ISSUES RELATED TO THE FORWARD PROBLEM

FOR ENDOSCOPIC NEAR-INFRARED DIFFUSE

OPTICAL TOMOGRAPHY

By

CAMERON HOLLIS MUSGROVE

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

2005

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2007

ISSUES RELATED TO THE FORWARD PROBLEM

FOR ENDOSCOPIC NEAR-INFRARED DIFFUSE

OPTICAL TOMOGRAPHY

Thesis Approved:

Dr. Charles F. Bunting

Thesis Advisor

Dr. Daqing Piao

Dr. James West

Dr. A. Gordon Emslie

Dean of the Graduate College

Acknowledgements

I thank my wife, Maryellen, it was her support, understanding, patience, and most importantly, love that made this work possible.

Thanks to my parents, for believing in me, my education, and always supporting me. Also, thanks to my grandparents who constantly remind me of the value of a good education, and encouragement to make the most of my education.

I am eternally grateful for my thesis advisor, Dr. Charles Bunting whose always provided positive encouragement, guidance, open discussions, and supporting me along my own path, wherever it lead.

I want to thank my committee. Dr. Daqing Piao, whose support and guidance made this research possible, and Dr. James West for always challenging me to learn more.

I also want to thank my friends for making life, and this thesis, much easier: Vignesh Rajamani, Jakkrit Kunthong, and Guan Xu. Especially Vignesh for his encouragement and advice.

I also want to acknowledge the financial support of Dr. Piao’s OSU-CEAT startup fund and Army Medical Research and Material Command, Grant #W81XWH-07-1-0247.

Table of Contents

List of Figures

Acronyms

NIR	Near-infrared
NIRFAST	Near-infrared Frequency domain Absorption and Scattering Tomography
DOT	Diffuse Optical Tomography
DA	Diffusion Approximation
RTE	Radiative Transfer Equation
PDE	Partial Differential Equation
FEM	Finite Element Method
FWHM	Full Width at Half Maximum

1 Introduction

Within the last decade there has been much development towards improving the diagnosis of breast cancer with diffuse optical tomography. With diffuse optical tomography researchers are able to determine the location of suspected tumors, monitor the physiological state of the tissue, and selectively image fluorescent compounds *in vivo* [1]. Diffuse optical tomography methods for breast cancer detection are being adapted to other kinds of cancer, such as prostate cancer. Over 200,000 men are diagnosed with prostate cancer each year [2]. When prostate cancer is suspected, the presence of cancer must be confirmed with a tissue biopsy [2]. The doctor uses a trans-rectal ultrasound probe to guide the biopsy needle through the prostate to take several samples at many different locations [2]. The reason for so many samples is that the contrast between the cancerous and healthy tissue is not clear on the ultrasound image, and it is difficult to 'see' the tumor on the ultrasound image; therefore the doctor samples several areas to increase the chance the biopsy needle will capture a piece of cancerous tissue. The accuracy of the biopsy needle could increase and the number of samples required could decrease if it would be possible for the doctor to 'see' the suspected tumor. This may be possible with a video-rate real-time imaging system that combines the near-infrared contrast with the spatial accuracy of ultrasound.

1.1 External Diffuse Optical Tomography Methods

Most applications of diffuse optical tomography have sensors outside of the body to image external organs or within extremities of the body. External diffuse optical tomography does not work well for imaging deep within internal organs because after a few centimeters most of the light is absorbed or scattered beyond quantum limit of detection for opto-electronics, and there are limits to the amount of power in the imaging beam to prevent harm to the patient.

Near-infrared light has a unique property within the body to distinguish between areas of high blood vessel concentration and areas with lower blood vessel concentration. In fact, this property of near-infrared light is commonly used in hospitals to monitor blood oxygenation in patients. Cancerous tissue typically has increased blood flow since it is using more nutrients than healthy tissue to grow faster [3]. Therefore, by using the vasculature contrast of near-infrared light it is possible for NIR diffuse optical tomography (DOT) systems to distinguish between healthy and cancerous tissues. The cancerous tissue absorbs more light than healthy tissue, and the NIR DOT reconstructed image shows areas of contrasting light absorption.

For a typical NIR DOT imaging system, the image reconstruction process can be separated into two different problems: forward and inverse [4]. The forward problem estimates the boundary measurements given information about the sources and material properties [4]. Usually, the diffusion approximation of the radiative transfer equation is the preferred algorithm for estimating these boundary measurement values for its speed and accuracy when the object of interest is optically far from the sources and boundaries. The inverse problem estimates the optical properties (produces a tomographic image) of tissue under study given boundary source information and boundary measurements [4]. Usually an iterative solver will ‘guess’ a solution, execute a forward model calculation to check the guess, and make another ‘guess’ until a solution is verified, however there are many different methods [4]. Without an accurate forward model, an accurate inverse problem is hopeless. Specific to breast imaging, a software package called, “NIRFAST” has been developed to solve the forward and inverse problems for optical properties, spectroscopic, and fluorescence imaging. NIRFAST has been validated against laboratory and clinical measurements to be an accurate predictor of system performance and reconstructed images for breast NIR DOT.

1.2 Internal Diffuse Optical Tomography

Images of the prostate can obtained by a trans-rectal ultrasound. Since the prostate is within $4\,\mathrm{cm}$ of the colon wall, a trans-rectal probe could potentially propagate the NIR light a relatively small distance, making the trans-rectal probe a logical choice.

Applying DOT inside the body becomes a different problem from its typical applications imaging outside the body. Reconstruction of optical material properties within the prostate requires a different methodology from typical methods. Typically, a ring of sensors surround the tissue from outside the body, increasing the chance that light from a source will reach all detectors. To image the prostate, the array of sensors will have to sense the surrounding tissue, with only a small number of detectors receiving the light; assuming light bounces back toward the probe. Also, for this trans-rectal or endoscopic probe, there are now two boundaries to consider for modeling: the probe surface and the outer simulation boundary.

1.2.1 First Probe Design

As a first attempt at trans-rectal endoscopic near-infrared diffuse optical tomography the typical source detector arrangement was inverted so the sources and detectors are looking beyond the ring, but the ring had to become much smaller ($20\,\text{mm}$) to coincide with the size of typical trans-rectal ultrasound probes (see Figure 1.1).

NIRFAST has been adjusted for this new endoscopic geometry to reconstruct images from simulated and experimental data. It was discovered that the simulations predicted by NIRFAST did not accurately match with measured results, and the simulations could not reconstruct the material parameters in a satisfactory quantitative or qualitative way [5].

Since NIRFAST can accurately reconstruct images for the external case, it was clear that the direct application of NIRFAST for the endoscopic problem is not sufficient. The imaging region of interest is close to the boundary, and the source-detector distances are much smaller; these small distances may violate the minimum distance requirement for the diffusion approximation. The biggest issue with the NIRFAST endoscopic simulations from the previous work [5] is the application of an air/tissue boundary on the exterior boundary. This boundary condition is not sufficient, since it is supposed to be an absorbing boundary. The external boundary does not represent the boundary of the body, it is only to limit the simulation to the region of interest and remain ‘invisible’ to the fluence calculations.

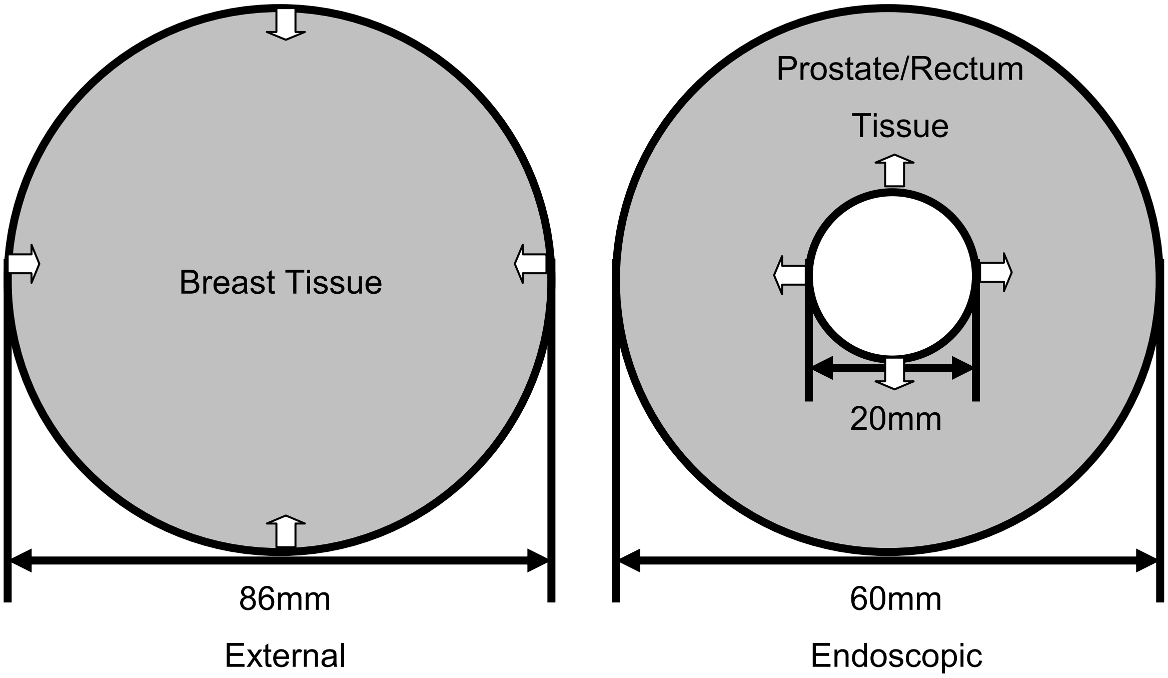


Figure 1.1: Comparison of External/Ring Geometry to Endoscopic Geometry

1.3 Thesis Organization

Chapter 2 describes how near-infrared light diffuse optical tomography can identify potentially cancerous tissue. Section 2.2 summarizes three methods for modeling light propagation within tissue, including a detailed description of the diffusion approximation and the Robin boundary condition. Some of the improvements made to the diffusion approximation to increase its accuracy are presented in section 2.2.4. Section 2.3 discusses the basic application of two software packages (COMSOL Multiphysics and NIRFAST) for the endoscopic diffusion approximation model. Chapter 3 specifies how to implement the diffusion approximation in COMSOL Multiphysics, and execute the necessary NIRFAST code. Six metrics are used to compare simulations; the calculation and purpose of each metric is explained. To ensure simulations are comparable, the source expression is evaluated for COMSOL and NIRFAST simulations to ensure they are a match. In chapter 4, the COMSOL simulation is evaluated against NIRFAST for the breast cancer geometry with and without a suspected tumor. The fluence difference in response to changing the external boundary condition from air/tissue to tissue/tissue is revealed. A sensitivity study evaluates the feasibility of the diffusion approximation to detect a realistic and ideal absorber. Chapter 5 describes the construction of a finite element diffusion approximation (FEDA) MATLAB script from the variational expression of the frequency-domain diffusion approximation. FEDA's performance is evaluated against COMSOL and NIRFAST for the external geometry, and COMSOL for the endoscopic geometry. A sensitivity study is presented to evaluate the capability of FEDA to detect a real and ideal absorber.

2 Background

This chapter describes how near-infrared light diffuse optical tomography can identify potentially cancerous tissue. Section 2.2 summarizes three methods for modeling light propagation within tissue, including a detailed description of the diffusion approximation and the Robin boundary condition. Some of the improvements made to the diffusion approximation to increase its accuracy are presented in section 2.2.4. Section 2.3 discusses the basic application of two software packages (COMSOL Multiphysics and NIRFAST) for the endoscopic diffusion approximation model.

2.1 Physics of Diffuse Optical Tomography

Light in the near-infrared portion (650nm – 900nm) of the electromagnetic spectrum has the unique property of being somewhat transparent to human tissue. However, due the fact that the near-infrared (NIR) wavelength is comparable to the dimension of cellular organelles, the light is highly scattered as it passes through tissue. This effect can be readily observed by the glow from shining a laser beam on one's skin. Or, by blocking a flashlight beam with one's hand in a dark room, as shown by Figure 2.1.

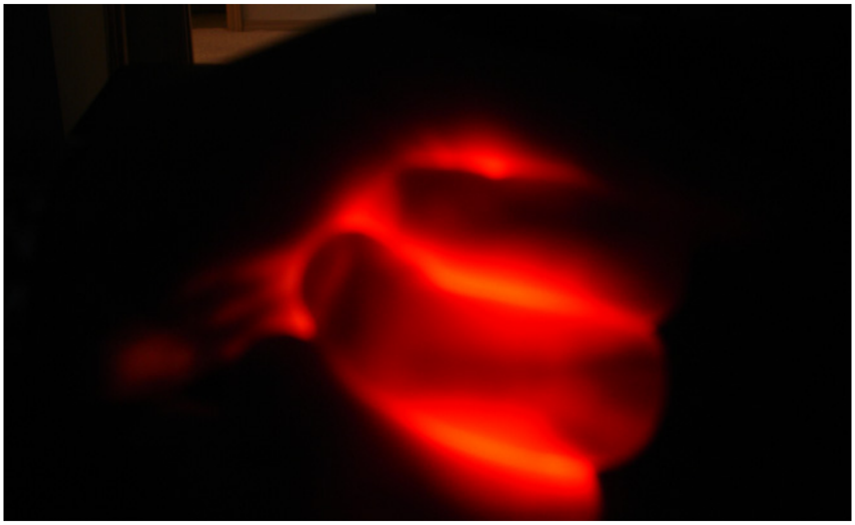


Figure 2.1: A Hand Covering a Flashlight Beam in a Dark Room

Some of the NIR is absorbed by certain compounds (or chromophores) within the tissue. The most relevant chromophore for detecting cancer is the oxy hemoglobin and deoxy hemoglobin within blood. One measure of a chromophore's absorption strength at a particular wavelength is the absorption coefficient. Figure 2.2 shows the absorption coefficient for oxy hemoglobin and deoxy hemoglobin.

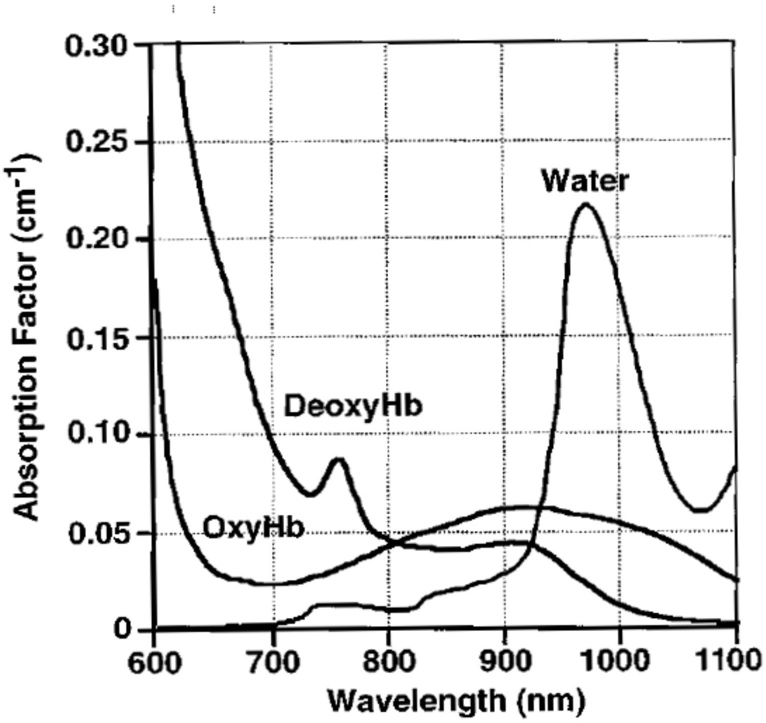


Figure 2.2: Absorption Coefficient of Oxy- and Deoxy- Hemoglobin [6]

Figure 2.2 indicates that near 800 nm the absorption coefficient values intersect; indicating that at wavelengths less than 800 nm oxy-hemoglobin absorption dominants, and vise-versa for wavelengths greater than 800 nm. Using this property, blood oxygenation detectors have been used in hospitals for several years. This property has also found a use to detect regions of tissue that have a higher concentration of oxy-hemoglobin than the surrounding tissue. This concentration difference is the contrast within diffuse optical tomography.

Cancer usually starts as a single cell. This one cell develops into a tumor when it divides to make copies of itself. Typically, cancer cells divide to grow into a tumor faster than healthy cells. To feed this growth, cancer cells need more nutrients than the surrounding tissue, and will encourage new blood vessels to grow within the tumor to supply the increased nutrients. This increase in blood vessels means more blood is present in this tumor area than before the tumor started. It is this increase in blood volume relative to surrounding tissue that diffuse optical tomography can use as a contrast because the increased absorption of NIR through the tumor area. Ultimately, the presence of a tumor must be confirmed by tissue biopsy. Diffuse optical tomography may be used as a tool to identify lesions to confine biopsy localization.

2.2 Light Propagation Models

There are many different computational methods to describe electromagnetic energy propagation. Since near-infrared (NIR) light is electromagnetic energy, Maxwell’s Equations accurately describe its propagation path. However, the computational methods applied for electromagnetic energy at NIR frequencies become computationally prohibitive, especially to describe multiple scattering events. Biological tissue has a bigger effect on light propagation than microwave energy passing through the atmosphere. This section will discuss methods other than Maxwell’s Equations to describe NIR light propagation through biological media.

2.2.1 Monte Carlo Methods

Monte Carlo modeling is a statistical process to model the interaction of individual photons as they move through a medium [7]. Typically, Monte Carlo methods use macroscopic properties of tissue, rather than detailed structure of individual cells, to model the photon path [7]. However, Monte Carlo methods have been well established as an accurate representation of light propagation through any defined optical property or tissue structure [8,9]. Monte Carlo methods are not practical for a real-time imaging system because the time to calculate a sufficient number of photons is much greater than other computational methods, like the radiative transfer equation [8,10].

2.2.2 Radiative Transfer Equation (RTE)

Another analytical model to describe light propagation is the radiative transfer equation (RTE). The RTE describes light propagation through multiple scattering media as an energy wave, ignoring the wave-like nature of light [11]. The RTE [12], equation 2.1, has applications from predicting interstellar light propagation to modeling neutron path within nuclear reactors to light propagation within biological tissues [12,13]. There are three forms of the transport equation: surface-integral, integral, and integro-differential [13]. Surface integral methods are useful to know the incoming and exiting angular fluxes [13], but for reconstructing images of tissue optical properties, the light propagation path within the tissue is important. The integral method is applicable for optically thin media [13]. For trans-rectal imaging of the prostate a minimum imaging depth of 20mm is desired, therefore the region of interest is optically thick and integral methods are not sufficient. The integro-differential form is widely used for optically thick media [13], and in modeling of infrared light through biological tissue [4]. The integro-differential equation can be solved by several methods: singular eigenfunction, discrete ordinates, and spherical harmonic expansion [13]. The integro–differential form is expressed as

$$\hat{s}\cdot\nabla L(\boldsymbol{r},\hat{s})+\mu_t(\boldsymbol{r})L(\boldsymbol{r},\hat{s})=\mu_s\int_{4\pi}p(\hat{s},\hat{s}')L(\boldsymbol{r},\hat{s}')d\omega'+S(\boldsymbol{r},\hat{s}) \qquad (2.1)$$

Where $L(\boldsymbol{r},\hat{s})$ is the radiance at position $\boldsymbol{r}$ in direction $\hat{s}$. μ_s is the scattering coefficient of the medium. μ_t is equal to $\mu_s+\mu_a$, where μ_a is the absorption coefficient of the medium. $p(\hat{s},\hat{s}')$ is the scattering phase function. $S(\boldsymbol{r},\hat{s})$ is the source term describing the unscattered energy.

2.2.2.1 Solution Methods

There are many methods and approximations to solve the radiative transfer equation. Each solution method has a particular application, depending on material parameters and simulation space. The singular eigenfunction expansion method is so difficult that it has only been used for numerical calculations in one-dimensional geometries [13]. Two popular methods, discrete ordinates and spherical harmonic expansion, approximate the integral term in the radiative transfer, equation 2.1, to make the equation ‘easier’ to solve.

The discrete ordinates method approximates the integral with a numerical quadrature, then solves the resulting differential equations with a finite difference technique [13]. Klose [14] presents a successful implementation of an upwind-difference discrete ordinates method for optical tomography. The spatial domain is separated into a rectangular mesh on which the radiance is solved using finite-difference equations that change depending on direction [14]. This upwind-difference discrete-ordinates method works for media containing both scattering and nonscattering regions [14]. As with typical finite-difference methods, the mesh discretization becomes problematic for circular and curved boundaries that do not coincide with gridpoints in that additional boundary conditions must be imposed to estimate boundary values.

A finite element solution of the RTE is presented by Tarvainen [9]. The finite element method has an advantage over finite-difference techniques in that the finite element mesh is more robust in definition of arbitrarily curved boundaries and adjusting node density. Tarvainen [9] shows the variational form of the RTE in matrix form. However, the challenge in solving the RTE is the computation requirements become enormous as the solution domain increases [9]. Therefore Tarvainen's work [9] combines the RTE with a faster method to compute the majority of the domain to make the entire problem feasible for diffuse optical tomography. This faster method is called the 'diffusion approximation'.

2.2.3 Diffusion Approximation (DA)

The most popular method to describe infrared light propagation in biological tissue for diffuse optical tomography is the P_1 or 'diffusion approximation'. The term P_1 arises from the fact that the spherical harmonic expansion of the radiance is multiplied by Legendre polynomials P_N, where N=0,1,2,3…. Using the first two terms of the spherical harmonic expansion sets N=1, therefore it is called the P_1 approximation. The term 'diffusion approximation' arises from the assumptions made that the media is much more scattering than absorbing and light propagation is far enough from sources and boundaries that it is assumed that it isotropically scatters [4,12]. The DA can be derived from not only the RTE, but from macroscopic principles of energy conservation and Fick's Law [12].

2.2.3.1 Diffusion Approximation of the RTE

For complete details on the derivation of the DA refer to [4] and [12]. The frequency domain form of the DA is [4]

$$-\nabla\cdot\kappa(r)\nabla\Phi(r,\omega)+\mu_a\Phi(r,\omega)+\frac{j\omega}{v}\Phi(r,\omega)=q_0(r,\omega) \qquad (2.2)$$

Where:

- the diffusion coefficient, $\kappa(r)=\dfrac{1}{3(\mu_a+\mu_s')}$ [4]
- μ_a is the absorption coefficient [4].
- $\Phi(r,\omega)$ is the fluence rate at position r and frequency ω
- $q_0(r,\omega)$ is the source fluence rate at position r and frequency ω
- v is the speed of light
- the reduced scattering coefficient, $\mu_s'=(1-g)\mu_s$
- ω is the modulation frequency of the incident light beam

The absorption coefficient, μ_a, is defined in per unit length (usually mm^{-1}) that defines the probability a photon is absorbed over a distance [15]. The scattering coefficient, μ_s, is defined in per unit length (usually mm^{-1}) that describes the probability a photon is scattered (changes direction) over a distance [15]. The anisotropy parameter, g, represents the cosine of the mean scattering angle from the photon's direction before scattering to the photon's direction after scattering [12]. The reduced scattering coefficient, μ_s', is used for highly scattering media (where the diffusion approximation applies) to account for the averaging of g and μ_s over many scattering events [16].

2.2.3.2 Boundary Conditions for Diffusion Approximation

As with typical electromagnetic computational methods, there are two different boundary conditions utilized: Dirichlet and Neumann. The Dirichlet boundary condition imposes a constant value of the fluence rate at the boundary. When this fluence rate value is set to zero, it is the physical equivalent of a perfect absorbing material surrounding the region of interest [17]. The Neumann boundary condition is a mixed condition of the photon density and photon current on the boundary [17]. The Neumann condition is also known as a Robin boundary condition (RBC). The Robin boundary condition can be further modified to account for refractive index mismatch at the boundary; this is called the modified Robin boundary condition. For NIR DOT the modified Robin boundary condition is expressed as [17]

$$\Phi\left(r\right)+2\kappa \mathsf{A}\hat{n}\cdot\nabla\Phi\left(r\right)=0 \tag{2.3}$$

Where:

- $\Phi(r)$ is the energy fluence rate on the boundary
- κ is the diffusion coefficient (same as equation 2.2)
- A is a refractive index mismatch coefficient

Furthermore, the refractive index mismatch coefficient is defined as [4,17]

$$\mathsf{A}=\frac{\dfrac{2}{\left(1-R_0\right)}-1+\left|\cos\theta_c\right|^3}{1-\left|\cos\theta_c\right|^2} \tag{2.4}$$

$$R_0=\frac{\left(\dfrac{n_{inside}}{n_{outside}}-1\right)^2}{\left(\dfrac{n_{inside}}{n_{outside}}+1\right)^2} \tag{2.5}$$

$$\theta_c=\arcsin\left(n_{outside}/n_{inside}\right) \tag{2.6}$$

Where $n_{outside}$ is the refractive index outside the boundary, and n_{inside} is the refractive index inside the boundary and of the simulation domain. Although mathematically simpler, the Dirichlet boundary condition is not as accurate as the modified Robin boundary condition [17].

Another type of boundary condition is called the extrapolated boundary condition where a boundary is created outside of the real boundary, and the Dirichlet boundary condition is imposed on this new boundary [17]. It has been shown that the extrapolated boundary condition has comparable accuracy to the modified Robin boundary condition [17].

2.2.4 Diffusion Approximation Improvements

Generally, the diffusion approximation is only valid for highly scattering regions and far from sources and boundaries [12]. The accuracy of the diffusion approximation depends greatly upon many variables such as geometry, coordinate system, material properties, and simulation domain. There have been attempts by many to improve upon the accuracy of the diffusion approximation. A summary of the major improvements are discussed below.

2.2.4.1 Higher order terms

The diffusion approximation approximates the radiance term of the RTE by the first two terms of the spherical harmonic expansion. Including more terms of the spherical expansion series, typically four terms, can increase the accuracy of the approximation near sources and boundaries. This increased accuracy for slab geometry has been proven by [18]. One reason for increased accuracy is the additional terms of the expansion increase the number of terms that specify the boundary condition [12]. However this increase in accuracy may not apply to all cases. It is suggested that for spherical geometry, since the boundary conditions are specified differently, additional terms from the spherical expansion will not increase the accuracy [12].

2.2.4.2 Hybrid Models

Monte Carlo and the radiative transfer equation (RTE) are applicable to wider range of materials than the diffusion approximation. However, these methods take much more time to execute over a larger domain than the diffusion approximation calculation. Therefore there have been attempts to limit the domain Monte Carlo and RTE are applied to achieve a quick, but accurate solution. One group combines the RTE with the diffusion approximation by solving the RTE near boundaries and sources, and uses the RTE solution as a Dirichlet boundary condition for the diffusion approximation solution over the rest of the domain [9]. Another group has developed a Monte Carlo and diffusion approximation model to describe the regions where the diffusion theory is not valid but retain the quick diffusion calculation where applicable [8]. In fact, Monte Carlo analysis is the slowest where the diffusion theory calculation is applicable (regions that are scatter dominated) [8]. Both groups determined that the hybrid models are more accurate than the diffusion approximation over the entire domain [8,9].

2.2.4.3 Non-scattering Regions

The diffusion approximation must be applied for media that scatter light much more than absorb it. For certain regions of the body, such as the brain, there are areas for which light is not scattered at all [19]. The diffusion approximation has extreme difficulty modeling this area. However, [19] has shown modeling non-scattering regions with the diffusion approximation is possible. By creating additional source terms for each non-scattering region within the simulation domain, the diffusion approximation calculation is able to predict the light propagation reasonably well [19]. The drawback to this technique is the exact location of the void regions must be known to correct the diffusion approximation equation model [19].

2.2.5 Validation Techniques

To validate the diffusion approximation and its corrections the simulation results must be compared to results that are known to be true. The preferred method of validation is a Monte Carlo analysis. The Monte Carlo analysis is considered the most accurate simulation of photon path through scattering or non-scattering media [8]. Numerous people [20] have validated the diffusion approximation against Monte Carlo simulations and found the diffusion approximation to be a good estimate, with certain limitations.

2.2.6 Geometry Considerations

The implications of the geometry on the forward problem model are significant. Typical near-infrared imaging systems utilize an external arrangement of sources and detectors around extremities of the body [4,21]. The endoscopic geometry presents some important differences from the external/ring geometry. In the external/ring geometry simulation, the domain is bounded by a non-scattering medium (air) which reflects and absorbs light. To keep the endoscopic geometry within computational limits, the boundary must be set within the tissue, rather than simulating the entire cross-section of the body. However, the boundary must not be so close to the probe such that the boundary effects the results. Furthermore, the dynamic range of the external geometry could potentially be much lower than the endoscopic case; the light emitted from the external probe may not be reflected back the detectors.

2.3 Software

There is no professional software program available to the author's knowledge that uses the diffusion approximation or radiative transfer equation to model light propagation through user configurable geometry and material properties for the purposes of optical tomography. Each research group seems to have their own code customized for their measurement equipment, problem geometry, and material properties. NIRFAST is one of those codes developed by a research group at Dartmouth College.

2.3.1 NIRFAST

NIRFAST (Near-infrared Frequency-domain Absorption and Scattering Tomography) is one optical tomography algorithm that has been well validated for the diffusion approximation applied for ring geometry to reconstruct structural information, determine physiological condition of tissue with spectroscopy, and locate fluorescent chromophores [1].

The NIRFAST program solves the forward problem of diffuse optical tomography (DOT) by using the frequency domain diffusion approximation to the radiative transfer equation (equation 2.2). To begin any simulation, a finite element mesh must be specified to NIRFAST. This mesh can either be 2D or 3D, but must be provided by the user since NIRFAST does not have the capability to create a mesh. The diffusion equation is separated into a system of linear equations that are solved on a finite element mesh for the fluence rate at each node [22]. On the external boundary of the finite element mesh NIRFAST enforces the modified Robin boundary condition (equation 2.3). Resulting from the forward solution, data specifying the light intensity from each source at each detector is passed to the reconstruction routine. The reconstruction routine iteratively solves the inverse problem by adjusting material properties to match the source-detector intensity measurement data from the forward model or experimental system (for details see Dehghani [21]).

Available at no cost, the software consists of several MATLAB .m files and compiled executables that the user can adapt to integrate the program into their measurement apparatus. However, the multiple boundary conditions that need to be implemented for the endoscopic geometry go beyond the program's abilities. Endoscopic geometry will require two different boundary conditions: one on the probe surface and another on the simulation boundary. NIRFAST has been shown to work well for geometries with only one boundary, when that one boundary is a modified Robin (or Type III) boundary condition. This modified Robin boundary condition (as implemented within NIRFAST) assumes air is surrounding the simulation domain [22]. A Robin boundary condition is another name for a Neumann boundary condition [23]. The modified Robin boundary condition uses an additional term in the boundary condition to account for a difference in refractive index across the boundary [1,17]. For the endoscopic geometry the external simulation boundary must assume that *tissue*, not air, surrounds the simulation boundary. This is an important difference in terms of the refractive index calculation part of the modified Robin boundary condition [17,22].

2.3.2 COMSOL Multiphysics

COMSOL Multiphysics is a commercial software package that solves many different differential and partial differential equations using finite element techniques. It is an extremely powerful software package with built-in models and functions in almost every scientific and engineering field, except diffuse optical tomography. To use the diffusion equation in COMSOL, a general partial-differential equation (PDE) model is chosen where the coefficients are specified by the user. The general coefficient PDE equation has the form [23]

$$\nabla \cdot \left(-c\nabla u - \alpha u + \gamma\right) + \beta \cdot \nabla u + au = f \tag{2.7}$$

With boundary conditions specified by [23]

$$\text{Neumann: } \hat{n} \cdot \left(c\nabla u + \alpha u - \gamma\right) + qu = g \tag{2.8}$$

$$\text{Dirichlet: } hu = r \tag{2.9}$$

To enter the frequency domain diffusion approximation equation, equation 2.2, the required coefficients are

$$
\begin{array}{ll}
d_a = 0 & e_a = 0 \\
a = \mu_a + \dfrac{j\omega}{v} & \alpha = 0 \\
 & \beta = 0 \\
c = \dfrac{1}{3(\mu_a + \mu_s')} & \gamma = 0 \\
 & g = 0 \\
f = \exp\left[\dfrac{x^2}{2\sigma}\right]\exp\left[\dfrac{y^2}{2\sigma}\right]\exp\left[j0.15\right] & q = \dfrac{1}{2\mathsf{A}}
\end{array}
\tag{2.10}
$$

Where:

- v is the speed of light in the medium
- A is the refractive index mismatch term [17]
- a is a constant that uses the absorption coefficient and modulation frequency
- c is the diffusion coefficient, κ
- f is a source term. It is defined here as a Gaussian distributed beam source in the x – y plane with a beam spread factor of σ.

COMSOL also has a powerful CAD (computer aided design) interface to create objects and create a mesh. Using COMSOL’s mesh export feature and some coding in MATLAB to interpret the COMSOL file, a FEM mesh can be created for NIRFAST.

3 Validation of New Simulation

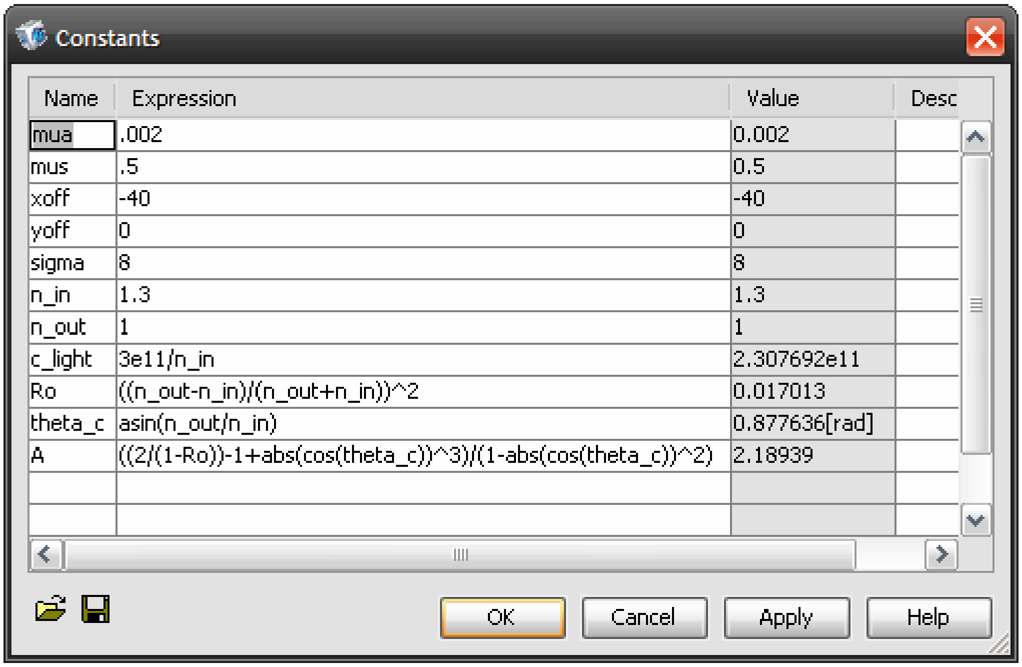
This chapter specifies how to implement the diffusion approximation in COMSOL Multiphysics, and execute the necessary NIRFAST code. Six metrics are used to compare simulations; the calculation and purpose of each metric is explained. To ensure simulations are comparable, the source expression is evaluated for COMSOL and NIRFAST simulations to ensure they are a match.

To create a new simulation for the endoscopic geometry, it must first be validated against a problem with a known solution. Two methods to validate a simulation are measurement data and data from another simulation that has been previously validated. Using the latter approach, NIRFAST is a convenient comparison tool for the first development stages of the new endoscopic model since, this endoscopic model is going to use the diffusion approximation to the radiative transfer equation, like NIRFAST. Fluence calculations over the external ring geometry (like figure 1.1) by NIRFAST will be used as the metric to validate the accuracy of the COMSOL simulation. The fluence is an ideal parameter to use for validation since it is the direct result of solving the frequency domain diffusion approximation. Also, it is for this external ring geometry of 86mm diameter that NIRFAST has been shown to provide accurate results [21, 22]. Both simulations will use identical mesh node coordinates to help keep the results comparable.

The first task to validate the new simulation is to ensure the Gaussian beam source for both NIRFAST and COMSOL are the same. It is assumed that the NIRFAST source is a Gaussian shaped pulse, however its shape and spread will have be interpreted from the data output since the source code to create the source is not available to the end user. The next task will be to verify the fluence calculated by COMSOL is nearly the same as NIRFAST, with the NIRFAST default Robin boundary conditions. The first simulation comparison will be over a homogeneous medium, then the next comparison will be a heterogeneous medium. The effects of a denser mesh will also be evaluated for each simulation case.

