

**Improvements in the robustness and accuracy of
bioluminescence tomographic reconstructions of distributed
sources within small animals**

Bradley J. Beattie

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Engineering Science
in the Fu Foundation School of Engineering
and Applied Science

COLUMBIA UNIVERSITY

2018

ABSTRACT

Improvements in the robustness and accuracy of bioluminescence tomographic reconstructions of distributed sources within small animals

Bradley J. Beattie

High quality three-dimensional bioluminescence tomographic (BLT) images, if available, would constitute a major advance and provide much more useful information than the two-dimensional bioluminescence images that are frequently used today. To-date, high quality BLT images have not been available, largely because of the poor quality of the data being input into the reconstruction process. Many significant confounds are not routinely corrected for and the noise in this data is unnecessarily large and poorly distributed. Moreover, many of the design choices affecting image quality are not well considered, including choices regarding the number and type of filters used when making multispectral measurements and choices regarding the frequency and uniformity of the sampling of both the range and domain of the BLT inverse problem. Finally, progress in BLT image quality is difficult to gauge owing to a lack of realistic gold-standard references that engage the full complexity and uncertainty within a small animal BLT imaging experiment.

Within this dissertation, I address all of these issues. I develop a Cerenkov-based gold-standard wherein a Positron Emission Tomography (PET) image can be used to gauge

improvements in the accuracy of BLT reconstruction algorithms. In the process of creating this reference, I discover and describe corrections for several confounds that if left uncorrected would introduce artifacts into the BLT images. This includes corrections for the angle of the animal's skin surface relative to the camera, for the height of each point on the skin surface relative to the focal plane, and for the variation in bioluminescence intensity as a function of luciferin concentration over time. Once applied, I go on to derive equations and algorithms that when employed are able to minimize the noise in the final images under the constraints of a multispectral BLT data acquisition. These equations and algorithms allow for an optimal choice of filters to be made and for the acquisition time to be optimally distributed among those filtered measurements. These optimizations make use of Barrett's and Moore-Penrose pseudoinverse matrices which also come into play in a paradigm I describe that can be used to guide choices regarding sampling of the domain and range.

Table of Contents

List of Figures	v
List of Tables	vi
1. INTRODUCTION	1
1.1. Background and motivation	1
1.2. The BLT inverse problem	5
1.3. Multispectral vs. hyperspectral	6
1.4. Overall Goals and Specific Aims	12
2. SPECIFIC AIM 1: MULTIMODALITY REGISTRATION	14
2.1. Overview	14
2.2. Scanners.	18
2.3. Overview of the registration procedure.	18
2.4. Registration of a 2D image to a 3D image set.	21
2.5. BLI camera model.	22
2.6. BLI camera calibration.	23

2.7.	Registration accuracy.	24
2.8.	Animals and imaging procedures.	26
2.9.	Imaging protocols.	28
2.10.	Registration accuracy.	29
2.11.	Experiments	30
2.12.	Light Fall-off Correction.	34
2.13.	Discussion	36
3.	SPECIFIC AIM 2: CERENKOV QUANTITATION	38
3.1.	Overview	38
3.2.	Modeling.	41
3.3.	Experimental Cerenkov measurements.	49
3.4.	Spectral measurements.	52
3.5.	Cerenkov efficiency as a function of refractive index.	53
3.6.	Cerenkov efficiency as a function of volume.	53
3.7.	Region of interest measurements and profiles.	54

3.8.	Corrections	55
3.9.	Comparisons	60
3.10.	Modeled Cerenkov production efficiencies as a function of refractive index.	67
3.11.	Modeled Cerenkov point spread functions.	69
3.12.	Discussion	69
4.	SPECIFIC AIM 3: OPTIMIZED ACQUISITION PROTOCOL	73
4.1.	Overview	73
4.2.	Camera noise model	75
4.3.	Digital mouse phantom	76
4.4.	Solving the forward model	77
4.5	Image Reconstruction Methods	80
4.6.	Derivation of the optimization expression	84
4.7.	Algorithm to select optimal filters	90
4.8.	Testing the optimization expression	92
4.9.	Two-wavelength LSQ simulation results	94

4.10.	Three-wavelength LSQ simulation	97
4.11.	Two-wavelength MLEM simulation	98
4.12.	Mouse simulations	99
4.13.	Guidance for improved conditioning and SNR through optimized sampling	103
4.14.	Discussion	107
5.	OVERALL SUMMARY AND CONCLUSIONS	109
	REFERENCES	112
	APPENDIX	121

List of Figures

1.1	Artifacts in BLT	1
2.1	Mouse Bed	19
2.2	Transgenic Mouse	26
2.3	Light Fall-off with Angle in Animal.....	31
2.4	Registered CT, Reflectance and BLI Images	32
2.5	Light Flux versus Time Curve	33
2.6	Parameter Defining Diagram	34
2.7	Light Fall-off with Angle in Phantom	35
3.1	Cerenkov versus Wavelength, Radioactivity and Height	56
3.2	Experimental Setup and Radionuclide Cerenkov Results	61
3.3	Cerenkov Point Spread Function.....	65
3.4	Secondary Electron Point Spread Function	66
3.5	Cerenkov versus Refractive Index.....	68
4.1	Pathlength Histogram	78
4.2	Log-mean and Log-standard-deviation of Pathlength Distribution	79
4.3	Importance of Politte's Correction.....	84
4.4	Optimal Time Distribution for 2-Wavelength Moore-Penrose Solution.....	95
4.5	Uncertainty for 2-Wavelength Moore-Penrose Solution.....	96
4.6	Optimal Time Distribution for 3-Wavelengths Moore-Penrose Solution.....	97
4.7	Uncertainty for 2-Wavelength MLEM Solution	98
4.8	Optimal Time Distribution for 2-Wavelength MLEM Solution	99

4.9	Reference Image, Source Spectra and Attenuation	100
4.10	Firefly and Cerenkov Optimal Time Distributions.....	101
4.11	Improved MLEM Images from Optimally Sampled Data.....	102
4.12	Improved Moore-Penrose Images from Optimally Sampled Data.....	103
4.13	Optimized Domain Sampling	106

List of Tables

2.1	Registration Error	30
3.1	Radionuclide Abundances	43
3.2	Ac-225 Daughter Abundances.....	44
3.3	Experiment Overview	50
3.4	Cerenkov Efficiencies	68
3.5	Cerenkov Point Spread Function Widths.....	69

1. INTRODUCTION

1.1. *Background and motivation*

Bioluminescence tomography (BLT) reconstruction algorithms have been available to researchers since Wang first described their theoretical basis in 2004 [1]. Today, BLT reconstruction capabilities are included as standard software accompanying the ubiquitous IVIS bioluminescence imaging systems (Perkin Elmer). However, these algorithms have seen relatively little use, as evidenced in the literature by the relative dearth of articles published that actually make use of BLT.

BLT has not been well embraced, I believe, because of the generally poor quality of the BLT reconstructed images (suffering from clear artifacts, see figure 1.1, and because they are often noisy and unstable) and because the accuracy of these images has not been validated under realistic conditions (i.e. researchers don't trust them to be accurate).

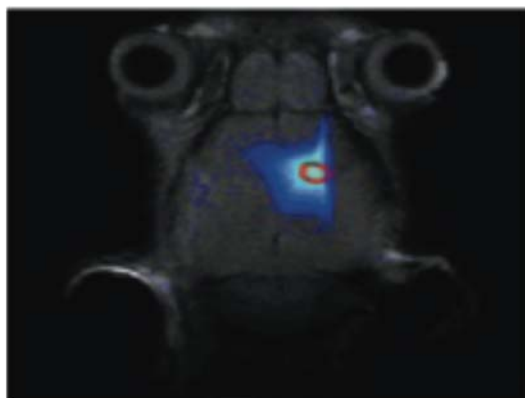


Figure 1.1 Although the bioluminescent tumor implanted in the brain of this mouse is roughly spherical, the source distribution shown here, reconstructed by Chaudhari et al. clearly is not.

It is well recognized that the BLT image reconstruction problem is ill-posed, requiring multispectral measurements to avoid degeneracy and often remains ill-conditioned even when multispectral measurements are made [2-4]. Thus it is typical for even small amounts of noise in the measured data to be greatly amplified during the reconstruction

process, producing images with a large degree of uncertainty, high noise levels and/or artifacts [5].

Methods of improving image quality can be roughly categorized into three different classes: 1) improvements in the precision and accuracy of the data being input into the image reconstruction algorithms, for example, the uncertainty in the measured data can be reduced by increasing the duration of the acquisition (i.e. counting more photons) [6]; 2) the condition number of the system matrix itself can be improved through design choices, for example, by reducing the number of unknowns (i.e. voxels) or increasing the information content of the measurements [‡]; and 3) images can be improved through the application of advanced reconstruction algorithms [7-12].

Most efforts aimed at improving BLT image quality to date have fallen into this third category and generally speaking these involve some means of applying *a priori* information to constrain the solution space [13]. These constraints can reflect the realities of the physical system (e.g. allowing only positive solutions, i.e. photon intensities ≥ 0) or they can come in the form of biases reflecting expectations regarding, for example, the smoothness of the image, its sparsity, or limits on the source location. Generally this will involve a tradeoff between noise and bias and will only be successful to the extent that the *a priori* information that is being applied, whether implicitly or explicitly, is itself accurate [14]. In avoidance of imposing these sorts of biases, I will

[‡] This can readily be appreciated by considering the extremes. A system in which we seek to determine the intensity of a single large homogeneous voxel (i.e. reducing the number of unknowns to 1), clearly has a low condition number. Similarly, increasing the number of unknowns to the point where they exceed the number of measurements, creates an ill-posed problem with multiple solutions.

instead focus on improving image quality through the approaches within the first two categories.

These improvements will come in the form of precise corrections for confounds encountered in the measurement process, including corrections for the varying heights and surface normals of the CCD measurements, improvements in the accuracy of camera and filter calibrations, and corrections accounting for the luciferin concentration dependent fluctuations in source intensity over time. The noise reduction and conditioning improvements that I propose, will make use of either the Moore-Penrose pseudoinverse [15, 16] of the system matrix, or of Barrett's error matrix [17, 18]. I will show that it is possible to use these matrices to predict the uncertainty in images reconstructed with least-squares and maximum-likelihood expectation maximization (MLEM) algorithms, respectively. I will then demonstrate how this capability can be leveraged in a paradigm that will, among other things, allow the optimization of the choice of filters to be used in a multispectral measurement, the distribution of time among those acquisitions and also to make improvements in the system matrix conditioning through rational choices regarding the spatial distribution of the unknowns (i.e. voxels) within the solution space.

In the context of making these improvements, I note that it is relatively straight forward to demonstrate through numerical simulations or simple phantoms that image quality can be improved using either the approaches I am proposing or through advanced image reconstruction algorithms. However, there remains the potential that other sources of uncertainty (for example, in the precise locations of the animal's internal

organs or in the light propagation properties of these tissues) will continue to dominate, resulting in severe artifacts that may render the reconstructed images useless for many purposes. Therefore, some means of gauging the accuracy of the images is needed.

To date, attempts to validate BLT results have generally used reference standards that are either overly simplistic or imprecise [19, 20]. The photon transport properties used in digital and physical phantoms generally do not portray the complexity or uncertainty of the in vivo setting. Light sources embedded within animals typically involve simple point-like geometries whereas in preclinical practice, source geometries tend to be more distributed, and while phantoms having distributed sources can be devised, in these cases the photon transport properties of the phantom are both simplistic and unrealistically well known. More realistic in vivo experiments using positron emission tomography (PET) and single photon computed tomography (SPECT) image sets as surrogates for the bioluminescence distribution have been tried [19, 21] but in the manner in which this was done, the radionuclide distribution was not identical to the bioluminescence distribution.

A more accurate reference image can be created by taking advantage of the Cerenkov light produced by the positron emissions of some radionuclides. Through a model of the Cerenkov light production process that I have developed, I will show that it is possible to make use of a PET image to precisely predict the Cerenkov light distribution, thus allowing the validation of the accuracy of a given reconstructed BLT image acquired under realistic conditions.

In summary, within this dissertation, I will address both the noise and the validation issues that I believe are currently hampering the adoption of BLT in the preclinical research setting. Through an understanding of the noise properties of the image acquisition and reconstruction processes and by making reasonable estimates of the source signal intensities, I will show that it is possible to both estimate the overall magnitude of the noise in the reconstructed images as well as to optimize the choice of wavelengths and the distribution of imaging times so as to minimize this noise. Moreover, by devising a gold reference standard within a live animal based upon PET, Cerenkov radiation and an accurate means of coregistering the PET-based reference with the reconstructed image sets, it will be possible to validate the improvement in accuracy made possible by these and other manipulations.

1.2. The BLT inverse problem

The recovery of three-dimensional (3D) images of the bioluminescence source distribution can be mathematically posed as a linear inverse source problem. In this model of the system, a vector Y of the measurements of the light emanating from the skin surface are said to be produced by the product of the system matrix, W , and a vector of the unknown source intensities, X [22]. In this case, each element of W describes the probability that a photon emanating from a given source location, will be detected by a given detector element. As such, the columns of W describe the projection of each source location onto the detector as a whole. Conversely, the rows of W describe the relative contributions of all of the voxels within the domain of the animal's interior to a given detector element (generally a CCD pixel viewing a location on the skin surface of the animal). While some investigators make use of somewhat

different definition of the problem, in these cases they are generally seeking a somewhat different solution, for example Han et. al. [23] takes a different approach but they are seeking to solve for both the source distribution and optical tissue properties simultaneously. Most investigators in BLT source reconstruction make use of essentially the same linear system I've described above [24-28].

For typical samplings of the surface and interior spaces, the number of interior space unknowns greatly exceeds the number of surface measurement locations; therefore nominally the problem is severely ill-posed [1]. Because of the Lambertian-like nature of the surface flux, imaging from additional angles around the animal does not add significant additional information (except to the extent that they visualize additional skin surface). And owing to the high degree of scatter, particularly for deeply placed sources, the spatial frequency content of the surface images is low and therefore increasing the spatial sampling frequency of these images likewise does not lead to significant improvements in the posedness of the system.

1.3. Multispectral vs. hyperspectral

Increasing the spectral sampling of the light, however, does have the potential to add independent new information [22], this primarily because the attenuation differs as a function of the color of the light. For hemoglobin, the predominant chromophore in mammalian tissues, the attenuation decreases monotonically over the range between about 560 and 660 nm. Thus the deeper the source, the greater the red shift in the spectrum of the light measured at the surface.

With a sufficient number of spectral measurements the system, in principle, will have a unique solution for any given spatial sampling frequency of the interior. This can best be appreciated by considering a case where the light is constrained to transit along a single depth-dimension (or equivalently, when there is no scatter) but attenuation remains. The sum of the light along this single dimension measured at N different wavelengths would allow for the determination of the intensity of N source locations along that length, this because the attenuation factor profiles for the different wavelengths are linearly independent. Moreover, without scatter the position of the source in the dimensions perpendicular to the depth dimension would be readily discernible at the resolution of the surface measurements. When scatter is present, however, the surface profiles co-vary greatly, especially for deep sources. In summary, attenuation in effect improves the posedness of the problem, but high scatter causes it to remain ill-conditioned.

In practice, the number of spectral measurements made, for example, on an IVIS 200 bioluminescence imager is six, each using a 20 nm band-pass filter centered at 560, 580, 600, 620, 640 and 660 nm. For an object the size of a mouse with typical spatial sampling, this results in a system matrix that is roughly square. Increasing the number of wavelengths measured has the potential to improve the conditioning of the system matrix if the tissue attenuation of each new wavelength is sufficiently different from that of the others. In this case, each additional wavelength will add a new set of rows to the system matrix. This additional information has the potential to increase the independence of the columns within the system matrix.

However, the emission spectrum of any given luciferase has a limited range, so there is no point measuring wavelengths beyond this range. Moreover, for wavelengths within the near-infrared, attenuation tends to flatten out and become roughly constant in the tissues of a mouse. Thus, no new information is gained by subdividing this range into multiple measurements. Similarly, there is little point in making measurements at the short (i.e. blue-green) wavelength end of the spectrum owing to similar redundancies in attenuation but also perhaps more importantly because the attenuation is so high at these wavelengths that no signal is detectable, except perhaps for the shallowest of sources [29].

The only remaining way to increase the number of wavelengths measured is to divide the spectrum over the luciferase emission peak more finely. If measuring with a filter this means using a narrower band-pass and throwing away more photons, reducing the signal to noise ratio. Methods have been proposed to make spectral bioluminescence measurements without filters [30], but without this specialized hardware there exists a tradeoff between adding new information (which improves conditioning) and reducing signal to noise (which exacerbates the effect of ill-conditioning).

Relatively little work has been done to determine the optimal number of spectral measurements, the range of wavelengths, or the requisite relative exposure times for BLT image reconstruction. Taylor et al. in 2015 [31] sought to optimize acquisition time and wavelength selection based on simulations comparing three sets of wavelengths ([560,580,600], [600,620,640] and [560,580,600,620,640]) and just two bandwidths (10 and 20 nm). Only acquisition times equally distributed among these wavelengths were

considered. Their conclusion was that using the three longer wavelengths was as good as using all five and that a 20 nm bandwidth was best. However, members of this same group led by Dehghani and Styles had earlier published an information-theoretic method of selecting wavelengths [32] which concluded that the 580 nm measurement contained the most information and that when imaging with two wavelengths (note - larger numbers of wavelengths were not considered), the combination 570 and 580 nm was optimal. This earlier result did not consider the noise in the measurements, so perhaps this explains these contradictory results, however, this explanation was not offered by the authors.

Although I will not pursue it here, a similar tradeoff exists when it comes to choices regarding the spatial sampling frequency of the surface measurements. Increasing the sampling frequency has the potential to improve conditioning, but for CCD cameras having variable charge binning (like those used on most bioluminescence imagers), the loss in signal to noise is greater than from a simple dividing of the photon counts among additional bins [33].

Lacking an easy means of adding directly measured information to improve the conditioning (though some attempts have been made [34-36]), investigators have proposed various means of adding a priori information (or assumptions) in the form of constraints on the solution space. These constraints can come in the form of restrictions on the number of spatial locations at which a solution is sought or constraints on the source intensities, or on the relationships among the source intensities at the solution sites. These constraints come in a variety of forms including pre-conditioners,

regularization/penalty-functions and basis functions among others. Adding constraints limits the degree to which potential solutions can co-vary with one another. This improves robustness and decreases noise, effectively trading noise for bias.

The degree to which this works depends upon the accuracy of the assumptions employed. Some constraints are clearly applicable to all source distributions, for example, limiting source intensities to $R \geq 0$. Other constraints, however, are less generally applicable, for example, Wang et. al. [37] proposed the use of a permissible source region, which essentially assumed that at least the gross location of the source was known *a priori*. Feng et. al. [38] later proposed a means of optimally defining this region. A related constraint first applied in BLT by Lu et. al [39], uses a compressed sensing approach that makes use of an L1-norm penalty to bias the results towards sparse solutions. Cong et. al. [40] proposed placing a limit on the total number of bioluminescent sources. Numerous other variants on these proposals all in some way promote sparsity in their solutions [10, 27, 41-52].

A bias towards sparseness, however, can lead to overly sparse solutions and not all bioluminescence source distributions are in fact sparse. A relatively shallow tumor of a given size will produce a surface signal similar to that of a smaller (i.e. more sparse) deeper tumor. As can be appreciated by a careful review of the experiments by Lu et. al. [39], a bias towards sparseness will tend to select the latter solution (i.e. deeper sources). Experiments involving metastatic tumors or animals genetically engineered to express luciferase in specific tissues often will have distributed (i.e. non-sparse) source

distributions. For these situations, other types of constraints that don't promote or assume sparsity may be superior.

Penalties placed on the gradients of the solution do not promote sparsity but they do limit the resolution of the images. Moreover, these penalties tend to apply this smoothing uniformly without regard to the depth-dependent resolution inherent to BLT. Constraints in the form of basis functions proposed by some investigators [53, 54] likewise don't promote sparsity and have the potential to address the depth-dependent resolution issue but to my knowledge basis functions specifically chosen to deal with this depth-dependence have not yet been pursued.

The intent of the constraint methods described thus far has been to compensate for the ill-conditioning of the system matrix. This contrasts with rank reductions employed with the goal of speeding up the image reconstruction or making it more computationally manageable [55-58]. In general, rank reductions for speed up will tend to exacerbate ill-conditioning in that some information (ideally information of little importance) is being thrown away. An exception to this rule is encountered when the discarding of information simultaneously involves a reduction in the signal to noise ratio of the measurements sufficient to overcome the reduction in the conditioning. This can occur when reducing the spatial sampling frequency of the surface measurements or when increasing the band-pass of spectral measurements. Some investigations of the latter have been made by Taylor et. al. [31, 59] but without regard to the tradeoff in the conditioning.

1.4. Overall Goals and Specific Aims

The overall goal of this dissertation is to address two major factors that are limiting the acceptance of BLT by the cancer research community. First, I will develop a realistic, complex reference standard against which the accuracy of the BLT reconstruction can be judged. And second, I will implement algorithms that will guide the data acquisition in order to increase in the robustness and accuracy of BLT reconstructions. These goals will be accomplished by pursuing 3 specific aims:

Specific aim 1. Construct a multimodality registration system consisting of a bed capable of maintaining a mouse in a rigid pose under isoflurane anesthesia while it is transported between BLT, CT, PET and MR imaging systems, along with mechanisms to co-register the datasets acquired from each. In the context of this work I will also describe corrections for various confounds affecting the accuracy of the measurements.

Specific aim 2. Develop and validate a quantitative model of Cerenkov light production for ^{18}F and other beta emitting radionuclides. This model will allow conversion of activity concentration as measured by PET to photon flux as a function of wavelength, thus enabling a PET image to be used as a reference standard in assessing the accuracy of a luminescence tomography reconstruction. In the context of this work I will also describe improvements to the calibration of the IVIS imaging device.

Specific aim 3. Develop algorithms and procedures to improve the quality of the data used to reconstruct images. This will be achieved by optimizing

the data sampling in a manner which takes into consideration the strength and spectrum of the light source, the choice of filters and the noise in the camera system.

Further background on these efforts will be provided in later sections of this document.

2. SPECIFIC AIM 1: MULTIMODALITY REGISTRATION

2.1. *Overview*

The overall objective of the first specific aim is to co-register the datasets acquired on optical (e.g. IVIS 200), PET (e.g. Siemens Focus 120), CT (e.g. microCAT) and MR (e.g. Bruker) scanners based on a calibrated positioning of the animal within each scanner's field of view. Accurate co-registration is a pre-requisite enabling the comparison of BLT reconstruction results to complex reference standards against which the accuracy of the BLT reconstruction can be judged.

In vivo planar optical bioluminescence imaging (BLI) of small animals provides a high sensitivity, low background, non-invasive means of monitoring gene and protein expression and other cellular events at low cost [60-65]. However, the information that BLI provides is severely limited in terms of its ability to determine either the concentration or precise location of the bioluminescence source. These limits stem from the manner in which light propagates through biological tissues [66]. Unlike the high energy X-ray and gamma-ray photons used in radiographic and nuclear imaging, photons at the wavelengths typical in BLI (400-800nm) do not predominantly travel in a straight line from their source to the detector. Instead, bioluminescent light is highly scattered and attenuated, processes that obscure and dissemble the location and intensity of the true source distribution. The region on the skin surface of the animal from which light is seen to emanate (in rough terms) is the surface point closest to light source and the magnitude of the light flux at the surface is heavily dependent upon this distance.

Bioluminescence tomography (BLT) has the potential to remove these limitations, providing both quantitative accuracy and information regarding the precise location and 3D distribution of the bioluminescence sources [67]. Recovering this information from the surface flux measurements however is difficult with results that are sensitive to the chosen light propagation model and the assigned tissue parameters [68]. Although it is known that the organs and tissues within an animal vary considerably in their light attenuating and scattering properties, BLT reconstruction algorithms often assume homogeneous tissue having composite attenuation and scatter parameters, this largely because knowledge of the internal anatomy is not available.

Precise knowledge of the shapes and locations of the major organs within an animal therefore has the potential to significantly improve the accuracy of BLT reconstructions. This knowledge could be garnered from, for example, magnetic resonance (MR) and/or X-ray computed tomography (CT) scans that have been spatially registered to the bioluminescence images. These anatomical datasets could be segmented according to tissue type and to each a different set of light propagation properties assigned.

To my knowledge, there have been just two previously published attempts to apply information regarding organ shapes and locations to assist in modeling the propagation of light through the tissues of a live mouse. The first of these estimated the shapes and positions of the major organs of the mouse using a generic segmented digital mouse atlas. This model was rotated, shifted, scaled and warped so that its exterior surface contours matched those of a CT taken of the animal after it had been frozen with liquid nitrogen in the pose it was in within the BLI imager [37]. The use of a generic mouse

atlas to estimate the mouse anatomy does not allow for abnormal anatomy (e.g. tumors) within the mouse. This particular deficiency was addressed in a study by another investigator, which applied similar methodology but this time to an MR image of the same animal from which the BLI images were obtained [69]. In spite of attempts to maintain the animal's pose, it was again necessary to spatially warp the 3D dataset to get the MRI surface contours (in this case) to match those of a surface determined using photogrammetric techniques within the BLI imager. Warping in this manner is problematic because there are no measurements to guide the internal deformations of the organs and thus can lead to significant errors in the light propagation estimate. Furthermore, the accuracy of this type of retrospective fitting procedure is data dependent, having potentially large errors when the contours are smooth.

In the approach I propose here, the animal is maintained in a fixed rigid pose across imaging sessions and thus I avoid the need for warping transforms. By using specialized hardware that allows precise positioning of the animal within each of the scanners, it is possible to use fixed a priori determined spatial transformations to register the image information among all modalities. The registration of the 3D data (CT and MR) to the 2D optical images is accomplished using a projective transformation that models the relative position, focal length, and field of view of the camera within the BLI system. Corrections are also made for the spatial distortions introduced by the BLI camera lens. This model of the BLI camera system can be used to transfer information in both directions between the anatomical and optical imaging spaces. For example, it can be used to map the BLI image data onto a skin surface determined from the 3D

anatomical data. Likewise, the skin or other surfaces can be mapped to a 2D image onto which the bioluminescence light signal can be superimposed.

Registration of the image spaces at this stage prior to BLT reconstruction allows for the use of information derived from the anatomical datasets regarding the location and spatial distribution of various organs to be used within the BLT reconstruction algorithm. BLT reconstructions based on this mapping are effectively pre-registered to the anatomical data. This registration provides important anatomical context to assist in the interpretation of the reconstructed luminescent distribution. Perhaps more importantly, given the questionable accuracy of current BLT reconstruction algorithms in vivo, by using sources visible on both the optical and anatomical modalities, the MR (or CT) determined source distributions can be used as a gold standard against which the results of the BLT reconstructions can be assessed and validated.

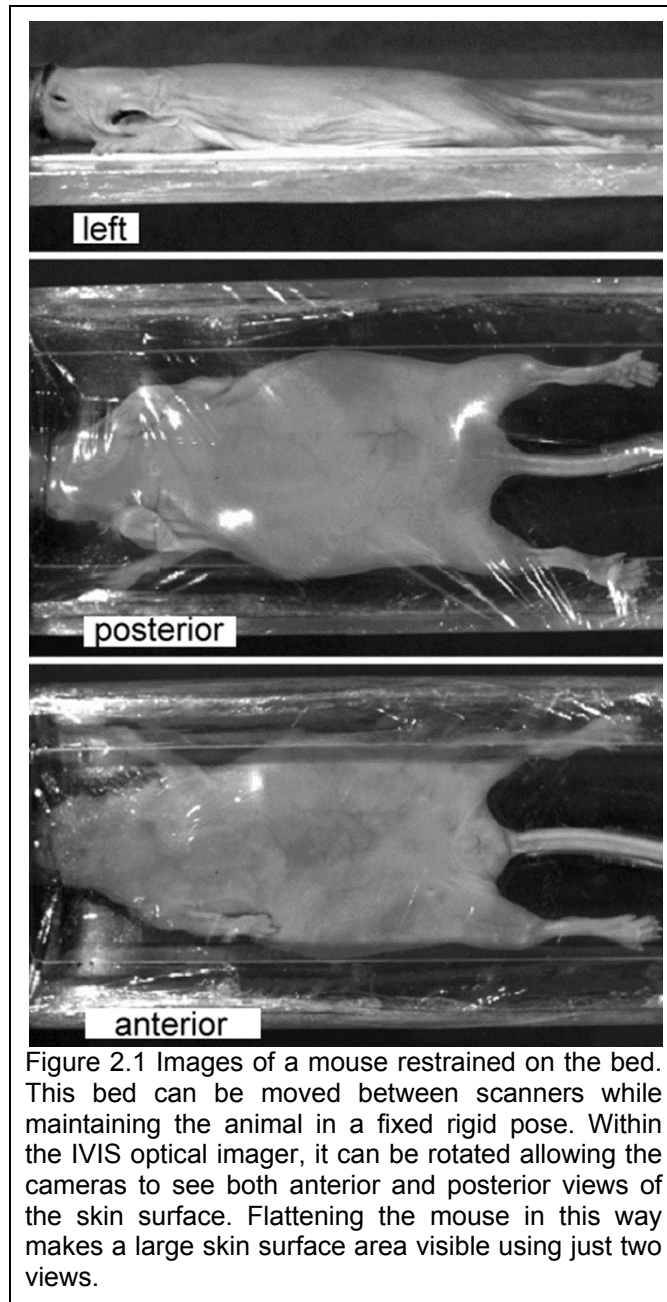
In this manuscript, I'll describe procedures to register MR and CT image sets of a mouse to a set of optical bioluminescence planar images, allowing each animal's own anatomy to define the spatial distribution of the attenuation and scatter parameters. By placing artificial light sources of known intensity within the animal that can be readily detected via CT and by using transgenic animals genetically engineered to have specific organs (visible on MR) express luciferase, I'll demonstrate a means by which the accuracy of a given BLT reconstruction can be assessed. In addition, by rotating a mouse while maintaining its fixed pose within the BLI imager, I am able to accurately determine the dependence of the measured light intensity on the angle of the surface normal relative to the BLI camera.

2.2. Scanners.

Brief descriptions of the three imaging systems used for the studies described in this manuscript are as follows. The IVIS 200 is a bioluminescence and fluorescence imaging system utilizing a 26x26 mm back-thinned, back-illuminated CCD, cryogenically cooled to -105° C. It has an adjustable field of view ranging from 4 to 26 cm and includes a light source and filter sets for fluorescence and multispectral bioluminescence imaging. The Bruker Biospec 47/40 (Bruker Biospin Inc., Karlsruhe, Germany) is a 4.7 Tesla 40 cm horizontal bore small animal imaging spectrometer equipped for multinuclear imaging studies and spectroscopy. The Siemens/CTI microCAT II (Siemens Medical Solutions, Malvern, PA) is a small animal CT scanner with an 8.5 cm axial by 5.0 cm transaxial FOV. It uses a 2048×3096 element CCD array coupled to a high-resolution phosphor screen via a fiber-optic taper and a Tungsten anode with a 6 micron focal spot. Its highest reconstructed resolution is about 15 microns in each dimension.

2.3. Overview of the registration procedure.

The overall objective here is to base the registrations on a calibrated positioning of the animal within each scanner's field of view. Between and during the imaging sessions, the animal is held in a rigid pose, at a fixed position relative to the animal bed. This is accomplished by wrapping the animal with a thin 0.01 mm polyethylene wrap while it is positioned atop a custom designed bed with a nose cone for the administration of oxygen and gaseous anesthesia. The wrap applies a light pressure over the entire body of the animal, gently and efficiently restricting its movement. Registration then amounts



to establishing a frame of reference relative to the bed for each scanner and calculating the rigid or projective transforms that map between them.

In these studies, I have used several different bed designs and many more are possible. Here I'll briefly describe one such bed that is particularly apt for use in BLT reconstruction and is the one used in the animal experiments described below. The bed is fashioned from a 6 x 25 cm rectangular sheet of 1 cm thick Lucite at the center of which is cut a 4 x 15 cm rectangular window. Over this window is stretched a single layer of 0.01 mm polyethylene plastic, adhering to the Lucite with the assistance of a restickable glue (3M

Glue Stick). This sheet of plastic forms the bed on top of which the animal is laid. The animal is then sandwiched and pressed by a second layer of polyethylene, effectively restraining its movement to less than 0.62 mm [70] and allowing equally clear views of the animal from above and below (see figure 2.1) as it is suspended above the window. Squeezing the animal in this manner has had no apparent adverse affect on the

animal's health in the dozens of studies conducted to date. At one end of the Lucite is attached a block of Delrin plastic into which are drilled a set of holes sized and spaced so as to mate with a corresponding set of pegs present on the bed mount adapters designed for each of the imaging modalities.

For the IVIS, the bed mount includes a platform referencing two of the inside edges of the IVIS' light-tight box. Thus, the bed mount and the attached bed can be consistently placed within the IVIS, thereby allowing precisely reproducible positioning of the animal relative to the camera for any given camera to subject distance. The bed and its mounting system were designed such that the bed can be pivoted about its long axis (inferior to superior axis of the mouse) in precisely calibrated 15° increments, allowing views of the animal from different vantage points.

The microCAT has a motorized bed positioning mechanism with optically encoded position readout calibrated to a precision of 0.01 mm and a repositioning accuracy of better than 0.1 mm. A custom adapter is used to attach the animal bed to this bed positioning mechanism in a reproducible manner. It can then be removed for placement on the other scanners using specialized bed mounts designed for each. The coordinate system defined by the microCAT's bed positioning mechanism was used as the reference frame to which both the Bruker and IVIS images are mapped.

Positioning of an animal within the field of view of the Bruker does not easily lend itself to such reproducibility because its field of view is located deep within its bore and thus is remote from any potential spatial reference. Moreover, references within the bore are generally blocked by the gradient and readout coils. Therefore, I established a set of

markers within the bed that are visible both on MR and CT. Using landmarks derived from these markers it is possible to place the MR image set into the microCAT's frame of reference. Alternatively, retrospective mutual information based volume registration methods work well when registering these two structural image datasets to one another. For a detailed description of the markers, the volume and landmark point based registration procedures and the effectiveness of the wrapping system in maintaining the rigidity of the animal, see Beattie, et al. [70].

2.4. Registration of a 2D image to a 3D image set.

The conventional notion of what it means to register two three-dimensional (3D) image sets is to rotate and shift a target image set, so that its resampled voxels are in locations equivalent to those of the corresponding voxels within the reference image set. For the purposes of this manuscript, the idea is to co-register a 3D image set with an image that has only two dimensions. Furthermore, this two dimensional (2D) image is generated from the summation of photons traveling along vectors entering the camera and thus its pixels do not correspond to points in 3D space (as opposed to the pixels of a 2D slice through 3D space). In this case, the conventional notion of 3D image registration is ill applied, so instead I'll make use of a paradigm more apt for the registration of 2D photographic images.

In this paradigm, two 2D images are co-registered when through a series of transformations applied to the target image, the position, orientation, focal length and distortions of the reference image's camera system are mimicked. In this way, the vectors associated with each pixel in the two images are made to overlay. Thus, for

these purposes, I will create a virtual camera capable of taking 2D images of the 3D image set information content. This virtual camera is simulated to have the same focal length and distortions and to be in the same position and orientation relative to the imaged object as the real 2D camera that acquired the reference (in this case bioluminescence) images. This virtual camera system can be made to visualize the 3D image set in a variety of ways, for example, it can slice through the 3D image set at an arbitrary depth and angle; or it can view maximum intensity projection information; or it can view the reflectance of virtual light sources off surfaces that have been segmented from the 3D data.

2.5. BLI camera model.

The camera model used was that of a basic pinhole camera as described by Hartley and Zisserman [71]. In this model, points in 3D space represented in homogenous coordinates $(X,Y,Z,T)^T$ are mapped onto the 2D image plane by the 3×4 projective transformation matrix which is decomposed and parameterized as follows:

$$\begin{bmatrix} f & 0 & p_u \\ 0 & f & p_v \\ 0 & 0 & 1 \end{bmatrix} \cdot [R_{xyz}] \cdot \begin{bmatrix} 1 & 0 & 0 & c_x \\ 0 & 1 & 0 & c_y \\ 0 & 0 & 1 & c_z \end{bmatrix} \cdot \begin{bmatrix} \cos \beta & 0 & -\sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot [Q_{xyz}] \cdot \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

Here, R_{xyz} and Q_{xyz} are rotation matrices having three parameters each. Point (p_u, p_v) is the center of the acquired 2D image, (c_x, c_y, c_z) is the camera center, and β describes the rotation of the bed about its axis. Vector $(t_x, 0, t_z)$ defines the translation which when combined with Q_{xyz} moves the bed from its position in the CT or MR coordinate system

onto the axis of the bed-mount in the BLI coordinate system. Altogether, this system requires 15 parameters (f , p_x , p_y , 3 for R , c_x , c_y , c_z , β , 3 for Q , t_x and t_z), three of which were fixed (p_x , p_y and the rotation angle β) leaving 12 parameters to be fit during the calibration procedure.

Distortion within the IVIS camera images was modeled using the radial distortion model described by Hartley and Zisserman [71]. In this model, the distortion is assumed to be solely a function of the radial distance from some central point and is estimated by a Taylor expansion, $L(r)=1+\kappa_1r+\kappa_2r^2+\kappa_3r^3+\dots$; with $r^2=(x-x_c)^2+(y-y_c)^2$ and where (x_c, y_c) is the central point. In this implementation, I have assumed that this central point corresponds to the principal point and image center (p_x, p_y) . An image of a grid was used to calculate three terms of the Taylor expansion of the radial distortion function by minimizing the distance between the gridline intersections and corresponding virtual lines formed by end points of each gridline on the image periphery (as suggested by Hartley and Zisserman).

2.6. BLI camera calibration.

In order to cross-calibrate the IVIS and microCAT coordinate systems, a phantom for which corresponding points can be identified on both imaging systems was devised. Specifically, this phantom was made out of a 3x2x10 cm plastic block into which a grid of 1 mm wide by 1 mm deep grooves were cut, spaced 5 mm apart. The grooves themselves were painted white, while the tile surfaces of the block were painted black. Grids were cut into all six surfaces of the block, although only three surfaces were used in the calibration procedures described here.

The corners of the tiles are readily identified on both the reconstructed CT image sets and in the reflectance images from the IVIS. Within the microCAT image sets, the 3D coordinates of twenty-eight tile corners were manually identified. These points were arranged in grids of 3×4 points covering the top and the two adjacent long sides of the phantom. Note the grids all extend to the edges of the phantom, therefore the top shares its left-most and right-most columns of points (4 points each) with each of the respective sides (thus $3 \times 3 \times 4 - 4 - 4 = 28$ points altogether).

These same tile corners were identified (again manually) in each of the 13 IVIS distortion corrected reflectance images in which the corners could be seen. Thus at -90° , 0° and $+90^\circ$, a single face and therefore 12 points were in view and at each of the other 10 angles, two faces and therefore 20 points were in view. All combined, 236 points within the 13 IVIS images were located. These same locations were modeled based on the 28 microCAT points and the known bed rotation angles. The 12 variable model parameters were adjusted to achieve a least squared error between the measured and modeled point locations using a constrained nonlinear fitting procedure (lsqnonlin in Matlab, The Mathworks Inc, Natick, MA).

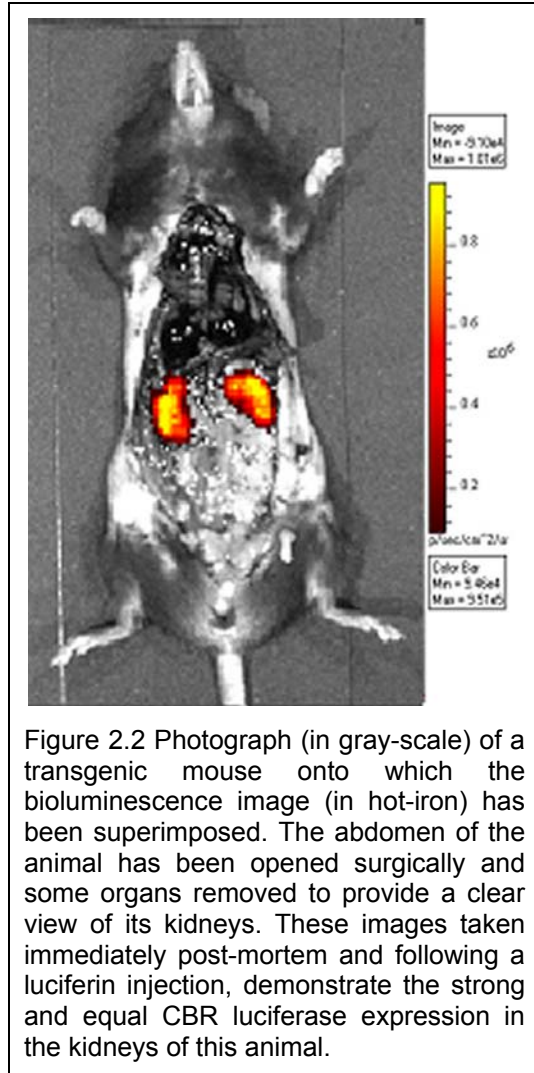
2.7. Registration accuracy.

The accuracy of the registration was estimated by performing a number of repeat studies involving a mouse-like phantom. The phantom was mouse-like in terms of its size, weight and rough shape, so that the forces applied to the enclosing plastic wrap and bed support structure would be similar to those encountered with an actual animal. On the surface of this mouse phantom was glued four gaseous tritium light source

(GTLS) beads [mb-microtec ag, Niederwangen, Switzerland], two on each of what were effectively the anterior and posterior surfaces. These small (2.3×0.9 mm) cylindrical glass tubes emit a small, virtually constant (tritium powered fluorescence) level of light and are readily distinguishable on both the CT and bioluminescence images.

Between each of the repeat studies, the bed was removed from its mount and the mounting platform removed from the IVIS, thus the measured accuracy takes into consideration the reproducibility of these bed positioning procedures. Note, errors due to the movement of the wrapped mouse were considered in my previously published work [70]. The bed repositioning was performed three times each time with images taken of the bed rotated at angles covering a full 360 degrees at 30 degree increments. Following each bioluminescence imaging session, the GTLS phantom was imaged on the CT each of which also involved the removal of the bed and its mount from the CT.

On each of the three CT datasets, the centers of the four GTLS beads were manually identified (i.e. with a computer cursor). To these I applied the perspective and distortion transforms calculated in the calibration procedures generating a set of 2D locations within the bioluminescence image space. The corresponding locations as seen on the bioluminescence images were also identified manually and the absolute distance between corresponding transformed CT and bioluminescence point pairs determined. This was done for the 9 combinations of the bioluminescence and CT repositioning studies, each involving 12 pairings (one for each angle) of each of the two GTLS beads visible at a given angle.



The mean and standard deviations and max of the errors were calculated for each bead location at each angle, so that unusually large errors for a given bead (i.e. location within the image) or for a given angle could be identified. Failing the identification of any outliers, the results were summarized by a single mean, standard deviation and overall max error.

2.8. Animals and imaging procedures.

Numerous animal studies have been undertaken utilizing these registration procedures. Here I will describe three preparatory studies (described here as experiments 1, 2 and 3) whose results have general application and implications across

all BLT reconstruction algorithms. In the first of these, experiment 1, I'll measure the dependence of the measured surface flux on the angle of the skin surface relative to the camera. In experiment 2, I'll provide direct evidence of the impact of inhomogeneous light propagation within the tissues of the mouse and in experiment 3, I'll demonstrate a method of correcting for the time dependent changes in total light flux seen in typical luciferase based bioluminescence imaging studies due to substrate transport and consumption. Accounting for this time-course is important in multispectral and multiview BLT studies [22, 72-74], which involve multiple sequential images.

In experiments 1 and 2, the light source was a GTLS bead placed within a small catheter that was in turn placed within the rectum of a nude mouse (nu/nu). Prior to this placement, anesthesia was induced with 3% isoflurane and the eyes of the mouse were dabbed with a sterile ocular lubricant (Pharmaderm - Paralube Vet Ointment) to prevent drying. The mouse was then placed on the bed and secured with the plastic wrap following which it was imaged within the IVIS imager and on the microCAT CT scanner. The details of each of these imaging sessions are provided below. Continuously throughout and between all imaging studies the mouse was maintained under anesthesia using 1% isoflurane, with only momentary disconnects to allow transport between imaging systems.

In experiment 3, I made use of a transgenic mouse that was genetically engineered such that both of its kidneys uniformly expressed click-beetle red luciferase (see figure 2.2). It is worth noting that (like a Cerenkov/PET reference standard) animals of this type can also be an effective means of testing the accuracy of BLT reconstruction algorithms for distributed (i.e. non-point-like) sources in-vivo, because the source distribution is more predictable across animals (compared to implanted, luciferase expressing tumors, for example) and because the organs expressing the luciferase are readily seen on MR. The use of this animal for the purposes here, however, is to demonstrate a means to correct for the time-course of bioluminescence light output following the luciferin injection.

Unlike the nude mice used in the first two experiments, the transgenic mouse has dark brown fur that can interfere with the measurement of the bioluminescence signal. To

avoid this interference, the abdomen of the animal was depilated prior to imaging. This animal received a bioluminescence image set followed by scans on the microCAT CT and Bruker MR. Details of the imaging protocols used on this animal are described below.

2.9. Imaging protocols.

Imaging on the IVIS varied somewhat with the experiment. For the angular dependence measurement, both reflectance and bioluminescence mode images were acquired for each angular position of the bed as it was rotated in 15 degree increments between \pm 90 degrees. All bioluminescence images were acquired in the “open” filter setting (i.e. with no filter present).

In experiment 2, demonstrating the affect of tissue heterogeneity, the animal was imaged from above in a prone position. Several attempts were made with slight adjustments to the position of the GTLS bead until the bioluminescence image achieved a bimodal surface flux suggesting preferential light pathways to either side of the spine. Upon achieving this position, the animal was imaged using the full set of 20 nm bandpass filters available on the IVIS 200 covering the range from 560 to 660 nm.

For experiment 3, imagesets of the mouse were taken from both the anterior and posterior views. Each imageset consisted of a reflectance image followed by the full set of 20 nm bandpass filter images. Bracketing and interposed between each of these, a short (10 sec) “open” filter setting image was acquired. This entire imaging sequence commenced two minutes following an intraperitoneal injection of luciferin (150 mg/kg in 100 μ L).

For the MicroCAT imaging, three hundred and sixty transmission images were acquired at 1° increments encircling the mouse. These images were reconstructed with a cone-beam 3D FBP algorithm (COBRA software from the Exxim Computing Corp. Pleasanton, CA) into a 192×192×384 matrix over a 4.38×4.38×8.76 cm FOV (i.e. 0.228×0.228×0.228 mm voxels).

Images acquired on the Bruker MR were made using a 7 cm Bruker birdcage coil tuned to 200.1 MHz and the 10 mT/m gradient coil system. 3D images were obtained with a fast spin-echo sequence with a repetition interval (TR)= 1.2s, effective echo time (TE)= 40ms, image matrix of 128×96×256, 8 repetitions per phase encoding step and a total imaging time of 61 minutes. The final voxel dimensions for these images are 0.341×0.333×0.333 mm.

2.10. Registration accuracy.

The results of the test of the BLI to CT registration accuracy showed no bias in the error, neither for the angle of rotation nor for the bead location (see Table 2.1). The mean error across all beads was 0.36 mm with a standard deviation of 0.23 mm. The maximum error encountered over all measurements was 1.08 mm.

Table 2.1. The mean error in the BLI to CT registration for individual beads as the bed is rotated.

angle (degrees)	-165	-135	-105	-75	-45	-15	15	45	75	105	135	165
bead 1a or 1b avg (mm)	0.56	0.17	0.35	0.40	0.31	0.52	0.25	0.29	0.25	0.24	0.12	0.24
bead 2a or 2b avg (mm)	0.44	0.33	0.53	0.44	0.55	0.31	0.39	0.40	0.36	0.38	0.48	0.42

2.11. Experiments

Experiment 1 - Angular dependence. By the time a given photon reaches the inner surface of an animal's skin, it has usually undergone numerous scattering events such that virtually all information regarding the direction of its source has been lost. Moreover, for this reason, photons impinge on the inner surface nearly isotropically. However, because of the change in refractive index when moving from skin to air, the exit angle is not isotropic and thus the apparent intensity of the light emanating from a given surface point is dependent upon the angle between the skin surface normal and the camera line of sight. For BLT this dependence needs to be accounted for or corrected when determining the surface flux at each surface point.

