Ollscoil na Gaillimhe
University of Galway

Provided by the author(s) and University of Galway in accordance with publisher policies. Please cite the published version when available.

Title	Development of a high-resolution gamma-ray imaging system with synthetic collimation.
Author(s)	Havelin, Ronan
Publication Date	2013-11-01
Item record	http://hdl.handle.net/10379/3797

Downloaded 2024-04-17T03:50:23Z

Some rights reserved. For more information, please see the item record link above.

Development of a high-resolution gamma-ray imaging system with synthetic collimation

A THESIS

Submitted by

Ronan Havelin B.Sc. M.Sc.

in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Physics
College of Science
National University of Ireland, Galway

Academic Supervisor:
Dr. Mark J. Foley

November 2013

Abstract

This work outlines the development of a multi-pinhole single-photon emission computed tomography (SPECT) system designed to produce a synthetic-collimator image of a small field of view. The synthetic-collimator image was formed by estimating the activity distribution within the FOV and summing the activities in the voxels along the axis perpendicular to the detector face. A large-area, CCD-based gamma-ray detector, called BazookaSPECT, was used to detect gamma rays. A focused multi-pinhole collimator was constructed using rapid-prototyping and casting techniques. The collimator projected the centre of the field of view (FOV) through forty-six pinholes when the detector was adjacent to the collimator. Synthetic collimation requires that data are acquired using several system configurations. The object-collimator distance remained constant and the magnification was controlled by changing the collimator-detector distance. The amount of pinhole-projection overlap (multiplexing) increased as the system magnification increased. An advantage of synthetic collimation is that multiplexed data can be used, thereby improving the efficiency of data acquisition. The first system configuration acquires non-multiplexed projection data and each subsequent system configuration will make more efficient use of the detector by acquiring multiplexed data that can be demultiplexed. Synthetic collimation enables the use of multiplexing to produce high-resolution, artefact-free reconstructions. There was no rotation in the system; a single tomographic angle was used in each system configuration. Image reconstruction was performed using maximum-likelihood expectation-maximization (MLEM) and an experimentally measured system matrix. The system matrix was measured for each system configuration by translating a point source through a sparsely-sampled grid encompassing the FOV. An algorithm for multi-pinhole identification and tracking was developed. A 2D elliptical Gaussian distribution was fitted to each pinhole projection. The system matrix was interpolated using the coefficients of the distributions. Simulations with a hot-rod phantom demonstrated the efficacy of combining low-resolution non-multiplexed data with high-resolution multiplexed data to produce high-resolution reconstructions. The system was validated experimentally with a hot-rod phantom, and a small-animal imaging study was also performed.

Dissemination of Work

Havelin R J, Miller B W, Barrett H H, Furenlid L R and Foley M J, "Synthetic-collimator imaging using multiplexed and non-multiplexed multi-pinhole data", Phys. Med. Biol. (submitted).

Havelin R J, Miller B W, Barrett H H, Furenlid L R and Foley M J, "A SPECT imager with synthetic collimation", Proc. SPIE Vol. 8853, 885309.

Havelin R J, Miller B W, Barrett H H, Furenlid L R, Murphy J M, Dwyer, R M and Foley M J, "Design and performance of a small-animal imaging system using synthetic collimation", Phys. Med. Biol. 58 3397, 2013.

Havelin R J, Miller B W, Barrett H H, Furenlid L R, Murphy J M and Foley M J, "Development of a SPECT imager with synthetic collimation", IOP Spring Weekend, Conference Poster & Oral Presentation 2013.

Roisin M. Dwyer, James Ryan, Ronan J. Havelin, John C. Morris, Brian W. Miller, Zhonglin Liu, Richard Flavin, Cathal O'Flatharta, Mark J. Foley, Harrison H. Barrett, J. Mary Murphy, Frank P. Barry, Timothy O'Brien, and Michael J. Kerin. "Mesenchymal stem cell-mediated delivery of the sodium iodide symporter supports radionuclide imaging and treatment of breast cancer", STEM CELLS, 29(7):1149-1157, 2011

Roisin M. Dwyer, James Ryan, Ronan Havelin, John C. Morris, Cathal O'Flatharta, Brian Miller, Zhonglin Liu, Mark Foley, Harrison H. Barrett, Mary Murphy, Frank P. Barry, Timothy O'Brien and Michael J. Kerin. "Mesenchymal Stem Cell (MSC) mediated delivery of the Sodium Iodide Symporter (NIS) supports radionuclide imaging and treatment of breast cancer", American Association for Cancer Research (AACR). Conference Poster 2011.

M J Foley, R Havelin, Brian W. Miller, H. H. Barrett. "Development of a novel synthetic collimator small animal SPECT imaging system", Proceedings of the European Medical Physics & Engineering Conference, September 2011

Cian Kennedy, Ronan Havelin, Cathal O Flatharta, Mark Foley, Mary Murphy, Harrison Barrett, Frank Barry, "A low cost non-invasive small animal in vivo imaging system to investigate mesenchymal stem cell homing in a mouse model of rheumatoid arthritis", Regenerative Medicine IASI, International Conference in Romania, July 2010

Dwyer R, Ryan J, Havelin R et al. "Sodium iodide symporter (NIS) mediated tracking of mesenchymal stem cell (MSC) migration to breast tumours in vivo", Human Gene Therapy, Volume 20, Issue 11, November 2009

Acknowledgements

I would like to thank my supervisor Dr. Mark Foley for his support and guidance during this work. I am grateful to Dr. Harrison Barrett for providing me with the opportunity to work with him and his team at the Center for Gamma-Ray Imaging, University of Arizona, Tucson, AZ, USA. I would like to thank Dr. Lars Furenlid for the technical discussions and advice during my visit to CGRI. I am thankful to Dr. Eric Clarkson for his mathematical advice during this work. I am grateful to Dr. Brad Barber for his advice and assistance in carrying out much of the experimental work. I would like to express my special gratitude to Dr. Brian Miller for all his advice during this work.

I want to thank all my friends in the MPRC, CGRI and the School of Physics for all their help and support during this work. I would like to thank Jin, Sean, Elaine, Kevin, Kevin, Jevon, Dan, Ronan, Vaibhav, Abhinav, Helen, Esen, Cecile, Heather, Karr, Mike, Gordon, Margaret, as well as many others for their help during my studies. I would also like to thank all my other friends for supporting me during this work.

Last but not least I wish to express my love and gratitude to my family for all their encouragement and support throughout my life.

Contents

Abstract i

Dissemination of Work ii

Acknowledgements iii

List of Figures vii

List of Tables x

Nomenclature xi

1 Introduction 1

2 Background and Methods 4

2.1 Molecular Imaging 5
2.2 The Physics of Gamma-Ray Imaging 9
2.3 Introduction to SPECT Imaging 11
2.4 Collimation in SPECT 15
2.4.1 Pinhole Imaging 15
2.4.2 Multi-Pinhole Imaging 17
2.4.3 Parallel-Hole Imaging 19
2.4.4 Synthetic-Collimator Imaging 20
2.5 Digital Imaging Systems 24
2.6 Iterative Reconstruction Algorithms 26
2.6.1 Maximum-Likelihood Estimation 29
2.6.2 Maximum-Likelihood Expectation-Maximization (MLEM) 30
2.6.3 Image Reconstruction with MLEM 31
2.6.4 Ordered Subsets Expectation-Maximization 32
2.6.5 Implementation with CUDA 34
2.6.5.1 MLEM 35
2.6.5.2 POSEM 37
2.6.6 CUDA vs MATLAB™ 37

3 System Design **40**
3.1 Introduction . . . 40
3.2 Gamma Camera . . . 42
3.2.1 XX1332 Image Intensifier . . . 44
3.2.2 Imaging Lens . . . 44
3.2.3 CCD Camera . . . 45
3.3 Photon Counting with BazookaSPECT . . . 47
3.4 Large-Area BazookaSPECT Detector . . . 49
3.5 3D Printing . . . 52
3.6 Focused Multi-Pinhole Collimator . . . 54
3.7 SyntheticSPECT . . . 57
3.8 System Operation . . . 60

4 Performance Characterization of SyntheticSPECT **66**
4.1 Calibration of the Imaging System Matrix . . . 67
4.2 Pinhole Identification and Gaussian Fitting . . . 72
4.2.1 Effect of the Number of Counts on the Fitting Procedure . . . 77
4.3 Pinhole Tracking . . . 78
4.4 Interpolation of the System Matrix . . . 80
4.4.1 Validation with Experimental Data . . . 84
4.5 Multiplexing . . . 86
4.6 Simulations using the System Matrix . . . 88
4.6.1 Reconstructions with Multiplexed Data . . . 90
4.6.1.1 Simulation 1 . . . 92
4.6.1.2 Simulation 2 . . . 94
4.7 Fourier Crosstalk Matrix and Spatial Resolution . . . 96
4.8 Energy Resolution . . . 100
4.9 Conclusion . . . 101

5 Imaging with BazookaSPECT and SyntheticSPECT **104**
5.1 BazookaSPECT . . . 106
5.2 SyntheticSPECT . . . 111
5.2.1 Hot-Rod Phantom . . . 111
5.2.2 MLEM Reconstruction . . . 113
5.2.3 POSEM Reconstruction . . . 117
5.2.4 Small-Animal Imaging . . . 120
5.3 Conclusions . . . 127

6 Conclusions and Future Work **128**
6.1 Conclusions . . . 128
6.2 Future Work . . . 132

Appendix **135**

A CUDA Code **135**

A.1 Program Headers 135
A.2 Utility Functions 136
A.3 Masks 141
A.4 Index the Coefficients 142
A.5 Gaussian Kernels 144
A.6 MLEM 147
A.6.1 Functions 147
A.6.2 MLEM Main 156
A.7 POSEM 164
A.7.1 Functions 164
A.8 Fourier Crosstalk Matrix 166
A.8.1 Functions 166
A.8.2 Crosstalk Main 169

B MATLAB Code **175**
B.1 Main MATLAB Script 175
B.2 MATLAB Functions 177
B.2.1 Gaussian Fitting 177
B.2.2 Pinhole Tracking 184
B.2.3 Extrapolation of Pinhole Projections 191
B.2.4 Interpolation of Pinhole Projections 201

Bibliography **207**

List of Figures

1.1 A flow chart outlining the organization of this thesis. 3

2.1 The molecular imaging matrix. 7
2.2 Pinhole camera geometry. 15
2.3 Simple multi-pinhole camera geometry. 17
2.4 Parallel-hole collimator. 19
2.5 POSEM subset distribution. 33
2.6 CUDA Implementation of MLEM. 36

3.1 BazookaSPECT detector. 41
3.2 Columnar CsI scintillator. 43
3.3 Microchannel plate. 43
3.4 XX1332 image intensifier. 45
3.5 CCD component of the BazookaSPECT detector. 46
3.6 Photon cluster extraction. 48
3.7 Frame parsing. 49
3.8 Fibre-optic taper. 50
3.9 Light loss from fibre-optic taper. 51
3.10 Large-area BazookaSPECT detector. 51
3.11 Objet Geometries Ltd. Connex350^{TM}printer. 52
3.12 Collimator casing. 55
3.13 Focused multi-pinhole collimator. 57
3.14 Geometry of an off-centre focused pinhole aperture. 58
3.15 SolidWorks rendering of the collimator. 59
3.16 Detector casing. 60
3.17 Detector support structure. 60
3.18 Collimator holder. 61
3.19 SolidWorks rendering of the SyntheticSPECT assembly. 61
3.20 Image of the collimator and phantom. 62
3.21 Image of the SyntheticSPECT system. 63
3.22 The LabVIEW control interface for SyntheticSPECT. 64
3.23 LabVIEW interface workflow. 65

4.1 An integrated projection image. 68

4.2 Total counts recorded for the first system configuration. 70
4.3 Total counts recorded for the second system configuration. 70
4.4 Total counts recorded for the third system configuration. 70
4.5 Total counts recorded for the fourth system configuration. 71
4.6 Total counts recorded for the fifth system configuration. 71
4.7 Total counts recorded for the sixth system configuration. 71
4.8 Total counts recorded through the centre of the field of view. . . . 72
4.9 Projection image from a point source in the corner of the field of view. 73
4.10 Segmented pinhole projections. 74
4.11 Single pinhole projection. 74
4.12 2D Gaussian fit is made to every pinhole projection. 76
4.13 2D Gaussian fit to a pinhole projection. 77
4.14 Counts required for Gaussian fitting. 78
4.15 Projection of the centroid of the 2D Gaussian fits. 81
4.16 Gaussian coefficients of a central pinhole. 82
4.17 Gaussian coefficients of a peripheral pinhole. 83
4.18 Effect of splines on the interpolation. 85
4.19 Multiplexing pinhole projections. 87
4.20 Percentage of pinhole projections. 88
4.21 Simulated low-magnification projection data. 90
4.22 Simulated high-magnification projection data. 91
4.23 Reconstruction of the simulated data. 91
4.24 Signal-to-noise ratio as a function of contrast. 92
4.25 Simulated phantom. 92
4.26 Simulation results. 95
4.27 Reconstruction of the simulated hot-rod phantom using only multiplexed data. 96
4.28 Reconstruction of a simulated hot-rod phantom using non-multiplexed and multiplexed data. 97
4.29 Fourier crosstalk matrix: Normalized MTF. 100
4.30 Energy resolution. 101

5.1 Timeline of *in vivo* mesenchymal stem cell-sodium iodide symporter imaging and therapy experiments. 107
5.2 BazookaSPECT images of mice. 109
5.3 *In vivo* radioiodine therapy of breast tumour xenografts. 111
5.4 Hot-rod phantom schematic. 112
5.5 Experimental projections of the hot-rod phantom. 114
5.6 (a) Slices through the reconstruction of the hot-rod phantom. . . 115
5.7 (b) Slices through the reconstruction of the hot-rod phantom. . . 116
5.8 Line profiles of the reconstructed rods of the hot-rod phantom. . 117
5.9 Reconstruction of the hot-rod phantom with MLEM iterations. . 118
5.10 POSEM reconstruction of the hot-rod phantom using one system configuration. 119

5.11 POSEM reconstruction of the hot-rod phantom using all system configurations. 120
5.12 Schematic of mouse translated through FOV. 122
5.13 Projections (1st section of animal) of a mouse injected with ^{99m}Tc-MDP. 123
5.14 Projections (2nd section of animal) of a mouse injected with ^{99m}Tc-MDP. 123
5.15 Projections (3rd section of animal) of a mouse injected with ^{99m}Tc-MDP. 124
5.16 Projections (4th section of animal) of a mouse injected with ^{99m}Tc-MDP. 124
5.17 Projections (5th section of animal) of a mouse injected with ^{99m}Tc-MDP. 125
5.18 Full-body reconstruction of a mouse injected with ^{99m}Tc-MDP. . 126

List of Tables

2.1 Common PET and SPECT nuclides used for imaging. 9
2.2 Physical and scintillation properties of scintillators. 11
2.3 Small-animal SPECT systems. 13
2.4 Graphics card specifications. 35
2.5 CUDA vs MATLAB™. 38

3.1 Phosphor screens for use with BazookaSPECT. 44
3.2 Lens specifications. 45
3.3 CCD Specifications. 46
3.4 3D printing materials data sheet. 53

4.1 System configurations. 69
4.2 Storage requirements of the system matrix. 84
4.3 Validation of the coefficient interpolation. 86
4.4 Simulated test cases. 93

5.1 *NIS* and *MRPL19* expression in animal tissues. 110
5.2 Acquisition settings for phantom study. 113
5.3 MLEM and POSEM reconstruction time. 120

Nomenclature

α	Opening angle of the pinhole.
$\bar{x}$	X-position of the Gaussian distribution.
$\bar{y}$	Y-position of the Gaussian distribution.
$\beta_{kk'}$	A measure of the degree of aliasing or crosstalk between the two different frequencies.
$\mathbf{f}$	The discrete distribution of radioactivity in object space.
$\mathbf{g}$	The discrete measurements in image space.
$\mathbf{H}$	The discrete system matrix that maps the object vector to an image.
f	The continuous distribution of radioactivity in object space.
Γ	Ideal tube integrals of a parallel-projection image.
$\hat{\vartheta}$	An estimate of the parameters of interest.
λ	The decay constant.
μ	The linear attenuation of the collimator material.
ν_{par}	The geometric sensitivity of a parallel-hole collimator.
ϕ	A basis function.
Ψ	Fourier-space equivalent of $\boldsymbol{H}$.
ρ	Correlation coefficient of the Gaussian distribution.

σ_x X-width of the Gaussian distribution.

σ_y Y-width of the Gaussian distribution.

θ Gamma-ray incidence angle measured from the plane of the pinhole.

ϑ The parameters of interest.

A Amplitude of the Gaussian distribution.

A_b The background activity within the simulated phantom.

a_e The parallel-hole collimator effective hole length.

A_s The rod activity within the simulated phantom.

C_e The Gaussian coefficients of the finely-sampled, experimentally-measured data.

C_i The Gaussian coefficients of the interpolated data.

d_e The effective diameter of the pinhole.

d_{FOV} Diameter of the spherical field of view.