3.1 COMSOL Simulation Parameters

To use the diffusion equation in COMSOL, a time-dependent "Partial Differential Equation, Coefficient Form" model is chosen where the PDE coefficients are specified by the user. First there are several constants defined for the simulation:



Name	Expression	Value	Desc
mua	.002	0.002	
mus	.5	0.5	
xoff	-40	-40	
yoff	0	0	
sigma	8	8	
n_in	1.3	1.3	
n_out	1	1	
c_light	3e11/n_in	2.307692e11	
Ro	((n_out-n_in)/(n_out+n_in))^2	0.017013	
theta_c	asin(n_out/n_in)	0.877636[rad]	
A	((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)	2.18939	

Figure 3.1: Simulation Constants

(Screenshot from COMSOL Multiphysics v3.3)

Whereby each variable can be described as:

- `mua` Absorption coefficient (mm^{-1})
- `mus` Reduced scattering coefficient (mm^{-1})
- `xoff` X coordinate for the source (mm)
- `yoff` Y coordinate for the source (mm)
- `sigma` Constant that determines the width of the Gaussian source
- `n_in` Index of refraction for the medium

- `n_out` Index of refraction outside the medium
- `c_light` Speed of light in the medium (mm/s)
- `Ro` Reflection Coefficient
- `theta_c` Critical angle
- `A` Refractive index mismatch term for the modified Robin boundary condition

By using the units indicated above for speed of light, absorption coefficient, and reduced scattering coefficient, the COMSOL distance unit can be treated as millimeters. Also, this simple conversion within COMSOL prevents converting the mesh coordinates to millimeters for importation into NIRFAST; since, NIRFAST assumes all coordinate locations are in millimeters.

The simulation geometry is defined by drawing a 86mm diameter circle centered at the origin under the COMSOL editor Draw Mode.

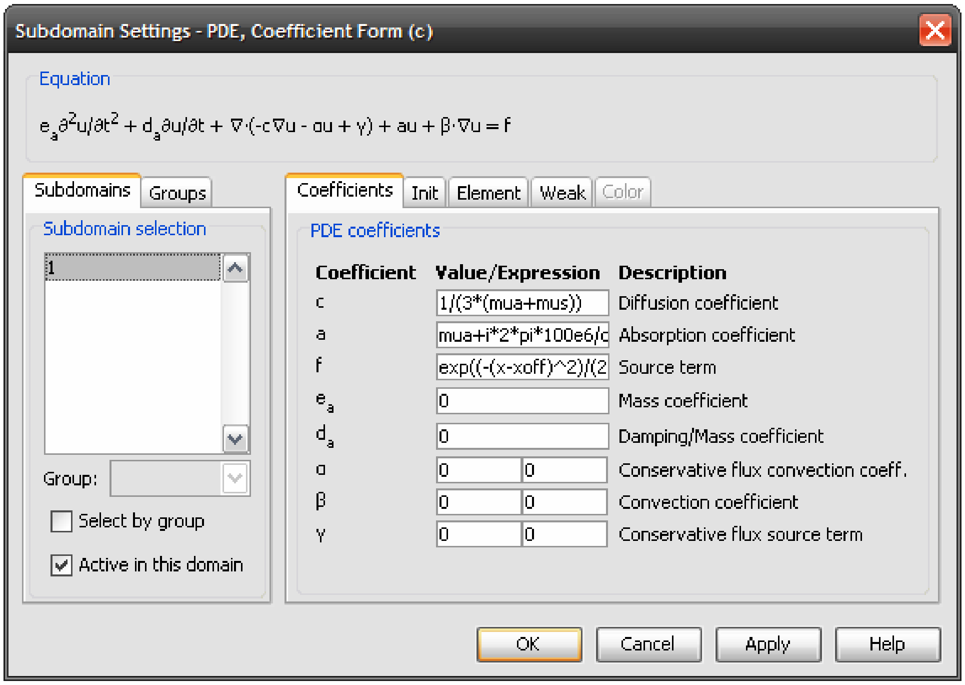
The general coefficient time-dependent PDE equation has the form [23]

$$e_a \frac{\partial^2 u}{\partial t^2} + d_a \frac{\partial u}{\partial t} + \nabla \cdot \left(-c\nabla u - \alpha u + \gamma \right) + \beta \cdot \nabla u + au = f \tag{3.1}$$

To enter the frequency domain diffusion equation into COMSOL, the subdomain settings are defined as shown in figure 3.2. Since the full length of the terms in figure 3.2 are hidden from view the coefficients have been expressed in equation 3.2.

$$\begin{aligned} c &= \texttt{1/(3*(mua+mus))} \\ a &= \texttt{mua+i*2*pi*100e6/(c\_light)} \\ f &= \texttt{exp((-(x-xoff)^2)/(2*sigma))*} \\ &\qquad \texttt{exp((-(y-yoff)^2)/(2*sigma))*exp(i*.15)} \end{aligned} \tag{3.2}$$

The source term, f , describes a Gaussian beam appearing within the subdomain at (`xoff`,`yoff`) coordinates.



Equation

$e_a\partial^2u/\partial t^2 + d_a\partial u/\partial t + \nabla\cdot(-c\nabla u - \alpha u + \gamma) + au + \beta\cdot\nabla u = f$

Subdomain selection

1

Group:

☐ Select by group

☑ Active in this domain

PDE coefficients

Coefficient	Value/Expression		Description
c	1/(3*(mua+mus))		Diffusion coefficient
a	mua+i*2*pi*100e6/c		Absorption coefficient
f	exp((-(x-xoff)^2)/(2		Source term
e_a	0		Mass coefficient
d_a	0		Damping/Mass coefficient
α	0	0	Conservative flux convection coeff.
β	0	0	Convection coefficient
γ	0	0	Conservative flux source term

Figure 3.2: Screenshot of Subdomain Settings

(Screenshot from COMSOL Multiphysics v3.3)

3.1.1 Modified Robin Boundary Condition

The COMSOL simulation will need to directly specify the modified Robin boundary condition as expressed in equation 2.3

$$\Phi(r) + 2\kappa A\hat{n}\cdot\nabla\Phi(r) = 0$$

into the form of equation 2.8

$$\hat{n}\cdot(c\nabla u + \alpha u - \gamma) + qu = g$$

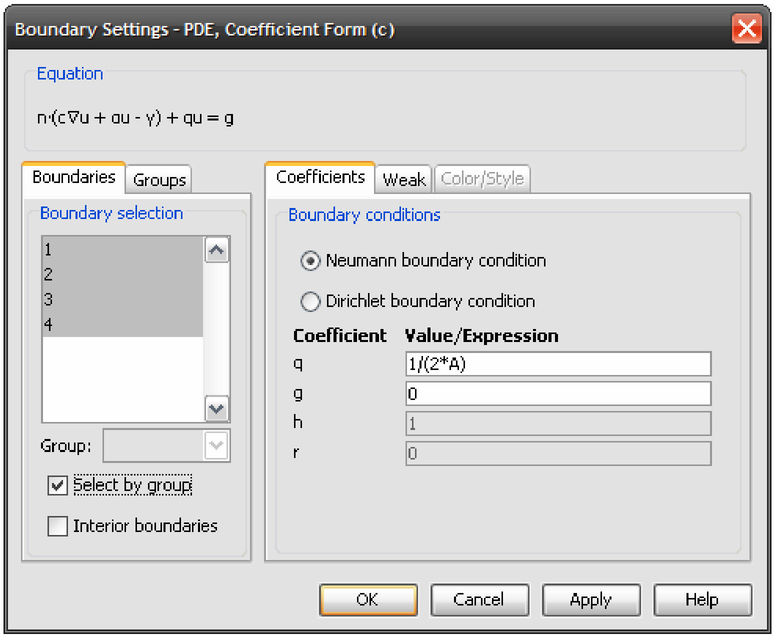
A direct comparison of the above two equations leads one to want to set the variable c in equation 2.8 to $2\kappa A$. However the variable c is already set within the subdomain settings as κ (the diffusion coefficient, equation 2.2) throughout the entire medium, and cannot be set to change its value at the boundaries. The only variables that are uniquely defined at the boundary are q and g. Variables $\alpha = \gamma = 0$ from the subdomain settings and setting $g = 0$, equation 2.8 can be written as

$$qu + c\hat{n}\cdot\nabla u = 0 \qquad (3.3)$$

Since $c = \kappa$, the diffusion coefficient, multiplying equation 3.3 by $2\mathrm{A}$ and setting $q = \dfrac{1}{2\mathrm{A}}$ produces the equivalent of equation 2.3, the modified Robin boundary condition.

$$2\mathrm{A}\left(qu + c\hat{n}\cdot\nabla u = 0\right) \Rightarrow \Phi\left(r\right) + 2\kappa\mathrm{A}\hat{n}\cdot\nabla\Phi\left(r\right) = 0 \tag{3.4}$$

Below, figure 3.3 shows a screenshot of the boundary settings dialog box with the correct settings for the homogeneous medium modified Robin boundary condition.



Boundary Settings - PDE, Coefficient Form (c)

Equation

n·(c∇u + αu - γ) + qu = g

Boundary selection: 1, 2, 3, 4

☑ Select by group

□ Interior boundaries

Boundary conditions

● Neumann boundary condition

○ Dirichlet boundary condition

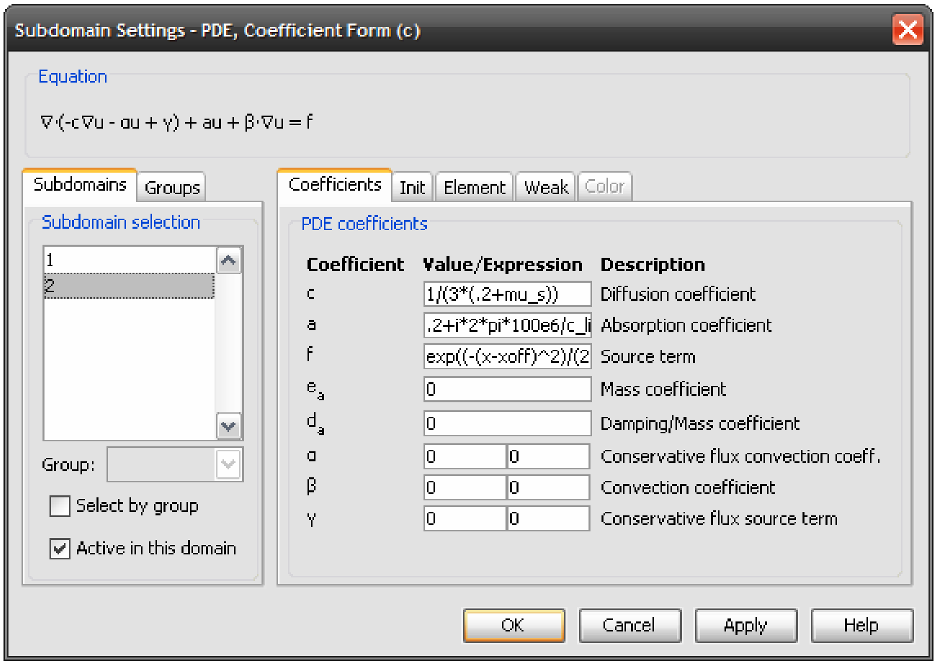
Coefficient	Value/Expression
q	1/(2*A)
g	0
h	1
r	0

Figure 3.3: Boundary Settings

(Screenshot from COMSOL Multiphysics v3.3)

3.1.2 Heterogeneous Media

The heterogeneous simulations made of two or more homogeneous subdomains, with different optical properties, that are part of the same simulation domain. The subdomain settings now must be applied to two subdomains: the background medium and the target. The background medium subdomain settings remain identical to the homogeneous case defined in figure 3.2 and equation 3.2. Figure 3.4 shows the target subdomain settings are identical to the homogeneous case, except the `mua` term has been replaced with the target's absorption coefficient value. The same Neumann boundary conditions defined above in figure 3.3 were entered for the external boundaries, while the internal boundaries were left undefined as grayed-out terms; figure 3.5(a) shows the external boundary conditions, while figure 3.5(b) shows the disabled terms for the interior boundaries.



Equation

$\nabla\cdot(-c\nabla u - \alpha u + \gamma) + au + \beta\cdot\nabla u = f$

Subdomain selection

1
2

Group:

☐ Select by group

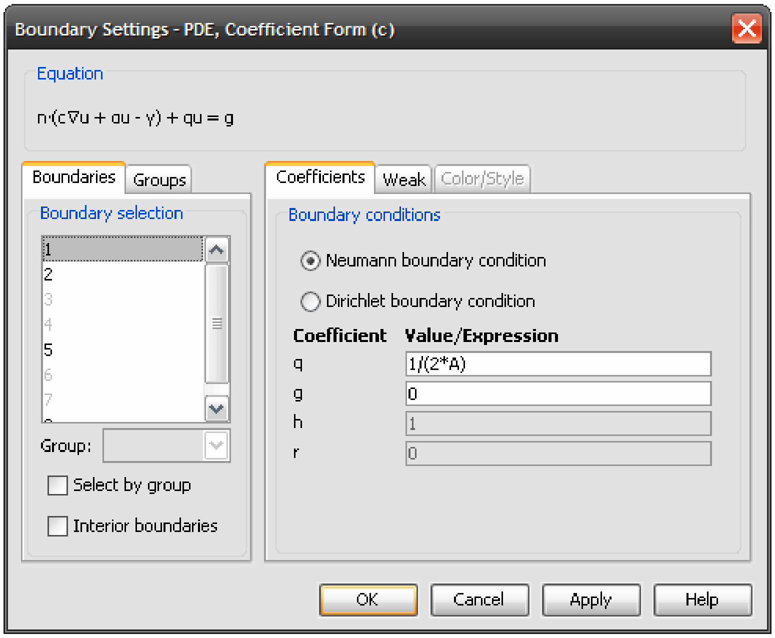
☑ Active in this domain

PDE coefficients

Coefficient	Value/Expression		Description
c	1/(3*(.2+mu_s))		Diffusion coefficient
a	.2+i*2*pi*100e6/c_li		Absorption coefficient
f	exp((-(x-xoff)^2)/(2		Source term
e_a	0		Mass coefficient
d_a	0		Damping/Mass coefficient
α	0	0	Conservative flux convection coeff.
β	0	0	Convection coefficient
γ	0	0	Conservative flux source term

Figure 3.4: Target Subdomain Settings

(Screenshot from COMSOL Multiphysics v3.3)



Equation

$n\cdot(c\nabla u + \alpha u - \gamma) + qu = g$

Boundary selection

1
2
3
4
5
6
7

Group:

☐ Select by group

☐ Interior boundaries

Boundary conditions

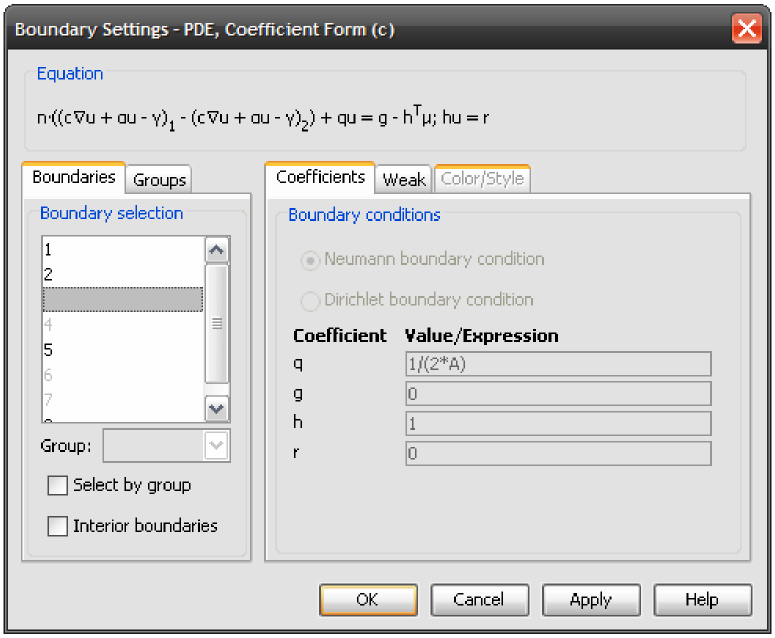
● Neumann boundary condition

○ Dirichlet boundary condition

Coefficient	Value/Expression
q	1/(2*A)
g	0
h	1
r	0

Figure 3.5(a): Heterogenous Boundary Settings for Exterior

(Screenshot from COMSOL Multiphysics v3.3)



Equation

$n\cdot((c\nabla u + \alpha u - \gamma)_1 - (c\nabla u + \alpha u - \gamma)_2) + qu = g - h^T\mu;\ hu = r$

Boundary conditions

● Neumann boundary condition

○ Dirichlet boundary condition

Coefficient	Value/Expression
q	1/(2*A)
g	0
h	1
r	0

Figure 3.5(b): Heterogeneous Boundary Settings for Interior

(Screenshot from COMSOL Multiphysics v3.3)

3.2 NIRFAST Simulation Parameters

The NIRFAST simulation is executed and configured within the MATLAB interface. The NIRFAST simulation parameters were set to the same external ring geometry, material parameters, and Gaussian beam source as the COMSOL simulation. Furthermore, the exact same node coordinates used for the COMSOL simulation were loaded into MATLAB, and made executable by adding additional mesh parameters necessary for NIRFAST simulation execution. NIRFAST requires more than one source to execute, however only the fluence calculations from one source was used for comparison. Other terms within NIRFAST that must be defined are the boundary nodes, source locations, and detector locations. As an example, Figure 3.6 shows the location of four sources (circles) and four detectors (Xs) added at 90° increments around the boundary ring, and the boundary nodes (dots) for the homogeneous media case.

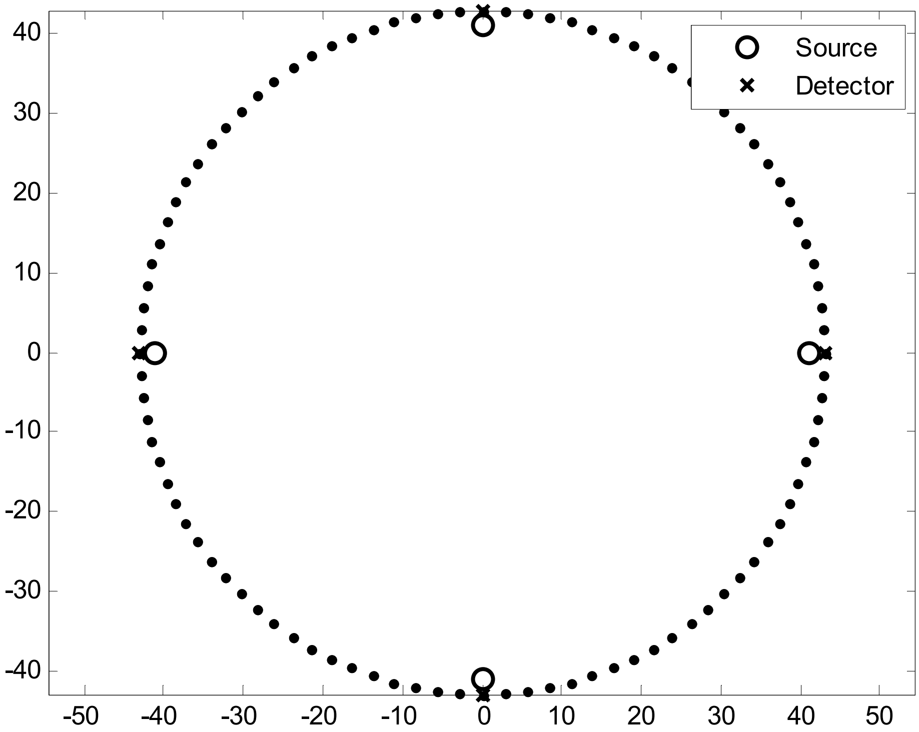


Figure 3.6: NIRFAST sources, detectors, and boundary nodes

3.2.1 Modified Robin Boundary Condition

From section 2.2.3.2, the definition of the modified Robin boundary condition is given. It is a mixed or Neumann boundary condition that takes into account the index of refraction of the media on both sides of the boundary. Within the NIRFAST mesh struct, the variable `ksi` is used in to account for the index of refraction mismatch at the boundary. The value of `ksi` is calculated by

$$\mathtt{ksi} = \frac{1}{2\mathrm{A}} \tag{3.5}$$

Where A is exactly as specified in equations 2.4, 2.5, and 2.6. It cannot be confirmed within the NIRFAST code that the boundary conditions are implemented in the same way COMSOL implements the boundary condition because the value of `mesh.ksi` is used within a compiled executable to create the finite element matrices. However, since the `mesh.ksi` value corresponds to the value for q in figure 3.5(a), the implementation may be similar.

For NIRFAST, the default interior index of refraction value is 1.3 which approximates the tissue as water. The user is capable of changing the index of refraction of the interior medium. The outer index of refraction is defined within the code to be air, at a value of 1.

3.2.2 Heterogeneous Media

Heterogeneous media simulations are created by running the additional NIRFAST function `add_blob.m`. This function will prompt the user for the target center coordinates, the radius of the target, absorption coefficient, reduced scattering coefficient, index of refraction, and region number. The function makes the necessary modifications to the parameters at the effected nodes.

3.3 Comparison Parameters

A total of six different metrics are used to assess the differences between simulation results between COMSOL and NIRFAST, and later, COMSOL and FEDA. All of the metrics are calculated using values from a 1mm by 1mm grid of interpolated values, except where noted the node values are used. The COMSOL interpolated values are obtained by the `postinterp` command, while the NIRFAST and FEDA interpolated values are obtained by the `griddata` function.

3.3.1 Fluence Comparison Plot

By plotting the fluence values for each simulation on the same graph, the match, or mismatch, between the values can be quickly assessed by visual inspection. The linear value plot allows better comparison of the fluence near the source. A log plot will exaggerate the differences far from the source region, where fluence values are small.

3.3.2 Difference Plot

A global difference plot is a quick assessment of the differences between the two simulations across the entire medium. This information can help troubleshoot simulation problems. For example, simulations where the difference is along a boundary could indicate an problem with the boundary conditions. The plot is constructed by using the magnitude normalized linear values of the fluence, Φ. The difference is simply calculated by

$$\Phi_{TEST} - \Phi_{REFERENCE} \tag{3.6}$$

A slice of the global fluence difference plot, through the Gaussian beam source, of the medium can help to see subtle changes in the fluence over the medium that may not be easy to see on the global difference plot. Sudden changes in difference on this plot can indicate problems with the simulation.

3.3.3 Error Calculations

Each plot has a maximum difference value for the data shown in the plot. Difference values are calculated as specified in equation 3.6. The maximum difference term can be useful when comparing different mesh densities to monitor the reduction or increase in fluence difference.

Each plot also has a RMS error value to provide a single number to represent the average squared difference across the entire simulation domain. This error number is also useful for quantizing the ‘match’ between simulations. The RMS error is calculated according to

$$RMS_{Error} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}\left[\Phi_{TEST}\left(i\right) - \Phi_{REFERENCE}\left(i\right)\right]^{2}} \tag{3.7}$$

Where i is the node and N is the total number of nodes. The calculation algorithm accounts for the missing interpolation values in the square grid that results from having a circle of values.

3.4 Source Evaluation

Reproducing the NIRFAST source term in COMSOL is crucial to compare the simulations against each other. The amplitude can be normalized out for comparisons, however the source width must be identical between simulations.

3.4.1 NIRFAST source

The NIRFAST source is created within a complied MATLAB executable file `gen_source` using the node coordinates, elements, source coordinates, and full width half maximum value. The code comments within `femdata_stnd` indicate that the source is a Gaussian with a user-specified width. Furthermore, the source values are complex with a phase of .15 radians. Figure 3.7 shows the real values of the NIRFAST source plotted over a 1mm by 1mm interpolation grid at X = -41mm and Y = $0\,\text{mm}$ over a $86\,\text{mm}$ diameter circle. The source is modeled $2\,\text{mm}$ within the outer boundary since an incident laser beam can be modeled as a Gaussian point source one scattering distance, $1/\mu_s'$, from the point of entry [16].

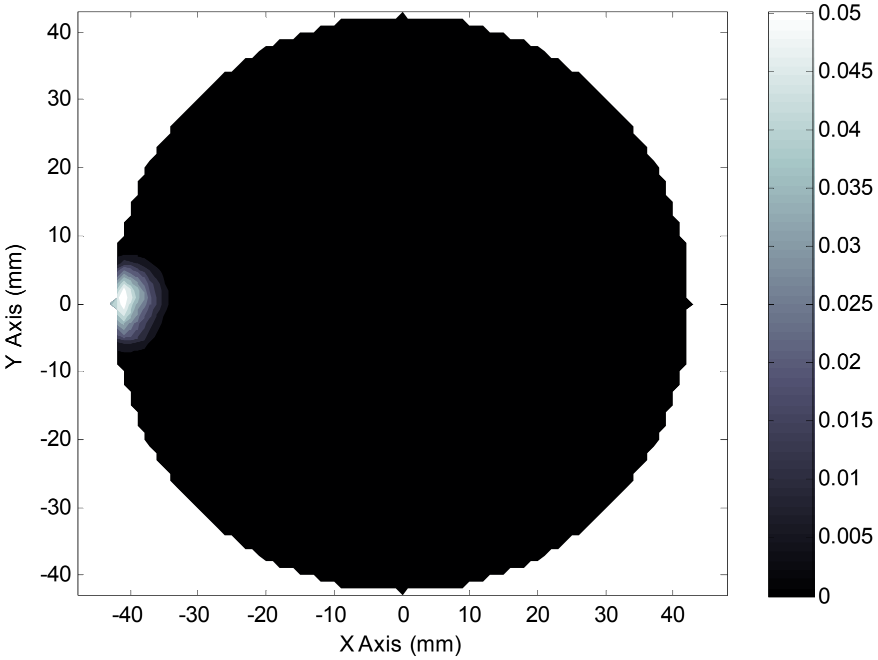


Figure 3.7: Default NIRFAST Source

3.4.2 Developing a COMSOL Source

The general equation for a Gaussian function is

$$\frac{1}{\sigma\sqrt{2\pi}}\exp\left[-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2\right] \tag{3.8}$$

The terms outside the exponential can be ignored since the output will be normalized for comparison to NIRFAST. To achieve a full width half maximum (FWHM) of $3\,\text{mm}$ (set $x = 1.5\,\text{mm}$), equation 3.9 can be solved for the value of σ that will match NIRFAST,

$$\frac{1}{2} = \exp\left[-\frac{1}{2}\left(\frac{1.5}{\sigma}\right)^2\right] \Rightarrow \sigma = 1.274 \qquad (3.9)$$

This Gaussian equation is applied to spatial coordinates in two dimensions by multiplying Gaussian expressions for each dimension as,

$$\exp\left[-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2\right]\exp\left[-\frac{1}{2}\left(\frac{y}{\sigma}\right)^2\right]\exp\left[j*0.15\right] \qquad (3.10)$$

This equation produces the source plotted below in figure 3.8, using the same interpolation grid as figure 3.7. The difference plot and RMS error calculation in figure 3.9 will show that there is a difference between the normalized NIRFAST default source and Gaussian $3\,\text{mm}$ FWHM source. It can been seen by the dark regions that there are large areas where the sources are different. Therefore, the NIRFAST source will have to be matched by trial-and-error, or overwritten by equation 3.10.

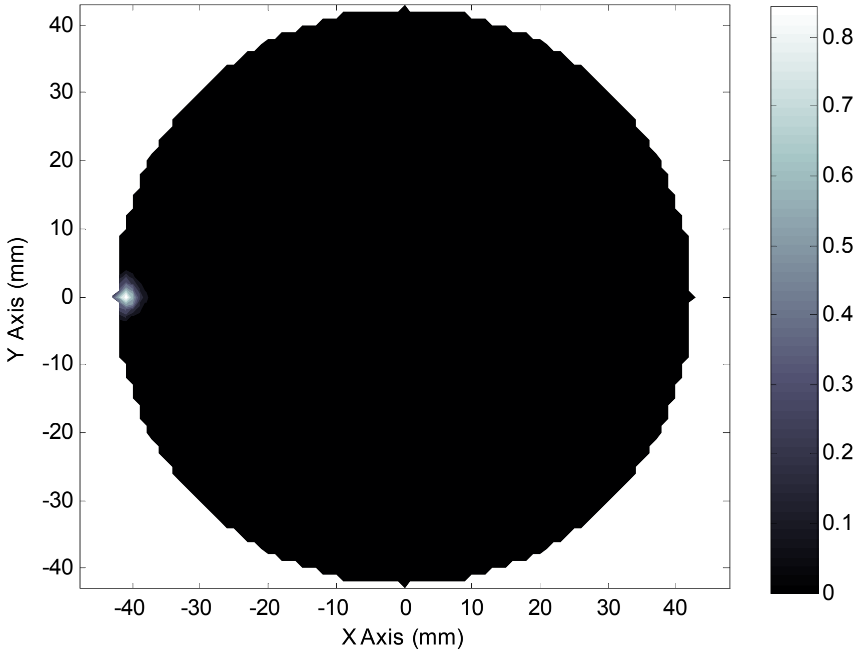


Figure 3.8: Gaussian Source for $\sigma = 1.274$

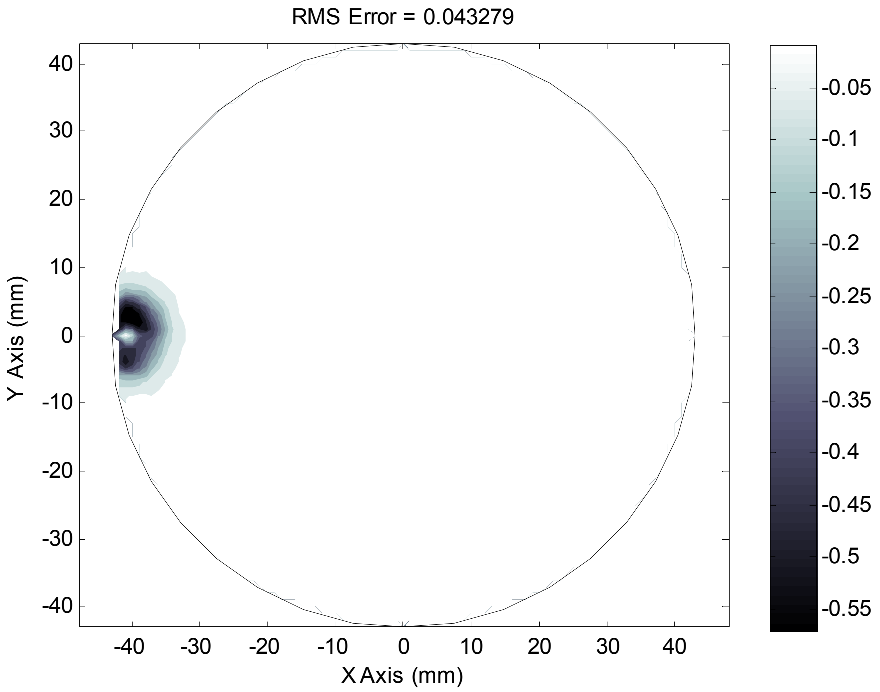


Figure 3.9: Difference Plot between NIRFAST Source and Gaussian 3mm FWHM

3.4.3 Overwriting the NIRFAST Source

The NIRFAST source can be overwritten by reassigning the values of the source variable, `qvec`, according to equation 3.10. The code change is made to `qvec` within `femdata_stnd.m` file after the `gen_source()` function is called. The following code is added to line 81 of `femdata_stnd.m`:

```
sigma=8;
for k=1:length(mesh.source.coord)
    xoff=mesh.source.coord(k,1);
    yoff=mesh.source.coord(k,2);
    qvec(:,k)=exp((-(mesh.nodes(:,1)-xoff).^2)/(2*sigma))...
        .*exp(-(mesh.nodes(:,2)-yoff).^2/(2*sigma))...
        .*complex(cos(.15),sin(.15));
end
```

It should be noted that the variable `sigma` is equivalent to σ^2 in equation 3.10; the value of 8 was empirically chosen to yield a similar, but not exact, beam width to the NIRFAST source. Ideally, the beam width needs to be the same as a laser beam yields, but since this is a comparison between simulations, as long as the source is the same, the results are comparable. The qvec expression matches the source expression, *f*, within COMSOL specified in equation 3.2. The source values interpolated to a 1mm by 1mm grid can be seen below in figure 3.10 for a 86mm diameter circular medium.

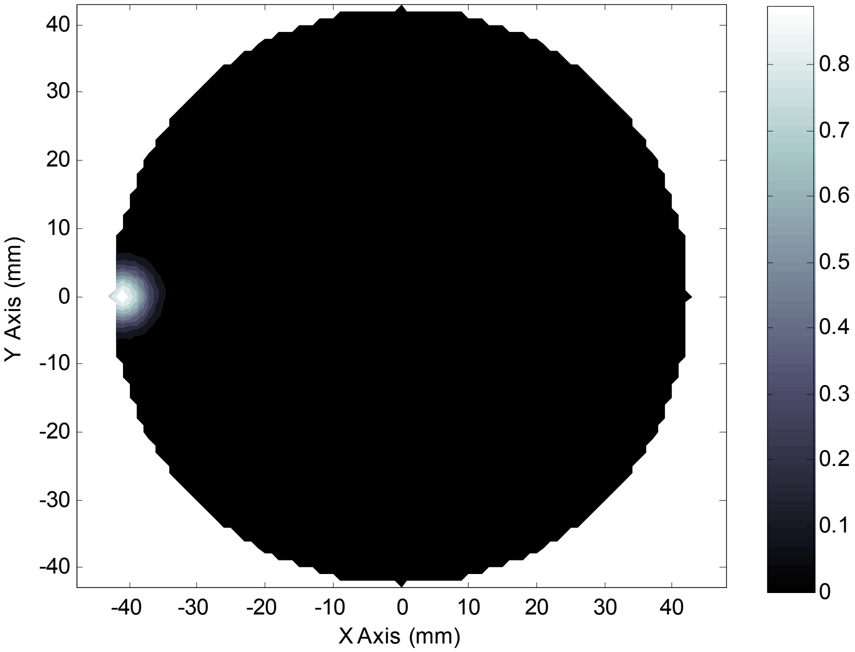


Figure 3.10: Magnitude Value Plot of Source Over the Medium

4 COMSOL Validation

The COMSOL simulation is evaluated against NIRFAST for the breast cancer geometry with and without a suspected tumor. The fluence difference in response to changing the external boundary condition from air/tissue to tissue/tissue is revealed. A sensitivity study evaluates the feasibility of the diffusion approximation to detect a realistic and ideal absorber.

A 86mm diameter external geometry with an air/tissue Robin boundary condition is chosen to coincide with previously validated NIRFAST results [1]. The source is location at X = -41mm and Y = 0mm. The absorption coefficient, μ_a, is $0.002\,\text{mm}^{-1}$. The reduced scattering coefficient, μ_s', is $0.5\,\text{mm}^{-1}$. The COMSOL fluence values where compared to NIRFAST's fluence calculations, according to the metrics described in section 3.3.

4.1 Homogeneous Media with Modified Robin Boundary Condition

The homogeneous media simulation contains absorption coefficient value of $0.2\,\text{mm}^{-1}$ and reduced scattering coefficient value of $0.5\,\text{mm}^{-1}$ that are constant throughout the entire simulation space. Figure 4.1(a)-(d) shows the results for a coarse mesh, while figure 4.2(a)-(d) shows the results for a dense mesh. The 2D value plots of figures 4.1(a) and 4.1(b) show that there are small differences throughout the medium. However, as indicated by figures 4.1(c) and 4.1(d), the largest differences between the simulations lie near the source location. These differences seem to be due to the mesh because the error is much less for the finer mesh. Figures 4.2(a)-(d) show the same trends as figures 4.1(a)-(d), but there is a better match throughout the medium; the global maximum difference and RMS error decrease from 0.094657 to 0.070797 and 0.012636 to 0.0043696.