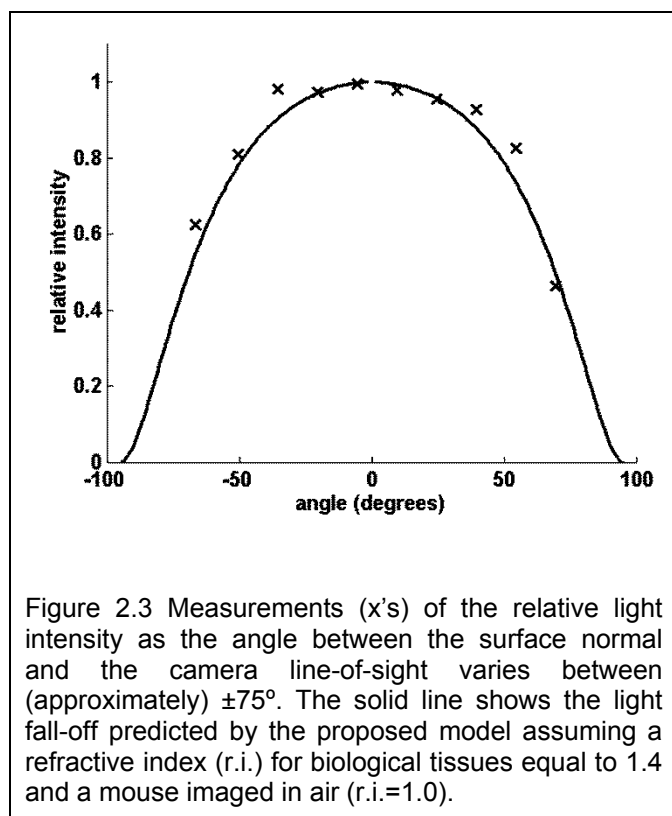


Figure 2.3 shows the measurements made in experiment 1 of relative light intensity as a function of the exit angle fitted with a curve modeled based upon the Snell and Fresnel equations and assuming a refractive index of 1.4 (the value measured in mammalian tissues by Bolin, et al. [75]). The derivation of this model will be described in section 2.15 of this manuscript. In measuring the angular dependence, I chose a set of surface points based upon a

threshold applied to the bioluminescence image taken at bead-angle zero. The surface normal for these points was determined from the CT image and the angle relative to the camera was garnered from the rotations to be applied to the CT data in registering the bioluminescence and CT data sets. Given the somewhat flattened body contour of the mouse (see figure 2.4a) the selected surface point normal vectors were all within 5 degrees of one another and therefore considered to correspond to a single mean angle (note this mean angle was not zero for the horizontal bed position since the surface of the mouse was at a slight angle relative to the bed). This same set of surface points were followed as the bed was rotated to different positions between $\pm 75^\circ$ in 15° increments. The intensities of the bioluminescence light at these surface points were

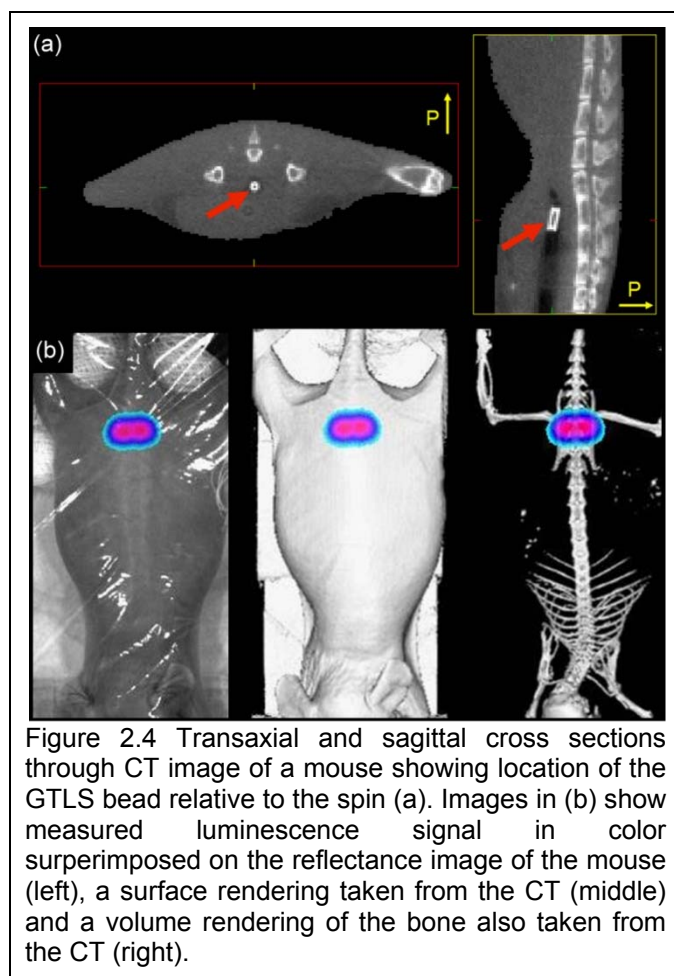


Figure 2.4 Transaxial and sagittal cross sections through CT image of a mouse showing location of the GTLS bead relative to the spine (a). Images in (b) show measured luminescence signal in color superimposed on the reflectance image of the mouse (left), a surface rendering taken from the CT (middle) and a volume rendering of the bone also taken from the CT (right).

averaged at each angle and associated with the bed angle plus the mouse-to-bed angular offset. The intensities were then normalized to have unit maximum amplitude.

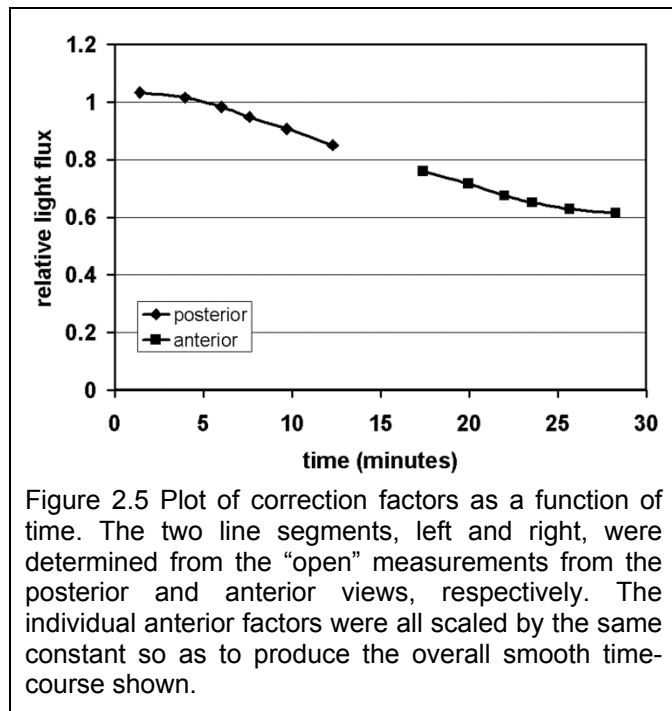
Experiment 2 – tissue heterogeneity.

Figure 2.4 shows a set of images taken in experiment 2. Based on the skin surface contour as seen on the CT (in figure 2.4a) one would not expect the bimodal surface flux seen in the bioluminescence images (figure 2.4b) if the underlying tissue was

homogeneous in its light propagation properties. The CT image shows that the GTLS bead is positioned directly beneath the spine in this animal suggesting that increased attenuation through the bone may explain the bimodal distribution.

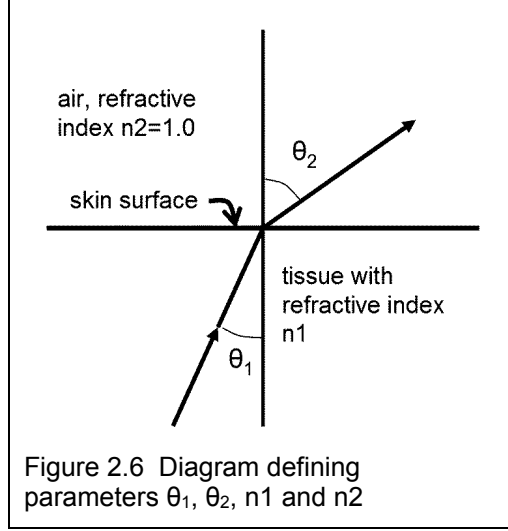
Experiment 3 – time course correction. As described in the Materials and Methods section, the in vivo multispectral bioluminescence imaging protocol includes “open” (i.e. unfiltered) acquisitions bracketing each of the images acquired with one of the 20 nm bandpass filters. The intent of these repeat measures is to monitor the time-course of luciferase enzyme-substrate activity (and perhaps other physiological changes) leading to changes in the measured surface flux. Note that here I am assuming there are no

changes in the spatial distribution of the enzyme or substrate nor changes affecting the spectrum of the light emissions (e.g. temperature induced red-shift [76]).



The procedure to correct for the enzyme-activity time-course is as follows. To each of the “open” images the same region of interest enveloping the bulk of the light emanating from the mouse was applied and the time of acquisition and total light flux (in photons per second per steradian) was recorded. The unfiltered light flux at the time of the filtered image acquisition

was estimated by linear interpolation of the bracketing unfiltered light flux measurements. The time-course correction factor for each filtered image taken from a given viewpoint (anterior or posterior) is simply the ratio of the interpolated flux relative to flux of the first open image from that viewpoint. The change in the time-course between views was determined by extrapolating the correction factors from the first view to the time of the first open acquisition of the second view. All of the second viewpoint correction factors are then scaled by this extrapolated factor. A plot of the resulting correction factors is shown in figure 2.5.



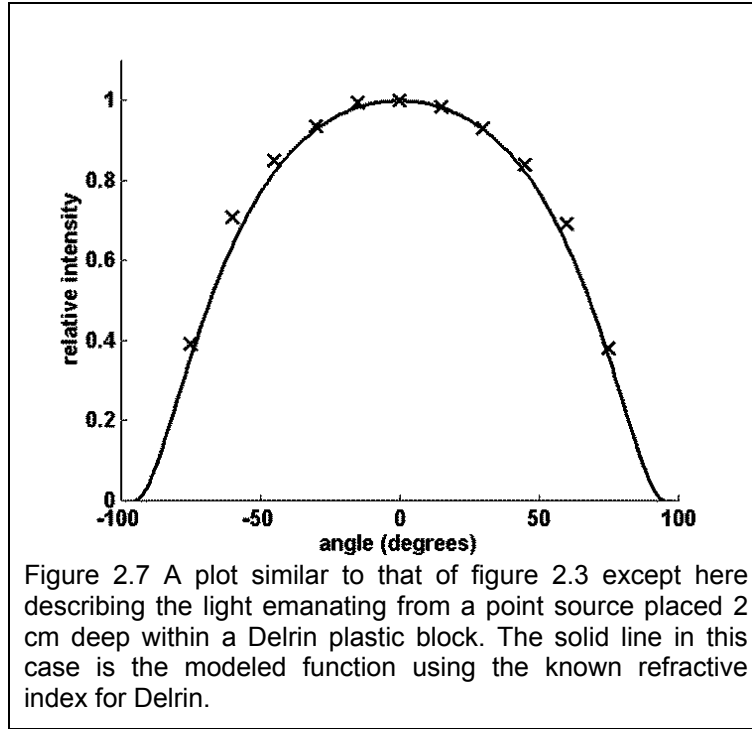
2.12. Light Fall-off Correction.

The model I propose for describing the fall-off in light intensity as the angle between the camera line-of-sight and the surface normal increases is derived from Snell's law and the Fresnel equations and assumes that photons just prior to exiting the animal are isotropic. Thus, the incident angle θ_1

(see figure 2.6) is uniformly distributed over $\pm 90^\circ$. When these photons are moving from the animal (with refractive index n_1) into air (with refractive index $n_2 < n_1$), if they are incident at an angle θ_1 greater than $\theta_{crit} = \sin^{-1}(n_2/n_1)$ then they are internally reflected. Whereas, if they are incident at an angle less than θ_{crit} their exit angles θ_2 are distributed over the range $\pm 90^\circ$. This distribution however is not uniform. Instead, for each arbitrarily small solid angle $d\theta_1$, there is a corresponding (larger) solid angle $d\theta_2$ into which the photons are distributed. The ratio of these solid angles $d\theta_1/d\theta_2$ determines the reduction in light flux and can easily be calculated by solving Snell's law of refraction formula (equation 1) for θ_1 and taking its derivative with respect to θ_2 (result shown in equation 2.2).

$$n_1 \cdot \sin(\theta_1) = n_2 \cdot \sin(\theta_2) \quad (2.2)$$

$$\frac{d\theta_1}{d\theta_2} = \frac{n_2 \cdot \cos(\theta_2)}{n_1 \cdot \sqrt{1 - \frac{n_2^2}{n_1^2} \sin^2(\theta_2)}} \quad (2.3)$$



This relationship is modified slightly by the partial reflections occurring for incident angles less than θ_{crit} described by the Fresnel equations. The Fresnel equation describing the fraction of light transmitted as a function of the incident (or exit) angle is shown in equation 2.3. The complete description of the angular

distribution of the exiting photons for isotropic incident photons is the product of equations 2.3 and 2.4, $T d\theta_1/d\theta_2$.

$$T = 1 - \frac{\left(\frac{\sin(\theta_2 - \theta_1)}{\sin(\theta_2 + \theta_1)} \right)^2 + \left(\frac{\tan(\theta_2 - \theta_1)}{\tan(\theta_2 + \theta_1)} \right)^2}{2} \quad (2.4)$$

This model was tested on a phantom consisting of a large Delrin plastic block 10×10×4 cm. In the center of one of the 10×10 cm sides was drilled a 2 cm deep cylindrical hole with a diameter of 0.5 cm. This in turn was filled by a snugly fitting cylindrical peg, also made of Delrin. Into the tip of the peg, a small hole was excavated, just large enough to accommodate a GTLS bead. The GTLS bead, so placed, was positioned in the center of the large Delrin block. The block, in turn, was placed on the bed mount within the IVIS imager and luminescent images were acquired with the block rotated at a series of angles between $\pm 75^\circ$ (at 15° increments). Delrin is known to have a refractive index of

about 1.48 [77] and this value worked well when fitting the model to the averaged surface flux (see figure 2.7).

2.13. Discussion

The majority of bioluminescence tomography reconstruction algorithms, when tested in vivo, lack a gold-standard reference describing the true light-source distribution. This lack of in vivo testing and validation has hampered both the continued refinement and the acceptance of BLT for routine use. The registration procedures I propose enable the development of a gold-standard reference to which the reconstructed luminescence source distribution can be compared.

Although it is yet an open question, further improvements in the accuracy and robustness of BLT reconstructions may require the incorporation of additional a priori information. In particular, the spatial distributions of tissues having differing light propagation properties may have significant impact. CT/MR to BLI registration allows a mechanism via which this type of information may be exploited in BLT. By using the animal's own MR or CT, even abnormal anatomies may be handled.

Lastly, I have demonstrated corrections for two confounds that play a role in many BLT acquisitions, the changes in bioluminescence flux as a function of time and as a function of angle between the surface normal and camera line-of-sight. In correcting for this latter confound I propose a model that relates the distribution of light propagation vectors for photons impinging on the inner surface of the skin to those exiting the skin surface. This model was tested using careful measurements made possible by the described registration procedures.

It is my hope and expectation that taken together these pieces form a platform upon which bioluminescence tomography reconstruction algorithms may be improved and refined and ultimately validated, paving the way for routine preclinical use.

3. SPECIFIC AIM 2: CERENKOV QUANTITATION

3.1. *Overview*

The reference standard images I propose to use are PET images that have been reconstructed within the same spatial frame of reference within which a luminescence dataset is acquired [78]. The luminescence data will come in the form of Cerenkov radiation, which will act as a surrogate for bioluminescence. Cerenkov radiation is produced by and is virtually 100% correlated with the locations of the positrons whose distribution is described by the PET image. PET images describe positron concentration and the intensity of the Cerenkov light produced by the positrons can be calculated directly from this information based on first principles [79]. Thus, with the appropriate scaling, PET images can provide ground truth against which a BLT algorithm's reconstruction can be tested.

Cerenkov radiation (CR), first described by Pavel Cerenkov nearly a century ago, is produced when a charged particle travels through a dielectric medium at a speed greater than the phase velocity of light in that medium (i.e. greater than the speed of light in a vacuum divided by the refractive index of the medium) [80, 81]. These conditions produce a photonic shockwave somewhat similar to the sonic shockwave (i.e. sonic boom) associated with supersonic bodies in air. Cerenkov photons have a broad frequency spectrum with intensity inversely proportional to the square of the photon's wavelength within and extending somewhat beyond the visible range.

Recent renewed interest in CR began following the demonstration of detectable amounts of light emanating from a radionuclide bearing live mouse [82, 83], suggesting the possibility of exploiting this phenomenon for medical research and possibly clinical purposes. In this context, a number of radionuclides have been tested for CR production (e.g. F-18, N-13, Cu-64, Zr-89, I-124, Lu-177, Y-90, I-131) [84, 85] including some radionuclides, In-111 and Ac-225, that one might not, upon initial consideration, expect to produce CR owing to their lack of a sufficiently high velocity charged particle emission. In-111 decays via electron capture and emits only γ -rays with significant abundance. Ac-225 is a virtually pure α emitter, but α 's in water become superluminal only at energies well beyond those of Ac-225's emissions. Never-the-less, experiments designed to measure CR conducted by multiple groups have detected light emanating from both In-111 and Ac-225 [84, 85]. However, to-date, clear evidence demonstrating that the Cerenkov mechanism is the source of this light has been lacking.

Of the potential biomedical uses of CR, the most commonly cited application is as a low cost, high throughput alternative to PET imaging [82, 83, 86] referred to as Cerenkov Luminescence Imaging (CLI). Other proposed applications include: an alternative to bremsstrahlung for imaging pure β^- emitting radionuclides [82, 86]; a higher resolution autoradiography method for high energy β 's [86]; intra-operative or endoscopic imaging of targeted structures in humans [85]; an excitation source for various fluorophores [87-89]; and most recently a renewed interest in using CR as a light source for photodynamic therapy [90, 91]. In each of these applications there are also disadvantages to using a Cerenkov derived signal (e.g. limited half-life, ionizing radiation, poor tissue penetration). As such, it is yet unclear whether any of these new

applications of Cerenkov imaging will prove to be clearly superior to extant techniques and enjoy widespread use.

However, one concrete and clearly advantageous proposed use of CR is as a means of validating the results of luminescence tomography reconstruction algorithms [82]. Thus far, manuscripts have been published using both SPECT [20] and PET [92, 93] imaging as validated reference standards. In these papers, the comparison of the reconstructed luminescence to the nuclear imaging reference was limited to a simple difference between centroid locations. One of these papers [93] looked at the linearity between the two signal intensities but did not establish a relationship in absolute terms that spanned the in vitro and in vivo conditions.

The work to be presented here seeks to establish such a cross-calibration between the signals derived from CR and nuclear tomographic imaging modalities, thus allowing nuclear imaging to better serve as a means of validating luminescence tomography reconstruction algorithms. Since PET and (in some cases) SPECT are already quantitative in terms of absolute radioactivity, establishing a cross-calibration amounts to determining the quantity of CR produced per unit radioactivity under imaging conditions and then measuring the light in absolute terms.

I accomplish this task using a set of computer models of CR production and apply the models to predict and tabulate the efficiency of CR production for a number of medical radionuclides under a variety of conditions affecting said efficiency. I also look at the intrinsic resolution of the Cerenkov light produced by these radionuclides. Experiments involving a subset of these radionuclides will be used to validate the model results. And

as an aside, I evaluate the CR production capacity of the two radionuclides for which this capability has been questioned, namely In-111 and Ac-225.

Finally, I propose here a simple system that uses CR as a low intensity light source able to calibrate luminescence imaging systems and thus avoids the expense of specially calibrated sources, integrating spheres and the like. I'll present data suggesting that this approach is more accurate than calibrations currently performed by manufacturers, including with regard to the calibration of spectral filters.

3.2. Modeling.

Overview. The radionuclides to be considered here decay primarily by α , β^+ , β^- and γ emissions. Neutrinos, conversion electrons, Auger electrons, characteristic x-rays, bremsstrahlung radiation, annihilation photons, δ -rays, e^+/e^- pairs and secondary electrons are also produced.

The α particles emitted by radionuclides generally are not of sufficient energy to be superluminal when transiting through water, biological tissues or other non-periodic mediums of moderate refractive indices, and should not produce CR. Likewise, the secondary electrons produced by the transiting α 's are not of sufficient velocity because each electron receives only a small fraction, a maximum of $5.48E-4 = 4mM / (M + m)^2$, of the α 's energy (where m and M are the rest masses of the electron and α , respectively). Neutrinos, Auger electrons, characteristic x-rays and e^+/e^- pairs (i.e. pair production), are all either not produced at significant quantities, are not of sufficient energy or do not interact with matter with sufficient efficiency to produce Cerenkov

radiation. Bremsstrahlung radiation can extend into the visible spectrum and thus conceivably could be confused with Cerenkov radiation, but the amount within the visible range is expected to be negligibly small and its wavelength distribution would be dissimilar to the characteristic one over wavelength squared Cerenkov distribution and thus can be easily distinguished. This leaves β^+ , β^- , δ -rays, conversion electrons and secondary electrons produced by both γ -rays and annihilation photons as the potential dominant sources of Cerenkov radiation. These are all, in essence, β particles (i.e. electrons or positrons) of varying origin. Over the range of β energies of interest here, there are negligible differences in the Cerenkov producing properties of β^+ and β^- particles and no difference what-so-ever among β^- , δ -rays, conversion electrons and secondary electrons [94].

Table 3.1 lists for each of the radionuclides to be modeled, the total abundance of each of the types of emissions at least some of which have sufficient energy to produce CR in water. Along with the half life and total abundance[95], I list the abundance of the portion of those emissions that are above the energy threshold of CR production in water (refractive index 1.33, threshold 263 keV [96]) and in mammalian tissue (refractive index of 1.4, threshold 219 keV [75]); these based on my integrations of the β energy spectra [95]. For example, while the overall β^+ abundance of F-18 is 97% the β^+ with energy ≥ 263 keV is only 43% and ≥ 219 keV is 54%. Note that since annihilations photons have a kinetic energy of 511 keV, they *are* above the threshold for both refractive indices (1.33 and 1.4) and their abundance is twice that of the β^+ total abundance.

Table 3.1. Total abundance and abundance of emissions having energy greater than CR thresholds in water and in biological tissues.

radio-nuclide	half life ^a	β^+ (%)			β^- (%)			conversion electrons (%)			γ -rays (%)		
		total	1.33	1.4	total	1.33	1.4	total	1.33	1.4	total	1.33	1.4
C-11	20.4 m	100	69	77									
N-13	9.97 m	100	79	84									
O-15	122 s	100	90	93									
F-18	109 m	97	43	54									
Cu-64	12.7 h	18	9	11	39	11	15	< 0.1			<0.1		
Ga-67	3.26 d							34	0	0	88	22	22
Ga-68	67.7 m	89	83	85				< 0.1			4	4	4
Zr-89	3.27 d	23	17	19				< 0.1			101	101	101
Y-90	2.67 d				100	89	91	< 0.1			<0.1		
In-111	2.80 d							16	0	1	185	0	94
In-114m	49.5 d							81	0	0	22	6	6
In-114	71.9 s				100	85	89	< 0.1			<0.1		
I-124	4.18 d	24	22	23				< 0.1			99	99	99
I-131	8.03 d				100	36	35	6	2	2	101	98	98
Ac-225	10.0 d							67	0	0	7	0	1

The radionuclides of interest for production of CR are listed in this table, and are modeled in this work. Characteristics of each radionuclide are given including half life, total abundance and abundance of emissions greater than the threshold for CR production. The CR abundance efficiencies are given for 1) water (refractive index 1.33, threshold 263 keV) and 2) mammalian tissues (refractive index 1.4, threshold 219 keV). ^a s - seconds, m - minutes, d – days.

Notable in this table is the lack of CR producing emissions for Ac-225 which has been reported to be a strong light producer [85]. The experiments with Ac-225 replicated this result so I thought to consider Ac-225's daughters which I expected to be in transient equilibrium with Ac-225 in these samples. Table 3.2 shows the CR capable abundances for Ac-225's daughter radionuclides along with their relative activities at transient equilibrium. These numbers suggest that Bi-213 and Pb-209 are the likely sources of the bulk of the detected CR.

Table 3.2. Ac-225 daughters abundance of emissions and those having energy greater than the CR threshold.

Ac-225 daughters	Half life ^a	% of Ac-225 activity at transient equilibrium	β^+ (%)			β^- (%)			conversion electrons (%)			γ -rays (%)		
			total	1.33	1.4	total	1.33	1.4	total	1.33	1.4	total	1.33	1.4
Fr-221	4.9 m	100				6	0	0	12	0	0			
At-217	32.3 ms	100				< 0.1			< 0.1					
Bi-213	45.59 m	100	98	65	71	5	5	5	27	27	27			
Tl-209	2.20 m	2.2	100	81	85	29	4	4	282	198	198			
Po-213	4.2 μ s	97.8				< 0.1			< 0.1					
Pb-209	3.253 h	100.01	100	28	35									
Bi-209	stable													

The alpha-emitting radionuclide Ac-225 has been identified as a strong producer of CR light. Assuming their stable equilibrium with Ac-225, I've listed the relative activities of the daughters. In this table I also list the characteristics of the daughter radionuclides, their total abundance and their capabilities to produce CR in water and tissue. ^a s - seconds, m - minutes, d - days

Note that In-114m (see Table 3.1) is a common long-lived impurity in samples of In-111. In-114m, in turn, decays to In-114. Because of In-114's short-half life its activity level in samples is in transient equilibrium with the In-114m within a few minutes and thus the two will have roughly equal activity levels. Samples of In-111 that are to be used clinically can have In-114m activity levels up to 0.15% [97] (and therefore an equal fraction of In-114) and this fraction will increase over time given In-111's faster rate of decay. In-114 has significant CR production potential from its highly abundant high energy β^- emission (see Table 3.1).

Modeling Cerenkov production per β of a given initial energy. The production of CR from a β particle is described by the Frank-Tamm formula [98] here integrated over a range of wavelengths.

$$\frac{\delta N}{\delta x} = 2\pi\alpha \left(\frac{1}{\lambda_1} - \frac{1}{\lambda_2} \right) \left(1 - \frac{1}{\varphi^2 n^2} \right) \text{ for } \varphi n > 1 \quad (3.1)$$

$$\frac{\delta N}{\delta x} = 0 \text{ for } \varphi n \leq 1$$

This formula gives the number of Cerenkov photons generated per unit path length having wavelengths within the interval from λ_1 to λ_2 expressed in the desired length unit. Here n is the mean refractive index and φ is the velocity of the β -particle divided by the speed of light in a vacuum and α is the fine structure constant.

The β -particle velocity, ν , can be determined from its energy, E , as follows:.

$$\nu = c \left(1 - \frac{E_0^2}{(E + E_0)^2} \right)^{1/2} \quad (3.2)$$

where c is the speed of light in a vacuum and E_0 is the rest mass of the β -particle expressed in the same units as E .

For a given initial energy, I used Euler's method to integrate equation (1) over the full path length of the β -particle as it decreased in energy while transiting through a medium presumed to be of infinite spatial extent. During this integration, the rate of energy loss was interpolated from the ESTAR Stopping Power and Range Tables provided by the National Institute of Standards and Technology [99]. The table used in the model had 250 logarithmically spaced points between 1 keV and 10 MeV. The energy step size for the Euler integration was 0.1 percent of the instantaneous β energy or 0.1 keV, whichever was larger. The total path lengths calculated, implicit in this process, were found to have a maximum error of 0.3% relative to the CSDA (continuous slowing down approximation) within the range of sampled energies. It should be noted that the full

path length was calculated for testing purposes only. During routine use the integration for each β is terminated when its energy drops below the CR threshold.

Modeling Cerenkov production by β 's and conversion electrons. In order to determine the average number of Cerenkov photons produced by β particles (or equivalently by conversion electrons) per disintegration for a given radionuclide, I weighted the above described integral by the relative probability of a β of a given energy being emitted by that radionuclide and then summed across all possible energies (i.e. a third integration of the original Frank-Tamm formula). The probability for each β energy was derived from the β spectra available from the Lund/LBNL Nuclear Data Search website [95]. The spectra were sampled at 1 keV intervals.

Modeling Cerenkov point spread function. In addition to the above described numerical models relating absolute number of Cerenkov photons to initial β energy, I developed a Monte-Carlo model to determine the Cerenkov point spread function (PSF) for a given radionuclide. In order to incorporate this spatial information, this model calculates the tortured path that each β particle makes as it generates Cerenkov photons and scatters off nuclei and electrons within the medium.

My model of this transport process closely followed the work of Levin and Hoffman who modeled positron transport for the purpose of determining the positron-electron annihilation PSF for various radionuclides [100, 101]. In brief, I too made use of Bethe's calculations of Moliere's theory of multiple elastic scattering from the nucleus [102] and Ritson's model of the δ -ray energy distribution [103]. For details, please see the original manuscripts. My model differed from Levin's in that instead of using the Bethe-Bloch

formula to determine the collisional energy loss rate, I used the aforementioned ESTAR table. And, instead of using the Wu and Moskowski and Daniel models of β energy spectra, I again used the aforementioned tables from Lund/LBNL. As a check on the accuracy of the model, I calculated the positron PSF for some of the same radionuclides described by Levin and found the results to be comparable.

At each step of the β transport, I calculated the number of Cerenkov photons produced based on the length of the step and on the energy of the β at the start of the step (applying equation 1 as before). The location of the Cerenkov photons was distributed linearly along the path of the β for that step. Any δ -rays of sufficient energy were set to generate Cerenkov photons in the same manner as the β particles.

Modeling Cerenkov from secondary electrons excited by γ -rays and annihilation photons. Secondary electrons produce CR in a manner identical to that of β^- particles. However, the location of the Cerenkov production is generally far away from the originating radionuclide. This is because the γ -rays or annihilation photons will often travel a long distance before undergoing the photoelectric or Compton interaction that ultimately gives rise to a secondary electron. As such, the total amount of CR produced by secondary electrons will be geometry dependent. Larger volumes will tend to have a larger fraction of total Cerenkov signal produced by secondary electrons but this is contingent on annihilation photons and/or γ -rays of sufficient energy to produce secondary electrons capable of producing Cerenkov photons within the given medium.

To investigate and quantify this effect, I created two Monte Carlo model variants describing the transport of γ -ray and annihilation photons. The first consisted of a

radionuclide point source within an infinite medium. This model was used to determine the PSF of CR due to secondary electrons. The second variant consisted of radionuclide evenly distributed within a cuboid-shaped medium. This model was specifically intended to mimic the conditions of the phantom studies (described below) that were designed to calibrate the luminescence scanner.

In both of these models, the initial directions of the simulated γ -rays emanating from the radionuclide source were randomly sampled so as to be uniformly distributed within the 4π solid angle about the source. Annihilation photon directions were similarly distributed but were created in pairs with members traveling in opposite directions. The distance traveled by each photon before interacting with the medium was randomly sampled from an exponential distribution, the log-slope of which was interpolated from a table of photon cross-sections, XCOM, available from the National Institute of Standards and Technology website [104]. The table made use of the standard grid available on the website but truncated to have energies between 1 keV and 10 MeV.

The total cross-section was used to determine the distance the photon traveled before interacting but the type of interaction, photoelectric, Compton or other, was randomly sampled reflecting the relative probabilities of each. When simulating a photoelectric interaction, all of the photon's energy was transferred to the secondary electron and the photon was terminated. For a Compton interaction, the Klein-Nishina formula was used to determine the scattering angle into which the photon was propagated, as well as the associated amount of energy transferred to the secondary electron. The CR associated with the secondary electron, if any, was determined from a lookup table calculated by

the Cerenkov model based on the electron's initial energy (i.e. the path integral of equation 1). For the PSF model, all Cerenkov radiation was attributed to the site of the photoelectric or Compton interaction. All other types of interaction were assumed *not* to produce CR (e.g. pair-production was ignored).

3.3. Experimental Cerenkov measurements.

In order to test these models, five types of experiments were conducted (see Table 3.3). In one type of experiment, I acquired a luminescence spectrum. In a second type, I varied the refractive index of the medium (water) by adding 25% by weight of sodium chloride. In the third type, the volume of the medium was varied while maintaining a constant amount of radionuclide, thus achieving a range of surface to volume ratios and radionuclide concentrations. All measurements for these first three types of experiment were made using one of three simple acrylic boxes having 2 mm thick walls; one had a 3.4x3.4x3.4 cm interior, another was 5.4x5.4x5.4 cm and the third was 9.6x9.6x9.6 cm. Henceforth these will be referred to as the 3.4, 5.4 and 9.6 cm boxes respectively. All three were painted on all surfaces with flat black spray paint (Krylon Fusion). Tests on the boxes without radionuclide present demonstrated that they did not phosphoresce significantly following exposure to visible light.

Table 3.3. Radionuclides tested and the types of experiments conducted on each.

Radionuclide	Experiment Conducted (and Number)				
	Spectrum only (1)	Refractive index (2)	Volume change (3)	β PSF (4)	Secondary electron PSF (5)
F-18	X	X	X	X	X
Ga-68	X	X		X	
Zr-89	X	X	X		
In-111	X	X			
I-131	X	X			
Ac-225	X				

Experiments were used to validate the computation model presented. This table lists the experiment types, as well as the radionuclides employed to evaluate them.

The remaining two types of experiment were designed to measure the beta and secondary electron PSF's. The fourth experiment type, measuring the beta PSF, used a 5x5x3 cm solid acrylic block into which was cut a 0.11x3 cm by 1 cm deep slot on the 5x5 cm surface. The slot was filled with a mixture of radionuclide, India ink and surfactant. The India ink significantly reduced the CR emanating from the slot itself, leaving predominantly CR produced in the acrylic, which was taken to have a refractive index of 1.491 [105]. The surfactant allowed the slot to be filled without significant air pockets.

The fifth experiment type, which measured the secondary electron PSF, made use of a 10x10x5 cm solid acrylic block onto which a small drop of radionuclide was placed in the center of its 10x10 cm face.

All CR measurements were made on Caliper Life Science's IVIS 200 luminescence imager with the phantom placed in the center of a 13x13 cm field-of-view. The camera focus was set at 1.5 cm above the platform (i.e. the surface on which the box rested). The IVIS 200 uses a fixed focus lens and adjusts the focal point by adjusting the

platform height relative to the camera. The 1.5 cm setting is the default focus point and I used this setting regardless of the height of fluid contained within the box. For the acrylic block phantom measurements, the camera was focused on the proximal surface of the block. All luminescence images were taken with an f-stop of $f1$ and a binning of 4 (i.e. 2x2 groups of pixels summed). Cosmic ray and background corrections were turned on. Total radioactivity of the radionuclide samples were measured with a Capintec Model CRC-127R dose calibrator (Capintec, Inc. Ramsey, NJ).

The IVIS 200 uses a cooled, back-thinned CCD (charge coupled device) detector. The signal measured from each pixel of this detector is roughly proportional to the number of photons impinging on the element during an image acquisition frame. The lens that focuses the light on the CCD includes an aperture which defines the solid angle of photon acceptance at a given focus point distance. The focus distance, along with the focal length of the lens, determines the surface area seen by each pixel. Thus the images acquired by the IVIS can be calibrated to photons per second per cm^2 per steradian. For isotropic sources, the pixel values can be summed and multiplied by 4π steradians and by the area covered by the pixels in cm^2 to arrive at the total photon flux in photons per second.

Although the direction in which Cerenkov photons propagate is dependent upon the direction of travel and energy of the charged particle, because the directions of the charged particles in these Cerenkov efficiency experiments are isotropic, so too on average are the Cerenkov photons. Note this is not precisely true of the secondary electrons or their associated Cerenkov photons in the Cerenkov efficiency experiments.

The initial direction of travel of secondary electrons relative to the parent photon direction is governed by the Klein-Nishina equation and is not strictly isotropic. However, as these electrons scatter producing CR along their path much of the directional bias is lost to the point where an isotropic assumption is acceptable. A similar reasoning applies to the β 's and associated Cerenkov photons in the PSF measurements.

The photon attenuation of deionized water and 25 percent NaCl in water is negligible over the range of wavelengths of the spectral measurements (550 to 670 nanometers). Thus, the total photon flux (within a range of wavelengths), after applying the corrections described below, is a direct estimate of the total CR production by the total radioactivity present. Thus the efficiency of the CR production can be simply calculated as the photon flux divided by the total radioactivity (e.g. photons per second per becquerel or equivalently, photons per disintegration). It should be noted that this means that the total photon flux measurement is by-in-large independent of the concentration of the radionuclide and that the pixel intensities varied predominantly with the surface area of the medium.

3.4. Spectral measurements.

Spectral luminescence measurements were made using the six 20-nanometer band-pass filters available on the IVIS 200, centered at 560, 580, 600, 620, 640 and 660 nanometers. Immediately preceding these image acquisitions, one or more open (i.e. without filter) images were acquired, varying the frame duration until a reasonably low-noise image of the Cerenkov radiation was achieved. The filtered image frame durations

were set to be 20 times that of the low-noise open image. The open measurements were used only to predict reasonable frame times for the filtered measurements. I did *not* model the open condition. Prior to the spectral acquisition and each of the open acquisitions, an associated reflected light image was acquired.

3.5. Cerenkov efficiency as a function of refractive index.

Radionuclide, initially in a volume no greater than 1 mL, was thoroughly mixed with deionized water achieving a total volume of 30 (or 100 in some experiments) mL at room temperature. The refractive index of this medium was taken to be 1.333 with a density of 0.998 g/cc at 20 °C. This solution was then transferred to the 3.4 (or the 5.4) cm box and a set of spectral measurements made with the IVIS imager.

The solution was then temporarily transferred to a container with a closeable top, to which was added 10 (or 33.3) grams of NaCl and shaken to dissolution, thus achieving a 25% by weight salt solution assumed to have a refractive index of 1.377 [96] and density of 1.281 g/cc [106]. The salt solution was then returned to the box container and imaged as before.

3.6. Cerenkov efficiency as a function of volume.

Radioactivity was initially mixed with 10 mL of deionized water and imaged in the 3.4 cm box placed in the center of the field of view. Without moving the box, 15 mL of deionized water was added bring the total to 25 mL. It was allowed to mix thoroughly and was reimaged. This sequence was then repeated but this time adding 25 mL (for a total of 50 mL). The whole volume was then transferred to the 9.6 cm box, also placed in the

center of the field of view. Another 50 mL of deionized water was added and imaged. The sequence was repeated two more times adding 150 and 250 mL for a total of 250 and 500 mL in the container, respectively.

3.7. Region of interest measurements and profiles.

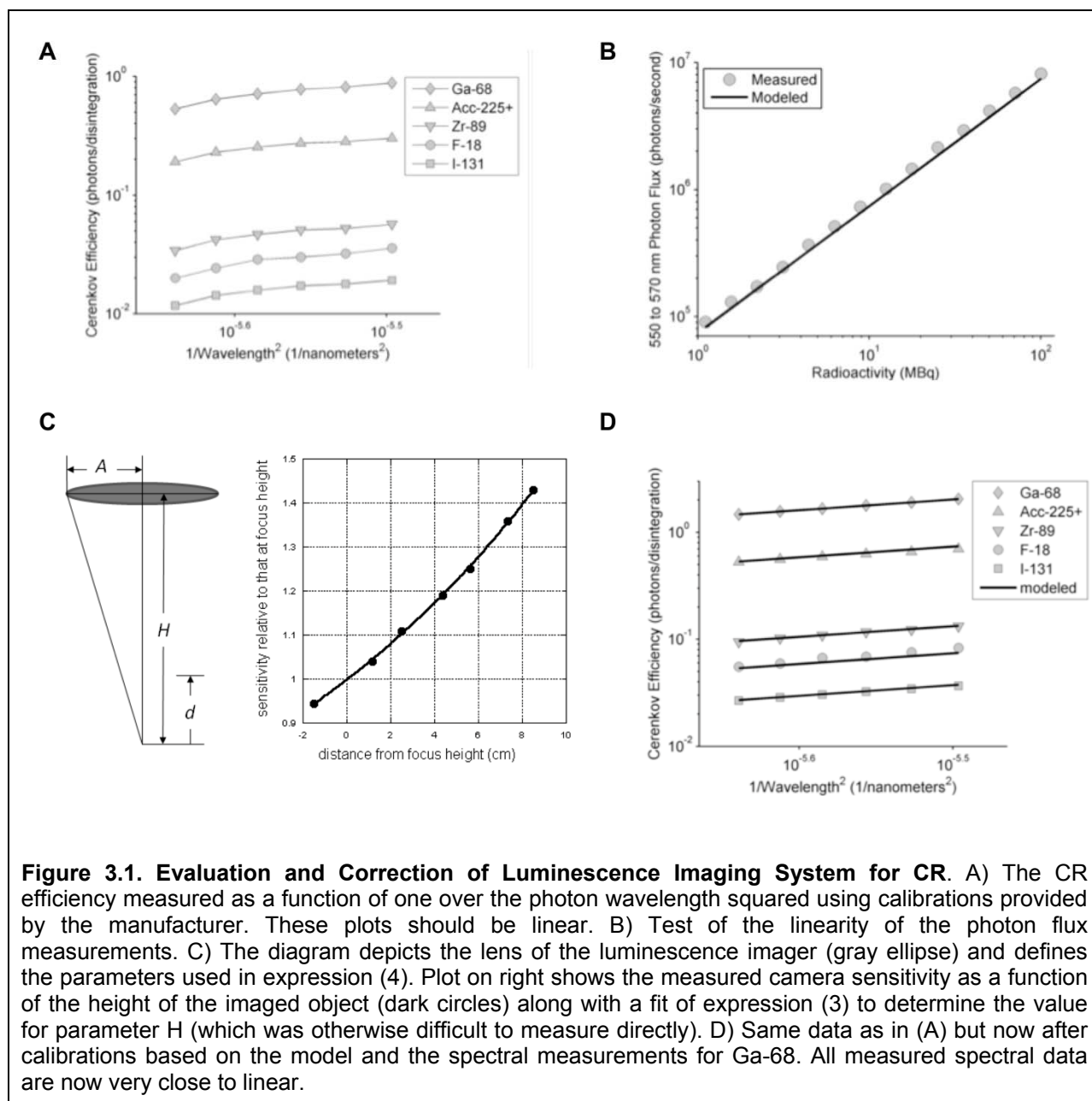
All region of interest measurements were made using the Living Image software (Caliper LifeSciences, Inc., Hopkinton, MA) which comes standard with the IVIS 200. This software is designed to provide quantitatively accurate images in radiance units (photons/second/cm²/steradian) and includes adjustments accounting for platform height, lens aperture setting, field inhomogeneity, pixel binning and various sources of background. For the Cerenkov efficiency measurements, I placed a large region of interest over the homogeneous region within and well away from the edges of the box containing the deionized water or salt solution medium. The mean radiance within this region was then multiplied by the known surface area of the box opening (e.g. 92.16 cm² for the 9.6 cm box) and by 4π steradians to arrive at the total Cerenkov photon flux in photons per second. Calculating the total flux in this manner, effectively corrects for light lost due to β particles entering the side walls of the container.

The PSF profiles were measured using custom code written in Matlab (The MathWorks, Inc., Natick, MA). For the Cerenkov profile due to β emissions, this entailed first rotating the image so that the slot was precisely aligned with the vertical axis of the image and then summing along the length of the slot to generate a profile extending perpendicular to the slot.

For the Cerenkov profile due to secondary electrons, the center of the drop (i.e. the radionuclide source) was identified within the image and the mean radiance surrounding that central point was plotted as a function of distance from that point.

3.8. Corrections

IVIS recalibration. During the course of this work it quickly became apparent that neither the global absolute calibration nor the relative calibration of the individual 20 nm band-pass filters of the IVIS imager was accurate. Specifically I noted that the Cerenkov spectra I was measuring did not have the characteristic one over wavelength squared shape I was expecting and yet the shape was consistent across radionuclides (Figure 3.1A). Examining the Cerenkov spectra published by others [84] I noted similarly consistent spectral curve shapes across radionuclides that were both different from the theoretical shape and different from that which I was measuring. I also noted, even after all corrections, that my Cerenkov measurements were consistently about half of that predicted by the models. Cleaning the lens and filters within the IVIS had a dramatic affect on the system's sensitivity but still failed to bring it in line with expectations. Rather than publish data based on what I believed to be a miscalibrated instrument, I decided to recalibrate the IVIS based on a single spectral measurement of the Cerenkov light given off by Ga-68 in deionized water. This amounted to multiplying each of the filtered measurements by a slightly different factor ranging between 2.3 and 2.8. This same set of constants was used for all subsequent measurements (i.e. in effect I recalibrated the filters).



Decay Correction. All doses were calculated as the mean dose present during the interval over which the image was acquired. This was accomplished by applying a decay factor to the dose calibrator measurement. The decay factor, DF , was calculated using the following well known formula:

$$DF = e^{-\lambda t} \left(\frac{1 - e^{-\lambda \tau}}{\lambda \tau} \right) \quad (3.3)$$

where $\lambda = \ln(2)/T_{1/2}$, $T_{1/2}$ is the radionuclide half life, t is the time between when the dose was measured and the start of the acquisition frame and τ is the frame duration.

Background Correction. The standard image processing on the IVIS 200 includes a correction for the roughly uniform background typically encountered in luminescence imaging. However, in these measurements there was an additional source of background when imaging some radionuclides. This background is due to the direct detection of x-ray, γ -ray and/or annihilation photons by the luminescence detector. Because these high energy photons are not focused by the IVIS's lens system, this background too is fairly uniform. To correct for this background, I subtracted a constant from the luminescence image. The constant was determined by taking the mean value of a large region of interest placed a few cm away from the radionuclide source in each acquisition.

Linearity Correction. If all other things are held constant, the amount of Cerenkov radiation produced by a radionuclide is directly proportional to the amount of radioactivity present. Thus, given the well and accurately known half-life of F-18, for example, multiple measurements of Cerenkov light made as a radionuclide decays make for a good test of the linearity of a luminescence imaging system.

To test the linearity of the IVIS 200, I started with 3.5 mCi of F-18 diluted in 150 mL of deionized water, placed in the 5.4 cm box and imaged it repeatedly over 6.5 half-lives,

11.9 hours total. The frame duration (i.e. time the shutter was open) was held constant at 5 minutes for each measurement. Images were acquired every 54.885 minutes (i.e. 1/2 of F-18's 109.77 minute half-life) with the 560 nm (20 nm band pass) filter in place.

In Figure 3.1B is shown a scatter-plot of the radioactivity level versus the background corrected total photon flux rate measured in each image. The solid line shows the amount of Cerenkov light predicted by the model. Based on these results I determined that a linearity correction was not necessary.

Source to Camera Distance Correction. As a point source moves closer to the camera system, the solid angle limiting which photons have a chance of being detected by the camera, increases. Thus, closer objects appear brighter than more distant objects. For the phantom studies, the camera is detecting light from sources distributed throughout the depth of the liquid medium. Sources at shallower depths, therefore, are being detected with greater efficiency. This effect is described by the expression:

$$\frac{1 - \cos\left(\tan^{-1}\left(A/(H - d)\right)\right)}{1 - \cos\left(\tan^{-1}\left(A/H\right)\right)} \quad (3.4)$$

where d is the depth relative to a reference distance H (e.g. the distance to the focus point) and A is the radius of the aperture at $f1$ which, for the IVIS 200, is 6.35 cm [107].

The value for H was determined by performing a nonlinear least-squares fit to a series of measurements of the total photon flux taken from a constant planar source positioned at various heights relative to the focus point (1.5 cm above the platform in all the

measurements). This procedure found H to be 51.2 cm. The parameter definitions, measurements and the fit are shown in Figure 3.1C.

In addition, because of the change in refractive index between the medium and the air above it, each plane at a given depth is magnified (a phenomenon well known to SCUBA diving enthusiasts, wherein objects under water appear to be closer than they really are). This magnification affect reduces the apparent radiance at a given depth in that the photons produced there appear to be generated over a larger surface area. Specifically, the magnification and thus the factor decrease in radiance, is described by the following:

$$M = \frac{D + L + F}{nD + L + F} \quad (3.5)$$

where D is the distance below the surface, L is the distance from the lens to the surface, F is the focal length of the lens and n is the refractive index of the medium.

To arrive at a correction for measurements taken from fluids of differing depths, I averaged expression (4) divided by expression (5) over the entire depth of the fluid medium.

Loss of Cerenkov at Surfaces Correction. The β -particles leaving the medium at its surfaces result in a loss of Cerenkov light production. This loss was estimated by the following:

$$C \cdot S \cdot \int_0^\infty \left[0.5 - \int_0^y psf(x) dx \right] dy \quad (3.6)$$

where $psf(x)$ is the CR point spread function, C is the radionuclide radioactivity concentration, S is the medium's surface area and x and y are both distances from the side of the container.

Because the measurements of total photon flux avoided losses at the sides of the box (by extrapolating the central homogeneous radiance to the edges), S refers only to the area of the top and bottom surfaces. This expression assumes infinite extent for the dimensions parallel to the edge and does not consider the overlap at the edges and vertices of the containers, which will become significant as the container dimensions approach the full width half max of the PSF. For the containers, however, this does not incur a significant error given the PSF's considered here.

3.9. Comparisons

Comparison of Measured and Modeled Cerenkov Efficiencies. Following the recalibration of the IVIS 200 imager based upon my measurements of the Ga-68 CR spectrum, all spectral measurements of CR demonstrated the characteristic one over wavelength squared functional form (see Figure 3.1D). This was true for both the deionized water and salt solution measurements and for all radionuclides, including the Ac-225 and In-111 measurements. Moreover the magnitude of the predicted relative to the measured CR efficiencies, following the recalibration, were all within the error of the dose calibrator measurements.

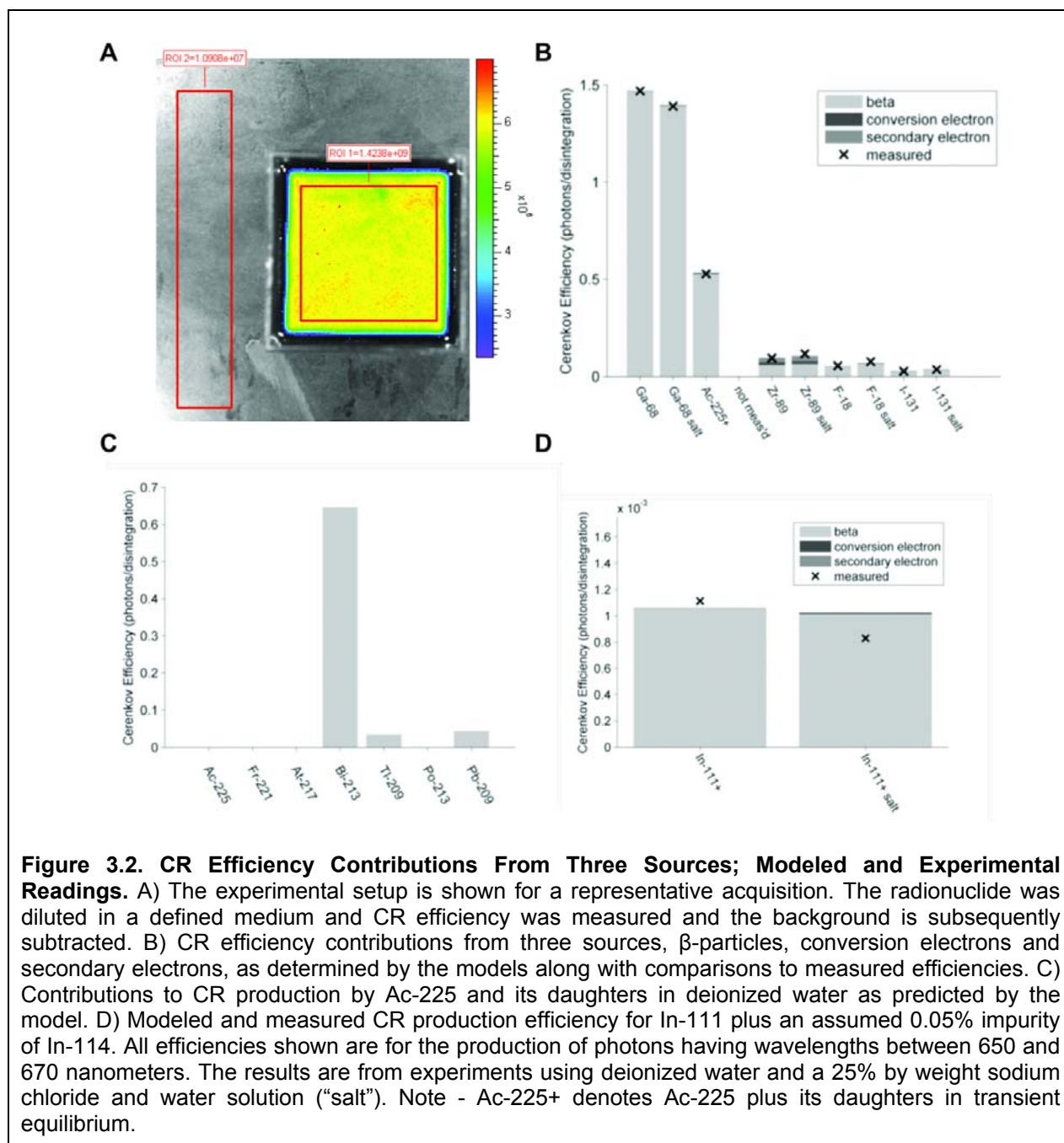


Figure 3.2A shows a representative acquisition for the experimental validation of the model results. The signal at a defined wavelength window (650-670 nm) from a radioactive source in a defined medium (for example, water) and in a defined volume. Extraneous background signal is subtracted. A chart, Figure 3.2B, with bars breaking

down the contributions for beta, conversion electron and secondary electron components for each radionuclide in water and in salt solution with X's showing measures made with the 660 +/- 10 nm band pass filter. Each of the measurements was made using a reasonably large volume of medium (~100 mL except for I-131 which was made in ~30 mL) and yet the CR contribution from secondary electrons in almost all cases was negligibly small. This was despite the high abundance of annihilation photons in Ga-68 and F-18. This can be understood by appreciating that Compton interactions in these mediums are far and away the dominant mechanism by which an annihilation photon (having a kinetic energy of 511 keV) gives rise to secondary electrons. Compton interactions allow a maximum transfer of energy to the secondary electron that is well below the energy of the photon; the so-called Compton edge. For a 511 keV photon, the maximum energy transfer in a Compton interaction is 340.7 keV. Most interactions transfer far less energy.

