z + \mu_z s_i \end{aligned} \,. \tag{2.14}$$

Cartesian coordinates are essential for simplicity to calculate new positions of the photon.

2.1.4 Photon Absorption

After a step of photon, the amount of absorbed photon weight ΔW is calculated and added to the local grid element.

$$\Delta W = \left(\frac{\mu_a}{\mu_t}\right) W , \tag{2.15}$$

$$A(r,z) \leftarrow A(r,z) + \Delta W . \tag{2.16}$$

The new photon weight is calculated as below,

$$W \leftarrow W - \Delta W . \tag{2.17}$$

2.1.5 Photon Scattering

The photon with the new weight after absorption will be scattered. Deflection angle and the azimuthal angle are calculated by the new generated random number. Henvey and Greenstein scattering function is used to specify probability distribution of the cosine of the deflection angle, $cos\theta$ [26-30].

$$p(\cos\theta) = \frac{1-g^2}{2(1+g^2-2g\cos\theta)^{3/2}} \tag{2.18}$$

where g is the anisotropy factor in the range of -1 and 1. But $cos\theta$ can be rearranged with random number as below,

$$\cos\theta = \begin{cases} \dfrac{1}{2g}\left\{1+g^2-\left[\dfrac{1-g^2}{1-g+2g\xi}\right]^2\right\} & g \neq 0 \\ 2\xi - 1 & g = 0 \end{cases} . \tag{2.19}$$

The azimuthal angle is ranged between 0 to 2π and can be expressed by,

$$\psi = 2\pi\xi . \tag{2.20}$$

The new direction of the photon can be obtained after the deflection and azimuthal angles are determined,

$$\mu_x' = \frac{\sin\theta(\mu_x\mu_z\cos\psi - \mu_y\sin\psi)}{\sqrt{1-\mu_z^2}} + \mu_x\cos\theta$$
$$\mu_y' = \frac{\sin\theta(\mu_y\mu_z\cos\psi - \mu_x\sin\psi)}{\sqrt{1-\mu_z^2}} + \mu_y\cos\theta \qquad (2.21)$$
$$\mu_z' = -\sin\theta\cos\psi\sqrt{1-\mu_z^2} + \mu_z\cos\theta$$

The following equations can be used if the angle is closed to the normal.

$$\mu_x' = \sin\theta\cos\psi$$
$$\mu_y' = \sin\theta\cos\psi \qquad (2.22)$$
$$\mu_z' = SIGN(\mu_z)\cos\theta$$

If μ_z is positive, $SIGN(\mu_z)$ equals to 1 otherwise $SIGN(\mu_z)$ equals to -1.

2.1.6 Internal Reflection

The photon may hit a boundary of consecutive layers during a step size and after hitting a boundary the photon can be transmitted across the boundary or reflected from the boundary. When the photon is reflected back into the current layer the photon continues its diffusion. It can also continue propagation if it passes through another tissue layer. But if it encounters air-tissue boundary it can be either reflected or transmitted. Stages should be defined for this process to understand these actions better [26-30].

For the first stage, the distance between photon position and the boundary of layer in the way of photon propagation should be calculated as below,

$$d_b = \begin{cases} \dfrac{(z_0 - z)}{\mu_z} & if\ \mu_z < 0 \\ \infty & if\ \mu_z = 0 \\ \dfrac{(z_1 - z)}{\mu_z} & if\ \mu_z > 0 \end{cases} \tag{2.23}$$

where z_0 and z_1 are the height of the top and the bottom boundaries of the layer.

In stage 2, if step size is greater than $d_b\mu_t$ then the photon hits the boundary and stepsize is rewritten as,

$$s \leftarrow s - d_b\mu_t . \tag{2.24}$$

If step size is not bigger than the product of distance and total coefficient, the photon moves with s/μ_t and s becomes zero. In this condition the photon is absorbed and scattered. Then with random number $-\ln(\xi)$ a generation of new step size is made and stage 1 is done again.

Stage 3: When the photon hits the boundary, the internal reflectance is calculated by using incident angle. α_i is zero for orthogonal incidence. α_i can be obtained by Eq. 2.25,

$$\alpha_i = \cos^{-1}(|\mu_z|). \tag{2.25}$$

The transmission angle α_t can be calculated by the Snell'law.

$$n_i \sin\alpha_i = n_t \sin\alpha_t . \tag{2.26}$$

If incident angle is bigger than critical angle when $n_i > n_t$, the internal reflectance $R(\alpha_i)$ equals to 1. If it is not, $R(\alpha_i)$ can be obtained by using Fresnel's formula.

$$R(\alpha_i) = \frac{1}{2}\left[\frac{\sin^2(\alpha_i - \alpha_t)}{\sin^2(\alpha_i + \alpha_t)} + \frac{\tan^2(\alpha_i - \alpha_t)}{\tan^2(\alpha_i + \alpha_t)}\right]. \tag{2.27}$$

Stage 4: A random number (ξ) is generated and try to consider the condition of the photon if it is internally reflected or transmitted. When $\xi \leq R(\alpha_i)$ the photon is internally reflected, otherwise ($\xi > R(\alpha_i)$) photon transmits.

For internaly reflection condition, the directional cosines becomes $(\mu_x, \mu_y, \mu_z) \leftarrow (\mu_x, \mu_y, -\mu_z)$ and stage 1 is repeated.

For transmission of photon, type of layer should be known. If the type of layer is a tissue layer then the direction of photon and the step size should be calculated as below,

$$\begin{aligned} \mu_x' &= \sin\alpha_t \mu_x / \sin\alpha_i \\ \mu_y' &= \sin\alpha_t \mu_y / \sin\alpha_i \\ \mu_z' &= SIGN(\mu_z)\cos\alpha_t \end{aligned} \quad . \tag{2.28}$$

With Snell's law it becomes,

$$\begin{aligned} \mu_x' &= \mu_x n_i / n_t \\ \mu_y' &= \mu_y n_i / n_t \\ \mu_z' &= SIGN(\mu_z)\cos\alpha_t \end{aligned} \quad . \tag{2.29}$$

Then continue with stage 1 for photon propagation.

If the type of layer is ambient layer two different conditions should be taken into account. If there is no scattering, reflectance and transmittance are calculated for unscattered condition. If there is scattering Diffuse Reflectance $R_d(r, \alpha_t)$ for z=0 and Transmittance $T_d(r, \alpha_t)$ for bottom of the tissue will be scored for current grid elements.

$$\begin{aligned} R_d(r, \alpha_t) &\leftarrow R_d(r, \alpha_t) + W \\ T_d(r, \alpha_t) &\leftarrow T_d(r, \alpha_t) + W \end{aligned} \quad . \tag{2.30}$$

2.1.7 Photon Termination

The weight of the photon goes down during the propagation of the photon and it reaches to a critical value. In this situation survival of photon should be determined. To find out this determination and also to provide conservation of energy, photon termination is essential. Russian roulette method is used to execute termination once $W \leq W_{th}$ condition occurs. The Russian roulette determines whether photon survives or not [28-30]. The roulette is based on a random number and can be expressed by,

$$W \leftarrow \begin{cases} mW & if\ \xi \leq 1/m \\ 0 & if\ \xi > 1/m \end{cases}. \tag{2.31}$$

This technique provides conservation of energy and termination of the photon finally.

2.1.8 Results of Monte Carlo Method

The goal of the Monte Carlo method is recording the absorption density A(r,z) for each grid elements inside the tissue. The fluence rate can be calculated by using absorption density and absorption coefficient as below,

$$F(r,z) = \frac{A(r,z)}{\mu_a}. \tag{2.32}$$

In the simulation multiple photon packets are propagated to increase the accuracy. The raw data of absorption density, diffuse reflectance and tranmittance are deposited in the program. To find out the total values in each grid element, the summation of 2D arrays and then 1D arrays is needed, respectively.

$$
\begin{aligned}
&A_z[i_z] = \sum_{i_r=0}^{N_r-1} A_{rz}[i_r, i_z] \\
&R_{d-r}[i_r] = \sum_{i_\alpha=0}^{N_\alpha-1} R_{d-r\alpha}[i_r, i_\alpha] \\
&R_{d-\alpha}[i_\alpha] = \sum_{i_r=0}^{N_r-1} R_{d-r\alpha}[i_r, i_\alpha] \\
&T_{d-r}[i_r] = \sum_{i_\alpha=0}^{N_\alpha-1} T_{d-r\alpha}[i_r, i_\alpha] \\
&T_{d-\alpha}[i_\alpha] = \sum_{i_r=0}^{N_r-1} T_{d-r\alpha}[i_r, i_\alpha] \\
&A_{layer} = \sum_{i_z} in\ layer\ A_z[i_z] \\
&A = \sum_{i_z=0}^{N_z-1} A_z[i_z] \\
&R_d = \sum_{i_r=0}^{N_r-1} R_{d-r}[i_r] \\
&T_d = \sum_{i_r=0}^{N_r-1} T_{d-r}[i_r]
\end{aligned}
\tag{2.33}
$$

where i_r, i_z, and i_α are indices of volume elements in r, z and α directions.

The raw data of each probability densities should be rewritten in unit area and solid angle as below,

$$
\begin{aligned}
&A_{rz}[i_r, i_z] \leftarrow A_{rz}[i_r, i_z] / (\Delta a \Delta z N) \\
&R_{d-r\alpha}[i_r, i_\alpha] \leftarrow R_{d-r\alpha}[i_r, i_\alpha] / (\Delta a \cos\alpha \Delta\Omega N) \\
&T_{d-r\alpha}[i_r, i_\alpha] \leftarrow T_{d-r\alpha}[i_r, i_\alpha] / (\Delta a \cos\alpha \Delta\Omega N)
\end{aligned}
\tag{2.34}
$$

where Δa is annular area and $\Delta\Omega$ is solid angle. They can be expressed as,

$$\Delta a = 2\pi(i_r + 0.5)(\Delta r)^2, \tag{2.35}$$

$$\Delta\Omega = 4\pi \sin[(i_\alpha + 0.5)\Delta\alpha]\sin(\Delta\alpha/2). \tag{2.36}$$

The raw data of 1D absorption, diffuse reflectance and transmittance are rewritten in unit area like 2D arrays as,

$$
\begin{aligned}
A_z[i_z] &\leftarrow A_z[i_z]/(\Delta z N) \\
R_{d-r}[i_r] &\leftarrow R_{d-r}[i_r]/(\Delta a N) \\
T_{d-r}[i_r] &\leftarrow T_{d-r}[i_r]/(\Delta a N) \\
R_{d-\alpha}[i_\alpha] &\leftarrow R_{d-\alpha}[i_\alpha]/(\Delta \Omega N) \\
T_{d-\alpha}[i_\alpha] &\leftarrow T_{d-\alpha}[i_\alpha]/(\Delta \Omega N)
\end{aligned}
\quad . \tag{2.37}
$$

Total diffuse reflectance and transmittance can be found by,

$$
\begin{aligned}
A_{layer} &\leftarrow A_{layer} / N \\
A &\leftarrow A / N \\
R_d &\leftarrow R_d / N \\
T_d &\leftarrow T_d / N
\end{aligned}
\quad . \tag{2.38}
$$

2.2 Convolution Program

Monte Carlo gives response to infinite laser beam to determine photon propagation. This responses should be converted into responses with variable size and shape of laser beams. In convolution program Green's function is essential to solve integrals for finite limits [10,30-32].

2.2.1 Principle of Convolution Program

After Monte Carlo Simulation the results should be convolved to find the responses in a finite laser beam by using Green's function [32]. The fluence rate response can be expressed with Green's function in cartesian coordinates as,

$$
\Phi(x, y, z) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G(x', y', z) S(x - x', y - y') dx' dy' \tag{2.39}
$$

where $S(x', y')$ is the source profile. Eq. 2.39 is rearranged for cylindrical coordinates step by step,

$$r = \sqrt{x^2 + y^2}\ , \tag{2.40}$$

$$\Phi(x, y, z) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G(\sqrt{x'^2 + y'^2}, z) S(\sqrt{(x - x')^2 + (y - y')^2})\, dx' dy'\ , \tag{2.41}$$

$$\Phi(r, z) = \int_{0}^{\alpha} G(r', z)\left[\int_{0}^{2\pi} S(\sqrt{r^2 + r'^2 - 2rr'\cos\theta})\, d\theta\right] r' dr'\ . \tag{2.42}$$

By using Bessel function this integral can be reduced into 1D to solve equation in a simple manner [32]. The fluence rate is solved for both gaussian beams and circular flat beams by the use of cylindirical symmetry.

2.2.2 Convolution for Gaussian Beams

The source function for gaussian beam with radius R [32] is expressed below,

$$S(r) = S_0 \exp(-2(r / R)^2)\ . \tag{2.43}$$

At r = 0 initial source value S_0 which depends on the power P is given by,

$$S_0 = 2P / \pi R^2\ . \tag{2.44}$$

By using the source function, Eq. 2.42 can be rewritten as,

$$\Phi(r, z) = S(r)\int_{0}^{\alpha} G(r', z) \exp(-2(r' / R)^2)\left[\int_{0}^{2\pi} \exp(4rr'\cos\theta / R^2)\, d\theta\right] r' dr'\ . \tag{2.45}$$

The integral over 0 to 2π in Eq. (2.45) can be replaced by zero ordered Bessel function (B_0) converted into cosine function and the fluence rate in Eq. 2.45 can be simplified as below,

$$B_0(x) = \frac{1}{2\pi}\int_0^{2\pi} \exp(x\cos\theta)d\theta\,, \tag{2.46}$$

$$\Phi(r,z) = S(r)\int_0^{\infty} G(r',z)\exp(-2(r'/R)^2)B_0(4rr'/R^2)2\pi r'dr'\,. \tag{2.47}$$

2.2.3 Convolution for Circular Flat Beams

The source function for flat beams can be expressed as [32],

$$S(r') = \begin{cases} \dfrac{P}{\pi R^2} & if\ r' \le R \\ 0 & if\ r' > R \end{cases}. \tag{2.48}$$

The fluence rate for flat beams can be obtained by using Eq. 2.42.

$$\Phi(r,z) = P/(\pi R^2)\int_0^{\infty} G(r',z)\Theta(r,r')2\pi r'dr' \tag{2.49}$$

where $\Theta(r,r')$ is given below,

$$\Theta(r,r') = \begin{cases} 1 & if\ 0 \le r \le R - r' \\ \left(\dfrac{1}{\pi}\right)\cos^{-1}\left[\dfrac{r'^2 + r^2 - R^2}{2rr'}\right] & if\ \left|R - r'\right| \le r \le R + r' \\ 0 & otherwise \end{cases}. \tag{2.50}$$

Eq. 2.49 is rewritten with new limits,

$$\Phi(r,z) = P/(\pi R^2)\int_a^{r+R} G(r',z)\Theta(r,r')2\pi r'dr'\,, \tag{2.51}$$

$$a = Max(0, r - R) \tag{2.52}$$

where Max() function uses the greater value inside the paranthesis.

2.2.4 Integration for Gaussian Beams

Computation is diffucult when Bessel function is not approximated by asymptotic form because of the overflow. For large arguments Bessel function can be rewritten by the approximation,

$$B_0(x) \approx \exp(x)/\sqrt{2\pi x}\,. \tag{2.53}$$

Based on the approximated Bessel function a new function defined to abate approximation and it is given below,

$$B_{0e}(x) = B_0(x)\exp(-x)\,. \tag{2.54}$$

Eq. 2.43, Eq. 2.44 and Eq. 2.54 are replaced into the Eq. 2.47 and it becomes,

$$\Phi(r,z) = \frac{4P}{R^2}\int_0^{\infty} G(r',z)\exp\left[-2\left(\frac{r'-r}{R}\right)^2\right] B_{0e}\left(\frac{4rr'}{R^2}\right) r'dr'\,. \tag{2.55}$$

The infinity limit of the integral can become finite by cutting exponential term of Eq. 2.55.

$$\left|r'-r\right| \le KR \quad \text{and} \quad r - KR \le r' \le r + KR \quad \rightarrow \quad K = 4\,. \tag{2.56}$$

After changing limits Eq. 2.55 becomes,

$$\Phi(r,z)=\frac{4P}{R^2}\int_a^b G(r',z)\exp\left[-2\left(\frac{r'-r}{R}\right)^2\right]B_{0e}\left(\frac{4rr'}{R^2}\right)r'dr', \tag{2.57}$$

$$\begin{aligned} a &= Max(0, r-KR) \\ b &= Min(r_{\max}, r+KR) \\ r_{\max} &= (N_r-0.5)\Delta r \end{aligned} \tag{2.58}$$

where Max() function uses the maximum value inside the brackets and Min() function uses the minumum value in Eq. 2.58.

In the convolution part each integrations are repeated for many times according to double array and this cause ineffective situation in the computation. This problem can be solved by using a dynamic data allocation. Binary tree is a good way to keep calculations of integrations and can provide a high memory for the program [32].