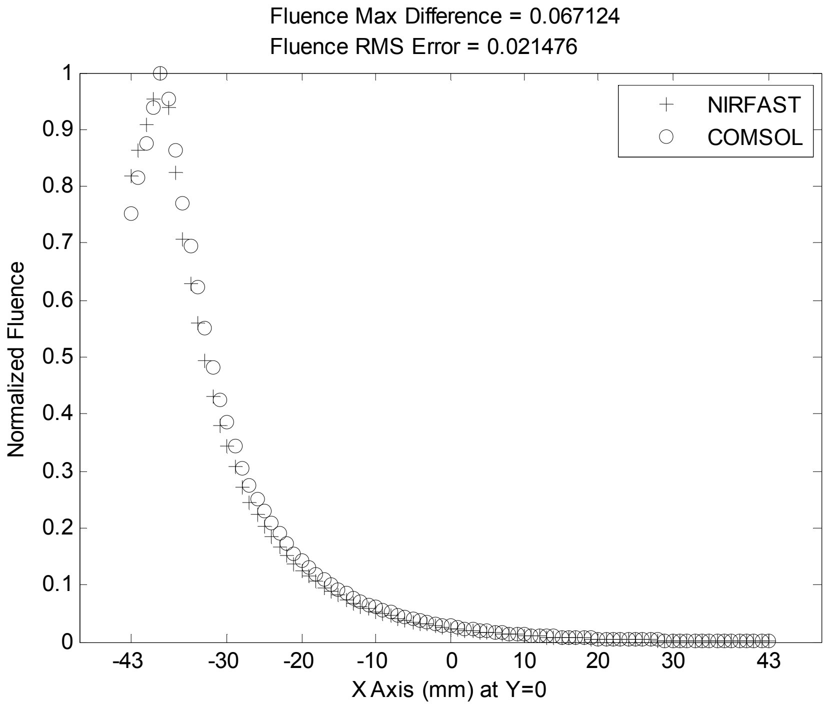


Figure 4.1(a): Linear Fluence Comparison for Homogeneous Media with Coarse Mesh

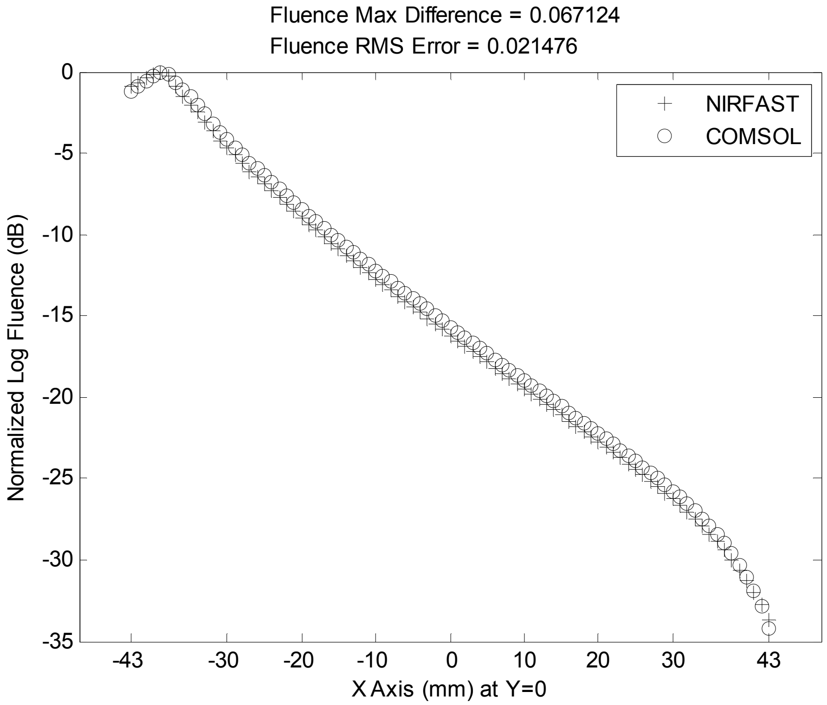


Figure 4.1(b): Log Fluence Comparison for Homogeneous Media with Coarse Mesh

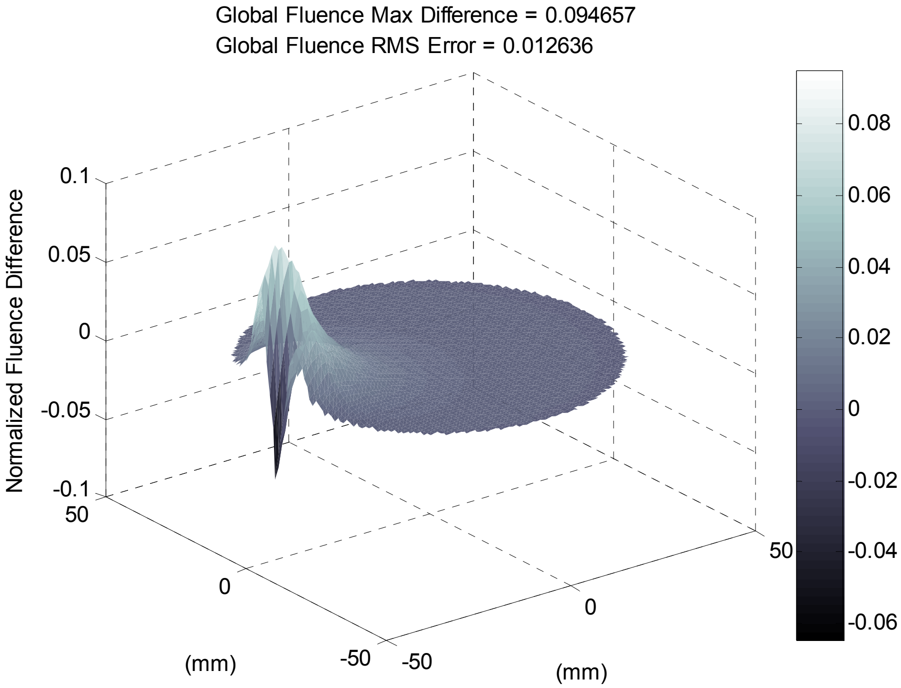


Figure 4.1(c): Global Fluence Difference Plot with Coarse Mesh

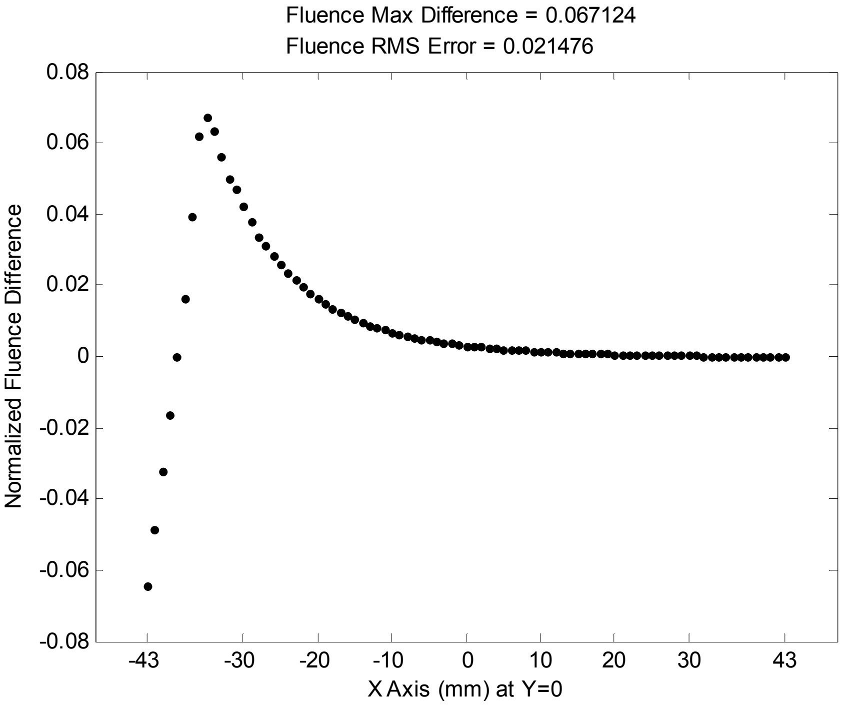


Figure 4.1(d): Fluence Difference Slice with Coarse Mesh

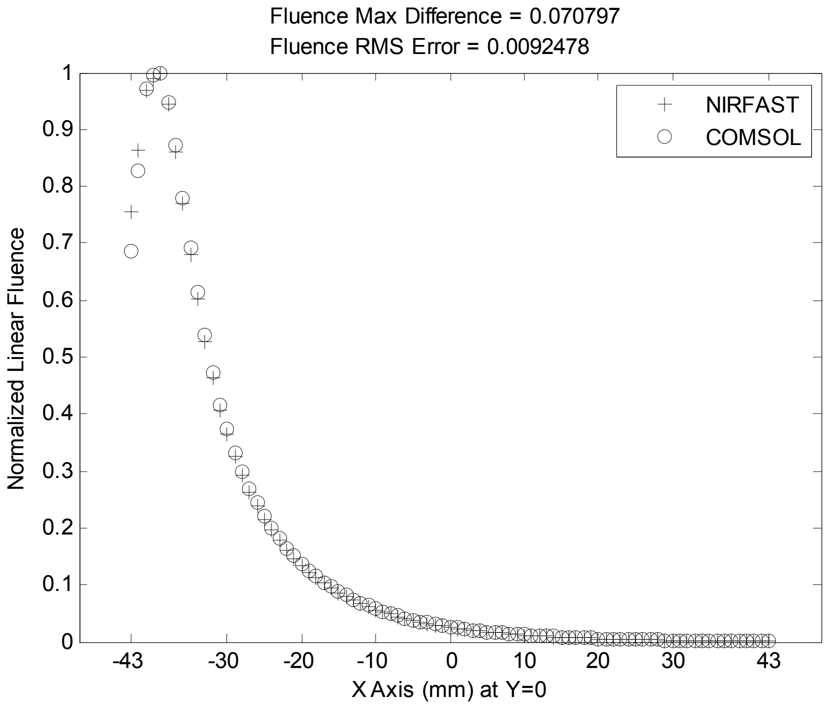


Figure 4.2(a): Linear Fluence Comparison for Homogeneous Media with Dense Mesh

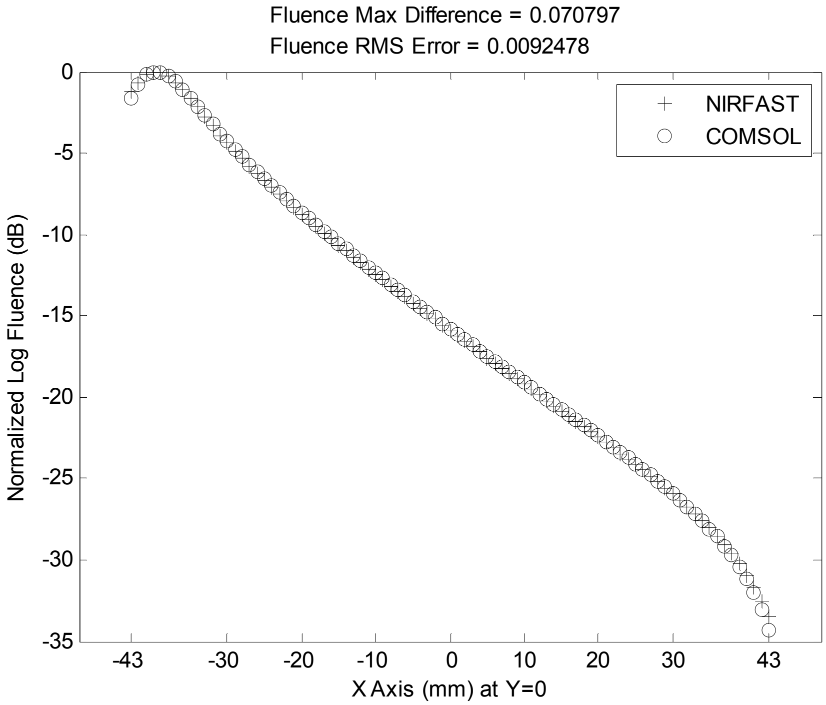


Figure 4.2(b): Log Fluence Comparison for Homogeneous Media with Dense Mesh

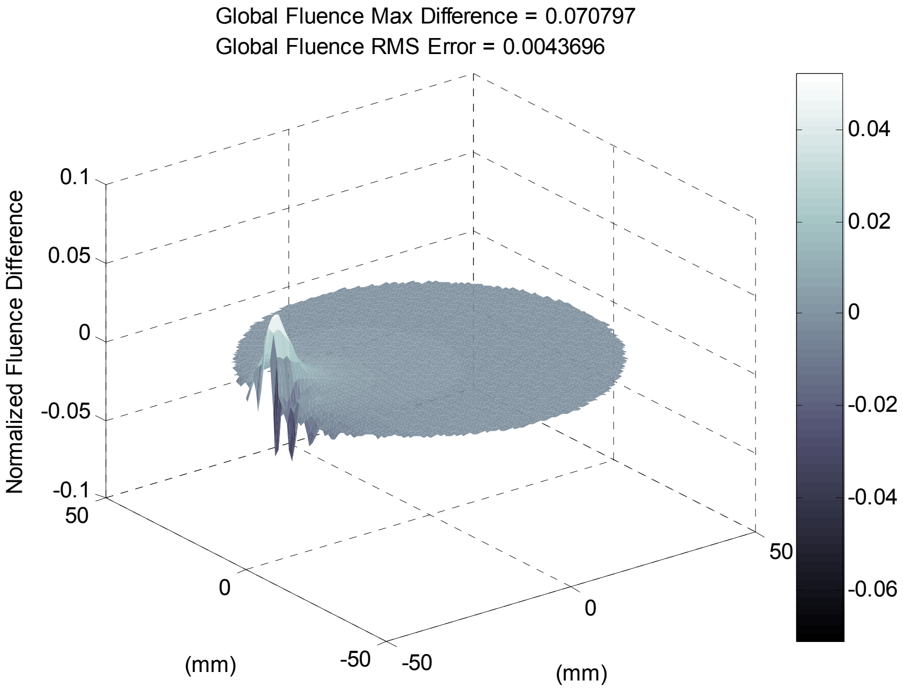


Figure 4.2(c): Global Fluence Difference Plot with Dense Mesh

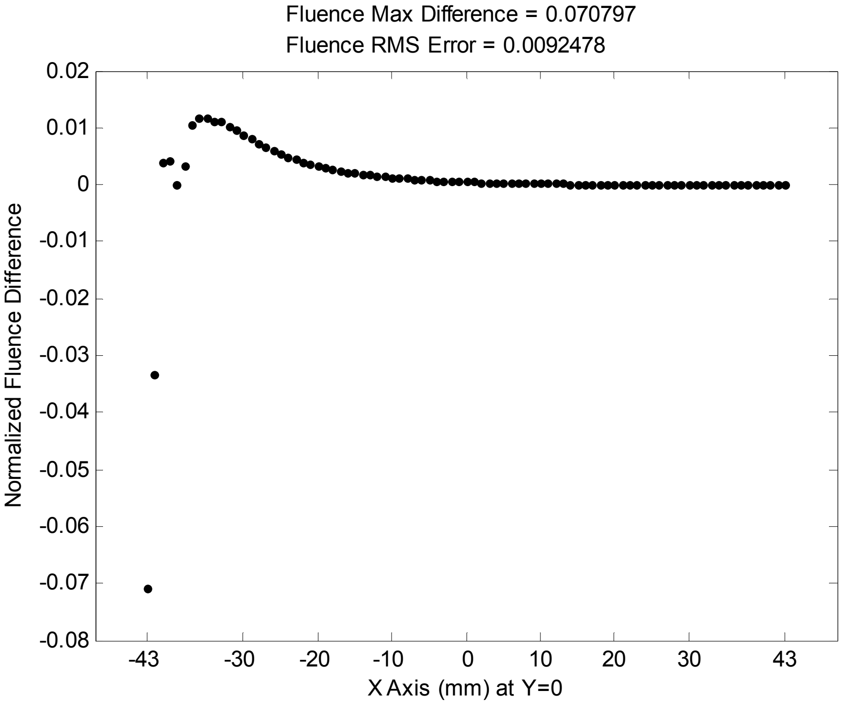


Figure 4.2(d): Fluence Difference Slice with Dense Mesh

4.2 Heterogeneous Media with Modified Robin Boundary Condition

For the heterogeneous media case the simulation was identical to the homogeneous case except a region, with a higher absorption coefficient, was added to the center of the simulation to approximate a suspected tumor. Illustrated in figure 4.3, the tumor suspect (or target) has been assigned an absorption coefficient, μ_a, value 100x the background at $0.2\,\mathrm{mm}^{-1}$. This extremely high absorption coefficient approximates an infinite absorber to help ensure that it is detectable, and it is relatively easy to reproduce in the lab, with respect to a prostate cancer phantom. However, this value is not realistic since the absorption coefficient for prostate cancer is only 26% greater than the prostate itself [24]. The reduced scattering coefficient, μ_s', remains the same value for the entire simulation domain at $0.5\,\mathrm{mm}^{-1}$.

As in the homogeneous case, the biggest error is found near the source, and the simulation error decreases as mesh density increases. Figures 4.4(a)-(d) show the results for a coarse mesh, and figures 4.5(a)-(d) show results for denser mesh. In figure 4.4(a), the fluence predicted by NIRFAST does not match at all with the COMSOL results between the source and the left boundary. However, with a denser mesh, figure 4.5(a) shows the NIRFAST fluence to match much better between the source and the boundary. It is clear that the coarse mesh had an impact upon the accuracy of NIRFAST, and COMSOL produces a more accurate result with the coarse mesh. It is suspected that the way NIRFAST calculates the source in the global FEM matrix assembly may be the cause of this error, however this cannot be confirmed as the NIRFAST source code is not available. There is a slight change of the error near the target boundary indicated by figures 4.4(d) and 4.5(d) . The cause of this is suspected to be due to subtle differences between COMSOL and NIRFAST in the target region boundary definition, but cannot be confirmed since information is not available for NIRFAST’s calculation methods.

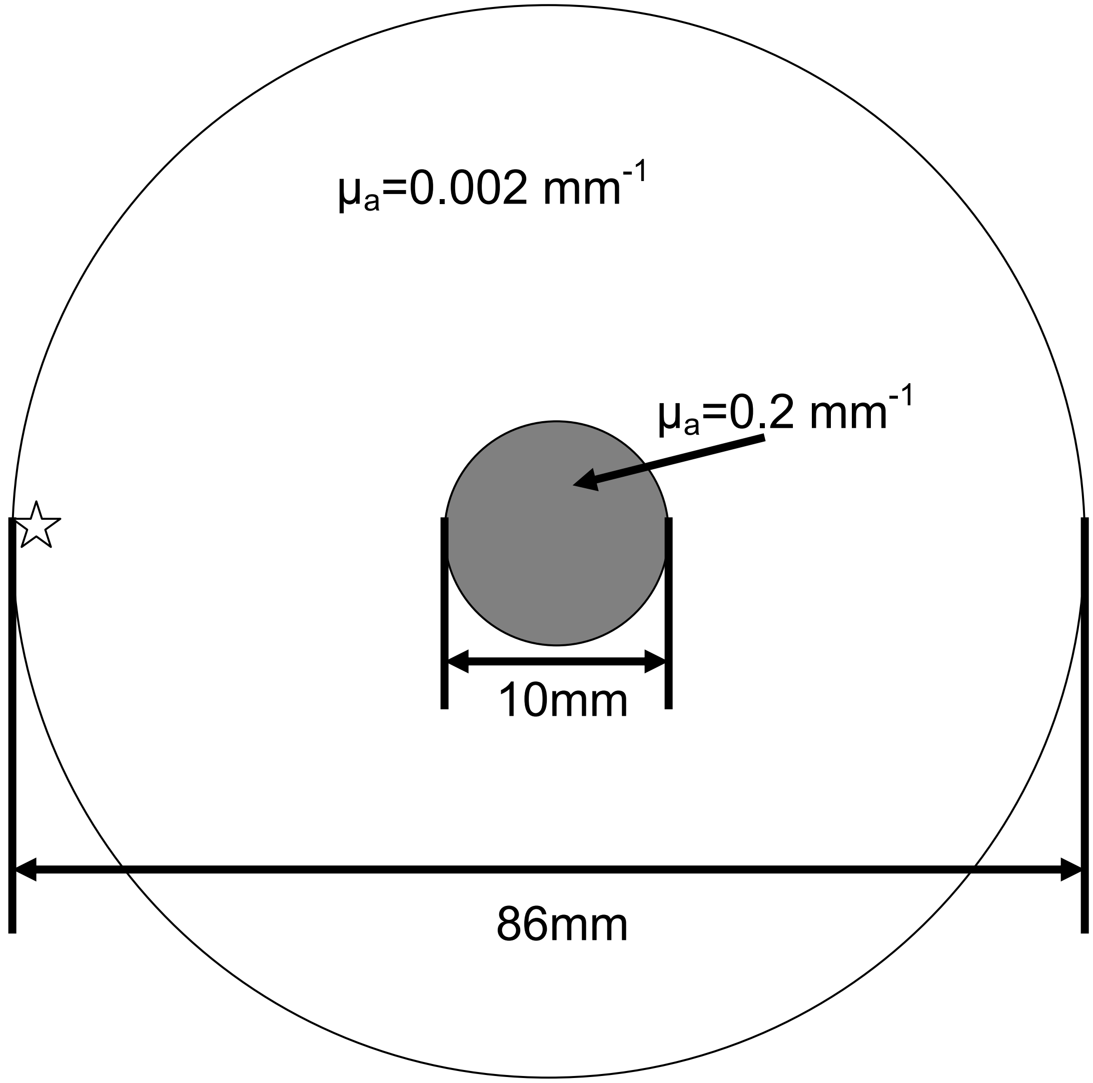


Figure 4.3: Heterogeneous Simulation Parameters

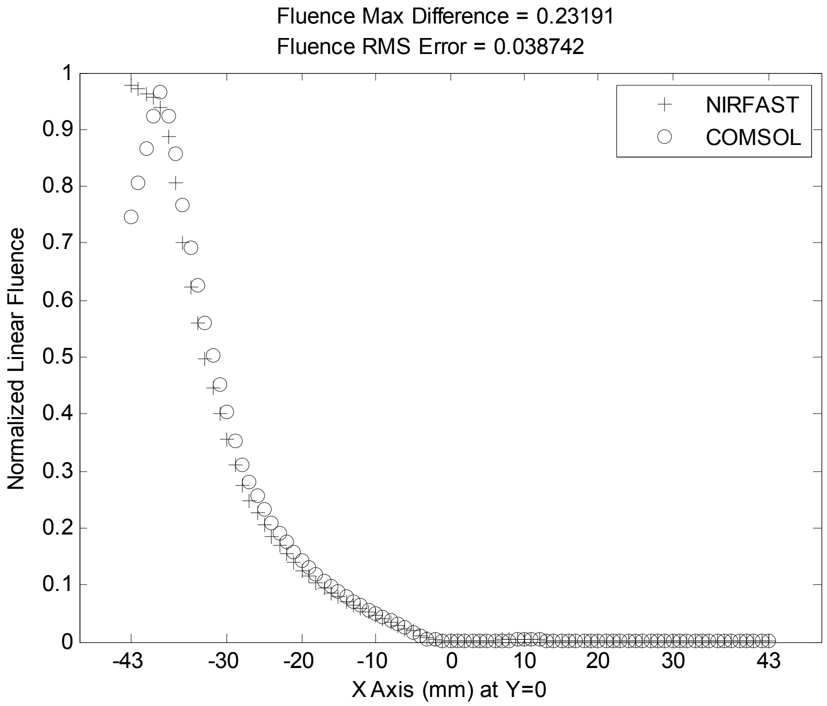


Figure 4.4(a): Linear Fluence Comparison for Heterogeneous Media with Coarse Mesh

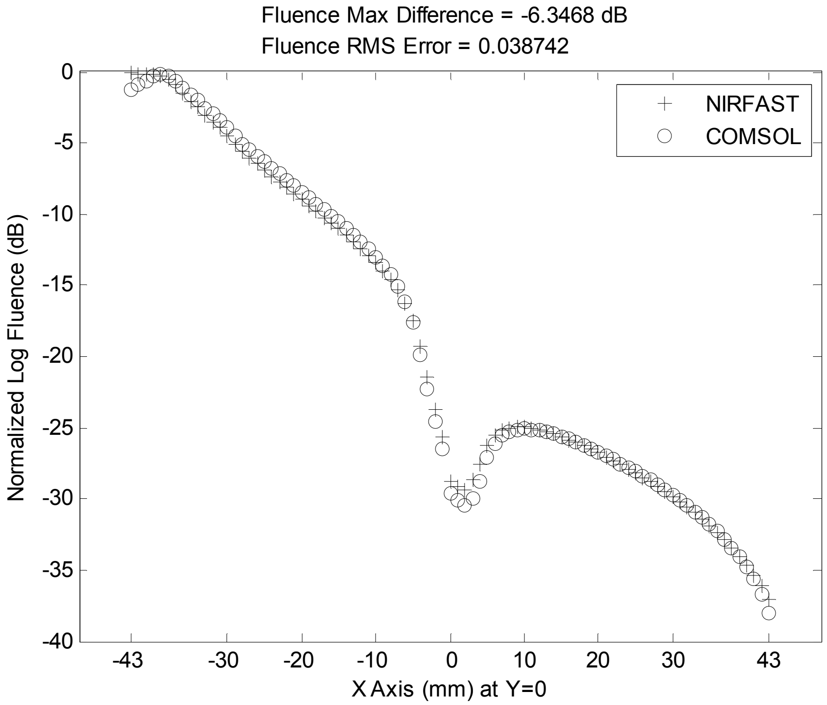


Figure 4.4(b): Log Fluence Comparison for Heterogeneous Media with Coarse Mesh

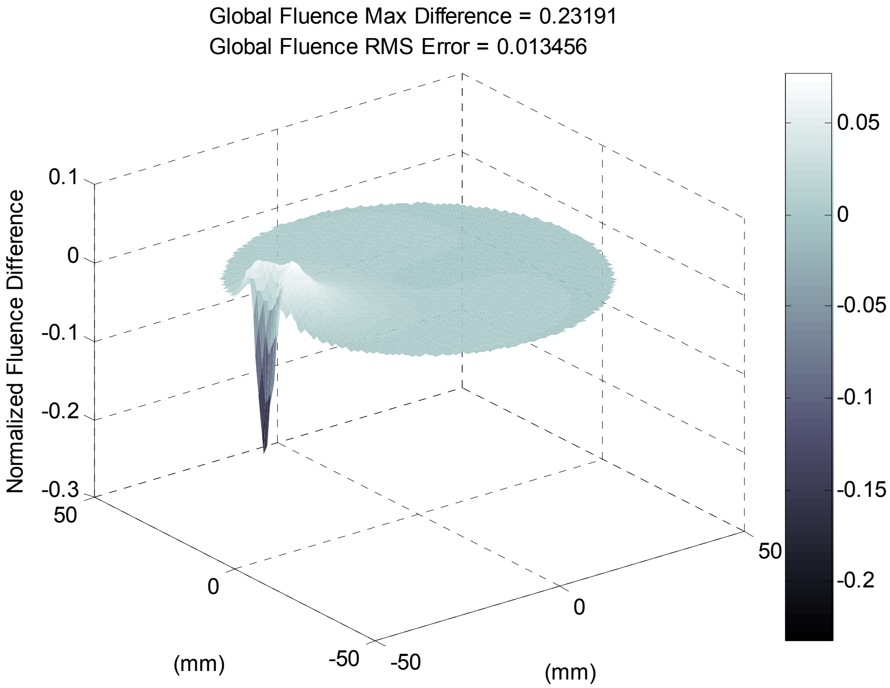


Figure 4.4(c): Global Fluence Difference Plot with Coarse Mesh

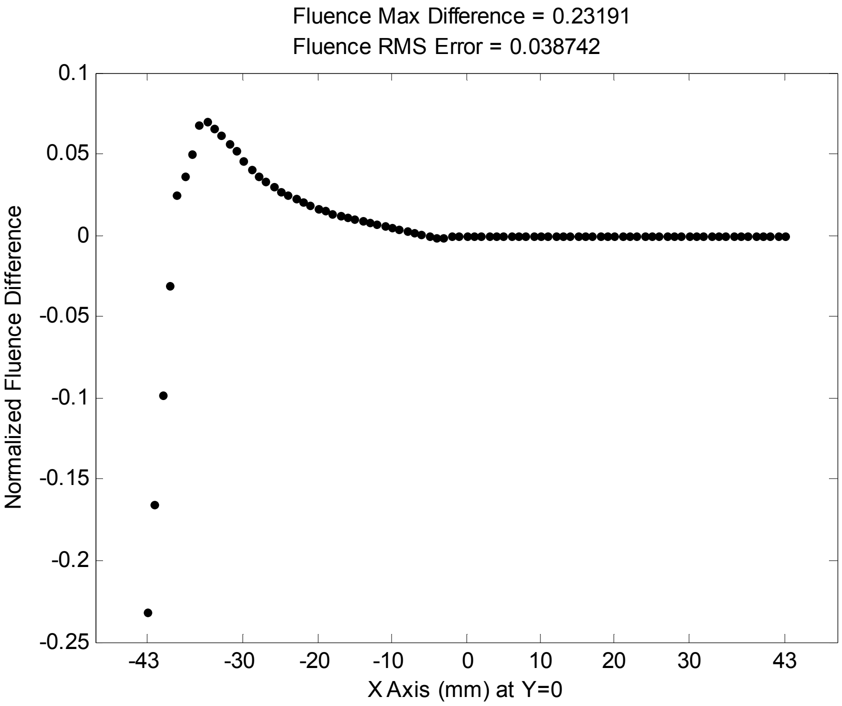


Figure 4.4(d): Fluence Difference Slice with Coarse Mesh

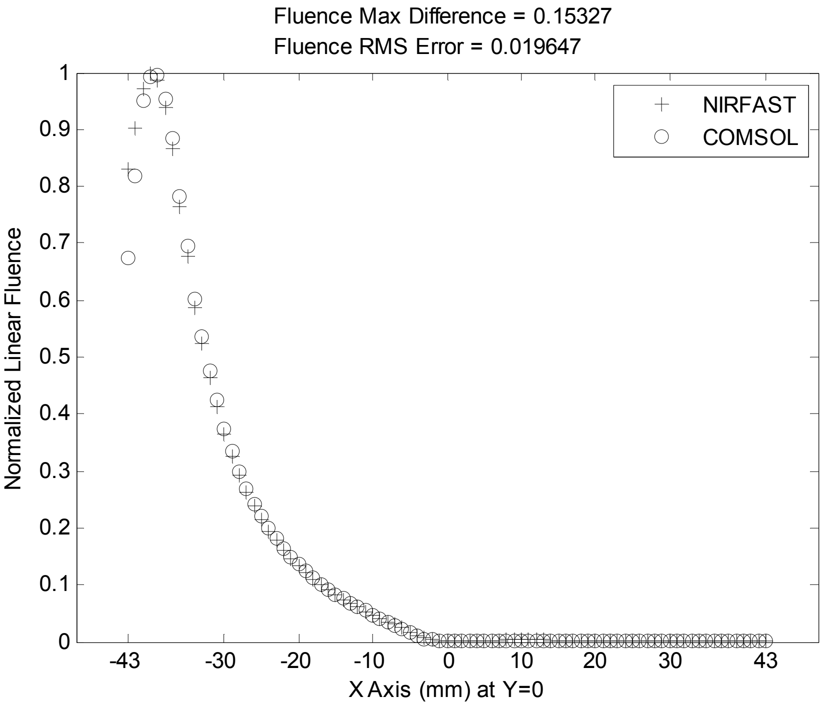


Figure 4.5(a): Linear Fluence Comparison for Heterogeneous Media with Dense Mesh

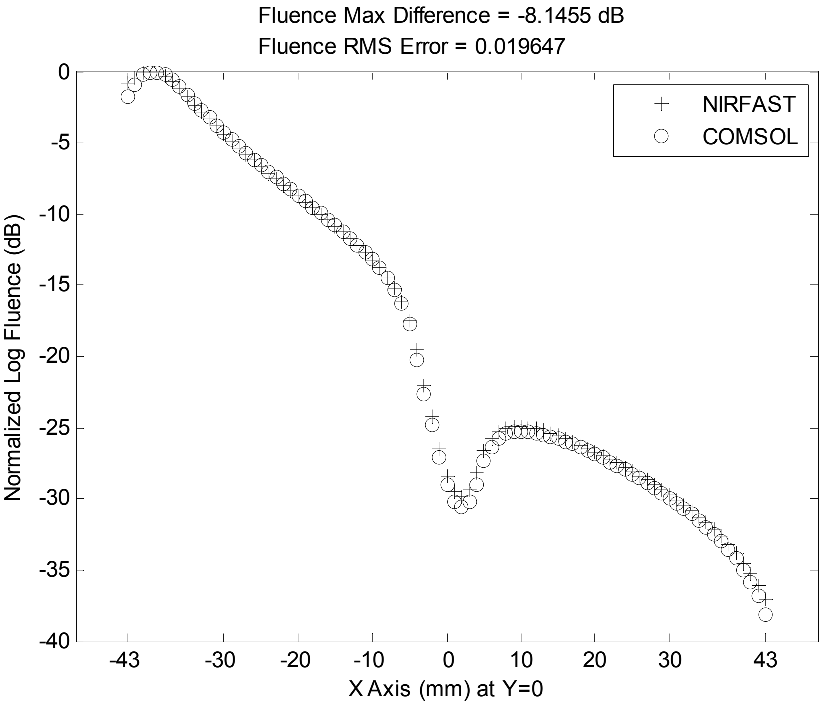


Figure 4.5(b): Log Fluence Comparison for Heterogeneous Media with Dense Mesh

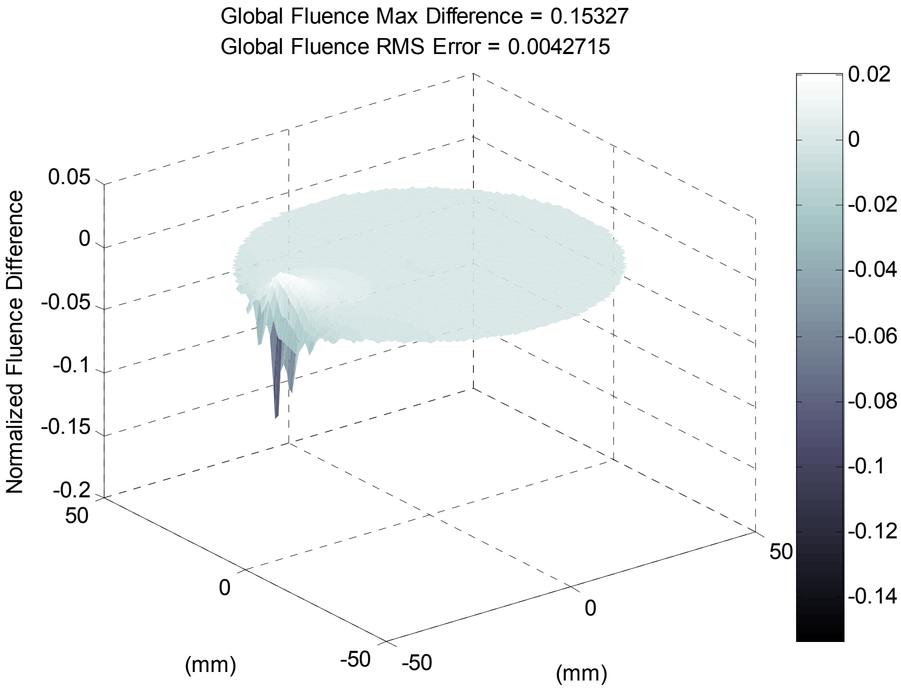


Figure 4.5(c): Global Fluence Difference Plot with Dense Mesh

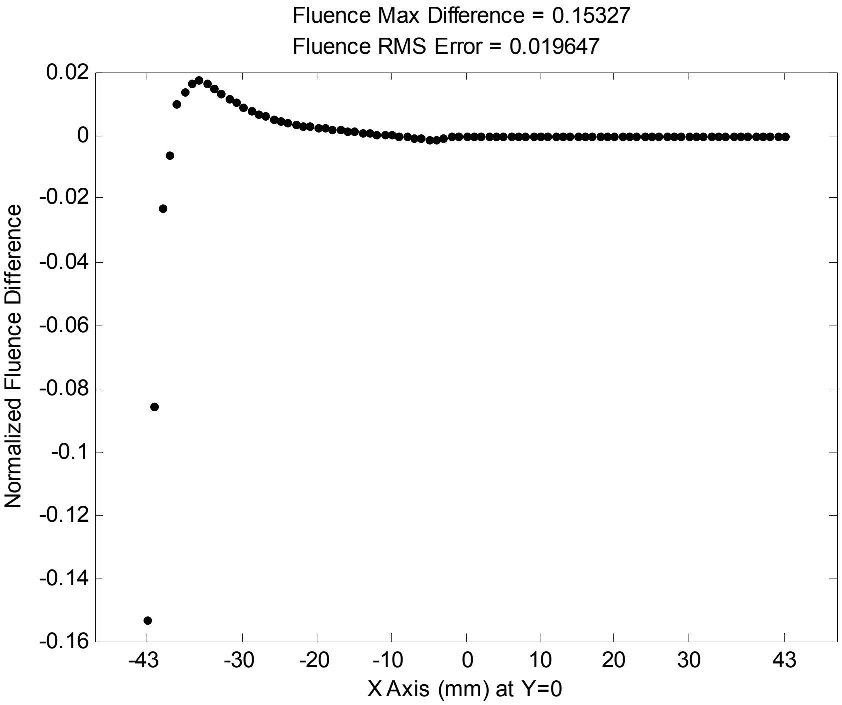


Figure 4.5(d): Fluence Difference Slice with Dense Mesh

4.3 Endoscopic Media

Now that the COMSOL simulation has been verified to accurately predict the fluence using the diffusion approximation, by duplicating NIRFAST's results, it can be used for the endoscopic geometry. The endoscopic geometry is not suited for NIRFAST because two *different* boundary conditions are required. Unfortunately, NIRFAST cannot implement more than one type of boundary condition. Whereas, COMSOL can accommodate multiple boundary conditions. Therefore, using NIRFAST to model the endoscopic geometry will mean that one of the boundaries will be modeled incorrectly. The simulation will be executed twice within COMSOL to compare the effect of changing the outer boundary condition from the incorrect air/tissue boundary condition, like NIRFAST simulates, to the correct tissue/tissue boundary condition.