Zr-89 on the other hand has an appreciable contribution from secondary electrons. In this case, however, these are not primarily resultant from Zr-89's annihilation photons but rather from its 100% abundant 909 keV gamma which can transfer up to 709.6 keV in a Compton interaction. The related conversion electrons also contribute significantly to Zr-89's CR production efficiency.

The observant reader will also have noticed that Ga-68's CR efficiency in the salt solution medium is lower than that in deionized water, a trend that runs contrary to the usual increase with increasing refractive index (see below and Figure 3.5A). The explanation for this can be found by noting that the salt solution also has a higher mass

density and therefore a higher β attenuation cross-section and concomitant reduced β -particle path length. Increased density therefore tends to reduce CR production efficiency, but for radionuclides having relatively low energy β 's, the increased refractive index overwhelms this reduction. For the high energy β 's of Ga-68 however, the impact of refractive index is small and the density effect dominates.

Comparison of Measured and Modeled for Actinium-225 and Indium-111. As can be seen in Figure 3.2B, the model does an excellent job predicting the amount of CR produced by Ac-225 and its daughters when it is assumed that transient equilibrium has been reached. It should be noted that the dose calibrator setting I used (Capintec cal #775 with a 5X multiplier) to quantify the dose, makes a similar assumption. For the volume of medium used in this experiment, the contribution from secondary electrons (and from conversion electrons in general), were negligible, leaving β -particles from Ac-225's daughters as the predominant source. The CR contribution from Ac-225 itself is non-existent (see Figure 3.2C) and the vast majority of the CR signal is attributed to Bi-213.

The model of In-111 in a 25% salt solution medium predicted a CR production efficiency of 2.57×10^{-5} photons per disintegration within the 550 to 570 nanometer range. This is just 2.5% of the light within this range that was measured emanating from the In-111 sample in the experiment. In deionized water, the model predicted zero contribution from In-111. If, however, I assumed that In-114 was present as an impurity in the sample at a level of 0.05% (i.e. within the FDA allowed 0.15% for this unexpired sample), the measured and modeled came within reasonable agreement (see Figure 3.2D)

especially considering that the background levels in these measurements were over 80% of the measured signal.

Comparison of Measured and Modeled Cerenkov from β , Point-Spread-Functions.

Figure 3.3A shows a Monte Carlo simulation of the paths taken by 200 β^+ particles emanating from a single point and having energies equivalent to those emanating from F-18. CR is produced all along these tracks until the β energy drops below the CR threshold.

The experiments measuring the Cerenkov from β PSF used a roughly planar source of radioactivity and was integrated over the two axes parallel to this plane; the depth dimension integration being done implicitly by the camera resulting in an image (see figure 3.3B) and the other during the post processing of the images. As such, these experiments did not measure the PSF directly but rather they measured (approximately) the projection of this function onto the axis perpendicular to the plane. Therefore, I adjusted the output of the model, which calculates the distribution of Cerenkov light about a point source, projecting this light onto a single axis.

Figures 3.3C and D show the integrated PSF profiles from this type of experiment for F-18 and Ga-68, respectively. The profiles extend through and beyond the radionuclide containing slot (i.e. plane) in both directions and thus there are two independent measurements of the projected PSF with a gap (the width of the slot) in between. The India ink greatly diminished but did not eliminate the Cerenkov light emanating from the

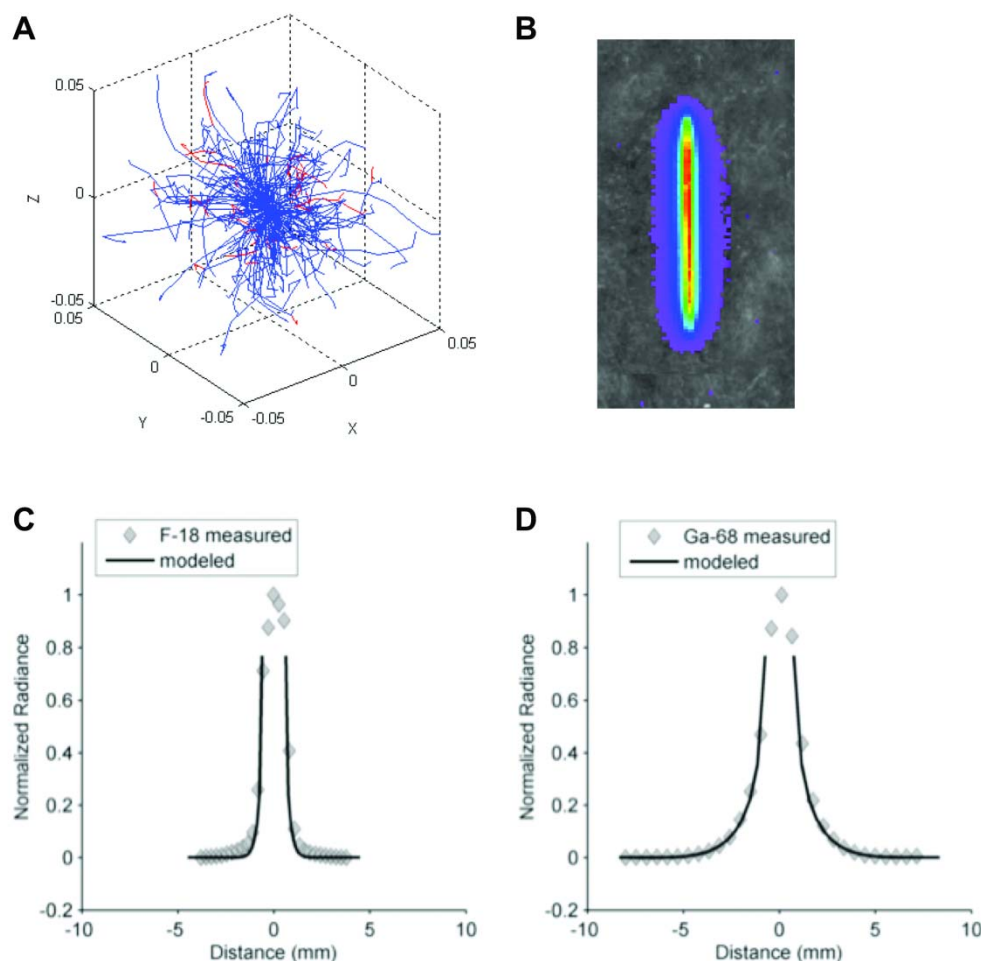
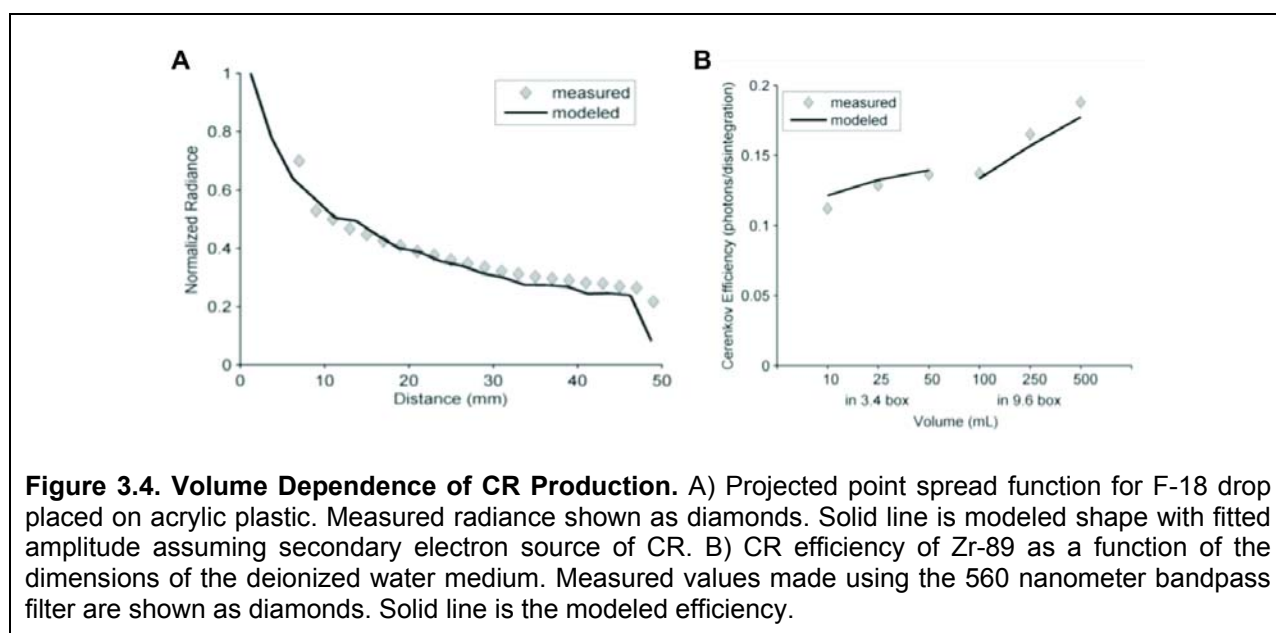


Figure 3.3. CR from β 's Point Spread Functions. A) Simulated β^+ tracks (blue) from an F-18 point source. Red tracks are from δ particles. B) A representative acquisition of the PSF experimental setup. This shows the channel in the acrylic block filled with a mixture of activity, surfactant and India ink. C) Integrated F-18 and D) Ga-68 measured radiance profiles shown as diamonds. Solid lines are modeled shapes with fitted amplitudes assuming β -particle source of CR.

slot proper, hence the signal attributed to this region seen in the graphs. The solid lines are the modeled PSF projections (one a mirrored version of the other and separated by the known gap width) scaled somewhat arbitrarily so as to achieve a good fit to the measured data.

Comparison of Measured and Modeled Cerenkov from Secondary Electrons, Point-Spread-Function. The measurement of the Cerenkov from secondary electrons

PSF, likewise, did not measure the PSF radial profile directly. Instead, in this measurement the camera first integrates the PSF over the depth dimension (i.e. that parallel to the direction in which the camera is pointing) and the resultant two-dimensional PSF is then projected onto a single radius during post processing. The output of the model was adjusted to mimic these projection operations and the result was scaled to fit the measured curve. The result is shown in Figure 3.4A.



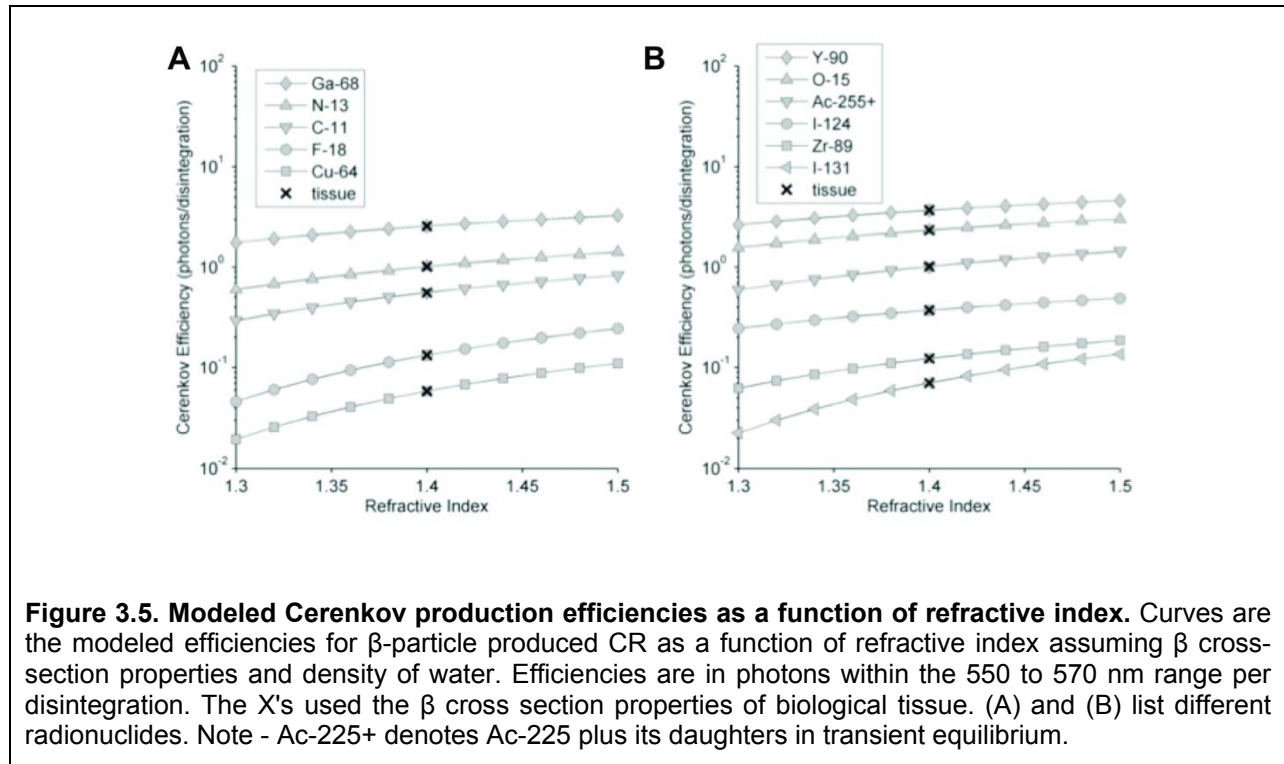
As can be seen in this plot the tail of the PSF would actually extend beyond the dimensions of the block. A block large enough to measure the PSF in its entirety would not be able to fit in the light tight enclosure of the IVIS camera system.

Comparison of Measured and Modeled Volume Dependence. The model of the loss of CR due to β 's and conversion electrons near the exterior surfaces in the experiments suggest that this affect is negligibly small for the volumes used. As noted previously though, CR production attributed to secondary electrons is expected to increase with

increases in the overall size of the medium. Figure 3.4B shows this dependency for the one radionuclide looked at having a significant CR contribution from secondary electrons, Zr-89. The predictions closely match the measured efficiencies.

3.10. Modeled Cerenkov production efficiencies as a function of refractive index.

Having validated the accuracy of the models, I thought it would be beneficial to use the models to characterize a larger list of radionuclides so that investigators might use this information when selecting a CR producing radionuclide for a given purpose. Towards this end, I present in Figures 3.5A,B the CR production efficiencies for photons within the 550 to 570 nm range from β emissions predicted by the model and plotted as a function of refractive index for a variety of radionuclides. Other wavelength ranges can readily be calculated from this information by applying knowledge of the CR spectral shape. These curves assume a medium with β and γ cross-sections and density equal to that of water at 20 °C, this in spite of the changing refractive index. While this is not entirely realistic, I felt the curves would be informative and reasonably accurate for water-like mediums such as biological tissue. To highlight this point I've included, on the same graph, points calculated for the cross-section [108], density and refractive index of tissue [75]. These efficiencies are also shown in Table 3.4 for the reader's convenience. The values shown do not include the CR production attributed to conversion electrons or to secondary electrons, which for the small animal geometries where this information is likely to be applied, are both expected to be small.



As can be appreciated in these curves, CR production efficiency generally increases with increasing refractive index but the rate of this increase is radionuclide dependent. Generally speaking, radionuclides having higher energy β emissions will have a lower proportional increase in CR per unit increase in refractive index whereas radionuclides having β 's closer to the CR threshold will have a greater proportional increase.

Table 3.4. CR from β Efficiencies.

Radionuclide	Efficiency	Radionuclide	Efficiency
C-11	0.5568	Zr-89	0.1230
N-13	1.0132	Y-90	3.7047
O-15	2.3301	I-124	0.3718
F-18	0.1328	I-131	0.0703
Cu-64	0.0583	Ac-225+	1.0143
Ga-68	2.5607		

The CR efficiencies for the radionuclides modeled in Figure 3.5A,B at the refractive index of tissue (1.4) are listed for convenience. Efficiencies are in photons within the 550 to 570 nm range per disintegration. Ac-225+ denotes Ac-225 plus its daughters in transient equilibrium.

3.11. Modeled Cerenkov point spread functions.

The Cerenkov from β PSF (prior to projection) is radially symmetric and therefore is described by its projection onto a single radius (i.e. integration over all angles). The resultant profile, it turns out, is reasonably well described by a sum of two exponentials. In order to arrive at robust values for the full-width at half-max (FWHM) and full-width at tenth-max (FWTM) values for this profile, I chose to fit the Monte-Carlo modeled data with a sum of two exponentials and calculate the metrics from the fitted curves using the modeled maximum value as the peak value. I present the results in Table 3.5 for the simulations of several commonly used radionuclides in biological tissue (i.e. refractive index 1.4 and tissue β attenuation).

Table 3.5. CR from β PSF width metrics.

Radionuclide	FWHM	FWTM	Radionuclide	FWHM	FWTM
C-11	0.712	1.824	Zr-89	0.712	1.664
N-13	0.816	2.330	Y-90	1.082	5.010
O-15	0.928	3.644	I-124	0.882	3.406
F-18	0.492	1.066	I-131	0.490	1.086
Cu-64	0.492	1.080	Ac-225+	0.790	2.194
Ga-68	0.928	3.996			

List of PSF of the modeled radionuclides at the refractive index of tissue (1.4). FWHM and FWTM values are in mm. Ac-225+ denotes Ac-225 plus its daughters in transient equilibrium.

3.12. Discussion

I have developed a set of models that accurately predict the CR production efficiency of various radionuclides through two mechanisms, directly from emitted β particles (and equivalently from conversion electrons) and from secondary electrons produced by the radionuclide's γ -rays or annihilation photons. The models allow both the refractive index and the photon cross-sections of the medium to be varied and thus should work for a

variety of materials, including biological tissues. I've applied these models in two geometries (a point source in an infinite medium and a uniformly filled cuboid medium) and validated them experimentally. These models can be readily adapted to geometries of arbitrary shape and source distribution.

In addition, I have used these models to tabulate, for a number of commonly used medical radionuclides, the CR production efficiency and parameters describing the β particle and secondary electron Cerenkov point spread functions. This information can be used to evaluate which radionuclides are most suitable for a given application.

In 1969, HH Ross [109] modeled CR based counting of β emissions as an alternative to scintillation counting for radionuclide calibration purposes. My work builds on Ross' with improvements in accuracy and extensions specifically suited to imaging applications.

While this manuscript was under initial review, a paper by Mitchell et. al. that described modeling of Cerenkov production was published [110]. My work differs from theirs in that their models utilized Monte Carlo techniques at an earlier stage and they did not consider CR production by secondary electrons. Nor did they attempt to validate many of their results. Although their model was based on entirely different computer code, the Cerenkov efficiency results they reported are virtually identical to the values I calculate with my CR from β 's model.

Since the radioactivity level of many radionuclides can be determined with great accuracy, the Cerenkov efficiency information allows for a simple means of calibrating imaging systems capable of measuring low levels of light. Pure positron emitters (such

as F-18 or Ga-68) in water will have little volume dependency and can be calibrated accurately in a dose calibrator. Ga-68 in particular is insensitive to small changes in refractive index in the vicinity of 1.33 and thus measurements from it are robust to temperature fluctuations and other factors affecting the refractive index of the medium. A β^- emitter having only relatively low energy γ 's (for dose calibration) may be even better. Using a simple setup, such as one of the boxes I described, the measured light in photons per second corresponds directly to the total dose of radionuclide. The corrections for background, source to camera distance and surface loss were all very small; as was the secondary electron contribution. Thus a simple multiple integration of the Frank-Tamm formula provides a robust and direct estimate of the true photon flux. By choosing a radionuclide with a moderate half-life, the linearity of the system can also readily be tested and nonlinearities corrected.

I investigated the mechanism of the light production for two radionuclides, Ac-225 and In-111, for which the Cerenkov mechanism was called into question. My analysis suggests that Ac-225 per se does *not* generate Cerenkov light, but that one of its β emitting daughters, Bi-213, is responsible for the bulk of the Cerenkov signal with significant contributions from Tl-209 and Pb-209. For In-111 I found that although it is theoretically capable of producing CR, the amount of light produced is extremely small and significantly smaller than that which was measured. I show evidence that the amount of CR produced is consistent with an In-114 impurity as its source.

As mentioned in the Introduction, the primary goal in developing these models is to determine the amount of CR produced by radionuclides placed within biological tissues.

For this purpose, accurate knowledge of the refractive index of the tissue is necessary. However, there is a fair amount of uncertainty in the literature regarding the refractive indices of tissues [111] and even small differences can have a large impact on the amount of CR produced. There is also likely to be variation from one organ to another within the animal and certainly the refractive index will be very different for structures such as the urinary bladder. Radionuclides having higher energy β 's are less sensitive to these variations in refractive index and therefore may be more desirable although at the cost of a reduction in resolution. In another context, the application of a controlled electron energy source may prove to be an accurate method of assaying the refractive index of a given tissue.

4. SPECIFIC AIM 3: OPTIMIZED ACQUISITION PROTOCOL

4.1. *Overview*

When an investigator plans an experiment in which he or she will be acquiring data with a bioluminescence imager for the purpose of generating BLT reconstructed images, they are faced with several choices regarding how to acquire that data. They need to decide what the overall duration of the acquisition should be, what filters to use during the acquisition and how they should distribute the overall time among the different filtered measurements. They also need to decide what spatial sampling frequency to use (i.e. the height of the camera and the binning of the CCD) and what voxel size to use in the reconstruction. Currently, there is little to no guidance to help investigators make these decisions. It is my goal here to rectify this situation.

Specifically I will be proposing procedures and algorithms that seek to reduce the noise and improve the overall quality of the data that is used to reconstruct BLT images. In order to do this optimally, it will be necessary to establish a relationship between the noise in the data and the noise in the final image. This relationship ultimately depends on the specific reconstruction algorithm used. For the purposes here, I will do this for two reconstruction algorithms, a maximum likelihood expectation maximization algorithm (MLEM) [112] and a direct reconstruction algorithm employing the Moore-Penrose pseudoinverse [15, 16]. Although the Moore-Penrose is seldom if ever used in practice, its use here (as I'll detail below) is somewhat pedagogical. Moreover, both the Moore-Penrose and MLEM methods can be considered canonical in that each finds a

solution using a pure classic target function, unbiased by regularizations, penalty functions, et cetera.

The Moore-Penrose pseudoinverse matrix is well known for its capacity to compute a 'best fit' least-squares solution to a system of linear equations like those which are encountered in BLT image reconstruction [16] (for details see section 4.5). Because this solution is a linear operation it can also be used to determine the uncertainties in the solution as a function of the uncertainties in the measurements. If one expresses the measurement uncertainty as a standard deviation about the expected values, the elements of the pseudoinverse can be seen to be coefficients weighting the relative contributions of the measurements to the uncertainty in each voxel solution. Following standard propagation of uncertainty rules, the standard deviations sum in quadrature wherein these coefficients are also squared [113].

As I will demonstrate below, given an accurate estimate of the measurement noise level this operation provides an exact estimate of the uncertainty in the least-squares solution to the BLT inverse problem and in addition, allows for an optimal selection of filters within a specified range of wavelengths along with the optimal distribution of acquisition times among those filters given an overall duration for the experiment. Because it provides an uncertainty estimate for each individual voxel, this information can also be used to guide the sampling of the solution space (i.e. the number, size and distribution of voxels within the animal or objects interior).

However, while all of this works perfectly for least-squares solutions, it is rare in BLT that a least-squares solution is sought. Without a constraint requiring only positive

valued solutions, the noise in the resultant images is simply much too high. To get around this problem an iterative MLEM algorithm (for details see section 4.5) can be used to reconstruct the images.

In 1994 Barrett et al. [17] undertook an investigation of the noise properties of the MLEM algorithm and determined that the uncertainty in the solution after k iterations, $\epsilon^{(k)}$, was log-normally distributed and could be approximated from a linear operation acting on the measurement noise, η , specifically:

$$\epsilon_n^{(k)} = \sum_{m=1}^M U_{n,m}^{(k)} \cdot \eta_m \quad (4.1)$$

wherein $U^{(k)}$ is calculated in an iterative procedure (see Barrett [17] for details).

For the purposes here, I will make use of the $U^{(k)}$ matrix for MLEM solutions in manner similar to my use of the Moore-Penrose pseudoinverse for least-squares solutions. However, because of the approximations used by Barrett in the derivation of $U^{(k)}$, the noise estimates are not as accurate and the time distributions are in some cases slightly suboptimal.

4.2. *Camera noise model*

In bioluminescence imaging there are practical limits on the maximum source intensity per voxel within an animal. Given this maximum and a specified useful dynamic range, one can infer a threshold difference in source intensities that one would want to be able to reasonably detect. Summed across all wavelengths, in vivo photon flux rates in Colo 26-luc2 cells (for example) have been found to be about 250 photons per second per

cell [114]. Assuming 10^6 cells per μL , a voxel volume of $1 \mu\text{L}$ and a target dynamic range of 1000:1, this would mean that we'd want to reliably distinguish voxels having a difference in photon flux of 2.5×10^5 photons per second.

For the purposes of this dissertation it will be assumed that the noise in the measurements made by the BLT camera system are due to photon shot noise, CCD readout noise and CCD dark current. Thus the camera sensitivity and noise will not vary over the field of view but they will vary with the resolution of the CCD image (i.e. the image matrix size) and the band-width of the filters. The dark current noise will be assumed to increase linearly with time, whereas the read noise will be time invariant.

The signal to noise ratio of this system thus can be described by the following expression:

$$SNR = \frac{PQ_e t}{\sqrt{PQ_e t + Dt + N_r^2}} \quad (4.2)$$

where: P = photon flux incident on the CCD (photons/pixel/second)
 Q_e = quantum efficiency of the CCD (85%)
 D = dark current (1.82×10^{-4} electrons/second per $13.5 \times 13.5 \mu\text{m}$ pixel)
 N_r = read noise (5 electrons rms/pixel regardless of pixel size)
 t = integration time (seconds)

4.3. *Digital mouse phantom*

In order to assess the benefit of the proposed algorithms it is important to start with a system matrix based on a model that demonstrates the depth dependent resolution and

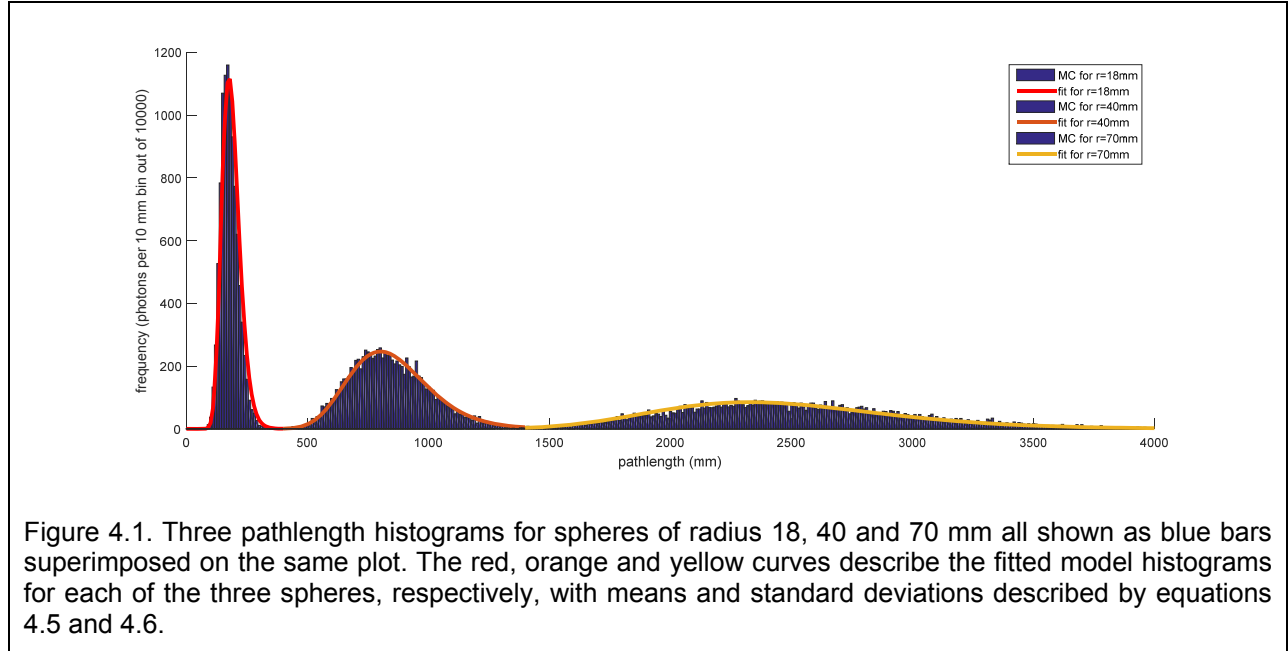
spectral shift dependencies encountered in live animals. However, it is also important that the accuracy of the inverse problem results not be adversely affected by inaccuracies of the system model owing to uncertainties regarding the light propagation properties of the tissue.

These types of uncertainties are prevalent in live animal models, therefore I made use of a digital mouse phantom derived from a whole-body CT image of a live mouse. The CT image was segmented using MIPAV image processing software (NIH, Center for Information Technology) applying a simple threshold to separate the body from the surrounding air followed by the application of its “fill holes” morphological filter. The voxels immediately outside the mouse were then identified using MIPAV’s “find edges” function.

4.4. Solving the forward model

The BLT system matrix was calculated based on an empirical model fitted to data originating from a Monte Carlo simulation (code I’ve developed similar to the beta particle transport Monte Carlo code that I wrote for specific Aim 2 – see Appendix for details) of photon propagation from a point source centered within a sphere of “tissue” having a reduced scattering coefficient of 0.92 mm^{-1} (this being a rough average value for a variety of tissues within the visible range). Each iteration of the Monte Carlo code simulated 10,000 photons and calculated the path-length each traveled before reaching the surface for a given sphere radius without attenuation. For each sphere radius (ranging from 0.5 to 100 mm) a histogram of the path-lengths was averaged over 10 iterations. The resultant mean histograms (see figure 4.1) all appeared to be log-normal

with log-means (i.e. the mean of the log of the path-length distribution) and log-stdevs that varied with the sphere radius.



Fitting this data with empirically derived models (i.e. functions that fit well) arrived at the following expressions for the log-mean of the path-length distribution (LMD) and the log-standard deviation of the distribution (LSD):

$$LMD(r) = \ln(0.7339r^{1.907} + 1.66) \quad (4.3)$$

$$LSD(r) = 0.1973(1 - e^{-0.5558r}) + 0.3146e^{-1.403r} \quad (4.4)$$

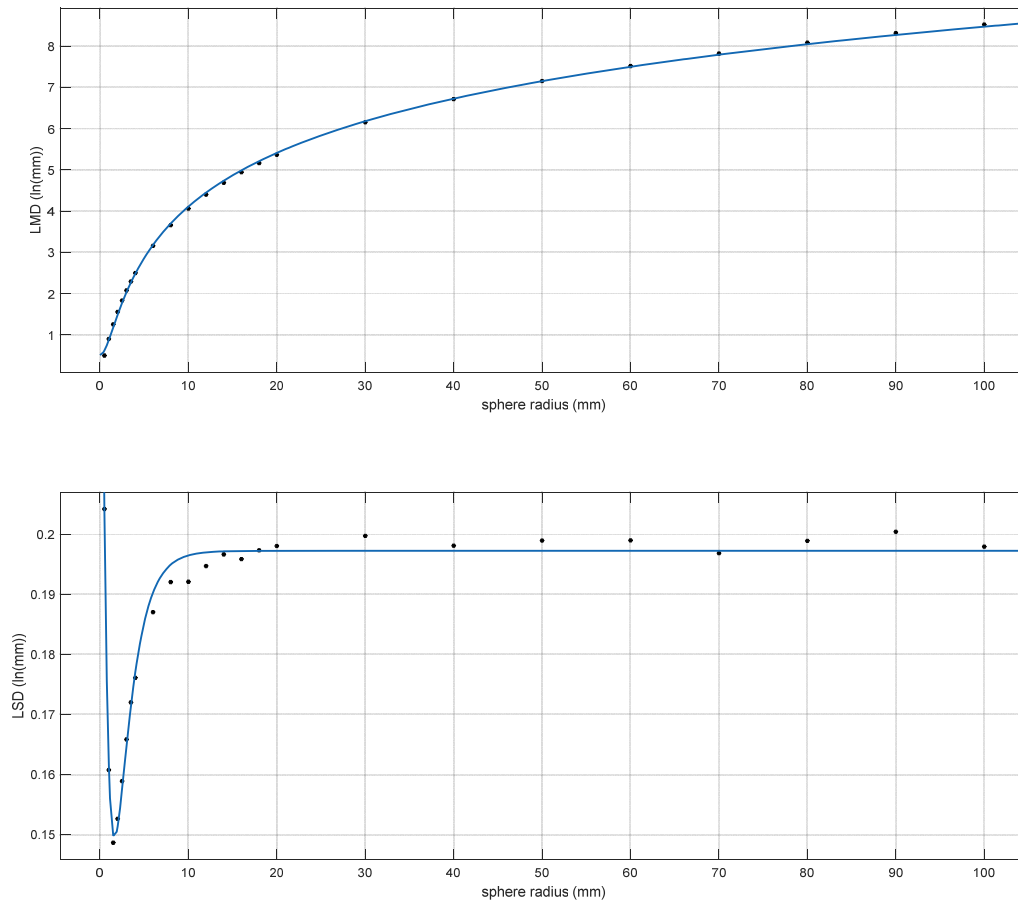


Figure 4.2 Data points describing the log-mean of the pathlength distribution (A) and the log-standard deviation of the pathlength distribution (B). The fitted curves are described by equations 4.3 and 4.4, respectively.

Using these formulas, it was possible to resurrect the path-length distributions expected for a given voxel to surface-point distance (see figure 4.2A and B). And then using the pathlength distribution information, it was in turn possible to determine what the photon intensity would be at the surface for a given linear attenuation coefficient. This was done by scaling (the bins of) the frequency distribution by the attenuation expected for the path-length (i.e. bin location). The result was then integrated to get the overall probability of a photon traveling that far, and these in turn were assumed to be distributed over the surface of a sphere of that radius.

4.5 Image Reconstruction Methods

For the purposes of this dissertation I made use of two different reconstruction algorithms, one direct and the other iterative. The direct solution was calculated using the Moore-Penrose pseudoinverse of the system forward model matrix W . Solutions to linear matrix problems (such as the one posed by BLT) when solved using the Moore-Penrose pseudoinverse are well known to produce a least-squares solution. A short proof of this assertion (adapted from [115]) and derivation of the pseudoinverse from a singular value decomposition of W , are as follows.

By the singular value decomposition theorem, any real valued matrix, W , can be factored into a product of three matrices, $W=USV^T$, where the columns of U and V are orthonormal (i.e. mutually orthogonal and of unit length) and S is diagonal with positive real entries (known as the singular values of W). Thus the inverses of U and V are equal to their transposes and the inverse of S is a diagonal matrix, S^+ , in which the elements of S have been replaced by their reciprocals. The pseudoinverse of W (commonly denoted as W^+), is defined as follows:

$$W^+ = (USV^T)^{-1} \quad (4.5)$$

which resolves to:

$$W^+ = VS^+U^T \quad (4.6)$$

In section 1.2 of this manuscript I defined the BLT system equation to be:

$$Y = WX \quad (4.7)$$

where X is the vector of unknown voxel source intensities and Y is the vector of measured spectral surface intensities. However, owing to the noise in Y it is no longer necessarily in the range of W and therefore this equality does not hold. So instead we seek a solution X_0 that minimizes the L2-norm of the residual (i.e. the least squares solution). In other words, we seek a specific X_0 which produces a smaller residual norm than any general X :

$$\|WX_0 - Y\| \leq \|WX - Y\| \quad (4.8)$$

This solution can be found using the Moore-Penrose pseudoinverse using a simple matrix multiply as demonstrated in the following:

$$WX - Y = WX - Y + WW^+Y - WW^+Y \quad (4.9)$$

which after rearranging and factoring becomes:

$$WX - Y = W(X - W^+Y) + (I - WW^+)(-Y) \quad (4.10)$$

then taking the norms (i.e. the lengths of the vector components) and applying the Pythagorean theorem:

$$\|WX - Y\|^2 = \|W(X - W^+Y)\|^2 + \|(I - WW^+)(-Y)\|^2 \quad (4.11)$$

and now defining X_0 calculated from Y using the pseudoinverse W^+ like so:

$$X_0 = W^+Y \quad (4.12)$$

and then substituting into 4.11 after distributing $(-Y)$:

$$\|WX - Y\|^2 = \|W(X - X_0)\|^2 + \|(WX_0 - Y)\|^2 \quad (4.13)$$

And since all the terms in 4.13 are positive, it can easily be seen that 4.8 holds true and thus that the X_0 calculated using the pseudoinverse is the least-squares solution. It is worth noting here that this method allows negative values in the solution for X , which of course is nonsensical when describing the intensity of a light source.

The second BLT image reconstruction algorithm I used was the maximum likelihood expectation maximization algorithm (MLEM). It constrains the solution to have only positive values and assumes that the noise in Y is Poisson distributed [116]. It is summarized by the following expressions:

$$X_n^{(k+1)} = \frac{X_n^{(k)}}{S_n} \cdot \sum_{j=1}^M \left[W_{n,j}^T \left(\frac{Y_m}{\sum_{i=1}^N W_{m,i} X_i^{(k)}} \right) \right] \quad (4.14)$$

where M is the number of measurements and N is the number of unknowns and where S_n is defined as:

$$S_n = \sum_{m=1}^M W_{n,m} \quad (4.15)$$

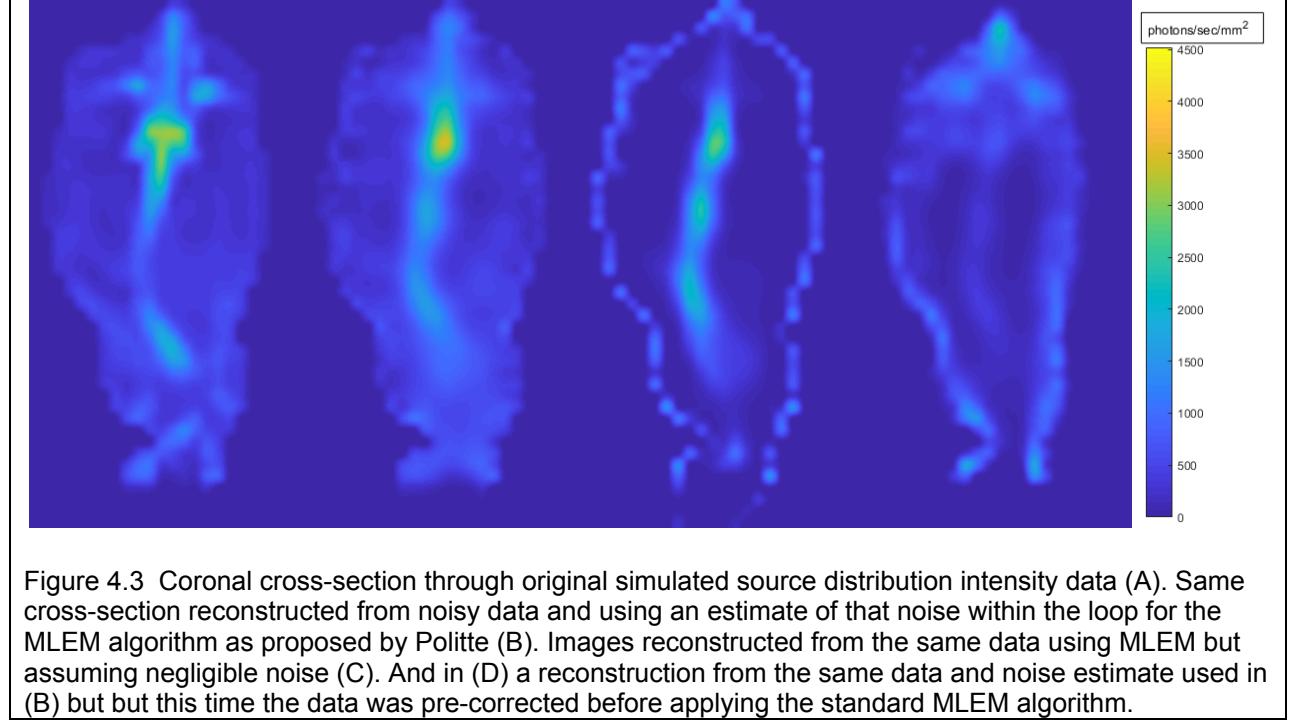
However, if Y is corrected for counts stemming from the CCD dark current, cosmic rays and other sources of background, by subtracting estimates of these counts from Y prior to entering into the MLEM algorithm, the noise is no longer Poisson distributed. This deficit was recognized and was addressed in the context of PET image reconstruction by Politte et. al. in 1991 [117]. The simple expedient Politte proposed to get around this problem was to add an estimate of the noise, σ_m , to the product WX within the loop of the iterations as shown in expression 4.16.

$$X_n^{(k+1)} = \frac{X_n^{(k)}}{S_n} \cdot \sum_{j=1}^M \left[W_{n,j}^T \left(\frac{Y_m}{\sum_{i=1}^N W_{m,i} X_i^{(k)} + \sigma_m} \right) \right] \quad (4.14)$$

In this way the uncorrected Y is used and retains its Poisson statistics.

The need for such an adjustment may not be well appreciated by some investigators making use of the MLEM in the context of BLT or it may be assumed to be a small effect, so I will take this opportunity to give an example of the magnitude of error when this adjustment is *not* made. This example here made use of the simulated mouse model which will be described in detail in a later section (4.12) of this manuscript.

Looking at the images of Figure 4.3 the importance of Politte's approach is readily apparent. The image on the left (4.3A) is a coronal cross-section through the original un-noised source distribution and is provided here a reference for the ground truth. The image calculated from 160 iterations of the MLEM algorithm and employing Politte's adjustments is shown in figure 4.3B. The image in Figure 4.3C shows the impact of assuming that the noise is negligible and *not* accounting for it within the loop. While Figure 4.3D shows what happens when Y is corrected for noise prior to entering the MLEM algorithm.



4.6. Derivation of the optimization expression

In this section I will derive a formula that will allow an investigator to determine the optimal distribution of acquisition times for a given set of filtered measurements. To do this I'll start by considering a vector with elements X_n describing the photon fluence rate for each voxel ($1 \leq n \leq N$, where N is the number of voxels) within an object (e.g. a mouse) being imaged within a bioluminescence imager. The expected fluence rate $Y_{m,j}$ measured by each of the detectors (e.g. each of M pixels, $1 \leq m \leq M$, within a CCD camera) at a given wavelength j ($1 \leq j \leq J$, where J is the number of wavelengths) acquired with a given band-pass filter within the bioluminescence imager, can be modeled as a matrix multiplication $Y_m = \sum_{n=1}^N W_{m,n} X_n$. Each wavelength will have its own weight matrix (a subset of the full matrix W), which I'll designate as W_j with

elements $W_{m,j,n}$. and each is acquired for a given duration, T_j , to yield a separate vector of photon counts C_j (with elements $C_{m,j}$) for each wavelength. Note here that in this definition each weight matrix W_j (for each wavelength) is accounting for the relative intensity of the light source at a given wavelength (i.e. for the source spectrum). Thus, for example, all other things being equal, if the relative amplitude at wavelength j for a given luciferase is double that of another luciferase, then W_j of the former will be double that of the latter.

Because C involves counting statistics, the uncertainty associated with each element of C is Poisson distributed and is equal to the square root of the counts, $\sigma_{C_{m,j}} = \sqrt{C_{m,j}}$. However, we are interested in the uncertainty in Y and ultimately its impact on the uncertainty in X . To get the uncertainty in Y it is necessary to divide the uncertainty in C by the acquisition time $\sigma_{Y_{m,j}} = \frac{\sigma_{C_{m,j}}}{T_j} = \frac{\sqrt{C_{m,j}}}{T_j} = \frac{\sqrt{Y_{m,j}T_j}}{T_j}$. There are of course other sources of noise in the measurement of C and I will incorporate these later, but for now I'll assume this simple Poisson noise model.

There are several means of solving for X given Y , but for the purposes here I'll start by making use of the Moore-Penrose pseudoinverse of W which when matrix multiplied by Y estimates X in a manner that minimizes its squared error. I'll designate this matrix as W^+ , which like W can also be partitioned into submatrices, W_j^+ , each corresponding to a different wavelength. Because matrix multiplication is a linear operation, when multiplying Y by W^+ , the error in Y is also multiplied by W^+ . However, uncertainties sum in quadrature and thus the expression for the overall uncertainty X , σ_X , is as follows:

$$\sigma_X = \sqrt{\frac{1}{N} \sum_{n=1}^N \sum_{m,j=1}^{M \cdot J} \left(W_{n,m,j}^+ \cdot \sigma_{Y_{m,j}} \right)^2} \quad (4.15)$$

which in turn expands to:

$$\sigma_X = \sqrt{\frac{1}{N} \sum_{n=1}^N \sum_{m,j=1}^{M \cdot J} \left(W_{n,m,j}^+ \frac{\sqrt{Y_{m,j} T_j}}{T_j} \right)^2} \quad (4.16)$$

Distributing the square and using the notation $\left(W_{n,m,j}^+ \right)^2 = W_{n,m,j}^{+2}$ gives:

$$\sigma_X = \sqrt{\frac{1}{N} \sum_{n=1}^N \sum_{m,j=1}^{M \cdot J} W_{n,m,j}^{+2} \frac{Y_{m,j}}{T_j}} \quad (4.17)$$

and finally – defining $Q_j = \frac{1}{N} \sum_{n=1}^N \sum_{m,j=1}^{M \cdot J} W_{n,m,j}^{+2} Y_{m,j}$ this simplifies to:

$$\sigma_X = \sqrt{\sum_{j=1}^J \frac{Q_j}{T_j}} \quad (4.18)$$

When the acquisition times for each wavelength are specified and there exists a reasonable estimate of $Y_{m,j}$, this expression provides an estimate of the overall uncertainty in X (a useful result in its own right), but it also provides the basis upon which it is possible to determine the optimal time distribution among a given set of filtered measurements (i.e. wavelengths). To demonstrate this, I'll start by considering just two wavelengths.

$$\sigma_X = \sqrt{\frac{Q_1}{T_1} + \frac{Q_2}{T_2}} \quad (4.19)$$

Now, define $T = T_1 + T_2$ and $f = \frac{T_1}{T}$ to get:

$$\sigma_X = \sqrt{\frac{Q_1}{fT} + \frac{Q_2}{(1-f)T}} \quad (4.20)$$

Based on this expression we will seek the value of f which minimizes σ_X and start by noting that squaring both sides does not affect the optimal f value. Similarly the value of T is inconsequential.

$$T^2 \sigma_X^2 = Q_1 f^{-1} + Q_2 (1-f)^{-1} \quad (4.21)$$

Now, taking the derivative wrt f , setting the result to zero and rearranging:

$$Q_1 f^{-2} = Q_2 (1-f)^{-2} \quad (4.22)$$

$$\frac{Q_1}{Q_2} = \frac{f^2}{(1-f)^2} \quad (4.23)$$

and now taking the square root of both sides:

$$\frac{\sqrt{Q_1}}{\sqrt{Q_2}} = \frac{f}{(1-f)} \quad (4.24)$$

and putting back T_1 and T_2

$$\frac{\sqrt{Q_1}}{\sqrt{Q_2}} = \frac{T_1}{T_2} \quad (4.25)$$

The expression for f becomes:

$$\frac{\sqrt{Q_1}}{\sqrt{Q_1} + \sqrt{Q_2}} = f \quad (4.26)$$

This can be extended to more than two wavelengths to arrive at the final expression for determining the optimal time distribution:

$$\frac{\sqrt{Q_j}}{\sum_{i=1}^J \sqrt{Q_i}} = f_j \quad (4.27)$$

As promised earlier, I'll now consider the additional complication posed by other sources of noise inherent to the measurement of Y . For a CCD based system this primarily comes in the form of a time dependent dark current, D , and a time independent read noise, R . I'll assume here that both of these sources of noise are uniform across all detectors and wavelengths (but note that the following could also easily be adjusted for non-uniform noise). In any case, assuming uniform noise, the expression for the uncertainty in each element of Y is:

$$\sigma_{Y_{m \cdot j}} = \frac{\sqrt{(Y_{m \cdot j} + D)T_j + R}}{T_j} \quad (4.28)$$

Then following the same sequence of manipulations used previously:

$$\sigma_X = \sqrt{\frac{1}{N} \sum_{n=1}^N \sum_{m \cdot j=1}^{M \cdot J} \left(W_{n, m \cdot j}^+ \frac{\sqrt{(Y_{m \cdot j} + D)T_j + R}}{T_j} \right)^2} \quad (4.29)$$

$$\sigma_X = \sqrt{\frac{1}{N} \sum_{n=1}^N \sum_{m \cdot j=1}^{M \cdot J} W_{n, m \cdot j}^{+2} \frac{Y_{m \cdot j} + D + \frac{R}{T_j}}{T_j}} \quad (4.30)$$

and here adding definitions for $Q_D = \frac{D}{N} \cdot \sum_{n=1}^N \sum_{m \cdot j=1}^{M \cdot J} W_{n, m \cdot j}^{+2}$ and $Q_R = \frac{R}{N} \cdot$

$\sum_{n=1}^N \sum_{m \cdot j=1}^{M \cdot J} W_{n, m \cdot j}^{+2}$ to get:

$$\sigma_X = \sqrt{\sum_{j=1}^J \frac{Q_j + Q_D + \frac{Q_R}{T_j}}{T_j}} \quad (4.31)$$

$$\sigma_X = \sqrt{\frac{Q_1 + Q_D + \frac{Q_R}{T_1}}{T_1} + \frac{Q_2 + Q_D + \frac{Q_R}{T_2}}{T_2}} \quad (4.32)$$

$$\sigma_X = \sqrt{\frac{Q_1 + Q_D + \frac{Q_R}{fT}}{fT} + \frac{Q_2 + Q_D + \frac{Q_R}{(1-f)T}}{(1-f)T}} \quad (4.33)$$

This expression can readily be minimized using a nonlinear search but can also be more quickly and robustly minimized by noting first that if R is assumed to be negligible for a moment and defining \hat{Q}_j to include Q_D , i.e. $\hat{Q}_j = Q_j + Q_D$, then we get the same solution (4.27) as before. This can be used as the initial estimate of the optimal time distribution –

$$f_j^{(1)} = \frac{\sqrt{\hat{Q}_j}}{\sum_{i=1}^J \sqrt{\hat{Q}_i}} \quad (4.34)$$

but then iterated several times now incorporating a term accounting for the read noise.

$$f_j^{(k+1)} = \frac{\sqrt{\hat{Q}_j + \frac{Q_R}{f_j^{(k)}}}}{\sum_{i=1}^J \sqrt{\hat{Q}_i + \frac{Q_R}{f_i^{(k)}}}} \quad (4.35)$$

In practice, iterating this expression 10 or more times converges to greater than 4 significant digits. In all of the following simulations, for good measure the expression was iterated 20 times.