2.2.5 Integration for Circular Flat Beams

As mentioned before integrations need to be evaluated by using binary tree method to increase computation efficiency. Eq. 2.52 is rearranged by defining integral limits,

$$\Phi(r,z)=P/(\pi R^2)\int_a^b G(r',z)\Theta(r,r')2\pi r'dr' \tag{2.59}$$

$$\begin{aligned} a &= Max(0, r-KR) \\ b &= Min(r_{\max}, r+KR) \end{aligned}. \tag{2.60}$$

2.2.6 Solutions of Convolution Program

The Convolution program is computed for converting the results of Monte Carlo program to the finite beam size. Different types of laser beam are used in the convolution such as gaussian beam and circular flat beam.

In this program, the algorithm contains many integrations to reach a reasonable result. But it can cause time-cosuming for the computation and becomes unnecessarily large. This problem can be solved by binary tree method and this method increases the efficiency of the program.

2.3 Finite Difference Method for Thermal Diffusion

After simulating photon propagation to determine thermal changes in tissue Finite Difference Method is used to solve heat conduction equation. The fluence results of convolution program are multiplied by power and used as a source term in heat conduction equation [19-20, 33-35].

2.3.1 Heat Conduction Equation

The heat conduction equation was obtained from energy balance in small volume element of tissue in cylindrical coordinates [35]. Azimuthal symmetry is assumed and only r and z directions were drived according to suitability with heat source term calculated in the previous part. Heat diffusion is calculated by the use of energy conservation. Figure 2.1 shows volume element in cylindrical coordinates.

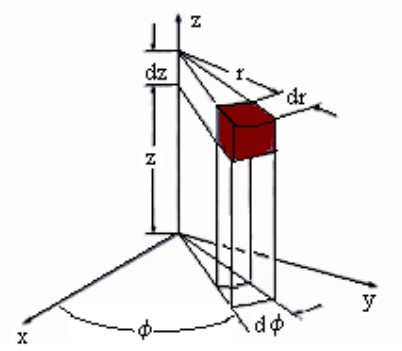


Figure 2.1 Illustration of the cylindrical volume element [35].

$$\dot{Q}_r + \dot{Q}_z - \dot{Q}_{r+\Delta r} - \dot{Q}_{z+\Delta z} + \dot{G}_{element} = \frac{\Delta E_{element}}{\Delta t} \tag{2.61}$$

where $\dot{Q}$ is rate of heat conduction and $\dot{G}$ is rate of heat generation.

The rate of energy change and the rate of heat generation can be expressed as below. (ρ, c and V are density, specific heat capacity and volume of the tissue element, respectively.)

$$\Delta E_{element} = E_{t+\Delta t} - E_t = mc(T_{t+\Delta t} - T_t) = \rho c V_{element}(T_{t+\Delta t} - T_t) \tag{2.62}$$

$$\dot{G}_{element} = \dot{q} V_{element} \tag{2.63}$$

where $\dot{q}$ is the source term for heat conduction equation. Eq. 2.62 and Eq. 2.63 are substituted in Eq. 2.61 and heat conduction equation is rewritten as follow,

$$\dot{Q}_r + \dot{Q}_z - \dot{Q}_{r+\Delta r} - \dot{Q}_{z+\Delta z} + \dot{q} V_{element} = \rho c V_{element} \frac{T_{t+\Delta t} - T_t}{\Delta t} \tag{2.64}$$

$$-\frac{\dot{Q}_{r+\Delta r} - \dot{Q}_r}{V_{element}} - \frac{\dot{Q}_{z+\Delta z} - \dot{Q}_z}{V_{element}} + \dot{q} = \rho c \frac{T_{t+\Delta t} - T_t}{\Delta t}. \tag{2.65}$$

By using definition of the derivative and Fourier's law of heat conduction Eq. 2.65 can be rewritten as Eq. 2.66,

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) + \frac{\partial^2 T}{\partial z^2} + \frac{\dot{q}}{k} = \frac{\rho c}{k}\frac{\partial T}{\partial t}\ . \tag{2.66}$$

2.3.2. Finite Difference Method

The heat diffusion equation can be used to find out thermal gradient of laser irradiated tissue by using explicit finite difference technique, assuming constant tissue thermal properties. The heat generation rate is obtained from the convolution program. After applying explicit method Eq. 2.66 becomes,

$$\frac{T_{i-1,j}^{n} - 2T_{i,j}^{n} + T_{i+1,j}^{n}}{(\Delta r)^2} + \frac{1}{i\Delta r}\frac{T_{i+1,j}^{n} - T_{i-1,j}^{n}}{2\Delta r} + \frac{T_{i,j-1}^{n} - 2T_{i,j}^{n} + T_{i,j+1}^{n}}{(\Delta z)^2} + \frac{q_{i,j}^{n}}{k} = \frac{\rho c}{k}\frac{T_{i,j}^{n+1} - T_{i,j}^{n}}{\Delta t} \quad (2.67)$$

$T_{i,j}^{n+1}$ and $T_{i,j}^{n}$ are the temperature matrix of the (i,j) node at the time of $(n+1)\Delta t$ and $n\Delta t$, respectively. For stability the size of the time step is limited to

$$\Delta t \leq \frac{1}{4}\frac{(\Delta r)^2}{\alpha} \quad (2.68)$$

where α is $k/\rho c$.

2.3.3 Boundary Conditions

At the top of the cylinder, the boundary surface loses heat due to the convection. This convection condition at the upper laser irradiated surface is given as,

$$-k\frac{\partial T}{\partial z} = h_{\infty}(T_b - T_{\infty}) \quad (2.69)$$

where T_b and T_{∞} are temperatures of the surface boundary and environment and h_{∞} is the convective heat transfer coefficient. At the bottom and the outer surface of the cylinder which are located far from the laser source, the boundary conditions are $\partial T/\partial r = 0$ and $\partial T/\partial z = 0$, respectively, assumed as isolated boundaries.

At the center of the cylinder when r approaches zero the condition is $\partial T/\partial r = 0$ and for $r = 0$ it becomes

$$\lim_{r\to 0}\left(\frac{1}{r}\frac{\partial T}{\partial r}\right) = \frac{\partial^2 T}{\partial r^2} \ . \tag{2.70}$$

From this approximation it can be said that at the origin the points around the center line has the same temperature value. This can be used to calculate the temperature values at $r = 0$ by substituting Eq. 2.70 into Eq. 2.67.

In summary, by iterating over all of the individual space steps for $i_r = 0,1,2 \ldots n_r - 1$ and $i_z = 0,1,2 \ldots n_z - 1$ at each time steps $t = 0,1,2 \ldots n_t - 1$ the temperature gradient of the tissue after laser application can be simulated.

3. CONCLUSION

In this study a Java Simulation Application was developed for estimating light propagation and heat diffusion of laser irradiated biological medium. There are three main parts in this program listed below:

- Monte Carlo Model : simulate light propagation by using optical properties;
- Convolution Program : convolved the results of Monte Carlo;
- Finite Difference Method : simulate heat diffusion by using thermal properties.

This program can be very useful for different laser applications such as photochemical therapy and photothermal techniques. The fluence rate can be calculated for different laser beam sizes. The user can select either gaussian beam or circular flat beam. Not only fluence rate but also thermal gradients can be plotted with this simulation. It is very advantegous for users because there is no need for another program such as Matlab or Excel, etc. The whole parts needed for simulating optical and thermal response of laser irradiated tissue are included in Java based simulation program.

All the parameters needed for the simulation can be chosen optionally or you can set them into program. In the first part optical properties of the tissue is essential for light distribution. Second part convolved the impulse responses to finite beam size so laser parameters should be entered by the user. The last part of the program simulates thermal gradients of the tissue after selecting thermal properties of the medium.

The output of the program consists of different variables such as absorption probability density, reflectance, transmittance, fluence rates and thermal results. The fluence rate can be plotted by 2D surface plotting and also thermal gradients can be illustrated too.

In the program the cylindrical coordinate system is used because complex algorithms are needed to be simplified with symetrical geometries. The radius and the height of the cylinder can be set into the program. As mentioned before cylinder is divided into multiple volume elements and each element filled during the simulation of light propagation. The grid element dimensions can be changed optionally as well. In the Monte Carlo the fluence rate is calculated for infinite laser beams. This results are needed to be convolved according to convert responses to finite beam size.

3.1 Results and Discussion

The fluence rate (J/cm²) was graphed for differerent tissue types such as brain and liver and for different laser parameters. For example, in different laser powers thermal changes of the same tissue type were compared. Also, changes in thermal distributions of the tissues for different wavelengths were observed. A sample simulation for multilayered tissue was given to see the ability of the program for various usages. Finally, the results were compared and discussed to emphasize the program capability and additional developments for future work were explained.

Optical and thermal parameters of different tissue types for different wavelengths are given in Table 3.1-3.5 [36, 37]. Radial distance and depth of the tissue were taken 0.1 cm for each simulation. Figure 3.1 and 3.2 represents the fluence and thermal contours of (a) human brain and (b) liver, respectively. All the laser parameters for these simulations are the same. For example, the wavelenth, power and exposure time of the laser were chosen same for each tissue type. The fluence contour of brain tissue is totally different than that of liver tissue. The fluence diffusion of brain is circular and nearer to the surface than the fluence diffusion of liver. That can be explained by the optical differences of each tissue type because absorption coefficient of the tissue determines the penetration depth. By comparing the absorption coefficient of each tissue type in Table 3.1 and 3.3 it can be seen that the absorption coefficient of the brain is greater than the liver's. The penetration depth is inversely proportional to absorption coefficient so the thermal diffusion of liver can reach to the more deep regions.

Table 3.1
Optical parameters of Brain Parts at 1064nm [36].

Optical Paramaters of Brain Parts at 1064nm				
Brain parts	n	μ_a (1/cm)	μ_s (1/cm)	g
White matter	1.3	3.2	469	0.87
Grey matter	1.3	5	134	0.9

Table 3.2
Thermal Parameters of Brain Parts [37].

Thermal Parameters of Brain Parts			
Brain parts	c (J/gr °C)	k (W/cm°C)	density (gr/cm³)
White matter	3.7	0.0049	1.03
Grey matter	3.7	0.0049	1.03

Table 3.3
Optical Parameters of Liver at 1064nm [36].

Optical Parameters of Liver at 1064nm			
n	μ_a (1/cm)	μ_s (1/cm)	g
1.3	0.3	150	0.93

Table 3.4
Optical Parameters of Liver at 630 nm [36].

Optical Parameters of Liver at 630nm			
n	μ_a (1/cm)	μ_s (1/cm)	g
1.3	3.2	414	0.95

Table 3.5
Thermal Parameters of Liver [37].

Thermal Parameters of Liver		
c (J/gr °C)	k (W/cm°C)	density (gr/cm³)
3.6	0.0046	1.05

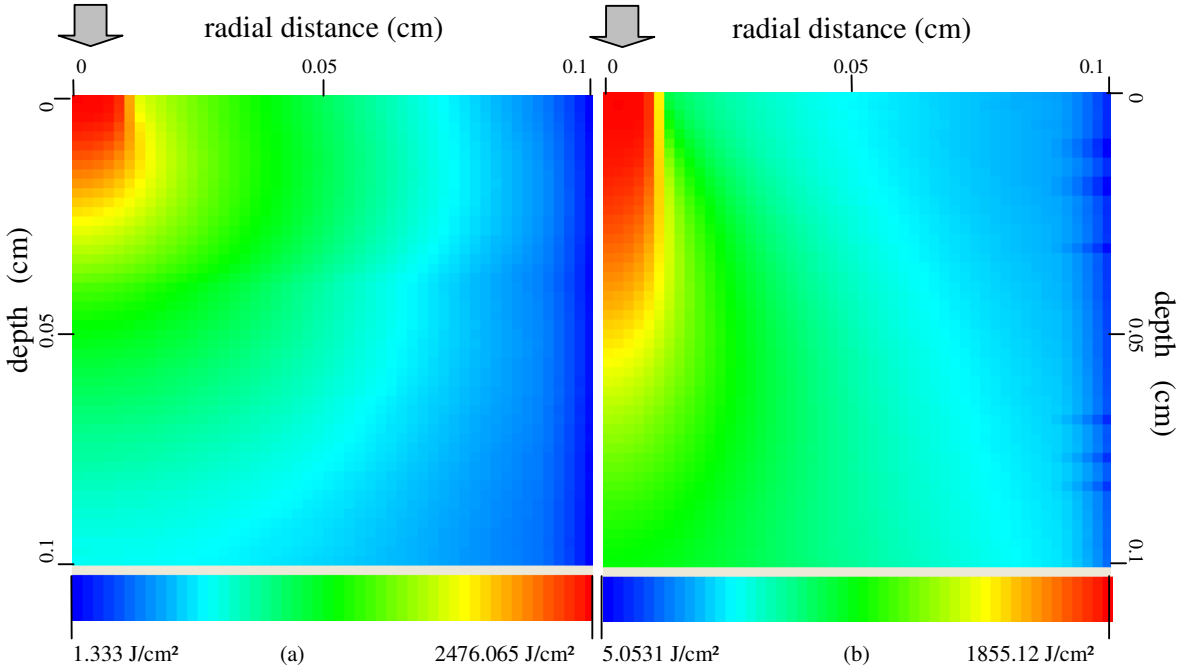


Figure 3.1 Fluence contours of (a) brain and (b) liver tissue at 1064 nm from left to right. Same laser power, exposure time and beam radius are selected. (P=0.5W, t=2sec, R=0.01cm) Arrow ( ) indicates the position of laser beam.

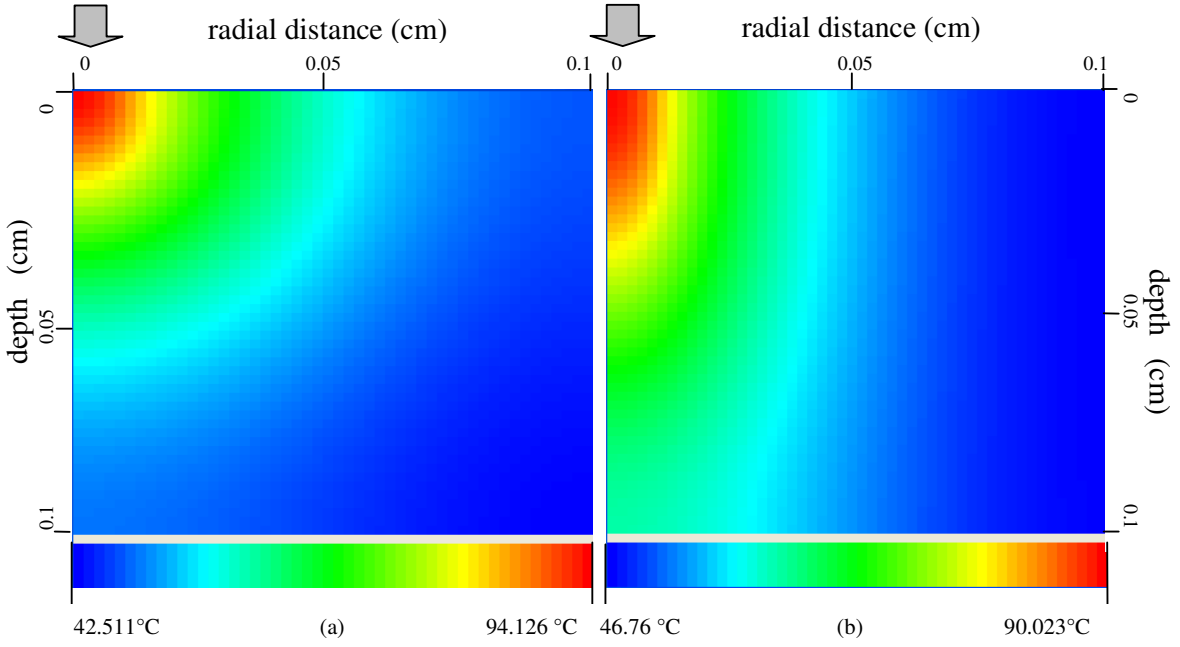


Figure 3.2 Thermal contours of (a) brain and (b) liver tissue at 1064 nm from left to right. Same laser power, exposure time and beam radius are selected. (P=0.5W, t=2sec, R=0.01cm) Arrow ( ) indicates the position of laser beam.

The fluence and thermal simulations of liver tissue with different laser powers, (a) 0.5W and (b) 0.1W, are shown in Figure 3.3 and 3.4, respectively. The wavelegth and exposure time are same for each simulations. When the power of laser is different, the fluence range changes for each condition as seen in Figure 3.3. Also, thermal diffusion of the same tissue type alters according to power of laser. High power causes more wide thermal diffusion as expected. The difference of the temperature range for each simulation can be seen in Figure 3.4.

Different laser exposure time also affects the thermal distribution of laser irradiated tissue. When exposure time of the laser increases, thermal gradients diverse proportionally. The effect of the laser exposure time to the thermal simulation of the tissue irradiated with 0.5W laser power at 630 nm can be seen clearly in Figure 3.5 and 3.6.

In Figure 3.7 and 3.8, the effect of various wavelenths are pointed out for the same tissue type irradiated with same laser power, 0.5W, and exposure time, 2sec. The optical parameters of the tissue changes due to the various laser wavelengths. At 630nm the absorption coefficient of the liver tissue is samaller than that at 1064nm. So the thermal diffusion at 1064nm can reach to the deeper regions of the tissue. The relation between penetration depth and absorption coefficient explains the difference of those thermal simulations.