D_i Diameter of an image on the detector.

d_{OP} The object-collimator distance.

d_{PD} The collimator-detector distance.

d_p The pinhole diameter.

F Fourier coefficients.

h_m The sensitivity function that specifies the response of the detector element.

K The parallel-hole collimator hole shape factor.

k A multi-index using for Fourier representation.

m Detector element index.

N The activity of a radioactive sample.

n 3D location index in object space.

P_{16} A POSEM subset pattern.

r Three-dimensional position vector.

R_0 The total system resolution.

r_1 The radius of the FOV.

r_2 The radius of the projection on the detector.

R_g The geometric resolution.

R_i The detector intrinsic resolution.

S The support region of an object.

S_{ph} Sensitivity of the pinhole aperture.

t The parallel-hole collimator septal thickness.

t_j Tube function within the region of the jth vertical tube.

w The parallel-hole collimator hole width.

CHAPTER 1

Introduction

Molecular imaging of small laboratory animals is a vital tool in the study of disease. Small-animal models of human disease provide important information regarding molecular pathways associated with many different diseases and pathological processes. There have been many advances in molecular imaging because of improvements in molecular and cell biology techniques as well as improvements in imaging instrumentation.

Chapter 2 provides some background information on molecular imaging. This work will focus on single-photon emission computed tomography (SPECT) and its role in improving our understanding of the biological processes within small animals. Some of the different methods of collimation used in nuclear imaging are outlined in section 2.4. A mathematical representation of a digital imaging system is given in section 2.5. The system operator, $\mathbf{H}$, that maps the object vector to an image is discussed in this section. Section 2.6 outlines how $\mathbf{H}$ can be used with iterative reconstruction algorithms to estimate the activity distribution of a radiotracer within the field of view, given a set of projection data.

This work focused on further development of high-resolution scintillation detectors that use a charge-coupled device (CCD) sensor and a columnar scintillator, and development of a multi-pinhole collimator for use with traditional SPECT and synthetic collimation. A BazookaSPECT system was developed at the National University of Ireland Galway (NUIG) in collaboration with the Regenerative Medicine Institute (REMEDI). The SyntheticSPECT system was developed

in collaboration with the Center for Gamma-ray Imaging (CGRI), University of Arizona, Tucson, AZ, USA. The contributions to the work by the author and the collaborators are outlined in detail in each chapter.

Chapter 3 describes the imaging system developed during this work and the novel fabrication of the multi-pinhole collimator used. The group at CGRI developed the BazookaSPECT detector which was central to this project. The BazookaSPECT detector is discussed in section 3.2. Section 3.4 discusses the merits of coupling the BazookaSPECT detector to a fibre-optic taper to increase its active area. The collimator was fabricated using 3D rapid prototyping (section 3.5) and was cast with a tungsten epoxy. Constructing this collimator with more traditional machining methods would have been extremely difficult and expensive. The SyntheticSPECT imaging system (section 3.7) uses the multi-pinhole collimator and a large-area BazookaSPECT detector to synthesize parallel-projection images. The author designed the SyntheticSPECT system, fabricated the multi-pinhole collimator and developed the software required to process the calibration, phantom and small-animal data acquired by the system.

Chapter 4 presents the characterization of the synthetic-collimator imaging system. Calibration was performed by translating a point source through the field of view in a 3D grid pattern. A Gaussian distribution was fitted to every pinhole projection. The coefficients of these fits were then interpolated to a finer grid. Synthetic collimation allows for the overlap of projections on the detector. Section 4.5 shows the level of multiplexing at each system configuration. The interpolated system matrix was used to simulate the imaging system and to test the reconstruction algorithm (section 4.6). The Fourier crosstalk matrix is outlined in section 4.7. The crosstalk matrix provides a summary measure of resolution of the hardware alone, averaged over the field of view (FOV).

Chapter 5 presents some initial imaging results from the BazookaSPECT system at NUIG and the SyntheticSPECT system at CGRI. A small-animal imaging study using the BazookaSPECT was conducted at NUIG in collaboration with REMEDI. This work demonstrated the ability of the BazookaSPECT to image biological processes *in vivo*. A phantom and a small-animal imaging study were carried out at CGRI using the SyntheticSPECT system. This work experimentally validated the theory of the synthetic collimator. It has previously been demonstrated that synthetic collimation can be used to reconstruct an object using slit-slat collimators and multiple tomographic angles. The work presented

here experimentally demonstrates synthetic collimation using a multi-pinhole collimator and a single tomographic angle.

Chapter 6 summarizes the contribution of this research and provides suggestions for future work.

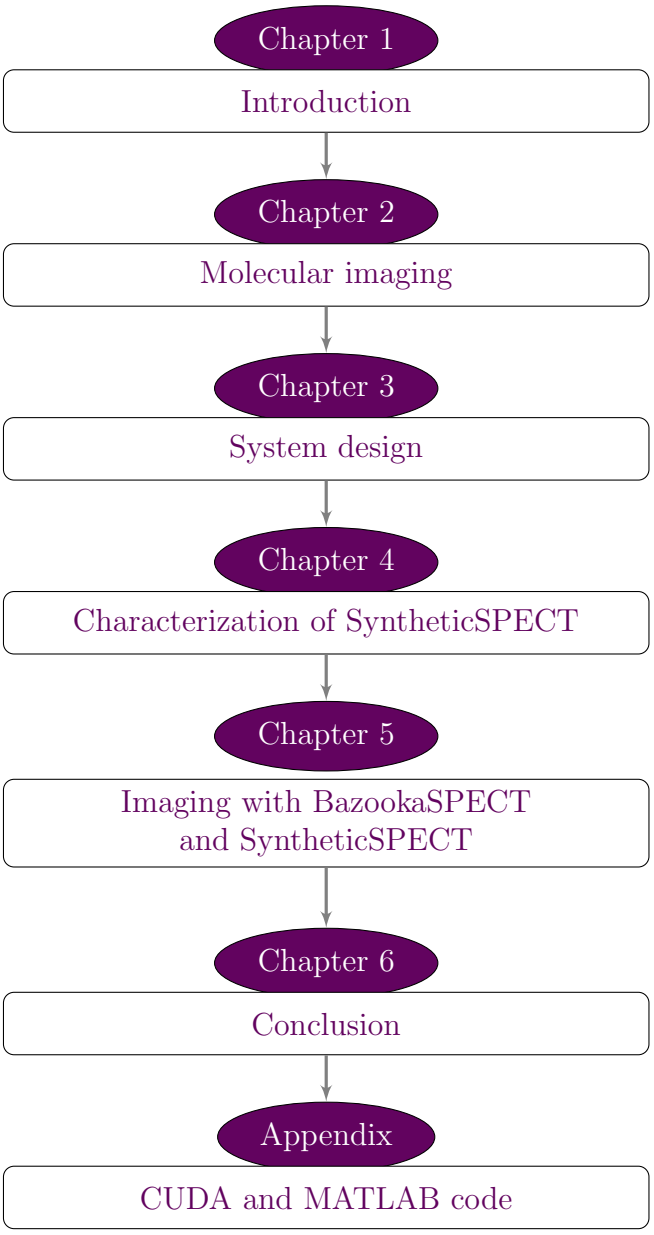


FIGURE 1.1: A flow chart outlining the organization of this thesis.

CHAPTER 2

Background and Methods

This chapter gives a short introduction to the motivation and theory behind nuclear imaging. Section 2.1 gives a short overview of molecular imaging. The physics of gamma-ray imaging is discussed in section 2.2. A brief history of SPECT, as well as some recent advances, is given in section 2.3.

Most SPECT systems use some form of mechanical collimation for image formation. A collimator limits the photon pathways between the detector and the source. Several methods of gamma-ray collimation are described in section 2.4. The focus of this work was the development of a synthetic collimator (section 2.4.4). The synthetic collimator uses multi-pinhole collimation and acquires images at different magnifications. These data are combined to synthesize a parallel-projection image. It is difficult to fabricate a parallel-hole collimator with all the characteristics required for small-animal SPECT. Pinhole collimators are much easier to fabricate rapidly and are more suitable for certain tasks. The synthetic collimator, first introduced by Clarkson et al., 1998, is designed to overcome the disadvantages of the parallel-hole collimator (section 2.4.3), and to produce ideal parallel-projection images.

A mathematical description of a digital imaging system is provided in section 2.5. This section introduces the system matrix, a matrix that maps an object vector to an image. This matrix can contain information about the system geometry, attenuation and scattering of radiation, radiometric factors, pinhole penetration, and detector blur.

Section 2.6 gives a short introduction to iterative reconstruction algorithms. The maximum-likelihood expectation-maximization (MLEM) algorithm and an accelerated ordered subsets algorithm were used to reconstruct the object in this work. Section 2.6.5 outlines the implementation of these reconstruction algorithms using graphics processing units (GPU).

2.1 Molecular Imaging

Molecular imaging is based on a variety of non-invasive imaging techniques, including several based on the tracer principle (Chiewitz and Hevesy, 1935). Small-animal molecular imaging is an important tool in the study of disease and it enables the visualization of biological processes *in vivo* in small animals (Cherry, 2004; Meikle et al., 2005). These imaging techniques are used to detect signals that originate from injected molecules and to observe their interaction with cellular targets *in vivo* (Massoud and Gambhir, 2003; Meikle et al., 2005). The measurements can be used to locate the position in the body where the proteins or genes are expressed, the strength of expression, and their spatial and temporal changes. The modalities used in molecular imaging are computed tomography (CT), magnetic resonance imaging (MRI), ultrasound (US), optical imaging (OI), positron emission tomography (PET), and single-photon emission computed tomography (SPECT). All of these have their advantages and disadvantages but are usually most useful when used simultaneously to image a subject. Hybrid imaging systems allow the combination of functional and anatomical image formation (Beyer et al., 2011; Hasegawa et al., 1993; Kemp et al., 2010; Townsend, 2008). While these imaging modalities are extremely useful in determining the efficacy of treatment in humans (clinical), animal studies (preclinical) are usually required to precede human tests for approval of biomedical products. Small-animal models represent an important bridge between discoveries at the molecular level and implementation of clinically relevant diagnostics or therapeutics (Franc et al., 2008). Animal models have been crucial in the understanding of molecular pathways associated with cardiovascular disease (Golestani et al., 2010; Recchia and Lionetti, 2007; Snider and Conway, 2011), cancer (Jong and Maina, 2010; Peters et al., 2007), and many other pathological processes (Acton and Kung, 2003; Franc et al., 2008). They are also important due to the ethical and practical concerns of human experimentation (Jong and Maina, 2010). Animal

studies can provide pharmacological and toxicological data and may predict the clinical efficacy of new compounds (Cherry, 2001). Rodents are most commonly used for these studies due to their size, cost, short gestation and life span, and genomic similarity to humans. Although mice and humans are genomically similar, their respective phenotypes can be very different. While animal studies can be of great value, their results are not always applicable to humans due to the inherent biological differences (Jong and Maina, 2010). For example, the heart rate of a mouse is approximately six hundred beats per minute, compared with eighty per minute for a human. This faster rate means that biological processes that may take hours to develop in a human could take seconds or minutes in a mouse. Despite these differences, the success of the animal models has led to an increased demand for *in vivo* imaging in the preclinical arena. Initially, small-animal studies often relied on methods requiring euthanasia of the animal. These methods limited the possibility of studying a single animal over time. It was assumed that data from different animals at different time points accurately represented a molecular process. Many dedicated small-animal imaging systems have been developed in recent years to overcome these problems and to enable dynamic imaging of molecular processes (section 2.3). Due to the small size of the subjects to be imaged, preclinical imaging offers an attractive area to explore new technologies. It is possible to make improvements to each of the modalities to make them more suitable to small-animal molecular imaging (Cherry, 2004). Improvements in spatial, energy and temporal resolutions, sensitivity, and quantification ability are arguably the most desired, although size and ease of use are also considered. These improvements will allow a better understanding of gene expression by mapping it to high-resolution anatomical information or enabling imaging of multiple gene expressions simultaneously. Preclinical imaging is viewed as a testing ground for new detector technologies, although some developments often prove too difficult to apply to clinical applications (Peterson and Furenlid, 2011).

Optical techniques, such as diffuse optical tomography, fluorescence tomography, and photoacoustic tomography are widely used in preclinical molecular imaging (Weissleder, 2002). This is mainly due to the wide range of biologically relevant molecules that can be tagged with different optical contrast agents, providing flexibility to the contrast mechanism and the measurement technique (Gibson and Dehghani, 2009). A significant advantage of using optical techniques

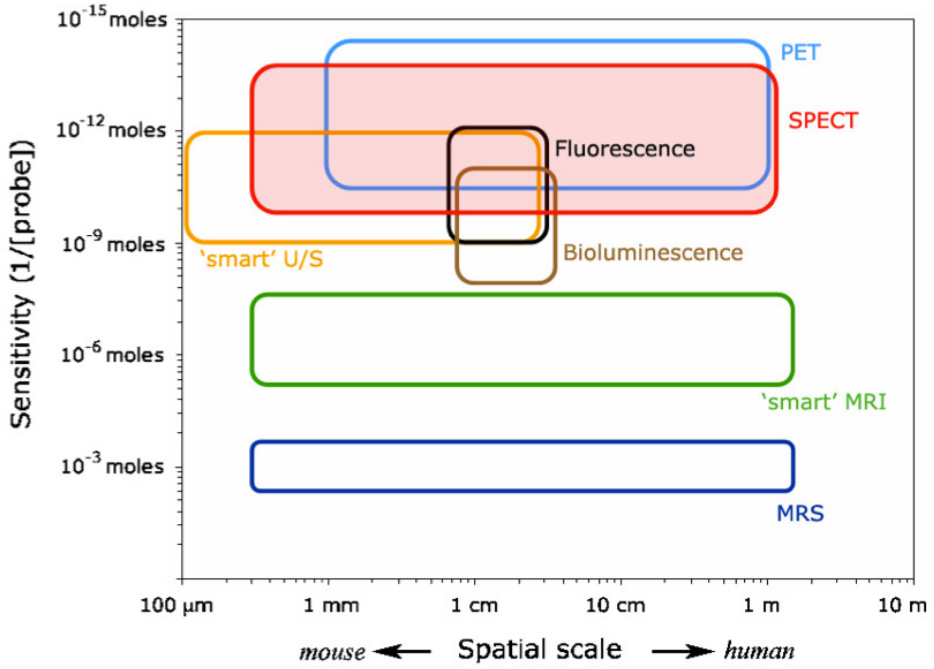


FIGURE 2.1: The molecular imaging matrix indicating the relative position of some non-invasive imaging techniques. In this figure, sensitivity refers to the *in vivo* concentration of the labelled contrast agent required to elicit a certain signal-to-noise ratio. The spatial scale is the domain bounded by the limiting resolution (left limit) and the depth of penetration of the signal-generating radiation (right limit) for a given technology. 'Smart U/S' refers to high-frequency ultrasound imaging using activatable contrast agents. (Reproduced from Meikle et al., 2005.)

for molecular imaging is that the translational route from small-animal imaging to clinical imaging is well established. Fluorescent breast imaging has been demonstrated, suggesting that fluorescent signals can be detected even through dense tissue such as the breast (Hawrysz and Sevick-Muraca, 2000). Another optical approach that has been gaining interest is photoacoustic imaging (Xu and Wang, 2006). Near infra-red light illuminates the volume of interest and the energy is preferentially absorbed by chromophores in the target molecules. Regions with increased chromophore density absorb more energy, and the region heats up and expands, emitting a short ultrasonic pulse. A functional image may be reconstructed by detecting and processing these pulses. Very high resolution is achievable (<50 μm), but the source of the ultrasound needs to be near the surface of the object.

For certain applications, nuclear imaging has emerged as possibly the most useful molecular imaging modality in preclinical studies (Meikle et al., 2005).

SPECT and PET are the most sensitive modalities (figure 2.1) and can detect tracers at nanomolar and picomolar concentrations (Spanoudaki and Ziegler, 2008). SPECT and PET are both useful due to the wide range of nuclides available, each having different characteristics (table 2.1). These isotopes can be attached to a wide variety of pharmaceuticals to probe specific biological targets and functions (Welch and Redvanly, 2005). The spatial resolution of PET imaging is limited by the fundamental process of positron emission and annihilation (1 - 2 mm). The radiotracers used in PET are labelled with positron emitters. A proton of the nucleus is converted to a neutron with the simultaneous emission of a positron and neutrino. This positron, depending on its energy, will travel a specific distance in the surrounding material, depositing energy until it annihilates with an electron, resulting in the emission of two photons. These two photons travel in approximately opposite directions and can be simultaneously detected using two detectors (Phelps et al., 1975). This information can be used to determine a line along which the annihilation probably occurred. This association between the two detected photons is called electronic collimation. Sensitivity of PET systems is usually greater than SPECT systems because mechanical collimation is not required. SPECT does not involve the simultaneous detection of correlated photons. This technique records the single gamma ray directly after emission and is not affected by the same fundamental resolution limit as PET. Isotopes used in SPECT emit gamma rays at different energies. It is possible to image several radionuclides simultaneously if the energy resolution of the detector is sufficient. The energy resolution of a detector indicates its ability to discriminate between events of different energies.