Incorporating the additional sources of noise into expression (4.18) gives the final expression predicting the overall image noise (root mean squared error):

$$\sigma_X = \frac{1}{\sqrt{T}} \sqrt{\sum_{j=1}^J \frac{Q_j + Q_D + \frac{Q_R}{T_j}}{f_j}} \quad (4.36)$$

As mentioned in the Introduction to this specific aim, the above result works perfectly well for least-squares solutions but does not work for MLEM solutions. However, returning to equation (4.16) and substituting Barrett's U matrix gives (note ellipses are used to emphasize that $\ddot{\sigma}_X$ is log-normal distributed):

$$\ddot{\sigma}_X = \sqrt{\frac{1}{N} \sum_{n=1}^N \sum_{m,j=1}^{M \cdot J} \left(U_{n,m,j} \frac{\sqrt{Y_{m,j} T_j}}{T_j} \right)^2} \quad (4.37)$$

All the manipulations (4.17) through (4.36) apply equally to this expression but substituting $U_{n,m,j}$ for $W_{n,m,j}^+$ and $\ddot{\sigma}_X$ for σ_X .

4.7. Algorithm to select optimal filters

The calculations described above allow for the determination of the optimal distribution of time among a set of measurements given that the filters for those measurements had already been chosen. These same equations, however, can also be employed to make an optimal selection of filters within a specified range of wavelengths using the following algorithm (see Appendix for Matlab code):

- 1) Start by selecting the range of wavelengths. Generally this will be a range spanning the luciferase source spectrum, but in the case of a Cerenkov source

the range is only limited by the sensitivity of the CCD and attenuation of the tissue.

- 2) Divide the range of wavelengths into discrete bands. The number of bands, J , here is only limited by computational capacity. For the geometry of interest, solve the forward model for each of the bands thereby determining the weight matrix, W_j , for each wavelength band. Again note that the magnitude of each W_j here incorporates the relative magnitude of the source spectrum at that wavelength.
- 3) Select an overall duration to be allotted to the entire acquisition and use expressions (4.34) and (4.35) – or their equivalents in the case of a planned MLEM reconstruction - to determine the optimal distribution of times among the current set of filters. Use equation (4.36), or (4.37) as appropriate, to estimate the expected degree of noise in the solution. Use this noise level as the current minimum.
- 4) Then, independently for each adjacent pair of wavelengths, sum the corresponding W_j matrices and recalculate U (or W^+). Based on this new U (or W^+), recalculate the optimal distribution and re-estimate the new noise level. Once this is done for all adjacent pairs, take the pairing having the minimum expected noise level and compare it to the current minimum. If greater than the current minimum, stop. Otherwise accept the summed pair of wavelengths having the minimum expected noise and repeat step 4, recalculating and testing the newly adjacent pairs.

4.8. Testing the optimization expression

Owing to correlations between the system matrix and the errors in any given measurement, there is in fact no single time distribution that will for all samplings produce a solution with minimal noise. Instead, the optimality of the time distribution only becomes apparent when considering the *expected* noise level (i.e. the noise level averaged over many samplings). Moreover, in order to get a gold standard reference based upon which the true optimal distribution can be assessed, it is necessary to calculate the expected noise-level for all possible time distributions. Because of compute limitations, this is simply impractical for a realistic, highly sampled object (e.g. mouse) with many wavelengths. So instead, I sought to verify the optimization calculations making use of a very small toy system involving just 3 internal voxels (i.e. X having just 3 elements) and 5 detector elements (i.e. Y with 5 elements). Thus each W_j is a 5x3 matrix and each element of W_j was calculated as $s_j e^{-\lambda_j d}$ where d is a random distance between 0 and 10 mm, and λ_j and s_j are the wavelength's attenuation coefficient and relative spectral intensity (of the photon source), respectively.

Using this simple system, it was possible to simulate photon transport and detection, reconstruct images and calculate the expected root-mean-squared error (RMSE) when seeking a least squares solution to the image reconstruction problem or calculate the expected root-mean-squared log error (RMSLE) when using an MLEM reconstruction. The means of these errors were determined based on 10,000 acquisitions with independently sampled random noise. The noise was added using Matlab's `poissrnd` function which takes as its argument the mean of the desired noise distribution. This

was specified as $(W_j * X + D)f_j T + R$. The dark current and read noise parameter values, D and R respectively, were set based on the IVIS 200 specifications ($D=0.009787$ cnts/sec/pixel and $R=1.995$ cnts/pixel).

A cohort of 10,000 simulated acquisitions was repeated for each of 25 different time distributions, $f_1 \in \{0.02: 0.04: 0.98\}$, in a two wavelength system. This was then repeated another $25 \times 25 = 625$ times (same $f_1 \in \{0.02: 0.04: 0.98\}$ but the remainder of the time split between f_2 and f_3 with $f_2 \in \{0.02: 0.04: 0.98\}$) in a 3 wavelength system. Among all the time distributions tried, the distribution resulting in lowest expected (i.e. averaged over 10,000 trials) RMSE or RMSLE was identified. This is the *brute force determined optimal time distribution*. The RMSE and RMSLE error level for the *analytically determined optimal time distribution* (as determined using the iterative procedure described by expression (4.35) was also calculated once again averaging over 10,000 noise samplings. This was also repeated for an *analytically determined optimal time distribution* based on a uniform source distribution (i.e. where it was assumed that there was no knowledge of the distribution of X) and finally for a naïve *uniform time distribution*. In addition, for each simulation a prediction of the noise level was made using expressions (4.36) and (4.37) and these were compared to the measured mean noise level.

For the two-wavelength system, the entire process was repeated 100 times, each time using a different randomly chosen source spectrum, set of distances, source intensities, attenuation coefficients and overall acquisition time (between 1 and 36,000 seconds). For each of these 100 repetitions the percent error between the brute-force and metric

predicted error levels was calculated and the mean, standard deviation and the maximum of the percent errors was determined. Similarly, the three-wavelength system simulations were repeated but owing to the significant additional computational burden this was done just 10 times.

4.9. Two-wavelength LSQ simulation results

Monte-Carlo simulations were undertaken to demonstrate definitively the accuracy with which the calculations derived here are able to determine the optimal distribution of times among the filtered acquisitions. Two equations were investigated: 1) appropriate for least-squares solutions to the inverse problem and making use of the Moore-Penrose pseudoinverse and; 2) appropriate for MLEM solutions to the inverse problem and making use of Barrett's error estimating matrix. MLEM and other constrained solutions are of more practical utility but because Barrett's U matrix involves some approximating assumptions, it can fail when those assumptions are violated.

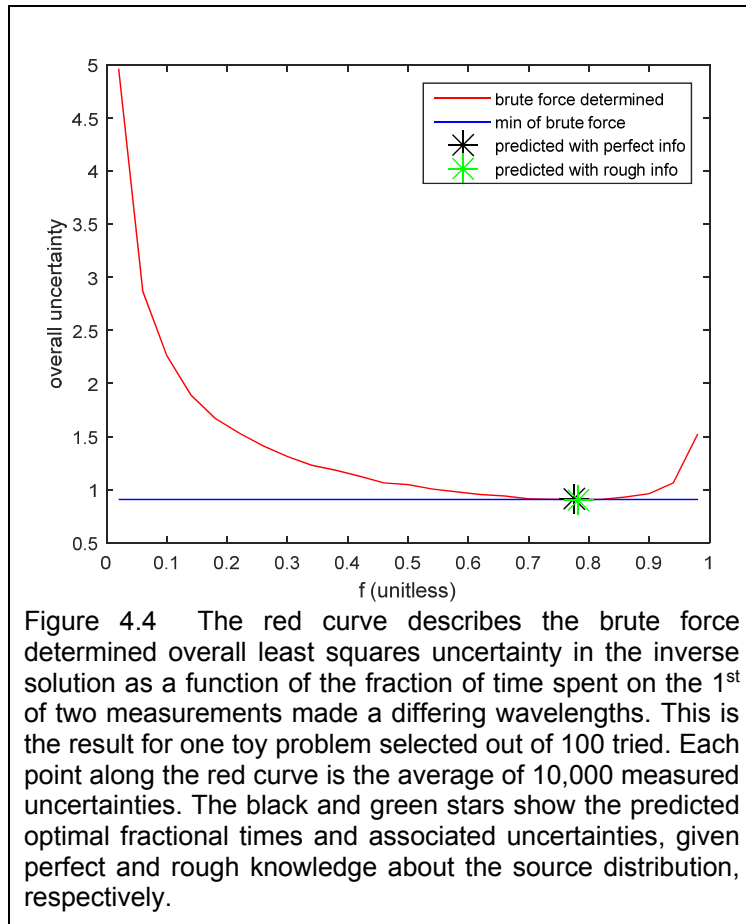


Figure 4.4 The red curve describes the brute force determined overall least squares uncertainty in the inverse solution as a function of the fraction of time spent on the 1st of two measurements made at differing wavelengths. This is the result for one toy problem selected out of 100 tried. Each point along the red curve is the average of 10,000 measured uncertainties. The black and green stars show the predicted optimal fractional times and associated uncertainties, given perfect and rough knowledge about the source distribution, respectively.

Unconstrained least-squares solutions for the BLT problems investigated here, on the other hand, generally produce uselessly noisy images, however, the Moore-Penrose calculations should be exact in every instance and hence the reason for their inclusion here.

Each “toy” two-wavelength system was simulated 10,000 times (differing only in noise) at each of 20 different time ratios between

the two measurements. One hundred such toy systems were simulated. Figure 4.4 shows the results from one of these. Each point along the red line shows the average of 10,000 calculations of the RMSE relative to the true voxel intensities at a given relative duration of measurement for the first wavelength. The X-location at which this curve reaches its nadir is the brute force determined optimal fraction of time that should be spent measuring the first filtered image given the parameters (source spectrum, attenuation, geometry, etc.) of this specific toy system. The horizontal blue line shows the level of that minimum. The two stars, show the optimal fractional time determined using equation (4.36) when the true distribution is known perfectly (black star) and when just rough information regarding the overall signal magnitude (assumed to be distributed

uniformly) is available (green star) for the calculation. In practice, the true source distribution will not be known in advance, so the green star shows more realistically what can be achieved in practice. Overall, these results demonstrate that the formulas derived in section 4.6 are consistently able to find the optimal distribution of acquisition times between the two filtered measurements.

Figure 4.5 A and B show a comparison of the predicted and actual (i.e. measured) mean RMSE uncertainties at the calculated optimal time distribution for each of the 100 toy systems modeled, given perfect and rough information about the source distribution, respectively. These results show that the equation I derived predict the image noise almost perfectly.

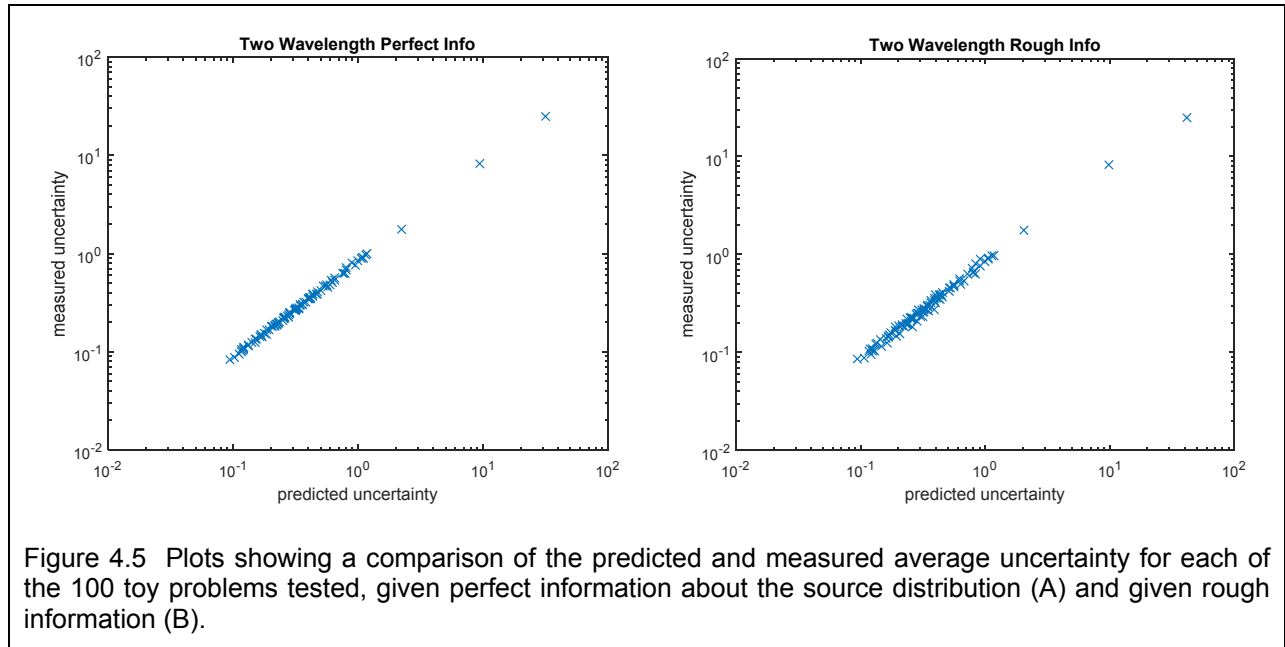
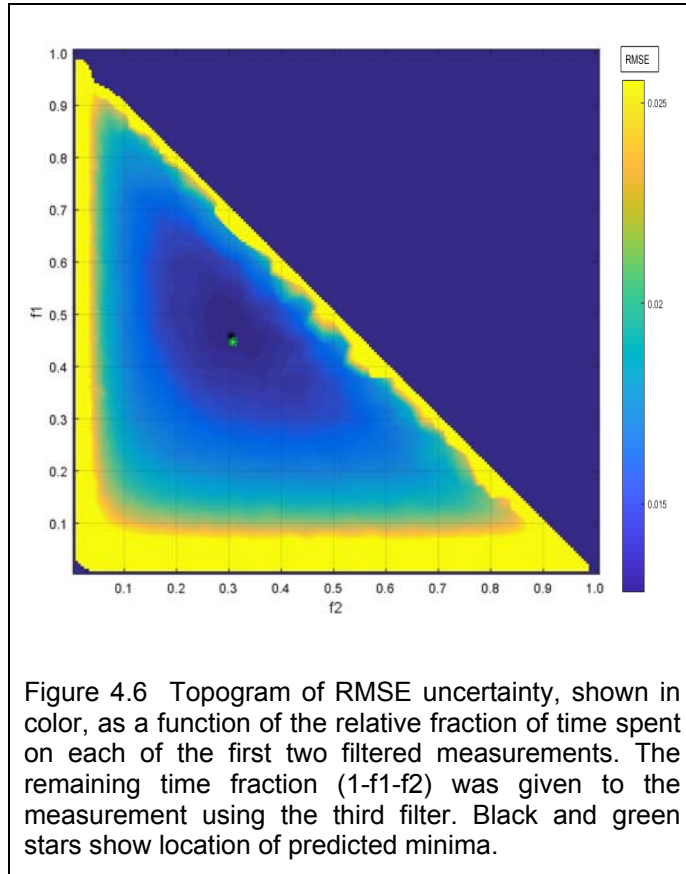


Figure 4.5 Plots showing a comparison of the predicted and measured average uncertainty for each of the 100 toy problems tested, given perfect information about the source distribution (A) and given rough information (B).

4.10. Three-wavelength LSQ simulation

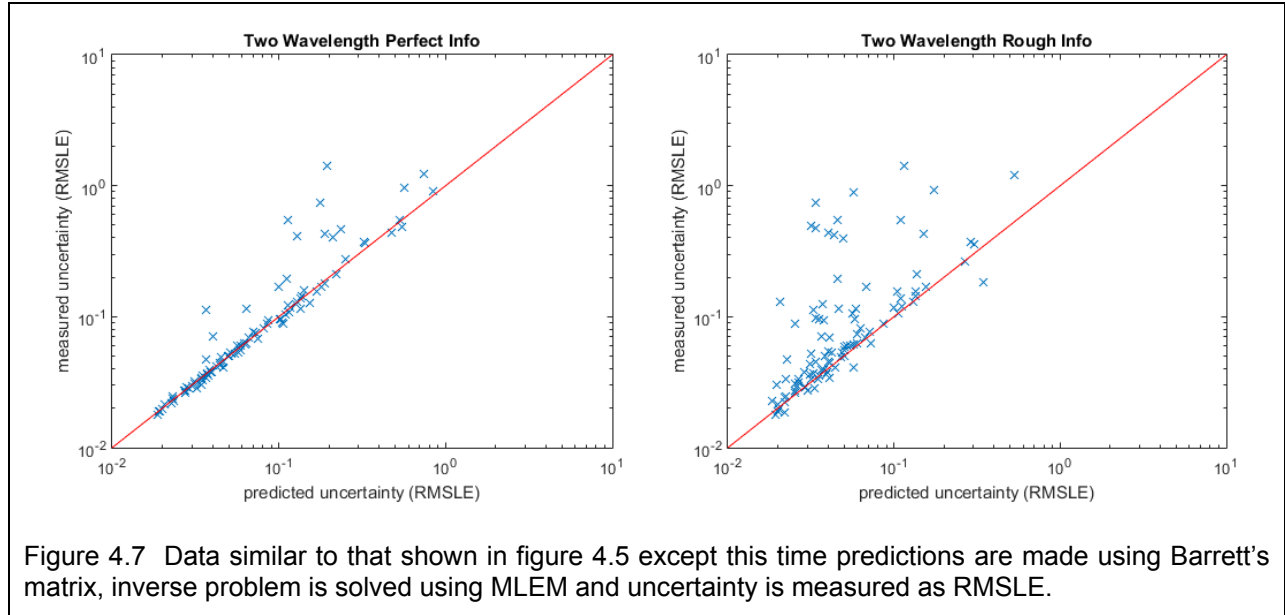


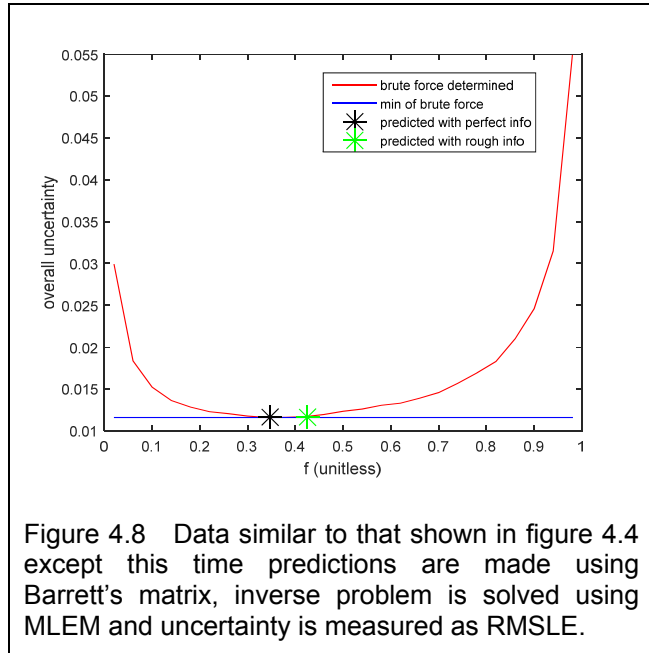
Given how equation (4.36) was derived, I had some concern that the extrapolation to more than two wavelengths might be problematic. Therefore, a similar set of brute-force Monte-Carlo calculations of the optimal time distribution were undertaken, however, this time involving three wavelengths. In this case only ten toy systems (each parameterized by a different set of random values) were modeled, but

each tried at 125 different time distributions, averaged over 10,000 runs (1,250,000 simulations altogether). The resultant 125 points in 3D (fraction 1 by fraction 2 by RMSE) was fitted to a surface using Matlab's *griddata* function. This surface is displayed as a colorized 2D image in figure 4.6, with RMSE represented by the pixel color. The black and green stars again show the optimal time distribution determined by equation (4.36) given perfect and rough information, respectively. This result shows that the accuracy of the predictions demonstrated previously for the two-filter case (figure 4.4), also extend to a larger number of filters.

4.11. Two-wavelength MLEM simulation

As mentioned previously, the intent in showing the above results for an unconstrained LSQ solution is to demonstrate that the overall approach works precisely. As such, they also serve to demonstrate that the occasional poor performance of the MLEM targeted optimization seen in the following results, are therefore due to approximations implicit to the calculations of Bartlett's U matrix, not because of problems or inaccuracies in the optimization per se.





Figures 4.7 A and B mirror the results shown in figures 4.5 A and B except this time the optimal time distribution predictions were based on Bartlett's U matrix, the images were reconstructed using an MLEM algorithm and the uncertainty was measured in RMSLE. Similarly figure 4.8 mirrors figure 4.4 again depicting a selected result out of

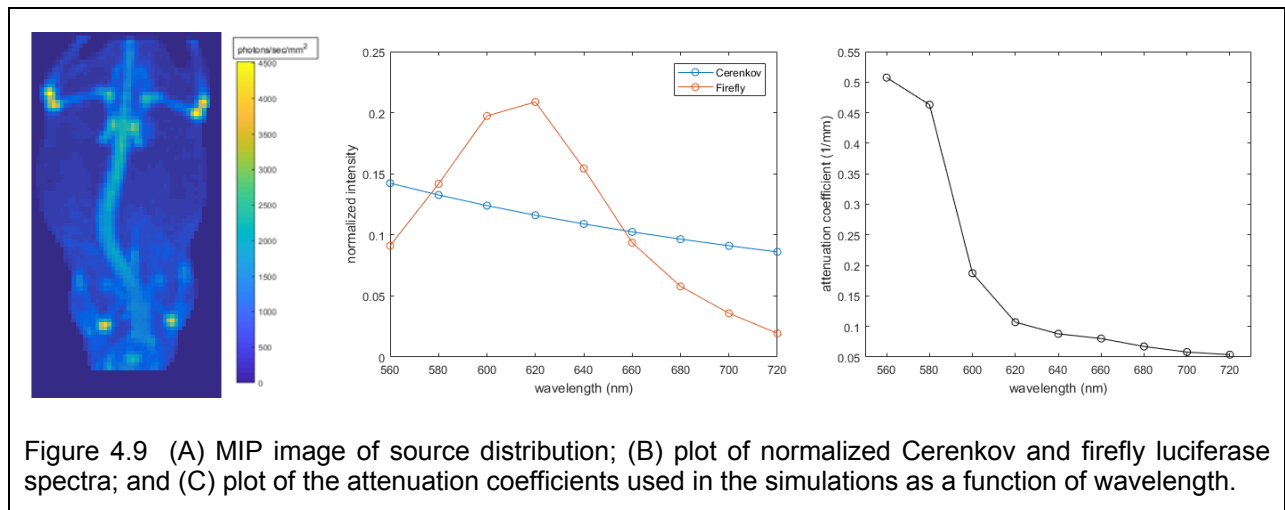
the 100 toy systems simulated. Never-the-less, it can be appreciated from these plots that the utility of the expression derived for the least-squares solution case, also extend to MLEM reconstructed images.

4.12. Mouse simulations

Although the toy systems used to test the optimization calculations were designed to mimic the salient characteristics of real BLT problems, it can be argued that they do not reflect the potential complications of a full sized system encountered in the preclinical setting. Therefore, I created a model of light propagation and detection based on a set of CT and PET images of an actual live adult nude mouse, co-registered using the procedures described in Specific Aim 1. The CT was used to define the geometry of the mouse and the PET images were used to define the source distribution of the light emanating from within the mouse based upon the predictions of the Cerenkov models

described in Specific Aim 2. This source distribution was then converted to light source intensities for individual wavelengths assuming both Cerenkov and firefly luciferase source spectra. The PET tracer that was used was ^{89}Zr oxalate which is known to uptake into the bones. A maximum intensity projection (MIP) of the PET image data is shown in figure 4.9A. The Cerenkov and firefly source spectra are shown in figure 4.9B.

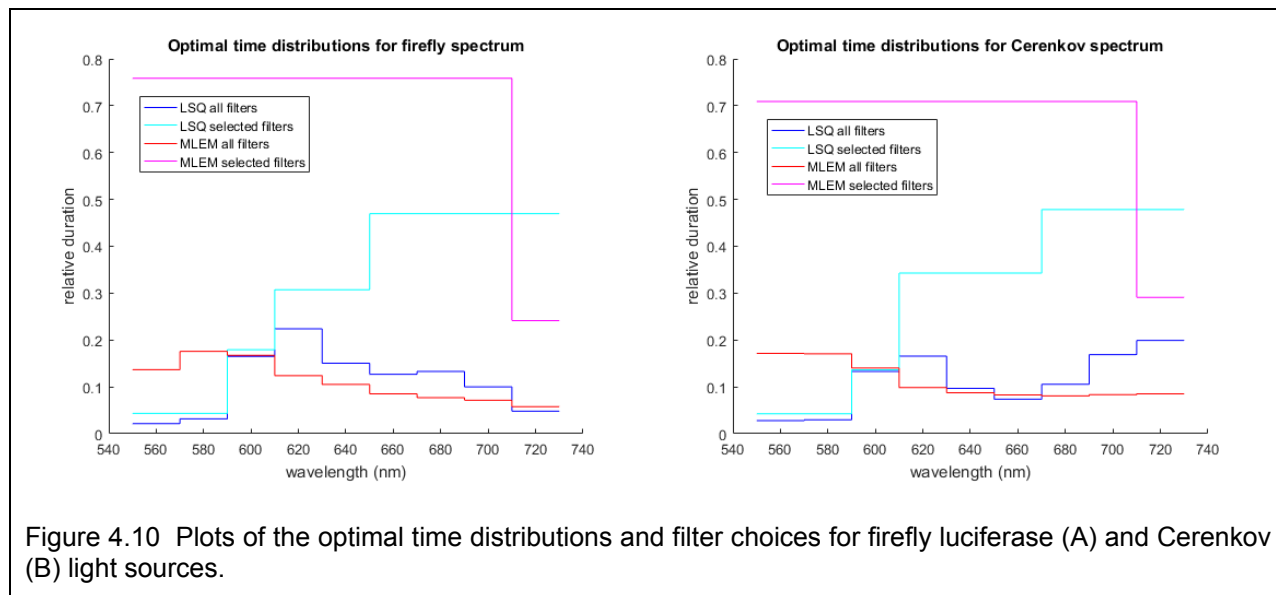
The propagation of the light through the tissues of the mouse was modeled using the equations described in Section 4.4 and using the wavelength dependent attenuation coefficients shown in figure 4.9C.



Based on this model, the optimal time distribution among eight 20 nm bandpass filters ranging from 560 to 720 nm, was determined assuming either Cerenkov or firefly luciferase spectra and either MLEM or LSQ reconstructions, four optimizations of this type in all. A similar set of optimizations was again run for these four conditions, however, this time the algorithm was allowed to optimally combine filters together mimicking an acquisition of the same overall duration but involving some filters having a

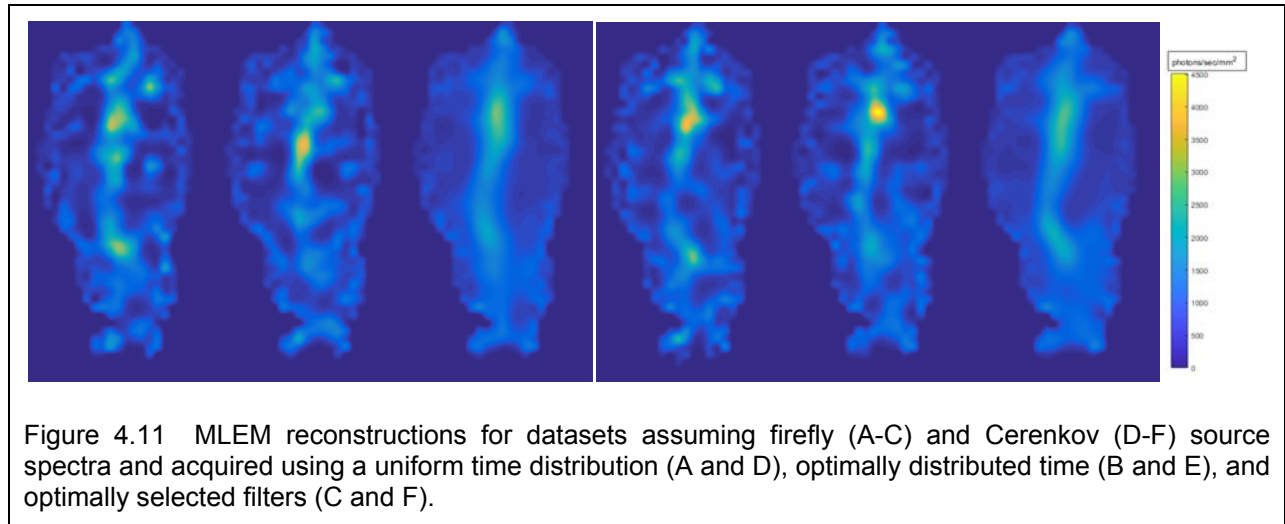
larger bandpass. The results of these optimizations are summarized in figures 4.10A (firefly) and 4.10B (Cerenkov).

From this data it is clear that the optimal acquisition protocol depends on what image reconstruction algorithm is to be used. This is not something that was considered in previous time optimization efforts [31, 32, 118]. It is also interesting to note that the filter selection process tended to prefer the additional count efficiency provided by the wide bandpass filters over the additional information provided by more wavelengths, in the MLEM case reducing the number of filters down to just two. Indeed, as we'll see below, quite accurate reconstructions can be generated using just the two wavelengths indicated.



Each of the optimized image acquisition protocols was then simulated assuming the PET derived source distribution described previously. A naïve uniform time acquisition protocol was also simulated. The total acquisition time was taken to be one hour. Coronal cross sectional images based on MLEM reconstructions of data acquired using

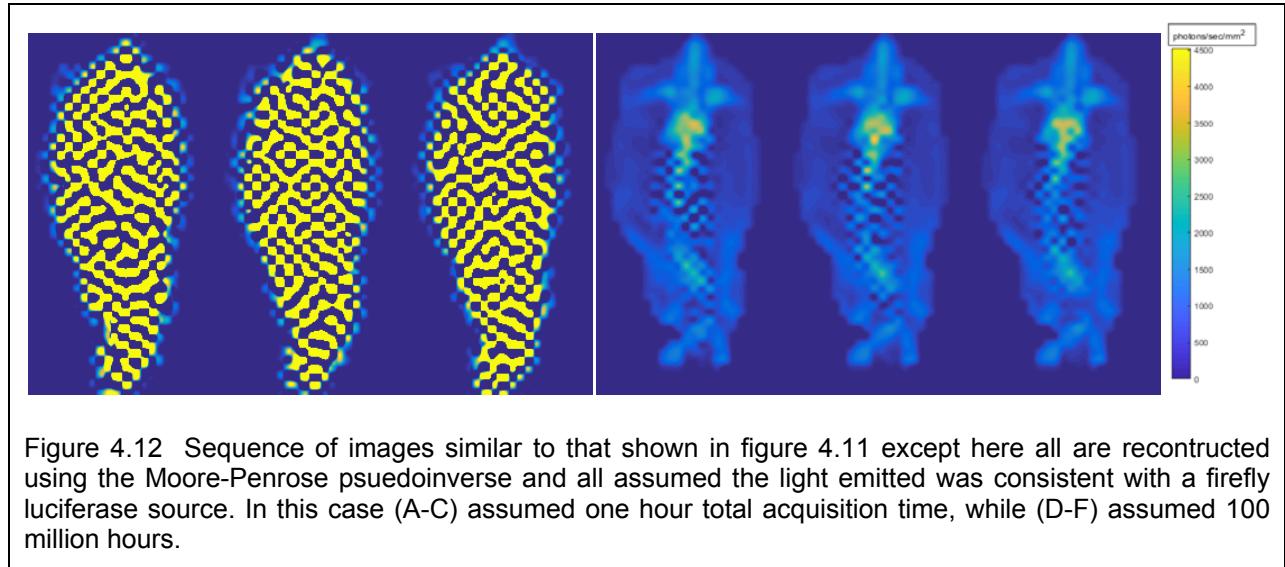
these acquisition protocols for the firefly luciferase source spectrum are shown in figure 4.11A-C for the uniform, optimal all filter, and optimal reduced filter protocols, respectively. The corresponding RMSLE values were 3.55, 3.27 and 0.71, respectively. Significant improvement in image quality is readily apparent when using the optimized data acquisition protocols.



An equivalent set of images, this time simulated using the Cerenkov source spectrum but again using MLEM reconstruction, is shown in figures 4.11D-F. The corresponding RMSLE values in this case were 4.40, 4.06 and 0.98. As can be appreciated when viewing these images, there is some improvement when using the all-filter optimally distributed acquisition time protocol relative to a naïve uniform time protocol, but this improvement is small, largely because uniform time is already close to optimal for these filters. A much larger improvement is gained when the number of filters is reduced and the bandwidths are extended.

At this same overall acquisition duration, the LSQ reconstructed image sets are so noisy as to be rendered useless (see figure 4.12A-C). The improvement gains for the

optimized protocols, never-the-less, can still be appreciated from the reductions in RMSE: 17598, 12805 and 11983 for the uniform, optimal full and optimal reduced filter sets, respectively.



In order to actually visually perceive an improvement in LSQ reconstructed images, it was necessary to simulate an acquisition that is on the order of 100 million times longer (see figure 4.12D-F). Even here the improvement is subtle at best but is confirmed by the RMSE values which in this case were 1.66, 1.42 and 1.13. Of particular note in these images is the extreme heterogeneity in the spatial distribution of the noise, wherein the center of the mouse is all but obliterated by noise but voxels near the surface are well resolved. In the next section, we will explore this further.

4.13. Guidance for improved conditioning and SNR through optimized sampling

As described previously, in luminescence tomography each column of the system matrix W specifies the surface profile of a given voxel and each row reflects the voxel

domain contributing to a given pixel on the skin surface. Reducing the number of columns is in effect a constraint on the solution space (i.e. domain). It can also be considered a type of preconditioning in that the intent is to transform matrix W into a new matrix having more favorable properties for iterative solution.

When seeking to optimize the time distributions, the target function that was minimized was the overall uncertainty in the image, summing the uncertainties in the individual voxels in quadrature. If instead we leave the voxel uncertainties separate, these values can be used to guide a variety of decisions including whether a given pair or group of voxels should be combined into a cluster and treated as a single large voxel. Combining voxels entails combining columns of the system matrix W and thus is a type of conditioning, wherein the condition number is improved (reduced).

The image in figure 4.13A shows a coronal cross-section through an image of these voxel uncertainties, in this case for an LSQ solution assuming a uniform source distribution. Given some estimate of the absolute intensity of a light source placed within a given voxel within this image, this information could be used to ask the very reasonable question, will it be possible to distinguish this voxel from its neighbors? And when the answer is no, this voxel and one or more of its neighbors can be combined.

A constraint of this type differs from other previously proposed domain constraint approaches in two major ways: 1) the changes to matrix W proposed here would be limited to spatially adjacent voxels, whereas most preconditioners manipulate the matrix without regard to this physical spatial context; 2) these matrix manipulations take into

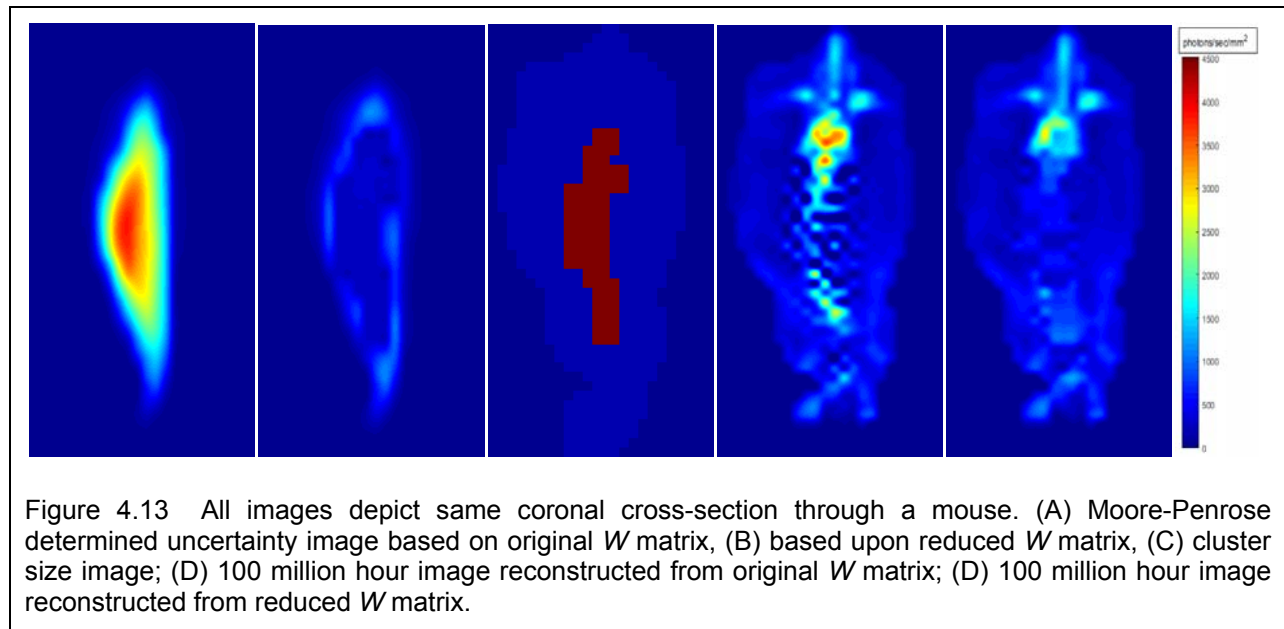
consideration a noise model and thus they are guided based on statistical considerations.

In solving the inverse problem, a single source intensity would be calculated for each cluster (i.e. the voxels of a cluster are assigned the same source intensity). The net effect is that the spatial domain within the animal is sampled non-uniformly in a manner that reflects the depth dependent variation in achievable resolution.

Clustering voxels based on this information could be handled in a number of differing ways, including algorithms that would place topographical constraints on the clusters. For the purposes here, I have implemented just one very simple approach but others are possible. The algorithm that I implemented (see Appendix for details) starts by assigning each voxel to its own cluster, determines the associated predicted noise level to that cluster and identifies the initial 26 neighboring clusters. It then makes multiple passes through the cluster data, each time selecting the cluster with the greatest uncertainty. And if that uncertainty is greater than a specified threshold (chosen based on the aforementioned statistical considerations), the algorithm combines that cluster with all of its immediate neighbors. When voxels are clustered together, their corresponding columns in W are averaged together.

The results from the application of this algorithm can be seen in figure 4.13. Figure 4.13B shows the cluster uncertainties for same cross-section as in 4.13A but after applying the clustering algorithm. Figure 4.13C is an image of the cluster sizes. Blue depicts voxels belonging to clusters having a single voxel member and red depicts voxels belonging to clusters having 27 voxel members (although in principle clusters

having other numbers of voxels are also possible). The original system matrix, W , had a condition number (as calculated by Matlab's `cond` function) of $9.7e6$. After combining the columns as directed by the clustering algorithm, the condition number was reduced over three-fold to $3.0e6$. The images in figures 4.13D and E show the resultant reconstructions when using the original and the post clustering system matrices, respectively. Here it can be appreciated that the noise within the central region of the mouse has been greatly reduced by the clustering, albeit at a loss of resolution in this region.



Reducing the rows of W could follow a similar process but for somewhat different purposes. Generally speaking, reductions in the rows of W would not be expected to improve (and in fact could worsen) the condition number. However, changes in the row sampling can also improve the overall noise characteristics of the system, resulting in improved solutions in addition to reduced computational burden. For example, for some CCD cameras using a reduced resolution mode (i.e. charge binning) can improve the

signal to noise ratio beyond what one would expect from a simple averaging of pixel groups. However, demonstrations of the improvement that might be gained from these types of manipulation will not be pursued here.

4.14. Discussion

Through a series of simulations, I have demonstrated a novel means by which it is possible to make accurate predictions of the optimal acquisition time distribution for a multispectral bioluminescence tomography measurement. Moreover I show for the first time that this optimal distribution is dependent, not only upon the source spectrum, but also upon the algorithm that will be used to reconstruct the images. These simulations also suggest, however, that optimally distributing the acquisition time (in most cases) will not result in dramatic noise reductions relative a uniform time distribution protocol, at least not for filter sets that constitute reasonable samplings of the source spectrum (i.e. when all filters are within the spectral peak). The reason for this can best be appreciated by noting the broad shallow basin in the brute-force determined uncertainty curves of figures 4.4 and 4.8. In these and in the other simulated systems not shown, the fold change in uncertainty when going from the optimal to uniform time is quite small. However, I also show that by making use of the same calculations to optimally select the filters and their bandwidths, significant reductions in noise can be achieved. And, once again, the optimal filter selection is shown to be dependent upon both the source spectrum and the choice of reconstruction algorithm. I also argue that this same basic approach can be used to guide other acquisition protocol design choices, including specifically the distribution of the sampling frequency of both the solution space within the animal and of the measurements made of the light emanating from the animal

surface. And finally I demonstrate that in the former case, such guided decisions can lead to improved image quality.

5. OVERALL SUMMARY AND CONCLUSIONS

From the outset of this work it has been my intent to make progress on what I perceived to be *the* major problems limiting the successful implementation of bioluminescence tomography in the preclinical small animal imaging setting. Specifically, I felt that the lack of a gold standard reference against which improvements could be gauged and poor data quality/system conditioning issues, were the major problems hampering progress in BLT. Moreover, I recognized that there was a degree of synergy between these two problems. Establishing a gold standard reference of the 3D source distribution would require registrations, projective transforms, corrections and quantitative calibrations that were all requisite components of the platform upon which optimizations to reduce noise-levels and methodologies to guide system conditioning could be based. For example, the projective transform and registration to CT 3D space determined in Aim 1, are also steps used in the image reconstructions of Aim 3. The corrections for the luciferin time-course and light falloff as a function of the angle of the surface normal, are critical in maintaining the consistency of the measurements that would otherwise introduce artifacts into the reconstructed images independent of the noise in the data. Similarly, the calibration, filter sensitivity and source to camera distance corrections described in Aim 2, improve the accuracy and integrity of the data. It is only after the data is made accurate and consistent by correcting for all confounds (steps pursued in Aims 1 and 2), that it then it becomes meaningful to try to address questions regarding the precision of the data and how the resultant uncertainty propagates through the reconstruction process (the topic of Aim 3).

Corrections for confounds in most imaging systems would typically be expected to be applied automatically by the camera's hardware and associated reconstruction software (though I've identified several that are not). However, choices impacting precision are often left to the user. This perhaps is especially true of BLT, where the user needs to decide how long to image, what filters to use, how to distribute the acquisition time among the chosen filters, what resolution the CCD images should be acquired at and what voxel sizes to use (or what smoothing parameter values to apply) during the reconstruction. To-date, there has been little-to-no guidance for the user to help them make these decisions. Although there is certainly more to be done, this was one of the main purposes behind the work in Aim 3.

In Aim 3, I demonstrated that it is possible to determine in advance the optimal filter selection and acquisition time distribution to minimize the uncertainty in the reconstructed images. The equations derived to make these calculations showed, somewhat counter intuitively, that (roughly speaking) less time should be spent acquiring weaker signals and more time spent acquiring stronger signals. I've also demonstrated (to my knowledge, for the first time) that the optimal filter and time selections depend upon what reconstruction algorithm will be used to generate the images. Previous efforts had not considered the potential impact of this choice.

Many of the 3D tomographic imaging modalities have difficulty getting measurement data from locations deep within the subject being imaged. However, I think it is safe to say that for BLT this problem is extreme. Based on the data in figures 4.2A and 4.9C it can be readily seen that of the photons emanating from a depth of just 1 cm, only 5% of

the 660 nm (red) photons will make it out of the animal (much less being detected), while virtually none of the blue or green photons will make it out. This coupled with the greater degree of scatter experienced by the more deeply sourced photons, means that the achievable resolution at depth is greatly limited. One can conclude therefore, that while it may be reasonable in many 3D imaging modalities to use a uniform grid to sample the solution space (i.e. the same voxel spacing for both deep and shallow locations), this is not at all the case for BLT. Within Aim 3 I describe what I think could be the paradigm upon which these and related decisions can be made, whether it's the choice of voxel size, or selection of an appropriate basis function, the optimization of a regularization parameter value.

In future work, I hope to build upon the intermodality registration capabilities described here, to address what I believe to be the most important remaining problem in BLT image reconstruction, that of incorporating organ location information (potentially gleaned from MR images) so that the forward models may accurately reflect the heterogeneity in photon transport within the animal.

6. REFERENCES

1. Wang G, Li Y, Jiang M. Uniqueness theorems in bioluminescence tomography. *Med Phys*. 2004;31(8):2289-99. PubMed PMID: ISI:000223316600015.
2. Jiang M, Wang G. Image reconstruction for bioluminescence tomography. *Developments in X-Ray Tomography Iv*. 2004;5535:335-51. PubMed PMID: WOS:000225665000035.
3. Qin CH, Zhu SP, Feng JC, Zhong JH, Ma XB, Wu P, et al. Comparison of permissible source region and multispectral data using efficient bioluminescence tomography method. *J Biophotonics*. 2011;4(11-12):824-39. PubMed PMID: WOS:000297740500005.
4. Jiang M, Wang G. Uniqueness results for multi-spectral bioluminescence tomography. *Crm Ser*. 2008;7:153-72. PubMed PMID: WOS:000268435400008.
5. Ill-Conditioned Definition [cited 2017 Sept 25, 2017]. Available from: <http://www.statisticshowto.com/ill-conditioned/>.
6. Law of Large Numbers. Available from: https://en.wikipedia.org/wiki/Law_of_large_numbers.
7. Tang JP, Han B, Han WM, Bi B, Li L. Mixed Total Variation and L-1 Regularization Method for Optical Tomography Based on Radiative Transfer Equation. *Comput Math Method M*. 2017. PubMed PMID: WOS:000394952900001.
8. Wang YQ, Feng JC, Jia KB, Sun ZH, Wei HJ. A Novel Reconstruction Algorithm for Bioluminescent Tomography Based on Bayesian Compressive Sensing. *Medical Imaging 2016-Biomedical Applications in Molecular, Structural, and Functional Imaging*. 2016;9788. PubMed PMID: WOS:000378223800026.
9. Hu YF, Liu J, Leng CC, An Y, Zhang S, Wang K. Lp Regularization for Bioluminescence Tomography Based on the Split Bregman Method. *Molecular Imaging and Biology*. 2016;18(6):830-7. PubMed PMID: WOS:000387367100005.
10. Leng CC, Yu DD, Zhang S, An Y, Hu YF. Reconstruction Method for Optical Tomography Based on the Linearized Bregman Iteration with Sparse Regularization. *Comput Math Method M*. 2015. PubMed PMID: WOS:000361241900001.
11. Wu P, Hu YF, Wang K, Tian J. Bioluminescence Tomography by an Iterative Reweighted $l(2)$ -Norm Optimization. *Ieee T Bio-Med Eng*. 2014;61(1):189-96. PubMed PMID: WOS:000333263300020.
12. Chen XL, Yang DF, Zhang QT, Liang JM. L-1/2 regularization based numerical method for effective reconstruction of bioluminescence tomography. *J Appl Phys*. 2014;115(18). PubMed PMID: WOS:000336919400052.
13. Darne C, Lu YJ, Seveck-Muraca EM. Small animal fluorescence and bioluminescence tomography: a review of approaches, algorithms and technology update. *Phys Med Biol*. 2014;59(1):R1-R64. PubMed PMID: WOS:000328549200001.