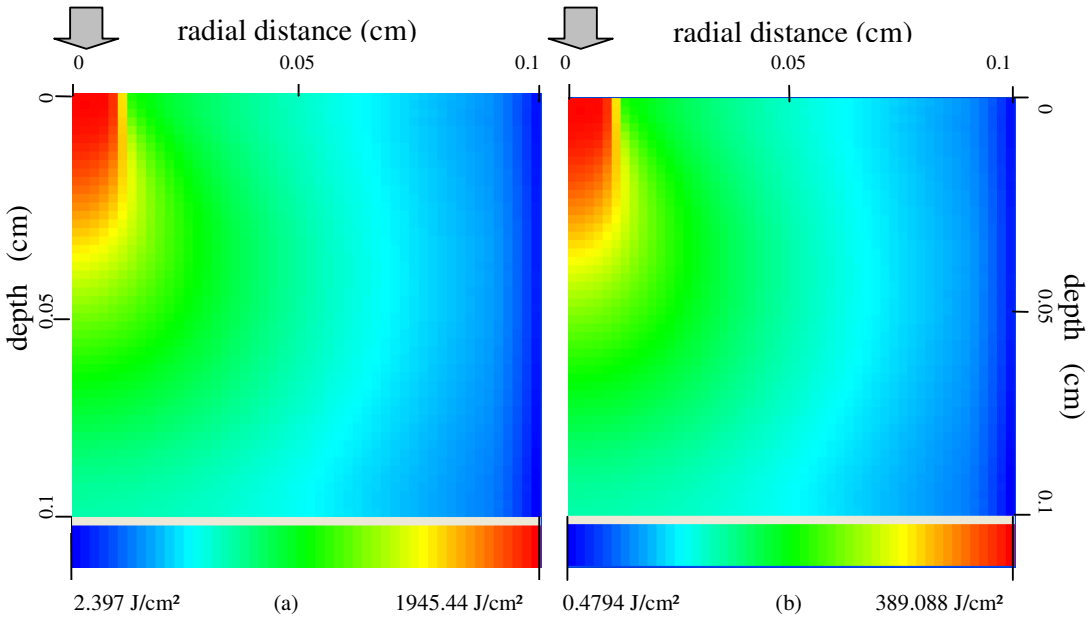


Figure 3.3 Fluence contours for liver tissue with (a) 0.5W and (b) 0.1W, respectively. (λ= 630nm, t=2sec and R=0.01cm) Arrow ( ) indicates the position of laser beam.

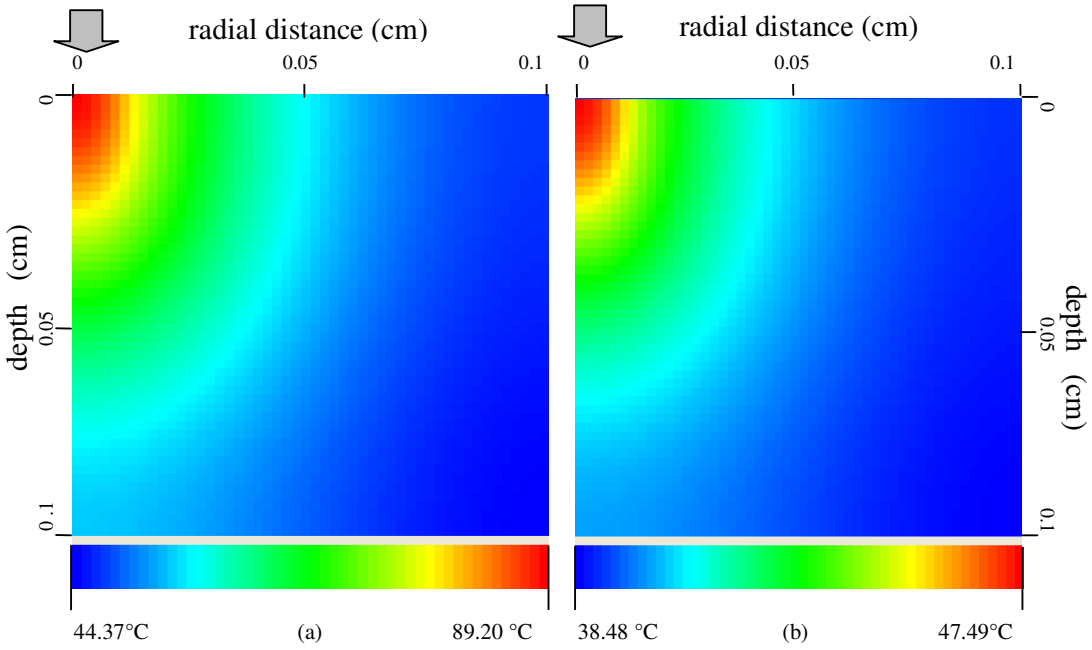


Figure 3.4 Thermal contours for liver tissue with (a) 0.5W and (b) 0.1W, respectively. (λ= 630nm, t=2sec and R=0.01cm) Arrow ( ) indicates the position of laser beam.

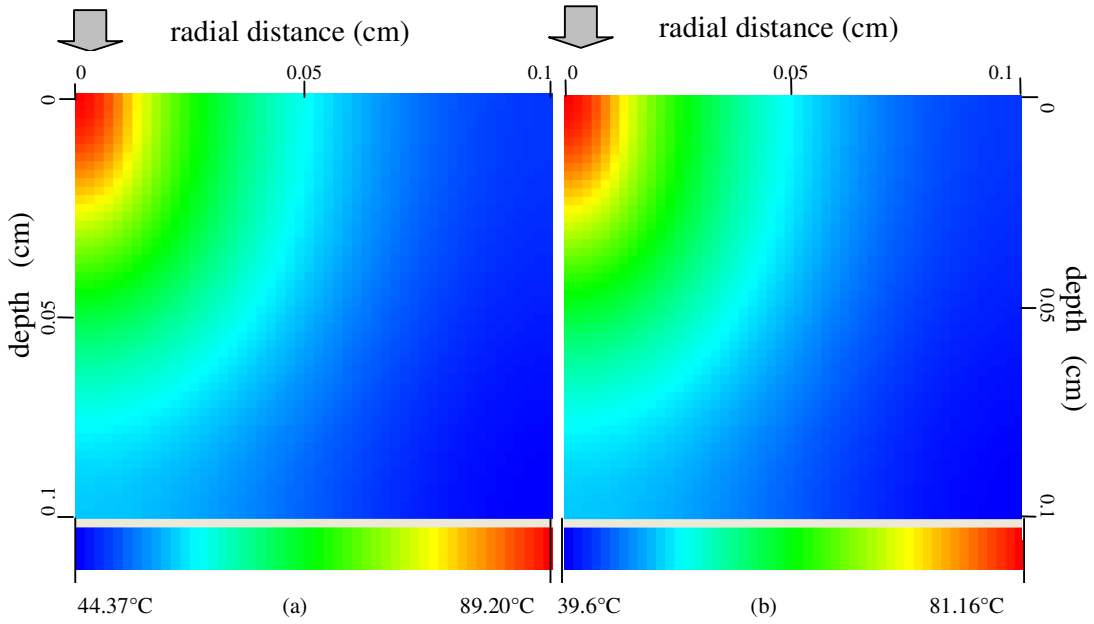


Figure 3.5 Thermal contours for liver tissue at (a) t=2sec and (b) t=1sec from left to right. (λ= 630 nm, P=0.5 W, R=0.01cm) Arrow ( ) indicates the position of laser beam.

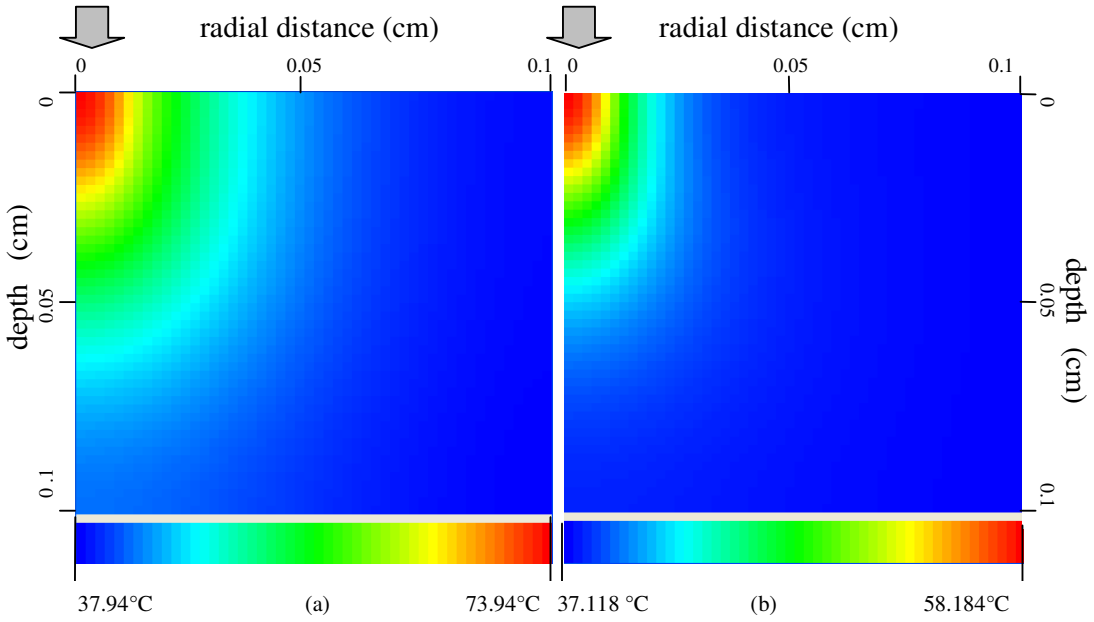


Figure 3.6 Thermal contours for liver tissue at (a) t=0.5sec and (b) t=0.1sec from left to right. (λ= 630 nm, P=0.5 W, R=0.01cm) Arrow ( ) indicates the position of laser beam.

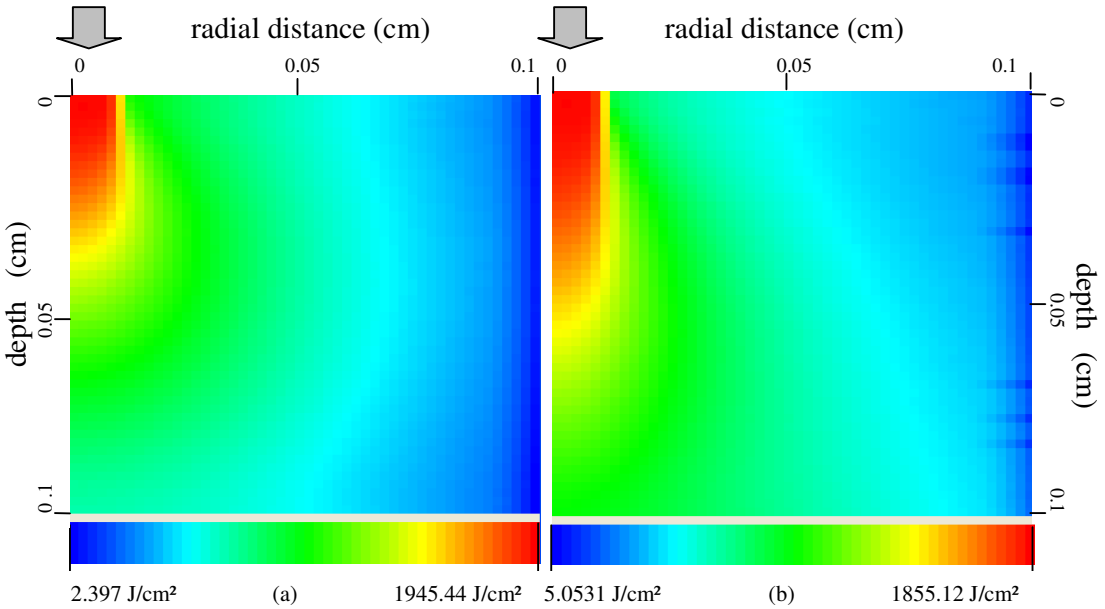


Figure 3.7 Fluence contours for liver tissue at (a) 630 nm and (b) 1064 nm from left to right. Laser power and exposure time are taken constant for each simulation. (P=0.5W, t=2sec, R=0.01cm) Arrow ( ) indicates the position of laser beam.

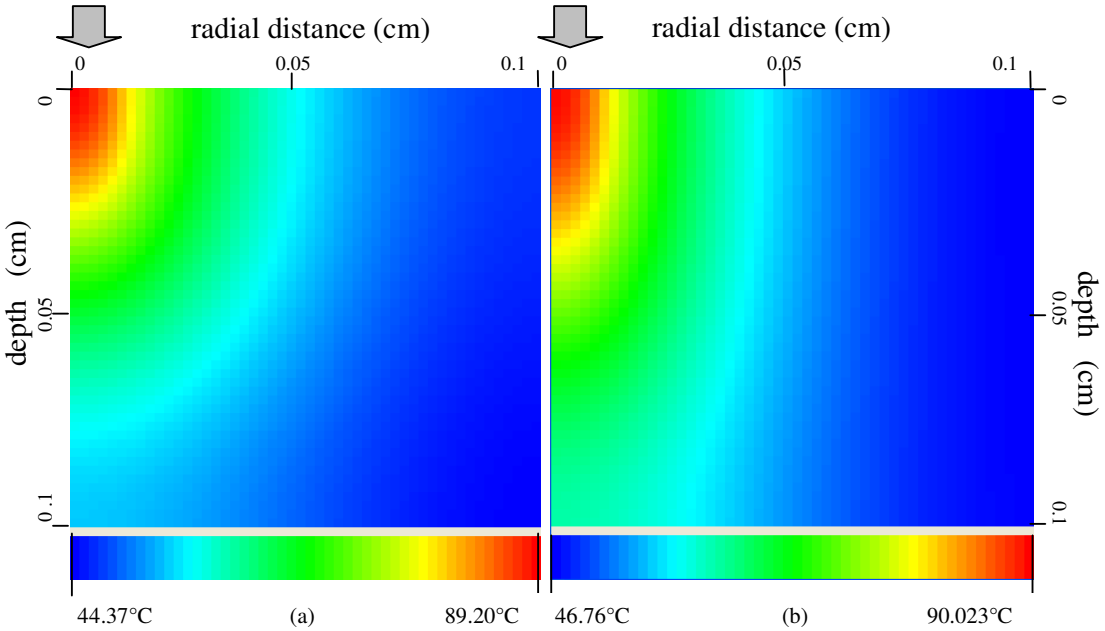


Figure 3.8 Thermal contours for liver tissue at (a) 630 nm and (b)1064 nm from left to right. Laser power and exposure time are taken constant for each simulation . (P=0.5W, t=2sec, R=0.01cm) Arrow ( ) indicates the position of laser beam.

The measurements of fluence and thermal contours of laser irradiated tissues are essential to compare results for different optical properties and laser parameters to see how this program works. The fluence and thermal graphs are reasonable for each situation but the results are compared just by using this program. But the simulation program is needed to be compared by another research for testing it. This test is required to ensure about the accuracy of results obtained from the Java simulation application.

In the test four layered skin tissue was irradiated by 0.005 W laser power with 0.01 cm beam radius. The thicknesses of each layer are 0.01 cm, 0.02 cm, 0.02 cm, 0.09 cm from top to bottom, respectively [38]. The optical properties of skin tissue is given in Table 3.6 and the fluence contour of four layered skin tissue is shown in Figure 3.9. The fluence results are same and graph is completely matched with that one in the refered article. As a result the simulation program in this study is validated by comparing it with another study.

Table 3.6
Optical parameters of skin layers [38].