SPECT was identified as the ideal modality to achieve the objectives of the Regenerative Medicine Institute, NUIG, a collaborator in this work. The objective of the work was to investigate the mesenchymal stem cell-mediated delivery of the sodium iodide symporter (section 5.1). Mesenchymal stem cells (MSCs) play a primary role in tissue regeneration and they have the ability to migrate to the site of multiple tumour types (Spaeth et al., 2008). MSCs have the ability to bypass host immune surveillance which makes them an attractive vehicle for tumour-targeted delivery of therapeutic agents. The ability to dynamically image MSC migration and engraftment in real time to confirm tumour targeting is essential to the understanding of MSC-mediated delivery of therapeutic agents. The MSCs can uptake ^{99m}Tc (a common SPECT radionuclide) to allow for non-invasive imaging to track MSC engraftment at the tumour site. The ability of

SPECT to track the migration and engraftment of MSCs was the main impetus behind the development of the imaging systems described in this work.

TABLE 2.1: Common PET and SPECT nuclides used for imaging (Rowland and Cherry, 2008).

Radionuclide	Half-life	Principle γ-rays (keV)	Modality
^{11}C	20.39 (min)	511	PET
^{13}N	9.97 (min)	511	PET
^{15}O	2.04 (min)	511	PET
^{18}F	109.77 (min)	511	PET
^{99m}Tc	6.02 (h)	140	SPECT
^{111}In	2.8 (days)	171, 245	SPECT
^{123}I	13.22 (h)	159	SPECT
^{131}I	8.03 (days)	364	SPECT

2.2 The Physics of Gamma-Ray Imaging

Most SPECT studies use radiotracers that emit gamma rays in the 30-250 keV energy range. Within this range, the gamma ray interacts with the scintillator, and within the object itself, through photoelectric absorption and Compton scatter. The interactions most likely occur due to photoelectric absorption at the low energy ranges used in SPECT. These interactions result in the emission of light from the scintillator material in a process called scintillation (Knoll, 1979). A simple model of this process is that electron-hole pairs that are created in the scintillator remain bound to each other and recombine at luminescent centres where they emit light that is not reabsorbed by the scintillator (Peterson and Furenlid, 2011). In photoelectric absorption a gamma ray interacts with a core electron of one of the atomic constituents of the detector material. The energy of the gamma ray is divided between the binding energy of the core electron before its interaction and its kinetic energy after excitation. In Compton scattering, a gamma ray interacts with a loosely bound electron and is deflected from its original path, but in the process it transfers some of its energy and momentum to the electron. If the first interaction of a photon is via Compton scattering the resulting secondary photon could travel a significant distance within the scintillator before undergoing a photoelectric interaction. This could lead to incorrect localization of the interaction. It is preferable to have the photoelectric effect be

the primary method of conversion due to the localized energy deposition. The probability of photoelectric interaction scales as Z^4, so it is desirable to have a detector with a high effective atomic number.

Scintillators in SPECT are typically made from NaI(Tl), CsI(Tl), $LaCl_3$(Ce), $LaBr_3$(Ce) and $YAlO_3$(Ce) (Peterson and Furenlid, 2011). The gamma rays interact with the scintillators and optical photons are emitted. The emitted optical photons can be converted into electrical charge in many different ways. Examples of current scintillation based devices are PMTs, position-sensitive PMTs (Watanabe et al., 1999), multi-anode PMTs (Kubo et al., 2007), photodiodes (Gruber et al., 2002), avalanche photodiodes (Shah et al., 2001), position-sensitive avalanche photodiodes (Funk et al., 2006), charge-coupled devices (CCD) (Miller et al., 2007), CMOS detectors (Heanue et al., 1995), and the silicon photomultiplier (SiPM) (Herbert et al., 2006). Peterson and Furenlid, 2011 provides a good overview of each of these detectors.

Semiconductor radiation detectors are an alternative to scintillator-based single-photon imaging systems. These are direct-conversion devices, and do not require a separate scintillator to absorb the gamma rays. These detectors are commonly made from Si, Ge, CdTe, CdZnTe, HgI_2 (Barber and Woolfenden, 2006).

Irrespective of the gamma-ray absorption and read-out method, the objective is to convert the energy of the gamma-ray photon into an electrical signal. This electrical charge is then digitized and sent to a computer for further processing. The objective is usually to estimate the interaction position of the gamma ray in the scintillator and the energy of the photon. The interaction position is often estimated using Anger arithmetic which involves calculating the centroid using weighted detector outputs (Anger, 1958). Estimating these parameters while accounting for the statistical nature of the interaction process typically results in improvement performance (Barrett et al., 2009). The probability that a gamma ray will interact within a material is proportional to the density (ρ) and atomic number (Z) of the material (Hubbell and Seltzer, 1995). The greater these values, the more likely it will absorb a gamma ray. The physical and scintillation properties of some common scintillators used in high-resolution imaging are shown in table 2.2 (Spanoudaki and Ziegler, 2008; Weber and Ivanovic, 1999). Increasing the thickness of a scintillator will increase the number of gamma rays that are detected, but this can also lead to a degradation in spatial and energy resolutions due to the uncertainty in the depth at which the interaction occurred (Barrett

et al., 2009). Photons can also interact with matter inside the object. This interaction can lead to incorrect localization of the radiotracer within the object, and it is a source of "scatter". The energies of scattered photons are lower than those of unscattered or primary photons, and energy windowing can be used to filter these events from the data. The decay time of the material is the time over which 67% of the light from an interaction is released. This affects the count rate capability of the detector and its ability to distinguish events. Hygroscopic materials require a moisture-free environment to prevent degradation of the scintillator. The light output of a scintillator is proportional to the energy of the incident gamma ray (Dorenbos, 1995), and it is an important factor is determining the energy and spatial resolutions of the system. Light output obeys Poisson statistics, and a higher light output results in a higher signal-to-noise ratio. The wavelength of the light should be well matched to the absorption characteristics of the detector to maximize the number of photo-generated electrons. The ratio of the number of photo-generated electrons to the number of incident scintillation photons is known as the quantum efficiency of the detector.

Table 2.2: Physical and scintillation properties of scintillators (Spanoudaki and Ziegler, 2008; Weber and Ivanovic, 1999).

Scintillator	ρ(g/cm^3)	Effective Z	Decay Time (ns)	Photon Yield / keV	Hygroscopic	Peak Emission (nm)
NaI(Tl)	3.67	50.6	230	38	Y	415
CsI(Tl)	4.51	54.1	1000	52	N	540
BGO	7.13	74.2	350	8	N	480
LSO	7.4	65.5	42	20-30	N	420
$LaBr_3$	5.29	46	26	63	Y	380

2.3 Introduction to SPECT Imaging

In the 1950s Anger developed the scintillation camera which led to the practical implementation of gamma-ray imaging in the clinic. The basic design of this imaging system involved placing a large scintillation crystal, which was read out by an array of photomultiplier tubes, behind a collimator. This design is commonly referred to as an "Anger camera" (Anger, 1952). With the recent demand

for high-resolution imaging there have been many developments in SPECT detector technology. Currently, preclinical SPECT systems are required to have spatial resolutions which are over a factor of ten better than is available for human imaging. This is required to image the pathways and processes that interest preclinical researchers. The desire for improved spatial, temporal and energy resolutions, as well as improved sensitivity, has spawned many different methods of collimation and detection (Beekman et al., 2005; Mahmood et al., 2010; Miller et al., 2006; Peterson and Furenlid, 2011; Shokouhi et al., 2009). There are currently many different types of detectors, both scintillator based and direct conversion devices. Each of these detectors and scintillation materials have their respective merits, but there is usually a trade-off between desired properties. The geometries of the collimator can also be optimized (Cao et al., 2005; Mok et al., 2005; Nuyts et al., 2009; Rentmeester et al., 2007). The relative importance of the different aspects of performance varies by application and imaging task (Barrett et al., 1995). These systems can be general purpose SPECT imagers or they can be developed specifically for an application, such as cardiac imaging (Gambhir et al., 2009; Garcia et al., 2011), or rodent brain imaging (Rogulski et al., 1993). Table 2.3 gives a brief description of some of the small-animal SPECT systems in the literature.

Most SPECT systems use a collimator for image formation. A collimator limits the photon pathways between the source and the detector. Common examples of collimators include parallel-hole collimators and pinhole collimators, the merits of which have been well studied (Barrett and Swindell, 1981; Gunter, 1996; Jaszczak et al., 1994). Coded-aperture collimators are also implemented in some systems (Accorsi et al., 2001; Meikle et al., 2002). The collimator increases the spatial resolution of the photons by reducing the possible emission locations of each detected photon. The parallel-hole collimator has no focal point and ideally allows only photons with paths perpendicular to each detector element to be detected. In effect, each detector element would be estimating the total activity in a vertical tube above it and would not respond to activity elsewhere. Actual parallel-hole collimators fall short of this ideal due to depth-dependent blur and septal penetration. The pinhole collimator produces a cone-beam image on the detector with the focus at the pinhole. Single-pinhole imaging is often plagued by relatively poor geometric efficiency, and one method for increasing the sensitivity is to use multi-pinhole collimators (Barrett and Swindell, 1981). A brief description of each of these collimation methods is given in section 2.4.

TABLE 2.3: Some small-animal SPECT systems from the literature.

Reference	Description	Resolution (mm)
Goertzen et al., 2005	mouseSPECT: Eight 1 mm pinholes with rotating collimator	1.7 mm
Kim et al., 2006	CZT detector with eight 0.5 mm pinholes	1.45 mm
Zeniya et al., 2006	Pixellated gamma camera with 1.0 mm pinhole	2.5 - 2.8 mm
Accorsi, 2007	CdTe detector with single 0.4 mm pinhole	2.1 mm
Vastenhouw and Beekman, 2007	Stationary SPECT with 75 0.6 mm pinholes	0.45 mm
Miller et al., 2008	BazookaSPECT with coded-aperture (674 25 μm pinholes)	0.03 mm (planar)
Qian et al., 2008	Pixellated NaI array with 5 1 mm pinholes.	1.0 mm
Have et al., 2009	Stationary SPECT with 75 0.6 mm and 1.0 mm pinholes	0.45, 0.8 mm
Meng et al., 2009	Multi-pinhole EMCCD SPECT (150, 200, 300 μm)	< 0.2 mm
Shokouhi et al., 2009	SiliSPECT: 127 250 μm pinholes with 10 mm FOV	0.1 mm

A SPECT system that does not use collimation has been proposed by Mitchell and Cherry, 2009.

A basic SPECT system consists of one or more gamma cameras that rotate around the object to acquire data from several tomographic angles. The gamma cameras can also be kept stationary and the object can rotate within the field of view (Habraken et al., 2001). Equivalent data are acquired using both methods. The projection images that are acquired are then used to reconstruct the source distribution within the object. Each projection image is the two-dimensional compression of the three-dimensional tracer distribution. Regardless of the method of collimation, the 2D image is usually a summation of

all the slices within the object. The presence of under- and over-laying tracer obscures the data at a given depth. By imaging the object many different angles it is possible to estimate the 3D tracer distribution. The 3D tracer distribution may be accurately reconstructed if the data collected spans a continuous set of projection angles around the object. Errors are introduced by the finite step-size of the detector, noise in the data and other detector limitations (Barrett, 1990a). Using traditional image reconstruction algorithms, such as the filtered back projection (FBP) algorithm, these errors could result in significant artefacts in the reconstructed image (Beekman et al., 2001; Herman, 1980; Tsui et al., 1994). These artefacts can be reduced, and superior image reconstructions produced, if statistical iterative reconstruction methods (section 2.6) are used (Bruyant, 2002).

Two imaging systems based on single-photon emission are discussed in this work: BazookaSPECT and SyntheticSPECT. SPECT data were acquired by BazookaSPECT by rotating the animal 360 degrees and acquiring data at fixed angular steps. SyntheticSPECT acquired limited-angular tomographic data and an accurate estimate of the 3D tracer distribution was not possible. Although only limited-angle tomography was possible, it does not follow that this system produced no relevant information. As discussed briefly, the objective of the work in REMEDI was to track MSCs. The targeted location was a flank tumour, with no under- or over-laying structures. Limited-angular tomography provided the ideal solution to this application by providing good lateral resolution with rough depth discrimination; a full SPECT reconstruction was unnecessary. The SyntheticSPECT system, designed to implement synthetic collimation (section 2.4.4), uses limited-angle tomographic data to synthesize an ideal parallel-projection image but to also provide some depth information. The term 'SPECT' was retained in name of the system because of its potential to acquire SPECT data if an application required it.

2.4 Collimation in SPECT

2.4.1 Pinhole Imaging

An inverted image of the field of view is produced when it is imaged through a pinhole. The size of the projected image depends on the relative distance between the pinhole and the object and the pinhole and the projection surface. Pinhole collimators have been used in nuclear medicine since the earliest gamma cameras (Anger, 1958; Mallard and Myers, 1963). A brief overview of pinhole collimation is given in Beekman and Have, 2007, but a more thorough discussion of pinhole imaging can be found in Barrett and Swindell, 1981.

The spatial resolution of a pinhole imaging system, R_o, is given by

$$R_0 \simeq [R_g^2 + (\frac{d_{OP}}{d_{PD}} R_i)^2]^{0.5}, \tag{2.1}$$

where R_g is the geometric resolution of the system, R_i is the intrinsic resolution of the detector, d_{OP} is the distance from the object to the pinhole, and d_{PD} is the distance from the pinhole to the detector (figure 2.2). The magnification is the ratio of d_{PD} to d_{OP}. A larger magnification can be used to compensate for the limited resolution of the detector, but the FOV will be reduced. Equation

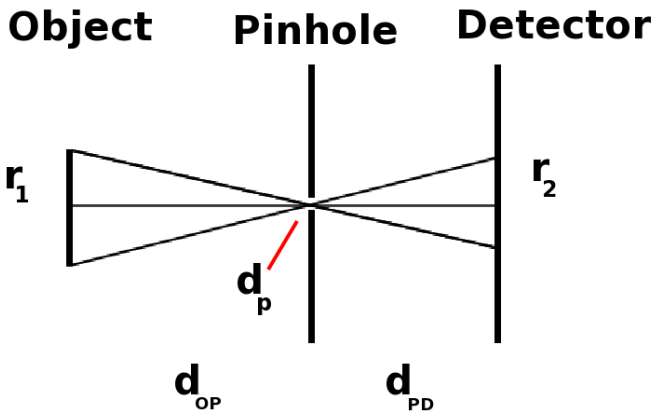


FIGURE 2.2: Pinhole camera geometry: r_1 is the radius of the object, r_2 is the radius of the projection on the detector, d_p is the pinhole diameter, d_{OP} is the distance from the object to the pinhole, and d_{PD} is the distance from the pinhole to the detector. The magnification is the ratio of d_{PD} to d_{OP}.

2.1 shows that the planar resolution of the system will approach the diameter

of the pinhole much faster with a high-resolution detector than a low-resolution detector. The geometric resolution, R_g, of a pinhole collimator is given by

$$R_g = (d_{PD} + d_{OP})d_e/d_{PD}, \tag{2.2}$$

where d_e is the effective diameter of the pinhole aperture. Due to penetration at the edge of the pinhole, the effective pinhole diameter, d_e, is given by

$$d_e = [d_p(d_p + \frac{2}{\mu}tan(\alpha/2))]^{0.5}, \tag{2.3}$$

where d_p is the physical pinhole diameter, α is the opening angle of the pinhole, and μ is the linear attenuation of the collimator material at a particular gamma-ray energy.

The sensitivity of an ideal pinhole is given by

$$S_{ph} = \frac{sin^3\theta}{16}(\frac{d_e}{d_{OP}})^2, \tag{2.4}$$

where θ is the incidence angle measured from the plane of the pinhole ($\theta = \pi/2$ for normal incidence). Equations 2.2 and 2.4 show that reducing pinhole diameter will result in an improved spatial resolution, but the geometric sensitivity will be considerably reduced. The sensitivity-resolution trade-off at the centre of the FOV is given by

$$S_{ph} = \frac{R_g^2}{16}(\frac{1}{d_{OP} + d_{PD}})^2(\frac{d_{PD}}{d_{OP}})^2. \tag{2.5}$$

Consequently, longer acquisition times may be required for high-resolution imaging using a pinhole system. This could be a problem due to the half-life of the isotope being used or the length of time the subject may be anaesthetized.

Assuming an ideal pinhole, the spherical FOV of diameter d_{FOV} will project an image of diameter D_i onto the detector. Using simple geometry it can be found that

$$D_i = d_{FOV}\frac{d_{PD}}{(d_{OP}^2 - (d_{FOV}/2)^2)^{\frac{1}{2}}}. \tag{2.6}$$

From equations 2.2 - 2.6 it is clear that a compromise must be reached between the desired FOV, spatial resolution and sensitivity of a pinhole system. Consequently, it was suggested that multiple pinholes should be used to image the object (Vogel et al., 1978). The sensitivity scales with the number of pinholes that project the object onto the detector.