14. Johansen TA. On Tikhonov regularization, bias and variance in nonlinear system identification. *Automatica*. 1997;33(3):441-6. PubMed PMID: WOS:A1997WU49200015.
15. Moore E. On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*. 1920;26(9):394-5.
16. Penrose R. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*. 1955;51:406-13.
17. Barrett HH, Wilson DW, Tsui BMW. Noise Properties of the Em Algorithm .1. Theory. *Phys Med Biol*. 1994;39(5):833-46. PubMed PMID: WOS:A1994NP47900004.
18. Wilson DW, Tsui BMW, Barrett HH. Noise Properties of the Em Algorithm .2. Monte-Carlo Simulations. *Phys Med Biol*. 1994;39(5):847-71. PubMed PMID: WOS:A1994NP47900005.
19. Lu YJ, Machado HB, Bao QN, Stout D, Herschman H, Chatziioannou AF. In Vivo Mouse Bioluminescence Tomography with Radionuclide-Based Imaging Validation. *Molecular Imaging and Biology*. 2011;13(1):53-8. PubMed PMID: ISI:000286395600008.
20. Hu ZH, Liang JM, Yang WD, Fan WW, Li CY, Ma XW, et al. Experimental Cerenkov luminescence tomography of the mouse model with SPECT imaging validation. *Opt Express*. 2010;18(24):24441-50. PubMed PMID: WOS:000285586800058.
21. Liang ZX, Qiang YG, Liao YH, Zhu XS, Huang Z, Zhang XP, et al. In vivo mouse ^{99m}Tc SPECT scans with bioluminescence imaging validation. *J X-Ray Sci Technol*. 2013;21(1):85-91. Epub 2013/03/20. PubMed PMID: 23507854.
22. Dehghani H, Davis SC, Jiang SD, Pogue BW, Paulsen KD, Patterson MS. Spectrally resolved bioluminescence optical tomography. *Opt Lett*. 2006;31(3):365-7. PubMed PMID: ISI:000234961100025.
23. Han WM, Kazmi K, Cong WX, Wang G. Bioluminescence tomography with optimized optical parameters. *Inverse Probl*. 2007;23(3):1215-28. PubMed PMID: ISI:000246789100022.
24. Ahn S, Chaudhari AJ, Darvas F, Bouman CA, Leahy RM. Fast iterative image reconstruction methods for fully 3D multispectral bioluminescence tomography. *Phys Med Biol*. 2008;53(14):3921-42. PubMed PMID: ISI:000257338700013.
25. Behrooz A, Kuo C, Xu H, Rice B. Adaptive row-action inverse solver for fast noise-robust three-dimensional reconstructions in bioluminescence tomography: theory and dual-modality optical/computed tomography in vivo studies. *Journal of Biomedical Optics*. 2013;18(7). PubMed PMID: WOS:000323030400018.
26. Feng JC, Qin CH, Jia KB, Zhu SP, Liu K, Han D, et al. Total variation regularization for bioluminescence tomography with the split Bregman method. *Appl Optics*. 2012;51(19):4501-12. PubMed PMID: ISI:000306100100029.

27. Gao H, Zhao HK. Multilevel bioluminescence tomography based on radiative transfer equation Part 1: l1 regularization. *Opt Express*. 2010;18(3):1854-71. PubMed PMID: ISI:000274791200006.
28. Jiang M, Zhou T, Cheng JT, Cong WX, Wang G. Image reconstruction for bioluminescence tomography from partial measurement. *Opt Express*. 2007;15(18):11095-116. PubMed PMID: ISI:000249339800005.
29. Virostko J, Powers AC, Jansen ED. Validation of luminescent source reconstruction using single-view spectrally resolved bioluminescence images. *Appl Optics*. 2007;46(13):2540-7. PubMed PMID: WOS:000245969100020.
30. Wang G, Shen H, Liu Y, Cong A, Cong WX, Wang Y, et al. Digital spectral separation methods and systems for bioluminescence imaging. *Opt Express*. 2008;16(3):1719-32. PubMed PMID: WOS:000252932500038.
31. Taylor SL, Mason SKG, Grinton S, Cobbold M, Styles IB, Dehghani H. Optimisation of acquisition time in bioluminescence imaging. *Proc Spie*. 2015;9319. PubMed PMID: WOS:000353631300033.
32. Basevi HRA, Guggenheim JA, Dehghani H, Styles IB. Information-theoretic method for wavelength selection in bioluminescence tomography. *Diffuse Optical Imaging Iv*. 2013;8799. PubMed PMID: WOS:000323554600008.
33. CCD binning: Andor; [Sept. 25, 2017]. Available from: <http://www.andor.com/learning-academy/ccd-binning-what-does-binning-mean>.
34. Bal G, Schotland JC. Ultrasound-modulated bioluminescence tomography. *Phys Rev E*. 2014;89(3). PubMed PMID: WOS:000333646400001.
35. Han WW, Shen HO, Kazmi K, Cong WX, Wang G. Studies of a mathematical model for temperature-modulated bioluminescence tomography. *Appl Anal*. 2009;88(2):193-213. PubMed PMID: WOS:000266276800005.
36. Huynh NT, Hayes-Gill BR, Zhang F, Morgan SP. Ultrasound modulated imaging of luminescence generated within a scattering medium. *Journal of Biomedical Optics*. 2013;18(2). PubMed PMID: WOS:000315159900006.
37. Wang G, Cong WX, Durairaj K, Qian X, Shen H, Sinn P, et al. In vivo mouse studies with bioluminescence tomography. *Optics Express*. 2006;14(17):7801-9. PubMed PMID: WOS:000240164100037.
38. Feng JC, Jia KB, Yan GR, Zhu SP, Qin CH, Lv YJ, et al. An optimal permissible source region strategy for multispectral bioluminescence tomography. *Opt Express*. 2008;16(20):15640-54. PubMed PMID: ISI:000260091300038.
39. Lu YJ, Zhang XQ, Douraghy A, Stout D, Tian J, Chan TF, et al. Source Reconstruction for Spectrally-resolved Bioluminescence Tomography with Sparse A priori Information. *Opt Express*. 2009;17(10):8062-80. PubMed PMID: ISI:000266381900036.

40. Cong A, Cong WX, Lu YJ, Santago P, Chatziioannou A, Wang G. Differential Evolution Approach for Regularized Bioluminescence Tomography. *Ieee T Bio-Med Eng.* 2010;57(9):2229-38. PubMed PMID: WOS:000283068800016.
41. Basevi HRA, Tichauer KM, Leblond F, Dehghani H, Guggenheim JA, Holt RW, et al. Compressive sensing based reconstruction in bioluminescence tomography improves image resolution and robustness to noise. *Biomed Opt Express.* 2012;3(9):2131-41. PubMed PMID: ISI:000308861100016.
42. Cong W, Wang G. Bioluminescence tomography based on the phase approximation model. *J Opt Soc Am A.* 2010;27(2):174-9. PubMed PMID: WOS:000274210400006.
43. Feng JC, Jia KB, Qin CH, Zhu SP, Yang X, Tian J. Sparse Bayesian reconstruction method for multispectral bioluminescence tomography. *Chin Opt Lett.* 2010;8(10):1010-4. PubMed PMID: ISI:000282923300027.
44. Guo W, Jia KB, Zhang Q, Liu XY, Feng JC, Qin CH, et al. Sparse Reconstruction for Bioluminescence Tomography Based on the Semigreedy Method. *Comput Math Method M.* 2012. PubMed PMID: ISI:000308206900001.
45. Liu K, Tian J, Qin CH, Yang X, Zhu SP, Han D, et al. Tomographic bioluminescence imaging reconstruction via a dynamically sparse regularized global method in mouse models. *Journal of Biomedical Optics.* 2011;16(4). PubMed PMID: WOS:000291031400023.
46. Yu JJ, He XW, Geng GH, Liu F, Jiao LC. Hybrid Multilevel Sparse Reconstruction for a Whole Domain Bioluminescence Tomography Using Adaptive Finite Element. *Comput Math Method M.* 2013. PubMed PMID: WOS:000316139500001.
47. Yu JJ, Liu F, Wu JA, Jiao LC, He XW. Fast Source Reconstruction for Bioluminescence Tomography Based on Sparse Regularization. *Ieee T Bio-Med Eng.* 2010;57(10):2583-6. PubMed PMID: ISI:000283590000010.
48. Zhang XQ, Lu YJ, Chan T. A Novel Sparsity Reconstruction Method from Poisson Data for 3D Bioluminescence Tomography. *J Sci Comput.* 2012;50(3):519-35. PubMed PMID: WOS:000302257300003.
49. Jin A, Yazici B, Ale A, Ntziachristos V. Preconditioning of the fluorescence diffuse optical tomography sensing matrix based on compressive sensing. *Opt Lett.* 2012;37(20):4326-8. PubMed PMID: WOS:000310052800054.
50. Lee O, Kim JM, Bresler Y, Ye JC. Compressive Diffuse Optical Tomography: Noniterative Exact Reconstruction Using Joint Sparsity. *Ieee T Med Imaging.* 2011;30(5):1129-42. PubMed PMID: WOS:000290167500011.
51. Niu H, Lin ZJ, Tian F, Dhamne S, Liu H. Comprehensive investigation of three-dimensional diffuse optical tomography with depth compensation algorithm. *J Biomed Opt.* 2010;15(4):046005. PubMed PMID: 20799807; PubMed Central PMCID: PMC2921418.

52. Suzen M, Giannoula A, Durduran T. Compressed sensing in diffuse optical tomography. *Opt Express*. 2010;18(23):23676-90. PubMed PMID: WOS:000283940900037.
53. Gao H, Zhao HK, Cong WX, Wang G. Bioluminescence tomography with Gaussian prior. *Biomed Opt Express*. 2010;1(5):1259-77. PubMed PMID: ISI:000208209700002.
54. An Y, Liu J, Zhang GL, Ye JZ, Mao YM, Jiang SX, et al. Meshless reconstruction method for fluorescence molecular tomography based on compactly supported radial basis function. *Journal of Biomedical Optics*. 2015;20(10). PubMed PMID: WOS:000366017100010.
55. Eames ME, Pogue BW, Yalavarthy PK, Dehghani H. An efficient Jacobian reduction method for diffuse optical image reconstruction. *Opt Express*. 2007;15(24):15908-19. PubMed PMID: WOS:000251223900029.
56. Gupta S, Yalavarthy PK, Roy D, Piao D, Vasu RM. Singular value decomposition based computationally efficient algorithm for rapid dynamic near-infrared diffuse optical tomography. *Med Phys*. 2009;36(12):5559.
57. Saibaba AK, Kilmer M, Miller EL, Fantini S. Fast Algorithms for Hyperspectral Diffuse Optical Tomography. *SIAM Journal on Scientific Computing*. 2015;37(5):B712-B43.
58. Zhai YH, Cummer SA. Fast tomographic reconstruction strategy for diffuse optical tomography. *Opt Express*. 2009;17(7):5285-97. PubMed PMID: WOS:000264747500044.
59. Taylor SL, Mason SKG, Grinton SL, Cobbold M, Dehghani H. Accounting for filter bandwidth improves the quantitative accuracy of bioluminescence tomography. *Journal of Biomedical Optics*. 2015;20(9). PubMed PMID: WOS:000365128100019.
60. Contag CH, Bachmann MH. Advances in vivo bioluminescence imaging of gene expression. *Annual Review of Biomedical Engineering*. 2002;4:235-60. PubMed PMID: WOS:000177827800011.
61. Klerk CP, Overmeer RM, Niers TM, Versteeg H, Richel DJ, Buckle T, et al. Validity of bioluminescence measurements for noninvasive in vivo imaging of tumor load in small animals. *Biotechniques*. 2007;43(1 Suppl):7-13, 30.
62. Luker KE, Luker GD. Applications of bioluminescence imaging to antiviral research and therapy: multiple luciferase enzymes and quantitation. *Antiviral research*. 2008;78(3):179-87.
63. Sato A, Klaunberg B, Tolwani R. In vivo bioluminescence imaging. *Comparative medicine*. 2004;54(6):631-4.
64. Soling A, Rainov NG. Bioluminescence imaging in vivo - application to cancer research. *Expert Opinion on Biological Therapy*. 2003;3(7):1163-72.
65. Blasberg RG. In vivo molecular-genetic imaging: multi-modality nuclear and optical combinations. *Nuclear Medicine and Biology*. 2003;30(8):879-88. PubMed PMID: WOS:000188041100012.

66. Rice BW, Cable MD, Nelson MB. In vivo imaging of light-emitting probes. *Journal of Biomedical Optics*. 2001;6(4):432-40. PubMed PMID: WOS:000172895700007.
67. Wang G, Cong WX, Shen HO, Qian X, Henry M, Wang Y. Overview of bioluminescence tomography-a new molecular imaging modality. *Frontiers in Bioscience*. 2008;13:1281-93. PubMed PMID: WOS:000255775700105.
68. Klose AD, Hielscher AH. Optical tomography with the equation of radiative transfer. *Int J Numer Method H*. 2008;18(3-4):443-64. PubMed PMID: ISI:000256864200010.
69. Allard M, Cote D, Davidson L, Dazai J, Henkelman RM. Combined magnetic resonance and bioluminescence imaging of live mice. *Journal of Biomedical Optics*. 2007;12(3):Article No. 034018. PubMed PMID: BIOSIS:PREV200800116214.
70. Beattie BJ, Forster GJ, Govantes R, Le CH, Longo VA, Zanzonico PB, et al. Multimodality registration without a dedicated multimodality scanner. *Molecular Imaging*. 2007;6(2):108-20. PubMed PMID: WOS:000247785500004.
71. Hartley R, Zisserman A. *Multiple View Geometry in Computer Vision*. Second ed. Cambridge: Cambridge University Press; 2004.
72. Dehghani H, Davis SC, Pogue BW. Spectrally resolved bioluminescence tomography using the reciprocity approach. *Medical Physics*. 2008;35(11):4863-71. PubMed PMID: WOS:000260484400013.
73. Chaudhari AJ, Darvas F, Bading JR, Moats RA, Conti PS, Smith DJ, et al. Hyperspectral and multispectral bioluminescence optical tomography for small animal imaging. *Physics in Medicine and Biology*. 2005;50(23):5421-41. PubMed PMID: WOS:000234055900002.
74. Han WM, Wang G. Theoretical and numerical analysis on multispectral bioluminescence tomography. *Ima Journal of Applied Mathematics*. 2007;72(1):67-85. PubMed PMID: WOS:000243806100006.
75. Bolin FP, Preuss LE, Taylor RC, Ference RJ. Refractive-Index of Some Mammalian-Tissues Using a Fiber Optic Cladding Method. *Appl Optics*. 1989;28(12):2297-303. PubMed PMID: ISI:A1989AC65000027.
76. Zhao H, H. Emission spectra of bioluminescent reporters and interaction with mammalian tissue determine the sensitivity of detection in vivo. *Journal of biomedical optics*. 2005;10(4).
77. Nardo L, Brega A, Bondani M, Andreoni A. Non-tissue-like features in the time-of-flight distributions of plastic tissue phantoms. *Applied Optics*. 2008;47(13):2477-85. PubMed PMID: WOS:000256365000088.
78. Beattie BJ, Klose AD, Le CH, Longo VA, Dobrenkov K, Vider J, et al. Registration of planar bioluminescence to magnetic resonance and x-ray computed tomography images as a platform for the development of bioluminescence tomography reconstruction algorithms. *Journal of Biomedical Optics*. 2009;14(2). PubMed PMID: ISI:000266868500067.

79. Beattie BJ, Thorek DLJ, Schmidtlein CR, Pentlow KS, Humm JL, Hielscher AH. Quantitative Modeling of Cerenkov Light Production Efficiency from Medical Radionuclides. *PLoS One*. 2012;7(2). PubMed PMID: ISI:000302871500039.
80. Cerenkov PA. Visible emission of clean liquids by action of γ radiation. *Compt Rend Dokl Akad Nauk SSSR*. 1934;8:451.
81. Cerenkov PA. Visible Radiation Produced by Electron Moving in a Medium with Velocities Exceeding that of Light. *Phys Rev*. 1937;52(4):378-9.
82. Robertson R, Germanos MS, Li C, Mitchell GS, Cherry SR, Silva MD. Optical imaging of Cerenkov light generation from positron-emitting radiotracers. *Phys Med Biol*. 2009;54(16):N355-N65. PubMed PMID: WOS:000268664600015.
83. Spinelli AE, D'Ambrosio D, Calderan L, Marengo M, Sbarbati A, Boschi F. Cerenkov radiation allows in vivo optical imaging of positron emitting radiotracers. *Physics in Medicine and Biology*. 2010;55(2):483-95. PubMed PMID: ISI:000272960400010.
84. Liu HG, Ren G, Miao Z, Zhang XF, Tang XD, Han PZ, et al. Molecular Optical Imaging with Radioactive Probes. *Plos One*. 2010;5(3). PubMed PMID: ISI:000274997100016.
85. Ruggiero A, Holland JP, Lewis JS, Grimm J. Cerenkov Luminescence Imaging of Medical Isotopes. *Journal of Nuclear Medicine*. 2010;51(7):1123-30. PubMed PMID: ISI:000279430900020.
86. Lucignani G. Cerenkov radioactive optical imaging: a promising new strategy. *European Journal of Nuclear Medicine and Molecular Imaging*. 2011;38(3):592-5. PubMed PMID: ISI:000287098000021.
87. Axelsson J, Davis SC, Gladstone DJ, Pogue BW. Cerenkov emission induced by external beam radiation stimulates molecular fluorescence. *Med Phys*. 2011;38(7):4127-32. PubMed PMID: ISI:000292521100028.
88. Dothager RS, Goiffon RJ, Jackson E, Harpstrite S, Piwnica-Worms D. Cerenkov Radiation Energy Transfer (CRET) Imaging: A Novel Method for Optical Imaging of PET Isotopes in Biological Systems. *Plos One*. 2010;5(10). PubMed PMID: ISI:000282748100026.
89. Liu HG, Zhang XF, Xing BG, Han PZ, Gambhir SS, Cheng Z. Radiation-Luminescence-Excited Quantum Dots for in vivo Multiplexed Optical Imaging. *Small*. 2010;6(10):1087-91. PubMed PMID: ISI:000278629300004.
90. Kozirowski J, Ballangrud AM, McDevitt MR, Yang WH, Sgouros G, Balatoni JA, et al. Combined radionuclide- and photodynamic therapy: The activation of photosensitizers by Cerenkov radiation. *Journal of Nuclear Medicine*. 2000;41(5):314. PubMed PMID: WOS:000089892400315.
91. Ran C, Zhang Z, Hooker J, Moore A. In Vivo Photoactivation Without "Light": Use of Cherenkov Radiation to Overcome the Penetration Limit of Light. *Mol Imaging Biol*. 2011. Epub 2011/05/04. PubMed PMID: 21538154.

92. Spinelli AE, Kuo C, Rice BW, Calandrino R, Marzola P, Sbarbati A, et al. Multispectral Cerenkov luminescence tomography for small animal optical imaging. *Opt Express*. 2011;19(13):12605-18. Epub 2011/07/01. PubMed PMID: 21716501.
93. Zhong J, Tian J, Yang X, Qin C. Whole-Body Cerenkov Luminescence Tomography with the Finite Element SP3 Method. *Annals of Biomedical Engineering*. 2011;39(6, Sp. Iss. SI). PubMed PMID: BIOSIS:PREV201100353109.
94. Rohrlich F, Carlson BC. Positron-Electron Differences in Energy Loss and Multiple Scattering. *Phys Rev*. 1954;93(1):38-44. PubMed PMID: ISI:A1954UB47200006.
95. Nuclear Data WWW Service: Lunds Universitet; 2011 [cited 2011 January 16]. Available from: <http://nucleardata.nuclear.lu.se/nucleardata/toi/perchart.htm>.
96. CRC, editor. *CRC Handbook of Chemistry and Physics*. 86th Edition ed: CRC Press; 2005.
97. FDA Approved Drug Products: Food and Drug Administration; 2001 [cited 2011 May 5]. Available from: http://www.accessdata.fda.gov/drugsatfda_docs/label/2002/in111mal021902LB.pdf.
98. Frank I, Tamm I. Coherent visible radiation of fast electrons passing through matter. *Compt Rend Dokl Akad Nauk SSSR*. 1937;14:109-14.
99. ESTAR Stopping Power and Range Tables: National Institute of Standards and Technology; 2011 [cited 2011 January 16]. Available from: <http://physics.nist.gov/PhysRefData/Star/Text/ESTAR.html>.
100. Levin CS, Hoffman EJ. Calculation of positron range and its effect on the fundamental limit of positron emission tomography system spatial resolution. *Physics in Medicine and Biology*. 1999;44(3):781-99. PubMed PMID: WOS:000079100800014.
101. Levin CS, Hoffman EJ. Calculation of positron range and its effect on the fundamental limit of positron emission tomography system spatial resolution (vol 44, pg 781, 1999). *Physics in Medicine and Biology*. 2000;45(2):559-. PubMed PMID: WOS:000085410400024.
102. Bethe HA. Moliere Theory of Multiple Scattering. *Phys Rev*. 1953;89(6):1256-66. PubMed PMID: ISI:A1953UB45300022.
103. Ritson D. *Techniques of High Energy Physics*: Interscience Publishers, Inc.; 1961.
104. XCOM Photon Cross Sections Database: National Institute of Standards and Technology; 2011 [cited 2011 January 16]. Available from: <http://www.nist.gov/pml/data/xcom/index.cfm>.
105. Refractive Index Database: Mikhail Polyanskiy 2011 [cited 2011 February 9]. Available from: <http://refractiveindex.info>.
106. CSG. Water Density Calculator 2011.
107. IVIS200-BR-01. IVIS 200 Series: Single View 3D Reconstruction. Caliper LifeSciences; 2009.

108. ICRP Composition of soft tissue: National Institute of Standards and Technology; 2011 [cited 2011 January 16]. Available from: <http://physics.nist.gov/cgi-bin/Star/compos.pl?matno=261>.
109. Ross HH. Measurement of Beta-Emitting Nuclides Using Cerenkov Radiation. Anal Chem. 1969;41(10):1260-&. PubMed PMID: ISI:A1969D730200024.
110. Mitchell GS, Gill RK, Boucher DL, Li C, Cherry SR. In vivo Cerenkov luminescence imaging: a new tool for molecular imaging. Philosophical transactions Series A, Mathematical, physical, and engineering sciences. 2011;369(1955):4605-19.
111. Dehghani H, Brooksby B, Vishwanath K, Pogue BW, Paulsen KD. The effects of internal refractive index variation in near-infrared optical tomography: a finite element modelling approach. Physics in Medicine and Biology. 2003;48(16):2713-27. PubMed PMID: ISI:000185507200010.
112. Dempster AP, Laird NM, Rubin DB. Maximum Likelihood from Incomplete Data Via Em Algorithm. J Roy Stat Soc B Met. 1977;39(1):1-38. PubMed PMID: WOS:A1977DM46400001.
113. Bevington P, Robinson D. Data Reduction and Error Analysis for the Physical Sciences. 2nd Edition ed1992.
114. Klementyeva NV, Shirmanova MV, Serebrovskaya EO, Fradkov AF, Meleshina AV, Snopova LB, et al. In Vivo Bioluminescence Imaging of Tumor Cells Using Optimized Firefly Luciferase luc2. Modern Technologies in Medicine. 2013;5(3):6-13.
115. MacAusland R. The Moore-Penrose Inverse and Least Squares 2014 [cited 2017 10/25/2017]. Available from: <http://buzzard.pugetsound.edu/courses/2014spring/420projects/math420-UPS-spring-2014-macausland-pseudo-inverse.pdf>.
116. Lange K, Carson R. Em Reconstruction Algorithms for Emission and Transmission Tomography. J Comput Assist Tomo. 1984;8(2):306-16. PubMed PMID: WOS:A1984SG17200024.
117. Politte DG, Snyder DL. Corrections for Accidental Coincidences and Attenuation in Maximum-Likelihood Image-Reconstruction for Positron-Emission Tomography. IEEE T Med Imaging. 1991;10(1):82-9. PubMed PMID: WOS:A1991FD82700009.
118. Eames ME, Wang J, Pogue BW, Dehghani H. Wavelength band optimization in spectral near-infrared optical tomography improves accuracy while reducing data acquisition and computational burden. Journal of Biomedical Optics. 2008;13(5). PubMed PMID: WOS:000261764900046.

7. APPENDIX

CODE ASSOCIATED WITH AIM 1

```
function movie = alphaslices(IVisources, imgSTRUCT, IVisrange, IMGrange)

green= [zeros(64, 1), (0:63)'/63, zeros(64, 1)];
red= [(0:63)'/63, zeros(64, 1), zeros(64, 1)];
yellow= [(0:63)'/63, (0:63)'/63, zeros(64, 1), ];
gray= [(0:63)'/63, (0:63)'/63, (0:63)'/63];
hotmap= hot(64);
hotmap(1:10, :)= 0;
map= hsv(64);
map(1, :)=0;

[angles, dirLST]= getIVISangles(IVisources);

n= length(angles);
for i=1:n
    angle= angles(i); % rotation of the bed in degrees
    [r1, c1]= size(imgSTRUCT(i).image);
    fprintf(1, 'Reference image max is %f and its size is %d x %d\n', max(max(imgSTRUCT(i).image)), r1, c1);
    f= [deblank(dirLST(i, :)), '\luminescent.tif'];
    lumiIMG= my_tiffread2(f);
    lumiIMG= double(lumiIMG);
    [r2, c2]= size(lumiIMG);
    fprintf(1, 'IVIS lumin image min is %f and max is %f and its size is %d x %d\n', min(min(lumiIMG)), max(max(lumiIMG)), r2, c2);
    if r1 ~= r2 | c1 ~= c2
        lumiIMG= interp2(lumiIMG, (1:r1)*(r2/r1), (1:c1)*(c2/c1));
    end
    composite=
    alphaslices(ind2rgb(floor(scal(imgSTRUCT(i).image, IMGrange(1), IMGrange(2), 0, 63)), gray), ...

    ind2rgb(floor(scal(lowclip(lumiIMG, IVisrange(1), 0, 0, IVisrange(2), 0, 63)), map), .3); figure;
    imshow(composite);
    outfile= ['composite', num2str(i), '.jpg'];
    imwrite(composite, outfile);
end

movie=1;
```

[Published with MATLAB® R2017a](#)

```
% define_geometry - takes a segmented imageset and identifies the boundary voxels
% segmented file is a series of unsigned bytes where OUTSIDE voxels are coded as 0's
```

```

% and INSIDE voxels have values other than 0

function m = define_geometry(segmented_fname, X, Y, Z)

DEBUG= 0;

%----- read segmented data -----

fp= fopen(segmented_fname, 'rb');
segi mg= fread(fp, [X*Y*Z], 'uchar');
segi mg= reshape(segi mg, X, Y, Z);
fclose(fp);

%----- pad by two voxels all around -----

padsegi mg= zeros(X+4, Y+4, Z+4);
padsegi mg(3: (X+2), 3: (Y+2), 3: (Z+2))= segi mg;

dZ= (X+4)*(Y+4);
dY= X+4;
dX= 1;
i_nsi de= padsegi mg ~= 0; % inside is same size as padsegi mg but contains only 0's and 1's where
1's indicate INSIDE
i_i_nsi de= find(i_nsi de); % get indices of all INSIDE voxels in this slice
% first pass edges have face-neighbors that are outside
i_edge1= find((i_nsi de(i_i_nsi de-dX) == 0) | (i_nsi de(i_i_nsi de+dX) == 0) | (i_nsi de(i_i_nsi de-dY) ==
0) | (i_nsi de(i_i_nsi de+dY) == 0) | (i_nsi de(i_i_nsi de-dZ) == 0) | (i_nsi de(i_i_nsi de+dZ) == 0));
i_edge1= i_i_nsi de(i_edge1); % adjust indices to refer to "inside" matrix
i_nsi de(i_edge1)= 2; % code first pass edge voxels as 2's
% second pass edges must also have face-neighbors that are inside (i.e. not other edges)
i_edge2= find((i_nsi de(i_edge1-dX) == 1) | (i_nsi de(i_edge1+dX) == 1) | (i_nsi de(i_edge1-dY) == 1) |
(i_nsi de(i_edge1+dY) == 1) | (i_nsi de(i_edge1-dZ) == 1) | (i_nsi de(i_edge1+dZ) == 1));
i_edge2= i_edge1(i_edge2); % adjust indices to refer to "inside" matrix
i_nsi de(i_edge2)= 3; % code second pass edge voxels as 3's

if DEBUG
    for i=3: (Z+2)
        figure; imshow(i_nsi de(:, :, i)); colormap('gray');
    end
end

[ex, ey, ez]= ind2sub([X+4, Y+4, Z+4], i_edge2);
e= [ex, ey, ez];

% add vectors pointing inside
nx= (i_nsi de(i_edge2+dX) == 1) - (i_nsi de(i_edge2-dX) == 1);
nx= nx + (i_nsi de(i_edge2+dX+dY) == 1) - (i_nsi de(i_edge2-dX+dY) == 1);
nx= nx + (i_nsi de(i_edge2+dX-dY) == 1) - (i_nsi de(i_edge2-dX-dY) == 1);
nx= nx + (i_nsi de(i_edge2+dX+dZ) == 1) - (i_nsi de(i_edge2-dX+dZ) == 1);
nx= nx + (i_nsi de(i_edge2+dX-dZ) == 1) - (i_nsi de(i_edge2-dX-dZ) == 1);
nx= nx + (i_nsi de(i_edge2+dX+dY+dZ) == 1) - (i_nsi de(i_edge2-dX+dY+dZ) == 1);
nx= nx + (i_nsi de(i_edge2+dX-dY-dZ) == 1) - (i_nsi de(i_edge2-dX-dY-dZ) == 1);
nx= nx + (i_nsi de(i_edge2+dX+dY-dZ) == 1) - (i_nsi de(i_edge2-dX+dY-dZ) == 1);

```

```

nx= nx + (i nsi de(i _edge2+dX-dY+dZ) == 1) - (i nsi de(i _edge2-dX-dY+dZ) == 1);

ny= (i nsi de(i _edge2+dY) == 1) - (i nsi de(i _edge2-dY) == 1);
ny= ny + (i nsi de(i _edge2+dY+dX) == 1) - (i nsi de(i _edge2-dY+dX) == 1);
ny= ny + (i nsi de(i _edge2+dY-dX) == 1) - (i nsi de(i _edge2-dY-dX) == 1);
ny= ny + (i nsi de(i _edge2+dY+dZ) == 1) - (i nsi de(i _edge2-dY+dZ) == 1);
ny= ny + (i nsi de(i _edge2+dY-dZ) == 1) - (i nsi de(i _edge2-dY-dZ) == 1);
ny= ny + (i nsi de(i _edge2+dY+dX+dZ) == 1) - (i nsi de(i _edge2-dY+dX+dZ) == 1);
ny= ny + (i nsi de(i _edge2+dY-dX-dZ) == 1) - (i nsi de(i _edge2-dY-dX-dZ) == 1);
ny= ny + (i nsi de(i _edge2+dY+dX-dZ) == 1) - (i nsi de(i _edge2-dY+dX-dZ) == 1);
ny= ny + (i nsi de(i _edge2+dY-dX+dZ) == 1) - (i nsi de(i _edge2-dY-dX+dZ) == 1);

nz= (i nsi de(i _edge2+dZ) == 1) - (i nsi de(i _edge2-dZ) == 1);
nz= nz + (i nsi de(i _edge2+dZ+dX) == 1) - (i nsi de(i _edge2-dZ+dX) == 1);
nz= nz + (i nsi de(i _edge2+dZ-dX) == 1) - (i nsi de(i _edge2-dZ-dX) == 1);
nz= nz + (i nsi de(i _edge2+dZ+dY) == 1) - (i nsi de(i _edge2-dZ+dY) == 1);
nz= nz + (i nsi de(i _edge2+dZ-dY) == 1) - (i nsi de(i _edge2-dZ-dY) == 1);
nz= nz + (i nsi de(i _edge2+dZ+dX+dY) == 1) - (i nsi de(i _edge2-dZ+dX+dY) == 1);
nz= nz + (i nsi de(i _edge2+dZ-dX-dY) == 1) - (i nsi de(i _edge2-dZ-dX-dY) == 1);
nz= nz + (i nsi de(i _edge2+dZ+dX-dY) == 1) - (i nsi de(i _edge2-dZ+dX-dY) == 1);
nz= nz + (i nsi de(i _edge2+dZ-dX+dY) == 1) - (i nsi de(i _edge2-dZ-dX+dY) == 1);

% normal points out (therefore minus)
n= -[nx, ny, nz];
normlengh= sqrt(sum(n.^2,2));
% for any zero length norms use vectors pointing OUTSIDE
if any(normlengh == 0)
    i_zero_edge= find(normlengh==0);
    i_zero= i_edge2(i_zero_edge); % adjust indices to refer to "inside" matrix
    % add vectors pointing outside
    nx= (i nsi de(i _zero+dX) == 0) - (i nsi de(i _zero-dX) == 0);
    ny= (i nsi de(i _zero+dY) == 0) - (i nsi de(i _zero-dY) == 0);
    nz= (i nsi de(i _zero+dZ) == 0) - (i nsi de(i _zero-dZ) == 0);
    n(i_zero_edge,:)= [nx, ny, nz];
end
normlengh= sqrt(sum(n.^2,2));
if any(normlengh == 0)
    i_zero_edge= find(normlengh==0);
    fprintf(1,'Encountered %d zero lengthed normals\n',length(i_zero_edge));
    inside(ex(i_zero_edge(1))-1:ex(i_zero_edge(1))+1,ey(i_zero_edge(1))-1:ey(i_zero_edge(1))+1,ez(i_zero_edge(1))-1:ez(i_zero_edge(1))+1)
    n(i_zero_edge(1),: )
    % error('stop');
end
n= n ./ [normlengh,normlengh,normlengh]; % force unit length
m= [e-3,n]; % subtract three from edge coord to get rid of padding and switch to indices starting at 0

%----- write output file -----

fp= fopen('normal vector.ini','w');
if fp == -1
    error('Cannot create normal vector.ini');

```

```

end

fprintf(fp, '%%NUMBER_OF_BOUNDARY_POINTS ');
fprintf(fp, '%d\n\n', length(i_edge2));
fprintf(fp, '\n\ngrid point in cartesian coordinates\n');
fprintf(fp, '(x, y, z)\n\n\n');
fprintf(fp, 'normal vector in cartesian coordinates \n');
fprintf(fp, 'n_x, n_y, n_z) = (sin a cos b, sin a sin b, cos a) \n\n');
fprintf(fp, 'a and b are spherical coordinates\n');
fprintf(fp, 'x y z n_x n_y n_z\n\n');
fprintf(fp, '%%NORMAL_VECTOR\n');

% Write surface points and normals.
for i=1:length(i_edge2)
    fprintf(fp, '%d %d %d %f %f %f\n', m(i, 1), m(i, 2), m(i, 3), m(i, 4), m(i, 5), m(i, 6));
end

% Attention! This line is necessary!
fprintf(fp, '\n');
fclose(fp);

fprintf(1, 'wrote normal vector. ini\n');

fp= fopen('segmentation.txt', 'w');
if fp == -1
    error('Cannot create segmentation.txt');
end

for z=1:Z
    for y=1:Y
        fprintf(fp, '%c ', segimg(:, y, z)+'0');
    end
end
fprintf(fp, '\n');
fclose(fp);

fprintf(1, 'wrote segmentation.txt\n');

dimensions=[X, Y, Z];
all_surface_coordinates= m(:, 1:3);
all_surface_normals= m(:, 4:6);
[i, j, k]= ind2sub(size(segimg), find(segimg>0));
all_interior_coordinates= [i, j, k] - 1;

save geometry all_surface_coordinates all_surface_normals all_interior_coordinates dimensions

fprintf(1, 'wrote geometry.mat\n');

```

[Published with MATLAB® R2017a](#)

```

function
forwardmodel2image(nvFile, subSampleFactor, crops, parameterSTRUCT, window, angles, n_freq, forwardmodel
, fmsf)
    nv= read_nvf(nvFile);

    [r,c]= size(nv);
    % the following assumes that the image was cropped first, then subsampled
    surfPTS= [nv(:,1:3)] * subSampleFactor; ones(1,r)];
    surfPTS(1,:)= surfPTS(1,:) + crops(1);          % add what was cropped off the image left side
    surfPTS(2,:)= surfPTS(2,:) + crops(2);          % add what was cropped off the image top (when
displayed top to bottom)
    surfPTS(3,:)= surfPTS(3,:) + crops(3);          % add what was cropped off the front (i.e. first
images)

    % camera model considers Z of CT to be the Y axis, and Y of CT to be the Z axis and X is
flipped
    tmp= surfPTS(3,:);
    surfPTS(3,:)= surfPTS(2,:);          % CT Y becomes Z
    surfPTS(1,:)= 191 - surfPTS(1,:);    % CT X becomes flipped X
    surfPTS(2,:)= tmp;                   % CT Z becomes Y

    % similar dimension reordering for norms
    norms= nv(:,4:6)';
    tmp= norms(3,:);
    norms(3,:)= norms(2,:);
    norms(1,:)= -norms(1,:);
    norms(2,:)= tmp;

    dimX= parameterSTRUCT.dims(1);          % IVIS image size
    dimY= parameterSTRUCT.dims(2);
    px= dimX/2; py= dimY/2;                % principal point
    offset fixed at center of image
    f= parameterSTRUCT.f;                  % focal length
    aor2yRPY= parameterSTRUCT.aor2yRPY;    % roll, pitch and
yaw to align axis of rotation of the mousebed to the Y-axis
    aor2yXYZ= [parameterSTRUCT.xzshft(1); 0; parameterSTRUCT.xzshft(2)]; % X and Z shift to
align axis of rotation of the mousebed to the Y-axis
    wcs2ccsRPY= parameterSTRUCT.wcs2ccsRPY; % roll, pitch and
yaw to align world coordinate system to camera coordinate system
    Ctilde= parameterSTRUCT.Ctilde;        % coordinates of the
camera center in the world coordinate system
    rdfs= parameterSTRUCT.rdfs;            % radial distortion
factors

    K= [f 0 px; 0 f py; 0 0 1];          % camera calibration matrix
    x= [0 0 0 1 wcs2ccsRPY 0];
    R= x2t(x', 'rpy');                   % rotation matrix representing the orientation of the camera
coordinate frame
    R= R(1:3,1:3);                       % reduce R to 3x3
    P3= K * R * [eye(3) -Ctilde'];       % projective transform model of the camera

    % determine transformation matrix which aligns the axis of rotation of the mousebed to the Y-
axis

```

```

x= [0 0 0 1 aor2yRPY 0];
R= x2t(x','rpy');
P1= R * [[eye(3) aor2yXYZ]; [0,0,0,1]];

k= 1;
n= length(angles);
for j=1:n_freq
    for i=1:n
        angle= angles(i); % rotation of the bed in degrees
        x= [0 0 0 1 0 angle*pi/180 0 0];
        P2= x2t(x','rpy'); % transformation matrix describing rotation
of the bed
        P= P3 * P2 * P1; % align bed, rotate bed, project to camera
coordinate frame

imgPTS= P * surfPTS;
imgPTS(1:2,:) = imgPTS(1:2,:) ./ repmat(imgPTS(3,:),2,1);
P2R= P2 * R;
P2R= P2R(1:3,1:3);
imgNRMS= P2R * norms;
rotSurfPTS= P2R * surfPTS(1:3,:);
x= round(imgPTS(1,:));
y= round(imgPTS(2,:));
% must be in the window and pointing in the negative Z direction (i.e. towards the
camera)
i_inWindow= find(x>window(1) & x<window(3) & y>window(2) & y<window(4) & imgNRMS(3,:)<0
& abs(imgNRMS(3,:))>abs(imgNRMS(2,:)) & abs(imgNRMS(3,:))>abs(imgNRMS(1,:)));
if i==1, si=1; end
ssi = si : fmsf: length(i_inWindow);
si = fmsf - (length(i_inWindow) - ssi(end));
fprintf(1,'angle %d, in window %d, ss %d\n',angle,length(i_inWindow),length(ssi));
x= x(i_inWindow(ssi));
y= y(i_inWindow(ssi));
sampleSites= zeros(dimX,dimY);
sampleSites(sub2ind([dimX,dimY],x,y))= forwardmodel(k:(k+length(x)-1));
k= k + length(x);
figure; imshow(sampleSites); colormap('gray');
end
end

```

[Published with MATLAB® R2017a](#)

```

function distortSTRUCT= get_distortion_data(distortIMG)
[xdim,ydim]= size(distortIMG);
imshow(distortIMG);
colormap('gray');
hold on;
xi = zeros(200,1);
yi = zeros(200,1);
j = 1;

```



```

while (1)
    [xl, yl, b]= ginput(2);
    if b(1) ~= 1, break; end
    plot(xl, yl);
    i = 1;
    while (1)
        [x, y, b]= ginput(1);
        if b ~= 1, break; end
        plot(x, y, 'rx');
        xi(i) = x;
        yi(i) = y;
        i = i + 1;
    end
    distortSTRUCT(j).line= [xl, yl];
    distortSTRUCT(j).points= [xi(1:(i-1)), yi(1:(i-1))];
    distortSTRUCT(j).xdim= xdim;
    distortSTRUCT(j).ydim= ydim;
    j= j + 1;
end

```

[Published with MATLAB® R2017a](#)

```

function calpts= getcalpts(tif)
t= my_imread(tif); %
mn= 0;
mx= overall_max(t)/4;
h= figure;
imshowc(t, [mn, mx]);
colormap('gray');
set(h, 'Position', [100, 100, 1000, 1000]);
i = 0;
while (1)
    [x, y, but]= ginput(1);
    if but == 1
        i = i + 1;
        calpts(i, 1)= x;
        calpts(i, 2)= y;
    end
    if but == 2
        a= input('adjust zoom and hit return: ');
        if strcmp(a, 'q') == 1
            break;
        end
    end
    if but == 3 & i > 1
        i = i - 1;
    end
    if i > 0
        calpts= calpts(1:i, :);
        hold off;
    end
end

```

```

        imshowc(t, [mn, mx]);
        hold on;
        plot(cal pts(:, 1), cal pts(:, 2), 'r-x');
        set(h, 'Position', [100, 100, 1000, 1000]);
    end
end
hold on; plot(cal pts(:, 1), cal pts(:, 2), 'r-x');
set(h, 'Position', [100, 100, 1000, 1000]);

```

[Published with MATLAB® R2017a](#)

```

% getIVISangles - gets list of directories containing IVIS data rotated at various angles
% directory names must follow convention of having a common root followed by 'm', 'p' or
% nothing followed by the angle number. 'm' is for minus and 'p' and nothing are for plus
function [angles, dirLST] = getIVISangles(rootname)
    d= dir([rootname, '*']);
    s= length(rootname) + 1;
    angleCNT= 0;
    dirLST= [];
    for i=1:length(d)
        % disp(d(i).name);
        if d(i).isdir == 1
            dirLST= strvcats(dirLST, d(i).name);
            angleCNT= angleCNT + 1;
            angleID= d(i).name(s:end);
            if angleID(1) == 'm'
                angles(angleCNT)= -str2num(angleID(2:end));
            elseif angleID(1) == 'p'
                angles(angleCNT)= str2num(angleID(2:end));
            else
                angles(angleCNT)= str2num(angleID(1:end));
            end
        end
    end
    [angles, i]= sort(angles);
    dirLST= dirLST(i,:);

```

[Published with MATLAB® R2017a](#)

```

% getIVIScalangles - gets list of txt files containing IVIS calibration points at various angles
% txt file names must follow convention of having a common root followed by 'm', 'p' or
% nothing followed by the angle number, followed by '.txt'. 'm' is for minus and 'p' and nothing
% is for plus
function [angles, cal fileLST] = getIVIScalangles(rootname)
    d= dir([rootname, '*.txt']);
    s= length(rootname) + 1;

```

```

angl eCNT= 0;
cal fil eLST= [];
angl es= [];
for i=1:length(d)
    if d(i).isdir == 0
        cal fil eLST= strvcats(cal fil eLST, debl ank(d(i). name));
        angl eCNT= angl eCNT + 1;
        angl eID= d(i). name(s: end-4);
        if angl eID(1) == 'm'
            angl es(angl eCNT)= -str2num(angl eID(2: end));
        elseif angl eID(1) == 'p'
            angl es(angl eCNT)= str2num(angl eID(2: end));
        else
            angl es(angl eCNT)= str2num(angl eID(1: end));
        end
    end
end
[angl es, i]= sort(angl es);
cal fil eLST= cal fil eLST(i, :);

```

[Published with MATLAB® R2017a](#)

```

% getIVISspectra - gets list of directories containing IVIS data taken with various filters
% directory names must be in the form Odd where dd is a two digit number
function [frequencies, dirLST] = getIVISspectra(dirname)
    frequencies= [];
    d= dir(dirname);
    freqCNT= 0;
    dirLST= [];
    for i=1:length(d)
        if d(i).isdir == 1
            [freqNUM, c]= sscanf(d(i). name, '%d', 1);
            if c > 0
                dirLST= strvcats(dirLST, d(i). name);
                freqCNT= freqCNT + 1;
                frequencies(freqCNT)= freqNUM;
            end
        end
    end
    [frequencies, i]= sort(frequencies);
    dirLST= dirLST(i, :);

```

[Published with MATLAB® R2017a](#)

```

function [refIntensities, refCoordinates, sampleSites]=
mapIVIS(nvFile, subSampleFactor, crops, parameterSTRUCT, IVISrootname, window, csc_factors)

```

```

nv= read_nvf(nvFile);

[r,c]= size(nv);
% the following assumes that the image was cropped first, then subsampled
surfPTS= [nv(:,1:3)' * subSampleFactor; ones(1,r)];
surfPTS(1,:)= surfPTS(1,:) + crops(1);          % add what was cropped off the image left side
surfPTS(2,:)= surfPTS(2,:) + crops(2);          % add what was cropped off the image top (when
displayed top to bottom)
surfPTS(3,:)= surfPTS(3,:) + crops(3);          % add what was cropped off the front (i.e. first
images)

% camera model considers Z of CT to be the Y axis, and Y of CT to be the Z axis and X is
flipped
tmp= surfPTS(3,:);
surfPTS(3,:)= surfPTS(2,:);          % CT Y becomes Z
surfPTS(1,:)= 191 - surfPTS(1,:);    % CT X becomes flipped X
surfPTS(2,:)= tmp;                   % CT Z becomes Y

% similar dimension reordering for norms
norms= nv(:,4:6)';
tmp= norms(3,:);
norms(3,:)= norms(2,:);
norms(1,:)= -norms(1,:);
norms(2,:)= tmp;

dimX= parameterSTRUCT.dims(1);          % IVIS image size
dimY= parameterSTRUCT.dims(2);
px= dimX/2; py= dimY/2;                  % principal point
offset fixed at center of image
f= parameterSTRUCT.f;                    % focal length
aor2yRPY= parameterSTRUCT.aor2yRPY;     % roll, pitch and
yaw to align axis of rotation of the mousebed to the Y-axis
aor2yXYZ= [parameterSTRUCT.xzshft(1); 0; parameterSTRUCT.xzshft(2)]; % X and Z shift to
align axis of rotation of the mousebed to the Y-axis
wcs2ccsRPY= parameterSTRUCT.wcs2ccsRPY; % roll, pitch and
yaw to align world coordinate system to camera coordinate system
Ctilde= parameterSTRUCT.Ctilde;          % coordinates of the
camera center in the world coordinate system
rdfs= parameterSTRUCT.rdfs;              % radial distortion
factors

K= [f 0 px; 0 f py; 0 0 1];             % camera calibration matrix
x= [0 0 0 1 wcs2ccsRPY 0];
R= x2t(x', 'rpy');                       % rotation matrix representing the orientation of the camera
coordinate frame
R= R(1:3,1:3);                           % reduce R to 3x3
P3= K * R * [eye(3) -Ctilde'];          % projective transform model of the camera

% determine transformation matrix which aligns the axis of rotation of the mousebed to the Y-
axis
x= [0 0 0 1 aor2yRPY 0];
R= x2t(x', 'rpy');
P1= R * [[eye(3) aor2yXYZ]; [0,0,0,1]];

```

```

[angles, dirLST]= getIVISangles(IVISrootname);

k= 1;
n= length(angles);
for i=1:n
    angle= angles(i); % rotation of the bed in degrees
    x= [0 0 0 1 0 angle*pi/180 0 0];
    P2= x2t(x,'rpy'); % transformation matrix describing rotation of
the bed
    P= P3 * P2 * P1; % align bed, rotate bed, project to camera
coordinate frame
    [frequencies, freqLST]= getIVISspectra(dirLST(i,:));
    m1= length(frequencies);
    if m1 == 0, m2= 1; else m2= m1; end
    for j=1:m2
        if m1 == 0
            f= sprintf('%s\\luminescentFloatCorrected.tif', dirLST(i,:));
        else
            f= sprintf('%s\\%s\\luminescentFloatCorrected.tif', dirLST(i,:), freqLST(j,:));
        end
        lumnIMG= double(my_tiffread2(f));
        [lm_dmx, lm_dmy]= size(lumnIMG);
        if lm_dmx ~= dmx | lm_dmy ~= dmy
            dx= lm_dmx / dmx;
            dy= lm_dmy / dmy;
            [xi, yi]= meshgrid(dx:dx:lm_dmx, dy:dy:lm_dmy);
            lumnIMG= interp2(lumnIMG, xi, yi);
        end
        figure; imshow(lumnIMG);
        hold on;

        plot([window(1), window(3), window(3), window(1), window(1)], [window(2), window(2), window(4), window
(4), window(2)], 'r');
        imgPTS= P * surfPTS;
        imgPTS(1:2,:) = imgPTS(1:2,:) ./ repmat(imgPTS(3,:), 2, 1);
        P2R= P2 * R;
        P2R= P2R(1:3, 1:3);
        imgNRMS= P2R * norms;
        rotSurfPTS= P2R * surfPTS(1:3,:);
        sampleSites= zeros(dmx, dmy);
        x= round(imgPTS(1,:));
        y= round(imgPTS(2,:));
        % must be in the window and pointing in the negative Z direction (i.e. towards the
camera)
        i_nWindow= find(x>window(1) & x<window(3) & y>window(2) & y<window(4) & imgNRMS(3,:)<0
& abs(imgNRMS(3,:))>abs(imgNRMS(2,:)) & abs(imgNRMS(3,:))>abs(imgNRMS(1,:)));
        x= x(i_nWindow);
        y= y(i_nWindow);
        sampleSites(sub2ind([dmx, dmy], x, y))= -imgNRMS(3, i_nWindow);
        figure; imshow(sampleSites); colormap('gray');
        refIntensities= lumnIMG(sub2ind([dmx, dmy], x, y)); % divide by the following to
get cos adjustment ./ -imgNRMS(3, i_nWindow);