Optical Parameters of Skin layers				
	n	μ_a (1/cm)	μ_s (1/cm)	g
1. layer	1.5	32	165	0.72
2. layer	1.4	23	227	0.72
3. layer	1.4	40	246	0.72
4. layer	1.4	23	227	0.72

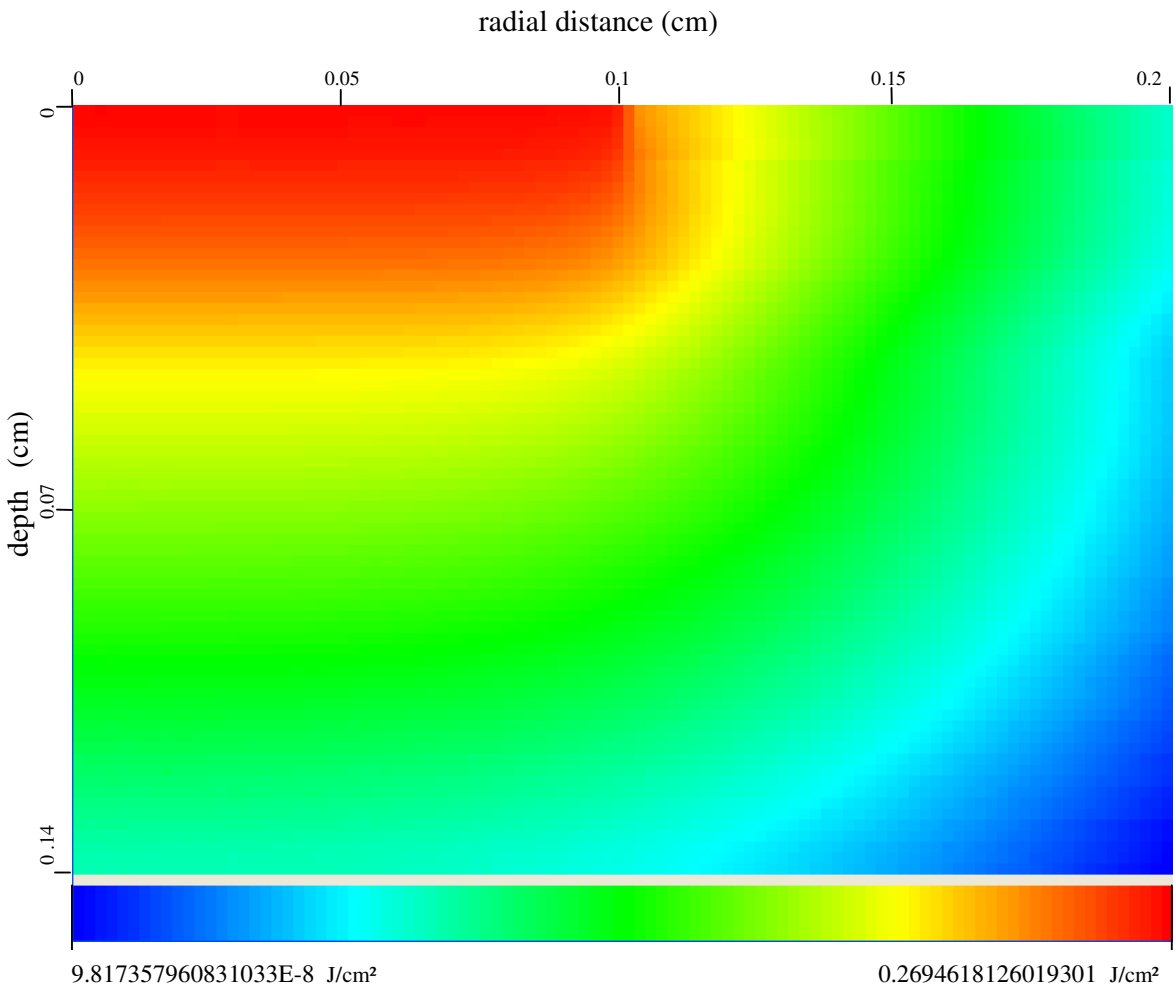


Figure 3.9 The fluence contour of four layered skin tissue with 0.005W laser power and 0.1cm beam radius.

3.2 Future Work

A multipurpose Java Application Program was designed for simulating optical and thermal response of laser irradiated tissue. Monte Carlo Simulations calculated the source terms of the heat conduction equation and Finite Difference Method was used for solving heat transfer equation. In the near future some additional work can be done for increasing the cabability of the program. For example, an additional part can be added to the heat transfer equation for calculating blood perfusion rates of tissues. Moreover, the interface of the Java program can be changed for simplifying the use of the program. Optical and

thermal parameters of various tissue types for different wavelenths can be set into the code and this can provide a user to select desired parameters of a tissue just by click on a button.

In the distant future, each part of the simulation program can be improved separately. For example, Finite Element Method can be used instead of Finite Difference Method to modify simulation for complex geometries. Also program can be redesigned for lesion buried tissues to determine opical and thermal response of laser irradiated tissue. Last but not least, functionality of the program will increase if 3D plotting is used.

APPENDIX A. MAIN PROGRAM

MonteInterface.java

```
package montecarlo.Face;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;
import javax.swing.JProgressBar;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JSpinner;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;

import montecarlo.Convolution.Face.BeamType;
import montecarlo.Convolution.Face.ConvError;
import montecarlo.Convolution.Face.Resolution;

import montecarlo.Draw.ImgMod29;

import montecarlo.Graph.GraphicsTher;

import montecarlo.Process.InitOutputData;
import montecarlo.Process.ScaleResultData;
```

```
import montecarlo.Thermal.Face.InitialTemperature;
import montecarlo.Thermal.Face.LayerThermalInterface;
import montecarlo.Thermal.LayerThermalData;


public class MonteInterface extends JFrame {

  public TissueParam Tis;
  public InitOutputData initoutputdata;
  public ScaleResultData resData;
  public LayerThermalInterface[] lyrThrmlInt;
  public int a = 0;
  public InitialTemperature initialtemperature;
  static String windowsClassName =
    "com.sun.java.swing.plaf.windows.WindowsLookAndFeel";

  private JButton btnstart = new JButton();
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenu1 = new JMenu();
  private JMenu jMenu2 = new JMenu();
  private JMenu jMenu3 = new JMenu();
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();
  private JMenuItem jMenuItem3 = new JMenuItem();
  private JMenuItem jMenuItem4 = new JMenuItem();
  private JMenuItem jMenuItem5 = new JMenuItem();
  private JMenuItem jMenuItem6 = new JMenuItem();
  private JSeparator jSeparator1 = new JSeparator();
  private JSpinner jSpinner1 = new JSpinner();
  private JProgressBar jProgressBar1 = new JProgressBar();
  private JPopupMenu jPopupMenu1 = new JPopupMenu();
  private JLabel jLabel1 = new JLabel();
  private JLabel jLabel2 = new JLabel();
  private JLabel jLabel3 = new JLabel();
  private JSpinner jSpinner2 = new JSpinner();
  private JSpinner jSpinner3 = new JSpinner();
  private JLabel jLabel4 = new JLabel();
  private JLabel jLabel5 = new JLabel();
  private JSpinner jSpinner4 = new JSpinner();
  private JSeparator jSeparator2 = new JSeparator();
  private JLabel jLabel11 = new JLabel();
  private JSeparator jSeparator3 = new JSeparator();
  private JTextField jTextField4 = new JTextField();
  private JSeparator jSeparator4 = new JSeparator();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private JButton jButton4 = new JButton();
  private JButton jButton5 = new JButton();
```

```
 private JButton jButton6 = new JButton();
 private JButton jButton7 = new JButton();
 private JButton jButton8 = new JButton();
 private JButton jButton9 = new JButton();
   private JButton jButton10 = new JButton();
   private JTextArea jTextArea1 = new JTextArea();
 private JButton jButton11 = new JButton();
private Thread ni ;
 private JSpinner jSpinner5 = new JSpinner();
   private JButton jButton12 = new JButton();
 protected int minValue = 0;
 protected int maxValue = 100;
 protected int counter = 0;
 Thread progressbar = null;


   public MonteInterface() {

   try {
    jbInit();
   } catch (Exception e) {
    e.printStackTrace();
   }
 }


 public class progressbar extends Thread {

   public void run() {
    counter = minValue;
    while (counter <= maxValue) {
     Runnable runme = new Runnable() {
      public void run() {
       jProgressBar1.setValue(counter);
      }
     };
     SwingUtilities.invokeLater(runme);
     counter++;
     try {
      Thread.sleep(1000);
     } catch (Exception ex) {
     }
    }
   }


 }

 private void jbInit() throws Exception {
```

```
this.getContentPane().setLayout(null);
this.setSize(new Dimension(986, 587));
btnstart.setText("Start");
btnstart.setBounds(new Rectangle(860, 490, 95, 25));
btnstart.setBorder(BorderFactory.createLineBorder(Color.black, 1));
btnstart.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      btnstart_actionPerformed(e);
    }
  });

jMenu1.setText("File");
jMenu2.setText("View");
jMenu3.setText("Help");
jMenuItem1.setText("Open");
jMenuItem2.setText("Save");
jMenuItem3.setText("Exit");
jMenuItem4.setText("Version");
jMenuItem5.setText("Help");
jMenuItem6.setText("Home Page");

jSeparator1.setBounds(new Rectangle(690, 135, 280, 10));
jSpinner1.setToolTipText("null");
jSpinner1.setMinimumSize(new Dimension(1, 1));
jSpinner1.setBorder(BorderFactory.createLineBorder(Color.black, 1));
jSpinner1.setBounds(new Rectangle(820, 25, 120, 20));
jProgressBar1.setBounds(new Rectangle(275, 490, 580, 25));
jProgressBar1.setMinimum(2);
jProgressBar1.setMaximum(100);
jProgressBar1.setStringPainted(true);



jPopupMenu1.setVisible(true);
jLabel1.setText("Scale (pixels / mm) :");
jLabel1.setBounds(new Rectangle(700, 20, 115, 30));
jLabel2.setText("Center View at ( mm )");
jLabel2.setBounds(new Rectangle(700, 60, 130, 30));
jLabel3.setText("x =");
jLabel3.setBounds(new Rectangle(705, 95, 30, 25));
jSpinner2.setBounds(new Rectangle(725, 95, 40, 20));
jSpinner2.setBorder(BorderFactory.createLineBorder(Color.black, 1));
jSpinner3.setBounds(new Rectangle(895, 95, 40, 20));
jSpinner3.setBorder(BorderFactory.createLineBorder(Color.black, 1));
jLabel4.setText("z =");
jLabel4.setBounds(new Rectangle(875, 95, 30, 25));
jLabel5.setText(" y =");
jLabel5.setBounds(new Rectangle(775, 95, 30, 25));
jSpinner4.setBounds(new Rectangle(805, 95, 40, 20));
```

```
jSpinner4.setBorder(BorderFactory.createLineBorder(Color.black, 1));
jSeparator2.setBounds(new Rectangle(690, 230, 280, 10));
jLabel11.setText("Sim time ( sec ) :");
jLabel11.setBounds(new Rectangle(700, 355, 100, 20));
jSeparator3.setBounds(new Rectangle(675, 10, 10, 465));
jSeparator3.setOrientation(SwingConstants.VERTICAL);
jTextField4.setBounds(new Rectangle(805, 355, 135, 20));
jSeparator4.setBounds(new Rectangle(695, 385, 275, 2));
jButton1.setText("Laser Parameters");
jButton1.setBounds(new Rectangle(705, 140, 250, 20));
jButton1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton1_actionPerformed(e);
    }
  });
jButton2.setText("Tissue Parameters");
jButton2.setBounds(new Rectangle(705, 170, 250, 20));
jButton2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton2_actionPerformed(e);
    }
  });

jButton3.setText("Output Data");
jButton3.setBounds(new Rectangle(705, 200, 250, 20));
jButton3.setToolTipText("null");
jButton3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton3_actionPerformed(e);
    }
  });

jButton4.setText("Laser Beam Type ");
jButton4.setBounds(new Rectangle(705, 245, 250, 20));
jButton4.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton4_actionPerformed(e);
    }
  });

jButton5.setText("Resolution");
jButton5.setBounds(new Rectangle(705, 275, 250, 20));
jButton5.setToolTipText("null");
jButton5.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton5_actionPerformed(e);
    }
  });
jButton6.setText("Convolution Error");
```

```
jButton6.setBounds(new Rectangle(705, 305, 250, 20));
jButton6.setToolTipText("null");
jButton6.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton6_actionPerformed(e);
    }
  });
jButton7.setText("Conv Start");
jButton7.setBounds(new Rectangle(145, 490, 100, 25));
jButton7.setToolTipText("null");
jButton7.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton7_actionPerformed(e);
    }
  });
jButton8.setText("Tissue Thermal Parameters");
jButton8.setBounds(new Rectangle(705, 395, 250, 20));
jButton8.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton8_actionPerformed(e);
    }
  });
jButton9.setText("Initial Temperature Values");
jButton9.setBounds(new Rectangle(705, 425, 250, 20));
jButton9.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton9_actionPerformed(e);
    }
  });
    jButton10.setText("Thermal Start");
    jButton10.setBounds(new Rectangle(30, 490, 100, 25));
    jButton10.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent e) {
                 jButton10_actionPerformed(e);
              }
           });
    jTextArea1.setBounds(new Rectangle(15, 90, 655, 380));
    jTextArea1.setFont(new Font("Dialog", 0, 14));
    jMenu1.add(jMenuItem1);
jMenu1.add(jMenuItem2);
jMenu1.add(jMenuItem3);
jMenuBar1.add(jMenu1);
jMenuBar1.add(jMenu2);
jMenu3.add(jMenuItem4);
jMenu3.add(jMenuItem5);
jMenu3.add(jMenuItem6);
jMenuBar1.add(jMenu3);
    this.getContentPane().add(jButton12, null);
    this.getContentPane().add(jSpinner5, null);
```

```
    this.getContentPane().add(jButton11, null);
    this.getContentPane().add(jTextArea1, null);
this.getContentPane().add(jButton10, null);
    this.getContentPane().add(jButton9, null);
this.getContentPane().add(jButton8, null);
this.getContentPane().add(jButton7, null);
this.getContentPane().add(jButton6, null);
this.getContentPane().add(jButton4, null);
this.getContentPane().add(jButton5, null);
this.getContentPane().add(jButton3, null);
this.getContentPane().add(jButton2, null);
this.getContentPane().add(jButton1, null);
this.getContentPane().add(jSeparator4, null);
this.getContentPane().add(jTextField4, null);
this.getContentPane().add(jSeparator3, null);
this.getContentPane().add(jLabel11, null);
this.getContentPane().add(jSeparator2, null);
this.getContentPane().add(jSpinner4, null);
this.getContentPane().add(jLabel5, null);
this.getContentPane().add(jLabel4, null);
this.getContentPane().add(jSpinner3, null);
this.getContentPane().add(jSpinner2, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(jLabel2, null);
this.getContentPane().add(jLabel1, null);
this.getContentPane().add(jProgressBar1, null);
this.getContentPane().add(jSeparator1, null);
this.getContentPane().add(btnstart, null);
this.getContentPane().add(jSpinner1, null);

this.setJMenuBar(jMenuBar1);
this.setVisible(true);
this.setTitle("Laser Thermal v1.1");
this.setBackground(new Color(219, 224, 233));
this.jMenuBar1.setVisible(true);

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = this.getSize();
this.setLocation((screenSize.width - frameSize.width) / 2,
            (screenSize.height - frameSize.height) / 2);

JScrollPane scrollPane = new JScrollPane(this.jTextArea1);
scrollPane.setBounds(new Rectangle(15, 100, 655, 370));
    jButton12.setText("Fluence Graph");
    jButton12.setBounds(new Rectangle(35, 5, 140, 30));
jButton12.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      jButton12_actionPerformed(e);
    }
```

```
    });
  jSpinner5.setBounds(new Rectangle(200, 30, 55, 25));
  jSpinner5.setBorder(BorderFactory.createLineBorder(Color.black, 1));
  jButton11.setText("Thermal Graph");
  jButton11.setBounds(new Rectangle(35, 45, 140, 30));
  jButton11.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton11_actionPerformed(e);
      }
    });
  this.getContentPane().add(scrollPane, null);
  this.addWindowListener(new WindowAdapter() {

      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
    jTextArea1.setFont(new Font("Tahoma", 0, 14));


  System.gc();
}

public static void main(String[] args) {

  System.out.println("Bahar Kurt");

  try {
    UIManager.setLookAndFeel(windowsClassName);
  } catch (Exception exc) {
    System.err.println("Error loading L&F: " + exc);
  }

  new MonteInterface();

}

public void finalize() {
  System.out.println("bahars");
}


private void btnstart_actionPerformed(ActionEvent e) {

  progressbar = new progressbar();
  Thread mainSttng1 = new mainSttng();
  mainSttng1.start();
  progressbar.start();
  this.jTextField4.setText("0");
```

```
}

private void jButton1_actionPerformed(ActionEvent e) {
  //Laser parameters
  LaserInterface lase = new LaserInterface();
}

private void jButton2_actionPerformed(ActionEvent e) {
  this.Tis = new TissueParam();
}

public class  mainSttng extends Thread {

  double i_photon = 0.0;
  double num_photons = 0.0;
public void run() {
  initoutputdata = new InitOutputData();
  initoutputdata.outParam.setRsp(initoutputdata.Rspecular(Tis.lyrInt));
  num_photons = initoutputdata.inputParam.getnum_photons();

  i_photon = num_photons;
  //          PunchTime(0, "");

  do {
    initoutputdata.Launch_Photon(Tis.lyrInt);
    do {
      initoutputdata.HopDropSpin(Tis.lyrInt);

    } while (!initoutputdata.photonStruct.getdead());
  } while (--i_photon != 0);
  resData = new ScaleResultData(initoutputdata, Tis);
  jProgressBar1.setValue(100);
  counter=100;
}

}


private void jButton3_actionPerformed(ActionEvent e) {
  new OutputFace(this.initoutputdata, this.resData, this.Tis.lyrInt);
}

private void jButton4_actionPerformed(ActionEvent e) {
  new BeamType(this.initoutputdata.inputParam);
}

private void jButton5_actionPerformed(ActionEvent e) {
  new Resolution(this.initoutputdata.inputParam);
}
```

```
 private void jButton6_actionPerformed(ActionEvent e) {
  new ConvError(this.initoutputdata.inputParam);
 }

 private void jButton7_actionPerformed(ActionEvent e) {


  counter=1;
  jProgressBar1.setValue(1);
//  progressbar.start();
  this.initoutputdata.convMain();
  jProgressBar1.setValue(100);
  counter=100;

 }

 private void jButton8_actionPerformed(ActionEvent e) {
  this.LayerThermalInterfacePri();
 }

 private void jButton9_actionPerformed(ActionEvent e) {
   this.initialtemperature =  new InitialTemperature();
 }

 private void LayerThermalInterfacePri() {
  boolean ambient_medium = false;

  a = this.Tis.a;
  // InputParam.setnum_layers(a);
  // a += 2;

  lyrThrmlInt = new LayerThermalInterface[a];

  for (int i = 0; i < a; i++) {
    if (i == 0 || i == a - 1) {
      ambient_medium = true;
    } else {
      ambient_medium = false;
    }
    lyrThrmlInt[i] = new LayerThermalInterface(i, ambient_medium);
  }
 }

  private void jButton10_actionPerformed(ActionEvent e) {

    ni =  new
LayerThermalData(this.initoutputdata,this.resData,this.Tis.lyrInt,this.lyrThrmlInt,
this.initialtemperature.layerthermalinput,this.jTextArea1);
    ni.start();
```

```
    }

  private void jButton11_actionPerformed(ActionEvent e) {

     GraphicsTher hk = new
GraphicsTher(this.initoutputdata.outParam.getTem_rzt()[Integer.parseInt(jSpinner5.getVal
ue().toString())] ,8);
     Thread kk =  new Thread(hk);
     kk.start();
  }

  private void jButton12_actionPerformed(ActionEvent e) {

     GraphicsTher  jj = new GraphicsTher(this.initoutputdata.outParam.getF_rzc(),8);
     Thread kk =  new Thread(jj);
     kk.start();

    }

}
```