A disadvantage of pinhole imaging is that the magnification is different at all depths within the object. A pinhole-projection image is an integration of the slices through the object, and it can be difficult to interpret.

2.4.2 Multi-Pinhole Imaging

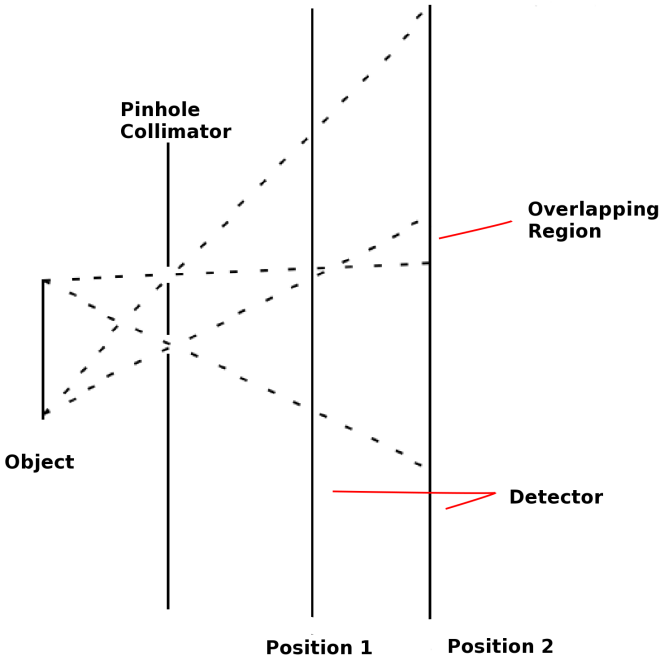


FIGURE 2.3: Basic multi-pinhole camera geometry. The amount of overlap depends on many factors, including pinhole spacing, detector distance, and FOV. The amount of overlap will be proportional to the magnification of the system. This schematic shows the entire object being imaged by both pinholes. SPECT imaging systems usually operate with the detector in position 1, with little or no projection overlap.

Typical multi-pinhole systems gather non-overlapping projection data from several pinholes onto one or more detectors (Beekman et al., 2005; DiFilippo, 2008). Each pinhole typically images either the entire FOV or a small section of the FOV (Schramm et al., 2003). Photons detected in overlapping regions contain less information than counts in non-overlapping regions due to the ambiguity of their origin (Barrett and Swindell, 1981; Defrise and Gullberg, 2006). Large amounts of projection overlap (multiplexing) in the projection images can result in artefacts in the reconstruction (Nuyts et al., 2009). A basic diagram of the geometry of a multi-pinhole collimator with overlapping projections is given in figure 2.3. Mok et al., 2009 investigated the trade-off between image degradation

and improved detection efficiency and resolution from allowing multiplexing in multi-pinhole SPECT. The authors concluded that the gains in improved detection efficiency and resolution by increased multiplexing are offset by increased image degradations. Currently there is much interest in developing imaging systems that make use of overlapping multi-pinhole projections but do not suffer from artefacts (Mahmood et al., 2010; Peterson and Furenlid, 2011; Schramm et al., 2003; Vunckx et al., 2008a). Using statistical reconstruction algorithms it is possible to maximize the information contained within each count in the overlapping regions (Mahmood et al., 2010; Wilson et al., 2000). Optimizing a multi-pinhole system is a complicated process and depends on the imaging task (Cao et al., 2005; Mok et al., 2005; Vunckx et al., 2008b). Some of the parameters to consider when designing a multi-pinhole collimator are

- System resolution.
- Sensitivity.
- Field of view.
- Pinhole diameter.
- Pinhole type.
- Collimator material.
- Pinhole opening angle.
- Detector size.
- Object-pinhole distance.
- Pinhole-detector distance.
- Number of pinholes.
- Inter-pinhole distance.
- Pinhole pattern.

A compromise must be reached between the parameters in the above list depending on the imaging task. These are discussed in more detail in section 3.6. Pinhole collimation is generally advantageous for high-resolution, high-sensitivity

imaging of a small field of view, but the sensitivity of the pinhole reduces rapidly as the object-pinhole distance is increased (Smith et al., 2003). Without a reconstruction algorithm, the multi-pinhole projection image is difficult to interpret, especially with multiplexing. For some applications, a parallel-hole collimator may be more suitable.

2.4.3 Parallel-Hole Imaging

An ideal, physically unrealizable, parallel-hole collimator would acquire a 2D projection of a 3D object with spatial resolution and sensitivity independent of position in the object (Barrett and Hunter, 2005). Each detector element placed directly behind a bore of this ideal collimator would be sensitive to radiation being emitted from anywhere in a tube-like region of space formed by extending the bore into the object and insensitive to radiation emanating from elsewhere (figure 2.4). Each detector would then measure the integral of the signal within the tube region associated with it. This collection of tube integrals forms the ideal planar projection. The properties of parallel-hole collimators are well es-

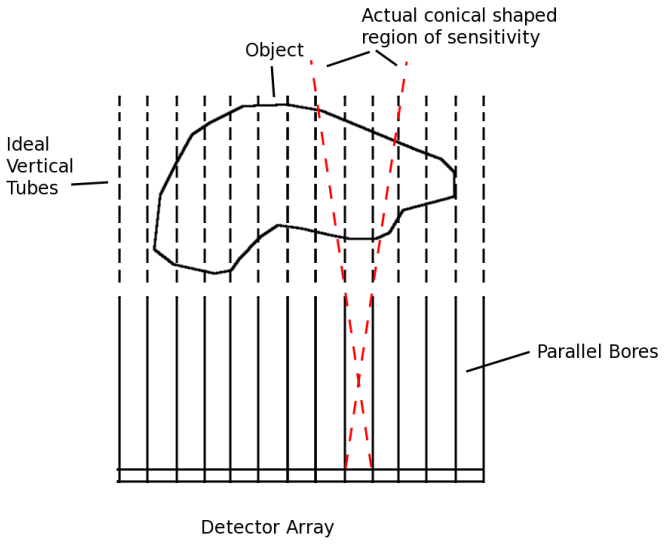


FIGURE 2.4: Parallel-hole collimator with ideal (black dash) and actual (red dash) sensitivity regions shown. Ideally each detector element measures the activity in a vertical tube above it, but it will actually measure the activity in a cone shaped region.

tablished (Gunter, 1996; Keller, 1967). The geometric sensitivity of a parallel-hole

collimator can be expressed as

$$\nu_{par} = \frac{Kw^2}{a_e(w+t)}, \tag{2.7}$$

where K is the collimator hole shape factor, w is the hole width, t is the septal thickness, and a_e is the effective hole length which is given by

$$a_e = a - \frac{2}{\mu}, \tag{2.8}$$

where a is the physical hole length, and μ is the attenuation coefficient of the material (Hubbell and Seltzer, 1995). The geometric resolution, R_g, defined as the full width at half maximum (FWHM) of the intensity distribution obtained from a point source, is given by

$$R_g = \frac{w(a_e + d_1 + d_2)}{a_e}, \tag{2.9}$$

where d_1 is the object to collimator distance, and d_2 is the collimator to detector distance. The total resolution of the system is

$$R_0 = [R_i^2 + R_g^2]^{0.5}, \tag{2.10}$$

where R_i is the intrinsic resolution of the detector. The spatial resolution may be improved by increasing the bore length and reducing the bore width. A trade-off exists between the spatial resolution and sensitivity of the system. For small-animal imaging, high resolution and high sensitivity are required. It is currently difficult and expensive to manufacture a high performance parallel-hole collimator with the required characteristics. One of the main advantages of the parallel-hole collimator is that the projections are easily interpreted.

2.4.4 Synthetic-Collimator Imaging

The objective of synthetic collimation is to synthesize an ideal parallel-projection image of a 3D activity distribution. To reconstruct an image from SPECT data, an object is usually decomposed into a three-dimensional grid of voxels. Using projection data from many different tomographic angles, the activity in each of the voxels may be estimated using a reconstruction algorithm. For synthetic

collimation, there is no rotation of the object or the system, and a single tomographic angle is used. The activities in the three-dimensional grid of voxels are estimated and are summed along the direction perpendicular to the detector face to synthesize a parallel-projection image.

The synthetic-collimator method uses a multi-pinhole collimator because of its capability for high sensitivity and high resolution. The resolution of a multi-pinhole collimator can be easily improved by decreasing the diameter of the pinholes or by increasing the magnification of the system. A multi-pinhole collimator is also very easily manufactured.

A major problem introduced with multi-pinhole collimators is multiplexing (Wagner et al., 1981). Multiplexing occurs when the projections through different pinholes overlap on the detector. The amount of multiplexing increases with magnification. The synthetic collimator collects multiplexed and non-multiplexed data and combines them to produce a high-resolution reconstruction (Wilson et al., 2000). The amount of projection overlap varies as a function of the system magnification when the object-to-pinhole distance (d_{OP}) is kept constant. Magnification is controlled by changing the pinhole-to-detector distance (d_{PD}). The data should contain information on how the projections are multiplexed and on how to remove the multiplexing effects. Each d_{OP}:d_{PD} combination is defined as a "system configuration" for this work. Data are acquired with several system configurations and combined to produce a non-multiplexed high-resolution reconstruction. There is currently much interest in developing systems that make use of multiplexed projections (Mahmood et al., 2011; Shokouhi et al., 2009).

The mathematical basis for the synthetic collimator was given by Clarkson et al., 1998, 1999, and it was demonstrated through simulations by Wilson et al., 2000. A short summary of the theory presented in these papers is given here.

The object is described by the function $f(r)$, where r is a three-dimensional position vector. The mth detector element records a measurement given by

$$g_m = \int_S d^3r\, f(r) h_m(r), \tag{2.11}$$

where S is the region of support of the object, and $h_m(r)$ is the sensitivity function that specifies the response of the detector element to radiation originating from point r. In this work, the parameters we are interested in are the ideal tube

integrals, given by

$$\Gamma_j = \int_S d^3r f(r) t_j(r), \tag{2.12}$$

where $t_j(r)$ is the tube function, defined as unity within the region of the jth vertical tube, and zero elsewhere. Each of the projection components represents the total activity due to $f(r)$ in the tube where $t_j(r) = 1$. If there are coefficients, B_{jm}, such that

$$t_j(r) = \sum_{m=1}^{M} B_{jm} h_m(r), \tag{2.13}$$

then the values of Γ_j should be recoverable. Assuming that the tube functions can be synthesized, the estimate for Γ_j is

$$\hat{\Gamma}_j = \sum_{m=1}^{M} B_{jm} g_m. \tag{2.14}$$

The difference between this estimator's expected value and the true value of the parameter is

$$\begin{aligned} \langle \hat{\Gamma}_j(g) - \Gamma_j(f) \rangle &= \sum_{m=1}^{M} B_{jm} \langle g_m \rangle - \Gamma_j(f) \\ &= \Big[\int_S \sum_{m=1}^{M} d^3r B_{jm} h_m(r) f(r) \Big] - \Gamma_j(f) \\ &= \Big[\int_S d^3r t_j(r) f(r) \Big] - \Gamma_j(f) \\ &= 0. \end{aligned} \tag{2.15}$$

Therefore an unbiased estimate for $\Gamma_j(f)$ exists and so it is an estimable parameter (Barrett, 1990b). Wilson et al., 2000 demonstrated that it was advantageous to estimate the activities in the voxels of a tube and to integrate the response, rather than estimating the tube activities in one step. These voxel estimates form an estimate of the 3D activity distribution. We define the voxel functions $v_n(r)$ for $n = 1, ..., N$ such that $v_n(r) = 1$ when r is in the nth voxel, and $v_n(r) = 0$ elsewhere. The corresponding parameters of the distribution are

$$\psi_n(f) = \int_S d^3r v_n(r) f(r), \tag{2.16}$$

and, similarly to before, these parameters are estimable if the voxel functions can be synthesized from the sensitivity functions

$$\psi_n(f) = \sum_{m=1}^{M} A_{jm} h_m(r), \tag{2.17}$$

where A_{jm} are coefficients that are assumed to exist. The unbiased estimator is given by

$$\hat{\psi}_n(g) = \sum_{m=1}^{M} A_{jm} g_m. \tag{2.18}$$

Maximum-likelihood estimation-maximization can be used to estimate the activity in the voxels (section 2.6). This is a non-linear algorithm, but the success of the synthesis still depends on the ability to represent the functions as a linear superposition of sensitivity functions.

It has been shown that changing the system parameters and taking more data will always improve the estimability of the projection integrals from the pinhole data (Clarkson et al., 1998). The system parameters that may be changed are:

- Number of pinholes,
- Location of pinholes,
- Number of object-pinhole, pinhole-detector pairs.

The most convenient way of taking more data is to increase the number of d_{OP}:d_{PD} pairs. The object-pinhole distance and/or the pinhole-detector distance may be changed. It is advantageous to keep the object-pinhole distance constant and only change the pinhole-detector distance. The object projection on the detector will then be a magnified (or demagnified) version of a previous projection. The first $d_{OP} : d_{PD}$ pair should result in no multiplexing on the detector. When combined with this initial data, any further data taken with higher magnification, and significant multiplexing, will contain information on how the projections overlap and should produce better estimates of the object distribution (Peterson et al., 2009; Wilson et al., 2000). Each $d_{OP} : d_{PD}$ combination is defined as a system configuration for this work. Data are acquired with several system configurations and combined to produce a non-multiplexed high-resolution reconstruction.

Depending on the system design constraints, such as collimator geometry and system magnification, a large-area detector can allow for a large number of non-multiplexed pinhole projections. The sensitivity and angular sampling are proportional to the number of pinhole projections acquired. Synthetic collimation requires that data are acquired using several system configurations. The only requirement of these configurations is that the system parameters are changed. The projection data need not consist of both non-multiplexed and multiplexed data, a series of system configurations that produce non-multiplexed data will suffice. An advantage of synthetic collimation is that multiplexed data can be used, thereby improving the efficiency of data acquisition. In the implementation discussed in this work the first system configuration acquires non-multiplexed projection data similar to a typical multi-pinhole system. Each subsequent system configuration will make more efficient use of the detector by acquiring multiplexed data that can be demultiplexed.

2.5 Digital Imaging Systems

The most general mathematical description of a linear, digital imaging system is a continuous-to-discrete mapping (Barrett and Myers, 2004; Chen et al., 2005)

$$g_m = \int_S d^3r f(r) h_m(r) + n_m, \tag{2.19}$$

where $m = 1, \ldots, M$, and $h_m(r)$ is the contribution to the mth measurement from the object at point r, and n is the noise. The measurements g_m form the elements of a vector $\boldsymbol{g}$ in an M-dimensional space which we call the data space. The continuous distribution of radioactivity in object space is denoted by f. This continuous f is approximated with a discrete $\mathbf{f}$ by voxelizing the object. The imaging equation is then written as

$$\mathbf{g} = \mathbf{H}\mathbf{f} + \mathbf{n}, \tag{2.20}$$

where $\mathbf{H}$ is the discrete system matrix that maps the object vector to an image, and the noise, $\mathbf{n}$, is an M-dimensional vector in data space. Iterative reconstruction algorithms require an accurate measurement of the system matrix for image

reconstruction. For pinhole SPECT, **H** should contain information about the system geometry, attenuation and scattering of radiation in the object, radiometric factors, pinhole penetration, and detector blur. **H** can be obtained analytically, with Monte Carlo simulations, or with experimental measurements.

From Barrett and Myers, 2004, a 3D object f(r) can be approximated by

$$f_a(r) = \sum_{n=1}^{N} \xi_n \phi_n(r), \tag{2.21}$$

where a denotes an approximation, n is a 3D location index in object space, and ϕ_n(r) is a basis function. This basis function is a voxel, a uniform cube centred on point $\mathbf{r}_n$. ξ_n are the expansion coefficients. The elements of **H** are expressed as

$$H_{mn} = \int_{\infty} d^3 r h_m(r) \phi_n(r), \tag{2.22}$$

where m is a 3D multi-index that specifies the projection angle and 2D location on the detector, and $h_m(r)$ represents the system sensitivity function. **H** can be thought of in terms of probabilities (Barrett and Myers, 2004). H_{mn} is the probability that a photon emitted from voxel n is detected in detector element m. The measurement of these probabilities is discussed further in chapter 4.