```

```

    if nargin == 7
        refIntensities= refIntensities * csc_factors(k);
    end
    k= k + 1;
    fp= fopen(sprintf('%s_%s_Intensities.txt', dirLST(i,:), freqLST(j,:)), 'w');
    fprintf(fp, '%f ', refIntensities);
    fclose(fp);
end
refCoordinates= nv(i_nWindow, 1:3);
fp= fopen(sprintf('%s_IntensityCoordinates.txt', dirLST(i,:)), 'w');
fprintf(fp, '%f %f %f\n', refCoordinates);
fclose(fp);
q= [rotSurfPTS', imgNRMS'];
figure; qui ver3(q(:, 1), q(:, 2), q(:, 3), q(:, 4), q(:, 5), q(:, 6)); xl label ('X'); yl label ('Y');
zl label ('Z');
axis image
figure;
plot3(nv(:, 1), nv(:, 2), nv(:, 3), 'b. ');
hold on;
plot3(refCoordinates(:, 1), refCoordinates(:, 2), refCoordinates(:, 3), 'rh');
xl label ('X'); yl label ('Y'); zl label ('Z');
axis image
end

```

[Published with MATLAB® R2017a](#)

```

function plot_nvfv(nvf_fname)
    qui vers= read_nvfv(nvf_fname);
    figure;
    qui ver3(qui vers(:, 1), qui vers(:, 2), qui vers(:, 3), qui vers(:, 4), qui vers(:, 5), qui vers(:, 6));
    % b= min(qui vers(:, 3));
    % e= max(qui vers(:, 3));
    % for i=b:e
    %     si= find(qui vers(:, 3)==i);
    %     figure; hold on;
    %     for j=1:length(si)
    %
    %         plot([qui vers(si(j), 1), qui vers(si(j), 1)+qui vers(si(j), 4)], [qui vers(si(j), 2), qui vers(si(j), 2)+qui vers(si(j), 5)]);
    %         plot(qui vers(si(j), 1)+qui vers(si(j), 4), qui vers(si(j), 2)+qui vers(si(j), 5), 'x');
    %     end
    %     axis equal;
    % end

```

[Published with MATLAB® R2017a](#)

```

function projections=
project_currentdiffeusion(cd_fname, subSampleFactor, crops, parameterSTRUCT, angles)
    cd_data= load(cd_fname);
    [r,c]= size(cd_data);
    intensities= cd_data(:,7);
    % the following assumes that the image was cropped first, then subsampled
    surfPTS= [cd_data(:,1:3)' * subSampleFactor; ones(1,r)];
    surfPTS(1,:)= surfPTS(1,:) + crops(1);      % add what was cropped off the left
    surfPTS(2,:)= surfPTS(2,:) + crops(2);      % add what was cropped off the top
    surfPTS(3,:)= surfPTS(3,:) + crops(3);      % add what was cropped off the front

    % camera model considers Z of CT to be the Y axis, and Y of CT to be the Z axis and X is
    flipped
    tmp= surfPTS(3,:);
    surfPTS(3,:)= surfPTS(2,:);      % CT Y becomes Z
    surfPTS(1,:)= 191 - surfPTS(1,:); % CT X becomes flipped X
    surfPTS(2,:)= tmp;               % CT Z becomes Y

    % similar dimension reordering for norms
    norms= cd_data(:,4:6)';
    tmp= norms(3,:);
    norms(3,:)= norms(2,:);
    norms(1,:)= -norms(1,:);
    norms(2,:)= tmp;

    di mX= parameterSTRUCT.dims(1);      % IVIS image size
    di mY= parameterSTRUCT.dims(2);
    px= di mX/2; py= di mY/2;            % principal point
    offset fixed at center of image
    f= parameterSTRUCT.f;                % focal length
    aor2yRPY= parameterSTRUCT.aor2yRPY; % roll, pitch and
    yaw to align axis of rotation of the mousebed to the Y-axis
    aor2yXYZ= [parameterSTRUCT.xzshft(1); 0; parameterSTRUCT.xzshft(2)]; % X and Z shift to
    align axis of rotation of the mousebed to the Y-axis
    wcs2ccsRPY= parameterSTRUCT.wcs2ccsRPY; % roll, pitch and
    yaw to align world coordinate system to camera coordinate system
    Ctilde= parameterSTRUCT.Ctilde;      % coordinates of the
    camera center in the world coordinate system
    rdfs= parameterSTRUCT.rdfs;           % radial distortion
    factors

    K= [f 0 px; 0 f py; 0 0 1];          % camera calibration matrix
    x= [0 0 0 1 wcs2ccsRPY 0];
    R= x2t(x', 'rpy');                   % rotation matrix representing the orientation of the camera
    coordinate frame
    R= R(1:3,1:3);                       % reduce R to 3x3
    P3= K * R * [eye(3) -Ctilde'];       % projective transform model of the camera

    % determine transformation matrix which aligns the axis of rotation of the mousebed to the Y-
    axis
    x= [0 0 0 1 aor2yRPY 0];
    R= x2t(x', 'rpy');

```

```

P1= R * [[eye(3) aor2yXYZ]; [0,0,0,1]];

projections= [];
[xi, yi] = meshgrid(1:dimX, 1:dimY);
n= length(angles);
for i=1:n
    angle= angles(i); % rotation of the bed in degrees
    x= [0 0 0 1 0 angle*pi/180 0 0];
    P2= x2t(x, 'rpy'); % transformation matrix describing rotation of
the bed
    P= P3 * P2 * P1; % align bed, rotate bed, project to camera
coordinate frame
    [K, R, S] = perspective_transform_decompose(P);
    imgPTS= P * surfPTS;
    imgPTS(1:2,:) = imgPTS(1:2,:) ./ repmat(imgPTS(3,:), 2, 1);
    imgNRMS= R * norms;
    rotSurfPTS= R * surfPTS(1:3,:);
    x= imgPTS(1,:);
    y= imgPTS(2,:);
    up= find(imgNRMS(3,:) < 0.0); % negative Z points up toward camera
    projection= griddata(x(up), y(up), intensities(up), xi, yi)';
    projection(isnan(projection))= 0;
% figure; imshowsc(projection);
    projections= [projections; projection];
end
projections= reshape(projections, dimX, dimY, n);

```

[Published with MATLAB® R2017a](#)

```

function nv= read_nvf(nvf_fname)
fp= fopen(nvf_fname, 'r');
if fp == -1, error(sprintf('Cannot open %s', nvf_fname)); end
while 1
    s= fgetl(fp);
    if strncmpi(s, '%NORMAL_VECTOR', 14), break; end
end
nv= fscanff(fp, '%f %f %f %f %f %f', [6, inf])';

```

[Published with MATLAB® R2017a](#)

```

function imgSTRUCT= registerCT_to_IVIS(CTrootname, IVISrootname, IVISparameterSTRUCT, thres, typ,
multifile, ext, start, machineformat)
if nargin < 5
    typ= 'int16';
    multifile= 3;
    ext= 'ct';

```



```

        start= 0;
        machineformat= 'IEEE-LE';
    end
    ct= read_raw(CTrootname, typ, [192, 192, 384], multi file, ext, start, machineformat);
    fv= isosurface(ct, thres);
    norms= isonormals(ct, fv.vertices);
    tmp= fv.vertices(:, 3);
    fv.vertices(:, 3)= fv.vertices(:, 1);           % original X becomes Z
    fv.vertices(:, 1)= 191 - fv.vertices(:, 2);     % original Y becomes flipped X
    fv.vertices(:, 2)= tmp;                         % original Z becomes Y
    tmp= norms(:, 3);
    norms(:, 3)= norms(:, 1);
    norms(:, 1)= -norms(:, 2);
    norms(:, 2)= tmp;

    di mX= I V I SparameterSTRUCT. di ms(1);        % I V I S image
size
    di mY= I V I SparameterSTRUCT. di ms(2);
    px= di mX/2; py= di mY/2;                      % principal
point offset fixed at center of image
    f= I V I SparameterSTRUCT. f;                  % focal length
    aor2yRPY= I V I SparameterSTRUCT. aor2yRPY;    % roll, pitch
and yaw to align axis of rotation of the mousebed to the Y-axis
    aor2yXYZ= [I V I SparameterSTRUCT. xzshft(1); 0; I V I SparameterSTRUCT. xzshft(2)]; % X and Z shift
to align axis of rotation of the mousebed to the Y-axis
    wcs2ccsRPY= I V I SparameterSTRUCT. wcs2ccsRPY; % roll, pitch
and yaw to align world coordinate system to camera coordinate system
    Ctilde= I V I SparameterSTRUCT. Ctilde;         % coordinates of
the camera center in the world coordinate system
    rdfs= I V I SparameterSTRUCT. rdfs;             % radial
distortion factors

    K= [f 0 px; 0 f py; 0 0 1];                    % camera calibration matrix
    x= [0 0 0 1 wcs2ccsRPY 0];
    R= x2t(x', 'rpy');                             % rotation matrix representing the orientation of the camera
coordinate frame
    R= R(1:3, 1:3);                                 % reduce R to 3x3
    P3= K * R * [eye(3) -Ctilde'];                 % projective transform model of the camera

    % determine transformation matrix which aligns the axis of rotation of the mousebed to the Y-
axis
    x= [0 0 0 1 aor2yRPY 0];
    R= x2t(x', 'rpy');
    P1= R * [[eye(3) aor2yXYZ]; [0, 0, 0, 1]];

    [angles, di rLST]= getI V I Sangles(I V I Srootname);

    n= length(angles);
    for i=1:n
        angl e= angles(i);                          % rotation of the bed in degrees
        x= [0 0 0 1 0 angl e*pi /180 0 0];
        P2= x2t(x', 'rpy');                         % transformation matrix describing rotation of
the bed

```

```

P= P3 * P2 * P1; % align bed, rotate bed, project to camera
coordinate frame
[img,zbuf]= zbuffer(fv,norms,dimX,dimY,P);
img= distort2D(img,[rdfs,1]);
imgSTRUCT(i).image= img;
imgSTRUCT(i).angle= angle;
figure;
imshow(img,[0 1]);
colormap('gray');
drawnow;
end

```

[Published with MATLAB® R2017a](#)

```

function imgSTRUCT= renderSurface(surface_points,surface_normals,IVisparameterSTRUCT,angles)
dimX= IVisparameterSTRUCT.dims(1); % I V I S image
size
dimY= IVisparameterSTRUCT.dims(2);
px= dimX/2; py= dimY/2; % principal
point offset fixed at center of image
f= IVisparameterSTRUCT.f; % focal length
aor2yRPY= IVisparameterSTRUCT.aor2yRPY; % roll, pitch
and yaw to align axis of rotation of the mousebed to the Y-axis
aor2yXYZ= [IVisparameterSTRUCT.xzshft(1);0;IVisparameterSTRUCT.xzshft(2)]; % X and Z shift
to align axis of rotation of the mousebed to the Y-axis
wcs2ccsRPY= IVisparameterSTRUCT.wcs2ccsRPY; % roll, pitch
and yaw to align world coordinate system to camera coordinate system
Ctilde= IVisparameterSTRUCT.Ctilde; % coordinates of
the camera center in the world coordinate system
rdfs= IVisparameterSTRUCT.rdfs; % radial
distortion factors

K= [f 0 px; 0 f py; 0 0 1]; % camera calibration matrix
x= [0 0 0 1 wcs2ccsRPY 0];
R= x2t(x','rpy'); % rotation matrix representing the orientation of the camera
coordinate frame
R= R(1:3,1:3); % reduce R to 3x3
P3= K * R * [eye(3) -Ctilde']; % projective transform model of the camera

% determine transformation matrix which aligns the axis of rotation of the mousebed to the Y-
axis
x= [0 0 0 1 aor2yRPY 0];
R= x2t(x','rpy');
P1= R * [[eye(3) aor2yXYZ]; [0,0,0,1]];

n= length(angles);
for i=1:n
    angle= angles(i); % rotation of the bed in degrees
    x= [0 0 0 1 0 angle*pi/180 0 0];
    P2= x2t(x','rpy'); % transformation matrix describing rotation of

```

```

the bed
    P= P3 * P2 * P1; % align bed, rotate bed, project to camera
coordinate frame
    [img, zbuf]= zbuffer(fv, norms, di mX, di mY, P);
    img= di stort2D(img, [rdfs, 1]);
    imgSTRUCT(i).i mage= img;
    imgSTRUCT(i). angl e= angl e;
    figure;
    imshowsc(img, [0 1]);
    colormap(' gray');
    drawnow;
end

```

[Published with MATLAB® R2017a](#)

```

function img= show_currentdi ffusi on(fname, X, Z, ythres)
    data= load(fname);
    img= zeros(X+2, Z+2);
    [r, c]= si ze(data);
    for i=1: r
        if data(i, 2)>ythres,
            img(data(i, 1)+1, data(i, 3)+1)= data(i, 7);
        end
    end
    figure; imagesc(img); axis i mage;
    img= zeros(X+2, Z+2);
    [r, c]= si ze(data);
    for i=1: r
        if data(i, 2)<ythres,
            img(data(i, 1)+1, data(i, 3)+1)= data(i, 7);
        end
    end
    figure; imagesc(img); axis i mage;

```

[Published with MATLAB® R2017a](#)

```

function m= vox2surfdi st(segfname, di ms, nvfname, shi ft, ski p)
    di ms= [di ms(2), di ms(1), di ms(3)]; % di ms ori gi na l l y x, y, z but Al ex reorders to y, x, z
    ncol = prod(di ms);
    [y, x, z]= i nd2sub(di ms, 1: ncol );
    v= load(segfname);
    v=reshape(v, 48, 23, 48);
    v=permute(v, [2, 1, 3]);
    v= reshape(v, 48*23*48, 1);
    nv= length(v);
    s= read_nvfv(nvfname);

```

```

ns= length(s);
i= shift: skip: ns;
s= s(i, 1: 3);
ns= length(s);
s= s';
i= find(v > 0);
ni= length(i);
m= zeros(ns, ni);
for j=1: ns
    m(j, :)= sqrt(sum(( [x(i)', y(i)', z(i)']' - repmat(s(:, j), 1, ni)).^2));
    fprintf(1, '%d\n', j);
end

```

CODE ASSOCIATED WITH AIM 2

```

% returns dE/dx for a given electron energy in a medium of specified atomic mass and number
function dEdx = bethe_bloch(electronEnergy, A, Z)
% theta is scattering angle in radians, 0 begin no scatter and pi being 180 degree back scatter
% thickness is the pathlength that the electron travels through the media. the units are
grams/cm^2 (ie density normalized cm)
% electronEnergy is the initial energy of the electron in MeV
% A is the atomic weight of media in grams per mole
% Z is the effective atomic number of the media
c = 299792458; % speed of light in a
vacuum in meters per second
me= 9.1093821545e-31; % electron rest mass
in kg
re= 2.817940289458e-15; % electron radius in
meters
NO= 6.0221417930e23; % Avogadro's number
MeVperJoule= 6.241506363e+12;

velocity= beta_velocity(electronEnergy); % the electron
velocity in meters per second
beta= velocity / c; % electron velocity
fraction of the speed of light
Lorentz_factor= 1 ./ sqrt(1-beta.^2); % Lorentz factor

I=1.201632998e-17; % 75 eV in Joules
Atmp= log(beta.*Lorentz_factor.*sqrt(Lorentz_factor-1)*me*c^2/I); % interim calculation
(untless)
Btmp= (1./(2*Lorentz_factor.^2)).* ((Lorentz_factor-1).^2/8 + 1 -
(2*Lorentz_factor.^2+2*Lorentz_factor-1)*log(2)); % interim calculation (untless)
dEdx= MeVperJoule*4e4*pi*re^2*me*c^2*NO*Z.*(Atmp+Btmp)./(A*beta.^2); % energy lost in MeV per
cm

```

```
% interpolates Bethe's Table II from paper "Moliere's Theory of Multiple Scattering" Physical
Review, vol 89, No 6, Mar 15, 1953
function [f1, f2] = Bethe_table(nu)
tab= [
    0.0  0.8456      2.4929
    0.2  0.7038      2.0694
    0.4  0.3437      1.0488
    0.6 -0.0777     -0.0044
    0.8 -0.3981     -0.6068
    1.0 -0.5285     -0.6359
    1.2 -0.4770     -0.3086
    1.4 -0.3183      0.0525
    1.6 -0.1396      0.2423
    1.8 -0.0006      0.2386
    2.0  0.0782      0.1316
    2.2  0.1054      0.0196
    2.4  0.1008     -0.0467
    2.6  0.08262   -0.0649
    2.8  0.06247   -0.0546
    3.0  0.04550   -0.03568
    3.2  0.03288   -0.01923
    3.4  0.02402   -0.00847
    3.6  0.01791   -0.00264
    3.8  0.01366   0.00005
    4.0  0.010638  0.0010741
];
f1n2= interp1(tab(:, 1), tab(:, [2, 3]), nu);
f1= f1n2(:, 1);
f2= f1n2(:, 2);
```

```
function [di stortPARAMS, y, ci, resnorm, varb, corrb]=
cal c_di storti on_parameters(di stortSTRUCT, nparams)
    n= length(di stortSTRUCT);
    npts= 0;
    for i=1:n
        npts= npts + si ze(di stortSTRUCT(i). poi nts, 1);
    end
    p0= zeros(1, nparams);
    pl b= zeros(1, nparams);
    pub= ones(1, nparams);
    opti ons= opti mset(' Di agnosti cs', ' on', ' MaxFunEval s', 100000);

    di sp(' Start i ng fi t');
```

```

[di stortPARAMS, y, ci , resnorm, varb, corrb] =
fi t_model ('poly_di stortion_md1', p0, di stortSTRUCT, zeros(npts, 1), pl b, pub, [], options);
di sp('Fi ni shed fi tti ng');

```

[Published with MATLAB® R2017a](#)

```

function parameterSTRUCT= cal I VI Scam(worl dptsfname, camptsfroot, di mX, di mY, rdfs)
% worl dptsfname - is the name of the file containing the calibration points (in 3D) as measured
% by Amira,
% where the first column contains the X-values, etc
% camptsfroot - is the root of the filename for the .txt files containing the calibration points
% visible at
% the given angle. The angle is specified in the filename as mX or pX where the X specifies the
% angle in degrees
% and 'm' is for minus and 'p' for plus. The first column contains the index number identifying
% the calibration
% point according to its order of appearance in the worl dptsfname file. Columns 2 and 3 are the X
% and Y values
% respectively.
% di mX and di mY are the dimensions of the I VIS images
% rdfs - is a vector of radial distortion factors (see Hartley and Zisserman's "Multiple View
% Geometry" p 191)

if nargin < 3
    di mX= 480;
end
if nargin < 4
    di mY= di mX;
end
if nargin < 5
    rdfs= [];
end

worl dpts= load(worl dptsfname); % reads worl dpts measured using Amira

% re-arrange axes to match the coordinate system used in Hartley and Zisserman's "Multiple
% View Geometry" p 154

worl dpts(:, 1)= 192 - worl dpts(:, 1); % new X is flipped old X
tmp= worl dpts(:, 2); % save Y
worl dpts(:, 2)= 384 - worl dpts(:, 3); % new Y is flipped old Z
worl dpts(:, 3)= tmp; % new Z is old Y

[angl es, cal fi l eLST]= getI VI Scal angl es(camptsfroot);

campts_all= [];
n= length(angl es);
for i=1: n
    campts= load(debl ank(cal fi l eLST(i, :))); % reads campts matrix
    worl dpts_i ndi ces= campts(:, 1); % extract indices of worl dpts visible at

```

```

current angle
    campts= campts(:, 2:3)'; % campts without indices transposed to column
vectors
    wpts= worldpts(worldpts_indices,:); % select worldpts visible at given angle
transposed to column vectors
    wpts= [wpts; ones(1, size(wpts, 2))]; % add row of ones
    worldptsSTRUCT(i).worldpts= wpts; % accumulate worldpts in structure
    worldptsSTRUCT(i).angle= angles(i); % along with associated angle
    P= perspective_transform_estimate(wpts, campts);
    campts_all = [campts_all, campts]; % accumulate campts
end
campts_all = reshape(campts_all, prod(size(campts_all)), 1); % reshape to a single column
vector

f= 3318;
aor2yRPY= [0, 0, 0];
xzshft= [-mean(worldpts(:, 1)), -mean(worldpts(:, 3))];
wcs2ccsRPY= [0, 0, 0];
Ctilde= [0, mean(worldpts(:, 2))*2, -4000];
p0= [f, aor2yRPY, xzshft, wcs2ccsRPY, Ctilde];
plb= [f-1000, [-pi/4 -pi/4 -pi/4], [-192 -192], [-pi/4 -pi/4 -pi/4], [-1000 -1000 -8000]];
pub= [f+1000, [pi/4 pi/4 pi/4], [192 192], [pi/4 pi/4 pi/4], [1000 1000 2000]];
options= optimset('Diagnostic', 'on', 'MaxFunEvals', 100000, 'MaxIter', 100000);

disp(' Starting fit ');
[p, y, ci, resnorm, varb, corrb] =
fit_model('IVI_Scam_mdl', p0, worldptsSTRUCT, campts_all, plb, pub, [], options, di mX, di mY, rdfs);
disp(' Finished fitting ');

parameterSTRUCT.f= p(1);
parameterSTRUCT.aor2yRPY= p(2:4);
parameterSTRUCT.xzshft= p(5:6);
parameterSTRUCT.wcs2ccsRPY= p(7:9);
parameterSTRUCT.Ctilde= p(10:12);
parameterSTRUCT.rdfs= rdfs;
parameterSTRUCT.dims= [di mX, di mY];

```

[Published with MATLAB® R2017a](#)

```

function world= cam2world(pts1, A1, pts2, A2)
x1= pts1(:, 1);
y1= pts1(:, 2);
x2= pts2(:, 1);
y2= pts2(:, 2);
[r, c]= size(pts1);
world= [];
for i=1:r
    b11=A1(1)-(x1(i)*A1(9));
    b12=A1(2)-(x1(i)*A1(10));
    b13=A1(3)-(x1(i)*A1(11));

```

```

b21=A1(5)-(y1(i)*A1(9));
b22=A1(6)-(y1(i)*A1(10));
b23=A1(7)-(y1(i)*A1(11));
c1=x1(i)-A1(4);
c2=y1(i)-A1(8);
b31=A2(1)-(x2(i)*A2(9));
b32=A2(2)-(x2(i)*A2(10));
b33=A2(3)-(x2(i)*A2(11));
b41=A2(5)-(y2(i)*A2(9));
b42=A2(6)-(y2(i)*A2(10));
b43=A2(7)-(y2(i)*A2(11));
c3=x2(i)-A2(4);
c4=y2(i)-A2(8);
B=[b11 b12 b13; b21 b22 b23; b31 b32 b33; b41 b42 b43];
C=[c1; c2; c3; c4];
R= pinv(B) * C;
world= [world; R'];
end

```

[Published with MATLAB® R2017a](#)

```

function [A1,A2]= camera_calibration(world,cam1,cam2)
[r,c]= size(world);
M=[];
b=[];
for i=1:r
%   M= [M; [0 0 0 0 -world(i,:) -1 (cam1(i,2) * world(i,:))]];
%   M= [M; [world(i,:) 1 0 0 0 0 (-cam1(i,1) * world(i,:))]];
M= [M; [world(i,:) 1 0 0 0 0 (-cam1(i,1) * world(i,:))]];
M= [M; [0 0 0 0 world(i,:) 1 (-cam1(i,2) * world(i,:))]];
b= [b; cam1(i,1); cam1(i,2)];
end
% [U,S,V]= svd(M);
% A1= V(:,end);
A1= pinv(M) * b;
M=[];
b=[];
for i=1:r
%   M= [M; [0 0 0 0 -world(i,:) 1 (cam2(i,2) * world(i,:))]];
%   M= [M; [world(i,:) 1 0 0 0 0 (-cam2(i,1) * world(i,:))]];
M= [M; [world(i,:) 1 0 0 0 0 (-cam2(i,1) * world(i,:))]];
M= [M; [0 0 0 0 world(i,:) 1 (-cam2(i,2) * world(i,:))]];
b= [b; cam2(i,1); cam2(i,2)];
end
% [U,S,V]= svd(M);
% A2= V(:,end);
A2= pinv(M) * b;

```

[Published with MATLAB® R2017a](#)


```

function [psf, distance] =
cerenkov_beta_psf(betaSpectrum, estar, N, mediumRefractiveIndex, A, Z, MAXDIST, NDI VS, FROM_SLOT_FLG)

if nargin < 9, FROM_SLOT_FLG=0; end % default manner in which distance is
determined is NOT from slot
if nargin < 8, NDI VS=25; end % default number of divisions
if nargin < 7, MAXDIST=0.2; end % default spatial extent of psf
output in cm
if nargin < 6, Z= 7.22; end % default Z effective of the
medium (this value is used by Estar and is in Levin's paper) - acrylic is 6.7 according to Levin
and formula in Cherry's book
if nargin < 5, A= 18; end % default atomic mass (this
value is for water)

psf= zeros(NDI VS, 1); % allocate result vector

me= 0.51099891013; % electron rest mass in MeV

betaSpectrum(:, 2)= betaSpectrum(:, 2) / sum(betaSpectrum(:, 2));
cur_beta_energy= bsxfun(@repvals, betaSpectrum(:, 1), round(betaSpectrum(:, 2)*N)); % generate -N
betas according to spectrum
N= numel (cur_beta_energy); % count number actually generated

cur_loc_x= zeros(N, 1); % start N betas at the origin
cur_loc_y= zeros(N, 1); % ...
cur_loc_z= zeros(N, 1); % ...
d0= zeros(N, 1); % distances start at zero
id0= ones(N, 1); % distance indices start at
one

paths_x= cell (N, 1); paths_y= cell (N, 1); paths_z= cell (N, 1);
%%%%%%%%%%%%%%
for i=1:N, paths_x{i}= cur_loc_x(i, 1); paths_y{i}= cur_loc_y(i, 1); paths_z{i}= cur_loc_z(i, 1);
end %%%%%%%%%%%%%%%
Norig= N;

rel_theta= acos(1-2*rand(N, 1)); % choose random isotropic
initial directions specified in polar coordinates with theta distributed uniformly over cos(0:pi)
rel_phi= 2 * pi * rand(N, 1); % and phi uniformly over
(0:2pi) --- see Rajon
sin_theta= sin(rel_theta); % avoid calculating twice (see
next two lines)

cur_dir_x= sin_theta .* cos(rel_phi); % calculate current direction
as Cartesian vector (note - "rel" angles treated as absolute angles)
cur_dir_y= sin_theta .* sin(rel_phi); % ...
cur_dir_z= cos(rel_theta); % ...
d= sqrt(cur_dir_x.^2+cur_dir_y.^2+cur_dir_z.^2); % length of direction vector
cur_dir_x= cur_dir_x ./ d; % normalize to unit length
cur_dir_y= cur_dir_y ./ d; % ...
cur_dir_z= cur_dir_z ./ d; % ...

```

```

i_left= 1:N; % initially, everyone is in
n_left= numel(i_left); % count number in

Estep= 0.030; % each beta will loose Estep
MeV per step
delta_rayThreshold= 0.050; % set threshold for delta ray
production in MeV
stopThreshold= 0.01; % set threshold energy at which
we will stop following a given beta in MeV

Norig= N; % save starting number of
betas (because deltas will be added to N)

while n_left > 0
    dEdx=estar(cur_beta_energy(i_left)); % given current beta energy,
    determine rate of energy loss in MeV per cm - 1st column is collisional loss rate
    r= Estep ./ dEdx(:,1); % distance beta travels
    (in cm) when loosing Estep keV

    cur_loc_x(i_left)= cur_loc_x(i_left) + r .* cur_dir_x(i_left); % determine new location
    cur_loc_y(i_left)= cur_loc_y(i_left) + r .* cur_dir_y(i_left); % ...
    cur_loc_z(i_left)= cur_loc_z(i_left) + r .* cur_dir_z(i_left); % ...

    photons_per_cm =
    frank_tamm_wavelength_integral(560,580,beta_velocity(cur_beta_energy(i_left)),mediumRefractiveIndex) / 100; % rate of Cerenkov production per cm
    photons_per_cm= colvectorize(photons_per_cm);
    % make sure it's a column vector
    if FROM_SLOT_FLG==1
        d= abs(cur_loc_x(i_left));
        % distance from slot along x-axis
    else
        d= sqrt(cur_loc_x(i_left).^2+cur_loc_y(i_left).^2+cur_loc_z(i_left).^2);
        % current distance from origin in cm
    end
    id= ceil(d*NDIVS/MAXDIST);
    % convert distance to indexs
    id(id > NDIVS)= NDIVS;
    % anything over the max goes into the last element
    distinctindx(psf, id0(i_left), id, photons_per_cm .* r);
    % update psf
    id0(i_left)= id;
    % new id now old

    for i=1:N, paths_x{i}= [paths_x{i},cur_loc_x(i,1)]; paths_y{i}= [paths_y{i},cur_loc_y(i,1)];
    paths_z{i}= [paths_z{i},cur_loc_z(i,1)]; end %%%%%%%%%%

    Pdelta= delta_ray_cdf(repvals(delta_rayThreshold,n_left),cur_beta_energy(i_left)) -
    delta_ray_cdf(cur_beta_energy(i_left),cur_beta_energy(i_left)); % area under delta_ray PDF
    between threshold and incident beta energy
    Pdelta= Pdelta .* r; % multiply number of deltas
    per cm by distance beta traveled in last step

```

```

tmp= rand(n_left,1);
i_delta= find(tmp <= Pdelta); % randomly select which
betas have deltas
Ndelta= numel(i_delta); % count number of deltas
% fprintf(1,'%d deltas created\n',Ndelta);
i_not_delta= find(tmp > Pdelta); % and identify which betas
which do not have deltas

if Ndelta > 0
    delta_ray_energies= rand_delta_ray(cur_beta_energy(i_left(i_delta)),delta_rayThreshold);
    % randomly select energy for each delta
    incident_energies= cur_beta_energy(i_left(i_delta));
    % select incident energies

    delta_phi= 2*pi*rand(Ndelta,1);
    % randomly select azimuthal angle for delta ray to be uniformly distributed between 0
and 2pi
    exiting_phi= mod(delta_phi+pi,2*pi);
    % the exiting beta is 180 away from this, still between 0 and 2 pi

    % zenith angles for delta and exiting beta conserve momentum - formulas from
http://www.irs.inms.nrc.ca/EGSnrc/pirs701/node43.html
    delta_theta=
acos(sqrt(delta_ray_energies./incident_energies.*(incident_energies+2*me)./(delta_ray_energies+2*
me)));
    exiting_theta= acos(sqrt((incident_energies-
delta_ray_energies)./(incident_energies.*(incident_energies+2*me)./(incident_energies-
delta_ray_energies+2*me)));

% for j=1:Ndelta, fprintf(1,'incident energy %f produces delta of energy %f traveling with
phi of %f and %f and theta of %f and
%f\n',incident_energies(j),delta_ray_energies(j),exiting_phi(j),delta_phi(j),exiting_theta(j),del
ta_theta(j)); end

cur_beta_energy(i_left(i_delta))= incident_energies - delta_ray_energies;
% decrease energy of incident betas
cur_beta_energy= [cur_beta_energy;delta_ray_energies];
% append delta energies to end of list
id0= [id0;ceil(d(i_delta)*NDIVS/MAXDIST)];
% append starting id to end of list

cur_loc_x= [cur_loc_x;cur_loc_x(i_left(i_delta))];
% append delta locations to end of list (same location as incident)
cur_loc_y= [cur_loc_y;cur_loc_y(i_left(i_delta))];
% ...
cur_loc_z= [cur_loc_z;cur_loc_z(i_left(i_delta))];
% ...

for i=N+1:N+Ndelta, paths_x{i}= cur_loc_x(i,1); paths_y{i}= cur_loc_y(i,1); paths_z{i}=
cur_loc_z(i,1); end %%%%%%%%%%%

cur_dir_x= [cur_dir_x;cur_dir_x(i_left(i_delta))];
% append delta directions to end of list (initially same direction as incident)

```

```

cur_dir_y= [cur_dir_y; cur_dir_y(i_left(i_delta))];
% ..
cur_dir_z= [cur_dir_z; cur_dir_z(i_left(i_delta))];
% ...

i_new_delta= (N+1):N+Ndelta;
% specify indices that point to these deltas
[cur_dir_x, cur_dir_y, cur_dir_z] =
update_dir(cur_dir_x, cur_dir_y, cur_dir_z, i_left(i_delta), exiting_theta, exiting_phi); %
update direction of incident betas
[cur_dir_x, cur_dir_y, cur_dir_z] =
update_dir(cur_dir_x, cur_dir_y, cur_dir_z, i_new_delta, delta_theta, delta_phi); % update
direction of deltas

N= N + Ndelta;
% and count them
end

% next four lines only for betas that did NOT produce deltas
rel_theta= rand_moliere(r(i_not_delta), cur_beta_energy(i_left(i_not_delta)), A, Z); %
randomly select angles distributed according to Moliere
rel_phi= 2 * pi * rand(n_left-Ndelta, 1);
% randomly sample n_left phi angles distributed uniformly over 0:2pi
[cur_dir_x, cur_dir_y, cur_dir_z] =
update_dir(cur_dir_x, cur_dir_y, cur_dir_z, i_left(i_not_delta), rel_theta, rel_phi); % update
direction
% [std(cur_dir_x), std(cur_dir_y), std(cur_dir_z)]
cur_beta_energy(i_left(i_not_delta))= cur_beta_energy(i_left(i_not_delta)) - Estep; %
reduce energy

i_left= find(cur_beta_energy > stopThreshold); % determine which betas are
left
n_left= numel(i_left); % count number left
end

distance= ((1:NDIVS)-0.5)*MAXDIST/NDIVS; % PSF x-axis
psf=psf/(2*trapz(distance, psf)); % normalize so psf from 0 to Inf is 0.5

PLOT=0;
if PLOT==1
figure; for i=1:Norig, plot3(paths_x{i}, paths_y{i}, paths_z{i}, 'LineWidth', 1); hold on; end
% plot beta paths in blue
for i=Norig+1:N, plot3(paths_x{i}, paths_y{i}, paths_z{i}, 'r', 'LineWidth', 1); hold on; end
% plot delta paths in red
axis equal;
xlabel('X'); ylabel('Y'); zlabel('Z');
set(gcf, 'Color', [1, 1, 1]);
set(gca, 'XGrid', 'on', 'YGrid', 'on', 'ZGrid', 'on');
keyboard
figure; plot(distance, psf, '-x'); % plot Cerenkov PSF
ppsf= zeros(NDIVS, 1);
d= sqrt(cur_loc_x(1:Norig).^2+cur_loc_y(1:Norig).^2+cur_loc_z(1:Norig).^2);
% final distance from origin in cm

```

```

id= ceil (d*NDI VS/MAXDI ST);
                                % convert distance to indexs
id(id > NDI VS)= NDI VS;
                                % anything over the max goes into the last element
incindex(ppsf, id);
                                % positron update psf
figure; plot(distance, ppsf, '-x');    % plot positron PSF
end

```

[Published with MATLAB® R2017a](#)

```

% cerenkov_secondary_electron_box- given a specified number of gammas of a given energy spectrum
traveling in isotropic directions uniformly distributed in a box filled with a medium of
specified photon and beta cross-sections and refractive index
% determine the number of Cerenkov photons produced within specified wavelength ranges
% note - the location of the Cerenkov production is taken to be the location where the gamma
knocks off the electron

% photonStartWavelength - vector of lower wavelength thresholds
% photonEndWavelength - vector of upper wavelength thresholds
% gammaSpectrum - two column matrix, 1st column is energy in MeV, 2nd column is abundance (0.511
gammas are taken to be positrons and abundance is that of positron not of 0.511 photons)
% xcom - function handle to photon cross-section function
% estar - function handle to beta cross-section function
% N - desired number of disintegrations
% mediumRefractiveIndex - refractive index of the medium
% boxDims - three element vector specifying box length, width and height
% returns number of Cerenkov photons per wavelength range

function nPhotons =
cerenkov_secondary_electron_box(photonStartWavelength, photonEndWavelength, gammaSpectrum, xcom, esta
r, N, mediumRefractiveIndex, boxDi ms)

nPhotons= zeros(numel (photonStartWavelength), 1);
stopThreshold= 0.01;
we will stop following a given gamma

cur_gamma_energy= bsxfun(@repvals, gammaSpectrum(:, 1), round(gammaSpectrum(:, 2)*N));    % generate
-N*abundance gammas according to spectrum

normalizer= N;
to normalize
N= numel (cur_gamma_energy);
actually generated

rel_theta= acos(1-2*rand(N, 1));
initial directions specified in polar coordinates with theta distributed uniformly over cos(0: pi)
rel_phi= 2 * pi * rand(N, 1);
(0: 2pi) --- see Rajon

```

```

i_511= find(cur_gamma_energy==0.511); % find annihilation photons
cur_gamma_energy= [cur_gamma_energy; cur_gamma_energy(i_511)]; % duplicate gamma and add onto
end
rel_phi= [rel_phi; rel_phi(i_511)]; % within same plane
rel_theta= [rel_theta; rel_theta(i_511)+pi]; % but going in the opposite
direction

fprintf(1, 'added %d 511 keV photons\n', numel (cur_gamma_energy)-N);
N= numel (cur_gamma_energy); % recount total number of gammas
actually generated

sin_theta= sin(rel_theta); % avoid calculating twice (see
next two lines)
cur_dir_x= sin_theta .* cos(rel_phi); % calculate current direction
as Cartesian vector (note - "rel" angles treated as absolute angles)
cur_dir_y= sin_theta .* sin(rel_phi); % ...
cur_dir_z= cos(rel_theta); % ...
d= sqrt(cur_dir_x.^2+cur_dir_y.^2+cur_dir_z.^2); % length of direction vector
cur_dir_x= cur_dir_x ./ d; % normalize to unit length
cur_dir_y= cur_dir_y ./ d; % ...
cur_dir_z= cur_dir_z ./ d; % ...

cur_loc_x= rand(N,1)*boxDims(1); % start N gammas uniformly sampled
within the box
cur_loc_y= rand(N,1)*boxDims(2); % ...
cur_loc_z= rand(N,1)*boxDims(3); % ...

paths_x= cell(N,1); paths_y= cell(N,1); paths_z= cell(N,1);
%%%%%%%%%%%%%%
for i=1:N, paths_x{i}= cur_loc_x(i,1); paths_y{i}= cur_loc_y(i,1); paths_z{i}= cur_loc_z(i,1);
end %%%%%%%%%%%%%%%

i_left= 1:N; % initially, everyone is in
n_left= numel (i_left); % count number in

ONCE=0;

while n_left > 0
    muvCompton= xcom(cur_gamma_energy(i_left), 'Compton'); % Compton cross-section
    muvPhotoelectric= xcom(cur_gamma_energy(i_left), 'Photoelectric'); % photoelectric cross-
section
    muv= muvCompton+muvPhotoelectric; % sum of Compton
scattering and photoelectric cross-sections
    mean_free_path = 1 ./ muv; % inverse is mean free
path
    r= mean_free_path .* rande(n_left,1); % randomly choose
distance traveled by each of the photons before interacting

    cur_loc_x(i_left)= cur_loc_x(i_left) + r .* cur_dir_x(i_left); % determine new
location
    cur_loc_y(i_left)= cur_loc_y(i_left) + r .* cur_dir_y(i_left); % ...
    cur_loc_z(i_left)= cur_loc_z(i_left) + r .* cur_dir_z(i_left); % ...

```

```

i_in= find(cur_loc_x(i_left)>0 & cur_loc_x(i_left)<boxDims(1) & ... %
determine which gammas are still in the box
cur_loc_y(i_left)>0 & cur_loc_y(i_left)<boxDims(2) & ...
cur_loc_z(i_left)>0 & cur_loc_z(i_left)<boxDims(3));

i_left= i_left(i_in); % throw out all
gammas that are outside the box
n_left= numel(i_left); % update count of
number in

fprintf(1,'fraction left= %f\n',n_left / N);

if n_left==0, break; end % escape if none
left

d= sqrt(cur_loc_x(i_left).^2+cur_loc_y(i_left).^2+cur_loc_z(i_left).^2); % current distance
from origin in cm

i_compton= rand(n_left,1) < (muvCompton(i_in) ./ (muvCompton(i_in)+muvPhotoelectric(i_in)));
% flag those undergoing Compton (ie incoherent) scatter
i_photoelectric= i_compton == 0; % assume the rest have
undergone photoelectric interactions

fprintf(1,'of %d, %d compton, %d photoelectric\n',n_left,sum(i_compton),sum(i_photoelectric));

if any(i_compton)
[rel_theta,ce_energy,cp_energy] = rand_compton(cur_gamma_energy(i_left(i_compton))); %
randomly sample thetas and corresponding electron and photon energies from Klein Nishina
cur_gamma_energy(i_left(i_compton))= cp_energy; % update gamma energies
rel_phi = 2 * pi * rand(sum(i_compton),1); % randomly sample
n_left phi angles distributed uniformly over 0:2pi
[cur_dir_x,cur_dir_y,cur_dir_z] =
update_dir(cur_dir_x,cur_dir_y,cur_dir_z,i_left(i_compton),rel_theta,rel_phi); % update
direction
nPhotons = nPhotons +
sum(frank_tamm_double_integral(photonStartWavelength,photonEndWavelength,ce_energy,estar,mediumRefractiveIndex),2); % determine Cerenkov produced by secondary electrons
end

if any(i_photoelectric)
pe_energy= cur_gamma_energy(i_left(i_photoelectric)); % all energy goes to
secondary electron
cur_gamma_energy(i_left(i_photoelectric))= 0; % ...
nPhotons = nPhotons +
sum(frank_tamm_double_integral(photonStartWavelength,photonEndWavelength,pe_energy,estar,mediumRefractiveIndex),2); % determine Cerenkov produced by secondary electrons
end

for i=1:N, paths_x{i}= [paths_x{i},cur_loc_x(i,1)]; paths_y{i}= [paths_y{i},cur_loc_y(i,1)];
paths_z{i}= [paths_z{i},cur_loc_z(i,1)]; end %%%%%%%%%%

i_left= i_left(i_compton & (cur_gamma_energy(i_left) > stopThreshold)); % determine
which gammas are left

```

```

n_left= numel(i_left); % count
number left

% if ONCE==0
%     ONCE=1;
%     figure; for i=1:n_left, plot3(paths_x{i_left(i)},paths_y{i_left(i)},paths_z{i_left(i)});
hold on; end % plot gamma paths in blue
%     axis equal;
%     xlabel('X'); ylabel('Y'); zlabel('Z');
% end
end

nPhotons= nPhotons / normalizer; % adjust to Cerenkov
photons per disintegration

PLOT=0;
if PLOT==1
    figure; for i=1:N, plot3(paths_x{i},paths_y{i},paths_z{i}); hold on; end % plot gamma
paths in blue
    axis equal;
    xlabel('X'); ylabel('Y'); zlabel('Z');
end

```

[Published with MATLAB® R2017a](#)

```

% cerenkov_secondary_electron_psf - given a specified number of gammas of a given energy spectrum
traveling in isotropic directions from the center of an infinite medium of specified photon and
beta cross-sections and refractive index
% determine the number of Cerenkov photons produced within 560 to 580 nm as a function of
distance from the source
% note - the location of the Cerenkov production is taken to be the location where the gamma
knocks off the electron

% gammaSpectrum - two column matrix, 1st column is energy in MeV, 2nd column is relative
frequency (frequencies should sum to 1)
% xcom - function handle to photon cross-section function
% estar - function handle to beta cross-section function
% N - desired number of photons
% mediumRefractiveIndex - refractive index of the medium
% MAXDIST - maximum range over which PSF will be tabulated
% NDIVS - number of elements in PSF

% returns PSF and associated distance scale

function [psf, distance] =
cerenkov_secondary_electron_psf(gammaSpectrum, xcom, estar, N, mediumRefractiveIndex, MAXDIST, NDIVS, FROM_LINE_FLG)

if nargin < 7, FROM_LINE_FLG=0; end
if nargin < 6, NDIVS=25; end % default number of divisions

```



```

if nargin < 5, MAXDIST=0.2; end % default spatial extent of psf
output in cm

psf= zeros(NDIVS,1); % allocate PSF result vector
nrm= zeros(NDIVS,1); % allocate space for normalizer
distance= ((1:NDIVS)-0.5)*MAXDIST/NDIVS; % PSF x-axis
stopThreshold= 0.01; % set threshold energy at which
we will stop following a given gamma

cur_gamma_energy= bsxfun(@repmat, gammaSpectrum(:,1), round(gammaSpectrum(:,2)*N)); % generate
~N gammas according to spectrum

rel_theta= acos(1-2*rand(N,1)); % choose random isotropic
initial directions specified in polar coordinates with theta distributed uniformly over cos(0:pi)
rel_phi= 2 * pi * rand(N,1); % and phi uniformly over
(0:2pi) --- see Rajon

i_511= find(cur_gamma_energy==0.511); % find annihilation photons
cur_gamma_energy= [cur_gamma_energy; cur_gamma_energy(i_511)]; % duplicate gamma and add onto
end
rel_phi= [rel_phi; rel_phi(i_511)]; % within same plane
rel_theta= [rel_theta; rel_theta(i_511)+pi]; % but going in the opposite
direction

N= numel(cur_gamma_energy); % count total number of gammas
actually generated

sin_theta= sin(rel_theta); % avoid calculating twice (see
next two lines)
cur_dir_x= sin_theta .* cos(rel_phi); % calculate current direction
as Cartesian vector (note - "rel" angles treated as absolute angles)
cur_dir_y= sin_theta .* sin(rel_phi); % ...
cur_dir_z= cos(rel_theta); % ...
d= sqrt(cur_dir_x.^2+cur_dir_y.^2+cur_dir_z.^2); % length of direction vector
cur_dir_x= cur_dir_x ./ d; % normalize to unit length
cur_dir_y= cur_dir_y ./ d; % ...
cur_dir_z= cur_dir_z ./ d; % ...

cur_loc_x= zeros(N,1); % start N gammas at the origin
cur_loc_y= zeros(N,1); % ...
cur_loc_z= zeros(N,1); % ...

paths_x= cell(N,1); paths_y= cell(N,1); paths_z= cell(N,1);
%%%%%%%%%%%%%%
for i=1:N, paths_x{i}= cur_loc_x(i,1); paths_y{i}= cur_loc_y(i,1); paths_z{i}= cur_loc_z(i,1);
end %%%%%%%%%%%%%%%

i_left= 1:N; % initially, everyone is in
n_left= numel(i_left); % count number in

while n_left > 0
    muvCompton= xcom(cur_gamma_energy(i_left), 'Compton'); % Compton cross-section
    muvPhotoelectric= xcom(cur_gamma_energy(i_left), 'Photoelectric'); % photoelectric cross-

```

```

section
    muv= muvCompton+muvPhotoelectric; % sum of Compton
scattering and photoelectric cross-sections
    mean_free_path = 1 ./ muv; % inverse is mean free
path
    r= mean_free_path .* rande(n_left,1); % randomly choose
distance traveled by each of the photons before interacting

    cur_loc_x(i_left)= cur_loc_x(i_left) + r .* cur_dir_x(i_left); % determine new
location
    cur_loc_y(i_left)= cur_loc_y(i_left) + r .* cur_dir_y(i_left); % ...
    cur_loc_z(i_left)= cur_loc_z(i_left) + r .* cur_dir_z(i_left); % ...

    if FROM_LINE_FLG==0
        d= sqrt(cur_loc_x(i_left).^2+cur_loc_y(i_left).^2+cur_loc_z(i_left).^2); % current
distance from origin in cm
    else
        d= sqrt(cur_loc_x(i_left).^2+cur_loc_y(i_left).^2); % current distance from line in cm
        (ie ignoring z)
    end

    i_compton= rand(n_left,1) < (muvCompton ./ (muvCompton+muvPhotoelectric)); % flag those
undergoing Compton (ie incoherent) scatter
    i_photoelectric= i_compton == 0; % assume the rest have
undergone photoelectric interactions

    if any(i_compton)
        [rel_theta, ce_energy, cp_energy] = rand_compton(cur_gamma_energy(i_left(i_compton))); %
randomly sample thetas and corresponding electron and photon energies from Klein Nishina
        cur_gamma_energy(i_left(i_compton))= cp_energy; % update gamma energies
        rel_phi = 2 * pi * rand(sum(i_compton), 1); % randomly sample
n_left phi angles distributed uniformly over 0: 2pi
        [cur_dir_x, cur_dir_y, cur_dir_z] =
update_dir(cur_dir_x, cur_dir_y, cur_dir_z, i_left(i_compton), rel_theta, rel_phi); % update
direction
        photons = frank_tamm_double_integral(560, 580, ce_energy, estar, mediumRefractivelndex); %
determine Cerenkov produced by secondary electrons
        id= ceil(d(i_compton)*NDI VS/MAXDIST); % convert distance to
indices
        id(id > NDI VS)= NDI VS; % anything over the
max goes into the last element
        incndx(psf, id, photons); % update PSF
        incndx(nrm, id); % update normalizer
    end

    if any(i_photoelectric)
        ce_energy= cur_gamma_energy(i_left(i_photoelectric)); % all energy goes to
secondary electron
        photons = frank_tamm_double_integral(560, 580, ce_energy, estar, mediumRefractivelndex); %
determine Cerenkov produced by secondary electrons
        id= ceil(d(i_photoelectric)*NDI VS/MAXDIST); % convert distance to
indices
        id(id > NDI VS)= NDI VS; % anything over the

```

```

max goes into the last element
    inci ndx(psf, i d, photons);           % update PSF
    inci ndx(nrm, i d);                   % update normalizer
end

    for i=1:N, paths_x{i}= [paths_x{i}, cur_loc_x(i, 1)]; paths_y{i}= [paths_y{i}, cur_loc_y(i, 1)];
    paths_z{i}= [paths_z{i}, cur_loc_z(i, 1)]; end %%%%%%%%%%%

    i_left= i_left(i_compton & cur_gamma_energy(i_left) > stopThreshold);           % determine
which gammas are left
    n_left= numel (i_left);               % count number left
end

psf= psf ./ nrm;

PLOT=0;
if PLOT==1
    figure; for i=1:N, plot3(paths_x{i}, paths_y{i}, paths_z{i}); hold on; end           % plot gamma
paths in blue
    axis equal;
    xlabel ('X'); ylabel ('Y'); zlabel ('Z');
    figure; plot(distance, psf, '-x');           % plot Cerenkov PSF
end