A.1 Monte Carlo

InitOutputData.java

```
package montecarlo.Process;

import montecarlo.Convolution.InputData;

import montecarlo.Face.LayerInterface;

import montecarlo.Param.InputParam;
import montecarlo.Param.OutputParam;
import montecarlo.Param.PhotonStruct;


public class InitOutputData {

   public InputParam inputParam = new InputParam();
   public OutputParam outParam = new OutputParam();
   public PhotonStruct photonStruct = new PhotonStruct();

   public double COSZERO = 1.0 - 1.0E-12;
   public double COS90D = 1.0E-6;
   public int PARTIALREFLECTION = 0;
   public int STANDARDTEST = 0;
   static boolean first_time = true;
   static int idum;
   static double CHANCE = 0.1;
   double MBIG = 1000000000;
   long MSEED = 161803398;
   int MZ = 0;
   float FAC = (float)1.0E-9;
   static int inext, inextp;
   static long[] ma = new long[56];
   static int iff = 0;

   public InitOutputData() {
      this.InitParam();

   }

   private void InitParam() {

      int nz = inputParam.getnz();
      int nr = inputParam.getnr();
      int na = inputParam.getna();
```

```
        int nl = inputParam.getnum_layers();

        outParam.setRsp(0.0);
        outParam.setRd(0.0);
        outParam.setA(0.0);
        outParam.setTt(0.0);

        this.outParam.setRd_ra(new double[nr - 1 - 0 + 1][na - 1 - 0 + 1]);
        this.outParam.setRd_r(new double[nr - 1 - 0 + 1]);
        this.outParam.setRd_a(new double[na - 1 - 0 + 1]);

        this.outParam.setA_rz(new double[nr - 1 - 0 + 1][nz - 1 - 0 + 1]);
        this.outParam.setA_z(new double[nz - 1 - 0 + 1]);
        this.outParam.setA_l(new double[nl + 1 + 1]);

        this.outParam.setTt_ra(new double[nr - 1 - 0 + 1][na - 1 - 0 + 1]);
        this.outParam.setTt_r(new double[nr - 1 - 0 + 1]);
        this.outParam.setTt_a(new double[na - 1 - 0 + 1]);

    }

    private void InitParamConv() {

      int    nz = inputParam.getnz();
      int    nr = (int)inputParam.getnrc();  /* use nrc instead of nr. */
      int    na = inputParam.getna();

      this.outParam.setRd_rac(new double[nr][na - 1 - 0 + 1]);
      this.outParam.setRd_rc(new double[nr]);
      this.outParam.setA_rzc(new double[nr][nz - 1 - 0 + 1]);
      this.outParam.setTt_rc(new double[nr]);
      this.outParam.setTt_rac(new double[nr][na - 1 - 0 + 1]);
      this.outParam.setF_rzc(new double[nz][nr]);

    }

    public double Rspecular(LayerInterface[] lyrInt) {
        double r1, r2;
        double temp;

        temp =
(lyrInt[0].layerdata.getn() - lyrInt[1].layerdata.getn()) / (lyrInt[0].layerdata.getn() +
lyrInt[1].layerdata.getn());
        r1 = temp * temp;

        if ((lyrInt[1].layerdata.getmua() == 0.0) &&
            (lyrInt[1].layerdata.getmus() == 0.0)) {

            temp =
```

```
(lyrInt[1].layerdata.getn() - lyrInt[2].layerdata.getn()) / (lyrInt[1].layerdata.getn() +
                                                    lyrInt[2].layerdata.getn());
            r2 = temp * temp;
            r1 = r1 + (1 - r1) * (1 - r1) * r2 / (1 - r1 * r2);

        }
        return r1;
    }

    public void Launch_Photon(LayerInterface[] lyrInt) {

        this.photonStruct.setw(1.0 - this.Rspecular(lyrInt));
        this.photonStruct.setdead(false);
        this.photonStruct.setlayer(1);
        this.photonStruct.sets(0.0);
        this.photonStruct.setsleft(0.0);

        this.photonStruct.setx(0.0);
        this.photonStruct.sety(0.0);
        this.photonStruct.setz(0.0);
        this.photonStruct.setux(0.0);
        this.photonStruct.setuy(0.0);
        this.photonStruct.setuz(1.0);

        if ((lyrInt[1].layerdata.getmua() == 0.0) &&
            (lyrInt[1].layerdata.getmus() == 0.0) &&
            (lyrInt[1].layerdata.getambient_medium() == false)) {
            this.photonStruct.setlayer(2);
            this.photonStruct.setz(lyrInt[2].layerdata.getz0());
        }

    }

    public void HopDropSpin(LayerInterface[] lyrInt) {

        int layer = this.photonStruct.getlayer();


        if ((lyrInt[layer].layerdata.getmua() == 0.0) &&
            (lyrInt[layer].layerdata.getmus() == 0.0) &&
            (lyrInt[layer].layerdata.getambient_medium() == false)) {

            this.HopInGlass(lyrInt);
        } else {
            this.HopDropSpinInTissue(lyrInt);
        }
        if (this.photonStruct.getw() < this.inputParam.getWth() &&
            !this.photonStruct.getdead())
            this.Roulette();
```

```
    }

    private void HopInGlass(LayerInterface[] lyrInt) {

        // double dl;

        if (this.photonStruct.getuz() == 0.0) {
            this.photonStruct.setdead(true);
        } else {

            this.StepSizeInGlass(lyrInt);
            this.Hop();
            this.CrossOrNot(lyrInt);

        }
    }

    private void StepSizeInGlass(LayerInterface[] lyrInt) {

        double dl_b;
        int layer = this.photonStruct.getlayer();
        double uz = this.photonStruct.getuz();

        if (uz > 0.0) {
            dl_b =
(lyrInt[layer].layerdata.getz1() - this.photonStruct.getz()) / uz;
        } else if (uz < 0.0) {
            dl_b =
(lyrInt[layer].layerdata.getz0() - this.photonStruct.getz()) / uz;
        } else {
            dl_b = 0.0;
        }
        this.photonStruct.sets(dl_b);
    }

    private void Hop() {

        double s = this.photonStruct.gets();

        this.photonStruct.setx(this.photonStruct.getx() +
                        s * this.photonStruct.getux());
        this.photonStruct.sety(this.photonStruct.gety() +
                        s * this.photonStruct.getuy());
        this.photonStruct.setz(this.photonStruct.getz() +
                        s * this.photonStruct.getuz());


    }
```

```
private void CrossOrNot(LayerInterface[] lyrInt) {

   if (this.photonStruct.getuz() < 0.0) {
      CrossUpOrNot(lyrInt);
   } else {
      CrossDnOrNot(lyrInt);
   }
}

private void CrossUpOrNot(LayerInterface[] lyrInt) {

   double uz = this.photonStruct.getuz();
   this.uz1 = 0.0;
   double r = 0.0;
   int layer = this.photonStruct.getlayer();
   double ni = lyrInt[layer].layerdata.getn();
   double nt = lyrInt[layer - 1].layerdata.getn();

   if (-uz <= lyrInt[layer].layerdata.getcos_crit0()) {
      r = 1.0;
   } else {
      r = this.RFresnel(ni, nt, -uz);
   }

   //   System.out.println(" r: "+r);

   if (PARTIALREFLECTION == 1) {

   } else if (RandomNum() > r) {
      //  System.out.println("burdaki if");
      if (layer == 1) {
         //     System.out.println("super ıf");
         this.photonStruct.setuz(-this.uz1);
         this.RecordR(0.0);
         this.photonStruct.setdead(true);
      } else {
         //     System.out.println("layer 1 degılse burda");
         this.photonStruct.setlayer(this.photonStruct.getlayer() - 1);
         this.photonStruct.setux(this.photonStruct.getux() * ni / nt);
         this.photonStruct.setuy(this.photonStruct.getuy() * ni / nt);
         this.photonStruct.setuz(-this.uz1);
      }
   } else {
      this.photonStruct.setuz(-uz);
   }
}

private double RFresnel(double n1, double n2, double ca1) {
```

```
    double r;

    if (n1 == n2) {
       this.uz1 = ca1;
       r = 0.0;
    } else if (ca1 > COSZERO) {
       this.uz1 = ca1;
       r = (n2 - n1) / (n2 + n1);
       r *= r;
    } else if (ca1 < COS90D) {
       this.uz1 = 0.0;
       r = 1.0;
    } else {
       double sa1, sa2;
       double ca2;

       sa1 = Math.sqrt(1 - ca1 * ca1);
       sa2 = n1 * sa1 / n2;
       if (sa2 >= 1.0) {
          this.uz1 = 0.0;
          r = 1.0;
       } else {
          double cap, cam;
          double sap, sam;

          ca2 = Math.sqrt(1 - sa2 * sa2);
          this.uz1 = Math.sqrt(1 - sa2 * sa2);

          cap = ca1 * ca2 - sa1 * sa2;
          cam = ca1 * ca2 + sa1 * sa2;
          sap = sa1 * ca2 + ca1 * sa2;
          sam = sa1 * ca2 - ca1 * sa2;
          r =
0.5 * sam * sam * (cam * cam + cap * cap) / (sap * sap * cam * cam);

       }

    }

    return r;
  }

  private void RecordR(double Refl) {
    double x = this.photonStruct.getx();
    double y = this.photonStruct.gety();
    int ir, ia;

    ir = (int)(Math.sqrt(x * x + y * y) / this.inputParam.getdr());
```

```
      if (ir > (this.inputParam.getnr() - 1)) {
         ir = this.inputParam.getnr() - 1;
      }

      ia =
(int)(Math.acos(-this.photonStruct.getuz()) / this.inputParam.getda());

      if (ia > (this.inputParam.getna() - 1)) {
         ia = this.inputParam.getna() - 1;
      }

      this.outParam.setRd_ra(this.outParam.getRd_ra()[ir][ia] +
                        this.photonStruct.getw() * (1.0 - Refl), ir,
                        ia);

      this.photonStruct.setw(this.photonStruct.getw() * Refl);


   }

   private double RandomNum() {
      double gok = 0;
      // Date hun = new Date();
      if (first_time) {

         // idum = -(int) hun.getTime()/1000%(1<<15);
         idum = -24221;
         ran3(idum);
         first_time = false;
         idum = 1;
      }

      gok = ran3(idum);
      return (gok);


   }

   private float ran3(int idum) {
      float kar = 0;
      long mj, mk;
      int i, ii, k;

      if (idum < 0 || iff == 0) {
         iff = 1;
         mj = MSEED - (idum < 0 ? -idum : idum);
         mj %= MBIG;
         ma[55] = mj;
         mk = 1;
```

```
            for (i = 1; i <= 54; i++) {
                ii = (21 * i) % 55;
                ma[ii] = mk;
                mk = mj - mk;
                if (mk < MZ)
                    mk += MBIG;
                mj = ma[ii];
            }
            for (k = 1; k <= 4; k++)
                for (i = 1; i <= 55; i++) {
                    ma[i] -= ma[1 + (i + 30) % 55];
                    if (ma[i] < MZ)
                        ma[i] += MBIG;
                }
            inext = 0;
            inextp = 31;
            idum = 1;
        }
        if (++inext == 56)
            inext = 1;
        if (++inextp == 56)
            inextp = 1;
        mj = ma[inext] - ma[inextp];
        if (mj < MZ)
            mj += MBIG;
        ma[inext] = mj;
        kar = mj * FAC;
        return kar;
    }

    double uz1;

    private void CrossDnOrNot(LayerInterface[] lyrInt) {

        double uz = this.photonStruct.getuz();
        this.uz1 = 0.0;
        double r = 0.0;
        int layer = this.photonStruct.getlayer();
        double ni = lyrInt[layer].layerdata.getn();
        double nt = lyrInt[layer + 1].layerdata.getn();

        if (uz <= lyrInt[layer].layerdata.getcos_crit1()) {
            r = 1.0;
        } else {
            r = RFresnel(ni, nt, uz);
        }

        // System.out.println("rR : "+ r);
```

```
    if (PARTIALREFLECTION == 1) {

    } else {
      if (RandomNum() > r) {
        // System.out.println("1 ıncı else");
        if (layer == InputParam.getnum_layers()) {
          //  System.out.println("2 ıncı else");
          this.photonStruct.setuz(uz1);
          RecordT(0.0);
          this.photonStruct.setdead(true);
        } else {
          this.photonStruct.setlayer(this.photonStruct.getlayer() +
                                     1);
          this.photonStruct.setux(this.photonStruct.getux() * ni /
                                  nt);
          this.photonStruct.setuy(this.photonStruct.getuy() * ni /
                                  nt);
          this.photonStruct.setuz(this.uz1);
        }
      } else {

        this.photonStruct.setuz(-uz);
      }
    }
  }

  private void RecordT(double Refl) {

    double x = this.photonStruct.getx();
    double y = this.photonStruct.gety();
    int ir, ia;

    ir = (int)(Math.sqrt(x * x + y * y) / this.inputParam.getdr());
    if (ir > this.inputParam.getnr() - 1) {
      ir = this.inputParam.getnr() - 1;
    }
    ia =
(int)(Math.acos(this.photonStruct.getuz()) / this.inputParam.getda());
    if (ia > this.inputParam.getna() - 1) {
      ia = this.inputParam.getna() - 1;
    }
    this.outParam.setTr_ra(this.outParam.getTt_ra(ir, ia) +
                           this.photonStruct.getw() * (1.0 - Refl), ir,
                           ia);
    this.photonStruct.setw(this.photonStruct.getw() * Refl);
  }

  private void HopDropSpinInTissue(LayerInterface[] lyrInt) {
```

```
        StepSizeInTissue(lyrInt);

        if (HitBoundary(lyrInt)) {
            Hop();
            CrossOrNot(lyrInt);
        } else {
            Hop();
            Drop(lyrInt);
            Spin(lyrInt[this.photonStruct.getlayer()].layerdata.getg());
        }
    }

    private void StepSizeInTissue(LayerInterface[] lyrInt) {

        int layer = this.photonStruct.getlayer();
        double mua = lyrInt[layer].layerdata.getmua();
        double mus = lyrInt[layer].layerdata.getmus();

        if (this.photonStruct.getsleft() == 0.0) {
            double rnd;
            do
                rnd = RandomNum();
            while (rnd <= 0.0);
            this.photonStruct.sets(-Math.log(rnd) / (mua + mus));
        } else {
            this.photonStruct.sets(this.photonStruct.getsleft() / (mua + mus));
            this.photonStruct.setsleft(00);
        }
    }


    private boolean HitBoundary(LayerInterface[] lyrInt) {

        double dl_b = 0.0;
        int layer = this.photonStruct.getlayer();
        double uz = this.photonStruct.getuz();
        boolean hit;
        /*  if (uz==0.8846732254854901) {
         fok= fok++;
       }
       System.out.println("uz   : "+ uz + "layer : " + layer );
       */
        if (uz > 0.0) {
            dl_b =
((lyrInt[layer].layerdata.getz1() - this.photonStruct.getz()) / uz);
        } else if (uz < 0.0)
            dl_b =
((lyrInt[layer].layerdata.getz0() - this.photonStruct.getz()) / uz);
```

```
    if (uz != 0.0 && this.photonStruct.gets() > dl_b) {
        double mut =
            lyrInt[layer].layerdata.getmua() + lyrInt[layer].layerdata.getmus();
        this.photonStruct.setsleft((this.photonStruct.gets() - dl_b) *
                                   mut);
        this.photonStruct.sets(dl_b);
        hit = true;
    } else {
        hit = false;
    }
    return (hit);
}
static double i;

private void Drop(LayerInterface[] lyrInt) {

    double dwa;
    double x = this.photonStruct.getx();
    double y = this.photonStruct.gety();
    int iz, ir;
    int layer = this.photonStruct.getlayer();
    double mua, mus;

    iz = (int)(this.photonStruct.getz() / this.inputParam.getdz());
    if (iz > this.inputParam.getnz() - 1) {
        iz = this.inputParam.getnz() - 1;
    }
    ir = (int)(Math.sqrt(x * x + y * y) / this.inputParam.getdr());

    if (ir > this.inputParam.getnr() - 1) {
        ir = this.inputParam.getnr() - 1;
    }
    mua = lyrInt[layer].layerdata.getmua();
    mus = lyrInt[layer].layerdata.getmus();
    dwa = this.photonStruct.getw() * mua / (mua + mus);
    this.photonStruct.setw(this.photonStruct.getw() - dwa);
    if (ir == 0 && iz == 0) {
        i += 1;
        //  System.out.println(this.outParam.getA_rz(ir,iz)+ "    " +i);

    }

    this.outParam.setA_rz(this.outParam.getA_rz(ir, iz) + dwa, ir, iz);
}

private void Spin(double g) {

    double cost, sint;
    double cosp, sinp;
```

```
    double ux = this.photonStruct.getux();
    double uy = this.photonStruct.getuy();
    double uz = this.photonStruct.getuz();
    double psi;

    cost = SpinTheta(g);
    sint = Math.sqrt(1.0 - cost * cost);
    psi = 2.0 * Math.PI * RandomNum();
    cosp = Math.cos(psi);
    if (psi < Math.PI)
       sinp = Math.sqrt(1.0 - cosp * cosp);
    else
       sinp = -Math.sqrt(1.0 - cosp * cosp);

    if (Math.abs(uz) > COSZERO) {
       this.photonStruct.setux(sint * cosp);
       this.photonStruct.setuy(sint * sinp);
       this.photonStruct.setuz(cost * Math.signum(uz));


    } else {
       double temp = Math.sqrt(1.0 - uz * uz);
       this.photonStruct.setux(sint * (ux * uz * cosp - uy * sinp) /
                          temp + ux * cost);
       this.photonStruct.setuy(sint * (uy * uz * cosp + ux * sinp) /
                          temp + uy * cost);
       this.photonStruct.setuz(-sint * cosp * temp + uz * cost);
    }
}

private double SpinTheta(double g) {

    double cost;

    if (g == 0.0)
       cost = 2 * RandomNum() - 1;
    else {
       double temp = (1 - g * g) / (1 - g + 2 * g * this.RandomNum());
       cost = (1 + g * g - temp * temp) / (2 * g);

       if (cost < -1) {
          cost = -1;
       } else if (cost > 1) {
          cost = 1;
       }
    }
    return (cost);
}
```

```
    private void Roulette() {
        if (this.photonStruct.getw() == 0.0)
            this.photonStruct.setdead(true);
        else if (this.RandomNum() < CHANCE) /* survived the roulette.*/
            this.photonStruct.setw(this.photonStruct.getw() / CHANCE);
        else
            photonStruct.setdead(true);
    }

    public void convMain() {
        this.InitParamConv();
        InputData indata = new InputData(inputParam, outParam);

    }
}
```