Using the equations above, a Fourier representation of a digital imaging system is given by (Barrett et al., 1996)

$$g = \sum_k F_k H\{\phi_k(r)\} + n = \Psi F + n. \tag{2.23}$$

Or, in component form,

$$g_m = \sum_k \Psi_{mk} F_k + n_m, \tag{2.24}$$

where

$$\Psi_{mk} = [H\{\phi_k(r)\}]_m = \int_S d^3 r exp(2\pi i p_k \cdot r) h_m(r), \tag{2.25}$$

and the Fourier coefficients are given by

$$F_k = \frac{1}{L^2} \int_{\infty} d^3 r f(r) exp(-2\pi i p_k \cdot r) s(r), \tag{2.26}$$

where k is a multi-index consisting of three integers k_1,k_2,k_3, and p_k has components $\frac{k_1}{L}$, $\frac{k_2}{L}$, $\frac{k_3}{L}$. The sum runs over the infinite three-dimensional lattice of points, $-\infty < k_i < \infty$, where k_i is an integer value. The complex matrix, Ψ, is equivalent to the operator $\boldsymbol{H}$. The latter maps the object to the discrete data set, the former maps the doubly infinite set of object Fourier coefficients F to g. Each Ψ_{mk} is the data vector produced by passing the basis function $\phi_k(r)$ through the imaging system. This can be used to calculate the Fourier crosstalk matrix (Barrett et al., 1995), whose elements are given by

$$\beta_{kk'} = \sum_{m=1}^{M} \Psi^*_{mk} \Psi_{mk'}, \tag{2.27}$$

where $k \neq k'$, $\beta_{kk'}$ is a measure of the degree of aliasing or crosstalk between the two different frequencies ρ_k and $\rho_{k'}$. If equation 2.27 is zero, then the Fourier components k and k' make linearly independent contributions to the data and can be easily separated by any inversion algorithm (Barrett et al., 1996). The degree of aliasing is defined as the angle between the two data vectors and is given by

$$|cos(\theta_{kk'})| = \frac{|\beta_{kk'}|}{\|H\{\phi_k(r)\}\|\|H\{\phi_{k'}(r)\}\|} = \frac{|\beta_{kk'}|}{\sqrt{\beta_{kk}\beta_{k'k'}}}. \tag{2.28}$$

The two data vectors are completely aliased if $|cos(\theta_{kk'})| = 1$ and the contributions of $\phi_k(r)$ and $\phi_{k'}(r)$ can not be resolved. If $k = k'$ we get

$$\beta_{kk} = \|H\{\phi_k(r)\}\|^2. \tag{2.29}$$

A diagonal element of the crosstalk matrix is a measure of the strength of the data when the object consists of a single Fourier component, $F_j = 1$ if $j = k$, and 0 otherwise. The set of diagonal elements β_{kk} constitutes a generalized transfer function of the system, while the off-diagonal elements give information about aliasing or linear dependence of different Fourier components (Barrett and Gifford, 1994). Further details on the Fourier crosstalk matrix are given in section 4.7.

2.6 Iterative Reconstruction Algorithms

In a radionuclide imaging system photons are emitted by the radioactive tracer and are detected using an external detector. These projection data may then be

used to mathematically reconstruct an image representing the radiotracer distribution within the object. If the physics of the system is modelled accurately then the probability that a photon emitted from voxel n is detected in detector element m is known. This probability contains information regarding the efficiency of the collimator, the attenuation in the object material, and the detection efficiency of the detector. Using this, the detector measurements can be coupled to the radiotracer distribution in the object. In principle, this is a simple problem to solve and requires a matrix inversion to determine the radiotracer distribution. Unfortunately this direct method is not possible in a real imaging for several reasons:

- The probability matrix is very large, but due to technological improvements this is becoming less of a problem.
- The detector values are not known exactly because of noise. There is a statistical uncertainty in the measurements in each detector element.
- Most of the probability matrix elements are zero due to the small number of detector elements recording photon emission from each voxel.

Knowledge of the system matrix, **H**, of a SPECT imager makes it possible to estimate the radiotracer distribution within an object from a set of detector measurements. It has been clear since the 1980s that iterative algorithms that model the Poisson statistics of the data achieve better image quality than filtered back projection algorithms (Defrise and Gullberg, 2006). FBP algorithms make no allowance for the physics of radioactive emission but gained widespread acceptance initially due to their computational speed. Iterative algorithms are becoming increasingly popular due to increased computational power, the development of accelerated methods, and the quality of the reconstructed images (Branderhorst et al., 2010; Hudson and Larkin, 1994; Shepp and Vardi, 1982). Iterative algorithms rely on an accurate system model which incorporates the collimator response function, the attenuation, and the scatter. It has been suggested that in the future SPECT may rely on a full Monte Carlo based reconstruction (Defrise and Gullberg, 2006). Iterative methods work by making an initial estimate of the activity distribution and improving on this estimate with each iteration. The estimate is projected through the system and compared with the measured data. Correction terms are then calculated and are applied to the activity estimate.

The estimate should become increasingly consistent with the measured data with each iteration. The various algorithms differ in the way the correction terms are determined.

There are many different iterative reconstruction algorithms available to reconstruct tomographic data (Bruyant, 2002; Defrise and Gullberg, 2006; Hutton et al., 1997; Qi and Leahy, 2006). The most common are: the algebraic reconstruction technique (ART) (Gordon et al., 1970), simultaneous algebraic reconstruction technique (SART), simultaneous iterative reconstruction technique (SIRT), maximum-likelihood expectation-maximization (MLEM) (Shepp and Vardi, 1982), and maximum a posteriori (MAP) (Green, 1990). The MLEM algorithm is the reconstruction algorithm that was chosen for this work. This algorithm maximizes a likelihood function, and is discussed extensively in section 2.6.2. It is often useful to constrain the possible solutions of an algorithm to ones that are acceptable. The positivity constraint is applied to the MLEM algorithm by providing a positive estimate of the activity distribution prior to the first iteration. The MLEM algorithm is a multiplicative algorithm and its estimation of voxel activity will remain non-negative. This is a reasonable constraint as it is not possible for an activity distribution to have a negative value.

Several methods have been proposed to experimentally measure the system matrix (Beque et al., 2003; Defrise et al., 2008; Furenlid et al., 2004; Have et al., 2008; Miller et al., 2012b; Rowe et al., 1993). The most comprehensive method is to exhaustively measure the response of the system to a point source positioned at the centre of each voxel. The detector response to a point source is known as the point spread function (PSF). The system matrix, $\mathbf{H}$, is a concatenation of the PSFs for every voxel in the field of view. The resolution of the reconstructed image can be no finer than the sampling of the object space. It is very time consuming to experimentally measure a high-resolution system matrix. It may take several seconds to record a projection image and then translate the point source to the next voxel location. High-resolution small-animal SPECT systems typically require voxel sizes on the order of 0.5 mm or smaller. Experimentally measuring a system matrix with this resolution for any appreciable FOV could be prohibitively time consuming. The SPECT system discussed in this work, SyntheticSPECT, was developed to implement synthetic collimation for image formation and reconstruction. This collimation method requires several projection images, each taken at a different system magnification, for the reconstruction.

The high-magnification data are multiplexed, with several pinholes projecting the FOV onto the same detector element, while the low-magnification data are not multiplexed. Each system magnification is termed a "system configuration" because the magnification is controlled by adjusting the collimator-detector distance. For SyntheticSPECT, the system matrix needs to be determined for every system configuration, and experimental measurement of multiple finely-sampled system matrices is impractical.

Instead, the system matrix was experimentally measured by translating a point source through a sparsely-sampled cubic grid encompassing the FOV and measuring the system response at each voxel position (section 4.1). The MLEM (section 2.6.2), and pixel-ordered-subsets expectation-maximization (POSEM) algorithms (section 2.6.4) were investigated in this work. POSEM is an accelerated algorithm that is particularly suitable for multi-pinhole systems and which produces results similar to MLEM. In this work, the treatment of iterative reconstruction methods closely follows that given in Barrett and Myers, 2004.

2.6.1 Maximum-Likelihood Estimation

Measured data, which depend on several parameters and noise sources, can be described by a probability law, denoted by $pr(g|\vartheta)$ (Barrett and Myers, 2004). ϑ is a P x 1 parameter vector, and $\mathbf{g}$ is denoted as a M x 1 vector describing random data, where g is an individual sample taken from $pr(g|\vartheta)$. The likelihood function of the parameters to be estimated given a measured data vector is defined as

$$L(\vartheta|g) = pr(g|\vartheta). \tag{2.30}$$

The likelihood is used to obtain estimates of the parameters in ϑ, denoted as $\hat{\vartheta}$. A value of θ that maximizes the probability of the observed g is sought. This technique is known as maximum-likelihood (ML) estimation and is defined by:

$$\hat{\vartheta}_{ML} \equiv \arg\max_{\vartheta} pr(g|\vartheta). \tag{2.31}$$

The *argmax* function returns the ϑ argument at which $pr(g|\vartheta)$ is maximized. This can be written equivalently as

$$\hat{\vartheta}_{ML} = \arg\max_{\vartheta} \ln[pr(g|\vartheta)]. \tag{2.32}$$

2.6.2 Maximum-Likelihood Expectation-Maximization (MLEM)

A ML estimate may be calculated using an iterative technique known as expectation-maximization (EM). It is so called because it can be derived by alternating expectation (E) and maximization (M) steps. Shepp and Vardi, 1982 first demonstrated this method for tomographic applications. The MLEM iterative formula is derived as given in Barrett and Myers, 2004. Each detector element is independent, and the number of photons detected in each detector element obeys Poisson statistics, therefore the probability law for the data vector g is given by

$$Pr(g|\vartheta) = \prod_{m=1}^{M} \exp(-\bar{g}_m)\frac{(\bar{g}_m)^{g_m}}{g_m!}, \tag{2.33}$$

where the mean data vector is

$$\bar{g} = H\vartheta. \tag{2.34}$$

The log-likelihood function is

$$\ln[Pr(g|\vartheta)] = \sum_{m=1}^{M}[-(H\vartheta)_m + g_m \ln(H\vartheta)_m - \ln(g_m!)]. \tag{2.35}$$

This likelihood function represents the probabilistic likelihood that the measured data (g_m) is statistically equivalent to the estimated detector data $\bar{g}$. The extremum of this function occurs at a point where the first derivatives are zero:

$$\frac{\partial}{\partial\vartheta_n}\ln[Pr(g|\vartheta)] = \sum_{m=1}^{M}\left[-H_{mn} + g_m\frac{H_{mn}}{(H\vartheta)_m}\right] = 0, n = 1, 2, ..., N. \tag{2.36}$$

The second derivatives are

$$\frac{\partial^2}{\partial\vartheta_n\partial\vartheta_k}\ln[Pr(g|\vartheta)] = \sum_{m=1}^{M}\left[-g_m\frac{H_{mn}H_{mk}}{[(H\vartheta)_m]^2}\right]. \tag{2.37}$$

The components of **g** and **H** are non-negative, therefore the second derivatives of the likelihood function are negative everywhere. Any extremum must therefore be a maximum. Equation 2.36 can be rearranged to

$$\frac{1}{\sum_{m=1}^{M} H_{mn}}\sum_{m=1}^{M}\frac{g_m}{(H\vartheta)_m}H_{mn} = 1. \tag{2.38}$$

Both sides are now multiplied by ϑ_n, which we assume is the radiotracer distribution that is the best estimate of the true radiotracer distribution within the object, yielding

$$\vartheta_n = \frac{\vartheta_n}{\sum_{m=1}^{M} H_{mn}} \sum_{m=1}^{M} \frac{g_m}{(H\vartheta)_m} H_{mn} \tag{2.39}$$

To get an iterative algorithm, ϑ is replaced by a succession of estimates $\hat{\vartheta}^{(k)}$, and a fixed-point iteration procedure is used. The MLEM algorithm is given by

$$\hat{\vartheta}_n^{(k+1)} = \frac{\hat{\vartheta}_n^{(k)}}{\sum_{m'=1}^{M} H_{m'n}} \sum_{m=1}^{M} \frac{g_m}{\sum_{n'=1}^{N} H_{mn'} \hat{\vartheta}_{n'}^{(k)}} H_{mn}, \tag{2.40}$$

where k denotes the kth iterative estimate of the distribution.

2.6.3 Image Reconstruction with MLEM

The objective of image reconstruction is to estimate the radiotracer activity within each voxel of the object. The ϑ term in equation 2.40 is now replaced by f, the activity of the distribution. The activity within a voxel is assumed to be uniform. The MLEM algorithm can be rewritten as

$$\hat{f}_n^{(k+1)} = \frac{\hat{f}_n^{(k)}}{\sum_{m'=1}^{M} h_{m'n}} \sum_{m=1}^{M} \frac{g_m}{\sum_{n'=1}^{N} h_{mn'} \hat{f}_{n'}^{(k)} + s_m} h_{mn}, \tag{2.41}$$

where $\hat{f}_n^{(k)}$ is the estimated activity in voxel n after the kth iteration, h_{mn} is an element of $\mathbf{H}$ that represents the probability that a photon emitted from voxel n is detected in detector element m, and s is the noise. The MLEM algorithm may be separated into a number of operations to allow for efficient computation. The sensitivity function ($\sum_{m'=1}^{M} h_{m'n}$) needs to be calculated once. This is done by integrating the detector response for each voxel over all the detector elements.

The next step requires that the current estimate for the voxel activity is projected using the measured $\mathbf{H}$ matrix. The measured data, $\mathbf{g}$, are divided element-by-element by the projection data. These correction factors are then back-projected through the system. This value is then multiplied by the current voxel activity estimate and divided by the sensitivity function to obtain a new

estimate of the activity in the voxel. These steps are repeated for a fixed number of iterations. The reconstructed images obtained with the MLEM algorithm can become noisy as the number of iterations increases. This is because noisy reconstructions may yield projections that are close to the measured projection images. Prior information, based on an assumption of the true activity distribution, is sometimes used to penalize noisy images. This regularization can enforce a smoothing condition on the reconstruction algorithm. The MAPEM algorithm (Green, 1990) is one such algorithm that employs regularization. In this work we will make no assumption on the activity distribution. The final reconstructed image will be post-smoothed with a Gaussian filter.

An attempt was made to estimate the contribution of scatter to the projection data. After the pinhole projections had been located and masked from the PSF data, the detector response was analyzed for other detected photons (section 4). Typically, most of the detections due to scatter are removed by energy filtering, but this technique is not applied here due to poor energy resolution of the system. This scatter response was assumed to be Poisson-distributed signal from a point source at a location in the FOV. The mean pixel value for each projection image, after the pinhole projections were masked, was used in the reconstruction step.

2.6.4 Ordered Subsets Expectation-Maximization

While MLEM is considered to be the gold standard of iterative reconstruction methods, its drawback is that it is very computationally intensive. Block-iterative algorithms, such as the ordered subset algorithm (OSEM)(Hudson and Larkin, 1994), process data in subsets (blocks) within each iteration which effectively accelerates convergence by a factor proportional to the number of subsets.The standard EM algorithm is applied to each subset in turn with the result being used as the starting value for the next iteration. The OSEM algorithm is

$$\hat{f}_n^{(i+1)}(k) = \frac{\hat{f}_n^{(i)}(k)}{\sum\limits_{m' \in S_i} h_{m'n}} \sum_{m \in S_i} \frac{g_m}{\sum\limits_{n'=1}^{N} h_{mn'} \hat{f}_{n'}^{(i)}(k)} h_{mn}, \tag{2.42}$$

where $\hat{f}_n^{(i+1)}(k)$ represents the image update after processing subset i, S_i contains the projections of subset i, and k represents the iteration number. To get OSEM solutions that are close to MLEM solutions it is important to preserve

subset balance, which means that the voxel activity should contribute approximately equally to all subsets (Hudson and Larkin, 1994). With multi-pinhole SPECT systems it is difficult to achieve subset balance due to voxel-dependent sensitivities. Branderhorst et al., 2010 proposed a pixel-based subset (POSEM) approach to subset selection which is especially suited to multi-pinhole systems. The authors have shown that this method allows for very high acceleration fac-

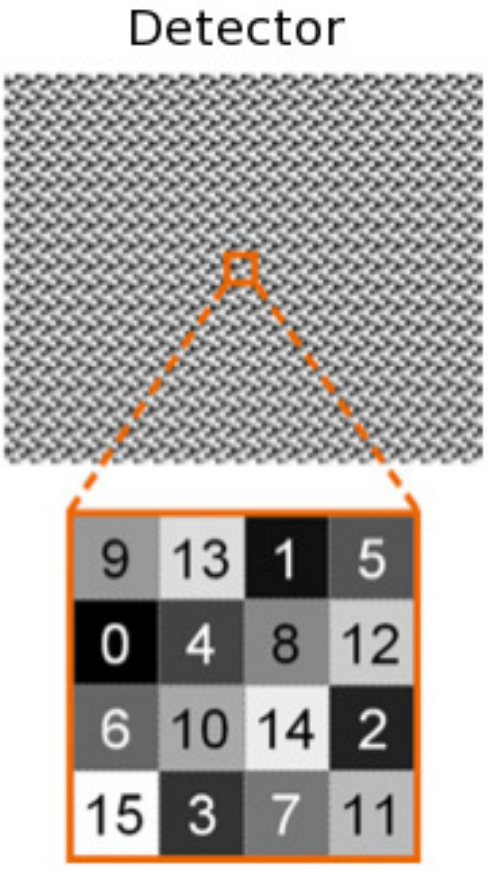


FIGURE 2.5: Example subset distribution for POSEM. Pixels having the same gray shade belong to the same subset. A detail of the repeating pattern used to create 16 POSEM subsets is magnified. This pattern can be replicated across several detectors or projections. Example: The first subset will include the pixels on each detector or projection that are labelled with '0' in the pattern. (Reproduced from Branderhorst et al., 2010.)

tors while producing results similar to MLEM. In POSEM, all detector pixels are distributed into subsets independently of the projections. Detector pixels are assigned to subsets according to a pattern which is repeated over the detector surface at each system configuration (figure 2.5). A sixteen-subset pattern (P_{16})

was chosen so that subsequent subsets were well separated.

$$\mathbf{P_{16}} = \begin{pmatrix} 9 & 13 & 1 & 5 \\ 0 & 4 & 8 & 12 \\ 6 & 10 & 14 & 2 \\ 15 & 3 & 7 & 11 \end{pmatrix}$$

In a typical pinhole system the object-pinhole distance fully determines the sensitivity of the system. In SyntheticSPECT the detector is moved to several positions behind the collimator, thus increasing the magnification of the system. As the magnification increases, the solid angle subtended by the detector and the number of pinhole projections that are detected are reduced, effectively reducing the sensitivity of the system. Therefore, the different system configurations should not be used as subsets due to the reduction in number of pinhole projections as the magnification of the system is increased. The POSEM algorithm was implemented because the subsets are balanced due to their independence of system configuration.