4.3.1 Endoscopic Simulation Setup

Figure 4.6 indicates the location of the different boundary conditions for the endoscopic geometry. The inner boundary represents the endoscopic probe and will be modeled as an air/tissue boundary. The outer boundary will need to modeled as a tissue/tissue boundary as though it is absorbing or 'invisible' to the simulation.

The tissue/tissue boundary condition is implemented by setting the factor $A=1$, as shown in figure 3.3, for the boundary settings on the external boundary. The background absorption coefficient is $0.002\,\text{mm}^{-1}$ and the reduced scattering coefficient is $0.5\,\text{mm}^{-1}$. The source is the Gaussian source described in section 3.4.3, located at X = $-12\,\text{mm}$ and Y = $0\,\text{mm}$. The star in figure 4.6 approximates the location of the source.

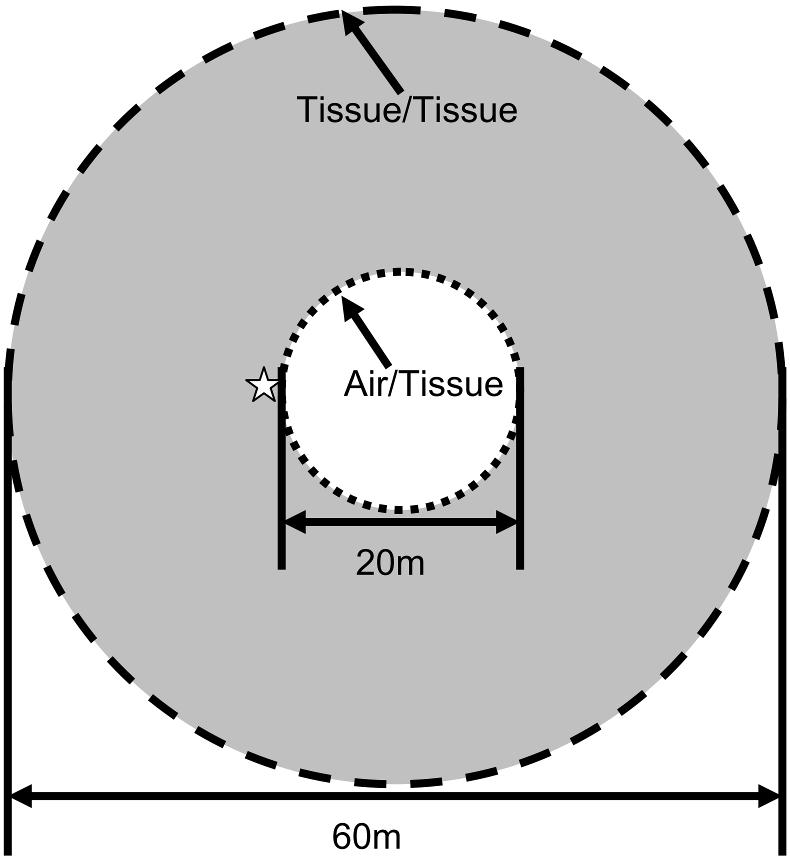


Figure 4.6: Endoscopic Geometry and Boundary Conditions

4.3.2 Homogeneous Endoscopic Simulation Results

Below, figures 4.7(a)-(d) and 4.8(a)-(d) show the results of the fluence difference between the incorrect boundary conditions and the correct boundary conditions. Figures 4.7(a) and 4.7(b) show that the fluence values for each of the two boundary conditions differ at the external boundary, and the incorrect boundary condition has a higher fluence value than the correct boundary condition. This is to be expected since the incorrect boundary condition will model a tissue/air interface at which some light will be internally refracted at the interface back into the tissue, thereby adding to the fluence value near the boundary. Furthermore, it should be noted that increasing the mesh density does not significantly change the error between the simulations, therefore the simulation results maintain a level of difference. Compare the global RMS error values for the coarse mesh at 0.010285 to the fine mesh at 0.0098547; these values are much closer than comparing the RMS error for the heterogeneous or homogeneous coarse to dense meshes.

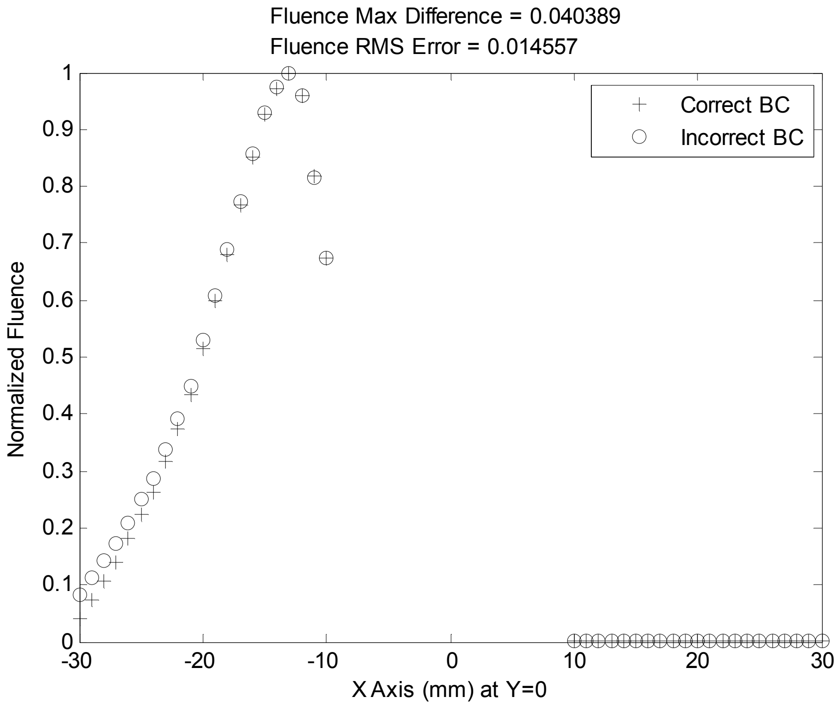


Figure 4.7(a): Linear Fluence Comparison for Coarse Mesh

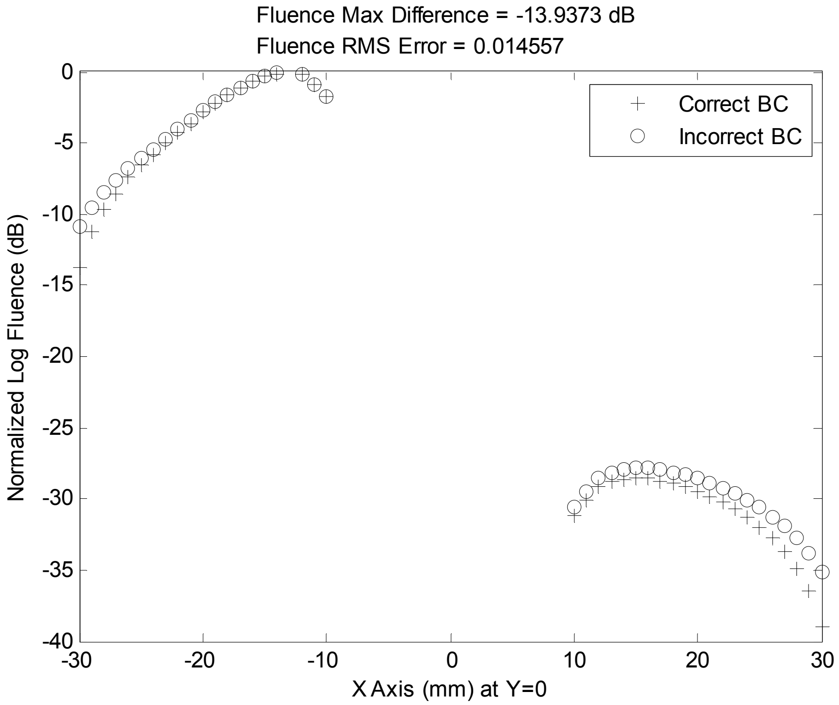


Figure 4.7(b): Log Fluence Comparison for Coarse Mesh

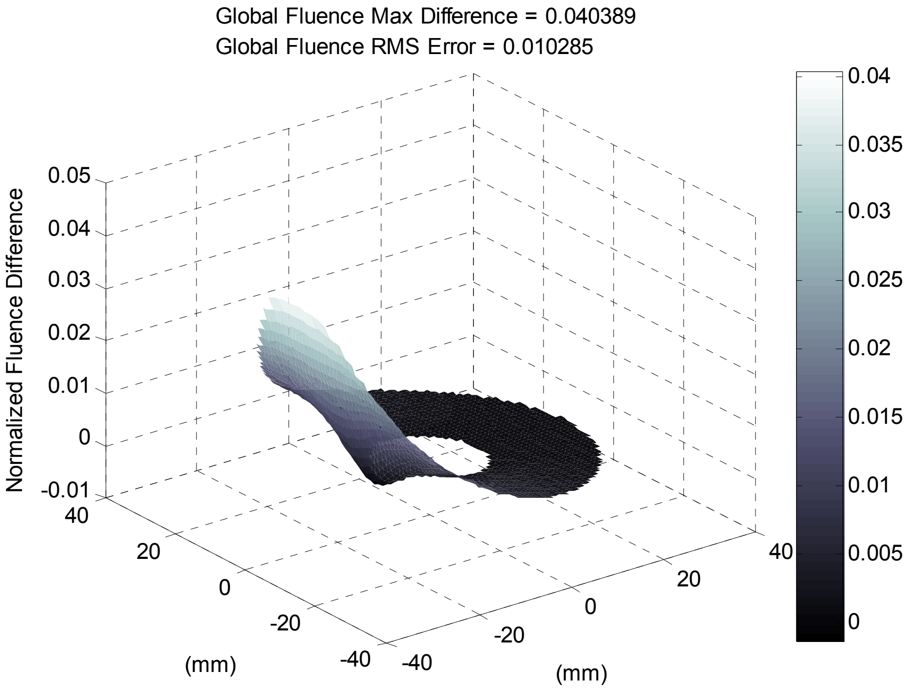


Figure 4.7(c): Global Fluence Difference Plot with Coarse Mesh

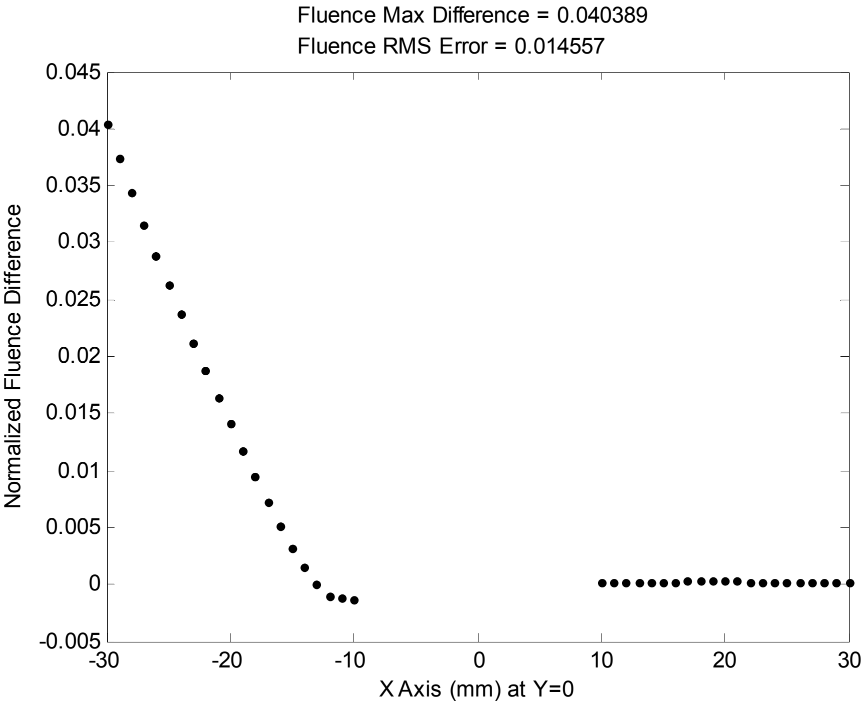


Figure 4.7(d): Fluence Difference Slice with Coarse Mesh

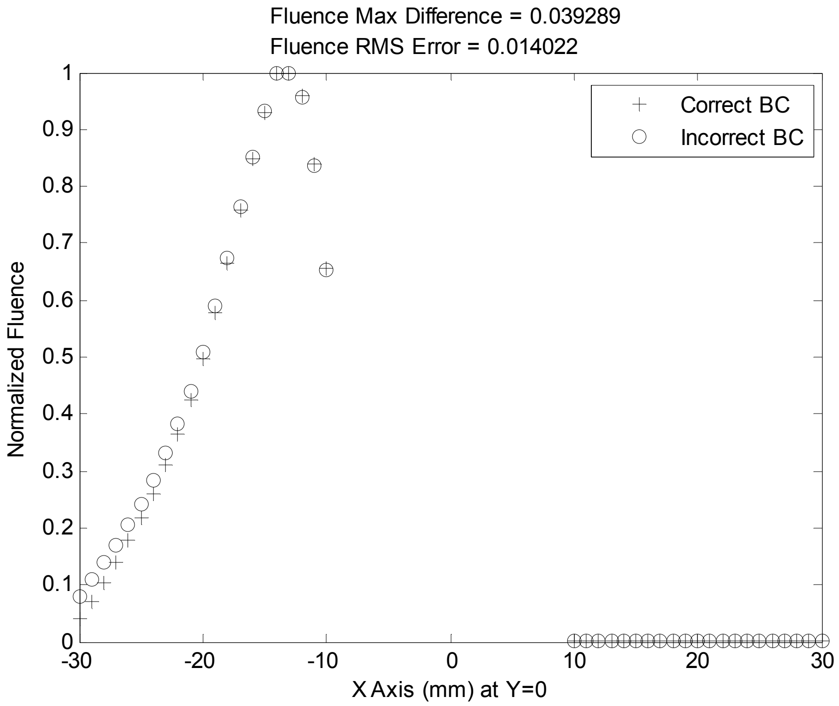


Figure 4.8(a): Linear Fluence Difference for Dense Mesh

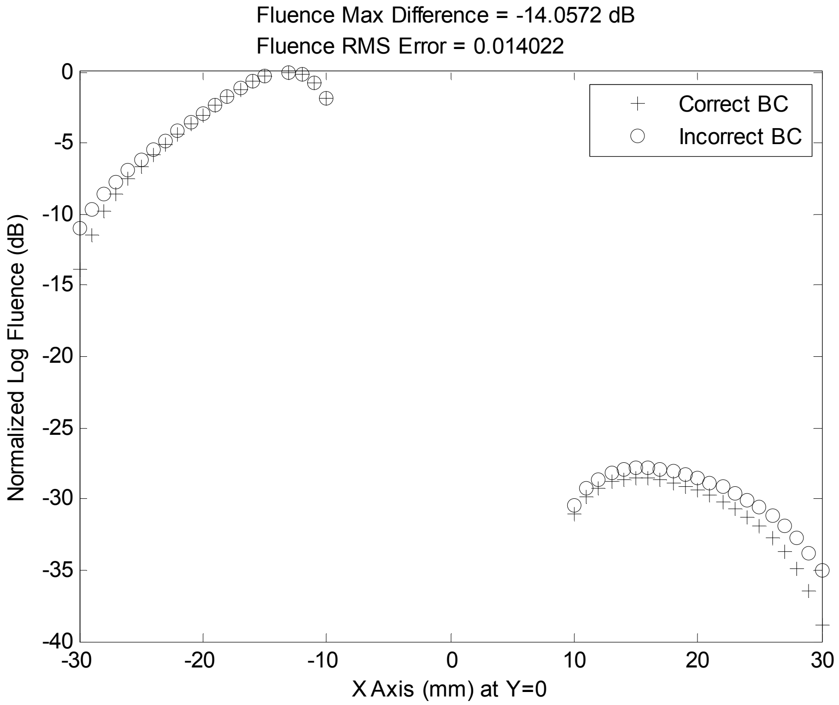


Figure 4.8(b): Log Fluence Difference for Dense Mesh

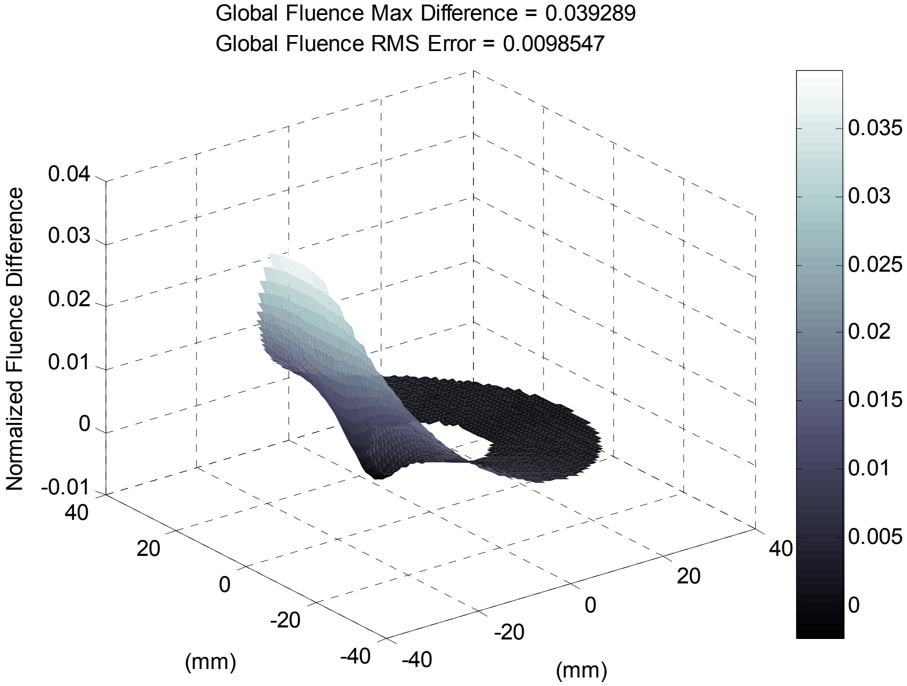


Figure 4.8(c): Global Fluence Difference Plot with Dense Mesh

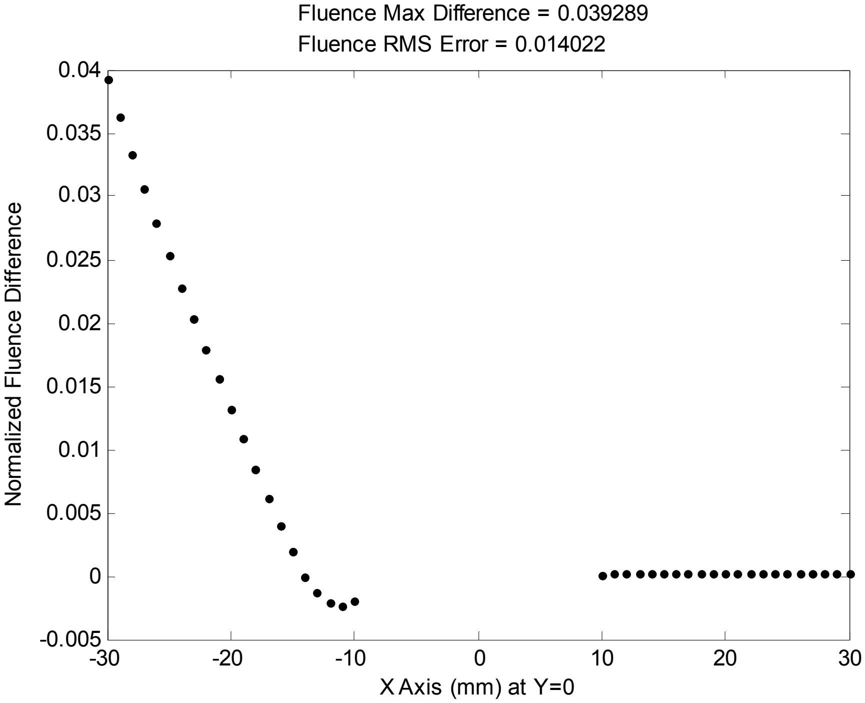


Figure 4.8(d): Fluence Difference Slice with Dense Mesh

4.3.3 Heterogeneous Endoscopic Sensitivity Study

A sensitivity study can help to evaluate the effectiveness of the endoscopic geometry to distinguish a suspected tumor from healthy tissue. A target can only be detected if its presence within the simulation has an effect upon the fluence at the detector location. It does not matter what the fluence is throughout the medium, or near the target, since the detector must be able to record the effects of the target. Since this is a non-invasive method, the detectors will remain outside of the tissue to be imaged, on the simulation boundary. The worst place for the detector is along the interior boundary opposite the source at $X = 10mm$ and $Y = 0mm$; this is where the lowest fluence level can be found. Figure 4.9 illustrates the location of the detector with a circle, opposite the source (represented by a star). Thus far, the target has been simulated as an infinite absorber, however a real target (tumor suspect) will have an absorption coefficient on the order of twice the background value [24].

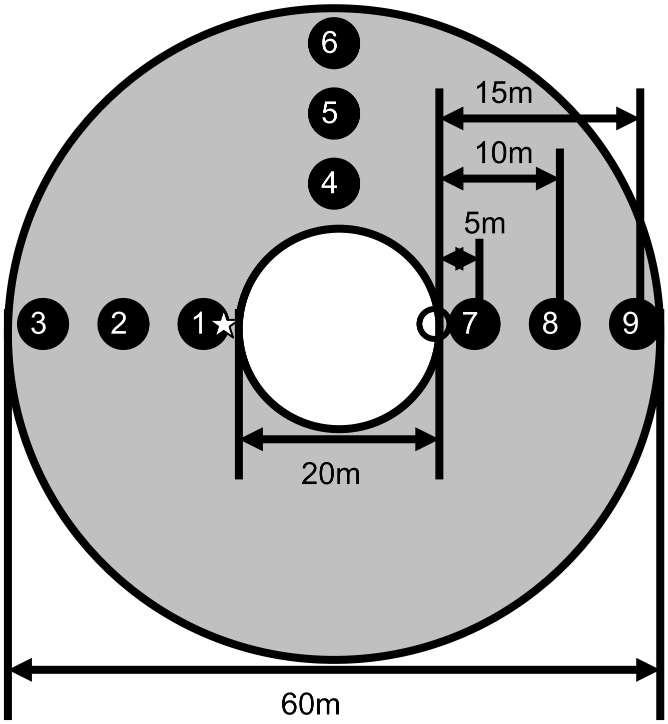


Figure 4.9: Sensitivity Simulation Geometry and Target Locations

To observe the effects of the target at various locations upon the worst-case detector location, simulations were run in COMSOL to measure the fluence at the detector location. Figure 4.9 also shows the nine locations of the target for each detector measurement. Furthermore, the simulation will be executed for two different target contrast values: 100x and 2x. Each location is numbered to correspond to the X axis for results shown in figures 4.10 and 4.11. The fluence values in figures 4.10 and 4.11 show the fluence at the detector for no target in the media, as a baseline value for comparison, as a dashed line. The X axis values in figures 4.10 and 4.11 are the location numbers of the target when the fluence is measured at the detector. The data in figures 4.10 and 4.11 is presented in three groups, to correspond with the three azimuth positions of the target.

It can be seen in figures 4.10 and 4.11 that the fluence value at the detector for any location of the target, is less than if the target was not present. It can also be noted that the closer the target is to the probe surface the more it decreases the fluence at the detector. It is worth noting in figure 4.10 the fluence is similar in value for locations 3, 6, and 9 with the target centered 15mm from the probe surface. There is also a similar fluence level between locations 2, 5, and 8. While this suggests ambiguity in the azimuth position by the detector, when multiple sources are used it is possible to determine the azimuth position. There is a general trend for the fluence at the detector to increase as the target increases its distance from the probe surface in figures 4.10 and 4.11. Although the range in values for the realistic case in figure 4.11 is much smaller than the ideal case in figure 4.10. Figure 4.11 does indicate the target with a realistic absorption contrast can change the fluence value at the worst detector location. In a practical system there would be multiple sources and detectors used to gather measurements for reconstruction, and the location of these sources can be optimized to yield the best sensitivity for the region of interest.

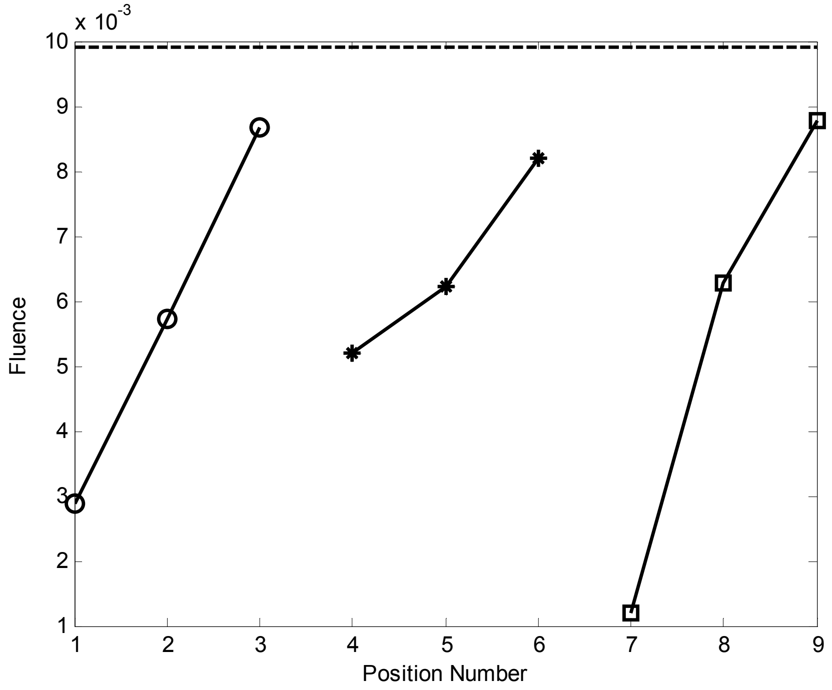


Figure 4.10: Fluence Comparison for 100x Target

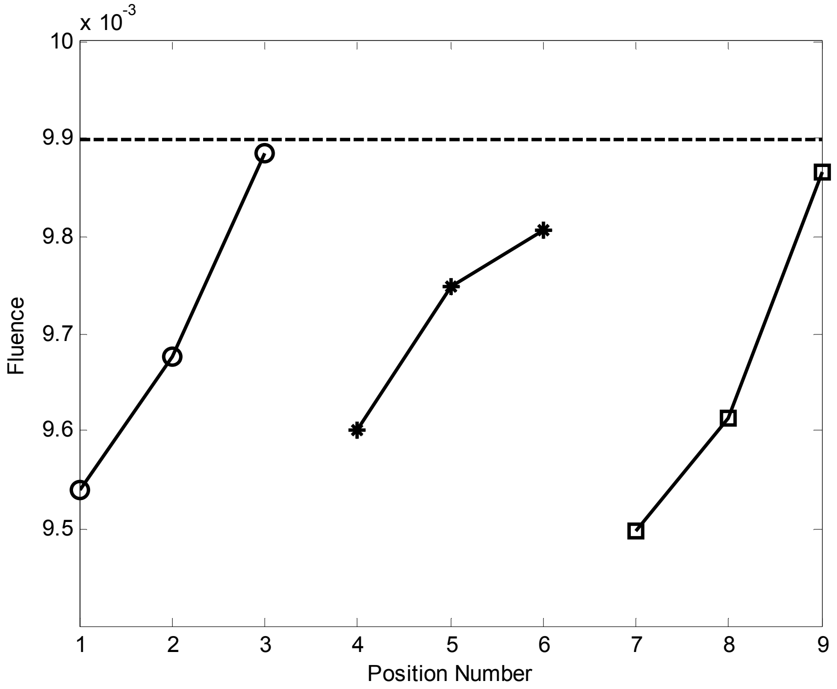


Figure 4.11: Fluence Comparison for 2x Target

5 FEDA Validation

This chapter describes the construction of a finite element diffusion approximation (FEDA) MATLAB script from the variational expression of the frequency-domain diffusion approximation. FEDA's performance is evaluated against COMSOL and NIRFAST for the external geometry, and COMSOL for the endoscopic geometry. A sensitivity study is presented to evaluate the capability of FEDA to detect a real and ideal absorber.

DOT image reconstruction is a computationally intensive, slow process. For breast cancer, the doctors can wait for post-processing to create images for later inspection; however to guide the prostate biopsy needle, doctors need a real-time video-rate image. Special processing hardware necessary to provide this real-time image construction requires the forward model to be executed multiple times per video frame image. As the forward model discussed in section 3.1 was constructed entirely within COMSOL, the computing platform must be capable of running COMSOL. Unfortunately, COMSOL is not supported for the computing platform selected for the prostate imaging system. A new simulation built from a general programming language, such as C, would be ideal for the prostate imaging system. After development and validation of the COMSOL frequency-domain diffusion approximation simulation, the author now has knowledge of how to implement a finite element diffusion approximation simulation, and possesses many tools to validate it. Since, these validation tools are in MATLAB code, the new FEDA simulation, called FEDA (Finite Element diffusion approximation), was coded in MATLAB code to take advantage of the validation tools and mathematical functions already in MATLAB. The final version of the FEDA code can be converted into C language for implementation in the prostate imaging system.

5.1 Finite Element Algorithm

The finite element method has found many uses in many different applications: from heat conduction to electromagnetic fields. The algorithm is best applied for problems that do not have a closed-form solution, irregularly shaped boundaries, and have changing material properties. The solution of the finite element method is over the entire simulation space, which is good and bad. This is good because the entire solution can be viewed over the simulation space. In the case of NIR DOT, the fluence is known for the entire medium and can be readily observed. This can also be bad because it increases the number of unknowns that must be solved and increases the memory requirements to solve the problem; some simulations are 'too large' for FEM to be practical. The finite element method can be divided into four basic steps [25]:

1. Discretize the simulation region into smaller, homogeneous elements
2. Define basis functions across each element
3. Assemble elemental basis functions for the entire simulation region and apply the boundary conditions of the problem
4. Solve the system of equations

FEDA uses mesh node coordinates defined by COMSOL with elements defined by MATLAB's `delaunay` function for 2D geometries. The mesh uses linear basis functions for triangle elements.

5.1.1 Variational Expression

The frequency-domain diffusion approximation (equation 2.2) can be considered an inhomogeneous wave equation (or second-order differential equation), $-\nabla^2\Phi + \beta\Phi = f$, by grouping together constants as

$$-\left[\kappa\left(r\right)\right]\nabla^2\Phi + \left[\mu_a + \frac{j\omega}{\nu}\right]\Phi = q_0 \qquad (5.1)$$

Arridge [4] presents the diffusion approximation variational expression in the weak form (equation 5.2-5.5) and matrix form (equation 5.6) to solve for the fluence, Φ , at each node in the simulation mesh.

$$K_{ij}^e = \kappa^e \int_\Omega \nabla u_i\left(r\right)\cdot\nabla u_j\left(r\right)d\Omega \qquad (5.2)$$

$$C_{ij}^{e} = \mu_a^e \int_{\Omega} u_i(r) \cdot u_j(r)\, d\Omega \tag{5.3}$$

$$B_{ij}^{e} = \int_{\Omega} u_i(r) \cdot u_j(r)\, d\Omega \tag{5.4}$$

$$A_{ij}^{e} = \int_{l} u_i(r) \cdot u_j(r)\, dl \tag{5.5}$$

$$\left[K(\kappa) + C(\mu_a) + \zeta A + \frac{j\omega}{\nu} B \right] \Phi = q_0 \tag{5.6}$$

Where:

- κ^e is the diffusion coefficient for the element
- μ_a^e is the absorption coefficient for the element
- $u_i(r)$ and $u_j(r)$ are the linear basis functions, $\Phi = a + bx + cy$, for the *i*th and *j*th local nodes [26]
- $\int_{\Omega} d\Omega$ is the integral of the area of the element
- $\int_{l} dl$ is the integral along the edge of the element that lies on the boundary
- ζ is a constant to implement the modified Robin boundary condition, $\zeta = \frac{1}{2\mathsf{A}}$, where A is defined in equation 2.4

5.1.2 Matrix Assembly

It should be noted that the simulation domain must be discretized so the values for κ^e and μ_a^e are constant throughout each element. Section 3.4.3 details how to implement the source term, q_0. The remainder of the terms in equation 5.6 within brackets are calculated by using Jin [27] and Reddy [26] to translate the weak form of equations 5.2-5.5 to the computational equivalent equations 5.7-5.10, defined below.

$$K_{ij}^{e} = \kappa^e \left(\frac{1}{4\mathrm{A}} \right) \begin{bmatrix} b_1^2 + c_1^2 & b_1 b_2 + c_1 c_2 & b_1 b_3 + c_1 c_3 \\ b_2 b_1 + c_2 c_1 & b_2^2 + c_2^2 & b_2 b_3 + c_2 c_3 \\ b_3 b_1 + c_3 c_1 & b_3 b_2 + c_3 c_2 & b_3^2 + c_3^2 \end{bmatrix} \tag{5.7}$$

$$C_{ij}^e = \mu_a^e\left(\frac{\mathrm{A}}{12}\right)\begin{bmatrix}2 & 1 & 1\\ 1 & 2 & 1\\ 1 & 1 & 2\end{bmatrix} \tag{5.8}$$

$$B_{ij}^e = \left(\frac{\mathrm{A}}{12}\right)\begin{bmatrix}2 & 1 & 1\\ 1 & 2 & 1\\ 1 & 1 & 2\end{bmatrix} \tag{5.9}$$

$$A_{ij}^s = \left(\frac{1}{2\mathrm{A}}\right)\left(\frac{l^s}{6}\right)\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix} \tag{5.10}$$

$$\mathrm{A} = \left(\frac{1}{2}\right)\begin{vmatrix}1 & x_1 & y_1\\ 1 & x_2 & y_2\\ 1 & x_3 & y_3\end{vmatrix} \tag{5.11}$$

$$\begin{aligned} b_1 &= y_2 - y_3 \qquad c_1 = x_3 - x_2\\ b_2 &= y_3 - y_1 \qquad c_2 = x_1 - x_3\\ b_3 &= y_1 - y_2 \qquad c_3 = x_2 - x_1 \end{aligned} \tag{5.12}$$

Equation 5.11 calculates the area of the element. Equation 5.12 defines the constants used in equation 5.7 in terms of the node coordinates. The constants in equation 5.12 are derived from the linear basis functions [26].