LayerInterface.java

```
package montecarlo.Face;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSeparator;
import javax.swing.JSpinner;
import javax.swing.SpinnerModel;
import javax.swing.SpinnerNumberModel;
import javax.swing.SwingConstants;

import montecarlo.Param.LayerStruct;


public class LayerInterface extends JFrame {

    private JSeparator jSeparator1 = new JSeparator();
    private JSeparator jSeparator2 = new JSeparator();
```

```
private JSeparator jSeparator5 = new JSeparator();
private JSeparator jSeparator8 = new JSeparator();
private JLabel jLabel2 = new JLabel();
private JLabel jLabel4 = new JLabel();
private JLabel jLabel5 = new JLabel();
private JLabel jLabel6 = new JLabel();
private JLabel jLabel7 = new JLabel();
private JLabel jLabel1 = new JLabel();
private JLabel jLabel3 = new JLabel();

int initValue = 0;
int min = 0;
int max = 1000;
double step = 0.01;

SpinnerModel model1 = new SpinnerNumberModel(initValue, min, max, step);
SpinnerModel model2 = new SpinnerNumberModel(initValue, min, max, step);
SpinnerModel model3 = new SpinnerNumberModel(initValue, min, max, step);
SpinnerModel model4 = new SpinnerNumberModel(initValue, min, max, step);
SpinnerModel model5 = new SpinnerNumberModel(initValue, min, max, step);

private JSpinner jSpinner1 = new JSpinner(model1);
private JSpinner jSpinner2 = new JSpinner(model2);
private JSpinner jSpinner4 = new JSpinner(model4);
private JSpinner jSpinner3 = new JSpinner(model3);
private JSpinner jSpinner5 = new JSpinner(model5);

private int i;
private JButton jButton1 = new JButton();

public LayerStruct layerdata;
private boolean ambient_medium = false;


public LayerInterface(int iw, boolean ambient) {

   this.ambient_medium = ambient;
   jLabel2.setText(iw + 1 + ". Layer");

   this.i = iw;
   try {
      jbInit();
   } catch (Exception e) {
      e.printStackTrace();
   }
}

public LayerInterface() {
   try {
```

```
        jbInit();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void jbInit() throws Exception {

    this.getContentPane().setLayout(null);
    this.setTitle("Tissue Parameters ");
    this.setSize(new Dimension(333, 366));
    jSeparator1.setBounds(new Rectangle(10, 295, 290, 5));
    jSeparator2.setBounds(new Rectangle(10, 40, 290, 5));
    jSeparator5.setBounds(new Rectangle(300, 40, 5, 255));
    jSeparator5.setOrientation(SwingConstants.VERTICAL);
    jSeparator8.setBounds(new Rectangle(10, 40, 5, 255));
    jSeparator8.setOrientation(SwingConstants.VERTICAL);
    jLabel2.setBounds(new Rectangle(20, 40, 110, 20));
    jLabel2.setToolTipText("null");
    jLabel4.setText("Refractive Index :");
    jLabel4.setBounds(new Rectangle(20, 70, 110, 20));
    jLabel4.setToolTipText("Refractive Index :");
    jLabel5.setText("Absorption Coefficient :");
    jLabel5.setBounds(new Rectangle(20, 105, 140, 20));
    jLabel5.setToolTipText("Absorption Coefficient :");
    jLabel6.setText("Scattering Coefficient :");
    jLabel6.setBounds(new Rectangle(20, 140, 130, 20));
    jLabel6.setToolTipText("Scattering Coefficient :");
    jLabel7.setText("Anisotropy Factor :");
    jLabel7.setBounds(new Rectangle(20, 175, 110, 20));
    jLabel7.setToolTipText("Anisotropy Factor :");
    jSpinner2.setBounds(new Rectangle(160, 105, 110, 20));
    jSpinner2.setBorder(BorderFactory.createLineBorder(Color.black, 1));
    jSpinner4.setBounds(new Rectangle(160, 175, 110, 20));
    jSpinner4.setBorder(BorderFactory.createLineBorder(Color.black, 1));
    jSpinner3.setBounds(new Rectangle(160, 140, 110, 20));
    jSpinner3.setBorder(BorderFactory.createLineBorder(Color.black, 1));
    jSpinner1.setBounds(new Rectangle(160, 70, 110, 20));
    jSpinner1.setBorder(BorderFactory.createLineBorder(Color.black, 1));
    jSpinner1.setToolTipText("null");
    jSpinner1.setFocusable(false);
    jSpinner1.setVerifyInputWhenFocusTarget(false);
    jButton1.setText("Save");
    jButton1.setBounds(new Rectangle(185, 260, 95, 20));
    jButton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                jButton1_actionPerformed(e);
            }
        });
```

```
jLabel1.setBounds(new Rectangle(15, 275, 295, 20));
jLabel3.setText("Thickness : ");
jLabel3.setBounds(new Rectangle(20, 210, 105, 25));
jSpinner5.setBounds(new Rectangle(160, 215, 110, 20));
jSpinner5.setBorder(BorderFactory.createLineBorder(Color.black, 1));
this.getContentPane().add(jSpinner5, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(jLabel1, null);
this.getContentPane().add(jButton1, null);
this.getContentPane().add(jSpinner1, null);
this.getContentPane().add(jSpinner3, null);
this.getContentPane().add(jSpinner4, null);
this.getContentPane().add(jSpinner2, null);
this.getContentPane().add(jLabel7, null);
this.getContentPane().add(jLabel6, null);
this.getContentPane().add(jLabel5, null);
this.getContentPane().add(jLabel4, null);
this.getContentPane().add(jLabel2, null);
this.getContentPane().add(jSeparator8, null);
this.getContentPane().add(jSeparator5, null);
this.getContentPane().add(jSeparator2, null);
this.getContentPane().add(jSeparator1, null);
this.setVisible(true);

this.setResizable(false);
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = this.getSize();

this.setLocation((screenSize.width - frameSize.width) / 2 + i * 30,
            (screenSize.height - frameSize.height) / 2 + i * 30);

this.addWindowListener(new WindowAdapter() {

          public void windowClosing(WindowEvent e) {

          }
       });

if (i == 1) {
   this.jSpinner1.setValue(1.3);
   this.jSpinner2.setValue(3.2);
   this.jSpinner3.setValue(414);
   this.jSpinner4.setValue(0.95);
   this.jSpinner5.setValue(0.1);
} else if (i == 2) {
   this.jSpinner1.setValue(1.3);
   this.jSpinner2.setValue(5);
   this.jSpinner3.setValue(134);
   this.jSpinner4.setValue(0.90);
```

```
        this.jSpinner5.setValue(0.06);

    } else if (i == 3) {
        this.jSpinner1.setValue(1.37);
        this.jSpinner2.setValue(2);
        this.jSpinner3.setValue(10);
        this.jSpinner4.setValue(0.7);
        this.jSpinner5.setValue(0.2);
    }


    if (this.ambient_medium == true) {
        this.jSpinner1.setValue(1);
        this.jSpinner2.setValue(0);
        this.jSpinner3.setValue(0);
        this.jSpinner4.setValue(0);
        this.jSpinner5.setValue(0);
        this.jSpinner1.setEnabled(false);
        this.jSpinner2.setEnabled(false);
        this.jSpinner3.setEnabled(false);
        this.jSpinner4.setEnabled(false);
        this.jSpinner5.setEnabled(false);
        this.jButton1.setEnabled(false);

        this.saveButton();
    }

}
//

private void jButton1_actionPerformed(ActionEvent e) {

    this.saveButton();
}

private void saveButton() {

    this.layerdata = new LayerStruct();
    this.layerdata.setn((Double.parseDouble(jSpinner1.getValue().toString())));
    this.layerdata.setmua((Double.parseDouble(jSpinner2.getValue().toString())));
    this.layerdata.setmus((Double.parseDouble(jSpinner3.getValue().toString())));
    this.layerdata.setg((Double.parseDouble(jSpinner4.getValue().toString())));
    this.layerdata.setd((Double.parseDouble(jSpinner5.getValue().toString())));
    this.layerdata.setambient_medium(this.ambient_medium);

    this.layerdata.setz0(LayerStruct.getZ_Ptr());
    LayerStruct.setZ_Ptr(LayerStruct.getZ_Ptr() + this.layerdata.getd());
    this.layerdata.setz1(LayerStruct.getZ_Ptr());
```

```
        System.out.println(layerdata.getn());
        System.out.println(layerdata.getmua());
        System.out.println(layerdata.getmus());
        System.out.println(layerdata.getg());
        System.out.println(layerdata.getd());

        if (this.ambient_medium == false) {
            this.setVisible(false);
        }
    }

}
```