```

[Published with MATLAB® R2017a](#)

```

function E= cerenkov_threshold(mediumRefracti vel ndex)
c = 299792458;           % c is the speed of light in a vacuum in meters per
second
E0= 0.511;               % rest mass-energy in MeV
v=c./mediumRefracti vel ndex; % threshold beta velocity
E= sqrt(E0.^2./(1-(v/c).^2))-E0; % threshold beta energy

```

[Published with MATLAB® R2017a](#)

```

function checkI VI Scamcal (worldptsfname, camptsfroot, I VI Srootname, I VI SparameterSTRUCT)

worldpts= load(worldptsfname);           % reads worldpts measured using Amira

% re-arrange axes to match the coordinate system used in Hartley and Zisserman's "Multiple
View Geometry" p 154

worldpts(:, 1)= 192 - worldpts(:, 1);           % new X is flipped old X
tmp= worldpts(:, 2);                           % save Y
worldpts(:, 2)= 384 - worldpts(:, 3);           % new Y is flipped old Z
worldpts(:, 3)= tmp;                           % new Z is old Y

```

```

    di mX= I VI SparameterSTRUCT. di ms(1); % I V I S i m a g e
size
    di mY= I VI SparameterSTRUCT. di ms(2);
    px= di mX/2; py= di mY/2; % p r i n c i p a l
point offset fixed at center of image
    f= I VI SparameterSTRUCT. f; % f o c a l l e n g t h
    aor2yRPY= I VI SparameterSTRUCT. aor2yRPY; % r o l l , p i t c h
and yaw to align axis of rotation of the mousebed to the Y-axis
    aor2yXYZ= [I VI SparameterSTRUCT. xzshft(1); 0; I VI SparameterSTRUCT. xzshft(2)]; % X and Z shift
to align axis of rotation of the mousebed to the Y-axis
    wcs2ccsRPY= I VI SparameterSTRUCT. wcs2ccsRPY; % r o l l , p t i c h
and yaw to align world coordinate system to camera coordinate system
    Ctilde= I VI SparameterSTRUCT. Ctilde; % c o o r d i n a t e s o f
the camera center in the world coordinate system
    rdfs= I VI SparameterSTRUCT. rdfs; % r a d i a l
distortion factors

    K= [f 0 px; 0 f py; 0 0 1]; % camera calibration matrix
    x= [0 0 0 1 wcs2ccsRPY 0];
    R= x2t(x', 'rpy'); % rotation matrix representing the orientation of the camera
coordinate frame
    R= R(1:3, 1:3); % reduce R to 3x3
    P3= K * R * [eye(3) -Ctilde']; % projective transform model of the camera

    % determine transformation matrix which aligns the axis of rotation of the mousebed to the Y-
axis
    x= [0 0 0 1 aor2yRPY 0];
    R= x2t(x', 'rpy');
    P1= R * [[eye(3) aor2yXYZ]; [0,0,0,1]];

    [angles, cal fileLST]= getI VI Scal angl es(camptsfroot);
    [angles, di rLST]= getI VI Sangl es(I VI Srootname);

    n= length(angles);
    for i=1:n
        angl e= angl es(i); % rotation of the bed in degrees
        x= [0 0 0 1 0 angl e*pi /180 0 0];
        P2= x2t(x', 'rpy'); % transformation matrix describing rotation of
the bed
        P= P3 * P2 * P1; % align bed, rotate bed, project to camera
coordinate frame

        campts= load(deblank(cal fileLST(i, :))); % reads campts matrix
        worl dpts_i ndices= campts(:, 1); % extract indices of worldpts visible at
current angle
        campts= campts(:, 2:3)'; % campts without indices transposed to column
vectors

        wpts= [worl dpts'; ones(1, size(worl dpts, 1))]; % transpose worldpts and add row of ones

        campts_est= P * wpts; % apply Ps
        campts_est= campts_est(1:2,:) ./ repmat(campts_est(3,:), 2, 1);

```

```

if ~isempty(rdfs)
    % add camera distortion
    r= sqrt((campts_est(1,:)-px).^2+(campts_est(1,:)-py).^2);
    Lr= polyval([rdfs,1],r);
    campts_est(1,:)= (campts_est(1,:)-px). *Lr+px;
    campts_est(2,:)= (campts_est(2,:)-py). *Lr+py;
end
figure;
t=my_image_read([deblank(directLST(i,:)) '\photograph.tif']);
imshow(t,[0,2000]);
colormap('gray');
hold on;
plot(campts_est(1,:),campts_est(2,:), 'r-x');
plot(campts_est(1,1),campts_est(2,1), 'ro');
plot(campts(1,:),campts(2,:), 'b-x');
plot(campts(1,1),campts(2,1), 'bo');
drawnow;
end

```

[Published with MATLAB® R2017a](#)

```

function [histo_c,histo_s] = coincidence_path_length_histogram(mask3d,density,nbins)

if nargin < 2, density= 100; end
if nargin < 3, nbins= 100; end

histo= zeros(nbins,1);           % allocate space for result
dims= size(mask3d);             % mask dimensions
maxlength= sqrt(sum(dims.^2));   % maximum possible pathlength through object is between
diagonal corners
l2dims= ceil(log2(dims));        % log2 of dimensions rounded up
newdims= 2.^l2dims;             % next largest power of two for each dimension
nlevels= min(l2dims);           % number of subsamplings before one of the
dimensions goes to 1

% calculate series of subsampled masks
m{1}= zeros(newdims);
m{1}(1:dims(1),1:dims(2),1:dims(3))= mask3d;
mask3d= m{1};
for llevel=2:nlevels
    m{llevel}= zoomout3(m{llevel-1}); % averages groups of 2x2x2 voxels
    m{llevel}(m{llevel}~=1)= 0;      % zero out all non-ones
end

% at each level, remove voxels corresponding to regions that were entirely within the object
at the lower levels
tmp1= zoomin3(m{nlevels});
for llevel=(nlevels-1):-1:1 % working from the bottom up
    tmp2= m{llevel};        % save current level
    m{llevel}(tmp1==1)= 0;   % zero out voxels corresponding to ones in the lower level
end

```

```

    tmp1= zoomin3(tmp2);          % use blow up of copy in next round
end
% at this point, each level contains only the 1's that are "new" to that level

% initialize photon starting points, final locations and directions
locations= [];
starts= [];
directions= [];

% work from the bottom level up
for level=nllevels:-1:1
    curdims= size(m{level});          % get dimensions of the current level
    indices= find(m{level}==1);       % find the 1's
    nindices= numel(indices);         % count them
    if nindices > 0
        [x,y,z]= ind2sub(curdims,indices);          % convert
to coordinate indices
        [new_locations,new_directions]= init_lines(x,y,z,density,level); % fill
each with photons
        locations= [locations,new_locations];       % add to
list of current locations
        starts= [starts,new_locations];            % add to
list of starting locations
        directions= [directions,new_directions];   % keep
full list of directions
        locations= update_lines(locations,directions,m{level});
    end
    if level > 1
        locations= locations * 2;
        starts= starts * 2;
    end
end
stopflg= 0;
while ~stopflg
    [locations,stopflg]= update_lines(locations,directions,mask3d);
end

R= 45*0.327;          % scanner radius
cylinder= [0.327*dims(1)/2,0.327*dims(2)/2,1,0.327*dims(1)/2,0.327*dims(2)/2,dims(3),R]; %
points at either end of axis followed by the radius
nlines= size(locations,2);
hitflg= zeros(1,nlines);
hitlocs= zeros(3,nlines,2);
for i=1:nlines
    points= intersectLineCylinder([locations(:,i,1)', locations(:,i,1)'+locations(:,i,2)'],
cylinder);
    if size(points,1) == 1
        hitflg(1,i)= 1;
        hitlocs(:,i,1)= points(1,:);
    end
    if size(points,1) == 2
        hitflg(1,i)= 2;
        hitlocs(:,i,1)= points(1,:);
    end
end

```

```

        hitlocs(:, i, 2) = points(2, :);
    end
end
coinc_hitflg = (hitflg == 2);
singl_hitflg = (hitflg >= 1);
d = sqrt(sum( (locations(:, coinc_hitflg, 1) - locations(:, coinc_hitflg, 2)).^2 ));
[h, x] = hist(d, nbins);
histo_c = [x; h];
d = sqrt(sum( cat(2, (locations(:, singl_hitflg, 1) - starts(:, singl_hitflg, 2)).^2,
(locations(:, singl_hitflg, 2) - starts(:, singl_hitflg, 2)).^2 ) ));
[h, x] = hist(d, nbins);
histo_s = [x; h];
p_coinc = sum(histo_c(2, :). * exp(-0.096 * 0.327 * histo_c(1, :))) / nlines; % total coincident
events (ie total activity) times this number estimates the number of trues
p_sngl = sum(histo_s(2, :). * exp(-0.096 * 0.327 * histo_s(1, :))) / nlines; % total singles
events (ie total activity * 2) times this number estimates the number of singles
keyboard
end

function [locations, directions] = init_lines(x, y, z, density, level)
    n = density * 8^(level - 1) * numel(x); % the total number of new rays is the density
times the voxel volume times the number of voxels
    x = rowvectorize(x); % make sure all are row vectors
    y = rowvectorize(y); % ...
    z = rowvectorize(z); % ...
    locations = repmat([x; y; z], 1, n/numel(x)) - rand(3, n); % random starting points within boxes
    locations = cat(3, locations, locations); % make into pair of locations that will travel in
opposite directions
    dir_theta = acos(1 - 2 * rand(1, n)); % choose random isotropic initial directions
specified in polar coordinates with theta distributed uniformly over cos(0:2pi)
    dir_phi = 2 * pi * rand(1, n); % and phi uniformly over (0:2pi) --- see Rajon
    directions = cat(1, dir_theta, dir_phi); % concatenate theta and phi to make a two row
direction matrix
end

function [locations, stopflg] = update_lines(locations, directions, mask)
    curdims = size(mask); % get current dimensions
    sin_theta = sin(directions(1, :)); % avoid calculating
twice (see next two lines)
    dir_x = sin_theta .* cos(directions(2, :)); % calculate current
direction as Cartesian vector
    dir_y = sin_theta .* sin(directions(2, :)); % ...
    dir_z = cos(directions(1, :)); % ...
    nrm = sqrt(dir_x.^2 + dir_y.^2 + dir_z.^2);
    dir_x = dir_x ./ nrm; dir_y = dir_y ./ nrm; dir_z = dir_z ./ nrm;

    stopflg = 1; % start by assuming
all are out

    left = locations(:, :, 1);
    head = left + cat(1, dir_x, dir_y, dir_z); % specify head of ray
as being unit distance away along line in specified direction
    ijk = ceil(left); % determine

```

```

indices of new bounding box within mask for each ray
    in= (ijk(1,:) >= 1) & (ijk(2,:) >= 1) & (ijk(3,:) >= 1) & (ijk(1,:) <= curdims(1)) & (ijk(2,:)
<= curdims(2)) & (ijk(3,:) <= curdims(3));          % vector of 1's and 0's (ie TRUES and FALSES)
indicating for each ray if it is in or out of the grid
    indx= sub2ind(curdims,ijk(1,in),ijk(2,in),ijk(3,in));          % list of indices
into mask that are not outside dimensions of mask - the length of this vector is equal to the
number of 1's in "in"
    in_in= (mask(indx) == 1);          % of these, which are
also in the object defined by mask
    in(in)= in_in;          % now 1's indicate
for each ray that it is both in the grid and within the object defined by the mask
    if any(in)
        stopflg= 0;          % don't
signal stop until all rays are out of object
        box_corner= floor(left);
        [p, t]= ray_box_intersect(left(:,in),head(:,in),box_corner(:,in));          % determine
intersection with sides of bounding box
        left(:,in)= p;          % this is the
new location
        left(:,in)= left(:,in) + cat(1,direct(:,in),direct(:,in),direct(:,in)) * 1.0e-10;          %
keep going just a little further to avoid being right on the face
    end

    right= locations(:, :, 2);
    head= right - cat(1,direct(:,in),direct(:,in),direct(:,in));          % specify head of
ray as being unit distance away along line in specified direction
    ijk= ceil(right);          % determine indices
of new bounding box within mask for each ray
    in= (ijk(1,:) >= 1) & (ijk(2,:) >= 1) & (ijk(3,:) >= 1) & (ijk(1,:) <= curdims(1)) & (ijk(2,:)
<= curdims(2)) & (ijk(3,:) <= curdims(3));          % vector of 1's and 0's (ie TRUES and FALSES)
indicating for each ray if it is in or out of the grid
    indx= sub2ind(curdims,ijk(1,in),ijk(2,in),ijk(3,in));          % list of indices
into mask that are not outside dimensions of mask - the length of this vector is equal to the
number of 1's in "in"
    in_in= (mask(indx) == 1);          % of these, which are
also in the object defined by mask
    in(in)= in_in;          % now 1's indicate
for each ray that it is both in the grid and within the object defined by the mask
    if any(in)
        stopflg= 0;          % don't
signal stop until all rays are out of object
        box_corner= floor(right);
        [p, t]= ray_box_intersect(right(:,in),head(:,in),box_corner(:,in));          % determine
intersection with sides of bounding box
        right(:,in)= p;          % this is
the new location
        right(:,in)= right(:,in) - cat(1,direct(:,in),direct(:,in),direct(:,in)) * 1.0e-10;          % keep
going just a little further to avoid being right on the face
    end

    locations= cat(3,left,right);          % re-pair
end

```



```
function [Locations, starts] = coincidence_paths(mask3d, density)

    if nargin < 2, density = 100; end

    dims = size(mask3d); % mask dimensions
    l2dims = ceil(log2(dims)); % log2 of dimensions rounded up
    newdims = 2.^l2dims; % next largest power of two for each dimension
    nlevels = min(l2dims); % number of subsamplings before one of the
    dimensions goes to 1

    % calculate series of subsampled masks
    m{1} = zeros(newdims);
    m{1}(1:dims(1), 1:dims(2), 1:dims(3)) = mask3d;
    mask3d = m{1};
    for level = 2:nlevels
        m{level} = zoomout3(m{level-1}); % averages groups of 2x2x2 voxels
        m{level}(m{level}~=1) = 0; % zero out all non-ones
    end

    % at each level, remove voxels corresponding to regions that were entirely within the object
    % at the lower levels
    tmp1 = zoomin3(m{nlevels});
    for level = (nlevels-1):-1:1 % working from the bottom up
        tmp2 = m{level}; % save current level
        m{level}(tmp1==1) = 0; % zero out voxels corresponding to ones in the lower level
        tmp1 = zoomin3(tmp2); % use blow up of copy in next round
    end
    % at this point, each level contains only the 1's that are "new" to that level

    % initialize photon starting points, final locations and directions
    Locations = [];
    starts = [];
    directions = [];

    % work from the bottom level up
    for level = nlevels:-1:1
        curdims = size(m{level}); % get dimensions of the current level
        indices = find(m{level}==1); % find the 1's
        nindices = numel(indices); % count them
        if nindices > 0
            [x, y, z] = ind2sub(curdims, indices); % convert
            to coordinate indices
            [new_locations, new_directions] = init_lines(x, y, z, density, level); % fill
            each with photons
            Locations = [Locations, new_locations]; % add to
            list of current locations
            starts = [starts, new_locations]; % add to
            list of starting locations
            directions = [directions, new_directions]; % keep
        end
    end
```

```

full list of directions
    locations= update_lines(locations, directions, m{level});
end
if level > 1 % when deeper than the 1st level
    locations= locations * 2; % scale doubles in anticipation of moving up
    starts= starts * 2; % ...
end
end
stopflag= 0; % assume no stop
while ~stopflag % loop until stopflag is set
    [locations, stopflag]= update_lines(locations, directions, mask3d); % update lines until
all are out
end
end

function [locations, directions] = init_lines(x, y, z, density, level)
    n= density * 8^(level-1) * numel(x); % the total number of new rays is the density
times the voxel volume times the number of voxels
    x= rowvectorize(x); % make sure all are row vectors
    y= rowvectorize(y); % ...
    z= rowvectorize(z); % ...
    locations= repmat([x; y; z], 1, n/numel(x)) - rand(3, n); % random starting points within boxes
    locations= cat(3, locations, locations); % make into pair of locations that will travel in
opposite directions
    dir_theta= acos(1-2*rand(1, n)); % choose random isotropic initial directions
specified in polar coordinates with theta distributed uniformly over cos(0:2pi)
    dir_phi= 2 * pi * rand(1, n); % and phi uniformly over (0:2pi) --- see Rajon
    directions= cat(1, dir_theta, dir_phi); % concatenate theta and phi to make a two row
direction matrix
end

function [locations, stopflag]= update_lines(locations, directions, mask)
    curdims= size(mask); % get current dimensions
    sin_theta= sin(directions(1,:)); % avoid calculating
twice (see next two lines)
    dir_x= sin_theta .* cos(directions(2,:)); % calculate current
direction as Cartesian vector
    dir_y= sin_theta .* sin(directions(2,:)); % ...
    dir_z= cos(directions(1,:)); % ...
    nrm= sqrt(dir_x.^2 + dir_y.^2 + dir_z.^2);
    dir_x= dir_x ./ nrm; dir_y= dir_y ./ nrm; dir_z= dir_z ./ nrm;

    stopflag= 1; % start by assuming
all are out

    left= locations(:, :, 1);
    head= left + cat(1, dir_x, dir_y, dir_z); % specify head of ray
as being unit distance away along line in specified direction
    ijk= ceil(left); % determine
indices of new bounding box within mask for each ray
    in= (ijk(1,:) >= 1) & (ijk(2,:) >= 1) & (ijk(3,:) >= 1) & (ijk(1,:) <= curdims(1)) & (ijk(2,:)
<= curdims(2)) & (ijk(3,:) <= curdims(3)); % vector of 1's and 0's (ie TRUES and FALSES)
indicating for each ray if it is in or out of the grid

```

```

    indx= sub2ind(curdims,ijk(1,in),ijk(2,in),ijk(3,in)); % list of indices
into mask that are not outside dimensions of mask - the length of this vector is equal to the
number of 1's in "in"
    in_in= (mask(indx) == 1); % of these, which are
also in the object defined by mask
    in(in)= in_in; % now 1's indicate
for each ray that it is both in the grid and within the object defined by the mask
    if any(in)
        stopflg= 0; % don't
signal stop until all rays are out of object
        box_corner= floor(left);
        [p,t]= ray_box_intersect(left(:,in),head(:,in),box_corner(:,in)); % determine
intersection with sides of bounding box
        left(:,in)= p; % this is the
new location
        left(:,in)= left(:,in) + cat(1,dx(:,in),dy(:,in),dz(:,in)) * 1.0e-10; %
keep going just a little further to avoid being right on the face
    end

    right= locations(:,2);
    head= right - cat(1,dx,dy,dz); % specify head of
ray as being unit distance away along line in specified direction
    ijk= ceil(right); % determine indices
of new bounding box within mask for each ray
    in= (ijk(1,:) >= 1) & (ijk(2,:) >= 1) & (ijk(3,:) >= 1) & (ijk(1,:) <= curdims(1)) & (ijk(2,:)
<= curdims(2)) & (ijk(3,:) <= curdims(3)); % vector of 1's and 0's (ie TRUES and FALSES)
indicating for each ray if it is in or out of the grid
    indx= sub2ind(curdims,ijk(1,in),ijk(2,in),ijk(3,in)); % list of indices
into mask that are not outside dimensions of mask - the length of this vector is equal to the
number of 1's in "in"
    in_in= (mask(indx) == 1); % of these, which are
also in the object defined by mask
    in(in)= in_in; % now 1's indicate
for each ray that it is both in the grid and within the object defined by the mask
    if any(in)
        stopflg= 0; % don't
signal stop until all rays are out of object
        box_corner= floor(right);
        [p,t]= ray_box_intersect(right(:,in),head(:,in),box_corner(:,in)); % determine
intersection with sides of bounding box
        right(:,in)= p; % this is
the new location
        right(:,in)= right(:,in) - cat(1,dx(:,in),dy(:,in),dz(:,in)) * 1.0e-10; % keep
going just a little further to avoid being right on the face
    end

    locations= cat(3,left,right); % re-pair
end

```

[Published with MATLAB® R2017a](#)

```

function pdf =
compton_electron_energy_spectrum_per_gamma_in_h2o(gamma_ni_tia_l Energy, nSteps, nGammas)
eHist=zeros(nSteps, 1);
for i=1:nGammas
    e= gamma_ni_tia_l Energy;
    while e > (gamma_ni_tia_l Energy / (nSteps*0.1))
        [photon_angle, ce_energy, e] = rand_compton(e);
        index= ceil(nSteps*ce_energy/gamma_ni_tia_l Energy);
        eHist(index)= eHist(index) + 1;
        fprintf(1, 'energy=%f\n', e);
    end
    fprintf(1, 'gamma %d\n', i);
end
energies= gamma_ni_tia_l Energy*(1:nSteps)/nSteps - 0.5 * gamma_ni_tia_l Energy / nSteps;
pdf= [energies', eHist/sum(eHist)];

```

[Published with MATLAB® R2017a](#)

```

function [T, dedT, nu]= compton_energy_spectrum(hv)
% eq references refer to "The Atomic Nucleus" by Robley Evans 1955
% nu is the scattered photon angle relative to incident photon
% hv is the energy of the incident photon
r0= 2.818e-13;      % classical electron radius pg 822
m0c2= 0.511;       % MeV
alpha= hv / m0c2;
nu= (1:179) * pi / 180;
phi = acot((1+alpha)*tan(nu/2));
cos_sq_phi = cos(phi).^2;
one_minus_cos_nu= 1 - cos(nu);
one_plus_alpha_sq= (1 + alpha)^2;
% eq 1.10 on pg 676
T= (hv*alpha*one_minus_cos_nu) ./ (1+alpha*one_minus_cos_nu);
hv_prime= hv - T;
hv_prime_over_hv= hv_prime / hv;
% eq 2.8 on pg 683
part1= (r0^2/2) * (hv_prime_over_hv.^2) .* ((1 ./ hv_prime_over_hv) + hv_prime_over_hv -
(sin(nu).^2));
% 2nd half of eq 5.2 on pg 692
part2= (2*pi/(alpha^2*m0c2)) * ((one_plus_alpha_sq - alpha^2 * cos_sq_phi) ./ (one_plus_alpha_sq
- alpha * (2+alpha) * cos_sq_phi)).^2;
% eq 5.2 on pg 692
dedT= part1 .* part2;
%dedT= part1;

```

[Published with MATLAB® R2017a](#)

```

function cdf = del ta_ray_cdf(del ta_rayEnergy, betaEnergy)
% del ta_rayEnergy - vector of pssoble del ta ray energies in MeV
% betaEnergy - energy of the beta particle in MeV
c = 299792458; % speed of light in a
vacuum in meters per second
me= 9. 1093821545e-31; % electron rest mass
in kg
re= 2. 817940289458e-15; % electron radius in
meters
NO= 6. 0221417930e23; % Avogadro's number

MeVperJoul e= 6. 241506363e+12; % Joules to MeV conversion
factor
vel oci ty= beta_vel oci ty(betaEnergy); % the beta particle
velocity in meters per second
beta= vel oci ty / c; % electron velocity
fraction of the speed of light

re= re * 100; % convert from
meters to cm
me= me * c^2; % convert from kg to
Joules
me= me * MeVperJoul e; % convert from Joules
to MeV

cdf= 2*pi*re^2*me*NO ./ (beta.^2.*del ta_rayEnergy); % result is number of
del ta rays per cm

```

[Published with MATLAB® R2017a](#)

```

function pdf = del ta_ray_pdf(del ta_rayEnergy, betaEnergy)
% del ta_rayEnergy - vector of pssoble del ta ray energies in MeV
% betaEnergy - energy of the beta particle in MeV
c = 299792458; % speed of light in a
vacuum in meters per second
me= 9. 1093821545e-31; % electron rest mass
in kg
re= 2. 817940289458e-15; % electron radius in
meters
NO= 6. 0221417930e23; % Avogadro's number

MeVperJoul e= 6. 241506363e+12; % Joules to MeV conversion
factor
vel oci ty= beta_vel oci ty(betaEnergy); % the beta particle
velocity in meters per second
beta= vel oci ty / c; % electron velocity
fraction of the speed of light

re= re * 100; % convert from

```

```

meters to cm
me= me * c^2; % convert from kg to
Joules
me= me * MeVperJoule; % convert from Joules
to MeV

pdf= 2*pi*re^2*me*N0 ./ (beta.^2.*del ta_rayEnergy.^2); % result is number of
del ta rays per cm per MeV

```

[Published with MATLAB® R2017a](#)

```

% calculates mean relative sensitivity of IVIS 200 camera over range distances from focal point
(1.5 cm below up to depth cm above that)
function factor = depth_adjustment_factor(depth, refi ndx)
% depth - depth of the fluid in cm
factor = quad(@(x)height_adj ustment_factor(x, refi ndx, depth), -1.5, depth-1.5) / depth;

```

[Published with MATLAB® R2017a](#)

```

% calculates expansion terms for determination of electron scattering pdf as a function of
normalized angle nu according to Moliere's theory
% see Bethe, "Moliere's Theory of Multiple Scattering", Physical Review, vol 89, No 6, Mar 15,
1953
% and Levin and Hoffman, "Calculation of positron range and its effect on the fundamental limit
of positron emission tomography system spatial resolution", PMB, vol 44, 1999
function [f0, f1, f2] = f_of_nu(nu)
x= nu.^2; % see Bethe eqn
24a
x=nu;
f0= 2*exp(-x); % see Bethe eqn 27
(Levin eqn 8 is wrong)
f1= zeros(size(f0)); % allocate space
f2= zeros(size(f0)); % ...
i= find(nu >= 4); % select large
nu
if numel(i) > 0 % for large nu -
use eqns
f1(i)= 2*(1-5*x(i).^(-2)).^(-4/5) ./ x(i).^4; % see
Levin eqn 8
f2(i)= 16*(log(x(i))+log(0.4)) ./ (x(i).^6 .* (1-9*x(i).^(-2)-24*x(i).^(-4))); % ...
end
j= find(nu < 4); % select small
nu
if numel(j) > 0 % for small nu -
use table
[tf1, tf2]= Bethe_table(nu(j)); % interpolate

```

```

Bethe's data
    f1(j)= tf1; % transcribe
entries
    f2(j)= tf2; % ...
end

```

[Published with MATLAB® R2017a](#)

```

function photonSecondsPerMeter =
frank_tamm(photonFrequency, particleVelocity, particleCharge, mediumRelativePermeability, mediumRefractiveIndex)
% photonFrequency is the frequency of the Cerenkov photon in (1/seconds)
% particleVelocity - is the speed of the particle (meters/sec)
% particleCharge is the electric charge of the particle (elementary charge i.e. # of protons)
% mediumRelativePermeability - is the permeability of the medium relative to that of free space
% mediumRefractiveIndex - is the index of refraction of the medium (unitless), it is a function
of lambda
% returns photonSecondsPerMeter

c = 299792458 ; % c
is the speed of light in a vacuum in meters per second
h = 6.62606896e-34; %
Planks constant in Joules*seconds
mu0= 4*pi*1e-7; %
the permeability of free space in Joules*seconds^2/(Coulombs^2*meters)
CoulombsPerElementaryCharge= 1.602176487e-19; % the number of Coulombs
per proton
mediumPermeability= mediumRelativePermeability * mu0; % mediumPermeability in
Joules*seconds^2/(Coulombs^2*meters)
betasq= c^2 ./ (particleVelocity.^2 .* mediumRefractiveIndex.^2); % beta is the ratio
of the speed of light in the medium to the speed of the particle
JouleSecondsPerMeter = (pi * mediumPermeability .*
(CoulombsPerElementaryCharge*particleCharge).^2) .* photonFrequency .* (1 - betasq); %
Cerenkov energy at specified wavelength
JouleSecondsPerMeter(JouleSecondsPerMeter<0)= 0; % Cerenkov radiation only
produced when beta > 1 so zero out negatives
photonEnergy= photonFrequency * h; % energy in
Joules of a single photon of specified wavelength
photonSecondsPerMeter= JouleSecondsPerMeter ./ photonEnergy; % convert to number of photon-
seconds per meter

```

[Published with MATLAB® R2017a](#)

```

% frank_tamm_double_integral - integrates Frank-Tamm formula over wavelength range(s) and over
full path length of beta
function photons =

```

```

frank_tamm_double_integral (photonStartWavelength, photonEndWavelength, initialBetaEnergy, estar, mediumRefractiveIndex)
% photonStartWavelength is the start of the range of wavelengths of the Cerenkov photons in
nanometers
% photonEndWavelength is the start of the range of wavelengths of the Cerenkov photons in
nanometers
% initialBetaEnergy - starting energy of the beta particle in MeV
% estar - handle to function determining energy loss in MeV per cm for a given beta energy
% mediumRefractiveIndex - is the index of refraction of the medium (unitless), it is a function
of lambda
% returns matrix of size (number of wavelengths, number of betas) containing the total number of
Cerenkov photons within range of wavelengths produced by betas of given initial energies and
parameters of the medium

c = 299792458; % c is the speed of light
in a vacuum in meters per second

nw= numel (photonStartWavelength); % the number of start
wavelengths
if numel (photonEndWavelength) ~= nw % must match the number of end
wavelengths
    error('number of start and end wavelengths must match');
end
nv= numel (initialBetaEnergy); % the number of betas

photons= zeros(nw, nv); % initialize photon
accumulator
betaEnergy= colvectorize(initialBetaEnergy); % initialize beta energy and
force to be a column vector
betaVelocity= beta_velocity(betaEnergy); % initialize beta
velocities

while any( (mediumRefractiveIndex*betaVelocity/c) > 1 ) % keep looping so long as at
least one beta velocity is superluminal
    Estep= betaEnergy * 1e-3; Estep(Estep<0.0001)= 0.0001; % specify energy step in MeV as
0.1% of betaEnergy but bottoming out at 0.1 keV
    photonsPerMeter =
frank_tamm_wavelength_integral (photonStartWavelength, photonEndWavelength, betaVelocity, mediumRefractiveIndex); % determine Cerenkov production rate in photons per meter
    dEdx= estar(betaEnergy, 'Total ');
% determine beta energy loss rate MeV per
cm
    distance= rowvectorize(1e-2 * Estep ./ dEdx);
% determine distance in meters that beta
moves in losing Estep
    distance(isnan(distance))= 0;
% zero out NaN's (i.e. Estep/dEdx = 0/0)
    distance(isinf(distance))= 0;
% zero out inf's (i.e. Estep/dEdx = ?/0)
    photons= photons + repmat(distance, nw, 1) .* photonsPerMeter;
% accumulate Cerenkov photons generated in that
distance
    betaEnergy= betaEnergy - Estep;

```



```

                                % loose Estep energy
betaEnergy(betaEnergy<0)= 0;

                                % careful not to go below zero
betaVelocity= beta_velocity(betaEnergy);

                                % recalc beta velocities
end

```

[Published with MATLAB® R2017a](#)

```

% frank_tamm_triple_integral - integrates Frank-Tamm formula over wavelength range(s), over full
% path length of beta and over a given beta spectrum
function photons =
frank_tamm_triple_integral (photonStartWavelength, photonEndWavelength, betaSpectrum, estar, mediumRefractiveIndex)
% photonStartWavelength is the start of the range of wavelengths of the Cerenkov photons in
% nanometers
% photonEndWavelength is the start of the range of wavelengths of the Cerenkov photons in
% nanometers
% betaSpectrum - table with two columns, energy in MeV and probability
% estar - handle to function determining energy loss in MeV per cm for a given beta energy
% mediumRefractiveIndex - is the index of refraction of the medium (unitless), it is a function
% of lambda
% returns vector of size equal to the number of wavelengths, containing the total number of
% Cerenkov photons within each range of wavelengths produced by betas of given beta spectrum and
% parameters of the medium

c = 299792458;                                % c is the speed of light
in a vacuum in meters per second

[ne, two]= size(betaSpectrum);                  % get beta spectrum table
dimensions
if two ~= 2
    error('beta spectrum must be two columns: energy and probability');
end
betaSpectrum(:,2)= betaSpectrum(:,2) / sum(betaSpectrum(:,2)); % force probabilities to sum to
1

nw= numel (photonStartWavelength);              % the number of start
wavelengths
if numel (photonEndWavelength) ~= nw            % must match the number of end
wavelengths
    error('number of start and end wavelengths must match');
end

% photons= zeros(nw,1);                        % initialize photon
accumulator

% for i=1:ne
% photons = photons + betaSpectrum(i,2) *
frank_tamm_double_integral (photonStartWavelength, photonEndWavelength, betaSpectrum(i,1), estar, medi

```

```

umRefractiveIndex);
% end

photons = sum(repmat(rowvectorize(betaSpectrum(:, 2)), nw, 1) .*
frank_tamm_double_integral (photonStartWavelength, photonEndWavelength, betaSpectrum(:, 1), estar, medi
umRefractiveIndex), 2);

```

[Published with MATLAB® R2017a](#)

```

% frank_tamm_wavelength_integral - calculates integral of Frank Tamm assuming charge of +/- 1
(i.e. a beta) and relative permeability of 1
function photonsPerMeter =
frank_tamm_wavelength_integral (photonStartWavelength, photonEndWavelength, particleVelocity, mediumR
efractiveIndex)
% photonStartWavelength is the start of the range of wavelengths of the Cerenkov photons in
nanometers
% photonEndWavelength is the start of the range of wavelengths of the Cerenkov photons in
nanometers
% particleVelocity - is the speed of the particle (meters/sec)
% mediumRefractiveIndex - is the index of refraction of the medium (unitless), it is a function
of lambda
% returns photonsPerMeter

c = 299792458; % c is the speed of light
in a vacuum in meters per second
alpha= 7.297352537650e-3; % the fine structure
constant (unitless)

photonStartWavelength= colvectorize(photonStartWavelength) / 1e9; % make into column and convert
to meters
photonEndWavelength= colvectorize(photonEndWavelength) / 1e9; % make into column and convert to
meters
if any(photonStartWavelength >= photonEndWavelength) % ensure order is correct
    error('start wavelength must be less than end wavelength');
end
particleVelocity= rowvectorize(particleVelocity); % make into row

nw= numel (photonStartWavelength); % the number of start
wavelengths
if numel (photonEndWavelength) ~= nw % must match the number of end
wavelengths
    error('number of start and end wavelengths must match');
end
nv= numel (particleVelocity); % the number of betas

beta= particleVelocity / c; % particle relative
phase velocity
% result is matrix of size nw by nv -- see Measurement of B-Emitting Nuclides Using Cerenkov
Radiation by HH Ross in Analytical Chemistry (41) 10, Aug 1969 p 1260
photonsPerMeter= 2*pi*alpha * ((1./photonStartWavelength)-(1./photonEndWavelength)) * (1 - (1 ./

```

```
(beta.^2 * mediumRefractiveIndex^2));
photonsPerMeter(photonsPerMeter<0)= 0; % zero out negatives which are
indicative of beta*mediumRefractiveIndex < 1
```

[Published with MATLAB® R2017a](#)

```
function f = height_adjustment_factor(d, reindex, depth)
% d - distance from focus point assuming 13 cm FOV on the IVIS 200
% depth - depth of the fluid
A= 6.35; % aperature radius in cm
H= 51.2; % lens to focal point distance in cm
C= 1-cos(atan(A/H));
f= (1-cos(atan(A ./ (H-d)))) / C; % sensitivity of camera relative to sensitivity at focal
point (i.e. relative to d=0)
% calc magnification correction factors http://scubageek.com/articles/wwwbigr.html
D= depth - d -1.5; % distance d is below the surface
L= H - depth + 1.5; % distance from lens to the fluid surface in cm
R= 5; % distance from lens to CCD
M = (D+L+R)/(D*reindex+L+R); % magnification factor
f= f ./ M; % sensitivity is inverse of magnification
```

[Published with MATLAB® R2017a](#)

```
% IVIScam_mdI - IVIS optical imager with rotating bed modelled as a basic pinhole camera - see
Hartley and Zisserman p153+
function campts_all = IVIScam_mdI (p, worldptsSTRUCT, dimX, dimY, rdfs)
px= dimX/2; py= dimY/2; % principal point offset fixed at center of image
f= p(1); % focal length
aor2yRPY= p(2:4); % roll, ptich and yaw to align axis of rotation of the mousebed to
the Y-axis
aor2yXYZ= [p(5);0;p(6)]; % X and Z shift to align axis of rotation of the mousebed to the Y-
axis
wcs2ccsRPY= p(7:9); % roll, ptich and yaw to align world coordinate system to camera
coordinate system
Ctilde= p(10:12); % coordinates of the camera center in the world coordinate system

K= [f 0 px; 0 f py; 0 0 1]; % camera calibration matrix
x= [0 0 0 1 wcs2ccsRPY 0];
R= x2t(x', 'rpy'); % rotation matrix representing the orientation of the camera
coordinate frame
R= R(1:3,1:3); % reduce R to 3x3
P3= K * R * [eye(3) -Ctilde']; % projective transform model of the camera

% determine transformation matrix which aligns the axis of rotation of the mousebed to the Y-
axis
x= [0 0 0 1 aor2yRPY 0];
```

```

R= x2t(x', 'rpy');
P1= R * [[eye(3) aor2yXYZ]; [0,0,0,1]];

campts_all = [];
n= length(worldptsSTRUCT);
for i=1:n
    angle= worldptsSTRUCT(i).angle; % rotation of the bed in degrees
    worldpts= worldptsSTRUCT(i).worldpts;
    x= [0 0 0 1 0 angle*pi/180 0 0];
    P2= x2t(x', 'rpy'); % transformation matrix
    describing rotation of the bed
    campts= P3 * P2 * P1 * worldpts; % align bed, rotate bed, project
    to camera coordinate frame
    campts= campts(1:2,:) ./ repmat(campts(3,:),2,1); % normalize to 2D space
    if nargin >= 5
        % add camera distortion
        r= sqrt((campts(1,:)-px).^2+(campts(2,:)-py).^2);
        Lr= polyval([rdfs,1],r);
        campts(1,:)= (campts(1,:)-px)./Lr+px;
        campts(2,:)= (campts(2,:)-py)./Lr+py;
    end
    campts_all = [campts_all, campts]; % accumulate campts
end
campts_all = reshape(campts_all, prod(size(campts_all)),1); % reshape as column vector

```

[Published with MATLAB® R2017a](#)

```

% returns the PDF for specified angle(s) for electrons undergoing multiple scattering events off
% nuclei as described by Moliere
function pdf = moliere_pdf(theta, thickness, electronEnergy, A, Z)
% theta is scattering angle in radians, 0 begin no scatter and pi being 180 degree back scatter
% thickness is the pathlength that the electron travels through the media. the units are
% grams/cm^2 (ie density normalized cm)
% electronEnergy is the initial energy of the electron in MeV
% A is the atomic weight of media in grams per mole
% Z is the effective atomic number of the media
c = 299792458; % speed of light in a
vacuum in meters per second
h= 6.6260689633e-34; % Planck's constant in
Joule*seconds = kg*meter^2/second
me= 9.1093821545e-31; % electron rest mass
in kg
alpha= 7.297352537650e-3; % the fine structure
constant (unitless)
NO= 6.0221417930e23; % Avogadro's number

velocity= beta_velocity(electronEnergy); % the electron
velocity in meters per second
hbar= h/(2*pi); % reduced Planck
beta= velocity / c; % electron velocity

```

```

fraction of the speed of light
Lorentz_factor= 1 ./ sqrt(1-beta.^2); % Lorentz factor
momentum= Lorentz_factor .* velocity .* me; % relativistic
momentum in kg * meters / second
lambda_bar= hbar./momentum; % the electron DeBroglie
wavelength in meters
e= sqrt(alpha*c*hbar); % using definition of
fine structure constant - calc e in kg^0.5*meters^1.5/second
a0= hbar^2/(me*e^2); % the Bohr radius in
meters (same as classical electron radius / square of fine structure constant

zalpha= Z*e^2./(hbar*velocity); % the alpha from
Bethe's eqn 21a;

X0= lambda_bar / (0.885*a0*Z^(-1/3)); % unitless critical
angle -see Levin eqn 5
Xa= sqrt(X0.^2.*(1.13+3.76*zalpha.^2)); % unitless
characteristic screening angle - see Levin eqn 4
C= 40000*pi*N0*0.885^2*hbar^2/(me^2*c^2*1.167*1.13);
C= 6680;
b = log(thickness*C*(Z+1)*Z^(1/3) ./ (beta.^2*A.*(1+(3.76/1.13)*zalpha.^2))); % normalized
distance parameter - see Bethe eqn 22
Xc= sqrt(1.167*Xa.^2.*exp(b)); % unitless minimum
scattering angle - see Levin eqn 2

if any(exp(b)<17), fprintf(1,'warning - number of collisions (%f) less than 17\n',min(exp(b)));
end
%N= N0 / A;
%Xc= sqrt(40000*pi*N*thickness*e^4*Z*(Z+1)/(momentum*velocity)^2)
%b= log(Xc^2/(1.167*Xa^2))

B= Bfun(b); % solve for B - see
Levin pg 784
nu= theta./(Xc.*sqrt(B)); % see Levin pg 784
[f0,f1,f2]= f_of_nu(nu); % calc f0, f1
and f2 according to Levin pg 784
pdf= f0 + f1./B + f2./B.^2; % and the answer is
...

function Bat_b = Bfun(at_b)
if any(at_b < 1) || any(at_b > 28)
    error('at_b must be between 1 and 28');
end
B=logspace(0,1.5,100);
b=B-log(B);
Bat_b= interp1(b,B,at_b);

```

[Published with MATLAB® R2017a](#)

```

function histo = path_length_histogram(mask3d, density, nbins)

    if nargin < 2, density= 100; end
    if nargin < 3, nbins= 100; end

    histo= zeros(nbins,1);           % allocate space for result
    dims= size(mask3d);             % mask dimensions
    maxlen= sqrt(sum(dims.^2));      % maximum possible pathlength through object is between
    diagonal corners
    l2dims= ceil(log2(dims));        % log2 of dimensions rounded up
    newdims= 2.^l2dims;              % next largest power of two for each dimension
    nlevels= min(l2dims);            % number of subsamplings before one of the
    dimensions goes to 1

    % calculate series of subsampled masks
    m{1}= zeros(newdims);
    m{1}(1:dims(1), 1:dims(2), 1:dims(3))= mask3d;
    mask3d= m{1};
    for level=2:nlevels
        m{level}= zoomout3(m{level-1}); % averages groups of 2x2x2 voxels
        m{level}(m{level}~=1)= 0;      % zero out all non-ones
    end

    % at each level, remove voxels corresponding to regions that were entirely within the object
    % at the lower levels
    tmp1= zoomin3(m{nlevels});
    for level=(nlevels-1):-1:1 % working from the bottom up
        tmp2= m{level}; % save current level
        m{level}(tmp1==1)= 0; % zero out voxels corresponding to ones in the lower level
        tmp1= zoomin3(tmp2); % use blow up of copy in next round
    end
    % at this point, each level contains only the 1's that are "new" to that level

    % initialize photon starting points, final locations and directions
    locations= [];
    starts= [];
    directions= [];

    % work from the bottom level up
    for level=nlevels:-1:1
        curdims= size(m{level}); % get dimensions of the current level
        indices= find(m{level}==1); % find the 1's
        nindices= numel(indices); % count them
        if nindices > 0
            [x,y,z]= ind2sub(curdims, indices); % convert
            to coordinate indices
            [new_locations, new_directions]= init_rays(x,y,z,density,level); % fill
            each with photons
            locations= [locations, new_locations]; % add to
            list of current locations
            starts= [starts, new_locations]; % add to
            list of starting locations

```

```

        directions= [directions, new_directions]; % keep
full list of directions
        locations= update_rays(locations, directions, m[level]);
    end
    if level > 1
        locations= locations * 2;
        starts= starts * 2;
    end
end
stopflag= 0;
while ~stopflag
    [locations, stopflag]= update_rays(locations, directions, mask3d);
end

% need to keep updating till all rays leave the mask -- also need to terminate based on mask
% only at the final level - instead keep rays on hold (ie don't extend but don't terminate)

d= sqrt(sum( (locations-starts).^2 ));
[h, x]= hist(d, nbins);
histo= [x, h];
keyboard
end

function [locations, directions] = init_rays(x, y, z, density, level)
    n= density * 8^(level-1) * numel(x); % the total number of new rays is the density
times the voxel volume times the number of voxels
    x= rowvectorize(x); % make sure all are row vectors
    y= rowvectorize(y); % ...
    z= rowvectorize(z); % ...
    locations= repmat([x; y; z], 1, n/numel(x)) - rand(3, n); % random starting points within boxes
    dir_theta= acos(1-2*rand(1, n)); % choose random isotropic initial directions
specified in polar coordinates with theta distributed uniformly over cos(0:2pi)
    dir_phi= 2 * pi * rand(1, n); % and phi uniformly over (0:2pi) --- see Rajon
    directions= cat(1, dir_theta, dir_phi); % concatenate theta and phi to make a two row
direction matrix
end

function [locations, stopflag]= update_rays(locations, directions, mask)
    curdims= size(mask); % get current dimensions
    sin_theta= sin(directions(1,:)); % avoid calculating
twice (see next two lines)
    dir_x= sin_theta .* cos(directions(2,:)); % calculate current
direction as Cartesian vector
    dir_y= sin_theta .* sin(directions(2,:)); % ...
    dir_z= cos(directions(1,:)); % ...
    nrm= sqrt(dir_x.^2 + dir_y.^2 + dir_z.^2);
    dir_x= dir_x ./ nrm; dir_y= dir_y ./ nrm; dir_z= dir_z ./ nrm;
    head= locations + cat(1, dir_x, dir_y, dir_z); % specify head of ray
as being unit distance away along line in specified direction
    ijk= ceil(locations); % determine indices
of new bounding box within mask for each ray
    in= (ijk(1,:) >= 1) & (ijk(2,:) >= 1) & (ijk(3,:) >= 1) & (ijk(1,:) <= curdims(1)) & (ijk(2,:)
<= curdims(2)) & (ijk(3,:) <= curdims(3)); % vector of 1's and 0's (ie TRUES and FALSES)

```

```

indicating for each ray if it is in or out of the grid
    indx= sub2ind(curdims,ijk(1,in),ijk(2,in),ijk(3,in)); % list of indices
into mask that are not outside dimensions of mask - the length of this vector is equal to the
number of 1's in "in"
    in_in= (mask(indx) == 1); % of these, which are
also in the object defined by mask
    in(in)= in_in; % now 1's indicate
for each ray that it is both in the grid and within the object defined by the mask
    if any(in)
        stopflg= 0; % don't signal
stop until all rays are out of object
        box_corner= floor(locations);
        [p,t]= ray_box_intersect(locations(:,in),head(:,in),box_corner(:,in)); %
determine intersection with sides of bounding box
        locations(:,in)= p; % this is the
new location
        locations(:,in)= locations(:,in) + cat(1,d_r_x(:,in),d_r_y(:,in),d_r_z(:,in)) * 1.0e-10;
% keep going just a little further to avoid being right on the face
    else
        stopflg= 1; % all out
    end
end
end

```

[Published with MATLAB® R2017a](#)

```

function pathlength_analysis(CTdirectory,PTdirectory)

pix_size= 1; % coordinate system
going forward will have isotropic voxels of size pix_size cm

ct= MSKread3DDicom([CTdirectory, '\\*']); % get CT data
ct_xsize= ct.dicomHeader.PixelSpacing(1) / 10; % get voxel sizes in cm
ct_ysize= ct.dicomHeader.PixelSpacing(2) / 10; % ...
ct_zsize= ct.dicomHeader.SpacingBetweenSlices / 10; % ...

new_xdim= double(round(ct.dicomHeader.Width*ct_xsize/pix_size)); % determine new dims to get
isotropic voxels
new_ydim= double(round(ct.dicomHeader.Height*ct_ysize/pix_size)); % ...
new_zdim= double(round(size(ct.data,3)*ct_zsize/pix_size)); % ...
[xi,yi,zi]=
meshgrid((1:new_xdim)*pix_size/ct_xsize,(1:new_ydim)*pix_size/ct_ysize,(1:new_zdim)*pix_size/ct_z
size); % resample mesh
tissue_distribution=interp3(ct.data,xi,yi,zi); %
tissue_distribution is isotropic
tissue_distribution= (tissue_distribution>-900 & tissue_distribution<-200) + ...
(tissue_distribution>=-200 & tissue_distribution<300)*2 + ...
(tissue_distribution>=300)*3; % segment
into air, lung, water and bone (0,1,2,3) mua values are (0,0.026,0.095,0.12)

pt= MSKread3DDicom([PTdirectory, '\\*']); % get PT data

```



```

pt_xsize= pt.di comHdr. Pixel Spacing(1) / 10; % get voxel sizes in cm
pt_ysize= pt.di comHdr. Pixel Spacing(2) / 10; % ...
pt_zsize= pt.di comHdr. SliceThickness / 10; % ...
[xi, yi, zi]=
meshgrid((1:new_xdim)*pi_xsize/pt_xsize, (1:new_ydim)*pi_ysize/pt_ysize, (1:new_zdim)*pi_xsize/pt_zsize); % resample mesh
source_distribution=interp3(pt.data, xi, yi, zi); %
source_distribution sampled same as tissue distribution

figure; orthosc(tissue_distribution)
figure; orthosc(source_distribution)

xcenter= pi_xsize*new_xdim/2; % coordinate of image
center
ycenter= pi_xsize*new_ydim/2; % ...
axial_extent= 15.7; % PT cylinder axial
extent in cm - be nice if this was in header
R= 88 / 2; % PT cylinder
radius in cm - be nice if this was in header
ptcyl = [xcenter, ycenter, 0, xcenter, ycenter, 47*pi_xsize, R]; % define PT cylinder
(points at either end of axis followed by radius

if pt.di comHdr. Units ~= 'BQML' % be sure of units
    error('error - data not stored in Bq/mL\n');
end
% set fudge factors based upon scanner type
if pt.di comHdr. ManufacturersModel Name == 'Discovery 690'
    detector_efficiency_ff= 0.6061;
    timing_window_ff= 0.9544;
elseif pt.di comHdr. ManufacturersModel Name == 'Discovery 600'
    detector_efficiency_ff= 0.7711;
    timing_window_ff= 1.1171;
elseif pt.di comHdr. ManufacturersModel Name == 'Discovery STE'
    fprintf(1, warning - 'fudge factors for DSTE not yet established\n');
    detector_efficiency_ff= 1;
    timing_window_ff= 1;
else
    fprintf(1, 'Unknown scanner\n');
    detector_efficiency_ff= 1;
    timing_window_ff= 1;
end

total_coinc= 0;
for i=1:size(pt.data, 3)
    total_activity_Bq= overall_sum(pt.data(:, :, i)) * pt_xsize * pt_ysize * pt_zsize; % sum
times voxel volume to get total activity for this slice
    total_activity_Bq= total_activity_Bq / (pt.di comHdrVector(i).DecayFactor *
pt.di comHdrVector(i).DeadTimeFactor); % adjust for dead time and decay
    total_coinc= total_coinc + total_activity_Bq * double(pt.di comHdrVector(i).acq_duration) *
pt.di comHdr. position_fraction; % accumulate expected total number of coincident events within
the FOV
end