2.6.5 Implementation with CUDA

The parallel computing architecture, Compute Unified Device Architecture (CUDA), from NVIDIA, was used to implement the MLEM and POSEM algorithms on the graphics processing units. In order to explain CUDA, some definitions are required first:

- Kernel: a function that runs simultaneously in many threads.
- Thread: a data element to be processed. These are organized into grids and blocks.
- Block: a group of threads which can communicate and share data with each other.
- Grid: a group of blocks. There is no communication between blocks.

Data are typically stored in the hard drive or the random-access memory (RAM); this is known as host memory. Data must be passed to device memory in order to be processed on the GPU. The GPU used in this work had 3 GB of global

memory available, and the host had 16 GB of RAM. Each block had 48 kB of shared memory available to it during kernel execution. Accessing data in shared memory is hundreds of times faster than accessing data from device global memory. The specifications of the GPU are given in table 2.4.

Table 2.4: Specifications of the CUDA-capable GPU used in this work.

Device Name	GeForce GTX 580
Core Clock	772 MHz
Memory Clock	4008 MHz
Shader Clock	1544 MHz
CUDA Cores	512
Global Memory	3072 MB DDR5
Shared Memory	48 kB
No. of Multiprocessors	16
Maximum Threads per Block	1024

The system matrix was stored in a large, linear array. The coefficients were first loaded in host memory, and were then sorted and indexed for rapid access. The coefficients for a single system configuration were loaded into GPU memory as needed and were then discarded. The number of memory transfers should be kept to a minimum when using CUDA. Although it is feasible, under certain circumstances, to load the coefficients for all the system configurations at once, it was decided to load them on an individual basis to make the algorithm as general as possible and to allow for incorporation of more data sets in the future.

The output of the intensifier was only imaged by approximately 20000 pixels out of the total 76800 available. With CUDA, care must be taken that each thread writes to a different data element. Data will become corrupted if more than one thread updates an element concurrently. If each thread within a block needs to read the same data it is usually advantageous to load these data into the shared memory available to that block. There is only a small amount of shared memory available to each microprocessor but it can be accessed several hundred times faster than the much larger global memory.

2.6.5.1 MLEM

In the MLEM implementation, each voxel to be reconstructed was assigned a thread. Each thread then generated the projection image for that voxel. Although

the memory usage was large, the execution time was small because each Gaussian fit needed to only access the pixels within a 3σ radius around the centroid. This method was used primarily for testing purposes and to allow for easy comparison of images to the raw data and diagnostic tests performed on MATLAB^{TM}. It was not an optimal algorithm due to number of instructions issued to each thread in a single kernel. Over six thousand voxels could be processed at a time with this method due to the memory limitations of the GPU. A flowchart outlining the implementation of the sensitivity calculation for the MLEM algorithm on CUDA is shown in figure 2.6. If done in isolation, the sensitivity function could be calculated very rapidly. Unfortunately, the MLEM algorithm required several operations (multiplication by the current voxel estimate, addition of all the pixel values) on the images that were generated on-the-fly.

A more efficient, cluster-by-cluster implementation of the MLEM algorithm was also developed. With this method, each cluster, characterized by six coefficients, was executed on its own block of threads. The six coefficients were written to the shared memory of the block and the values in the pixels within the cluster were calculated simultaneously. The code for this implementation is given in section A.6. The coefficients are discussed in more detail in section 4.

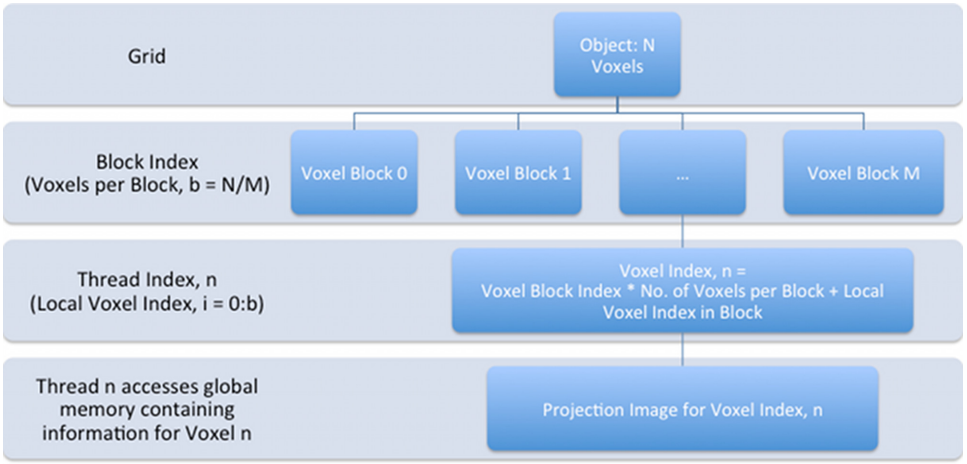


FIGURE 2.6: A flowchart describing the implementation of the MLEM algorithm with CUDA. An object consists of a set of N voxels. Each voxel contains information regarding the column of the H matrix associated with it. Each voxel is assigned a CUDA thread to process this information. There are M blocks, each containing N/M threads, that process these threads in parallel.

2.6.5.2 POSEM

In POSEM, the implementation was reversed. Each thread was assigned a pixel in the projection image subset and its response was calculated using the coefficients associated with each voxel. The implementation of POSEM used much less device memory than the MLEM implementation. To accelerate the kernel in which the pixel response was measured, the coefficients of the Gaussian fits of the pinhole projections for a given voxel were loaded into shared memory. It was found that the most efficient method to accomplish this was to ensure that the number of threads per block (table 2.4) was larger than the number of coefficients that needed to be loaded. This made each thread responsible for loading a single element of global memory to shared memory. In this work, each pinhole projection was modelled by a 2D Gaussian distribution which can be described using six coefficients. The number of threads per block should be $> 6\times$ the number of pinholes. From table 2.4, the maximum number of threads per block is 1024. In this work, the maximum number of pinhole projections from a single voxel was forty-six. The number of threads called in each block should therefore be > 276. Each thread can be instructed to load multiple coefficients if the number of coefficients exceeds the maximum number of threads. This implementation is therefore independent of collimator size and can be applied to any multi-pinhole system. Once all the coefficients for the pinhole projection fits for a particular voxel position were loaded, the response of each pixel was calculated using a Gaussian kernel. The indices of the pixels corresponding to the different subsets were stored in arrays in GPU memory, and each subset has $2 \times 10^4/16 \simeq 1250$ elements. Each of the pixels were accessed using an indexed list and were assigned a thread, and so the memory requirements were minimal, unlike the MLEM implementation which stored an entire projection image for each voxel position. Due to the nature of this method, the response of each pixel needed to be calculated against every pinhole projection. The pixel response was calculated if it was within approximately 3σ of the centroid of a pinhole projection.

2.6.6 CUDA vs MATLAB™

The advantages of using CUDA instead of a central processing unit (CPU) based implementation, such as MATLAB™, is investigated in this section. MATLAB™

is a matrix based computing environment and programming language for scientific calculations. It is capable of solving vectorized problems very rapidly, but calculations that require iterating over a set of commands, as is done in traditional programming languages (e.g. C/C++), can be time consuming. Recent versions of MATLAB^{TM} have incorporated CUDA capability, although this was not investigated in this work.

In the reconstruction steps outlined earlier the most common operation was to find the sensitivity of a detector pixel to a source emitting from a voxel in object space. These data were generated and discarded twice in each iteration of the reconstruction algorithm for every voxel location. The total sensitivity of the detector to a uniform activity at every point within the FOV was calculated with MATLAB^{TM} and CUDA using the interpolated system matrix (section 4). The algorithm for both implementations was:

- Read in the Gaussian coefficients of the pinhole projections for each voxel (section 4.2).
- Calculate pixel response within a 3σ radius of the centroid of each pinhole fit.
- Integrate the detector response from each voxel location.

The MATLAB^{TM} implementation was vectorized as much as possible, but loops were required due to the memory limitations of the system. The results of the two implementations are shown in table 2.5. It was found that CUDA implementation was much faster than the MATLAB^{TM} implementation for this task. Any manipulation of the generated data would significantly increase the execution time of the CUDA, but this is not investigated here.

Table 2.5: Comparison of algorithm (section 2.6.6) execution times between MATLAB^{TM} and CUDA.

Implementation	# voxels in object space	Time (s)
MATLAB^{TM}	$61 \times 61 \times 61$	387
MATLAB^{TM}	$97 \times 97 \times 97$	1608
CUDA	$61 \times 61 \times 61$	< 0.01
CUDA	$97 \times 97 \times 97$	< 0.01

In this work, CUDA was used for the frame-parsing algorithm and the MLEM reconstruction algorithm. This was chosen because the operations needed to

be performed very quickly and repeated many times. MATLAB™ was used for operations that needed to be performed only once and required much user interaction (e.g., pinhole tracking).

CHAPTER 3

System Design

3.1 Introduction

Two imaging systems were used in the course of this work:

- BazookaSPECT
- SyntheticSPECT

The BazookaSPECT detector was originally developed by Dr. Brian Miller at the Center for Gamma-Ray Imaging (CGRI), University of Arizona. Dr. Miller developed a library of functions that interfaced LabVIEW with the CCD camera. This library was used for image acquisition with BazookaSPECT and, with minor changes by the author, SyntheticSPECT.

The BazookaSPECT was given its title originally because it consisted of a long optical system, but subsequent versions have reduced in size (Miller et al., 2006). The BazookaSPECT detector consists of a scintillator, image intensifier, optical lens, and fast-frame-rate CCD camera (Miller et al., 2006, 2008). Data are transferred to the processing computer via a firewire interface, and the use of graphics processing units enables image processing in real time. The BazookaSPECT collimator has a single 1.0 mm diameter keel-edge pinhole (figure 3.1). A single-channel BazookaSPECT detector was used for work carried

(A) (B)

FIGURE 3.1: Images of the BazookaSPECT and the single-pinhole collimator. The BazookaSPECT detector consists of a scintillator, image intensifier, optical lens, and CCD camera. The collimator consists of a 1 mm tungsten pinhole inserted into a lead-lined container.

out in collaboration with the Regenerative Medicine Institute (REMEDI), National University of Ireland, Galway. Although the detector was developed by our collaborators at CGRI, the experiments conducted at REMEDI were designed by the author, details of which are given in later chapters. A thorough background on the BazookaSPECT detector is given in this chapter because it is the detector used for the work at REMEDI. The detector was also used in the SyntheticSPECT system, the system designed and developed by the author.

Section 3.2 describes how gamma rays were detected using the scintillator, image intensifier and CCD camera. Section 3.3 explains how these data were then used to estimate the position of interaction of the gamma ray with the detector. These two steps are common to both the BazookaSPECT and SyntheticSPECT systems.

The focus of this chapter is the design of the SyntheticSPECT system. This system was developed by the author during a study period at the Center for Gamma-Ray Imaging, University of Arizona, Tucson, AZ, USA. The SyntheticSPECT system consists of a multi-pinhole collimator and a BazookaSPECT gamma-ray detector coupled to a fibre-optic taper. As described in Chapter 2, a large-area detector is beneficial to the implementation of synthetic collimation. Development of this system required the fabrication of a novel multi-pinhole collimator to take full advantage of the large-area BazookaSPECT detector. The benefits and challenges of using a fibre-optic taper to increase the detector area are discussed in section 3.4. Development of this system relied heavily on rapid-prototyping using computer-aided design software and 3D printing technology,

and casting using a tungsten epoxy. A novel method for fabricating imaging apertures was developed by our collaborators at CGRI (Miller et al., 2011). The fabrication method used in the work presented here is slightly different, and it is discussed in section 3.5. Section 3.6 describes design and fabrication of the focused multi-pinhole collimator. The design and construction of this collimator is the principal novel contribution of the work outlined in this chapter. This collimator was designed to maximize the sensitivity and angular sampling of the system, while only using a single tomographic angle. Section 3.7 describes the fabrication of the other support structures required for this imaging system. Section 3.8 explains how data were acquired using the SyntheticSPECT imaging system.

3.2 Gamma Camera

A new generation of high-resolution single-photon counting scintillation detectors has recently been developed (Miller et al., 2007, 2008). These detectors use a CCD sensor to achieve high resolution and large space-bandwidth products (Barrett and Hunter, 2005). Detection of a single photon begins with the conversion of a gamma ray to many optical photons in a process called scintillation (Knoll, 1979). The photo-electric effect and Compton scattering produce high-energy electrons in the photon interaction. These excite many secondary ionizations that migrate to luminescent regions of the scintillation crystal, and optical photons are emitted as they decay. Several scintillation materials, both monolithic and structured, can be used with these detectors, but columnar CsI(Tl) and Kodak Lanex scintillators were used for this work. The scintillator was directly coupled to the photocathode of the intensifier for BazookaSPECT work. In SyntheticSPECT, a large-area fibre-optic taper was placed in between the scintillator and intensifier.

Columnar scintillators channel the scintillation light down small columns towards the exit face while limiting the lateral spread of light (figure 3.2). The scintillation light spreads to neighbouring fibres and is attenuated by an amount relating to the depth-of-interaction within the crystal. This light is then subject to amplification through an image intensifier (figure 3.3) before projection onto the CCD sensor.

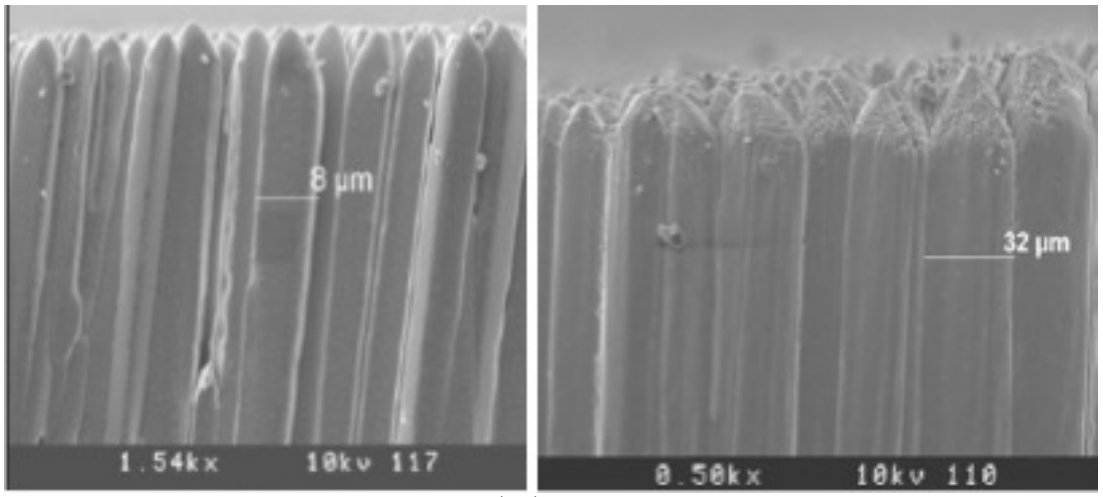


FIGURE 3.2: Two columnar CsI(Tl) crystals showing the structure of the scintillator. The thickness of the scintillator can range from 30 μm to 2000 μm. (Reproduced from Nagarkar et al., 1998.)