5.1.3 Boundary Condition

A modified Robin boundary condition is a boundary condition of the third kind. Jin [26] details the implement of the boundary condition of the third kind into the element expressions. Starting with the general form of the boundary condition

$$\nabla^2\Phi \cdot \hat{n} + \gamma\Phi = q \tag{5.13}$$

Jin [26] derives the elemental expressions

$$K_{ij}^s = \gamma^s \frac{l^s}{6}\begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix} \tag{5.14}$$

$$b_i^s = q^s \frac{l^s}{2} \tag{5.15}$$

Where:

- K_{ij}^s is the side element that assembles into $K(\kappa)$ in equation 5.6

- γ^s is defined as $\frac{1}{2\mathrm{A}}$ to meet the modified Robin boundary condition in equation 2.3
- l^s is the length of the segment of the element that along the boundary

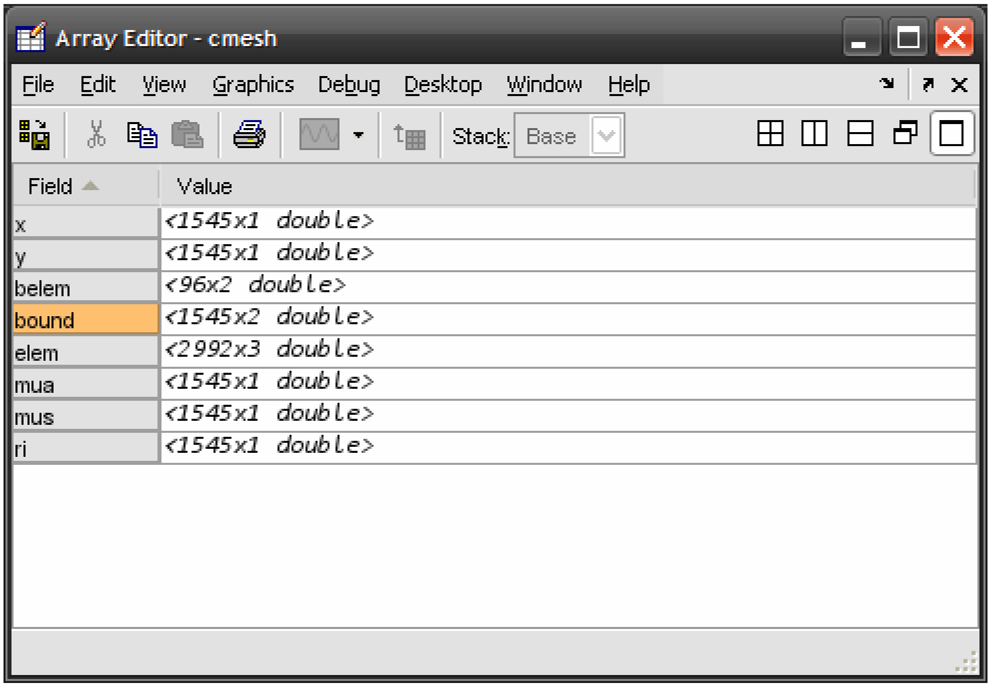
Recall for the modified Robin boundary condition, equation 2.3, that $q = 0$. Therefore, it follows that $q^s = 0$ and $b_i^s = 0$, and there does not need to be any terms added to the source matrix from the boundary. The terms calculated for the boundary nodes in equation 5.14 do need to be added to $K(\kappa)$ in the global matrix expression.

5.1.4 Solution

The solution of the assembled matrices is handled by the `ldivide` function within MATLAB. There are many different methods to solve the matrix equation 5.6, but for simplicity `ldivide.m` has been chosen in this implementation. The resulting (# of nodes)-by-1 matrix is the fluence values at each node. Like, NIRFAST, the program can be modified to simultaneously provide the results for multiple sources by adding a column to the source matrix q_0 for each source. The resulting fluence will be a matrix of columns for each source with fluence values at each node.

5.2 Implementation

To keep the FEDA comparable to COMSOL results, the mesh points are extracted directly from the exported COMSOL fem object. The elements are generated by the MATLAB function `delaunay.m`. Figure 5.1 shows all the variables that make up the struct `cmesh`, which contains all the necessary data to execute the simulation.



Field	Value
x	<1545x1 double>
y	<1545x1 double>
belem	<96x2 double>
bound	<1545x2 double>
elem	<2992x3 double>
mua	<1545x1 double>
mus	<1545x1 double>
ri	<1545x1 double>

Figure 5.1: Screenshot of `cmesh` Struct

The node coordinates values are stored within the `x` and `y` variables; there are a total of 1545 nodes for this mesh. The variable `belem` is directly obtained from the COMSOL fem object. For this 2D mesh, the `belem` lists boundary elements; each boundary element contains two nodes. This data can be used to find the boundary nodes for complicated structures. During element assembly, the boundary conditions are indicated by scanning the first column of `bound` for ones. A value of one indicates that node (row number) is on a boundary; likewise, a zero value indicates that node (row number) is not on a boundary. The second column in `bound` indicates the refractive index outside the simulation for calculating the boundary condition. `elem` contains the boundary elements, or segments for 2D geometries, from the COMSOL geometry. The nodes listed in `elem` may or may not be on the boundary; it depends on how COMSOL generated the data. The variables `mua`, `mus`, and `ri` contain the absorption coefficient, reduced scattering coefficient, and refractive index, respectively, for each node.

5.3 Validation

The validation tools described in section 3.3 can be readily applied to validate FEDA against COMSOL and NIRFAST. Since chapter 4 shows the dense mesh produces better results, only the dense mesh will be used for FEDA validation. First FEDA will be validated for the homogeneous mesh, then the heterogeneous mesh, and finally the endoscopic geometry.

5.3.1 Homogeneous Simulation Results

Using the identical simulation parameters as described in section 4.1, these results show that the FEDA code is in good agreement with COMSOL. The RMS error for FEDA and COMSOL is less than half of the error for NIRFAST and COMSOL. Although the error is low, it is curious that most of the error is located near the source term, as can be seen in figures 5.2(c) and 5.2(d). Figure 5.3(a) shows FEDA matches better with NIRFAST than COMSOL for the area to the right of the source, but the fluence difference at the boundary is much greater when compared to NIRFAST. Figure 5.3(d) shows the difference at the boundary to be 0.07833; it is over three times higher than the difference with COMSOL.

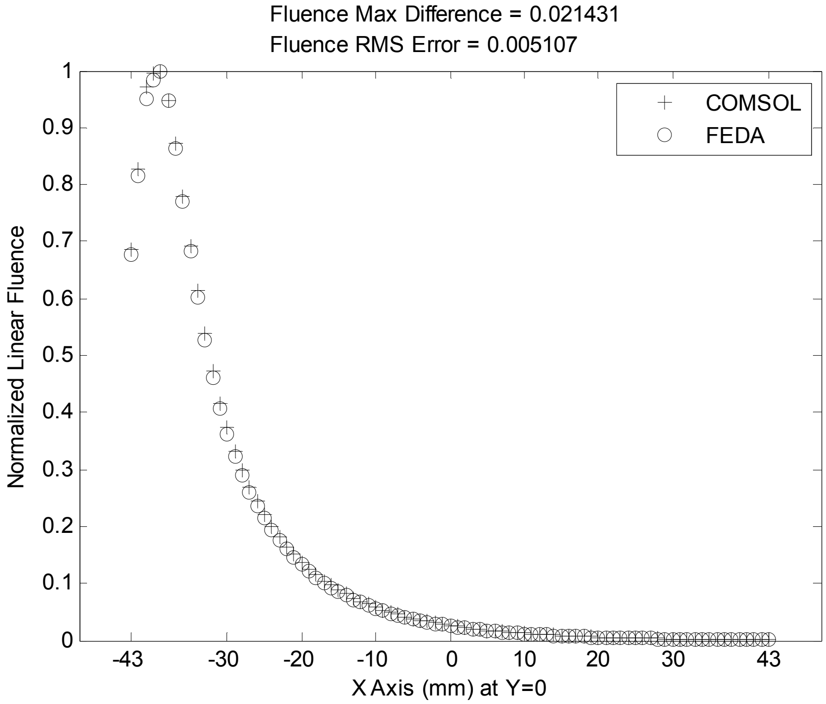


Figure 5.2(a): FEDA vs. COMSOL Linear Fluence Comparison for Homogeneous Media

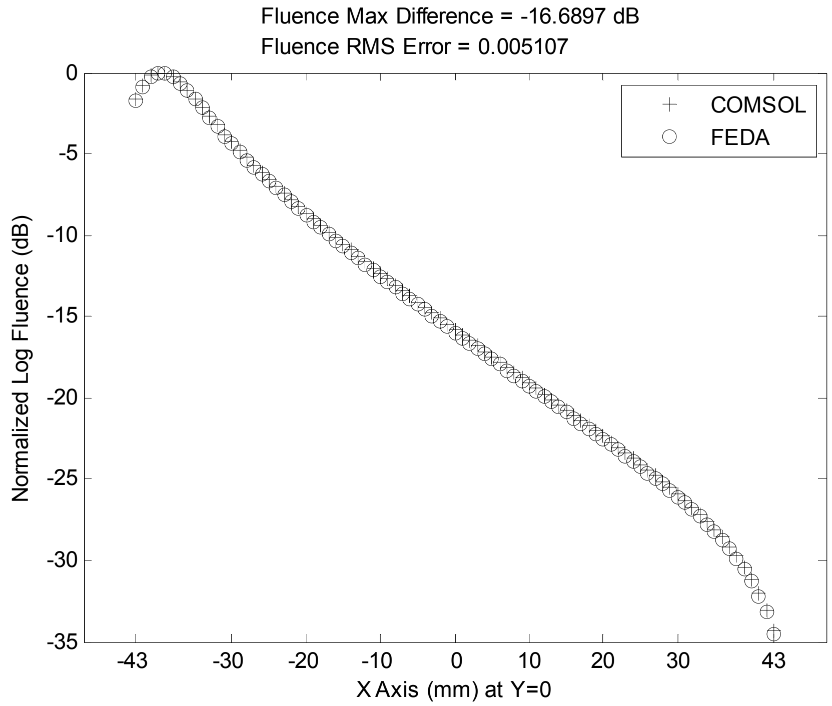


Figure 5.2(b): FEDA vs. COMSOL Log Fluence Comparison for Homogeneous Media

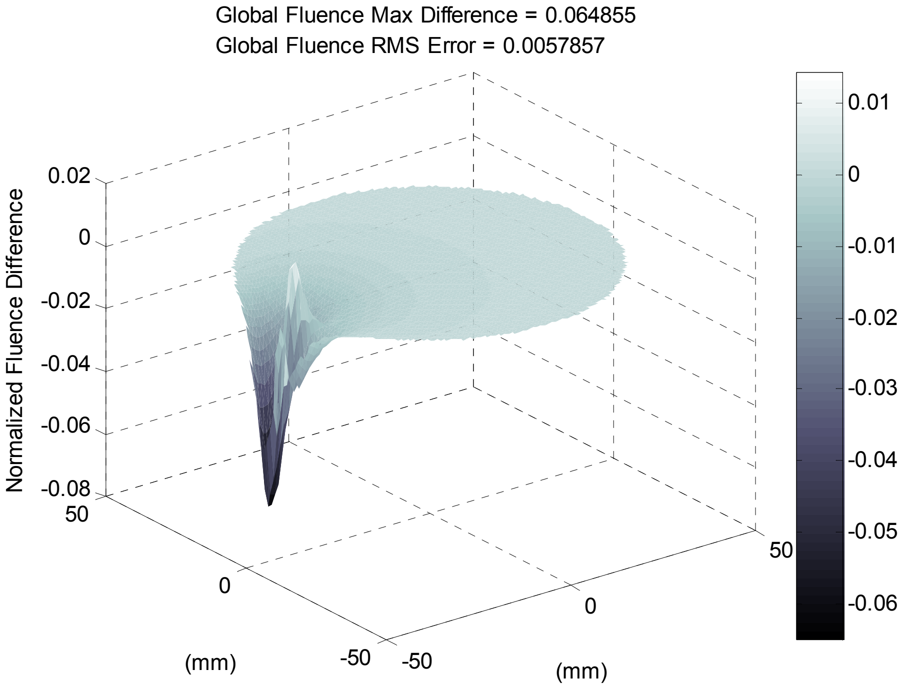


Figure 5.2(c): Global FEDA vs. COMSOL Fluence Difference Plot for Homogeneous Media

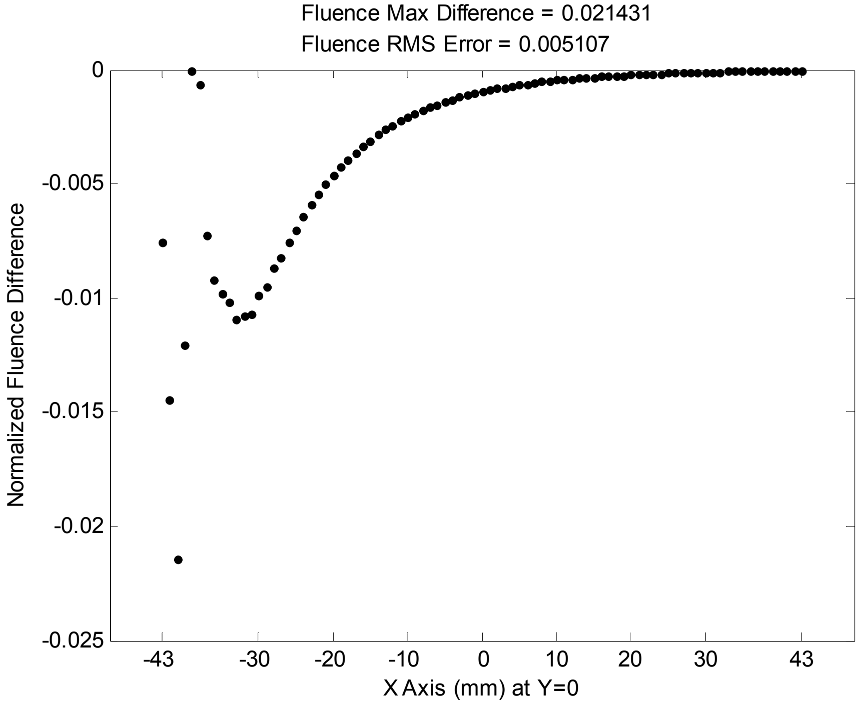


Figure 5.2(d): FEDA vs. COMSOL Fluence Difference Slice for Homogeneous Media

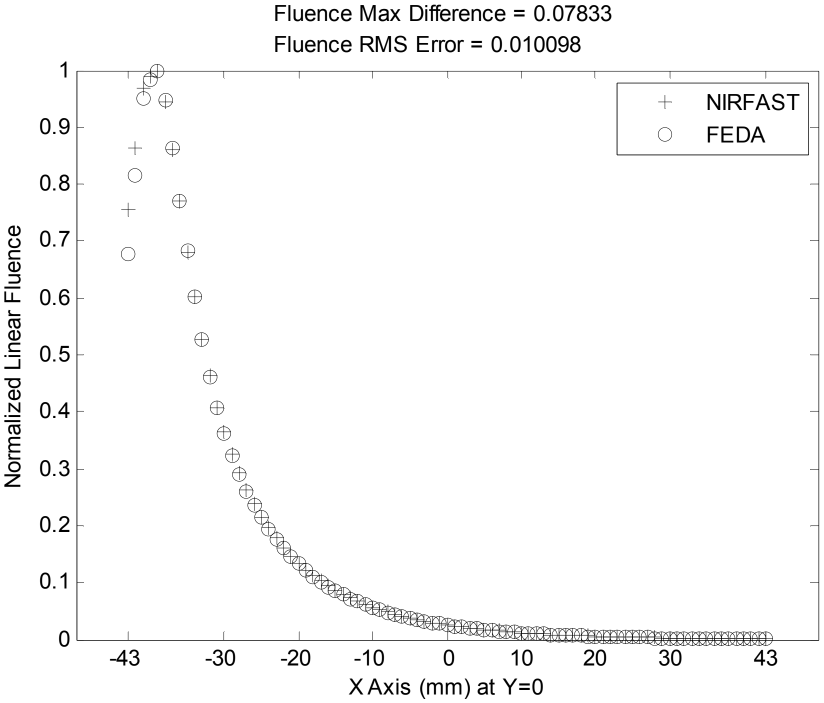


Figure 5.3(a): FEDA vs. NIRFAST Linear Fluence Comparison for Homogeneous Media

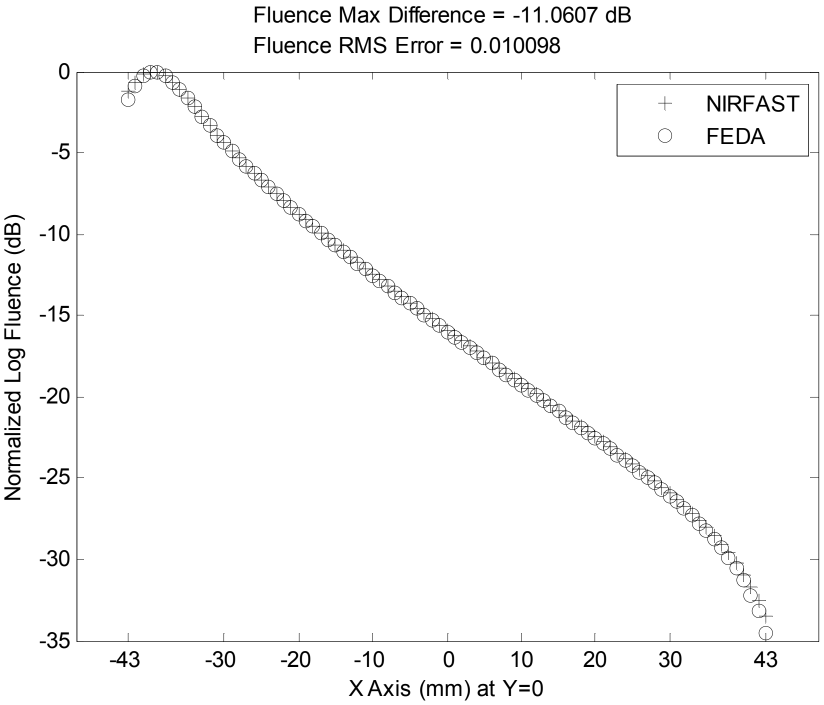


Figure 5.3(b): FEDA vs. NIRFAST Log Fluence Comparison for Homogeneous Media

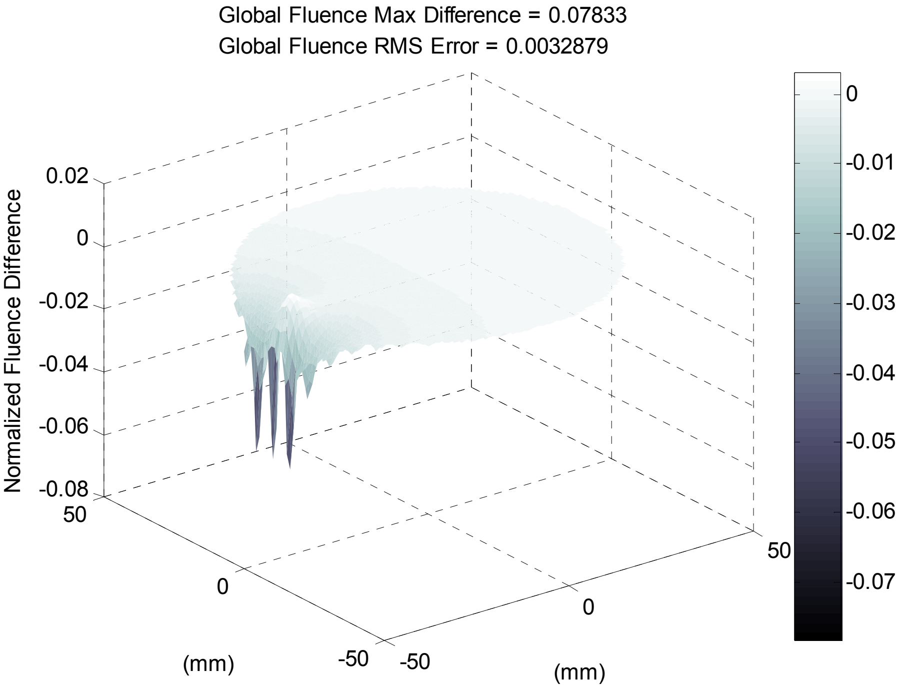


Figure 5.3(c): Global FEDA vs. NIRFAST Fluence Difference Plot for Homogeneous Media

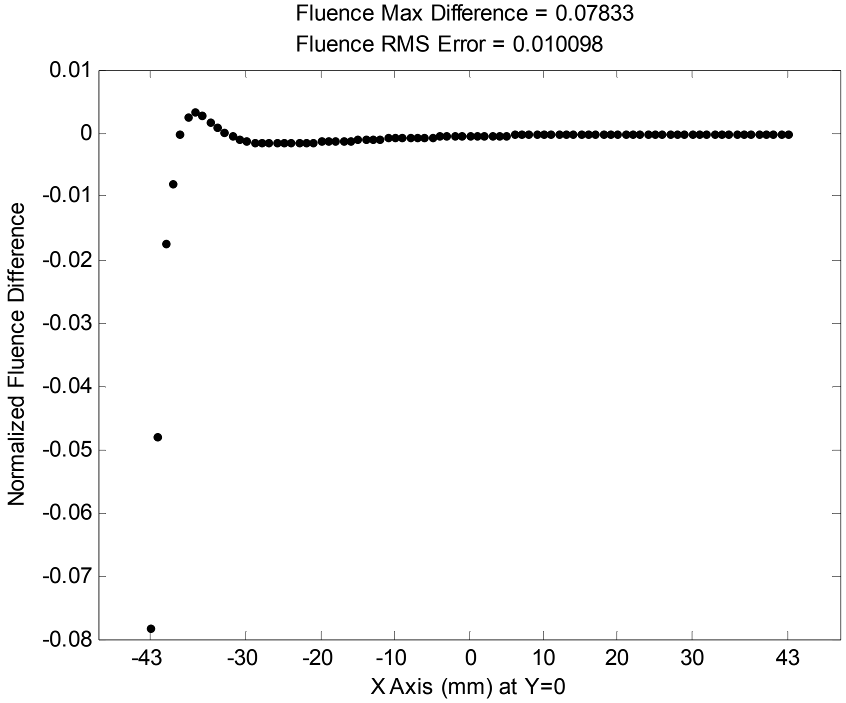


Figure 5.3(d): FEDA vs. NIRFAST Fluence Difference Plot for Homogeneous Media

5.3.2 Heterogeneous Simulation Results

The simulation geometry is identical to figure 4.3, and the simulation parameters are the same as described in section 4.2. Figures 5.4(a)-(d) show FEDA compared to COMSOL, and figures 5.5(a)-(d) show FEDA compared to NIRFAST. Like the homogeneous case, the RMS error between FEDA and COMSOL is less than the error between NIRFAST and COMSOL. However, the RMS error for FEDA vs. COMSOL and FEDA vs. NIRFAST is very close in value; compare 0.0031946 and 0.0029821. Also, like the homogeneous case, much of the error is localized near the source. Figure 5.4(b) shows COMSOL matches FEDA near the boundary, but not exactly through the target region. In contrast, figure 5.5(b) shows NIRFAST matches FEDA through the target region, but differs near the source and boundary.

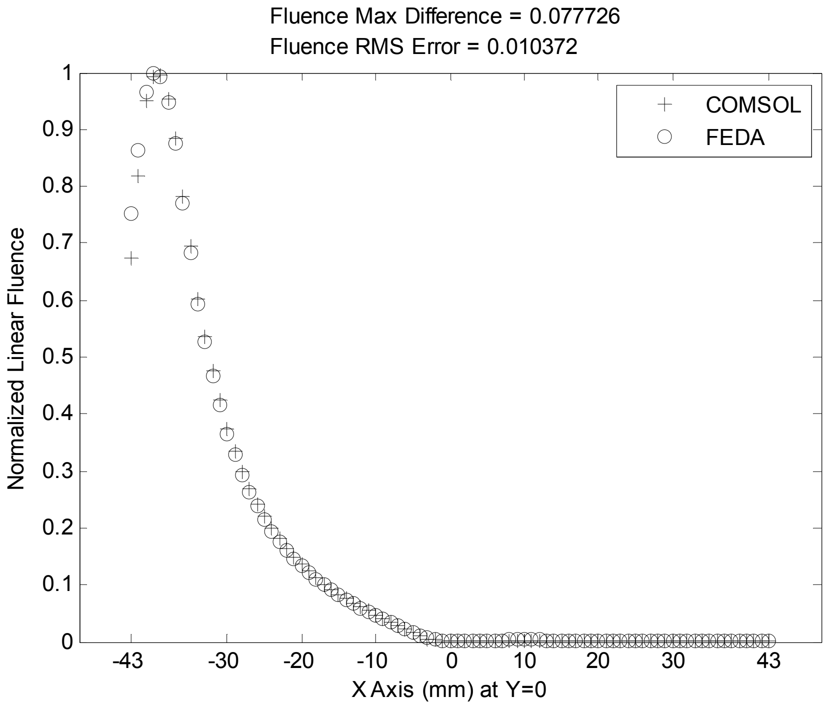


Figure 5.4(a): FEDA vs. COMSOL Linear Fluence Comparison for Heterogeneous Media

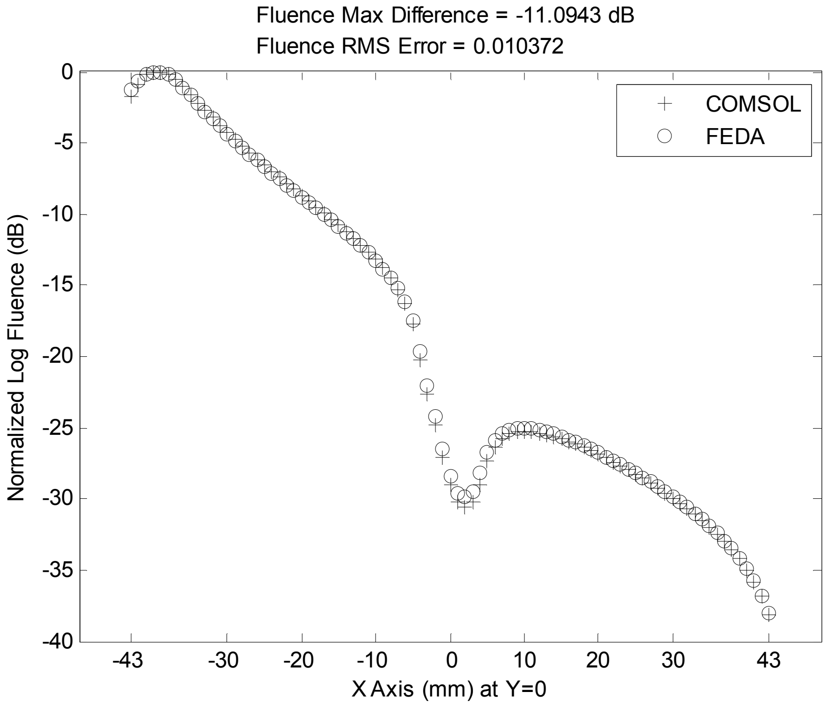


Figure 5.4(b): FEDA vs. COMSOL Log Fluence Comparison for Heterogeneous Media

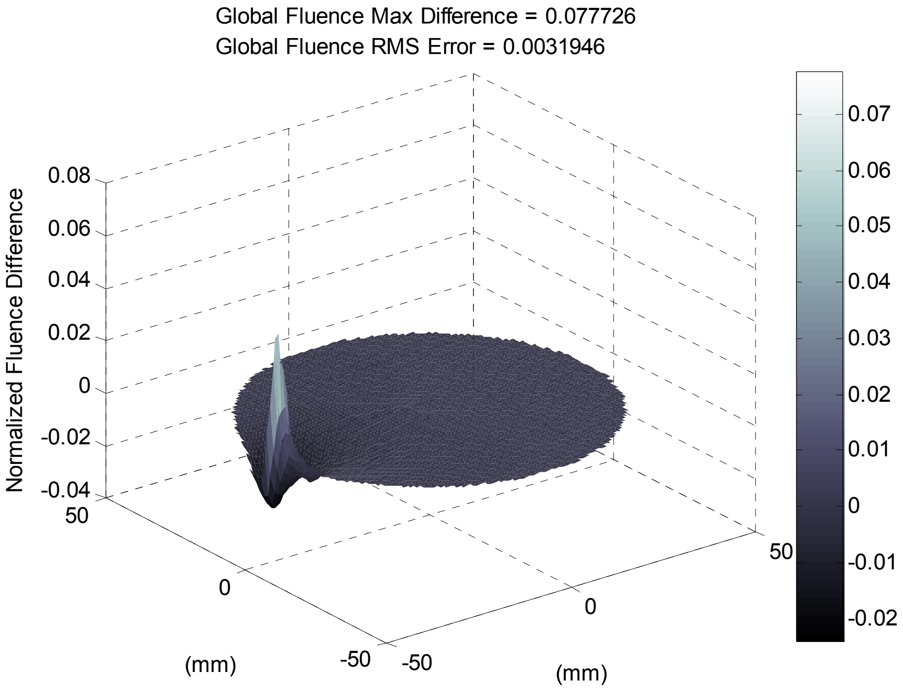


Figure 5.4(c): Global FEDA vs. COMSOL Fluence Difference Plot for Heterogeneous Media

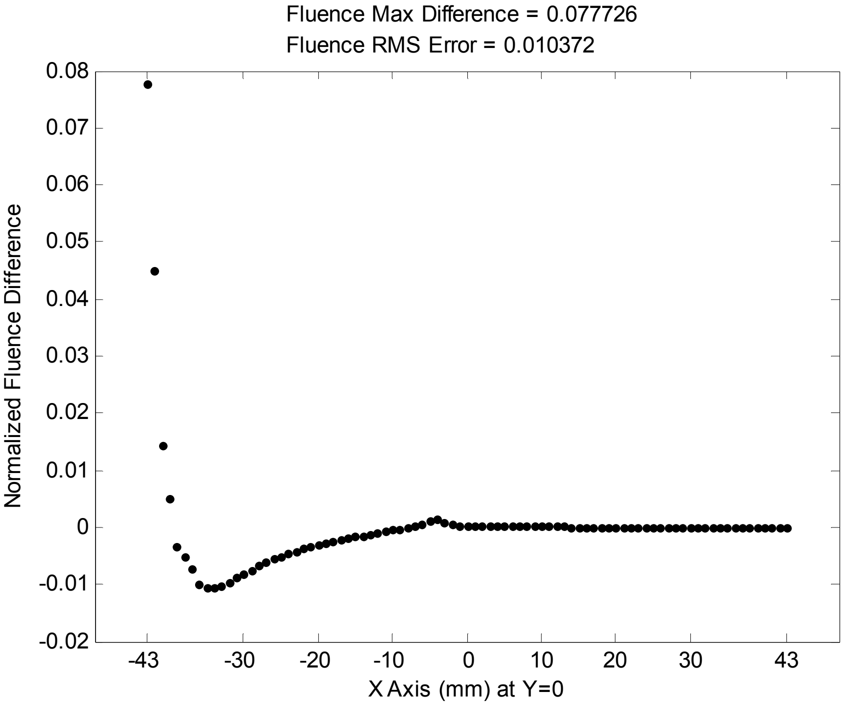


Figure 5.4(d): FEDA vs. COMSOL Fluence Difference Slice for Heterogeneous Media

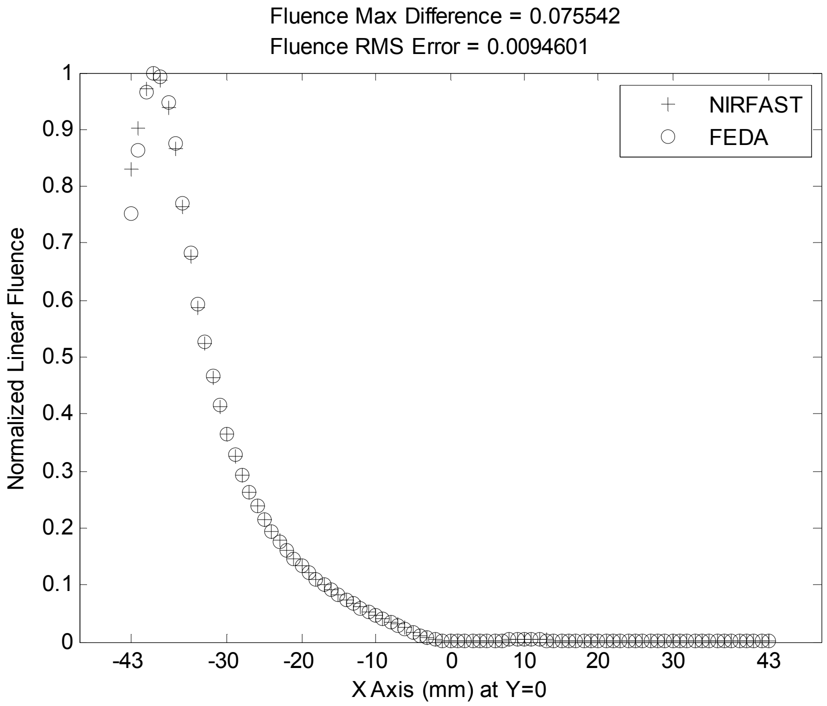


Figure 5.5(a): FEDA vs. NIRFAST Linear Fluence Comparison for Heterogeneous Media

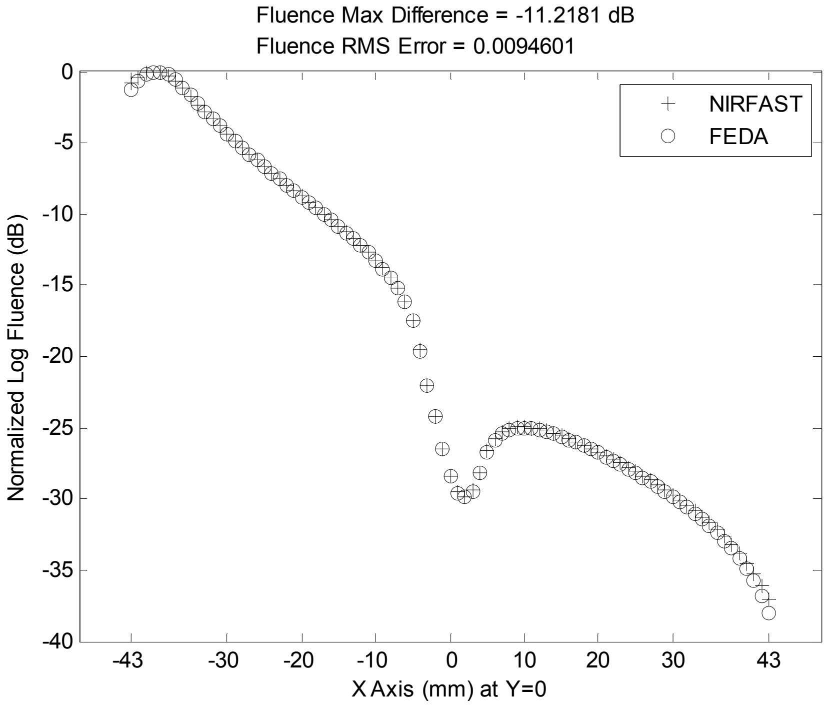


Figure 5.5(b): FEDA vs. NIRFAST Log Fluence Comparison for Heterogeneous Media

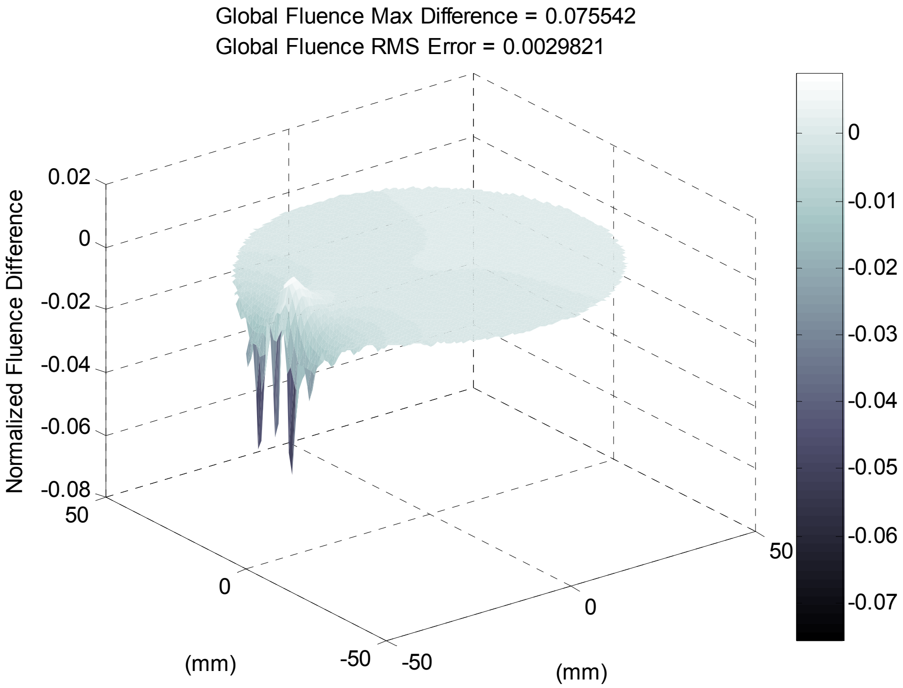


Figure 5.5(c): Global FEDA vs. NIRFAST Fluence Difference Plot for Heterogeneous Media

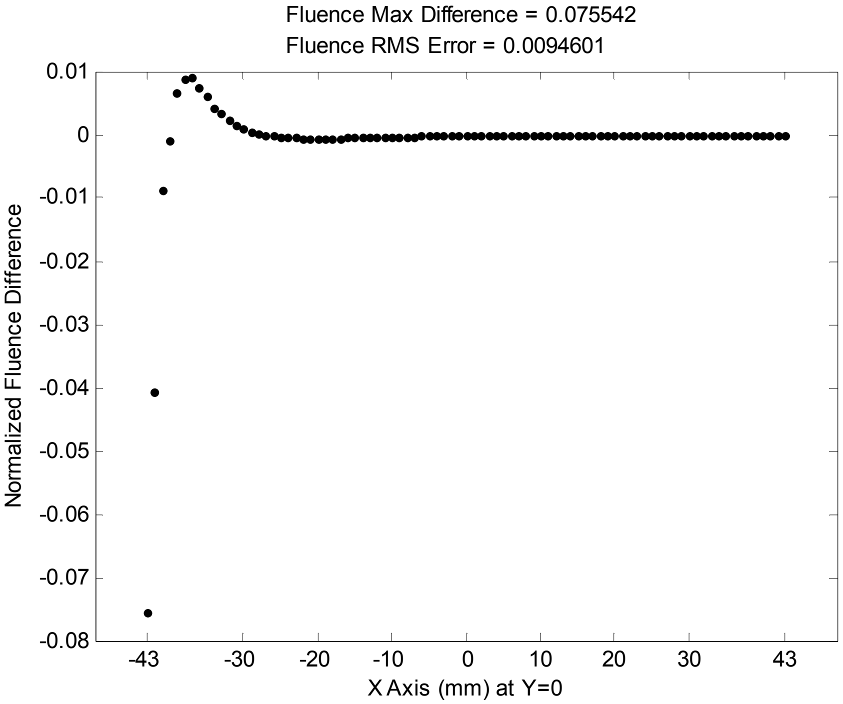


Figure 5.5(d): FEDA vs. NIRFAST Fluence Difference Slice for Heterogeneous Media

5.3.3 Endoscopic Simulation Results

The endoscopic simulation parameters are the same as the simulation parameters for the correct boundary condition simulation in section 4.3.1 with identical geometry and boundary configuration as figure 4.6. The fluence values presented below are those values at the nodes, not interpolated values as in previous comparisons. Since the meshes are identical between COMSOL and FEDA, the node locations are identical, too.