```

```

nEventsSimulated= 1e5;
[locations, directions, starts, lengths] =
pet_photon_sim(tissue_distribution, nEventsSimulated, source_distribution);
nLines= size(locations, 2); % total number
of annihilation pairs simulated
locations= locations * pi_x_size; % convert to cm
starts= starts * pi_x_size; % ...
lengths= lengths * pi_x_size; % ...

[h, x]=hist(lengths(2, : , 1)+lengths(2, : , 2), 100); figure; bar(x, h);
[h, x]=hist(lengths(3, : , 1)+lengths(3, : , 2), 100); figure; bar(x, h);
[h, x]=hist(lengths(4, : , 1)+lengths(4, : , 2), 100); figure; bar(x, h);

nSlices= double(pt. di comHdr. NumberOfSlices); % get number of
slices
nBeds= (nSlices-5) / 42; % assume 47 slices
per bed with 5 slice overlap
begSlices= ((1:nBeds)-1) * 42 + 1; % locations of 1st
slice of each bed position in "slice" units
bed_positions= begSlices * pt_zsize; % convert to cm

[hiflags, hitlocs] =
pet_geometry_filter(locations, [xcenter, ycenter], [axial_extent, R], bed_positions, [atan2(axial_extent, 2*R), 0]);

mualengths= lengths;
mualengths(1, : , :)= 0;
mualengths(2, : , :)= mualengths(2, : , :)* 0.095; % 0.026;
mualengths(3, : , :)= mualengths(3, : , :)* 0.095;
mualengths(4, : , :)= mualengths(4, : , :)* 0.12;
mualengths= squeeze(sum(mualengths, 1));

figure;
for i=1:nBeds
    j=round((i-1)*13.8462)+1;
    s(i)=overall_sum(source_distribution(:, : , j:(j+13)));
    t(i)=overall_sum(tissue_distribution(:, : , j:(j+13)));

    % determine measured trues - note: definition of prompts is different depending up the randoms
    % correction method applied
    midSlice= (i-1) * 42 + 1; % location
    of middle slice of current bed position in "slice" units
    fprintf(1, 'midSlice %d
location=%f\n', pt. di comHdrVector(midSlice). SeriesNumber, pt. di comHdrVector(midSlice). SliceLocation);
    randomness= pt. di comHdrVector(midSlice). total_delays; %
    measured randomness
    if pt. di comHdr. RandomsCorrectionMethod == 'SING'
        trues_meas= pt. di comHdrVector(midSlice). total_prompts * (1 -
pt. di comHdrVector(midSlice). ScatterFractionFactor) - randomness; % -
pt. di comHdrVector(midSlice). total_delays;
    % trues_meas= pt. di comHdrVector(midSlice). total_prompts; % -
pt. di comHdrVector(midSlice). total_delays;

```

```

else
    fprintf(1,'warning - untested conditions using other than singles-based randoms
correcti on\n');
    trues_meas= pt.di comHdrVector(mi dSl i ce). total _prompts * (1 -
pt. di comHdrVector(mi dSl i ce). ScatterFracti onFactor) - randoms_meas;
end
mtv(i)=trues_meas;
mrv(i)=randoms_meas;

coi nc_hi tfl g= (sum(hi tfl gs(:, :, i), 1) == 2); % separate
coi nc i dences from si ngl es
si ngl _hi tfl g= (sum(hi tfl gs(:, :, i), 1) == 1); % ...

a(i)= sum(sum(mual engths(coi nc_hi tfl g, :), 2));
b(i)= sum(sum(squeeze(l engths(3, coi nc_hi tfl g, :), 2));
p_coi nc= sum(exp(-sum(mual engths(coi nc_hi tfl g, :), 2))) / nLi nes; %
cal cul ate the probabi li ty of a coi nc i d e n t pai r
p_si ngl = sum(sum(exp(-mual engths(si ngl _hi tfl g, :), 2))) / nLi nes; %
cal cul ate the probabi li ty of a si ngl e
trues_esti mate= total _coi nc * p_coi nc * detector_effi ci ency_ff^2; % esti mate trues
randoms_esti mate= ((total _coi nc * p_si ngl * detector_effi ci ency_ff) /
doubl e(pt. di comHdrVector(mi dSl i ce). acq_durati on)) * ti mi ng_wi ndow_ff;
etv(i)=trues_esti mate;
erv(i)=randoms_esti mate;
yyy(i)= p_coi nc * nLi nes;

% fprintf(1,'\n\nBed position %d \n', i);
% fprintf(1,'estimated trues =\t%f\nmeasured trues =\t%f\noff by factor of
%f\n', trues_esti mate, trues_meas, trues_esti mate/trues_meas);
% fprintf(1,'estimated rndms =\t%f\nmeasured rndms =\t%f\noff by factor of
%f\n', randoms_esti mate, randoms_meas, randoms_esti mate/randoms_meas);
hold on; pl ot3(starts(1, coi nc_hi tfl g), starts(2, coi nc_hi tfl g), starts(3, coi nc_hi tfl g), 'x')

pl ot3(hi t l ocs(1, coi nc_hi tfl g, 2, i), hi t l ocs(2, coi nc_hi tfl g, 2, i), hi t l ocs(3, coi nc_hi tfl g, 2, i), 'rx')

pl ot3(hi t l ocs(1, coi nc_hi tfl g, 1, i), hi t l ocs(2, coi nc_hi tfl g, 1, i), hi t l ocs(3, coi nc_hi tfl g, 1, i), 'gx')
axis equal
end
figure; pl ot(mtv); hold on; pl ot(etv, 'r'); pl ot(s, 'g'); pl ot(yyy, 'm');
figure; pl ot(t);
figure; pl ot(a);
figure; pl ot(b);
keyboard

```

[Published with MATLAB® R2017a](#)

```

function P= perspective_transform_bui ld(pos, target, up, va)
v= target - pos; % v is vector describing the direction that the
camera is pointing
v= v / sqrt(sum(v.^2)); % normalize to unit length

```

```

up= up - dot(up,v) * v; % up is the camera's up direction, the real up is
perpendicular to v
up= up / sqrt(sum(up.^2)); % normalize up vector to unit length
r= cross(v, up); % r points to the camera's right
R= [[0;0;1],[0;1;0],[1;0;0]] \ [v',up',r']; % R will cause v to point to +Z, up to point to +Y
and r to point to +X
f= cot((va/2)*(pi/180)); % f is the focal length
K= diag([f,f,1]); % K is the camera internal calibration matrix
P= K * R * [eye(3),-pos']; % P is the camera matrix

```

[Published with MATLAB® R2017a](#)

```

% PERSPECTIVE_TRANSFORM_DECOMPOSE Extract K, R from camera matrix P.
%
% [K,R,Ctilde] = PERSPECTIVE_TRANSFORM_DECOMPOSE(P,[noscale]) finds K, R, t such that P =
K*R*[eye(3) -Ctilde].
% It is det(R)==1.
% K is scaled so that K(3,3)==1 and all diagonal elements of K are >0.
%
% Works also generally for any P of size N-by-(N+1).
% Works also for P of size N-by-N, then t is not computed.

% Brad Beattie

function [K, R, Ctilde] = perspective_transform_decompose(P,noscale)
N = size(P,1);
H = P(:,1:N);
[K,R] = vgg_rq(H);
Ctilde = -P(:,1:N)\P(:,end);
if nargin > 1
    for i=1:3
        R(i,:) = R(i,:) * sign(K(i,i));
        K(:,i) = K(:,i) * sign(K(i,i));
    end
    K= K / K(3,3);
end

```

[Published with MATLAB® R2017a](#)

```

%PHOTON_INTERACTION - given photon energy, material and path-length, calculates probability of
depositing specified energies
% [pedei,cdei,ppdei]= photon_interaction(hv,material,pathlength,at_energies)
% hv - photon energy in MeV
% material - type of material photon is interacting with (must be supported by mu_table)
% pathlength - mean path-length through material
% at_energies - vector of energies at which probabilities/intensities are calculated

```

```

% pedei - photoelectric deposited energy intensity
% cdei - Compton deposited energy intensity
% ppdei - pair production deposited energy intensity

function [pedei, cdei, ppdei, nu] = photon_interaction(hv, material, pathlength, at_energies)
% allocate result vectors
pedei = zeros(size(at_energies)); % photoelectric deposited energy intensity
cdei = zeros(size(at_energies)); % Compton deposited energy intensity
ppdei = zeros(size(at_energies)); % pair-production deposited energy intensity
if hv <= 0, return; end
% get mu's for incident photon energy
[mu_compton, mu_photoelectric, mu_pairproduction, mu_total] = mu_table(material, hv);
% apply mu to length to get probability of interaction
p_total = 1 - exp(-mu_total*pathlength);
p_compton = p_total .* mu_compton ./ mu_total;
p_photoelectric = p_total .* mu_photoelectric ./ mu_total;
p_pairproduction = p_total .* mu_pairproduction ./ mu_total;
% photoelectric
[de, i] = min(abs(hv-at_energies)); % find energy in at_energies closest to hv
if de > 0.1, warning('Photoelectric effect deposits energy far from requested range'); end
pedei(i) = p_photoelectric; % assign entire photoelectric fraction to that energy
% Compton
[T, dedT, nu] = compton_energy_spectrum(hv); % calculate energy spectrum for Compton electrons
cdei = interp1(T, dedT, at_energies); % interpolate to at_energies
cdei(isnan(cdei)) = 0; % zero values outside range of Compton energy table
sum_cdei = sum(cdei);
if sum_cdei > 0, cdei = p_compton * cdei / sum_cdei; end % force sum to Compton probability
if nargin == 4
    nu = interp1(T, nu, at_energies);
    nu(isnan(nu)) = 0;
end
% pair production
if hv > 1.022 % only occurs with energies greater than 2 * 0.511 MeV
    [de, i] = min(abs((hv-1.022)-at_energies)); % find energy in at_energies closest to hv-1.022
    if de > 0.1, warning('pair-production effect deposits energy far from requested range'); end
    ppdei(i) = p_pairproduction; % assign entire pair-production fraction to that energy
end

```

[Published with MATLAB® R2017a](#)

```

function [img, zbuf] = project3Dto2D(points, intensities, xDim, yDim, P)
% project3Dto2D
img = zeros(xDim, yDim);
zbuf = repmat(inf, xDim, yDim);

tmp = points(:, 3);
points(:, 3) = points(:, 1); % X becomes Z (i.e. CT Y becomes Z)
points(:, 1) = 191 - points(:, 2); % Y becomes flipped X (i.e. CT X becomes flipped X)
points(:, 2) = tmp; % Z becomes Y (i.e. CT Z becomes Y)

```

```

points= P * ([points, ones(size(points, 1), 1)]');
points(1:2,:) = round(points(1:2,:) ./ repmat(points(3,:), 2, 1));

inlmglndices= find(points(1,:) >= 1 & points(1,:) <= xDim & points(2,:) >= 1 & points(2,:) <= yDim);

nPoi nts= numel (i nlmglndices);
for i =1:nPoi nts
    j = i nlmglndices(i);
    x= poi nts(1,j);
    y= poi nts(2,j);
    if poi nts(3,j) < zbuf(x,y) % if z is closer than current closest
        zbuf(x,y)= poi nts(3,j); % update closest z
        img(x,y)= intensi ties(j); % assign pixel intensity based on given intensity
    end
end
end

```

[Published with MATLAB® R2017a](#)

```

% del ta_rayEnergy - vector of random del ta ray energies in MeV

function del ta_rayEnergy= rand_del ta_ray(betaEnergy, betaThreshol d)
% betaEnergy - energy of the beta particle in MeV
c = 299792458; % speed of light in a
vacuum in meters per second
me= 9.1093821545e-31; % electron rest mass
in kg
re= 2.817940289458e-15; % electron radius in
meters
NO= 6.0221417930e23; % Avogadro's number

MeVperJoul e= 6.241506363e+12;

%MetersPerPl anckLength= 1.61625281e-35;
%KgPerPl anckMass= 2.1764411e-8;
%SecondsPerPl anckTime= 5.3912427e-44;

vel oci ty= beta_vel oci ty(betaEnergy); % the beta particle
vel oci ty in meters per second
beta= vel oci ty / c; % electron velocity
fraction of the speed of light

re= re * 100; % convert from
meters to cm
me= me * c^2; % convert from kg to
Joules
me= me * MeVperJoul e; % convert from Joules
to MeV

% select constants B and C such that CDF starts at 0 and ends at 1

```

```

C= 2*pi*re^2*me*NO ./ (beta.^2.*betaThreshold);
B= 1 ./ (C - (2*pi*re^2*me*NO ./ (beta.^2.*betaEnergy)));

cdf= rand(size(betaEnergy)); % random number
between 0 and 1
delta_rayEnergy= 2*pi*re^2*me*NO ./ (beta.^2.*(C-cdf./B)); % invert
delta_ray_cdf

```

[Published with MATLAB® R2017a](#)

```

% rand_moliere - generates random numbers according to distribution originally described by
% Moliere and as calculated by Levin
function theta= rand_moliere(thickness, betaEnergy, A, Z);
% thickness - path length(s) in cm traveled by one or more betas
% betaEnergy - energy(ies) in MeV of one or more betas

% uses "rejection method" described by Levin and explained in INTRODUCTION TO MONTE CARLO METHODS
% by D.J.C. MACKAY of Department of Physics, Cambridge University.

Nt= numel(thickness); % number of path
lengths
Ne= numel(betaEnergy); % number of beta
energies

% either or both thickness and betaEnergy can be vectors, if both vectors they must be of same
% size
% if only one is a vector, then singular value of other applies to all
if Nt ~= Ne & Nt ~= 1 & Ne ~= 1
    error('thickness and betaEnergy must be of the same size or equal to one');
end
N=max([Nt, Ne]); % the length of the
thickness/betaEnergy vector(s) determines the number of angles to return
theta= repval(NaN, N); % allocate and
initialize thetas

i_left= (1:N)'; % start by
needing all
while N > 0
    pdf_max= moliere_pdf(0, thickness(i_left), betaEnergy(i_left), A, Z); % the max of Moliere's
    PDF is always at angle 0
    pdf_min= moliere_pdf(pi, thickness(i_left), betaEnergy(i_left), A, Z); % and the min is at
    the maximum deflection angle of +/- pi
    sigmasq= -pi^2 ./ (2*log(pdf_min./pdf_max)); % calculates variance
    of bounding Gaussian reference function

    x= randn(N, 1) .* sqrt(sigmasq); % randomly sample
    from Gaussian
    i= find(x > -pi & x <= pi); % only consider
    those between +/- pi
    g_ref= pdf_max(i) .* exp(-x(i).^2 ./ (2*sigmasq(i))); % determine

```

height of Gaussian at those sample points

```
pdf_ref= moliere_pdf(x(i), thickness(i_left(i)), betaEnergy(i_left(i)), A, Z); % determine
height of Moliere PDF at those same sample points
y= rand(size(pdf_ref)) .* g_ref; % for each
sample point, randomly sample a number from a uniform distribution maxing at g_ref
j= find(y<=pdf_ref); % accept value
for theta if random height value less than Moliere's function
    theta(i_left(i(j)))= x(i(j)); % assign to
return vector
i_left= find(isnan(theta)); % see who's
left unassigned
N= numel(i_left);
end
```

Published with MATLAB® R2017a

```
function histo = singles_path_length_histogram(mask3d, density, nbins)
```

```
if nargin < 2, density= 100; end
if nargin < 3, nbins= 100; end

histo= zeros(nbins,1); % allocate space for result
dims= size(mask3d); % mask dimensions
maxlength= sqrt(sum(dims.^2)); % maximum possible pathlength through object is between
diagonal corners
l2dims= ceil(log2(dims)); % log2 of dimensions rounded up
newdims= 2.^l2dims; % next largest power of two for each dimension
nlevels= min(l2dims); % number of subsamplings before one of the
dimensions goes to 1

% calculate series of subsampled masks
m{1}= zeros(newdims);
m{1}(1:dims(1), 1:dims(2), 1:dims(3))= mask3d;
mask3d= m{1};
for llevel=2:nlevels
    m{llevel}= zoomout3(m{llevel-1}); % averages groups of 2x2x2 voxels
    m{llevel}(m{llevel}~=1)= 0; % zero out all non-ones
end

% at each level, remove voxels corresponding to regions that were entirely within the object
at the lower levels
tmp1= zoomin3(m{nlevels});
for llevel=(nlevels-1):-1:1 % working from the bottom up
    tmp2= m{llevel}; % save current level
    m{llevel}(tmp1==1)= 0; % zero out voxels corresponding to ones in the lower level
    tmp1= zoomin3(tmp2); % use blow up of copy in next round
end
% at this point, each level contains only the 1's that are "new" to that level
```



```

% initialize photon starting points, final locations and directions
locations= [];
starts= [];
directions= [];

% work from the bottom level up
for level=nllevels:-1:1
    curdims= size(m{level});           % get dimensions of the current level
    indices= find(m{level}==1);        % find the 1's
    nindices= numel(indices);          % count them
    if nindices > 0
        [x, y, z]= ind2sub(curdims, indices);           % convert
to coordinate indices
        [new_locations, new_directions]= init_rays(x, y, z, density, level); % fill
each with photons
        locations= [locations, new_locations];          % add to
list of current locations
        starts= [starts, new_locations];                % add to
list of starting locations
        directions= [directions, new_directions];       % keep
full list of directions
        locations= update_rays(locations, directions, m{level});
    end
    if level > 1
        locations= locations * 2;
        starts= starts * 2;
    end
end
stopflag= 0;
while ~stopflag
    [locations, stopflag]= update_rays(locations, directions, mask3d);
end

% need to keep updating till all rays leave the mask -- also need to terminate based on mask
only at the final level - instead keep rays on hold (ie don't extend but don't terminate)

d= sqrt(sum( (locations-starts).^2 ));
[h, x]= hist(d, nbins);
histo= [x, h];
keyboard
end

function [locations, directions] = init_rays(x, y, z, density, level)
    n= density * 8^(level-1) * numel(x);           % the total number of new rays is the density
times the voxel volume times the number of voxels
    x= rowvectorize(x);                             % make sure all are row vectors
    y= rowvectorize(y);                             % ...
    z= rowvectorize(z);                             % ...
    locations= repmat([x; y; z], 1, n/numel(x)) - rand(3, n); % random starting points within boxes
    dir_theta= acos(1-2*rand(1, n));                  % choose random isotropic initial directions
specified in polar coordinates with theta distributed uniformly over cos(0:2pi)
    dir_phi= 2 * pi * rand(1, n);                    % and phi uniformly over (0:2pi) --- see Rajon
    directions= cat(1, dir_theta, dir_phi);           % concatenate theta and phi to make a two row

```

```

direction matrix
end

function [locations, stopflag]= update_rays(locations, directions, mask)
    curdims= size(mask); % get current dimensions
    sin_theta= sin(directions(1,:)); % avoid calculating
    twice (see next two lines)
    dir_x= sin_theta .* cos(directions(2,:)); % calculate current
    direction as Cartesian vector
    dir_y= sin_theta .* sin(directions(2,:)); % ...
    dir_z= cos(directions(1,:)); % ...
    nrm= sqrt(dir_x.^2 + dir_y.^2 + dir_z.^2);
    dir_x= dir_x ./ nrm; dir_y= dir_y ./ nrm; dir_z= dir_z ./ nrm;
    head= locations + cat(1,dir_x,dir_y,dir_z); % specify head of ray
    as being unit distance away along line in specified direction
    ijk= ceil(locations); % determine indices
    of new bounding box within mask for each ray
    in= (ijk(1,:) >= 1) & (ijk(2,:) >= 1) & (ijk(3,:) >= 1) & (ijk(1,:) <= curdims(1)) & (ijk(2,:)
    <= curdims(2)) & (ijk(3,:) <= curdims(3)); % vector of 1's and 0's (ie TRUES and FALSES)
    indicating for each ray if it is in or out of the grid
    indx= sub2ind(curdims,ijk(1,in),ijk(2,in),ijk(3,in)); % list of indices
    into mask that are not outside dimensions of mask - the length of this vector is equal to the
    number of 1's in "in"
    in_in= (mask(indx) == 1); % of these, which are
    also in the object defined by mask
    in(in)= in_in; % now 1's indicate
    for each ray that it is both in the grid and within the object defined by the mask
    if any(in)
        stopflag= 0; % don't signal
    stop until all rays are out of object
    box_corner= floor(locations);
    [p,t]= ray_box_intersect(locations(:,in),head(:,in),box_corner(:,in)); %
    determine intersection with sides of bounding box
    locations(:,in)= p; % this is the
    new location
    locations(:,in)= locations(:,in) + cat(1,dir_x(:,in),dir_y(:,in),dir_z(:,in)) * 1.0e-10;
    % keep going just a little further to avoid being right on the face
    else
        stopflag= 1; % all out
    end
end
end

```

[Published with MATLAB® R2017a](#)

```

function factor = surface_beta_loss_factor(x, psf)
cs_psf= cumtrapz(x, psf);
cs_psf= cs_psf * 0.5 / cs_psf(end);
factor= trapz(x, 0.5-cs_psf);

```

CODE ASSOCIATED WITH AIM 3

```
function [ M ] = applyCombi nes( combi nel ndi ces, M )
    [s1, s2, s3]=size(M);
    for k=1: numel (combi nel ndi ces)
        j=combi nel ndi ces(k);
        M(:, j, :)=M(:, j, :)+M(:, j+1, :);
        i=(1:s2)~=(j+1);
        M= M(:, i, :);
        s2=s2-1;
    end
end
```

```
function W = calcul ateModel ( interi orPoi nts, voxel Wi dth, detectorPoi nts, detectorWi dth, mu )
    [nVoxel s, three]=size(i nteri orPoi nts);
    if three ~= 3, error('interi orPoi nts must be nVoxel s by 3 in size'); end
    [nDetector s, three]=size(detectorPoi nts);
    if three ~= 3, error('detectorPoi nts must be nDetector s by 3 in size'); end
    nWavel engths= numel (mu);
    W= zeros(nDetector s, nWavel engths, nVoxel s);
    % create attenuation lookup table
    r=(1:500)*100/500; % range of source to surface distances is 1/5 to 100 mm
    max_pl d=-log(eps)/mi n(mu); % maximum pathlength distance that needs to be considered
    % is determined by the minimum attenuation
    pl d=(1:500)*max_pl d/500; % range of pathlengths
    dpl d= pl d(1); % step size
    Lut= zeros(500, nWavel engths); % lookup table to contain 500 source distances by
    nWavel engths
    for i=1: nWavel engths
        LMD= log(0.7339 * r.^1.907 + 1.66); % mu parameters of the lognormal
        % distribution describing the pathlengths
        LSD= 0.1973 * (1-exp(-0.5558*r)) + 0.3146 * exp(-1.403*r); % sigma parameter of the
        % lognormal distribution describing the pathlengths
        % fill in lookup table for each source distance and wavelength integrating over the lognormal
        % pathlength distribution
        for j=1: numel (r)
            Lut(j, i)= sum(exp(-mu(i)*pl d) .* lognpdf(pl d, LMD(j), LSD(j)) * dpl d);
        end
    end
    for i=1: nDetector s
        dx2= (detectorPoi nts(i, 1) - interi orPoi nts(:, 1)).^2; % squared distance in x of
        % detectorPoint from all interi orPoi nts
```

```

dy2= (detectorPoints(i,2) - interiorPoints(:,2)).^2; % same for y
dz2= (detectorPoints(i,3) - interiorPoints(:,3)).^2; % same for z
d2= (dx2+dy2+dz2) * voxelWidth.^2; % squared distance in mm^2 between
detectorPoint and all interiorPoints
d= sqrt(d2); % distance in mm
for j=1:nWavelengths
    % distribute probability over surface of a sphere, factor in detector sizes
    W(i,j,:)= detectorWidth^2 * (1./(4*pi*d2))' .* interp1(r,lut(:,j),d)';
    % multiplying by W should convert photons/second/voxel to photons/second/detector
end
end

```

[Published with MATLAB® R2017a](#)

```

function [ optimalTimes, estNoise, pi nvW ] = determineOptimalTimes( W3D, X, T, D, R )

[nDetectors, nWavelengths, nVoxels] = size(W3D);
W=reshape(W3D, nDetectors*nWavelengths, nVoxels); % reshape into 2D weight matrix
pi nvW=pi nv(W); % calculate pseudo inverse
pi nvW3D=reshape(pi nvW', nDetectors, nWavelengths, nVoxels); % make it look like W3D

for j=1:nWavelengths
    Wj=squeeze(W3D(:,j,:)); % extract W for wavelength j
    pWj=squeeze(pi nvW3D(:,j,:))'; % extract and invert pseudo inverse W for wavelength j
    pWj2=pWj.^2; % uncertainties sum in quadrature so weights are squared
    Y=Wj*X; % estimated measurements for specified source
distribution
    % calculate parameters for equations (see OptimalTimeCalculation_v9.doc)
    Q(j)=mean(pWj2*Y);
    QD(j)=mean(sum(pWj2*D,2));
    QR(j)=mean(sum(pWj2*R,2));
end
f=sqrt(Q+QD); f=f/sum(f); % initial estimate of f
for i=1:20 % iterate 20 times (very
likely convergent)
    f=sqrt(Q+QD+(QR./f)); f=f/sum(f); % update estimate of f
end
optimalTimes= f * T; % convert fractions to
actual times
estNoise=sqrt(sum( (Q+QD+(QR./optimalTimes)) ./ optimalTimes )); % estimated noise
end

```

[Published with MATLAB® R2017a](#)

```

function [ optimalTimes, optimalWavelengths, optimalBandwidths, newW3D, combinedIndices,
finalUncertainty, initialUncertainty ] = determineOptimalTimesAndWavelengths( curW3D, X, T, D, R,

```

```

curWavelengths, curBandwidths )

[nDetectors, nWavelengths, nVoxels]=size(curW3D);
[optimalTimes, initialUncertainty]=determineOptimalTimes(curW3D, X, T, D, R); % start by determining
optimal time distribution when using all wavelengths
%initialUncertainty=initialUncertainty/sqrt(nWavelengths); %TESTING
finalUncertainty=initialUncertainty % initialize final (i.e.
minimum) uncertainty
optimalWavelengths=curWavelengths; % initialize final
optimalwavelengths
optimalBandwidths=curBandwidths; % initialize bandwidths
combinedIndices=[]; % initialize record or
combinations
while nWavelengths > 2 % always have at least
two wavelengths
    newUncertainty=zeros(1, nWavelengths-1); % allocate space for
uncertainty info
    for j=1: (nWavelengths-1) % try combining each
adjacent pair of wavelengths in turn
        newW3D= curW3D; % using a copy of the
matrix ...
        newW3D(:,j,:)= newW3D(:,j,:) + newW3D(:,j+1,:); % combine (sum) matrix
for one pair of wavelengths
        i=(1: nWavelengths)~=(j+1); % identify element to
be removed
        newW3D= newW3D(:,i,:); % remove from matrix
        [tmp1,tmp2]= determineOptimalTimes( newW3D, X, T, D, R ); % calc uncertainty for
new system matrix
        % tmp2=tmp2/sqrt(nWavelengths-1); %TESTING
        newIntegrationTimes{j}=tmp1;
        newUncertainty(j)=tmp2
    end
    [minUncertainty,j]= min(newUncertainty); % select reduced
wavelength that most reduced uncertainty
    if minUncertainty < finalUncertainty % keep it only if
better than best so far
        combinedIndices=[combinedIndices, j]; % update
record of combinations
        finalUncertainty=minUncertainty % update
uncertainty
        optimalTimes=newIntegrationTimes{j}; % udate
integration times
        curW3D(:,j,:)= curW3D(:,j,:) + curW3D(:,j+1,:); % combine as
before but with current
        i=(1: nWavelengths)~=(j+1); % identify
element to be removed
        curW3D= curW3D(:,i,:); % remove extra
wavelength from matrix

optimalWavelengths(j)=(optimalWavelengths(j)*optimalBandwidths(j)+optimalWavelengths(j+1)*optimal
Bandwidths(j+1))/(optimalBandwidths(j)+optimalBandwidths(j+1)); % combine wavelengths (new is
weighted average of combined)
    optimalBandwidths(j)=optimalBandwidths(j)+optimalBandwidths(j+1); % sum two

```

```

bandwidths together
    optimalWavelengths=optimalWavelengths(i); % remove extra
wavelength
    optimalBandwidths=optimalBandwidths(i); % remove extra
bandwidth
    nWavelengths=nWavelengths-1; % reflect
reduction in number of wavelengths
    else
        break; % if nothing better -
stop
    end
end

```

[Published with MATLAB® R2017a](#)

```

function [ optimalTimes, optimalWavelengths, optimalBandwidths, newW3D, combinations,
finalUncertainty, initialUncertainty ] = determineOptimalTimesAndWavelengthsForEM( curW3D, X, T,
D, R, curWavelengths, curBandwidths )

[nDetectors, nWavelengths, nVoxels]=size(curW3D);
[optimalTimes, initialUncertainty]=determineOptimalTimesForEM(curW3D, X, T, D, R); % start by
determining optimal time distribution when using all wavelengths
%initialUncertainty=initialUncertainty/sqrt(nWavelengths); %TESTING
finalUncertainty=initialUncertainty % initialize final
(i.e. minimum) uncertainty
optimalWavelengths=curWavelengths; % initialize final
optimalBandwidths=curBandwidths; % initialize bandwidths
combinations=[]; % initialize record or
combinations
while nWavelengths > 2 % always have at least
two wavelengths
    newUncertainty=zeros(1, nWavelengths-1); % allocate space for
uncertainty info
    for j=1: (nWavelengths-1) % try combining each
adjacent pair of wavelengths in turn
        newW3D= curW3D; % using a copy of the
matrix ...
        newW3D(:,j,:)= newW3D(:,j,:) + newW3D(:,j+1,:); % combine (sum) matrix
for one pair of wavelengths
        i=(1:nWavelengths)~=(j+1); % identify element to
be removed
        newW3D= newW3D(:,i,:); % remove from matrix
        [tmp1, tmp2]= determineOptimalTimesForEM( newW3D, X, T, D, R ); % calc uncertainty
for new system matrix
        % tmp2=tmp2/sqrt(nWavelengths-1); %TESTING
        newIntegrationTimes{j}=tmp1;
        newUncertainty(j)=tmp2
    end
    [minUncertainty, j] = min(newUncertainty); % select reduced

```

```

wavelength that most reduced uncertainty
    if minUncertainty < finalUncertainty % keep it only if
better than best so far
        combinelndices=[combinelndices,j]; % update
record of combinations
        finalUncertainty=minUncertainty % update
uncertainty
        optimalTimes=newIntegrationTimes{j}; % update
integration times
        curW3D(:,j,:)= curW3D(:,j,:) + curW3D(:,j+1,:); % combine as
before but with current
        i=(1:nWavelengths)~(j+1); % identify
element to be removed
        curW3D= curW3D(:,i,:); % remove extra
wavelength from matrix

optimalWavelengths(j)=(optimalWavelengths(j)*optimalBandwidths(j)+optimalWavelengths(j+1)*optimal
Bandwidths(j+1))/(optimalBandwidths(j)+optimalBandwidths(j+1)); % combine wavelengths (new is
weighted average of combined)
        optimalBandwidths(j)=optimalBandwidths(j)+optimalBandwidths(j+1); % sum two
bandwidths together
        optimalWavelengths=optimalWavelengths(i); % remove extra
wavelength
        optimalBandwidths=optimalBandwidths(i); % remove extra
bandwidth
        nWavelengths=nWavelengths-1; % reflect
reduction in number of wavelengths
    else
        break; % if nothing better -
stop
    end
end
end

```

[Published with MATLAB® R2017a](#)

```

function [ optimalTimes, estNoise, U ] = determineOptimalTimesForEM( W3D, X, T, D, R )

[nDetectors, nWavelengths, nVoxels]= size(W3D);
W=reshape(W3D, nDetectors*nWavelengths, nVoxels); % reshape into 2D weight matrix
U=EMerrorMatrix1(W, X, 160); % calculate Barrett's inverse
U3D=reshape(U', nDetectors, nWavelengths, nVoxels); % make it look like W3D

for j=1: nWavelengths
    Wj=squeeze(W3D(:,j,:)); % extract W for wavelength j
    Uj=squeeze(U3D(:,j,:))'; % extract and invert Barrett's error inverse for
wavelength j
    Uj2=Uj.^2; % uncertainties sum in quadrature so weights are squared
    Y=Wj*X; % estimated measurements for specified source
distribution
    % calculate parameters for equations (see OptimalTimeCalculation_v9.doc)

```

```

        Q(j)=mean(Uj 2*Y);
        QD(j)=mean(sum(Uj 2*D, 2));
        QR(j)=mean(sum(Uj 2*R, 2));
    end
    f=sqrt(Q+QD); f=f/sum(f); % initial estimate of f
    for i=1: 20 % iterate 20 times (very
likely convergent)
        f=sqrt(Q+QD+(QR./f)); f=f/sum(f); % update estimate of f
    end
    optimalTimes= f * T; % convert fractions to
actual times
    estNoise=sqrt(sum( (Q+QD+(QR./optimalTimes)) ./ optimalTimes )); % estimated noise
end

```

Published with MATLAB® R2017a

```

function [a,b_prime] = em_w_noise(niter, b, m, n, a)
[r,c]= size(m);
m_prime= bsxfun(@rdivide,m,sum(m,1))';
if nargin < 5
    a= ones(c,1) * (sum(b) / c);
end
for i=1:niter
    b_prime= m * a + n;
    e= b ./ b_prime;
    a= a .* (m_prime * e);
end

```

Published with MATLAB® R2017a

```

function [U,a] = EErrorMatrix1(W,X,niter)
% U= bsxfun(@rdivide,W,sum(W,1))';
[M,N]= size(W);
a=ones(size(X))*mean(X); % a is Nx1
% a=X;
U=zeros(N,M); % U is NxM
s=sum(W)'; % s is Nx1
Y=W*X; % Y is Mx1
WT=W'; % WT is NxM
for i=1:niter
    Wa=(W*a)'; % Wa is 1xM
    B=diag(1./s)*WT*diag(1./Wa); % B is NxM
    A=B*W*diag(a); % A is NxN
    OmA=eye(N,N)-A;
    U=B+(OmA*U);
    e=Y./(Wa');
end

```



```

a=a.*(WT*e);
a=a./s;

end

end

```

[Published with MATLAB® R2017a](#)

```

function [ M ] = expandCombi nes( combi nel ndi ces, M )
for k=numel (combi nel ndi ces): -1: 1
[s1, s2, s3]=si ze(M);
newM=zeros(s1, s2+1, s3);
j =combi nel ndi ces(k);
i =(1: (s2+1))~=(j +1);
newM(:, i , :)= M;
newM(:, j +1, :)=newM(:, j , :);
M=newM;
end
end

```

[Published with MATLAB® R2017a](#)

```

function [ sensi tivi ty, darkCurrent, readNoise ] = I VIS200parameters( FOVsetting, pi xBi n, fstop,
detectorArea )
% parameters desccribing an I VIS 200
FOVdi m_cm. A=3. 9; FOVdi m_cm. B=6. 5; FOVdi m_cm. C=13; % FOV in cm for height
settings A, B, C, D, E
FOVdi m_cm. D=19. 5; FOVdi m_cm. E=26;
CCDdi m_cm=2. 6; % CCD dimension in cm
CCDdi m_pi x= 1920 / pi xBi n; % CCD dimension in pixels
(note: CCD is 2048x2048 but only 1920x1920 used)
darkCurrent=105. 6; % in counts/second/cm^2 (spec
is 100), here the cm^2 is measured on the CCD itself
readNoi se=6. 3848; % in counts/pixel RMS for
binning=1 (spec is 5)
refSensi tivi ty=6. 1e-5; % sensi tivi ty in
(counts/photon) at height C, f1, binning=1
refH=51. 2; % height of focal point in cm
above object when reference sensi tivi ty was determined
refA=6. 35; % radius of aperature in cm at
f1
heightChangeFactor= FOVdi m_cm. (FOVsetting) / FOVdi m_cm. C; % height change is
proportional to FOV change, reference setting is C
H= refH * heightChangeFactor; % new height
A= refA * (1.0 / fstop); % new aperature radius
acceptanceAngl eChangeFactor= atan2(A, H) / atan2(refA, refH); % acceptance angle change
relative to reference

```

```

imagePixelArea= (10 * FOVdim_cm. (FOVsetting) / CCDdim_px)^2; % size of pixel in mm^2 in
imagespace
sizeRatio= imagePixelArea / detectorArea; % pixels per virtual detector
% composite parameters of interest
sensitivity= refSensitivity * pixBin^2 * acceptanceAngleChangeFactor * sizeRatio; % in
(counts/photon)
darkCurrent= darkCurrent * CCDdim_cm^2/CCDdim_px^2 / sizeRatio; % in
counts/second/detector
readNoise= readNoise * sqrt(sizeRatio); %
readNoise goes down with sqrt of the increase in detector size
end

```

[Published with MATLAB® R2017a](#)

```

classdef VoxelClusterList < handle
    properties
        ImageSpace % 3D matrix of zeros and ones where the ones define the interior or
the imaged object
        ImageSpaceDims % size of the original ImageSpace for which this cluster list is
defined
        VoxelIndices % indices into ImageSpace of voxels within object
        MemberList % handle to object array of type VoxelClusterMember
        PPIlength % projection profile length (i.e. the number of rows in W)
    end
    methods

        function n = countValid(obj)
            n= 0;
            m= numel (obj.MemberList);
            for i=1:m
                if obj.MemberList(i).ValidFlag
                    n=n+1;
                end
            end
        end

        function obj = VoxelClusterList(ImageSpace, POIvalues, W)
            % VoxelClusterList Constructor
            % ImageSpace is a 3D matrix of zeros and ones where the ones define the interior or
the imaged object
            % POIvalues is a vector with elements corresponding to the 1's in ImageSpace and
containing the POI (Parameter of Interest)
            if nargin ~= 0
                obj.ImageSpace= ImageSpace;
                obj.ImageSpaceDims= size(ImageSpace);
                obj.VoxelIndices= find(ImageSpace == 1);
                obj.MemberList= VoxelClusterMember(ImageSpace, POIvalues, W);
                obj.PPIlength= size(W, 1);
            end
        end
    end
end

```

```

function [ ValidImg ] = extractAllValid( obj )
% Should regenerate ImageSpace (used for debugging)
ValidImg= zeros(obj.ImageSpaceDims);
m=numel(obj.MemberList);
for i=1:m
    if obj.MemberList(i).ValidFlag
        n= numel(obj.MemberList(i).VoxelList);
        if n ~= obj.MemberList(i).Volume
            error('VoxelList size and Volume mismatch');
        end
        for j=1:n
            [ix,iy,iz]= ind2sub(obj.ImageSpaceDims,obj.MemberList(i).VoxelList(j));
            ValidImg(ix,iy,iz)= ValidImg(ix,iy,iz) + 1;
        end
    end
end

function [ POIimg ] = extractPOIimg( obj )
% Creates 3D image of POI values
POIimg= zeros(obj.ImageSpaceDims);
m=numel(obj.MemberList);
for i=1:m
    if obj.MemberList(i).ValidFlag
        p= obj.MemberList(i).POI;
        v=obj.MemberList(i).Volume;
        for j=1:v
            [ix,iy,iz]= ind2sub(obj.ImageSpaceDims,obj.MemberList(i).VoxelList(j));
            POIimg(ix,iy,iz)= p;
        end
    end
end

% returns POI value for each cluster
function [ POIvalues ] = extractPOIvalues( obj )
% Creates 3D image of POI values
n= obj.countValid();
POIvalues= zeros(n,1);
m=numel(obj.MemberList);
j=1;
for i=1:m
    if obj.MemberList(i).ValidFlag
        POIvalues(j)= obj.MemberList(i).POI;
        j=j+1;
    end
end

function [ valueList ] = imageToValueList( obj , image )
n= obj.countValid();
valueList= zeros(n,1);

```

```

j=1;
m=numel(obj.MemberList);
for i=1:m
    if obj.MemberList(i).ValidFlag
        v=obj.MemberList(i).Volume;
        for k=1:v
            ii=obj.MemberList(i).VoxelList(k);
            valueList(j)=valueList(j)+image(ii);
        end
        valueList(j)=valueList(j)/v;
        j=j+1;
    end
end
if numel(valueList)~= (j-1)
    keyboard;
end

end

function [image]=valueListToImage(obj,valueList)
image=zeros(obj.ImageSpaceDims);
j=1;
m=numel(obj.MemberList);
for i=1:m
    if obj.MemberList(i).ValidFlag
        v=obj.MemberList(i).Volume;
        for k=1:v
            ii=obj.MemberList(i).VoxelList(k);
            image(ii)=valueList(j);
        end
        j=j+1;
    end
end
if numel(valueList)~= (j-1)
    keyboard;
end

end

function [W]=extractW(obj)
% Creates W matrix
m=numel(obj.MemberList);
n=0; for i=1:m, if obj.MemberList(i).ValidFlag==1, n=n+1; end, end
W=zeros(obj.PPIlength,n);
size(W)
j=1;
for i=1:m
    if obj.MemberList(i).ValidFlag
        W(:,j)=obj.MemberList(i).ProjectionProfile;
        j=j+1;
    end
end
if n~= (j-1)
    keyboard;
end

end

```

```

end

function [ P0I values ] = extractCompressedP0I ( obj )
    % Predicts what each cluster's P0I value will be if compressed
    n= obj . countValid();
    P0I values= zeros(n,1);
    m=numel (obj . MemberLi st);
    j =1;
    for i =1: m
        if obj . MemberLi st(i) . Val idFl ag
            p= obj . MemberLi st(i) . P0I ;
            v= obj . MemberLi st(i) . Vol ume;
            p= v * p . ^2;
            n= numel (obj . MemberLi st(i) . Nei ghborLi st);
            for k=1: n
                ii = obj . MemberLi st(i) . Nei ghborLi st(k);
                if obj . MemberLi st(ii) . Val idFl ag
                    tmpP= obj . MemberLi st(ii) . P0I ;
                    tmpV= obj . MemberLi st(ii) . Vol ume;
                    p= p + tmpV * tmpP . ^2;
                    v= v + tmpV;
                end
            end
            p= sqrt(p / v) / v;
            P0I values(j) = p;
            j =j +1;
        end
    end
end

function compress( obj , pThreshol d )
    % Combine clusters having a predicted P0I value above the speci fied threshol d, wi tgh al l
of its nei ghbors
    pass= 1;
    whi le 1
        fprintf(1, ' pass %#d\n' , pass);
        pass= pass + 1;
        P0I values= extractP0I values(obj );
        [mx, i V]= max(P0I values);
        if mx < pThreshol d
            break;
        end
        m=numel (obj . MemberLi st);
        j =0;
        for i =1: m
            if obj . MemberLi st(i) . Val idFl ag
                j =j +1;
            end
            if j ==i V
                i M=i ;
                break;
            end
        end
    end
end

```

```

        merge(obj, iM);
    end
end

function merge( obj, memIndex )
    % merges cluster identified by memIndex with all of its neighboring clusters
    if obj.MemberList(memIndex).ValidFlag == 0
        error(' something fucked up' );
    end
    nl = obj.MemberList(memIndex).NeighborList;
    n = numel(nl); % number of neighboring clusters
    fprintf(1, 'merging ID %d with %d neighbors\n', memIndex, n);
    p = obj.MemberList(memIndex).POI;
    v = obj.MemberList(memIndex).Volume;
    p = v * p^2;
    pp = obj.MemberList(memIndex).ProjectionProfile;
    pp = v * pp;
    for k=1:n
        nk=nl(k); % nk is the cluster ID of the kth neighbor
        tmpP = obj.MemberList(nk).POI;
        tmpV = obj.MemberList(nk).Volume;
        v = v + tmpV;
        p = p + tmpV * tmpP^2;
        tmpPP = obj.MemberList(nk).ProjectionProfile;
        if numel(pp) ~= numel(tmpPP), keyboard; end
        pp = pp + tmpV * tmpPP;
        obj.MemberList(memIndex).VoxelList =
unique([obj.MemberList(memIndex).VoxelList, obj.MemberList(nk).VoxelList]); % add neighbor's
voxels

        % copy over neighbor's neighbors
        i = obj.MemberList(memIndex).NeighborList ~= nk; % flags identifying all
other neighbors
        j = obj.MemberList(nk).NeighborList ~= memIndex; % flags identifying kth
neighbor's neighbors excluding current
        % flags i allow removal of kth neighbor from list
        % flags j allow addition of kth neighbor's neighbors excluding the current so it
doesn't consider itself to be a neighbor
        obj.MemberList(memIndex).NeighborList =
unique([obj.MemberList(memIndex).NeighborList(i); obj.MemberList(nk).NeighborList(j)]);
        % remove kth neighbor from list
        obj.MemberList(nk).ValidFlag = 0; % mark kth neighbor as
invalid
        % remove references to kth neighbor from its neighbors replacing with a reference
to the current
        nl2 = obj.MemberList(nk).NeighborList;
        nn = numel(nl2);
        for m=1:nn
            i = nl2(m); % i is mth neighbor of kth neighbor
            j = (nl2 == nk); % find reference to kth neighbor within mth neighbor's list
            obj.MemberList(i).NeighborList(j) = memIndex; % replace with reference to
current
        end
    end
end

```

```

obj.MemberList(memIndex).Volume= numel(obj.MemberList(memIndex).VoxelList);
obj.MemberList(memIndex).POI= sqrt( p / v) / v;
obj.MemberList(memIndex).ProjectionProfile= pp / v;
end

function hist = clusterSizeHist( obj )
    hist= zeros(1,1);
    m=numel(obj.MemberList);
    for i=1:m
        if obj.MemberList(i).ValidFlag
            n= obj.MemberList(i).Volume;
            if n > numel(hist)
                hist= [hist; zeros(n-numel(hist),1)];
            end
            hist(n)= hist(n) + 1;
        end
    end
end

function img = clusterSizeImage( obj )
    img= zeros(obj.ImageSpaceDims);
    for i=1: numel(obj.MemberList)
        if obj.MemberList(i).ValidFlag
            n= numel(obj.MemberList(i).VoxelList);
            if n ~= obj.MemberList(i).Volume
                error('VoxelList size and Volume mismatch');
            end
            for j=1:n
                [ix,iy,iz]= ind2sub(obj.ImageSpaceDims,obj.MemberList(i).VoxelList(j));
                img(ix,iy,iz)= n;
            end
        end
    end
end
end
end
end
end
end

```

[Published with MATLAB® R2017a](#)

```

classdef VoxelClusterMember < handle

    properties
        ValidFlag           % flag effectively allowing cluster member deletion (e.g. during
        Volume              % the volume of this cluster in voxels (i.e. the number of voxels
        in this cluster)
        VoxelList           % list of image space indices of the voxels within this cluster
        POI                 % the parameter of interest associated with each cluster
        NeighborList        % list of indices into VoxelClusterList identifying clusters that
        are neighbors to this one
    end
end

```

```

Origin          % 3D location of origin voxel within cluster (for debugging
purposes)
ProjectionProfile % column from W matrix corresponding to this cluster
end

methods
function obj = VoxelClusterMember(ImageSpace, POI values, W)
    % ImageSpace is a 3D matrix of zeros and ones where the ones define the interior or the
    imaged object
    % POI values is a vector with elements corresponding to the 1's in ImageSpace and
    containing the POI (Parameter of Interest)
    if nargin ~= 0
        [nX, nY, nZ] = size(ImageSpace); % get dimensions of the image space
        LastN= nX*nY*nZ; % last index
        Neighbors= [-1, -1-nX, -nX, +1-nX, +1, +1+nX, +nX, -1+nX, ...
            [0, -1, -1-nX, -nX, +1-nX, +1, +1+nX, +nX, -1+nX]-(nX*nY), ...
            [0, -1, -1-nX, -nX, +1-nX, +1, +1+nX, +nX, -1+nX]+(nX*nY)]; % determine relative
indices within ImageSpace of 26 neighbors
        VoxelIndices= find(ImageSpace == 1); % get indices of all voxels within
the imaged object
        N= numel(VoxelIndices); % how many are there
        if numel(POI values) ~= N
            error('The number of POI values must match the number of ones in ImageSpace');
        end
        if size(W, 2) ~= N
            error('The number of columns in W must match the number of ones in ImageSpace');
        end
        obj(N, 1) = VoxelClusterMember; % create one cluster member for each
and every voxel
        % create a cluster for each voxel within the object
        for i = 1:N
            obj(i, 1).ValidFlag= 1; % initially all
cluster members are valid
            obj(i, 1).Volume = 1; % initially just one
voxel
            obj(i, 1).VoxelList= [VoxelIndices(i)]; % this is the index
of the one voxel within ImageSpace
            tmpNeighborList= VoxelIndices(i) + Neighbors; % get all 6 neighbor
indices even if outside ImageSpace
            j= find(tmpNeighborList >=1 & tmpNeighborList <= LastN); % find those outside
ImageSpace
            tmpNeighborList= tmpNeighborList(j); % remove them
            j= find(ImageSpace(tmpNeighborList) == 1); % find those outside
the object
            tmpNeighborList= tmpNeighborList(j); % remove those too
            obj(i, 1).NeighborList= find(ismember(VoxelIndices, tmpNeighborList)); % find
corresponding indices into cluster MemberList
            obj(i, 1).POI = POI values(i); % assign associated
POI value
            obj(i, 1).ProjectionProfile= W(:, i); % assign associated
column of W
            [ix, iy, iz]= ind2sub([nX, nY, nZ], obj(i, 1).VoxelList); % Origin is redundant
to the original element in VoxelList

```



```
obj(i,1).Origin= [i x, i y, i z];  
    end  
end  
end  
end  
end
```

Published with MATLAB® R2017a