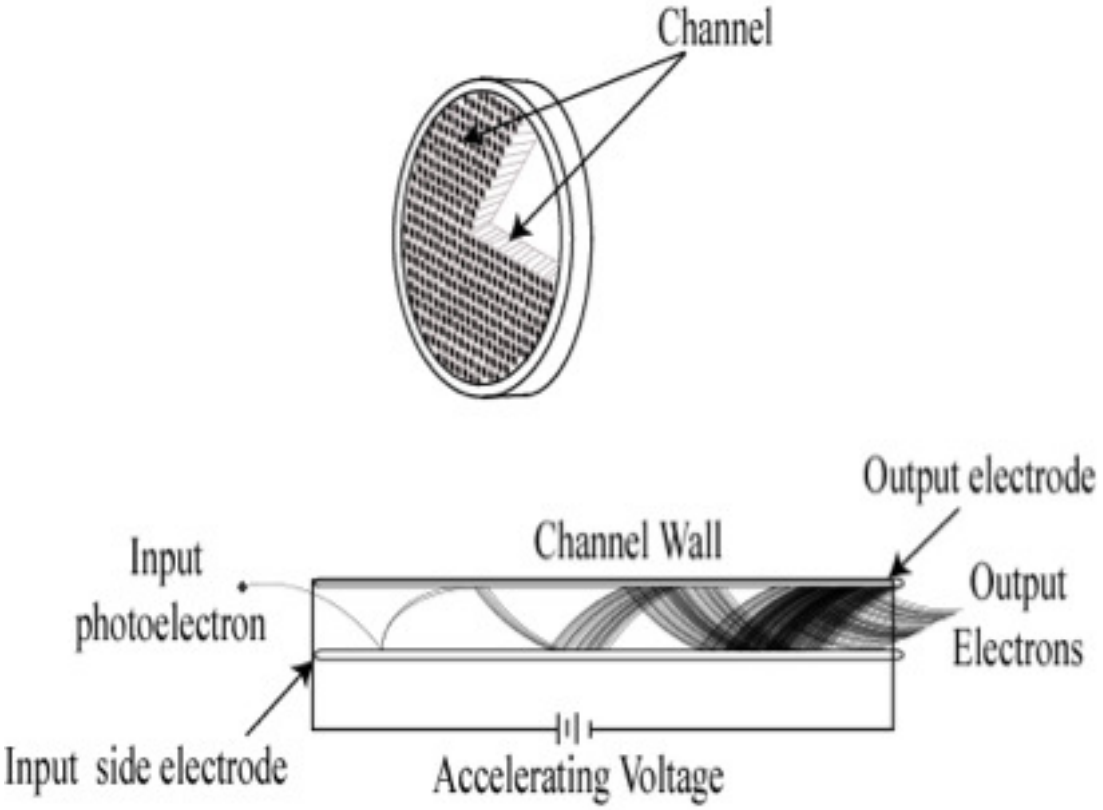


FIGURE 3.3: Microchannel plate (MCP) gain (10^4 - 10^7). Incident photoelectrons interact with the wall of the MCP which eventually leads to a cascade of electron emissions that are accelerated towards an output phosphor screen and converted to visible photons.

Non-amplifying detectors have been shown to be capable of identifying individual photon interactions, but the sensors required for this work had very low frame rates (Miller et al., 2007). BazookaSPECT detectors use a micro-channel

plate (MCP) image intensifier to provide up-front optical gain (Wiza, 1979). This amplification minimizes the effect of light-loss from imaging the intensifier output screen onto the CCD sensor. A micro-channel plate is a fibre-optic faceplate with etched-away fibre cores that have a relatively uniform resistance per unit length down the channel. Incident photoelectrons interact with the photocathode at the MCP entrance producing an electron cascade which is accelerated towards the output phosphor screen using a potential difference placed across the channel (figure 3.3).

3.2.1 XX1332 Image Intensifier

The BazookaSPECT system at NUIG used the Photonis XX1332 image intensifier (figure 3.4). This intensifier had a 50 mm diameter plano-concave fibre-optic window with an S-25 photocathode deposited on the curved surface. The tube uses electro-optical focusing to direct photoelectrons generated at the photocathode to the 40 mm diameter output MCP. After amplification at the MCP, the electrons are converted to optical photons by a P20 phosphor. There are several different phosphors suitable for BazookaSPECT detector (table 3.1). The CCD should be well matched to the output spectrum of the phosphor.

Table 3.1: Some phosphor screens that may be used for BazookaSPECT imaging (*Proxivision, Inc.* Wang and Blackburn, 2000).

Type	Composition	Light emission				Decay time	
		Range (nm)		Max (nm)	Colour	Decay of light intensity	
		From	To			From 90 % to to 10% in	From 10 % to 1% in
P20	ZnCdS:Ag	460	660	530	green		1 - 100 ms
P43	Gd_2O_2S:Tb	360	680	545	green	1 ms	1.6 ms
P46	$Y_3Al_5O_{12}$:Ce	490	620	530	yellow green	300 ns	90 μs
P47	Y_2SiO_5:Ce,Tb	370	480	400	blue white	100 ns	2.9 μs

3.2.2 Imaging Lens

A single Fujinon 6 mm, F/1.2 lens was used in both imaging systems to image the intensifier output onto the CCD. The specifications of this lens are given in table 3.2. The spacer rings attached to the lens were adjusted so that the output image from the image intensifier fills as much of the CCD sensor as possible.

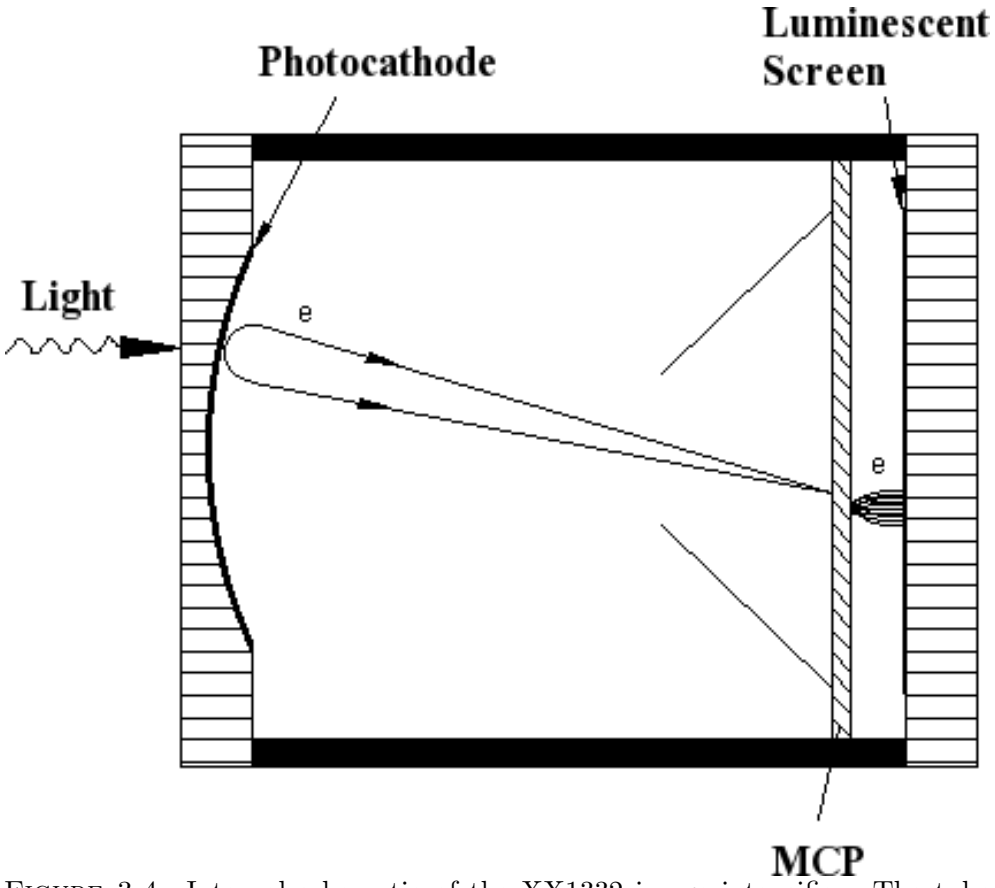


FIGURE 3.4: Internal schematic of the XX1332 image intensifier. The tube uses electro-optical focusing to direct photoelectrons generated at the photocathode to the 40 mm diameter output MCP. After amplification at the MCP the electrons are converted to optical photons by a P20 phosphor. (Reproduced from *Abex (UK).*)

TABLE 3.2: Specifications of the lens used to focus the output of the image intensifier onto the CCD sensor.

Lens	Fujinon DF6HA-1B
Focal length	6 mm
Iris range	F1.2 - F16
Focus & iris operation	Manual
Focusing range	$\infty \sim 0.1$ m
Back focal distance	11.44 mm
Exit pupil position	-46
Filter thread	M27 $\times$ 0.5
Mount	C
Mass	55 g

3.2.3 CCD Camera

For SyntheticSPECT, a Grasshopper Express CCD camera from Point Grey Research (*Point Grey Research, Inc.*) was used (figure 3.5), although nearly any

CCD/CMOS sensor would be suitable due to the optical gain provided by the MCP. A Dragonfly Express CCD camera was used for the BazookaSPECT system at NUIG (table 3.3). These CCDs were used due to their low cost and their ability to operate at room temperature.

FIGURE 3.5: Low-cost CCD detector: Grasshopper Express CCD from Point Grey Research, Inc. It has 1024 × 1024 5.5 μm pixels, and operates at up to 200 fps.

TABLE 3.3: Specifications of the CCD cameras used in this work.

Device Name	Grasshopper Express	Dragonfly Express
Sensor	Kodak KAI-01050 CCD 1/2", 5.5 μm	Kodak KAI-0340DM/C 1/3", 7.4 μm
Shutter	Global	Global
Resolution	1024 × 1024	640 × 480
Gain	24 dB	30 dB
Digital Interface	IEEE-1394-b	IEEE-1394-b
Transfer Rate	100 MB/s	100 MB/s
Dimensions	44 mm × 29 mm × 58 mm	63.5 mm × 50 mm × 13 mm
Camera Control	FlyCapture SDK	FlyCapture SDK

This camera should be able to operate at up to 200 frames per second (fps) with 640 x 480 x 8-bit images, or up to ∼350 fps with 320 x 240 x 8-bit images. It was observed that significant frame-tearing occurred at the highest frame rates. Frame-tearing is the corruption of data between image acquisitions; the captured frame will appear as a combination of several frames overlapping in different locations. This frame-tearing resulted in incorrect identification of cluster positions in the affected frames. The solution to this problem was to operate the CCD camera at much lower frame rates. It was found that operating at 200 fps with 320 x 240 x 8-bit images resulted in no image tearing. It is thought that insufficient memory bandwidth on the firewire bus led to these errors, although there are also many other possibilities (*Point Grey Research, Inc.*). The exit face of the 25 mm diameter dual-MCP used in SyntheticSPECT is demagnified eight times to be imaged on to the CCD sensor. The detector used at NUIG used a single-MCP, with a 50 mm diameter entrance face and a 40 mm diameter exit face, which is also demagnified.

The maximum frame rate achievable, without frame-tearing, corresponds to an exposure time of 5 ms per frame. A reduction in frame rate increases the exposure time per frame, but a reduction in exposure time does not correspond to an increase in frame rate. The CCD was read every 5 ms, but depending on the probability of cluster overlap, the shutter may be open for only a fraction of this time. Effectively, there may be a dead-time during which data were not acquired.

3.3 Photon Counting with BazookaSPECT

A CCD sensor works by integrating charge produced by optical photons over a period of time. For photon-counting the sensor is operated at a high frame rate, or short exposure time, so that the optical photons associated with each gamma-ray interaction have a low probability of overlapping. Each photon interaction appears as a signal spread over a small pixel region, which is called a cluster (figure 3.6). These data were processed in real time using a GPU and the listmode data was stored in a file to allow further processing

These pixel data can be used to estimate the gamma-ray interaction location (Barrett et al., 2009; Miller et al., 2009). This is referred to as frame parsing.

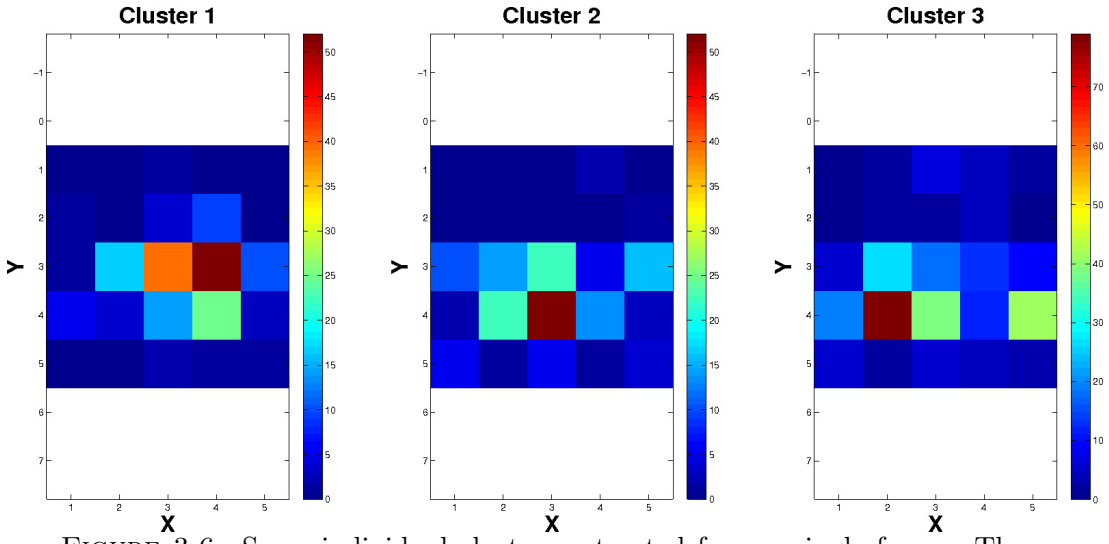


FIGURE 3.6: Some individual clusters extracted from a single frame. These data were extracted from 8 bit images with 320 x 240 pixels. Each cluster is 5 x 5 pixels at this resolution. The algorithm may be modified to increase the cluster size if required. These data were processed in real time using a GPU and the listmode data was stored in a file to allow further processing.

Apart from minor changes, the image acquisition and frame-parsing software was developed by our collaborators at CGRI. The frame parsing algorithm used in this work is:

1. Acquire a CCD frame.
2. Apply median filter to remove hot/noisy pixels.
3. Apply threshold to the filtered image.
4. Identify individual clusters using a fast, connected-components-labelling algorithm (Suzuki et al., 2000).
5. Extract pixels in each cluster.
6. Estimate interaction position using maximum-likelihood estimation (Barrett et al., 2009), or by calculating the centroid of the pixel distribution.

This algorithm was implemented using parallel processing techniques on graphics processing units (figure 3.7). The intrinsic resolution of the BazookaSPECT detector is ∼100 μm when measured with 140 keV photons and a Lanex scintillator (Miller et al., 2006). The energy resolution is known to be quite poor and the count rate is determined by frame-rate of the CCD and properties of the scintillator (Furenlid et al., 2000; Miller et al., 2006).

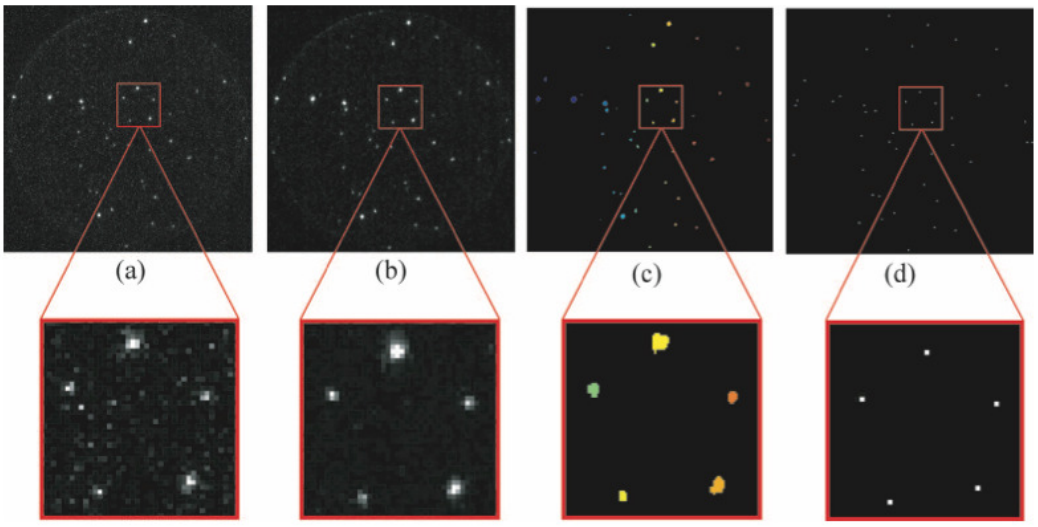


FIGURE 3.7: Frame-parsing algorithm implemented on an image with multiple gamma-ray events. The magnified region corresponds to a detector region with five gamma-ray photon interactions: (a) raw detector image, (b) median-filtered image to smooth data and remove noisy pixels, (c) filtered image with individual clusters identified using connected-components-labelling algorithm (Suzuki et al., 2000), and (d) the estimated 2D gamma-ray interaction positions. The associated cluster data is written to disk as a listmode event entry (Barrett et al., 2009). (Reproduced from Miller, 2011.)