The results of this simulation match very well with COMSOL. From the log plot in figure 5.6(b), it can be seen that there is strong match near the source and at the boundaries, however FEDA overestimates the fluence in the shadow region. Figures 5.6(c) and 5.6(d) show that the much of error is extremely small, on the order of 10^{-3}.

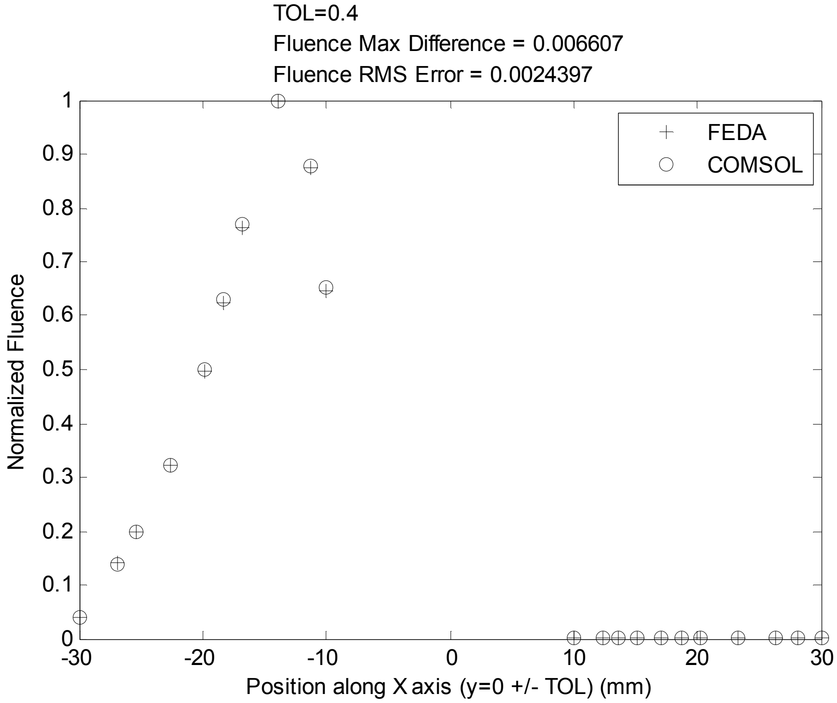


Figure 5.6(a): FEDA vs. COMSOL Linear Fluence Comparison for Endoscopic Geometry

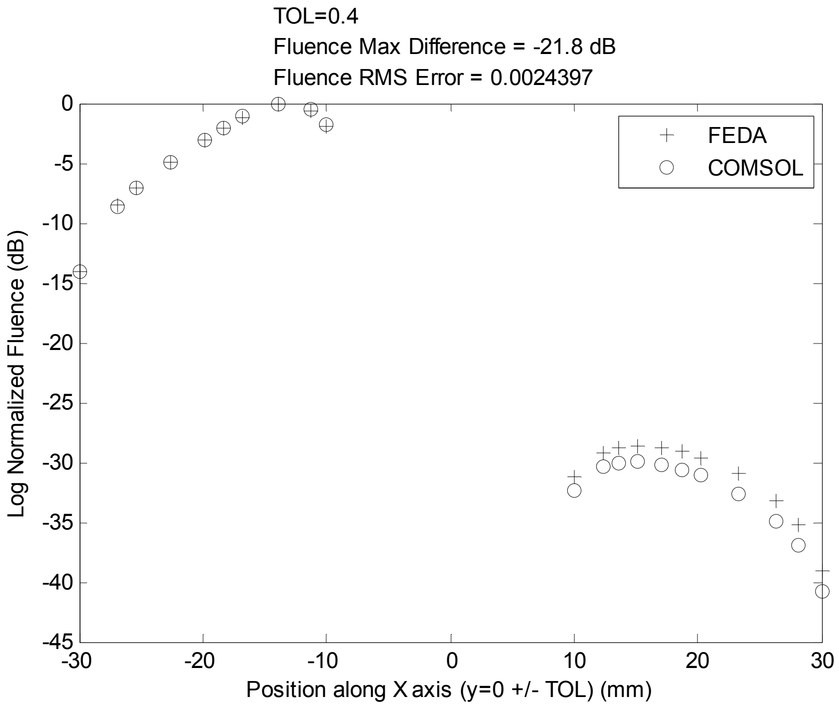


Figure 5.6(b): FEDA vs. COMSOL Log Fluence Comparison for Endoscopic Geometry

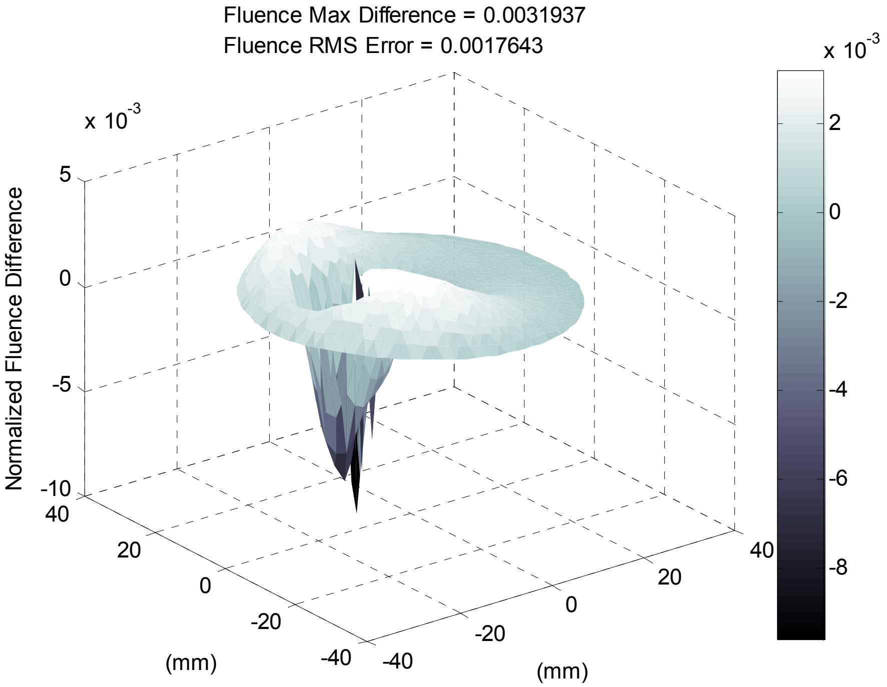


Figure 5.6(c): Global FEDA vs. COMSOL Fluence Difference Plot for Endoscopic Geometry

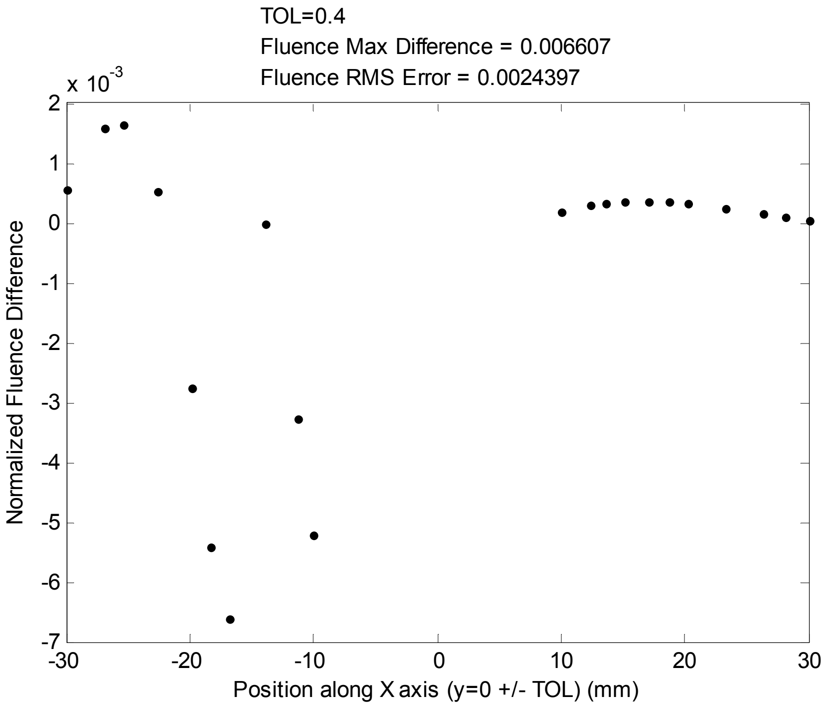


Figure 5.6(d): FEDA vs. COMSOL Fluence Difference Slice for Endoscopic Geometry

5.3.4 Heterogeneous Endoscopic Sensitivity Study

The sensitivity study presented here is identical in setup to the sensitivity study done with COMSOL in section 4.3.3. Figures 5.7 and 5.8 show that FEDA predicts the same general trend as COMSOL; as the target is moved away from the probe, the fluence value at the detector increases to the same fluence value as if no target was present. Also like the COMSOL simulation indicates, FEDA indicates that the fluence at the detector does change with the target present. Therefore, it is possible to detect the target using FEDA to model the diffusion approximation.

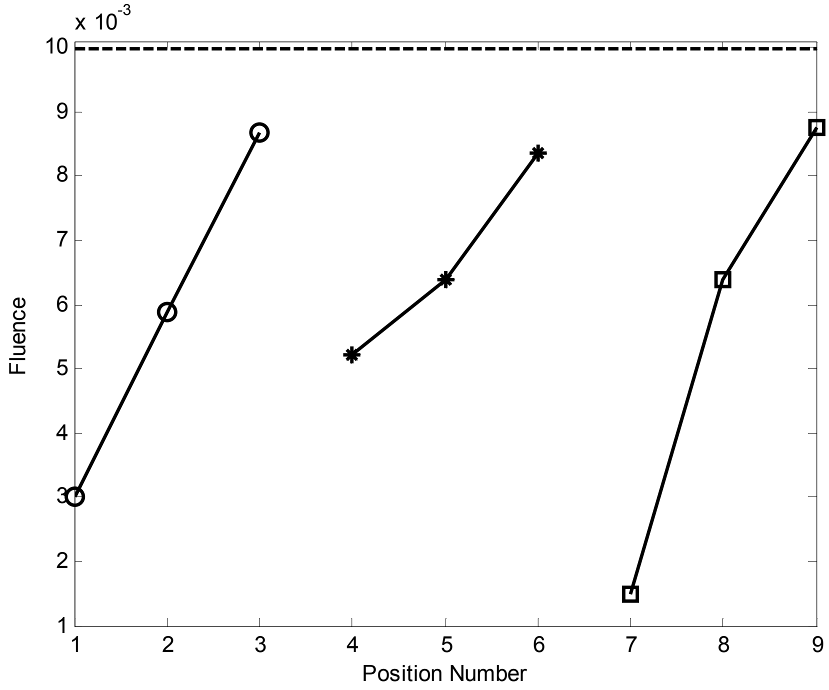


Figure 5.7: Fluence Comparison for 100x Target

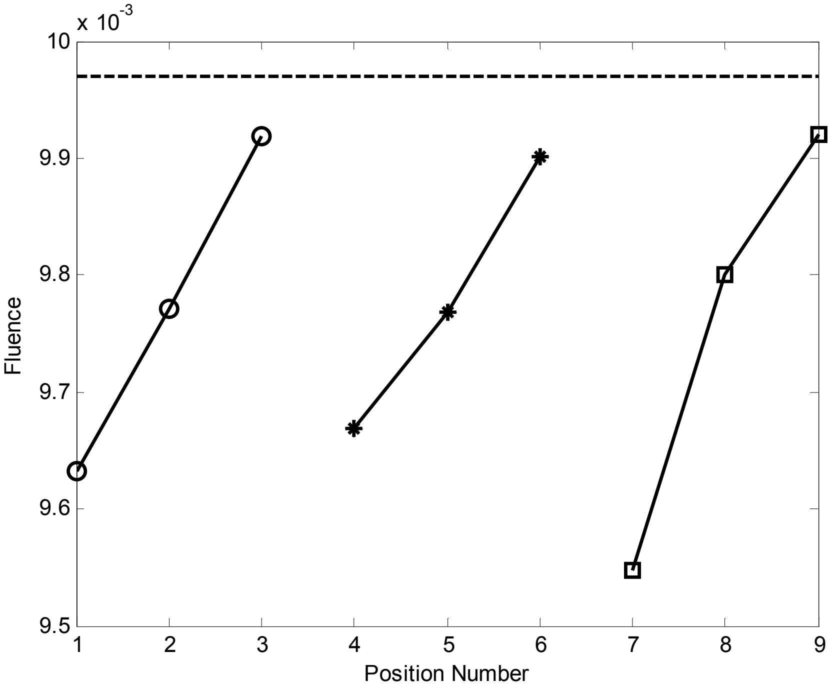


Figure 5.8: Fluence Comparison for 2x Target

6 Conclusion

To explore the problems found with applying NIRFAST for the endoscopic geometry, this thesis has detailed the construction of two programs to simulate the diffusion approximation equation with finite element methods.

This thesis shows how to match the source term for NIRFAST to the user's choice; it will be important in the future to accurately represent the light source from the endoscopic probe to achieve accurate image reconstruction in the laboratory.

The COMSOL simulation is capable of producing results similar to NIRFAST, and has an ability beyond NIRFAST to simulate multiple boundary conditions while allowing the user to modify the diffusion approximation equation, if necessary. The COMSOL simulation was used to verify that fluence does change when the external endoscopic boundary condition is changed to a tissue/tissue type. The effect of this fluence change upon the reconstructed image still remains to be seen. The sensitivity study using COMSOL revealed that a realistic target's presence within the simulation does effect the fluence measurement at a detector opposite the source.

It has been shown that FEDA yields similar results to NIRFAST and COMSOL for the external geometries. The fluence predicted by FEDA matches closely to COMSOL for the endoscopic geometry. In the future, comparison of FEDA results with laboratory measurements for endoscopic geometry will help to further assess the simulation accuracy. The sensitivity study shows that FEDA is capable of detecting a target, and future work should follow to determine the detection limits for distance, contrast, target size, and system noise. FEDA gives the user the flexibility to modify every aspect of the model, including element shape, basis functions, boundary conditions, and diffusion approximation equation. From FEDA it is possible to translate the MATLAB script into

C code and execute the algorithm on a variety of powerful computing platforms, such as an FPGA, to achieve the necessary image reconstruction speed for video-rate NIR DOT.

A preliminary sensitivity study shows that it is possible for the diffusion approximation, applied to endoscopic geometry, to detect a realistic cancer suspect (target). In the future, practical source-detector locations must be determined for the best sensitivity.

The boundary conditions have been treated as simple cases: air/tissue or tissue/tissue. In real applications, the probe boundary cannot be modeled as air/tissue. The probe is not wholly air, but a semi-reflective surface that may or may not have pockets of air near the probe. Also, the tissue near the prostate is not homogeneous, and the colon wall was not modeled. Future work must ascertain the proper implementation of the probe boundary, and the effect of boundary conditions between different organs within the body.

The next step from this work is to verify the fluence calculations predicted by FEDA can be measured in a laboratory. This work would verify the diffusion approximation applies for the endoscopic geometry, and the boundary conditions are accurate.

7 References

[1] Dehghani, H., Pogue, B.W., Poplack, S.P., and Paulsen, K.D., “Multiwavelength three-dimensional near-infrared tomography of the breast: initial simulation, phantom, and clinical results,” *Applied Optics*, Vol. 42, pg 135-145, 2003.

[2] American Cancer Society, “Overview: Prostate Cancer,” http://www.cancer.org/docroot/CRI/CRI_2_1x.asp?dt=36.

[3] DOBI Medical International, http://www.dobimedical.com/.

[4] Arridge, S.R., “Optical Tomography in medical imaging,” *Inverse Problems*, Vol. 15, pg R41-R93, 1998.

[5] Musgrove, C., Bunting, C.F., Dehghani, H., Pogue, B.W., and Piao, D., “Computational Aspects of Endoscopic (Trans-Rectal) Near-infrared Optical Tomography: Initial Investigations,” *Proceedings of SPIE*, Vol. 6434, 2007.

[6] Chance, B., Cope, M., Gratton, E., Ramanujam, N., Tromberg, B., “Phase Measurement of Light Absorption and Scatter In Human Tissue,” *Review of Scientific Instruments*, Vol. 69, No. 10, 1998.

[7] Jacques, S.L. and Wang, L., “Monte Carlo Modeling of Light Transport in Tissues,” *Optical-Thermal Response of Laser-Irradiated Tissue* edited by A.J. Welch and M.J.C. van Gemert, 1995.

[8] Wang, L. and Jacques, S.L., “Hybrid model of Monte Carlo simulation and diffusion theory for light reflectance by turbid media,” *Journal of the Optical Society of America*, Vol. 10, pg. 1746, 1993.

[9] Tarvainen, T., Vauhkonen, M., Kolehmainen, V., and Kaipio, J.P., “Hybrid radiative-transfer-diffusion model for optical tomography,” *Applied Optics*, Vol. 44, pg 876, 2005.

[10] Okada, E., Firbank, M., Schweiger, M., Arridge, S.R., Cope, M., and Delpy, D. T., “Theoretical and experimental investigation of near-infrared light propagation in a model of the adult head,” *Applied Optics*, Vol. 36, pg 21, 1997.

[11] Tarvainen, T., Vauhkonen, M., Kolehmainen, V., and Kaipio, J.P., “Finite element model for the coupled radiative transfer equation and diffusion approximation,” *International Journal for Numerical Methods in Engineering*, Vol. 65, pg. 383, 2006.

[12] Star, W.M., “Diffusion Theory of Light Transport,” *Optical-Thermal Response of Laser-Irradiated Tissue* edited by A.J. Welch and M.J.C. van Gemert, 1995.

[13] Sanchez, R. and McCormick, N.J., “A Review of Neutron Transport Approximations,” *Nuclear Science and Engineering*, Vol. 80, pg. 481-535, 1982.

[14] Klose, A.D., Netz, U., Beuthan, J., and Hielscher, A.H., “Optical Tomography Using the Time-Independent Equation of Radiative Transfer – Part 1: Forward Model,” *Journal of Quantitative Spectroscopy & Radiative Transfer*, Vol. 72, pg. 691-713, 2002.

[15] Welch, A.J., van Gemert, M.J.C., Star, W.M., and Wilson, B.C., “Definitions and Overview of Tissue Optics,” *Optical Thermal Response of Laser-Irradiated Tissue*, edited by Welch, A.J. and van Gemert, M.J.C., 1995.

[16] Jacques, S., “SC029: Tissue Optics,” course notes, Photonics West 2007, January 21, 2007.

[17] Schweiger, M., Arridge, S.R., Hiraoka, M., and Delpy, D.T., “The finite element method for the propagation of light in scattering media: Boundary and source conditions,” *Medical Physics*, Vol. 22, pg 1779-1792, 1995.

[18] Star, W.M., “Comparing the P3-Approximation with Diffusion Theory and with Monte Carlo Calculations of Light Propagation in a Slab Geometry,” in Muller, G.J. and Sliney, D.H. *Dosimetry of Laser Radiation in Medicine and Biology*, Volume IS 5, SPIE Press, 1988.

[19] Arridge, S.R., Dehghani, H., Schweiger, M., and Okada, E., “The Finite Element Model for the Propagation of Light in Scattering Media: A Direct Method for Domains with Nonscattering Regions,” *Medical Physics*, Vol. 27, pg. 252-264, 2000.

[20] Schweiger, M., Arridge, S.R., Hiraoka, M., and Delpy, D.T., “Application of the finite element method for the forward model in infrared absorption imaging,” *Proceedings of SPIE*, Vol. 1768, pg. 97-108, 1992.

[21] Dehghani, H., Pogue, B.W., Shudong, J., Brooksby, B., and Paulsen, K.D., "Three-dimensional optical tomography: resolution in small-object imaging," *Applied Optics*, Vol. 42, pg 3117, 2003.

[22] Dehghani, H., Brooksby, B., Vishwanath, K., Pogue, B.W., and Paulsen, K.D., "The effects of internal refractive index variation in near-infrared optical tomography: a finite element modeling approach," *Physics in Medicine and Biology*, Vol. 48, pg 2713-2727, 2003.

[23] COMSOL Multiphysics v3.2 User Documentation

[24] Li, C., Liengsawangwong, R., Choi, H., and Cheung, R., "Using *a priori* structural information from magnetic resonance imaging to investigate the feasibility of prostate diffuse optical tomography and spectroscopy: A simulation study," *Medical Physics*, Vol. 34(1), January 2007.

[25] Sadiku, Matthew N. O., Numerical Techniques in Electromagnetics, 2nd Ed., CRC Press, Boca Raton, 2001.

[26] Reddy, C.J., Deshpande, C.R., Cockrell, C.R., and Beck, Fred B., "Finite Element Method for Eigenvalue Problems in Electromagnetics," NASA Technical Paper 3485, December 1994.

[27] Jin, Jianming, *The Finite Element Method in Electromagnetics*, 2nd Ed., John Wiley & Sons, New York, 2002.

8 Appendix

The appendix contains a copy of all the source scripts that were used to do the work presented in this work. None of the NIRFAST script files have been reproduced here, as only one script file was modified, and the modification has been detailed in section 3.4.3.

8.1 Code for source_test.m

```
% Analysis of the source

% This plots NIRFAST's source
figure;
trimesh(mesh.elements,mesh.nodes(:,1),mesh.nodes(:,2),qvec(:,1));
trisurf(mesh.elements,mesh.nodes(:,1),mesh.nodes(:,2),nirfast.qvec(:,1)
);
shading interp;

sigma=8;
xoff=-40;
yoff=0;
comso=exp((-(mesh.nodes(:,1)-xoff).^2)/(2*sigma)).*...
    exp(-(mesh.nodes(:,2)-yoff).^2/(2*sigma));

figure;
trimesh(mesh.elements,mesh.nodes(:,1),mesh.nodes(:,2),comso);
%% This is for insertion of my own source flavor into NIRFAST
sigma=9;
xoff=-40;
yoff=1;
qvec(:,1)=exp((-(mesh.nodes(:,1)-xoff).^2)/(2*sigma)).*...
    exp(-(mesh.nodes(:,2)-yoff).^2/(2*sigma));

xoff=0;
yoff=40;
qvec(:,2)=exp((-(mesh.nodes(:,1)-xoff).^2)/(2*sigma)).*...
    exp(-(mesh.nodes(:,2)-yoff).^2/(2*sigma));

xoff=40;
yoff=0;
qvec(:,3)=exp((-(mesh.nodes(:,1)-xoff).^2)/(2*sigma)).*...
    exp(-(mesh.nodes(:,2)-yoff).^2/(2*sigma));

xoff=0;
yoff=-40;
qvec(:,4)=exp((-(mesh.nodes(:,1)-xoff).^2)/(2*sigma)).*...
    exp(-(mesh.nodes(:,2)-yoff).^2/(2*sigma));

%% Another version of my source into NIRFAST that creates a mesh of the
%% same type of the default qvec
sigma=8;
tt=qvec;
tt=full(tt);
for k=1:length(mesh.source.coord)
    xoff=mesh.source.coord(k,1);
    yoff=mesh.source.coord(k,2);
    tt(:,k)=exp((-(mesh.nodes(:,1)-xoff).^2)/(2*sigma)).*...
        exp(-(mesh.nodes(:,2)-yoff).^2/(2*sigma)).*...
        complex(cos(.15),sin(.15));
end

tt(find(tt<1e-4))=0;
```

```
tt=sparse(tt);

qvec=tt;


%% This will compare COMSOL and NIRFAST results for two different
meshes,
% normalize the data wrt to linear values, then display result in log
% compares values across x axis
figure;
tol=.1;
nir_indices=find(and(mesh.nodes(:,2)<tol,mesh.nodes(:,2)>-tol));

while length(nir_indices)<50
    tol=tol+.1;
    nir_indices=find(and(mesh.nodes(:,2)<tol,mesh.nodes(:,2)>-tol));

end
plot(mesh.nodes(nir_indices,1),...
    qvec(nir_indices,1)./max(qvec(nir_indices,1)),'b+');
hold on;
plot(mesh.nodes(nir_indices,1),...
    comso(nir_indices)./max(comso(nir_indices)),'g+');
%plot(comsol2.coord(com_indices,1),log(abs(comsol2.data(com_indices))./
...
    % max(abs(comsol2.data(com_indices)))),'r+');
hold off;
legend('NIRFAST','COMSOL');
title(['TOL=' num2str(tol)]);
xlabel('Position along X axis (y=0) (mm)');
ylabel('Fluence');
%% This plots a circle
hold on;
ph=0:pi/18:2*pi;
xc=43*cos(ph);
yc=43*sin(ph);
plot(xc,yc,'k')
hold off;
axis equal;

%% This will generate the default source graph for my thesis chapter 3
% it will also show the overwritten source when femdata is overwritten!
% To see the original NIRFAST source you must be sure that code is
active
% in femdata_stnd.m
cmesh=load_mesh('robin4_cmesh');
[nirfast,mesh]=femdata(cmesh,100);

x=-43:1:43;
[X,Y]=meshgrid(x,x);
ndata=griddata(cmesh.nodes(:,1),cmesh.nodes(:,2),...
    full(nirfast.qvec(:,1)),X,Y);
ndata=abs(ndata);
contourf(X,Y,ndata,10);
%surf(X,Y,ndata);
shading flat
colormap bone; colorbar;axis equal;
```

```
% hold on;
% ph=0:pi/18:2*pi;
% xc=43*cos(ph);
% yc=43*sin(ph);
% plot(xc,yc,'k')
% hold off;
% axis equal;

xlabel('X Axis (mm)');
ylabel('Y Axis (mm)');
%% Code to compare sources between comsol and nirfast for my thesis ch.
3

csource=exp(-.5*((mesh.nodes(:,1)+41)./1.274).^2).*...
    exp(-.5*(mesh.nodes(:,2)./1.274).^2).*exp(i*.15);
% sigma=8;
% xoff=-40;
% yoff=0;
% csource=exp((-(cmesh.nodes(:,1)-xoff).^2)/(2*sigma)).*...
    % exp(-(cmesh.nodes(:,2)-yoff).^2/(2*sigma)).*exp(i*.15);
cdata=griddata(cmesh.nodes(:,1),cmesh.nodes(:,2),csource,X,Y);
cdata=abs(cdata);
%this plots the comsol source
figure;
contourf(X,Y,cdata,10);
%surf(X,Y,cdata);
shading flat; colormap bone; colorbar;
% hold on;
% ph=0:pi/18:2*pi;
% xc=43*cos(ph);
% yc=43*sin(ph);
% plot(xc,yc,'k')
% hold off;
axis equal;
%set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
xlabel('X Axis (mm)');
ylabel('Y Axis (mm)');

%this plots the difference between the sources
d=cdata./max(max(cdata))-ndata./max(max(ndata));
figure;
contourf(X,Y,d,10);
%surf(X,Y,d);
shading flat; colormap bone; colorbar;
RMS_err=sqrt(mean(d(find(isfinite(d))).^2));
hold on;
ph=0:pi/18:2*pi;
xc=43*cos(ph);
yc=43*sin(ph);
plot(xc,yc,'k')
hold off;
axis equal;
xlabel('X Axis (mm)');
ylabel('Y Axis (mm)');
title(['RMS Error = ' num2str(RMS_err)]);
```

```
%% Checks the RMS_err calculation above

dummy=zeros(size(cdata));
ind=find(isfinite(d.^2));

figure;
plot(X(ind),Y(ind),'b.');

mean(mean(d(ind).^2))
%% Plot differences between sources as they are

[X,Y]=meshgrid(x,x);
ndata=griddata(cmesh.nodes(:,1),cmesh.nodes(:,2),...
    full(nirfast.qvec(:,1)),X,Y);
sigma=8;
xoff=-40;
yoff=0;
data=griddata(cmesh.nodes(:,1),cmesh.nodes(:,2),...
    exp((-(cmesh.nodes(:,1)-xoff).^2)/(2*sigma)).*...
    exp(-(cmesh.nodes(:,2)-yoff).^2/(2*sigma)).*exp(i*.15),X,Y);
cm_plot_terp(X,Y,data,ndata);

%% compares the location of node values (this could be for accessing
COMSOL
%% data)
```

8.2 Code for analyze_itv7.m

```
%% Run this after exporting the fem structure from COMSOL to compare
% results

%grabs the interpolated results from comsol
tic
x=-43:1:43;
len=length(x);
data=zeros(len,len);
for k=1:len
    y=x(k)*ones(1,len);
    data(k,:)=postinterp(fem,'u',[x;y]);
end
toc

% % this will create the NIRFAST simulation
% filename='robin5_cmesh';
%
% copyfile('seven_1_source.txt',[filename '_source.txt']);
% copyfile('seven_1_detector.txt',[filename '_detector.txt']);
% comsol.coord=fem.mesh.p(1:2,:)';
% comsol.belem=fem.mesh.e(1:2,:)';
% cmesh=cm_2Dmeshmakev4(0,filename,comsol);

%these are for re-executing
filename='hetero_robin2_cmesh';
cmesh=load_mesh(filename);

% this is for heterogeneous simulations
cmesh=insert_cm_blob(cmesh,.2,5,0,0);

[nirfast,mesh]=femdata(cmesh,100);
[X,Y]=meshgrid(x,x);
ndata=griddata(cmesh.nodes(:,1),cmesh.nodes(:,2),...
    full(nirfast.phi(:,1)),X,Y);

cm_plot_terp(X,Y,data,ndata,'hetero_rbc_dense3');
%save('hetero_rbc_dense2');
%% This is for the endoscopic simulations
tic
x=-30:1:30;
len=length(x);
ndata=zeros(len,len);
for k=1:len
    y=x(k)*ones(1,len);
    ndata(k,:)=postinterp(fem_correct,'u',[x;y]);
end
toc
tic
x=-30:1:30;
len=length(x);
data=zeros(len,len);
for k=1:len
    y=x(k)*ones(1,len);
    data(k,:)=postinterp(fem_incorrect,'u',[x;y]);
end
toc
[X,Y]=meshgrid(x,x);
cm_plot_terp_v3(X,Y,data,ndata,'endo_dense3');
save('e:\ir_tomo\Validation\Report_Figures\endo_dense3');
%% For endoscopic heterogeneous media
tic
x=-30:1:30;
len=length(x);
ndata=zeros(len,len);
for k=1:len
    y=x(k)*ones(1,len);
    ndata(k,:)=postinterp(fem,'u',[x;y]);
end
toc
% tic
% x=-30:1:30;
% len=length(x);
% data=zeros(len,len);
% for k=1:len
%     y=x(k)*ones(1,len);
%     data(k,:)=postinterp(fem_ref,'u',[x;y]);
% end
% toc
% gets the magnitude of the data
ndata=abs(ndata);
data=abs(data);
% need this for plots
[X,Y]=meshgrid(x,x);

figure;
surf(X,Y,ndata);
view(2); shading interp; colormap bone;
title('linear');

figure;
surf(X,Y,log(ndata));
view(2); shading interp; colormap bone;
title('log')

figure;
mid=round(length(x)/2);
plot(X(mid,:),ndata(mid,:),'k+',...
    X(mid,:),data(mid,:),'ko');
legend('With Target','Without Target');

figure;
plot(X(mid,:),log(ndata(mid,:)),'k+',...
    X(mid,:),log(data(mid,:)),'ko');
legend('With Target','Without Target');
xlabel('X Axis (mm) at Y=0');
ylabel('Fluence');

%figure; contourf(X,Y,ndata,10); shading flat; colormap bone;
%% This will compare the fluence or source results from two different
% versions of femdata calculations
```

```
cmesh=load_mesh('fine_robin1_cmesh');
%edit femdata_stnd for the full matrix
[nirfast,mesh]=femdata(cmesh,100);

%edit femdata_stnd for the sparse matrix
[nfast,mesh]=femdata(cmesh,100);

x=-43:1:43;
[X,Y]=meshgrid(x,x);
mdata=griddata(cmesh.nodes(:,1),cmesh.nodes(:,2),...
    full(nirfast.phi(:,1)),X,Y);
ndata=griddata(cmesh.nodes(:,1),cmesh.nodes(:,2),...
    full(nfast.phi(:,1)),X,Y);

cm_plot_terp(X,Y,mdata,ndata);

figure;
plot(X(44,:),log(mdata(44,:)),'b.',X(44,:),log(ndata(44,:)),'r.');
%%

savename='endo_hetero_shadow';
eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\' savename
'.jpg']);
saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename '.fig']);
```