A.2 Convolution

InputData.java

```
package montecarlo.Convolution;

import montecarlo.Convolution.Face.ConvStruct;

import montecarlo.Param.ConvVar;
import montecarlo.Param.InputParam;
import montecarlo.Param.OutputParam;


public class InputData {

   private InputParam inputParam;
   private OutputParam outputParam;
   private ConvVar convvar;
   private static final int JMAX = 30;
   private static final int GAUSSLIMIT = 4;
   private static double s ;
   private static int it;

   //public BinaryNode Link = new BinaryNode();

   public InputData(InputParam inputParam, OutputParam outputParam) {
      this.inputParam = inputParam;
      this.outputParam = outputParam;
      this.convvar = new ConvVar(inputParam, outputParam);
      this.outputParam.conved = new ConvStruct();
      this.BranchOutConvCmd();
   }

   private void BranchOutConvCmd() {

      this.BranchOutConvA();
      this.BranchOutConvF();
      this.BranchOutConvR();
      this.BranchOutConvT();

  //    TreeSet<double[][]> q2=new TreeSet<double[][]>();


   }

   private void BranchOutConvA() {
```

```
   this.ConvA_rz();
   this.WriteA_rzc();
}

private void ConvA_rz() {

   int irc, iz;
   double rc;
   double P = inputParam.beamstruct.getP();
   double R = inputParam.beamstruct.getR();

   for (irc = 0; irc < inputParam.getnrc(); irc++) {
      rc = (irc + 0.5) * inputParam.getdrc();
      convvar.setr(rc);
      for (iz = 0; iz < inputParam.getnz(); iz++) {
         convvar.setiz(iz);
         if (inputParam.beamstruct.getFlatGaussian() == "flat") {
            outputParam.setA_rzc(2 * P / (R * R) *
                        FlatIntegration("A_rzFGIntegrand"),
                        irc, iz);
         } else {
            outputParam.setA_rzc(4 * P / (R * R) *
                        GaussIntegration("A_rzFGIntegrand"),
                        irc, iz);
         }
      }
      convvar.tree.clear();
   }
   outputParam.conved.setA_rz(false);
}


private void WriteA_rzc() {

}

private double FlatIntegration(String func) {

   double rc = convvar.getr();
   double R = convvar.inputParam.beamstruct.getR();
   double b_max =
      (convvar.inputParam.getnr() - 0.5) * convvar.inputParam.getdr();
   double a = Math.max(0, rc - R);
   double b = Math.min(b_max, rc + R);

   if (a >= b)
      return (0);
   else
      return (this.qtrap(func, a, b, convvar.inputParam.geteps()));
```

```
}

private double qtrap(String func, double a, double b, double EPS) {

   int j;
   double s = 0, s_old = 0;
   for (j = 1; j <= JMAX; j++) {
      s = this.trapzd(func, a, b, j);
      if (j <= 3 ||
         Math.abs(s - s_old) > convvar.inputParam.geteps() * Math.abs(s_old))
         s_old = s;
      else
         break;
   }
   return (s);

}

private double trapzd(String func, double a, double b, double n) {

   double x, tnm, sum, del;
   int j;
   if (n == 1) {
      it = 1;
      if (func=="A_rzFGIntegrand"){
                  s = 0.5 * (b - a) * (A_rzFGIntegrand(a) + A_rzFGIntegrand(b));
               }else if (func=="Rd_rFGIntegrand"){
                  s = 0.5 * (b - a) * (Rd_rFGIntegrand(a)  + Rd_rFGIntegrand(b));
               }else if (func=="Rd_raFGIntegrand"){
                  s = 0.5 * (b - a) * (Rd_raFGIntegrand(a)  + Rd_raFGIntegrand(b));
               }else if (func=="Tt_rFGIntegrand"){
                  s = 0.5 * (b - a) * (Tt_rFGIntegrand(a)  + Tt_rFGIntegrand(b));
               }else if (func=="Tt_raFGIntegrand"){
                  s = 0.5 * (b - a) * (Tt_raFGIntegrand(a)  + Tt_raFGIntegrand(b));
               }

   } else {
      tnm = it;
      del = (b - a) / tnm;
      x = a + 0.5 * del;
      for (sum = 0.0, j = 1; j <= it; j++, x += del)
         sum += A_rzFGIntegrand(x);
      it *= 2;
      s = 0.5 * (s + (b - a) * sum / tnm);
   }
   return (s);
}
```

```
private double A_rzFGIntegrand(double r2) {
   double f;
   int nr = convvar.inputParam.getnr();
   double R, r, A_at_r2;

   A_at_r2 = A_rzInterp(convvar.outParam.getA_rz(), r2);
   R = convvar.inputParam.beamstruct.getR();
   r = convvar.getr();

   if ((convvar.tree.containsKey(r2) )) {
     f= convvar.tree.get(r2);
      //f = 0; //link->y;
   } else {
      if (convvar.inputParam.beamstruct.getFlatGaussian() == "flat") {
         f = ITheta(r, r2, R);
      } else {
         f = ExpBessI0(r, r2, R);
      }
      convvar.tree.put(r2,f);
   }
   f *= A_at_r2 * r2;
   return (f);
}


private double A_rzInterp(double[][] A_rz, double r2) {
   double ir2, A_lo, A_hi, A_at_r2;
   int iz, ir2lo, nr = convvar.inputParam.getnr();

   ir2 = r2 / convvar.inputParam.getdr();
   iz = convvar.getiz();
   if (nr < 3)
      A_at_r2 = A_rz[0][iz];
   else if (ir2 < nr - 1.5) {
      ir2lo = Math.max(0, (int)(ir2 - 0.5));
      A_lo = A_rz[ir2lo][iz];
      A_hi = A_rz[ir2lo + 1][iz];
      A_at_r2 = A_lo + (A_hi - A_lo) * (ir2 - ir2lo - 0.5);
   } else {
      ir2lo = nr - 3;
      A_lo = A_rz[ir2lo][iz];
      A_hi = A_rz[ir2lo + 1][iz];
      if (A_lo >= A_hi)
         A_at_r2 = A_lo + (A_hi - A_lo) * (ir2 - ir2lo - 0.5);
      else
         A_at_r2 = 0.0;
   }
   return (Math.max(0, A_at_r2));
}
```

```
    private double ITheta(double r, double r2, double R) {
        double temp;
        if (R >= r + r2)
            temp = 1;
        else if (Math.abs(r - r2) <= R) {
            temp = (r * r + r2 * r2 - R * R) / (2 * r * r2);
            if (Math.abs(temp) > 1)
                temp = Math.signum(temp);
            temp = Math.acos(temp) / Math.PI;
        } else
            temp = 0;

        return (temp);
    }

    private double ExpBessI0(double r, double r2, double R) {
        double expbess;
        double _RR = 1 / (R * R);
        double x = 4 * r * r2 * _RR;
        double y = 2 * (r2 * r2 + r * r) * _RR;

        expbess = Math.exp(-y + x) * BessI0(x);
        return (expbess);
    }

    private double BessI0(double x) {
        double ax, ans;
        double y;

        if ((ax = Math.abs(x)) < 3.75) {
            y = x / 3.75;
            y *= y;
            ans =
Math.exp(-ax) * (1.0 + y * (3.5156229 + y * (3.0899424 + y * (1.2067492 +
                                                y * (0.2659732 +
                                                    y *
                                                    (0.360768e-1 +
                                                     y *
                                                     0.45813e-2))))));
        } else {
            y = 3.75 / ax;
            ans =
(1 / Math.sqrt(ax)) * (0.39894228 + y * (0.1328592e-1 + y * (0.225319e-2 +
                                               y * (-0.157565e-2 +
                                                   y *
                                                   (0.916281e-2 +
                                                    y *
                                                    (-0.2057706e-1 +
                                                     y *
```

```
                                        (0.2635537e-1 +
                                         y *
                                         (-0.1647633e-1 +
                                          y *
                                          0.392377e-2))))))));
    }
    return ans;
}


private void BranchOutConvF() {

    this.ConvA_rz();
    this.WriteA_rzc();
}


private void WriteF_rzc() {

}

private void BranchOutConvR() {
    this.ConvRd_r();
    this.WriteRd_rc();
    this.ConvRd_ra();
    this.WriteRd_rac();
}

private void ConvRd_r() {

    int irc;
    double rc;
    double P = inputParam.beamstruct.getP();
    double R = inputParam.beamstruct.getR();
    double b_max = (inputParam.getnr() - 1) * inputParam.getdr();

    for (irc = 0; irc < inputParam.getnrc(); irc++) {
        rc = (irc + 0.5) * inputParam.getdrc();
        convvar.setr(rc);

        if (inputParam.beamstruct.getFlatGaussian() == "flat") {
            outputParam.setRd_rc(2 * P / (R * R) *
                        FlatIntegration("Rd_rFGIntegrand"), irc);
        } else {
            outputParam.setRd_rc(4 * P / (R * R) *
                        GaussIntegration("Rd_rFGIntegrand"), irc);
        }
    }
```

```
    outputParam.conved.setRd_r(false);

}

private void WriteRd_rc() {

}

private double Rd_rFGIntegrand(double r2) {

    double f, R, r, Rd_at_r2;
    int nr = inputParam.getnr();

    Rd_at_r2 = this.RT_rInterp(outputParam.getRd_r(), r2);
    R = inputParam.beamstruct.getR();
    r = convvar.getr();
    if (convvar.inputParam.beamstruct.getFlatGaussian() == "flat")
        f = Rd_at_r2 * ITheta(r, r2, R) * r2;
    else
        f = Rd_at_r2 * ExpBessI0(r, r2, R) * r2;
    return (f);
}

private double RT_rInterp(double[] RT_r, double r2) {

    double ir2, RT_lo, RT_hi, RT_at_r2;
    int ir2lo, nr = inputParam.getnr();

    ir2 = r2 / inputParam.getdr();

    if (nr < 3)
        RT_at_r2 = RT_r[0];
    else if (ir2 < nr - 1.5) {
        ir2lo = Math.max(0, (int)(ir2 - 0.5));
        RT_lo = RT_r[ir2lo];
        RT_hi = RT_r[ir2lo + 1];
        RT_at_r2 = RT_lo + (RT_hi - RT_lo) * (ir2 - ir2lo - 0.5);
    } else {
        ir2lo = nr - 3;
        RT_lo = RT_r[ir2lo];
        RT_hi = RT_r[ir2lo + 1];
        if (RT_lo >= RT_hi)
            RT_at_r2 = RT_lo + (RT_hi - RT_lo) * (ir2 - ir2lo - 0.5);
        else
            RT_at_r2 = 0.0;
    }

    return (Math.max(0, RT_at_r2));
}
```

```
private void ConvRd_ra() {

   int irc, ia;
   double rc;
   double P = inputParam.beamstruct.getP();
   double R = inputParam.beamstruct.getR();
   double b_max = (inputParam.getnr() - 1) * inputParam.getdr();

   for (irc = 0; irc < inputParam.getnrc(); irc++) {
      rc = (irc + 0.5) * inputParam.getdrc();
      convvar.setr(rc);
      //convvar.TREE = new BinarySearchTree();


      for (ia = 0; ia < inputParam.getna(); ia++) {
         convvar.setia(ia);
         if (inputParam.beamstruct.getFlatGaussian() == "flat") {
            outputParam.setRd_rac(2 * P / (R * R) *
                        FlatIntegration("Rd_raFGIntegrand"),
                        irc, ia);
         } else {
            outputParam.setRd_rac(4 * P / (R * R) *
                        GaussIntegration("Rd_raFGIntegrand"),
                        irc, ia);
         }
      }

      convvar.tree.clear();

   }
   outputParam.conved.setRd_ra(false);
}

private void WriteRd_rac() {

}

private double Rd_raFGIntegrand(double r2) {
   double f;
   int nr = convvar.inputParam.getnr();
   double R, r, Rd_at_r2;

   Rd_at_r2 = RT_raInterp(convvar.outParam.getRd_ra(), r2);

   R = convvar.inputParam.beamstruct.getR();
   r = convvar.getr();
  if ((convvar.tree.containsKey(r2) )) {
          f= convvar.tree.get(r2);
           //f = 0; //link->y;
```

```
  }else {
      if (convvar.inputParam.beamstruct.getFlatGaussian() == "flat")
         f = ITheta(r, r2, R);
      else
         f = ExpBessI0(r, r2, R);
      convvar.tree.put(r2,f);
   }
   f *= Rd_at_r2 * r2;
   return (f);
}

private double RT_raInterp(double[][] RT_ra, double r2) {

   double ir2, RT_lo, RT_hi, RT_at_r2;
   int ia, ir2lo, nr = convvar.inputParam.getnr();

   ir2 = r2 / convvar.inputParam.getdr();
   ia = convvar.getia();
   if (nr < 3)
      RT_at_r2 = RT_ra[0][ia];
   else if (ir2 < nr - 1.5) {
      ir2lo = Math.max(0, (int)(ir2 - 0.5));
      RT_lo = RT_ra[ir2lo][ia];
      RT_hi = RT_ra[ir2lo + 1][ia];
      RT_at_r2 = RT_lo + (RT_hi - RT_lo) * (ir2 - ir2lo - 0.5);
   } else {
      ir2lo = nr - 3;
      RT_lo = RT_ra[ir2lo][ia];
      RT_hi = RT_ra[ir2lo + 1][ia];
      if (RT_lo >= RT_hi)
         RT_at_r2 = RT_lo + (RT_hi - RT_lo) * (ir2 - ir2lo - 0.5);
      else
         RT_at_r2 = 0.0;
   }
   return (Math.max(0, RT_at_r2));
}

private void BranchOutConvT() {
   this.ConvTt_r();
   this.WriteTt_rc();
   this.ConvTt_ra();
   this.WriteTt_rac();
}

private void ConvTt_r() {

   int irc;
   double rc;
   double P = inputParam.beamstruct.getP();
```

```
    double R = inputParam.beamstruct.getR();
    double b_max = (inputParam.getnr() - 1) * inputParam.getdr();
    for (irc = 0; irc < inputParam.getnrc(); irc++) {
        rc = (irc + 0.5) * inputParam.getdrc();
        convvar.setr(rc);

        if (inputParam.beamstruct.getFlatGaussian() == "flat")

            outputParam.setTt_rc(2 * P / (R * R) *
                        FlatIntegration("Tt_rFGIntegrand"), irc);


        else {
            outputParam.setTt_rc(4 * P / (R * R) *
                        GaussIntegration("Tt_rFGIntegrand"), irc);
        }
    }
    outputParam.conved.setTt_r(false);
}

private void WriteTt_rc() {

}

private double Tt_rFGIntegrand(double r2) {

    double f, R, r, Tt_at_r2;
    int nr = convvar.inputParam.getnr();

    Tt_at_r2 = this.RT_rInterp(outputParam.getTt_r(), r2);
    R = inputParam.beamstruct.getR();
    r = convvar.getr();
    if (convvar.inputParam.beamstruct.getFlatGaussian() == "flat") {
        f = Tt_at_r2 * ITheta(r, r2, R) * r2;
    } else {
        f = Tt_at_r2 * ExpBessI0(r, r2, R) * r2;
    }
    return (f);
}

private void ConvTt_ra() {

    int irc, ia;
    double rc;
    double P = inputParam.beamstruct.getP();
    double R = inputParam.beamstruct.getR();
    double b_max = (inputParam.getnr() - 1) * inputParam.getdr();

    for (irc = 0; irc < inputParam.getnrc(); irc++) {
```

```
        rc = (irc + 0.5) * inputParam.getdrc();
        convvar.setr(rc);
        //convvar.TREE = new BinarySearchTree();


        for (ia = 0; ia < inputParam.getna(); ia++) {
           convvar.setia(ia);
           if (inputParam.beamstruct.getFlatGaussian() == "flat") {
              outputParam.setTt_rac(2 * P / (R * R) *
                                    FlatIntegration("Tt_raFGIntegrand"),
                                    irc, ia);
           } else {
              outputParam.setTt_rac(4 * P / (R * R) *
                                    GaussIntegration("Tt_raFGIntegrand"),
                                    irc, ia);
           }
        }
        convvar.tree.clear();
     }
     outputParam.conved.setTt_ra(false);
  }

  private void WriteTt_rac() {

  }

  private double Tt_raFGIntegrand(double r2) {
     double f;
     int nr = convvar.inputParam.getnr();
     double R, r, Tt_at_r2;

     Tt_at_r2 = RT_raInterp(convvar.outParam.getTt_ra(), r2);
     R = convvar.inputParam.beamstruct.getR();
     r = convvar.getr();
   if ((convvar.tree.containsKey(r2) )) {
            f= convvar.tree.get(r2);
             //f = 0; //link->y;
   }else {
        if (convvar.inputParam.beamstruct.getFlatGaussian() == "flat")
           f = ITheta(r, r2, R);
        else
           f = ExpBessI0(r, r2, R);
        convvar.tree.put(r2,f);
     }
     f *= Tt_at_r2 * r2;
     return (f);
  }

  private double GaussIntegration(String func) {
```

```
        double rc = convvar.getr();
        double R = convvar.inputParam.beamstruct.getR();
        double b_max =
           (convvar.inputParam.getnr() - 0.5) * convvar.inputParam.getdr();
        double a = Math.max(0, rc - GAUSSLIMIT * R);
        double b = Math.min(b_max, rc + GAUSSLIMIT * R);
        double temp = 0;

        if (a >= b)
           return(0);
        else
           return (this.qtrap(func, a, b, convvar.inputParam.geteps()));

     }


}
```

A.3 Finite Difference Method

InitialTemperature.java

```
package montecarlo.Thermal.Face;


import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSeparator;
import javax.swing.JSpinner;
import javax.swing.SpinnerModel;
import javax.swing.SpinnerNumberModel;

import montecarlo.Param.LayerThermal;
import montecarlo.Param.LayerThermalInput;


public class InitialTemperature extends JFrame {

  public LayerThermalInput layerthermalinput;

  int initValue = 0;
  int min = 0;
  int max = 1000;
  double step = 0.01;


  private SpinnerModel model1 =
    new SpinnerNumberModel(initValue, min, max, step);
  private SpinnerModel model2 =
    new SpinnerNumberModel(initValue, min, max, step);
  private JLabel jLabel1 = new JLabel();
  private JSpinner jSpinner1 = new JSpinner(model1);
  private JSeparator jSeparator1 = new JSeparator();
  private JLabel jLabel2 = new JLabel();
  private JSpinner jSpinner2 = new JSpinner(model2);
  private JButton jButton1 = new JButton();

  public InitialTemperature() {
```

```
  try {
    jbInit();
  } catch (Exception e) {
    e.printStackTrace();
  }
 }

 private void jbInit() throws Exception {
  this.getContentPane().setLayout(null);
  this.setTitle("Initial Temperature");
  this.setSize(new Dimension(425, 292));
  jLabel1.setText("T0 of Tissue");
  jLabel1.setBounds(new Rectangle(30, 45, 175, 30));
  jSpinner1.setBounds(new Rectangle(205, 45, 140, 30));
  jSeparator1.setBounds(new Rectangle(30, 100, 350, 2));
  jLabel2.setText("Temperature of Environment");
  jLabel2.setBounds(new Rectangle(30, 135, 170, 30));
  jSpinner2.setBounds(new Rectangle(205, 130, 140, 30));
  jButton1.setText("Save");
  jButton1.setBounds(new Rectangle(300, 200, 90, 30));
  jButton1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton1_actionPerformed(e);
      }
    });
  this.getContentPane().add(jButton1, null);
  this.getContentPane().add(jSpinner2, null);
  this.getContentPane().add(jLabel2, null);
  this.getContentPane().add(jSeparator1, null);
  this.getContentPane().add(jSpinner1, null);
  this.getContentPane().add(jLabel1, null);
  this.setVisible(true);

  this.jSpinner1.setValue(37);
  this.jSpinner2.setValue(23);
 }

 private void jButton1_actionPerformed(ActionEvent e) {

  this.layerthermalinput = new LayerThermalInput();
  layerthermalinput.setT0(Double.parseDouble(this.jSpinner1.getValue().toString()));
  layerthermalinput.setTe(Double.parseDouble(this.jSpinner2.getValue().toString()));
  this.dispose();

 }
}
```