8.3 Code for `cm_plot_terp.m`

```
function cm_plot_terp(X,Y,comsol,nirfast,savename)
% This function takes the fluence data only to plot the difference and
% calculate the error numbers.
% Created By Cameron Musgrove

comsol=real(abs(comsol));
nirfast=real(abs(nirfast));
% Calculates the difference
cdata=comsol/max(max(comsol));
ndata=nirfast/max(max(nirfast));
data=cdata-ndata;

%Calculates the max difference
max_diff=max(max(abs(data)));

%Calculates the RMS error
%this will reassign all the values in the simulation domain to data2,
for
%error calculations to remove the impact of zeros or NaN that exist
outside
%the boundaries
data2=data(find(isfinite(data)));
RMS=sqrt(mean(mean(data2.^2)));
% find(isfinite(d(:,1))) will return the indices that are not NaN

% Makes a 3D Surf Plot
figure;
%set(axes,'FontSize',14);
surf(X,Y,data);
% load('MyColorMap','c')
% colormap(c);
colormap bone;
colorbar;
shading interp;
xlabel('(mm)');
ylabel('(mm)');
zlabel('Normalized Fluence Difference');
title(['Global Fluence Max Difference = ' num2str(max_diff) '
\newlineGlobal Fluence RMS Error = ' num2str(RMS)]);

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_surf.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_surf.fig']);
    %close
end

% % Makes a Contour Plot
% figure;
% set(axes,'FontSize',14);
% contour3(X,Y,data,20);
% %plots the circle
% hold on;
% ph=0:pi/18:2*pi;
% xc=43*cos(ph);
% yc=43*sin(ph);
% plot(xc,yc,'k')
% hold off;
% view(2);
% colorbar;
% set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
% axis equal;
% xlabel('(mm)');
% ylabel('(mm)');
% title(['Fluence Max Difference = ' num2str(max_diff) '
\newlineFluence RMS Error = ' num2str(RMS)]);
%


% Makes a plot of the error along a slice
figure;
%set(axes,'FontSize',14);
en=round(length(X)/2);
plot(X(en,:),cdata(en,:)-ndata(en,:),'k.');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Fluence Difference');
set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
d2=real(cdata(en,:))-real(ndata(en,:));
max_d=max(max(abs(d2)));
RMS2=sqrt(mean(mean(d2.^2)));
title(['Fluence Max Difference = ' num2str(max_d) ' \newlineFluence RMS
Error = ' num2str(RMS2)]);

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_dif.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_dif.fig']);
    %close
end

% Makes a linear slice plot
figure;
%set(axes,'FontSize',14);
en=round(length(X)/2);
plot(X(en,:),ndata(en,:),'k+',...
    X(en,:),cdata(en,:),'ko');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Linear Fluence');
set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
d2=cdata(en,:)-ndata(en,:);
max_d=max(max(abs(d2)));
RMS2=sqrt(mean(mean(d2.^2)));
title(['Fluence Max Difference = ' num2str(max_d) ' \newlineFluence RMS
Error = ' num2str(RMS2)]);
```

```
legend('NIRFAST','COMSOL');

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_lin.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_lin.fig']);
    %close
end

% Makes a log slice plot
figure;
%set(axes,'FontSize',14);
plot(X(en,:),10*log10(ndata(en,:)),'k+',...
    X(en,:),10*log10(cdata(en,:)),'ko');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Log Fluence (dB)');
set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
title(['Fluence Max Difference = ' num2str(10*log10(max_d)) ' dB
\newlineFluence RMS Error = ' num2str(RMS2)]);
legend('NIRFAST','COMSOL');

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_log.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_log.fig']);
    %close
end
```

8.4 Code for `cm_plot_terp_v3.m`

```
function cm_plot_terp(X,Y,comsol,nirfast,savename)
% This function takes the fluence data only to plot the difference and
% calculate the error numbers.
% Created By Cameron Musgrove

% This version is for Endoscopic simulations

comsol=real(abs(comsol));
nirfast=real(abs(nirfast));
% Calculates the difference
cdata=comsol/max(max(comsol));
ndata=nirfast/max(max(nirfast));
data=cdata-ndata;

%Calculates the max difference
max_diff=max(max(abs(data)));

%Calculates the RMS error
%this will reassign all the values in the simulation domain to data2,
for
%error calculations to remove the impact of zeros or NaN that exist
outside
%the boundaries
data2=data(find(isfinite(data)));
RMS=sqrt(mean(mean(data2.^2)));
% find(isfinite(d(:,1))) will return the indices that are not NaN

% Makes a 3D Surf Plot
figure;
%set(axes,'FontSize',14);
surf(X,Y,data);
% load('MyColorMap','c')
% colormap(c);
colormap bone;
colorbar;
shading interp;
xlabel('(mm)');
ylabel('(mm)');
zlabel('Normalized Fluence Difference');
title(['Global Fluence Max Difference = ' num2str(max_diff) '
\newlineGlobal Fluence RMS Error = ' num2str(RMS)]);

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_surf.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_surf.fig']);
    %close
end

% % Makes a Contour Plot
% figure;
% set(axes,'FontSize',14);
% contour3(X,Y,data,20);
% %plots the circle
% hold on;
% ph=0:pi/18:2*pi;
% xc=43*cos(ph);
% yc=43*sin(ph);
% plot(xc,yc,'k')
% hold off;
% view(2);
% colorbar;
% set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
% axis equal;
% xlabel('(mm)');
% ylabel('(mm)');
% title(['Fluence Max Difference = ' num2str(max_diff) '
\newlineFluence RMS Error = ' num2str(RMS)]);
%


% Makes a plot of the error along a slice
figure;
%set(axes,'FontSize',14);
en=round(length(X)/2);
plot(X(en,:),cdata(en,:)-ndata(en,:),'k.');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Fluence Difference');
set(gca,'xtick',[-30 -20 -10 0 10 20 30]);
d2=cdata(en,:)-ndata(en,:);
d2=d2(find(isfinite(d2)));
max_d=max(max(abs(d2)));
RMS2=sqrt(mean(mean(d2.^2)));
title(['Fluence Max Difference = ' num2str(max_d) ' \newlineFluence RMS
Error = ' num2str(RMS2)]);

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_dif.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_dif.fig']);
    %close
end

% Makes a linear slice plot
figure;
%set(axes,'FontSize',14);
en=round(length(X)/2);
plot(X(en,:),ndata(en,:),'k+',...
    X(en,:),cdata(en,:),'ko');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Linear Fluence');
set(gca,'xtick',[-30 -20 -10 0 10 20 30]);
d2=cdata(en,:)-ndata(en,:);
d2=d2(find(isfinite(d2)));
```

```
max_d=max(max(abs(d2)));
RMS2=sqrt(mean(mean(d2.^2)));
title(['Fluence Max Difference = ' num2str(max_d) ' \newlineFluence RMS
Error = ' num2str(RMS2)]);
legend('Correct BC','Incorrect BC');

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_lin.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_lin.fig']);
    %close
end

% Makes a log slice plot
figure;
%set(axes,'FontSize',14);
plot(X(en,:),10*log10(ndata(en,:)),'k+',...
    X(en,:),10*log10(cdata(en,:)),'ko');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Log Fluence (dB)');
set(gca,'xtick',[-30 -20 -10 0 10 20 30]);
title(['Fluence Max Difference = ' num2str(10*log10(max_d)) ' dB
\newlineFluence RMS Error = ' num2str(RMS2)]);
legend('Correct BC','Incorrect BC');

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_log.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_log.fig']);
    %close
end
```

8.5 Code for `insert_cm_blob.m`

```
% this inserts a blob where cameron says

function cmesh=insert_cm_blob(mesh,mua,radius,x,y)
cmesh=mesh;
%recenter entire geometry so that the blob is the center!
mesh.nodes(:,1)=mesh.nodes(:,1)-x;
mesh.nodes(:,2)=mesh.nodes(:,2)-y;

for i=1:length(mesh.nodes)
    node_radius=sqrt(mesh.nodes(i,1)^2+mesh.nodes(i,2)^2);
    if node_radius<.99*radius %node should be within blob
        mesh.mua(i)=mua;
    end
end

cmesh.mua=mesh.mua;
cmesh.kappa = 1./(3.*(cmesh.mua+cmesh.mus));
```

8.6 Code for `cm_2Dmeshmakev4.m`

```
%% this function is based on cameron_2Dmeshmake.m to run as a function
to
% return the created mesh
% Further modified from cm_2Dmeshmakev2 to grab mesh information from
the comsol data
% file (instead of direct file access via comsol2matlab)
% this file is written for external/ring geometry
function mesh=cm_2Dmeshmakev4(flag,filename,comsol)
%clear all
% filename='First_try.mphtxt';
% nodes=1052;
% elements=1989;
% boundary=137;
%[cmesh.elements,cmesh.nodes,cmesh.bound]=comsol2matlab_2D([filename...
%   '.mphtxt'],elements,nodes,boundary);

% cmesh.elements=comsol.elem;
cmesh.elements=delaunay(comsol.coord(:,1),comsol.coord(:,2));
cmesh.nodes=comsol.coord;
cmesh.bound=comsol.belem;

cmesh.bndvtx = zeros(length(cmesh.nodes),1);
cmesh.bndvtx(1:max(max(cmesh.bound))) = 1;

cmesh.source.fixed=1;
eval(['load -ascii ' filename '_source.txt']);
eval(['cmesh.source.coord=' filename '_source;']);

cmesh.meas.fixed=1;
eval(['load -ascii ' filename '_detector.txt']);
eval(['cmesh.meas.coord=' filename '_detector;']);

%this will override the boundary stuff above, that should work when
COMSOL
%behaves
cmesh.bndvtx(:)=0;
cmesh=make_external_ring_boundary(cmesh);

n=length(cmesh.meas.coord);
link = 1 : n;
cmesh.link = repmat(link,n,1);

cmesh.mua = ones(size(cmesh.bndvtx)).*0.002;
cmesh.mus = ones(size(cmesh.bndvtx)).*0.5;
cmesh.kappa = 1./(3.*(cmesh.mua+cmesh.mus));
cmesh.ri = ones(size(cmesh.bndvtx)).*1.3;
cmesh.type='stnd';

cmesh.dimension=2;
%graphsd(cmesh)
%trisurf(cmesh.elements,cmesh.nodes(:,1),cmesh.nodes(:,2),cmesh.nodes(:
,3));
% axis equal
```

```
[ind,int_func] = tsearchn(cmesh.nodes(:,1:2),...
                cmesh.elements,...
                cmesh.meas.coord(:,1:2));
if any(isnan(ind)) == 1
  disp('Detectors outside mesh');
else
  cmesh.meas.int_func = [ind int_func];
end
if flag
    %corrects units from meters to mm units for nirfast
    cmesh.nodes=cmesh.nodes*1000;
    cmesh.source.coord=cmesh.source.coord*1000;
    cmesh.meas.coord=cmesh.meas.coord*1000;
end

save_mesh(cmesh,[filename '_cmesh']);
mesh=load_mesh([filename '_cmesh']);
```

8.7 Code for `make_external_ring_boundary.m`

```
% wipes the internal boundaries from a NIRFAST mesh
% this is applicable only to external/ring geometries

function cmesh=make_external_ring_boundary(mesh)

max_radius=max(max(mesh.nodes));

%boundary_indices=find(mesh.bndvtx==1);

%for k=1:length(boundary_indices)
for index=1:length(mesh.bndvtx)
    %index=boundary_indices(k);
    node_radius=sqrt(mesh.nodes(index,1)^2+mesh.nodes(index,2)^2);
    if node_radius>.99*max_radius %add it as a boundary
        mesh.bndvtx(index)=1;
    end
end

cmesh=mesh;
```

8.8 Code for `analyze_itv8.m`

```
%% Run this after exporting the fem structure from COMSOL to compare
% results for the homogeneous case and heterogeneous case if cm_2D_v2
is
% modified

%grabs the interpolated results from comsol
tic
x=-43:1:43;
len=length(x);
data=zeros(len,len);
for k=1:len
    y=x(k)*ones(1,len);
    data(k,:)=postinterp(fem,'u',[x;y]);
end
toc

[field,cmesh]=cm_2D_v2(fem);

[X,Y]=meshgrid(x,x);
ndata=griddata(cmesh.x,cmesh.y,field,X,Y);

cm_plot_terp_vcm(X,Y,ndata,data);

%% comparison to NIRFAST

%these are for re-executing
filename='robin2_cmesh';
mesh=load_mesh(filename);

% this is for heterogeneous simulations
%mesh=insert_cm_blob(mesh,.2,5,0,0);

[nirfast,mesh]=femdata(mesh,100);
[X,Y]=meshgrid(x,x);
data=griddata(mesh.nodes(:,1),mesh.nodes(:,2),full(nirfast.phi(:,1)),X,
Y);

cm_plot_terp(X,Y,ndata,data);
%% Runs for Debugging...

[field,cmesh]=cm_2D_v3(fem);

[X,Y]=meshgrid(x,x);
ndata=griddata(cmesh.x,cmesh.y,field,X,Y);

cm_plot_terp_vcm(X,Y,ndata,data);

%% Plots the boundary condition locations

b1=find(cmesh.bound(:,1));
b2=find(cmesh.bound(:,3));

plot(cmesh.x(b1),cmesh.y(b1),'b.'); hold on;
plot(cmesh.x(b2),cmesh.y(b2),'r.'); hold off;
legend(['Boundary w/ n=' num2str(mean(cmesh.bound(b1,2)))],...
    ['Boundary w/ n=' num2str(mean(cmesh.bound(b2,4)))]);

%% Having trouble with griddata to accurately interpolate my values;
% therefore this code is to grab the node data for comparison!
% this code is for running the endoscopic simulation

filename='endo1_dense';
comsol=read_it_in_v3([filename '.txt'],'y');

[field,cmesh]=cm_2D_v4(comsol);

%normalize data
field=abs(field)./max(abs(field));
comsol.data=abs(comsol.data)./max(abs(comsol.data));

data=field-comsol.data;

max_diff=max(data);
RMS=sqrt(mean(data.^2));

figure;
trisurf(cmesh.elem,cmesh.x,cmesh.y,data);
shading interp; colormap bone;
title(['Fluence Max Difference = ' num2str(max_diff) ...
    ' \newlineFluence RMS Error = ' num2str(RMS)]);
% plot3(comsol.coord(:,1),comsol.coord(:,2),...
% abs(comsol.data)./max(abs(comsol.data)),'b+');
% hold on;
% plot3(cmesh.x,cmesh.y,,'ro');
% hold off;

%% This will compare FEMDAC and COMSOL results for two different
meshes,
% normalize the data wrt to linear values, then display result in log
% compares values across x axis
tol=.1;
indices=find(and(cmesh.y<tol,cmesh.y>-tol));
com_indices=find(and(comsol.coord(:,2)<tol,comsol.coord(:,2)>-tol));
while length(indices)<20 && length(com_indices)<20
    tol=tol+.1;
    indices=find(and(cmesh.y<tol,cmesh.y>-tol));
    com_indices=find(and(comsol.coord(:,2)<tol,comsol.coord(:,2)>-
tol));
end
figure;
plot(cmesh.x(indices),field(indices),'b+');
hold on;
plot(comsol.coord(com_indices,1),comsol.data(com_indices),'ro');
hold off;
legend('FEMDAC','COMSOL');
title(['TOL=' num2str(tol)]);
xlabel('Position along X axis (y=0 +/- TOL) (mm)');
ylabel('Normalized Fluence');

figure;
```

```
plot(cmesh.x(indices),log(field(indices)),'b+');
hold on;
plot(comsol.coord(com_indices,1),log(comsol.data(com_indices)),'ro');
hold off;
legend('FEMDAC','COMSOL');
title(['TOL=' num2str(tol)]);
xlabel('Position along X axis (y=0 +/- TOL) (mm)');
ylabel('Normalized Fluence');

figure;
plot(cmesh.x(indices),field(indices)-comsol.data(com_indices),'k.');
title(['TOL=' num2str(tol)]);
xlabel('Position along X axis (y=0 +/- TOL) (mm)');
ylabel('Normalized Fluence Difference');
```

8.9 Code for `cm_2D_v2.m`

```
% this script initializes the data required to execute make_2d()
% this is used for development and can be modified to make use of
make_2d()
% for user-specific applications.
% this version applies for the external/ring geometry!!
% Created by Cameron Musgrove 2007!

function [field,cmesh]=cm_2D_v2(fem)

%pull node coordinates from COMSOL object fem
cmesh.x=fem.mesh.p(1,:)';
cmesh.y=fem.mesh.p(2,:)';
cmesh.belem=fem.mesh.e(1:2,:)'; %boundary elements

% Determine boundary nodes, assuming boundary nodes are the first
nodes...
cmesh.bound=zeros(length(cmesh.x),2);
% Sets the boundary nodes; will have to be modified for multiple
boundaries
cmesh.bound(1:max(max(cmesh.belem)),1)=1;
% sets the refractive index outside the boundary at each node
% a value of 1 is for air
cmesh.bound(1:max(max(cmesh.belem)),2)=1;

% creates elements
cmesh.elem=delaunay(cmesh.x,cmesh.y);

% Sets Material Parameters for Simulation Background Values
node=length(cmesh.x);
cmesh.mua=.002*ones(node,1);
cmesh.mus=.5*ones(node,1);
cmesh.ri=1.3*ones(node,1);

% corrects boundary conditions for heterogeneous simulations
cmesh=make_external_ring_boundary_vcm(cmesh);
% adds blob for heterogeneous simuatltions
%cmesh=insert_cm_blob_vcm(cmesh,.2,5,0,0);

% Modulation Frequency
f=100e6;

% Finite Element Crunch
A=make_2d_3(cmesh,f);

% Source Term
xoff=-41;
yoff=0;
sigma=8;
source=exp((-(cmesh.x-xoff).^2)/(2*sigma))...
    .*exp(-(cmesh.y-yoff).^2/(2*sigma)).*complex(cos(.15),sin(.15));

% Fluence Calculation
field=A\source;
```

8.10 Code for `make_2d.m`

```
% This script will assemble the A matrix to solve the Ax=b problem,
% or in the case of NIR DOT, to solve the A*phi=source
% This will incorporate robin boundary conditions

% Created by Cameron Musgrove 2007!

function all=make_2d(m,f)

%m=cmesh; %debug statement

% finds the total number of nodes
node=length(m.x);
% allocates the matrices
K=sparse(node,node);
C=sparse(node,node);
B=sparse(node,node);
A=sparse(node,node);

% %constants that will be removed, eventually
% n_in=1.3;
% n_out=1;
% mua=.002;
% mus=.5;

% the FEM THANG!
for k=1:length(m.elem)
    %define local nodes
    n1=m.elem(k,1);
    n2=m.elem(k,2);
    n3=m.elem(k,3);
    % forces nodes to number counter-clockwise
        %sets the zero point to calculate polar degrees from a midpoint
        xval=[m.x(n1); m.x(n2); m.x(n3)];
        xval=sortrows(xval);
        xcen=(xval(3)-xval(1))/2+xval(1);
        yval=[m.y(n1); m.y(n2); m.y(n3)];
        yval=sortrows(yval);
        ycen=(yval(3)-yval(1))/2+yval(1);
        %evaluates the degrees of each local node
        theta1=(180/pi)*atan2((m.y(n1)-ycen),(m.x(n1)-xcen));
        theta2=(180/pi)*atan2((m.y(n2)-ycen),(m.x(n2)-xcen));
        theta3=(180/pi)*atan2((m.y(n3)-ycen),(m.x(n3)-xcen));
        if(theta1<0)
            theta1=theta1+360;
        end
        if(theta2<0)
            theta2=theta2+360;
        end
        if(theta3<0)
            theta3=theta3+360;
        end
        comp=[1 theta1; 2 theta2; 3 theta3];
        %reorders the nodes for ascending degree increase
        comp=sortrows(comp,2);
        %redefine nodes in counter-clockwise order
        n1=m.elem(k,comp(1,1));
        n2=m.elem(k,comp(2,1));
        n3=m.elem(k,comp(3,1));
    % define material properties
    % the simulation domain must be properly discretized so each
element is
    % within a region

    % mua calculation
    % if the mua values at all nodes are the same, use any node value
for
    % the element
    if m.mua(n1)==m.mua(n2) && m.mua(n1)==m.mua(n3)
        mua=m.mua(n1);
    else
        muas=sort([m.mua(n1) m.mua(n2) m.mua(n3)]);
        low_mua=muas(1); % lowest value
        high_mua=muas(3); % highest value
        ref_mua=(high_mua+low_mua)/2; % mid value
        avg_mua=mean(muas); % average value for all three nodes
        if avg_mua>ref_mua % if the average is higher than the
reference
            mua=high_mua; % assume the element is in the higher region
        else
            mua=low_mua; % otherwise it must be in the lower region
        end
    end

    %mus calculation
    % same as the mua calculation
    if m.mus(n1)==m.mus(n2) && m.mus(n1)==m.mus(n3)
        mus=m.mus(n1);
    else
        muss=sort([m.mus(n1) m.mus(n2) m.mus(n3)]);
        low_mus=muss(1);
        high_mus=muss(3);
        ref_mus=(high_mus+low_mus)/2;
        avg_mus=mean(muss);
        if avg_mus>ref_mus
            mus=high_mus;
        else
            mus=low_mus;
        end
    end

    % n_in calculation
    %  same as the mua calculation
    if m.ri(n1)==m.ri(n3) && m.ri(n1)==m.ri(n3)
        n_in=m.ri(n1);
    else
        n_ins=sort([m.ri(n1) m.ri(n2) m.ri(n3)]);
        low_n_in=n_ins(1);
        high_n_in=n_ins(3);
        ref_n_in=(high_n_in+low_n_in)/2;
        avg_n_in=mean(n_ins);
        if avg_n_in>ref_n_in
            n_in=high_n_in;
        else
            n_in=low_n_in;
        end
    end
    %calculates area
    area=.5*det([1 m.x(n1) m.y(n1); 1 m.x(n2) m.y(n2); 1 m.x(n3)
m.y(n3)]);

    % calculates the element matrix for K
    b1=m.y(n2)-m.y(n3);
    b2=m.y(n3)-m.y(n1);
    b3=m.y(n1)-m.y(n2);

    g1=m.x(n3)-m.x(n2);
    g2=m.x(n1)-m.x(n3);
    g3=m.x(n2)-m.x(n1);

    diff_co=1/(3*(mua+mus));

    K_e=diff_co*(1/(4*area))*[b1^2+g1^2 b1*b2+g1*g2 b1*b3+g1*g3;...
        b2*b1+g2*g1 b2^2+g2^2 b2*b3+g2*g3;...
        b3*b1+g3*g1 b3*b2+g3*g2 b3^2+g3^2];

    % Incorporates K_e (elemental matrix) into K
    K(n1,n1)=K(n1,n1)+K_e(1,1);
    K(n1,n2)=K(n1,n2)+K_e(1,2);
    K(n1,n3)=K(n1,n3)+K_e(1,3);
    K(n2,n1)=K(n2,n1)+K_e(2,1);
    K(n2,n2)=K(n2,n2)+K_e(2,2);
    K(n2,n3)=K(n2,n3)+K_e(2,3);
    K(n3,n1)=K(n3,n1)+K_e(3,1);
    K(n3,n2)=K(n3,n2)+K_e(3,2);
    K(n3,n3)=K(n3,n3)+K_e(3,3);

    % calculates the element matrix for C
    % this will use a constant for mua value, but will have to changed
    % eventually; probably use a shape function to define a value over
the
    % element....
    C_e=mua*(area/12)*[2 1 1; 1 2 1; 1 1 2];
    % Incorporates C_e (elemental matrix) into C
    C(n1,n1)=C(n1,n1)+C_e(1,1);
    C(n1,n2)=C(n1,n2)+C_e(1,2);
    C(n1,n3)=C(n1,n3)+C_e(1,3);
    C(n2,n1)=C(n2,n1)+C_e(2,1);
    C(n2,n2)=C(n2,n2)+C_e(2,2);
    C(n2,n3)=C(n2,n3)+C_e(2,3);
    C(n3,n1)=C(n3,n1)+C_e(3,1);
    C(n3,n2)=C(n3,n2)+C_e(3,2);
    C(n3,n3)=C(n3,n3)+C_e(3,3);

    % calculates the element matrix for B
    % this will use a constant for speed of light value, but will have
to be
```

```
    % changed eventually; probably use a shape function to define a
value
    % over the element with respect to refractive index....
    B_e=(1/(3e11/n_in))*(area/12)*[2 1 1; 1 2 1; 1 1 2];
    % Incorporates C_e (elemental matrix) into C
    B(n1,n1)=B(n1,n1)+B_e(1,1);
    B(n1,n2)=B(n1,n2)+B_e(1,2);
    B(n1,n3)=B(n1,n3)+B_e(1,3);
    B(n2,n1)=B(n2,n1)+B_e(2,1);
    B(n2,n2)=B(n2,n2)+B_e(2,2);
    B(n2,n3)=B(n2,n3)+B_e(2,3);
    B(n3,n1)=B(n3,n1)+B_e(3,1);
    B(n3,n2)=B(n3,n2)+B_e(3,2);
    B(n3,n3)=B(n3,n3)+B_e(3,3);

    % Boundary condition applied to edge elements (segments)
    % tests each side of the element to see if it is on the boundary.
    % There will be problems when all three elements are on the
boundary,
    % but I will not worry about that now.

    % Since this code segment is for a specific kind of boundary, it
will
    % look to the m.bound column 1 to see if node is on boundary 1, and
    % look to m.bound column 2 for the exterior index of refraction

    flaggy=0;
    if m.bound(n1,1) && m.bound(n2,1)
        b1=n1;
        b2=n2;
        flaggy=1;
    elseif m.bound(n1,1) && m.bound(n3,1)
        b1=n1;
        b2=n3;
        flaggy=1;
    elseif m.bound(n2,1) && m.bound(n3,1)
        b1=n2;
        b2=n3;
        flaggy=1;
    end
    if flaggy % the element is on a boundary
        % set the external index of refraction value
        if m.bound(b1,2)==m.bound(b2,2)
            % external index of refraction is equal
            n_out=m.bound(b1,2);
        else
            % external index of refraction is not the same for the
boundary
            % nodes
            disp('problem with boundary nodes indices of refraction not
consistent for edge elements');
            break;
        end
        %length calculation
        l=sqrt((m.x(b1)-m.x(b2))^2+(m.y(b1)-m.y(b2))^2);
        %constants for refractive index (ri) term
        Ro=((n_out-n_in)/(n_out+n_in))^2;
        theta_c=asin(n_out/n_in);
        ri=((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2);
        % Elemental matrix calculation
        A_e=(1/(2*ri))*(l/6)*[2 1; 1 2];
        % Incorporates A_e (segment matrix) into A
        A(b1,b1)=A(b1,b1)+A_e(1,1);
        A(b1,b2)=A(b1,b2)+A_e(1,2);
        A(b2,b1)=A(b2,b1)+A_e(2,1);
        A(b2,b2)=A(b2,b2)+A_e(2,2);
    end
end

% assembles the final connectivity matrix
all=K+C+A+i*2*pi*f*B;
```

8.11 Code for cm_2d_v4.m

```
% this script initializes the data required to execute make_2d()
% this is used for development and can be modified to make use of
make_2d()
% for user-specific applications.

% v3 pulls coordinate data directly from exported fem object
% v4 uses data from a comsol .txt file
% developed for endoscopic

% Created by Cameron Musgrove 2007!

function [field,cmesh]=cm_2D_v4(fem)

%pull node coordinates from COMSOL object fem
% cmesh.x=fem.mesh.p(1,:)';
% cmesh.y=fem.mesh.p(2,:)';
% cmesh.belem=fem.mesh.e(1:2,:)'; %boundary elements
cmesh.x=fem.coord(:,1);
cmesh.y=fem.coord(:,2);
cmesh.belem=fem.belem; %boundary elements

% creates elements
cmesh.elem=delaunay(cmesh.x,cmesh.y);
cmesh=element_delete_vcm(cmesh);
% Determine boundary nodes, assuming boundary nodes are the first
nodes...
cmesh=divide_bc(cmesh,1.3,1);
% verifies the boundary split was successful
% figure;
plot(cmesh.x(find(cmesh.bound(:,1))),cmesh.y(find(cmesh.bound(:,1))),'r
.',cmesh.x(find(cmesh.bound(:,3))),cmesh.y(find(cmesh.bound(:,3))),'c.'
);



% Sets Material Parameters for Simulation Background Values
node=length(cmesh.x);
cmesh.mua=.002*ones(node,1);
cmesh.mus=.5*ones(node,1);
cmesh.ri=1.3*ones(node,1);

% corrects boundary conditions for heterogeneous simulations
% cmesh=make_external_ring_boundary_vcm(cmesh);
% adds blob for heterogeneous simuatltions
% cmesh=insert_cm_blob_vcm(cmesh,.2,5,0,0);

% Modulation Frequency
f=100e6;

% Finite Element Crunch
A=make_2d_2(cmesh,f);

% Source Term
```

```
xoff=-12;
yoff=0;
sigma=8;
source=exp((-(cmesh.x-xoff).^2)/(2*sigma)).*...
    exp((-(cmesh.y-yoff).^2)/(2*sigma)).*complex(cos(.15),sin(.15));

% Fluence Calculation
%field=A\source;
field=A; %added for new source implementation
```

8.12 Code for divide_bc.m

```
function cmesh=divide_bc(cmesh,ri_ext,ri_int)

% cmesh is the mesh
% ri_ext is the external refractive index
% ri_int is the internal refractive index
% t is the threshold value that separates a interior from exterior
circular
% boundary

% this script will find the boundary nodes, and then decide if each
% boundary node is an exterior or interior boundary, then assign the
% appropriate index of refraction for that boundary.  This script will
also
% delete any boundaries within the geometry, this is useful for
% heterogeneous simulations

% this script assumes that the boundary nodes are the first nodes!!!

% obtains list of boundary nodes
nodes=length(cmesh.x);

% allocates the boundary matrices
cmesh.bound=zeros(nodes,4);

% determine the maximum radius
max_rad=max(cmesh.x);

% find minimum radius
min_rad=min(sqrt(cmesh.x.^2+cmesh.y.^2));
count=0;
for i=1:nodes % go through all the nodes
    % convert cartesian coordinates to polar for ease
    rad=sqrt(cmesh.x(i).^2+cmesh.y(i).^2);
    % if rad is less than the (minimum radius + tol value), it is an
internal boundary
    if rad<1.01*min_rad
        % interior boundary, use columns 1 and 2
        cmesh.bound(i,1)=1;
        cmesh.bound(i,2)=ri_int;
        count=count+1;
    elseif rad>.99*max_rad
        % exterior boundary, use columns 3 and 4
        cmesh.bound(i,3)=1;
        cmesh.bound(i,4)=ri_ext;
        count=count+1;
    end
end
```

8.13 Code for make_2d_2.m

```
% This script will assemble the A matrix to solve the Ax=b problem,
% or in the case of NIR DOT, to solve the A*phi=source
% This will incorporate robin boundary conditions
% Created by Cameron Musgrove 2007!

function field=make_2d_2(m,f)

%m=cmesh; %debug statement

% finds the total number of nodes
node=length(m.x);
% allocates the matrices
K=sparse(node,node);
C=sparse(node,node);
B=sparse(node,node);
A=sparse(node,node);
source=sparse(node,1);

% %constants that will be removed, eventually
% n_in=1.3;
% n_out=1;
% mua=.002;
% mus=.5;
xoff=-12;
yoff=0;
sigma=8;

% the FEM THANG!
for k=1:length(m.elem)
    %define local nodes
    n1=m.elem(k,1);
    n2=m.elem(k,2);
    n3=m.elem(k,3);
    % forces nodes to number counter-clockwise
        %sets the zero point to calculate polar degrees from a midpoint
        xval=[m.x(n1); m.x(n2); m.x(n3)];
        xval=sortrows(xval);
        xcen=(xval(3)-xval(1))/2+xval(1);
        yval=[m.y(n1); m.y(n2); m.y(n3)];
        yval=sortrows(yval);
        ycen=(yval(3)-yval(1))/2+yval(1);
        %evaluates the degrees of each local node
        theta1=(180/pi)*atan2((m.y(n1)-ycen),(m.x(n1)-xcen));
        theta2=(180/pi)*atan2((m.y(n2)-ycen),(m.x(n2)-xcen));
        theta3=(180/pi)*atan2((m.y(n3)-ycen),(m.x(n3)-xcen));
        if(theta1<0)
            theta1=theta1+360;
        end
        if(theta2<0)
            theta2=theta2+360;
        end
        if(theta3<0)
            theta3=theta3+360;
        end
        comp=[1 theta1; 2 theta2; 3 theta3];
        %reorders the nodes for ascending degree increase
        comp=sortrows(comp,2);
        %redefine nodes in counter-clockwise order
        n1=m.elem(k,comp(1,1));
        n2=m.elem(k,comp(2,1));
        n3=m.elem(k,comp(3,1));
    % define material properties
    % the simulation domain must be properly discretized so each
element is
    % within a region

    % mua calculation
    % if the mua values at all nodes are the same, use any node value
for
    % the element
    if m.mua(n1)==m.mua(n2) && m.mua(n1)==m.mua(n3)
        mua=m.mua(n1);
    else
        muas=sort([m.mua(n1) m.mua(n2) m.mua(n3)]);
        low_mua=muas(1); % lowest value
        high_mua=muas(3); % highest value
        ref_mua=(high_mua+low_mua)/2; % mid value
        avg_mua=mean(muas); % average value for all three nodes
        if avg_mua>ref_mua % if the average is higher than the
reference
            mua=high_mua; % assume the element is in the higher region
        else
            mua=low_mua; % otherwise it must be in the lower region
        end
    end

    %mus calculation
    % same as the mua calculation
    if m.mus(n1)==m.mus(n2) && m.mus(n1)==m.mus(n3)
        mus=m.mus(n1);
    else
        muss=sort([m.mus(n1) m.mus(n2) m.mus(n3)]);
        low_mus=muss(1);
        high_mus=muss(3);
        ref_mus=(high_mus+low_mus)/2;
        avg_mus=mean(muss);
        if avg_mus>ref_mus
            mus=high_mus;
        else
            mus=low_mus;
        end
    end

    % n_in calculation
    %  same as the mua calculation
    if m.ri(n1)==m.ri(n3) && m.ri(n1)==m.ri(n3)
        n_in=m.ri(n1);
    else
        n_ins=sort([m.ri(n1) m.ri(n2) m.ri(n3)]);
        low_n_in=n_ins(1);
        high_n_in=n_ins(3);
        ref_n_in=(high_n_in+low_n_in)/2;
        avg_n_in=mean(n_ins);
        if avg_n_in>ref_n_in
            n_in=high_n_in;
        else
            n_in=low_n_in;
        end
    end
    %calculates area
    area=.5*det([1 m.x(n1) m.y(n1); 1 m.x(n2) m.y(n2); 1 m.x(n3)
m.y(n3)]);

    % calculates the element matrix for K
    b1=m.y(n2)-m.y(n3);
    b2=m.y(n3)-m.y(n1);
    b3=m.y(n1)-m.y(n2);

    g1=m.x(n3)-m.x(n2);
    g2=m.x(n1)-m.x(n3);
    g3=m.x(n2)-m.x(n1);

    diff_co=1/(3*(mua+mus));

    K_e=diff_co*(1/(4*area))*[b1^2+g1^2 b1*b2+g1*g2 b1*b3+g1*g3;...
        b2*b1+g2*g1 b2^2+g2^2 b2*b3+g2*g3;...
        b3*b1+g3*g1 b3*b2+g3*g2 b3^2+g3^2];

    % Incorporates K_e (elemental matrix) into K
    K(n1,n1)=K(n1,n1)+K_e(1,1);
    K(n1,n2)=K(n1,n2)+K_e(1,2);
    K(n1,n3)=K(n1,n3)+K_e(1,3);
    K(n2,n1)=K(n2,n1)+K_e(2,1);
    K(n2,n2)=K(n2,n2)+K_e(2,2);
    K(n2,n3)=K(n2,n3)+K_e(2,3);
    K(n3,n1)=K(n3,n1)+K_e(3,1);
    K(n3,n2)=K(n3,n2)+K_e(3,2);
    K(n3,n3)=K(n3,n3)+K_e(3,3);

    % calculates the element matrix for C
    % this will use a constant for mua value, but will have to changed
    % eventually; probably use a shape function to define a value over
the
    % element....
    C_e=mua*(area/12)*[2 1 1; 1 2 1; 1 1 2];
    % Incorporates C_e (elemental matrix) into C
    C(n1,n1)=C(n1,n1)+C_e(1,1);
    C(n1,n2)=C(n1,n2)+C_e(1,2);
    C(n1,n3)=C(n1,n3)+C_e(1,3);
    C(n2,n1)=C(n2,n1)+C_e(2,1);
    C(n2,n2)=C(n2,n2)+C_e(2,2);
    C(n2,n3)=C(n2,n3)+C_e(2,3);
    C(n3,n1)=C(n3,n1)+C_e(3,1);
    C(n3,n2)=C(n3,n2)+C_e(3,2);
    C(n3,n3)=C(n3,n3)+C_e(3,3);

    % calculates the element matrix for B
    % this will use a constant for speed of light value, but will have
to be
    % changed eventually; probably use a shape function to define a
value
    % over the element with respect to refractive index....
    B_e=(1/(3e11/n_in))*(area/12)*[2 1 1; 1 2 1; 1 1 2];
    % Incorporates C_e (elemental matrix) into C
    B(n1,n1)=B(n1,n1)+B_e(1,1);
    B(n1,n2)=B(n1,n2)+B_e(1,2);
    B(n1,n3)=B(n1,n3)+B_e(1,3);
    B(n2,n1)=B(n2,n1)+B_e(2,1);
    B(n2,n2)=B(n2,n2)+B_e(2,2);
    B(n2,n3)=B(n2,n3)+B_e(2,3);
    B(n3,n1)=B(n3,n1)+B_e(3,1);
    B(n3,n2)=B(n3,n2)+B_e(3,2);
    B(n3,n3)=B(n3,n3)+B_e(3,3);

    % Boundary condition applied to edge elements (segments)
    % tests each side of the element to see if it is on the boundary.
    % There will be problems when all three elements are on the
boundary,
    % but I will not worry about that now.

    % Since this code segment is for a specific kind of boundary, it
will
    % look to the m.bound column 1 to see if node is on boundary 1, and
    % look to m.bound column 2 for the exterior index of refraction

    %tracking variables
    flaggy=0; % this simply indicates that a boundary element is found
    b_type=0; % this indicates which column to find the index of
refraction
    % this is for the first boundary, or the internal
    if m.bound(n1,1) && m.bound(n2,1)
        b1=n1;
        b2=n2;
        flaggy=1;
        b_type=2;
    elseif m.bound(n1,1) && m.bound(n3,1)
        b1=n1;
        b2=n3;
        flaggy=1;
        b_type=2;
    elseif m.bound(n2,1) && m.bound(n3,1)
        b1=n2;
        b2=n3;
        flaggy=1;
        b_type=2;
    end
    % this is for the second boundary, or the internal
    if m.bound(n1,3) && m.bound(n2,3)
        b1=n1;
        b2=n2;
        flaggy=1;
        b_type=4;
    elseif m.bound(n1,3) && m.bound(n3,3)
        b1=n1;
        b2=n3;
        flaggy=1;
        b_type=4;
    elseif m.bound(n2,3) && m.bound(n3,3)
        b1=n2;
        b2=n3;
        flaggy=1;
        b_type=4;
    end
    if flaggy % the element is on a boundary
        % set the external index of refraction value
        if m.bound(b1,b_type)==m.bound(b2,b_type)
            % external index of refraction is equal
            n_out=m.bound(b1,b_type);
        else
            % external index of refraction is not the same for the
boundary
            % nodes
            disp('problem with boundary nodes indices of refraction not
consistent for edge elements');
            break;
        end
        %length calculation
        l=sqrt((m.x(b1)-m.x(b2))^2+(m.y(b1)-m.y(b2))^2);
        %constants for refractive index (ri) term
        Ro=((n_out-n_in)/(n_out+n_in))^2;
        theta_c=asin(n_out/n_in);
        ri=((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2);
        % Elemental matrix calculation
        A_e=(1/(2*ri))*(l/6)*[2 1; 1 2];
        % Incorporates A_e (segment matrix) into A
        A(b1,b1)=A(b1,b1)+A_e(1,1);
        A(b1,b2)=A(b1,b2)+A_e(1,2);
        A(b2,b1)=A(b2,b1)+A_e(2,1);
        A(b2,b2)=A(b2,b2)+A_e(2,2);
    end

    % Source matrix calculation - added to test new source
implementation
    s1=(area/3)*exp((-(m.x(n1)-xoff).^2)/(2*sigma))...
        .*exp((-(m.y(n1)-
yoff).^2)/(2*sigma)).*complex(cos(.15),sin(.15));
    s2=(area/3)*exp((-(m.x(n2)-xoff).^2)/(2*sigma))...
        .*exp((-(m.y(n2)-
yoff).^2)/(2*sigma)).*complex(cos(.15),sin(.15));
    s3=(area/3)*exp((-(m.x(n3)-xoff).^2)/(2*sigma))...
        .*exp((-(m.y(n3)-
yoff).^2)/(2*sigma)).*complex(cos(.15),sin(.15));
    source(n1)=source(n1)+s1;
    source(n2)=source(n2)+s2;
    source(n3)=source(n3)+s3;
end

% assembles the final connectivity matrix
all=K+C+A+i*2*pi*f*B;

field=all\source; %added to test new source implementation
```

8.14 Code for `cm_plot_terp_vcm.m`

```
function cm_plot_terp(X,Y,comsol,nirfast,savename)
% This function takes the fluence data only to plot the difference and
% calculate the error numbers.
% Think of comsol as the 'tested' data against the 'reference' (aka
% NIRFAST) data
% Created By Cameron Musgrove

%for the _vcm version.... comsol is now cameron's field values
% and nirfast is comsol values, since it is cameron's field values that
% need to be tested against the comsol results for accuracy, the comsol
% values are now the reference.

% removes imaginary and processes error as magnitude
comsol=abs(comsol);
nirfast=abs(nirfast);

% Normalizes the data
cdata=comsol/max(max(comsol));
ndata=nirfast/max(max(nirfast));
% Calculates the difference
data=cdata-ndata;

%Calculates the max difference
max_diff=max(max(abs(data)));

%Calculates the RMS error
%this will reassign all the values in the simulation domain to data2,
for
%error calculations to remove the impact of zeros or NaN that exist
outside
%the boundaries
data2=data(find(isfinite(data)));
RMS=sqrt(mean(mean(data2.^2)));
% find(isfinite(d(:,1))) will return the indices that are not NaN

% Makes a 3D Surf Plot
figure;
%set(axes,'FontSize',14);
surf(X,Y,data);
colormap bone;
colorbar;
shading interp;
xlabel('(mm)');
ylabel('(mm)');
zlabel('Normalized Fluence Difference');
title(['Global Fluence Max Difference = ' num2str(max_diff) '
\newlineGlobal Fluence RMS Error = ' num2str(RMS)]);

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_surf.jpg']);
```

```
    saveas(gcf,[savename '_surf.fig']);
    %close
end

% % Makes a Contour Plot
% figure;
% %set(axes,'FontSize',14);
% contour3(X,Y,data,20);
% %plots the circle
% hold on;
% ph=0:pi/18:2*pi;
% xc=43*cos(ph);
% yc=43*sin(ph);
% plot(xc,yc,'k')
% hold off;
% view(2);
% colorbar;
% set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
% axis equal;
% xlabel('(mm)');
% ylabel('(mm)');
% title(['Fluence Max Difference = ' num2str(max_diff) '
\newlineFluence RMS Error = ' num2str(RMS)]);
%
% % If there is a third arguement, save the ploted figure as a jpeg and
fig
% if nargin == 5
%     eval(['print -djpeg100 ' savename '_con.jpg']);
%     saveas(gcf,[savename '_con.fig']);
%     %close
% end

% Makes a plot of the error along a slice
figure;
%set(axes,'FontSize',14);
en=round(length(X)/2);
plot(X(en,:),cdata(en,:)-ndata(en,:),'k.');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Fluence Difference');
set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
d2=real(cdata(en,:))-real(ndata(en,:));
max_d=max(max(abs(d2)));
RMS2=sqrt(mean(mean(d2.^2)));
title(['Fluence Max Difference = ' num2str(max_d) ' \newlineFluence RMS
Error = ' num2str(RMS2)]);

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_dif.jpg']);
    saveas(gcf,['e:\ir_tomo\Validation\Report_Figures\' savename
'_dif.fig']);
    %close
end

% Makes a linear slice plot
```

```
figure;
%set(axes,'FontSize',14);
en=round(length(X)/2);
plot(X(en,:),ndata(en,:),'k+',...
    X(en,:),cdata(en,:),'ko');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Linear Fluence');
set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
d2=cdata(en,:)-ndata(en,:);
d2=d2(find(isfinite(d2)));
max_d=max(max(abs(d2)));
RMS2=sqrt(mean(mean(d2.^2)));
title(['Fluence Max Difference = ' num2str(max_d) ' \newlineFluence RMS
Error = ' num2str(RMS2)]);
legend('COMSOL','FEDA');

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_lin.jpg']);
    saveas(gcf,[savename '_lin.fig']);
    %close
end

% Makes a log slice plot
figure;
%set(axes,'FontSize',14);
plot(X(en,:),10*log10(ndata(en,:)),'k+',...
    X(en,:),10*log10(cdata(en,:)),'ko');
xlabel('X Axis (mm) at Y=0');
ylabel('Normalized Log Fluence (dB)');
set(gca,'xtick',[-43 -30 -20 -10 0 10 20 30 43]);
title(['Fluence Max Difference = ' num2str(10*log10(max_d)) ' dB
\newlineFluence RMS Error = ' num2str(RMS2)]);
legend('COMSOL','FEDA');

% If there is a fifth arguement, save the ploted figure as a jpeg and
fig
if nargin == 5
    eval(['print -djpeg100 e:\ir_tomo\Validation\Report_Figures\'
savename '_log.jpg']);
    saveas(gcf,[savename '_log.fig']);
    %close
end
```

8.15 Code for `element_delete_vcm.m`

```
% this will delete the internal nodes that result from delaunay in
matlab

function mesh=element_delete_vcm(mesh)

min_radius=min(sqrt(mesh.x.^2+mesh.y.^2));


int_nodes=find(sqrt(mesh.x.^2+mesh.y.^2)<min_radius*1.01);
newmesh.elements=zeros(length(mesh.elem)-length(int_nodes),3);
j=1;

for k=1:length(mesh.elem)
    %each should return 0 (false) if the elements contains a inner
boundary
    %node
    flag1=isempty(find(int_nodes(:)==mesh.elem(k,1)));
    flag2=isempty(find(int_nodes(:)==mesh.elem(k,2)));
    flag3=isempty(find(int_nodes(:)==mesh.elem(k,3)));
    %if one of the nodes in the element is not on the boundary, then
the
    %element can survive; flag will be true and the element will be
copied
    %to the new matrix.  If all nodes in the element are on the
boundary,
    %flag will be false, and the element will die.
    flag=flag1 || flag2 || flag3;

    if flag
        newmesh.elements(j,1)=mesh.elem(k,1);
        newmesh.elements(j,2)=mesh.elem(k,2);
        newmesh.elements(j,3)=mesh.elem(k,3);
        j=j+1;
    end

end

mesh.elem=newmesh.elements;
```

8.16 Code for `insert_cm_blob_vcm.m`

```
% this inserts a blob where cameron says

function cmesh=insert_cm_blob_vcm(mesh,mua,radius,x,y)
cmesh=mesh;
%recenter entire geometry so that the blob is the center!
mesh.x=mesh.x-x;
mesh.y=mesh.y-y;

for i=1:length(mesh.x)
    node_radius=sqrt(mesh.x(i)^2+mesh.y(i)^2);
    if node_radius<.99*radius %node should be within blob
        mesh.mua(i)=mua;
    end
end

cmesh.mua=mesh.mua;
```

8.17 Code for `sensitivity.m`

```
% sensitivity.m
% Written by Cameron Musgrove
% This file executes all the COMSOL simulations to create one figure to
% compare the results.

% absorption coefficient of target
mua_b=.004;

% Detector location
xval=10;

val=zeros(9,1);

ref=endoscopic_1(xval);
val(1)=endoscopic_1_hetero_n1(mua_b,xval);
val(2)=endoscopic_1_hetero_n2(mua_b,xval);
val(3)=endoscopic_1_hetero_n3(mua_b,xval);
val(4)=endoscopic_1_hetero_n4(mua_b,xval);
val(5)=endoscopic_1_hetero_n5(mua_b,xval);
val(6)=endoscopic_1_hetero_n6(mua_b,xval);
val(7)=endoscopic_1_hetero_n7(mua_b,xval);
val(8)=endoscopic_1_hetero_n8(mua_b,xval);
val(9)=endoscopic_1_hetero_n9(mua_b,xval);

%figure; plot(1:3,abs(ref)-abs(val(1:3)),4:6,abs(ref)-
abs(val(4:6)),7:9,abs(ref)-abs(val(7:9)));
ref2=abs(ref)*ones(9,1);
figure; plot(1:3,abs(val(1:3)),'-ko',...
    4:6,abs(val(4:6)),'-k*',...
    7:9,abs(val(7:9)),'-ks','LineWidth',2,'MarkerSize',8);
hold on; plot(1:9,ref2,'--k','LineWidth',2,'MarkerSize',8); hold off;

set(gca,'yTick',.009:.0001:.010);  % this does not do what it is
supposed to!!!!!!!

ylabel('Fluence');
xlabel('Position Number');

%% Code for the data pull
tic
x=-30:1:30;
len=length(x);
data=zeros(1,len);
y=0*ones(1,len);
data(1,:)=postinterp(fem,'u',[x;y]);
toc

figure; plot(x,log(data))

%%
savename='double_value';
eval(['print -djpeg100 ' savename '.jpg']);
saveas(gcf,[savename '.fig']);
```

8.18 Code for `endoscopic_1.m`

```
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

function data_out=endoscopic_1(xval)

flclear fem

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.3';
vrsn.ext = '';
vrsn.major = 0;
vrsn.build = 405;
vrsn.rcs = '$Name:  $';
vrsn.date = '$Date: 2006/08/31 18:03:47 $';
fem.version = vrsn;

% Geometry
g1=ellip2(1,1,'base','center','pos',[0,0]);
g1=scale(g1,10,10,0,0);
[g2]=geomcopy({g1});
[g3]=geomcopy({g2});
g3=move(g3,[0,0]);
g3=scale(g3,3,3,0,0);
g4=geomcomp({g3},'ns',{'g3'},'sf','g3','edge','none');
g5=geomcomp({g1},'ns',{'g1'},'sf','g1','edge','none');
g6=geomcomp({g4,g5},'ns',{'g4','g5'},'sf','g4-g5','edge','none');

% Geometry objects
clear s
s.objs={g6};
s.name={'CO3'};
s.tags={'g6'};

fem.draw=struct('s',s);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear pnt
pnt.weak = {};
pnt.dweak = {};
pnt.constr = {};
pnt.name = {};
pnt.ind = [];
appl.pnt = pnt;
```

```
clear bnd
bnd.type = {};
bnd.r = {};
bnd.h = {};
bnd.weak = {};
bnd.dweak = {};
bnd.constr = {};
bnd.g = {};
bnd.q = {};
bnd.name = {};
bnd.ind = [];
appl.bnd = bnd;
clear equ
equ.init = {};
equ.be = {};
equ.c = {};
equ.dweak = {};
equ.ea = {};
equ.constr = {};
equ.cporder = {};
equ.da = {};
equ.gporder = {};
equ.al = {};
equ.a = {};
equ.weak = {};
equ.f = {};
equ.usage = {};
equ.ga = {};
equ.name = {};
equ.dinit = {};
equ.ind = [];
equ.bnd.gporder = {};
equ.bnd.weak = {};
equ.bnd.ind = [];
appl.equ = equ;
fem.appl{1} = appl;
fem.sdim = {'x','y'};
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Geometry

% Analyzed geometry
clear s
s.objs={g6};
s.name={'CO3'};
s.tags={'g6'};
```

```
fem.draw=struct('s',s);
fem.geom=geomcsg(fem);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'dir';
bnd.ind = [1,1,1,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-40', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A_out','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-
abs(cos(theta_c))^2)'};

% Initialize mesh
fem.mesh=meshinit(fem, ...
                  'hauto',5);

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-40', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-40', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = {'1/2','1/(2*A)'};
bnd.ind = [1,1,2,2,1,2,2,1];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
```

```
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-48.98367346938776,48.983673469387746,-33,33,-1,1]);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = '1/(2*A)';
bnd.ind = [1,1,1,1,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-49.938274116862154,58.005029866781165,-
35.7172619047619,35.71726190476191,-1,1]);

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-12', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = '1/(2*A)';
bnd.ind = [1,1,1,1,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});
```

```
% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-48.98367346938776,57.05042921930677,-
37.97451495101063,37.97451495101065,-1,1]);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = {'1/(2*A)','1/(2)'};
bnd.ind = [2,2,1,1,2,1,1,2];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
```

```
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-50.09312976974834,75.36077496333117,-
41.51132869719678,41.511328697196795,-1,1]);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Refine mesh
fem.mesh=meshrefine(fem, ...
                    'mcase',0, ...
                    'rmethod','regular');

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = {'1/(2*A)','1/(2)'};
bnd.ind = [2,2,1,1,2,1,1,2];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Mapping current solution to extended mesh
init = asseminit(fem,'init',fem0.sol,'xmesh',fem0.xmesh);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',init, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-48.98367346938776,83.87144156658354,-
45.54336665761068,43.96030845828198,-1,1]);

% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-12', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-12', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = {'1/(2*A)','1/(2)'};
bnd.ind = [2,2,1,1,2,1,1,2];
appl.bnd = bnd;
```

```
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-55.071348798161274,89.39729728965122,-
47.80309920962449,47.80309920962449,-1,1]);

% data pull!
x=-30:1:30;
len=length(x);
data=zeros(1,len);
y=0*ones(1,len);
data(1,:)=postinterp(fem,'u',[x;y]);

ind=find(x==xval);
data_out=data(ind);
end
```

8.19 Code for endoscopic_1_hetero_n1.m

```
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

function data_out=endoscopic_1_hetero_n1(mua_b,xval)

flclear fem

% COMSOL version
clear vrsn
vrsn.name = 'COMSOL 3.3';
vrsn.ext = '';
vrsn.major = 0;
vrsn.build = 405;
vrsn.rcs = '$Name:  $';
vrsn.date = '$Date: 2006/08/31 18:03:47 $';
fem.version = vrsn;

% Geometry
g1=ellip2(1,1,'base','center','pos',[0,0]);
g1=scale(g1,10,10,0,0);
[g2]=geomcopy({g1});
[g3]=geomcopy({g2});
g3=move(g3,[0,0]);
g3=scale(g3,3,3,0,0);
g4=geomcomp({g3},'ns',{'g3'},'sf','g3','edge','none');
g5=geomcomp({g1},'ns',{'g1'},'sf','g1','edge','none');
g6=geomcomp({g4,g5},'ns',{'g4','g5'},'sf','g4-g5','edge','none');

% Geometry objects
clear s
s.objs={g6};
s.name={'CO3'};
s.tags={'g6'};

fem.draw=struct('s',s);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear pnt
pnt.weak = {};
pnt.dweak = {};
pnt.constr = {};
pnt.name = {};
pnt.ind = [];
appl.pnt = pnt;
```

```
clear bnd
bnd.type = {};
bnd.r = {};
bnd.h = {};
bnd.weak = {};
bnd.dweak = {};
bnd.constr = {};
bnd.g = {};
bnd.q = {};
bnd.name = {};
bnd.ind = [];
appl.bnd = bnd;
clear equ
equ.init = {};
equ.be = {};
equ.c = {};
equ.dweak = {};
equ.ea = {};
equ.constr = {};
equ.cporder = {};
equ.da = {};
equ.gporder = {};
equ.al = {};
equ.a = {};
equ.weak = {};
equ.f = {};
equ.usage = {};
equ.ga = {};
equ.name = {};
equ.dinit = {};
equ.ind = [];
equ.bnd.gporder = {};
equ.bnd.weak = {};
equ.bnd.ind = [];
appl.equ = equ;
fem.appl{1} = appl;
fem.sdim = {'x','y'};
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Geometry

% Analyzed geometry
clear s
s.objs={g6};
s.name={'CO3'};
s.tags={'g6'};
```

```
fem.draw=struct('s',s);
fem.geom=geomcsg(fem);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'dir';
bnd.ind = [1,1,1,1,1,1,1,1];
appl.bnd = bnd;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-40', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A_out','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-
abs(cos(theta_c))^2)'};

% Initialize mesh
fem.mesh=meshinit(fem, ...
                  'hauto',5);

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-40', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-40', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = {'1/2','1/(2*A)'};
bnd.ind = [1,1,2,2,1,2,2,1];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
```

```
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-48.98367346938776,48.983673469387746,-33,33,-1,1]);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = '1/(2*A)';
bnd.ind = [1,1,1,1,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-49.938274116862154,58.005029866781165,-
35.7172619047619,35.71726190476191,-1,1]);

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-12', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = '1/(2*A)';
bnd.ind = [1,1,1,1,1,1,1,1];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});
```

```
% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-48.98367346938776,57.05042921930677,-
37.97451495101063,37.97451495101065,-1,1]);

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = {'1/(2*A)','1/(2)'};
bnd.ind = [2,2,1,1,2,1,1,2];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',fem0.sol, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
```

```
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-50.09312976974834,75.36077496333117,-
41.51132869719678,41.511328697196795,-1,1]);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Refine mesh
fem.mesh=meshrefine(fem, ...
                    'mcase',0, ...
                    'rmethod','regular');

% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = {'1/(2*A)','1/(2)'};
bnd.ind = [2,2,1,1,2,1,1,2];
appl.bnd = bnd;
clear equ
equ.c = '1/(3*(mua+mus))';
equ.da = 0;
equ.a = 'mua+i*2*pi*100e6/c_light';
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Mapping current solution to extended mesh
init = asseminit(fem,'init',fem0.sol,'xmesh',fem0.xmesh);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',init, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-48.98367346938776,83.87144156658354,-
45.54336665761068,43.96030845828198,-1,1]);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-12', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)'};

% Geometry
g1=circ2(5,'base','center','pos',[5,0]);
g2=circ2(5,'base','center','pos',[-19,0]);
[g3]=geomcopy({g2});
[g4]=geomcopy({g3});
g4=move(g4,[0,0]);
g5=geomcoerce('solid',{g4});

% Geometry objects
clear s
s.objs={g6,g2,g5};
s.name={'CO3','C1','CO1'};
s.tags={'g6','g2','g5'};

fem.draw=struct('s',s);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Geometry
g5=move(g5,[8,0]);
g5=move(g5,[-8,0]);
g5=move(g5,[4,0]);
g5=move(g5,[-10,0]);

% Geometry objects
clear s
s.objs={g6,g5};
```

```
s.name={'CO3','CO1'};
s.tags={'g6','g5'};

fem.draw=struct('s',s);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Geometry
g5=move(g5,[10,0]);

% Geometry objects
clear s
s.objs={g6,g5};
s.name={'CO3','CO1'};
s.tags={'g6','g5'};

fem.draw=struct('s',s);
% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Geometry
[g2]=geomcopy({g5});
[g3]=geomcopy({g5});
g4=geomcomp({g6,g5},'ns',{'CO3','CO1'},'sf','CO3-CO1','edge','none');
[g7]=geomcopy({g3});
g7=move(g7,[0,0]);

% Geometry objects
clear s
s.objs={g4,g7};
s.name={'CO2','CO1'};
s.tags={'g4','g7'};

fem.draw=struct('s',s);

% COMSOL Multiphysics Model M-file
% Generated by COMSOL 3.3 (COMSOL 3.3.0.405, $Date: 2006/08/31 18:03:47
$)

% Geometry

% Analyzed geometry
clear s
s.objs={g4,g7};
s.name={'CO2','CO1'};
s.tags={'g4','g7'};

fem.draw=struct('s',s);
fem.geom=geomcsg(fem);

% Constants
fem.const = {'mua','.002', ...
  'mus','.5', ...
  'sigma','8', ...
  'xoff','-12', ...
  'yoff','0', ...
  'n_in','1.3', ...
  'n_out','1', ...
  'c_light','3e11/n_in', ...
  'Ro','((n_out-n_in)/(n_out+n_in))^2', ...
  'theta_c','asin(n_out/n_in)', ...
  'A','((2/(1-Ro))-1+abs(cos(theta_c))^3)/(1-abs(cos(theta_c))^2)', ...
  'mua_b',num2str(mua_b)};

% Initialize mesh
fem.mesh=meshinit(fem, ...
                  'hauto',5);
fem.mesh=meshrefine(fem, ...
                    'mcase',0, ...
                    'rmethod','regular');
% (Default values are not included)

% Application mode 1
clear appl
appl.mode.class = 'FlPDEC';
appl.assignsuffix = '_c';
clear prop
prop.elemdefault='Lag1';
appl.prop = prop;
clear bnd
bnd.type = 'neu';
bnd.q = {'1/(2*A)','1/(2)',0};
bnd.ind = [2,2,3,3,3,3,1,1,2,1,1,2];
appl.bnd = bnd;
clear equ
equ.c = {'1/(3*(mua+mus))','1/(3*(mua_b+mus))'};
equ.da = 0;
equ.a = {'mua+i*2*pi*100e6/c_light','mua_b+i*2*pi*100e6/c_light'};
equ.f = 'exp((-(x-xoff)^2)/(2*sigma))*exp(-(y-
yoff)^2/(2*sigma))*complex(cos(.15),sin(.15))';
equ.ind = [1,2];
equ.bnd.ind = [1,1];
appl.equ = equ;
fem.appl{1} = appl;
fem.frame = {'ref'};
fem.border = 1;
clear units;
units.basesystem = 'SI';
fem.units = units;

% Multiphysics
fem=multiphysics(fem);

% Extend mesh
fem.xmesh=meshextend(fem);

% Mapping current solution to extended mesh
init = asseminit(fem,'init',fem0.sol,'xmesh',fem0.xmesh);

% Solve problem
fem.sol=femstatic(fem, ...
                  'init',init, ...
                  'solcomp',{'u'}, ...
                  'outcomp',{'u'});

% Save current fem structure for restart purposes
fem0=fem;

% Plot solution
postplot(fem, ...
         'tridata',{'u','cont','internal'}, ...
         'trimap','jet(1024)', ...
         'title','Surface: u', ...
         'axis',[-48.98367346938775,48.98367346938775,-33,33,-1,1]);
% data pull!
x=-30:1:30;
len=length(x);
data=zeros(1,len);
y=0*ones(1,len);
data(1,:)=postinterp(fem,'u',[x;y]);

ind=find(x==xval);
data_out=data(ind);
end
```

8.20 Code for sensitivity_vcm.m

```
% sensitivity.m
% Written by Cameron Musgrove

% This file executes multiple FEDA simulations to create one figure to
% compare the results.

% absorption coefficient of target
mua_b=.2;

% Detector location
xval=10;
yval=0;

% Captures
val=zeros(9,1);

filename='endo1_dense';
comsol=read_it_in_v3([filename '.txt'],'y');

% determine index for the node at detector
% this is going to work for COMSOL meshes, in general, but really
because
% I know that a node sits right on the place I want the detector to
be!!
xind=find(comsol.coord(:,1)==xval);
ind=find(comsol.coord(xind,2)==yval);
index=xind(ind);

[field,cmesh]=cm_2D_v4(comsol);
ref=field(index);

[field,cmesh]=cm_2D_v5(comsol,mua_b,5,-15,0);
val(1)=field(index);
[field,cmesh]=cm_2D_v5(comsol,mua_b,5,-20,0);
val(2)=field(index);
[field,cmesh]=cm_2D_v5(comsol,mua_b,5,-25,0);
val(3)=field(index);
[field,cmesh]=cm_2D_v5(comsol,mua_b,5,0,15);
val(4)=field(index);
[field,cmesh]=cm_2D_v5(comsol,mua_b,5,0,20);
val(5)=field(index);
[field,cmesh]=cm_2D_v5(comsol,mua_b,5,0,25);
val(6)=field(index);
[field,cmesh]=cm_2D_v5(comsol,mua_b,5,15,0);
val(7)=field(index);
[field,cmesh]=cm_2D_v5(comsol,mua_b,5,20,0);
val(8)=field(index);
[field,cmesh]=cm_2D_v5(comsol,mua_b,5,25,0);
val(9)=field(index);

%figure; plot(1:3,abs(ref)-abs(val(1:3)),4:6,abs(ref)-
abs(val(4:6)),7:9,abs(ref)-abs(val(7:9)));
ref2=abs(ref)*ones(9,1);
```

```
figure; plot(1:3,abs(val(1:3)),'-ko',...
    4:6,abs(val(4:6)),'-k*',...
    7:9,abs(val(7:9)),'-ks','LineWidth',2,'MarkerSize',8);
hold on; plot(1:9,ref2,'--k','LineWidth',2,'MarkerSize',8); hold off;

%set(gca,'yTick',.009:.0001:.010);  % this does not do what it is
supposed to!!!!!!!

ylabel('Fluence');
xlabel('Position Number');

%%
savename='FEDA_double_value';
eval(['print -djpeg100 ' savename '.jpg']);
saveas(gcf,[savename '.fig']);
```

8.21 Code for `cm_2D_v5.m`

```
% this script initializes the data required to execute make_2d()
% this is used for development and can be modified to make use of
make_2d()
% for user-specific applications.

% v3 pulls coordinate data directly from exported fem object
% v4 uses data from a comsol .txt file
% v5 modified the input arguements for the sensitivity study
% developed for endoscopic

% Created by Cameron Musgrove 2007!

function [field,cmesh]=cm_2D_v5(fem,mua_b,radius,x_loc,y_loc)

%pull node coordinates from COMSOL object fem
% cmesh.x=fem.mesh.p(1,:)';
% cmesh.y=fem.mesh.p(2,:)';
% cmesh.belem=fem.mesh.e(1:2,:)'; %boundary elements
cmesh.x=fem.coord(:,1);
cmesh.y=fem.coord(:,2);
cmesh.belem=fem.belem; %boundary elements

% creates elements
cmesh.elem=delaunay(cmesh.x,cmesh.y);
cmesh=element_delete_vcm(cmesh);
% Determine boundary nodes, assuming boundary nodes are the first
nodes...
cmesh=divide_bc(cmesh,1.3,1);
% verifies the boundary split was successful
% figure;
plot(cmesh.x(find(cmesh.bound(:,1))),cmesh.y(find(cmesh.bound(:,1))),'r
.',cmesh.x(find(cmesh.bound(:,3))),cmesh.y(find(cmesh.bound(:,3))),'c.'
);



% Sets Material Parameters for Simulation Background Values
node=length(cmesh.x);
cmesh.mua=.002*ones(node,1);
cmesh.mus=.5*ones(node,1);
cmesh.ri=1.3*ones(node,1);

% corrects boundary conditions for heterogeneous simulations
% cmesh=make_external_ring_boundary_vcm(cmesh);
% adds blob for heterogeneous simualtions
cmesh=insert_cm_blob_vcm(cmesh,mua_b,radius,x_loc,y_loc);

% Modulation Frequency
f=100e6;

% Finite Element Crunch
A=make_2d_2(cmesh,f);
```

```
% Source Term
xoff=-12;
yoff=0;
sigma=8;
source=exp((-(cmesh.x-xoff).^2)/(2*sigma)).*...
    exp((-(cmesh.y-yoff).^2)/(2*sigma)).*complex(cos(.15),sin(.15));

% Fluence Calculation
%field=A\source;
field=A; %added for new source implementation
```

8.22 Code for `read_it_in_v3.m`

```
function dataout=read_it_in_v3(filename,flag)
%filename='First_try.txt';
fid=fopen(filename);

if nargin==2
    %find the coordinates and elements, too
    %junks the coordinates header!
    junk=fgetl(fid);

    %initalize the coordiate list for the nodes
    coord=str2num(fgetl(fid));

    line=fgetl(fid);
    while(not(isequal('% Elements (triangular)',line)))
        coord=[coord; str2num(line)];
        line=fgetl(fid);
    end

    % checks the file to see if COMSOL backward-ed the data
    % checks to see if the coord data is actually the boundary data,
then
    % it reassigns the data in coord to bcoord
    check=size(coord);
    if check(2)==3
        %then need to re-assign the coord data to bcoord data
        %but first grab the boundary elemental data
        elem=str2num(fgetl(fid));
        line=fgetl(fid);
        while(not(isequal('% Coordinates',line)))
            elem=[elem; str2num(line)];
            line=fgetl(fid);
        end
        bcoord=coord;
        clear coord
        belem=elem;
        clear elem
        %grab the REAL coordinates
        coord=str2num(fgetl(fid));
        line=fgetl(fid);
        while(not(isequal('% Elements (triangular)',line)))
            coord=[coord; str2num(line)];
            line=fgetl(fid);
        end
    end

    %initialize the element list
    elem=str2num(fgetl(fid));

    line=fgetl(fid);
%    while(not(isequal('% Data (u)',line(1:10))))
%    while(not(isequal('% Data,',line(1:7)))) %added
    while(not(isequal('% Data',line(1:6)))) %added another species
        elem=[elem; str2num(line)];
        line=fgetl(fid);
    end
    if check(2)==3
        %if the boundary data was first, then don't look for it at the
end
        %just grab the data information at the end
        data=str2num(fgetl(fid));
        while(not(feof(fid)))
            data=[data; str2num(fgetl(fid))];
        end
        fclose(fid);
    else
        %the boundary data is at the end of the file
        %initialize the data list
        data=str2num(fgetl(fid));
    %     while(not(feof(fid)))
        line=fgetl(fid); %added
        while(not(isequal('% Coordinates',line))) %added
    %         data=[data; str2num(fgetl(fid))];
            data=[data; str2num(line)]; %added
            line=fgetl(fid);%added
        end

        %initialize boundary coordinate list
        bcoord=str2num(fgetl(fid));
        while(not(isequal('% Elements (triangular)',line)))
            bcoord=[bcoord; str2num(line)];
            line=fgetl(fid);
        end

        %initialize boundary element list
        belem=str2num(fgetl(fid));
        while(not(feof(fid)))
            belem=[belem; str2num(fgetl(fid))];
        end
        fclose(fid);
    end
    dataout.coord=coord;
    dataout.elem=elem;
    dataout.data=data;
    dataout.bcoord=bcoord;
    dataout.belem=belem;
elseif nargin==1 %if there is only one input arguement, just return the
data information
    line=fgetl(fid);
    %burn down the file to the data information
    while(not(isequal('% Data (u)',line(1:10))))
        line=fgetl(fid);
    end
    %initialize the data list
    data=str2num(fgetl(fid));
    while(not(feof(fid)))
        data=[data; str2num(fgetl(fid))];
    end
    fclose(fid);
    dataout=data;
end
```

```
% %%
% trisurf(elem,coord(:,1),coord(:,2),data);
% view(2); shading interp; axis equal; grid off;
%
% %to view the log data
% figure; trisurf(elem,coord(:,1),coord(:,2),log(abs(data)));
% view(2); shading interp; axis equal; grid off;
```

VITA

Cameron Hollis Musgrove

Candidate for the Degree of Master of Science

Thesis: ISSUES RELATED TO THE FORWARD PROBLEM FOR ENDOSCOPIC NEAR-INFRARED DIFFUSE OPTICAL TOMOGRAPHY

Major Field: Electrical Engineering

Biographical:

Personal Data:

Interests include remote sensing, RF propagation, electromagnetic modeling, and hardware development and testing.

Education:

Bachelor of Science in Electrical Engineering, Magna Cum Laude, Honors College Degree, Oklahoma State University, Stillwater, Oklahoma, December 2005.
Completed the requirements for the Master of Science in Electrical Engineering at Oklahoma State University at Stillwater, Oklahoma in December 2007.

Experience:

Graduate Research Assistant, 2007
"Trans-rectal Near-Infrared Optical Tomography for Prostate Imaging," Department of Defense Prostate Cancer Research Program Award
Oklahoma State University, Stillwater, Oklahoma
Design Engineer Intern, 2007
Garmin International, Olathe, KS
Graduate Research Assistant, 2006
"Collaborative Research: Engineering Students for the 21st Century," NSF Grant
Oklahoma State University, Stillwater, Oklahoma
Intern/Student Engineer, 2004
International Space Station Electrical Power System
The Boeing Company, Houston, Texas

Professional Memberships:

Eta Kappa Nu
IEEE (Institute of Electrical and Electronics Engineers)
SPIE (Society of Photo-Optical Instrumentation Engineers)

Name: Cameron Musgrove Date of Degree: December 2007

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: ISSUES RELATED TO THE FORWARD PROBLEM FOR ENDOSCOPIC NEAR-INFRARED DIFFUSE OPTICAL TOMOGRAPHY

Pages in Study: 145 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study:

Near-infrared diffuse optical tomography (NIR DOT) imaging technology has been shown to be capable of detecting the presence of certain types of cancer within the body in a non-invasive way. At Oklahoma State University, researchers are attempting to apply the NIR DOT imaging to help doctors define suspicious lesions for targeted prostate tissue biopsy. Available software for NIR DOT imaging have proven to be inaccurate when applied to prostate imaging. A new software package must be developed to create accurate optical images of the prostate. As part of this new software package, this work details the development and validation of a new frequency-domain diffusion approximation finite element model to predict the light fluence throughout the prostate.

Findings and Conclusions:

Using a simplified model of the breast and cancer suspect, the new model is shown to be nearly identical to another software package, NIRFAST, that has been proven to be an accurate NIR DOT breast cancer imaging software package. Applying the new model for a simplified prostate model has indicated a different response from the inaccurate NIRFAST result. The new frequency-domain diffusion approximation finite element model is ready to be tested against laboratory measurements.

ADVISOR’S APPROVAL: Dr. Charles F. Bunting .