LayerThermalInterface.java

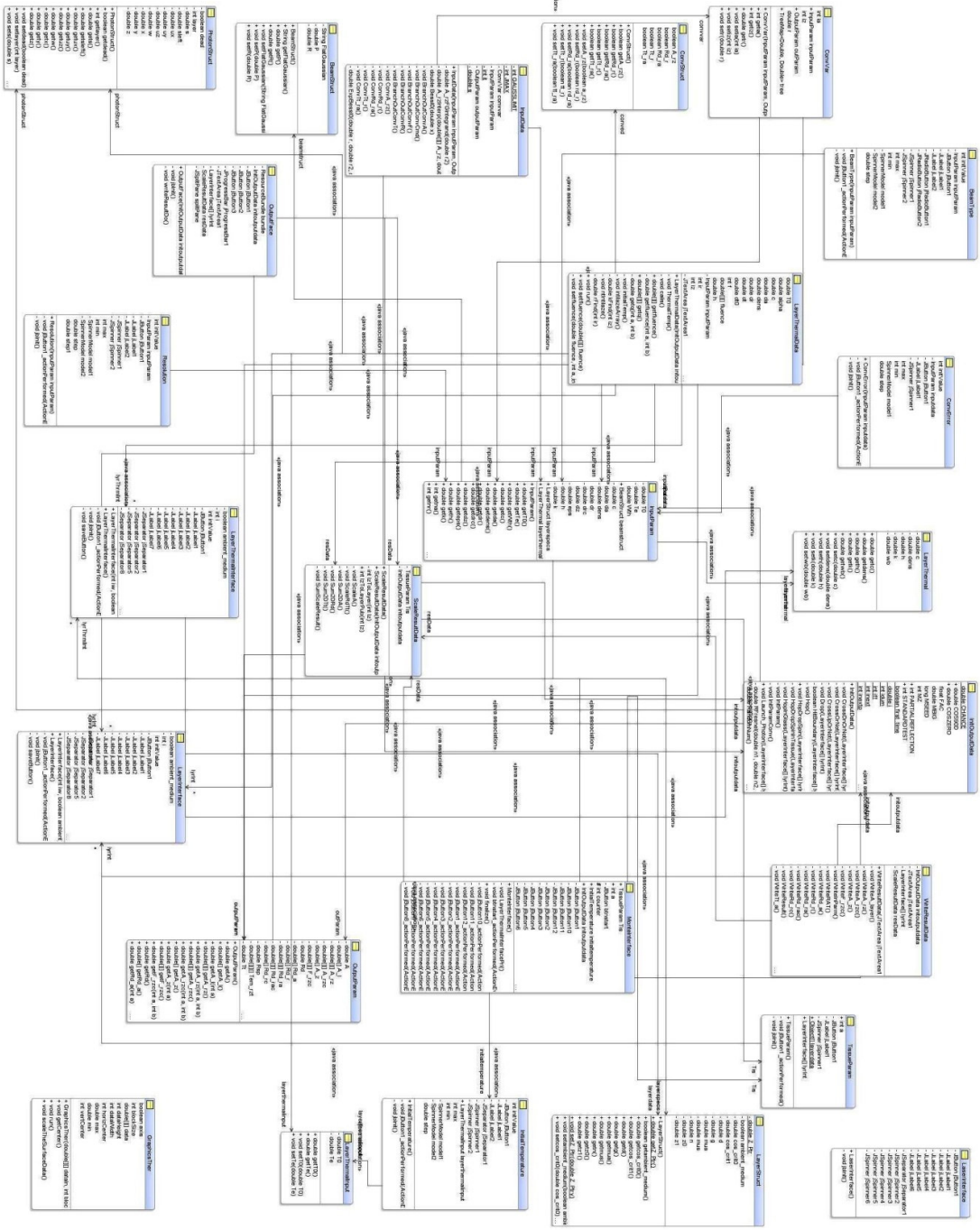
```
package montecarlo.Thermal.Face;


import java.awt.Color;
import java.awt.Dimension;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JSeparator;
import javax.swing.JSpinner;
import javax.swing.SpinnerModel;
import javax.swing.SpinnerNumberModel;
import javax.swing.SwingConstants;

import montecarlo.Param.LayerThermal;


public class LayerThermalInterface extends JFrame {

    private JSeparator jSeparator1 = new JSeparator();
    private JSeparator jSeparator2 = new JSeparator();
    private JSeparator jSeparator5 = new JSeparator();
    private JSeparator jSeparator8 = new JSeparator();
    private JLabel jLabel2 = new JLabel();
    private JLabel jLabel4 = new JLabel();
    private JLabel jLabel5 = new JLabel();
    private JLabel jLabel6 = new JLabel();
    private JLabel jLabel7 = new JLabel();
    private JLabel jLabel1 = new JLabel();
    private JLabel jLabel3 = new JLabel();

    int initValue = 0;
    int min = 0;
    int max = 1000;
    double step = 0.01;

    SpinnerModel model1 = new SpinnerNumberModel(initValue, min, max, step);
    SpinnerModel model2 = new SpinnerNumberModel(initValue, min, max, step);
```

```
SpinnerModel model3 = new SpinnerNumberModel(initValue, min, max, step);
SpinnerModel model4 = new SpinnerNumberModel(initValue, min, max, step);
SpinnerModel model5 = new SpinnerNumberModel(initValue, min, max, step);

private JSpinner jSpinner1 = new JSpinner(model1);
private JSpinner jSpinner2 = new JSpinner(model2);
private JSpinner jSpinner4 = new JSpinner(model4);
private JSpinner jSpinner3 = new JSpinner(model3);
private JSpinner jSpinner5 = new JSpinner(model5);

private int i;
private JButton jButton1 = new JButton();
public LayerThermal layerthermal;
private boolean ambient_medium = false;


public LayerThermalInterface(int iw, boolean ambient) {

   this.ambient_medium = ambient;
   jLabel2.setText(iw + 1 + ". Layer");

   this.i = iw;
   try {
      jbInit();
   } catch (Exception e) {
      e.printStackTrace();
   }
}

public LayerThermalInterface() {
   try {
      jbInit();
   } catch (Exception e) {
      e.printStackTrace();
   }
}

private void jbInit() throws Exception {

   this.getContentPane().setLayout(null);
   this.setTitle("Tissue Thermal Parameters ");
   this.setSize(new Dimension(333, 366));
   jSeparator1.setBounds(new Rectangle(10, 295, 290, 5));
   jSeparator2.setBounds(new Rectangle(10, 40, 290, 5));
   jSeparator5.setBounds(new Rectangle(300, 40, 5, 255));
   jSeparator5.setOrientation(SwingConstants.VERTICAL);
   jSeparator8.setBounds(new Rectangle(10, 40, 5, 255));
   jSeparator8.setOrientation(SwingConstants.VERTICAL);
   jLabel2.setBounds(new Rectangle(20, 40, 110, 20));
```

```
jLabel2.setToolTipText("null");
jLabel4.setText("Heat Capacity :");
jLabel4.setBounds(new Rectangle(20, 70, 110, 20));
jLabel4.setToolTipText("Heat Capacity :");
jLabel5.setText("Thermal Conductivity :");
jLabel5.setBounds(new Rectangle(20, 105, 140, 20));
jLabel5.setToolTipText("Thermal Conductivity :");
jLabel6.setText("Density :");
jLabel6.setBounds(new Rectangle(20, 140, 130, 20));
jLabel6.setToolTipText("Density :");
jLabel7.setText("Blood Perfusion Rate :");
jLabel7.setBounds(new Rectangle(20, 175, 110, 20));
jLabel7.setToolTipText("Blood Perfusion Rate :");
jSpinner2.setBounds(new Rectangle(160, 105, 110, 20));
jSpinner2.setBorder(BorderFactory.createLineBorder(Color.black, 1));
jSpinner4.setBounds(new Rectangle(160, 175, 110, 20));
jSpinner4.setBorder(BorderFactory.createLineBorder(Color.black, 1));
jSpinner3.setBounds(new Rectangle(160, 140, 110, 20));
jSpinner3.setBorder(BorderFactory.createLineBorder(Color.black, 1));
jSpinner1.setBounds(new Rectangle(160, 70, 110, 20));
jSpinner1.setBorder(BorderFactory.createLineBorder(Color.black, 1));
jSpinner1.setToolTipText("null");
jSpinner1.setFocusable(false);
jSpinner1.setVerifyInputWhenFocusTarget(false);
jButton1.setText("Save");
jButton1.setBounds(new Rectangle(185, 260, 95, 20));
jButton1.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
            jButton1_actionPerformed(e);
          }
        });
jLabel1.setBounds(new Rectangle(15, 275, 295, 20));
jLabel3.setText("Thickness : ");
jLabel3.setBounds(new Rectangle(20, 210, 105, 25));
jSpinner5.setBounds(new Rectangle(160, 215, 110, 20));
jSpinner5.setBorder(BorderFactory.createLineBorder(Color.black, 1));
this.getContentPane().add(jSpinner5, null);
this.getContentPane().add(jLabel3, null);
this.getContentPane().add(jLabel1, null);
this.getContentPane().add(jButton1, null);
this.getContentPane().add(jSpinner1, null);
this.getContentPane().add(jSpinner3, null);
this.getContentPane().add(jSpinner4, null);
this.getContentPane().add(jSpinner2, null);
this.getContentPane().add(jLabel7, null);
this.getContentPane().add(jLabel6, null);
this.getContentPane().add(jLabel5, null);
this.getContentPane().add(jLabel4, null);
this.getContentPane().add(jLabel2, null);
```

```
this.getContentPane().add(jSeparator8, null);
this.getContentPane().add(jSeparator5, null);
this.getContentPane().add(jSeparator2, null);
this.getContentPane().add(jSeparator1, null);
this.setVisible(true);

this.setResizable(false);
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = this.getSize();

this.setLocation((screenSize.width - frameSize.width) / 2 + i * 30,
            (screenSize.height - frameSize.height) / 2 + i * 30);

this.addWindowListener(new WindowAdapter() {
          public void windowClosing(WindowEvent e) {

           }
       });

if (i == 1) {
   this.jSpinner1.setValue(3.6);
   this.jSpinner2.setValue(0.0046);
   this.jSpinner3.setValue(1.05);
   this.jSpinner4.setValue(0.0);
   this.jSpinner5.setValue(0.4);
} else if (i == 2) {
   this.jSpinner1.setValue(3.7);
   this.jSpinner2.setValue(0.0049);
   this.jSpinner3.setValue(1.03);
   this.jSpinner4.setValue(0.0);
   this.jSpinner5.setValue(0.1);
} else if (i == 3) {
   this.jSpinner1.setValue(3.06);
   this.jSpinner2.setValue(0.0021);
   this.jSpinner3.setValue(1.0);
   this.jSpinner4.setValue(0.0);
   this.jSpinner5.setValue(0.2);
}


if (this.ambient_medium == true) {

   this.jSpinner1.setValue(1);
   this.jSpinner2.setValue(0);
   this.jSpinner3.setValue(0);
   this.jSpinner4.setValue(0);
   this.jSpinner5.setValue(0);
   this.jSpinner1.setEnabled(false);
   this.jSpinner2.setEnabled(false);
```

```
            this.jSpinner3.setEnabled(false);
            this.jSpinner4.setEnabled(false);
            this.jSpinner5.setEnabled(false);
            this.jButton1.setEnabled(false);

            this.saveButton();
        }

    }


    private void jButton1_actionPerformed(ActionEvent e) {

        this.saveButton();
    }

    private void saveButton() {

        this.layerthermal = new LayerThermal();
        this.layerthermal.setc((Double.parseDouble(jSpinner1.getValue().toString())));
        this.layerthermal.setk((Double.parseDouble(jSpinner2.getValue().toString())));
        this.layerthermal.setdens((Double.parseDouble(jSpinner3.getValue().toString())));
        this.layerthermal.setwb((Double.parseDouble(jSpinner4.getValue().toString())));

        if (this.ambient_medium == false) {
            this.setVisible(false);
        }
    }

}
```



REFERENCES

1. Waynant, R.W., “Lasers in Medicine”, CRC Press, 2002.

2. Niemz, M.H., “Laser-Tissue Interactions: Fundamentals and Applications”, Springer, 1996.

3. Branco, G., “The Development and Evaluation of Head Probes for Optical Imagıng of the Infant Head”, Department of Medical Physics of UCL, January 2007.

4. Florian, E.W.S., “Development of a Time-Resolved Optical Tomography System for Neonatal Brain Imaging”, Department of Medical Physics of UCL, November 1999.

5. Ishimaru, A., “Wave Propagation and Scattering in Random Media Rough Surfaces”, IEEE, Vol 79, No. 10. October 1991.

6. Tuchin, V., “Tissue Optics: Light Scattering Methods and Instruments for Medical Diagnosis”, SPIE Press, 2007.

7. Plass, G.N., G.W. Kattawar, and F.E. Catchings, “Matrix Operator Theory of Radiative Transfer. 1: Rayleigh Scattering”, Appl. Opt. 12, 314–329, 1973.

8. Peng, Q., A. Juzeniene, J. Chen, L.O. Svaasand, T. Warloe, K. Giercksky, and J. Moan, “Lasers in Medicine”, Rep. Prog. Phys., 2008.

9. Henyey, L., and J. Greenstein, “Diffuse Radiation in The Galaxy”. Astrophys. J. 93, 70–83, 1941.

10. Welch, A.J., and M.J.C. Vangemert, “Optical-Thermal Reponse of Laser-Irradiated Tissue”, Springer, 1995.

11. Lisa Carroll, M.D., R. Tatyana, and M.D. Humphreys, “Laser-Tissue Interactions”, Clinics in Dermatology, Vol. 24, p. 2–7, 2006.

12. Sturesson, C., and S.A. Engels, “A Mathematical Model for Predicting the Temperature Distribution in Laser-induced Hyperthermia: Experimental Evaluation and Applications”, Phys. Med. Biol., Vol. 40, p. 2037-2052, 1995.

13. Crochet, J.J., and S.C. Gnyawali, “Temperature Distribution in Selective Laser-Tissue Interaction”, Journal of Biomedical Optics, Vol. 11, No. 3, May/June 2006.

14. Farina, B., S. Saponaro, E. Pignoli, S. Tomatis, and R. Marchesini, “Monte Carlo Simulation of Light Fluence in Tissue in a Cylindrical Diffusing Fibre Geometry”, Phys. Med. Biol., Vol. 44, p. 1-11,1999.

15. Manns, F., D. Borja, and J.M. Parel, “Semianalytical Thermal Model for Subablative Laser Heating of Homogeneous Nonperfused Biological Tissue: Application to Laser Thermokeratoplasty”, Journal of Biomedical Optics, Vol. 8 No. 2, p. 288–297, April 2003.

16. Glenn, T.N., S. Rastegar, and S.L. Jacques, “Finite Element Analysis of Temperature Controlled Coagulation in Laser Irradiated Tissue”, IEEE Transactions on Biomedical Engineering, Vol. 43, No. 1, January 1996.

17. Mohammed, Y., and J.F. Verhey, “A FiniteEelement Method Model to Simulate Laser Interstitial Thermotherapy in Anatomical Inhomogeneous Regions”, BioMedical Engineering OnLine, 2005.

18. Glenn, T.N., S. Rastegar, and S.L. Jacques, “Finite Element Analysis of Temperature Controlled Coagulation in Laser Irradiated Tissue”, IEEE Transactions on Biomedical Engineering, Vol. 43, No. 1, January 1996.

19. Özışık, M.N., “Heat Conduction”, New York : Wiley, 1980.

20. Özışık, M.N., “Basic Heat Transfer”, New York : McGraw-Hill, 1977.

21. Page, A.J., S. Coyle, T.M. Keane, T.J. Naughton, C. Markham, and T. Ward, “Distributed Monte Carlo Simulation of Light Transportation in Tissue”,

22. Flock, S.T., M.S. Patterson, B.C. Wilson, and D.R. Wyman, “Monte Carlo Modeling of Light Propagation in Highly Scattering Tissues-I: Model Predictions and Comparison with Diffusion Theory”, IEEE Transactions on Biomedical Engineering, Vol.36, No. 12, 1989.

23. Patwardhan, S.V., A.P. Dhawan, “Monte Carlo Simulation of Light-Tissue Interaction: Three-Dimensional Simulation for Trans-Illumination-Based Imaging of Skin Lesions”, IEEE Transactions on Biomedical Engineering, Vol.52, No. 7, 2005.

24. Kumar, D., and M. Singh, “Characterization and Imaging of Compositional Variation in Tissues”, IEEE Transactions on Biomedical Engineering, Vol.50, No. 8, August 2003.

25. Chicea, D., and I. Turcu, “A Random Walk Monte Carlo Approach To Simulate Multiple Light Scattering On Biological Suspensions”, Romanian Reports in Physics, Vol. 57, No. 3, p. 418-425, 2005.

26. Prahl, S. A., M. Keijzer, S. L. Jacques, and A. J. Welch, “A Monte Carlo Model of Light Propagation in Tissue”, SPIE Institute Series, Vol. 5, 1989.

27. Prahl, S.A., “Light Transport in Tissue”, The University of Texas at Austin, December, 1988.

28. Wang, L., S.L. Jacques, and L. Zheng, “Monte Carlo Modeling of Light Transport in Multi-Layered tissues”, Computer Methods and Programs in Biomedicine, Vol. 47, p. 131-146, 1995.

29. Wang, L., and S.L. Jacques, “Monte Carlo Modeling of Light Transport in Multi-layered Tissues in Standard C”, University of Texas, 1992.

30. Jacques, S.L., and S.A. Prahl, “Modeling Optical and Thermal Distributions in Tissue During Laser Irradiation”, Lasers Surg. Med. 6, 494–503, 1987.

31. Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Veterling, “Numerical Recipes inC”, Cambridge University Press, 1992.

32. Wang, L., S.L. Jacques, and L. Zheng, “Convolution for Responses to a Finite Diameter Photon Beam Incident on Multi-layered Tissues”, Computer Methods and Programs in Biomedicine, Vol. 54, p.141-150, 1997.

33. Van Gemert, M.J.C., G.W. Lucassen, and A.J. Welch, “Time Constants in Thermal Laser Medicine: II. Distributions of Time Constants and Thermal Relaxation of Tissue”, Phys. Med. Biol. 41, p. 1381–1399, 1999.

34. Hodsont, D.A., J.C. Barbenel, and G. Eason, “Modelling Transient Heat Transfer Through The Skin and a Contact Material”, Phys. Med. Biol., Vol. 34, No 10, p.1493-1507, 1989.

35. Çengel, Y.A., “Heat Transfer: A Practical Approach”, McGraw Hill Professional, 2003.

36. Cheong, W. F., S. A. Prahl, and A. J. Welch, “A Review of the Optical Properties of Biological Tissues”, IEEE J. Quantum Electronics, 26, p.2166-2185, 1990.

37. Cooper, T. E. and G. J. Trezek, “Correlation of Thermal Properties of Some Human Tissue with Water Content”, Aerospace Med., Vol.42, p.24–27, 1971.

38. Drakaki, E., M. Makropoulou, and A. A. Serafetinides, “In Vitro Fluorescence Measurements and Monte Carlo Simulation of Laser Irradiation Propagation in Porcine Skin Tissue”, Lasers Med. Sci., 2007.