3.4 Large-Area BazookaSPECT Detector

MCP intensifier tubes are available in diameters ranging from 18 to 150 mm. Some devices, such as the XX1332 image intensifier, focus a large-area photocathode onto a smaller-area MCP, but this results in distortion at the edges and increased dark current at the centre (Miller et al., 2012). In a proximity-focused intensifier, the photocathode, MCP and phosphor screen are placed in close proximity which results in a very compact design. The proximity-focused intensifier used in the SyntheticSPECT system uses a double-stack MCP for a gain of approximately $10^6 W/W$. The area of the detector is limited by the active area of the intensifier if the scintillator is placed in direct contact with it. The largest intensifier currently available is 150 mm diameter and costs $> \$100k$ (*Photek*). Another approach to increase the active area is to insert a fibre-optic taper between the scintillator and the intensifier.

A coherent bundle of optical fibres can be used to relay an image plane from one end of a bundle to the other (figure 3.8). A magnifying taper is constructed by fusing the bundle together, heating the middle and pulling the ends apart

until the required dimensions are reached. This is then cut in half to produce two fibre-optic tapers (Roper Scientific, 2000), the magnification of which is given by the ratio of the two diameters. The taper used in this work was a 4:1 taper with an opening diameter (D) of 10 cm. The output window had a diameter

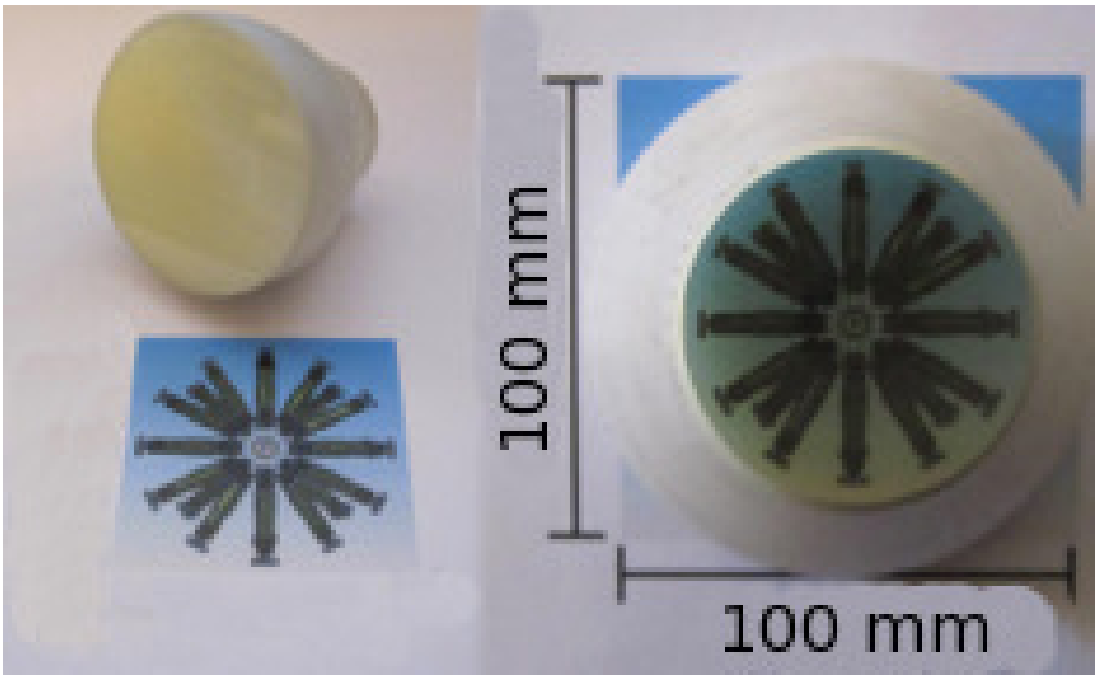


FIGURE 3.8: An example of a demagnifying fibre-optic taper. The image on the left is demagnified to produce the image on the right.

(D_{min}) of 2.5 cm, and it was directly coupled to the input window of the dual-MCP using index matching fluid. Using a fibre-optic taper in this configuration can lead to light loss if light enters at an angle greater than the acceptance angle (figure 3.9). The numerical aperture (NA) is a measure of the maximum incident angle that light rays are transmitted down the fibre via total internal reflection. It is calculated by

$$NA = nsin(\theta_{max}) = \sqrt{n_1^2 - n_2^2}, \tag{3.1}$$

where n, n_1, and n_2 are the refractive indices of the medium outside the fibre, the core and the cladding, respectively. The effective NA of the fibre-optic taper is

$$NA_{eff} = NAx\frac{D_{min}}{D}, \tag{3.2}$$

where D_{min} is the smaller diameter, and D is the larger diameter. If the incident light is well coupled to the acceptance cone, as described by equation 3.2, the taper may be used to increase the detector area.

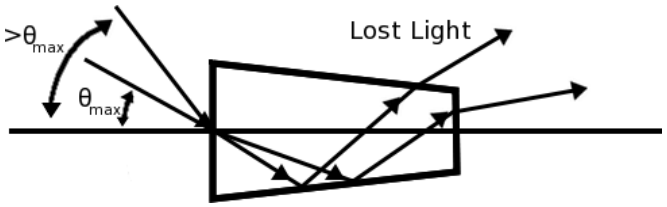
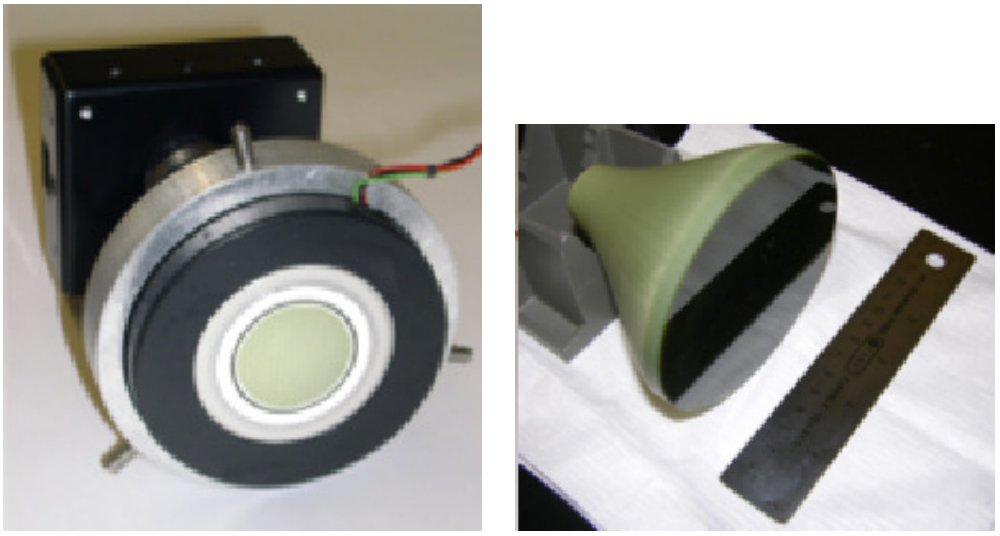


FIGURE 3.9: Light that enters the fibre at an angle greater than the maximum acceptance angle is lost. (Reproduced from Miller, 2011.)



(A) (B)

FIGURE 3.10: Large-area BazookaSPECT detector used with synthetic collimation. (a) Proxitronic Detector Systems GmbH, proximity-focused image intensifier: 25 mm diameter input/output windows; 2 MCPs in V-stack assembly; bialkali photocathode; P43 output phosphor screen; coupled to a CCD. (b) 4:1 (100 : 25 mm) fibre-optic taper, produced by Incom, Inc., that provides a 16 × increase in active area when using the 25 mm diameter image intensifier.

There is an inherent light loss from minification. A double-stack MCP was required for photon counting with a fibre-optic taper (figure 3.10); a single-MCP did not amplify the transmitted light sufficiently to distinguish it from noise in the CCD. The MCP voltage is regulated by an external control voltage (0 - 5 V). In this work, the single MCP was usually operated at 5 V, while the dual-MCP was operated at 1 V. Excess noise was observed when the dual-MCP was operated at higher voltages.

The detector area determines the FOV in a system with a parallel-hole collimator. With pinhole imaging, however, area alone is not what matters. The space-bandwidth (Sp-BW) product, defined as

$$\text{Sp-BW} = \text{Area of Detector}/\text{Area of PSF}, \tag{3.3}$$

determines how many independent measurements the detector can obtain and it is recognized as a figure of merit for imaging performance (Barrett and Hunter, 2005; DiFilippo, 2008). High Sp-BW detectors can provide significant improvement in overall efficiency when compared to large-area, low Sp-BW detectors (Miller, 2011). Using minification with a high Sp-BW detector results in the same image resolution that is achievable from a low Sp-BW detector at high magnification (Barrett and Hunter, 2005; Rogulski et al., 1993), but in the former case many more pinholes may be used, which increases the overall system efficiency. It was found that the BazookaSPECT system, using the 50 mm diameter intensifier with a 450 μm thick CsI(Tl) scintillator, has a space-bandwidth product of $\sim 1.81 \times 10^5$ (Miller, 2011). The intrinsic resolution of the large-area BazookaSPECT detector was measured as ~200 μm (Miller et al., 2012a). A large-area detector with high intrinsic resolution is desired for synthetic-collimation systems (Wilson et al., 2000).

3.5 3D Printing

3D printing technology was used extensively in this project (Miller et al., 2011). At CGRI, a 3D printer (Objet Geometries, Ltd., Connex350^{TM}, *Objet Geometries Ltd*) was used to build many of the parts used in the system (figure 3.11). This printer uses polymer-jetting technology, a process that involves the deposition of thin layers of photopolymer.

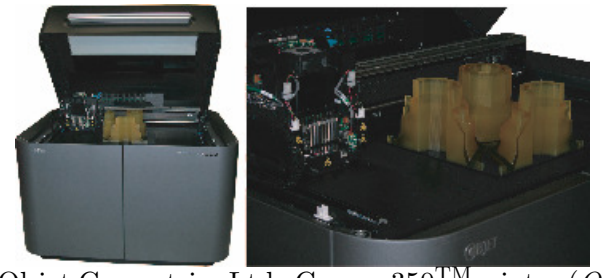


FIGURE 3.11: Objet Geometries Ltd. Connex350^{TM}printer (*Objet Geometries Ltd*). The printer has a 35 × 35 × 20 cm^3 build volume and prints slices 16 μm thick with a lateral resolution of 42 μm. The printer can print parts consisting of two different materials.

The part must be first designed using a 3D modelling program (SolidWorks). This file may then be sent to the printer as a series of slice images. The printer was able to deposit a layer 16 μm thick with a lateral resolution of 42 μm. Two

types of material were used: a model-building material, and a support material. Support material is required for complicated structures, but it must be cleaned out of the model after fabrication. The support material was cleaned out with a high pressure water jet. Delicate structures may get damaged by this process, and so each part was left to soak in an aqueous solution of 10% NaOH prior to cleaning. This solution softens the support material so that it can be more easily washed away, although prolonged exposure to the NaOH will also distort the printed part. There are many different types of model-building material available, each with their own characteristics. The 3D parts used in this work were built with FullCure720, TangoPlus, and VeroGrey (table 3.4).

TABLE 3.4: The physical characteristics of the 3D printing materials (*Objet Geometries Ltd*).

Material Name	Material Type	Tensile Strength (MPa)	Modulus of Elasticity (MPa)	Flexural Strength (MPa)	Flexural Modulus (MPa)	Polymerized Density (g/cm^3)
FullCure720	Transparent	50-65	2000-3000	80-100	2700-3200	1.18
VeroGrey	Rigid & Opaque	50-65	2000-3000	75-110	2200-3200	1.17
TangoPlus	Rubber-like	0.8 - 1.5	-	-	-	1.12

FullCure and VeroGrey are rigid materials, and TangoPlus is a rubber-like material. Initial attempts at designing the multi-pinhole collimator used FullCure and TangoPlus. The FullCure was used as the casing, while the TangoPlus was used for the pinholes. It was thought that the TangoPlus could be scraped away from the collimator once tungsten casting (*Tungsten Heavy Powder, Inc.*) was completed. This was indeed possible, but it was found that the material was too weak at the narrowest part of the double-knife-edge pinhole profiles to endure the cleaning process. This meant that the pinholes were filled with tungsten and would not transmit gamma rays. It was then decided that the entire casing be constructed with FullCure. It was not possible to remove this material from the collimator once tungsten casting was completed. The ability of the material to withstand the cleaning process is the limiting factor in the pinhole size with this form of fabrication.

The design and construction of the multi-pinhole collimator is discussed in the next section. Constructing this collimator would have been extremely difficult

using traditional machining methods due to the focused pinholes, the pseudo-random pinhole positions and unique aperture opening angles. Many other parts of the system were also built using 3D printing, including hot-rod phantoms, adapter plates, support plates, and detector housing.

3.6 Focused Multi-Pinhole Collimator

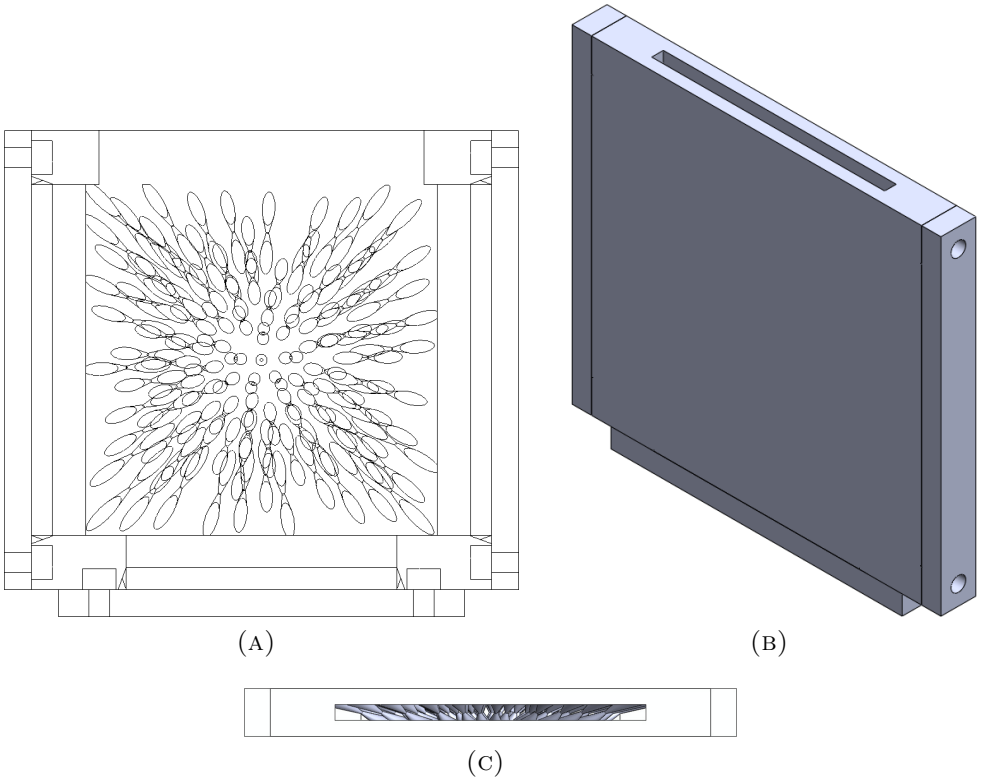
There are many factors to consider in the design of a multi-pinhole collimator (Cao et al., 2005; Rentmeester et al., 2007). A list of the key parameters is given in section 2.4.2. As discussed in section 2.4.1, a tradeoff exists between many of the parameters of a multi-pinhole collimator. Before considering these tradeoffs, it was determined that the size of the FOV could not be compromised. Laboratory mice typically have a diameter of 20 - 25 mm, and a length of > 80 mm. It was decided that the FOV should be able to accommodate the diameter of a mouse, and that the mouse could be translated through the FOV for a full-body scan. The FOV was set at 25.4 mm.

The system resolution and sensitivity cannot be considered independently. The lateral resolution and the sensitivity of the system are primarily controlled by the pinhole diameter, the magnification, the collimator material and aperture opening angle. With no rotation of the object or detector, angular sampling is directly proportional to the number of pinhole apertures; improved angular sampling will result in increased depth resolution. The size and position of the FOV determined the opening angles of the focused pinhole apertures. If the FOV was positioned too far from the collimator then the sensitivity would be very low, irrespective of the number of pinhole apertures, and if the FOV was positioned too close to the collimator then the geometry of the focused apertures would overlap with each other which would result in increased collimator penetration. The centre of the FOV was placed 23 mm from the collimator to maximize sensitivity while minimizing aperture overlap.

The collimator was made of a tungsten epoxy (*Tungsten Heavy Powder, Inc.*), and had a density of $\sim$12 g/cm^3. When solidified, the tungsten epoxy may have some cracks or bubbles in the material which could reduce the effectiveness of the collimator. Due to structural concerns, it was recommended that the collimator be several millimetres thick. The collimator was made to be 6 mm thick, which

corresponds to ~ 18 half value layers when used with ^{99m}Tc (Hubbell and Seltzer, 1995). The pinholes were located on the central plane of the collimator and had a double-knife-edge (DKE) profile (Accorsi and Metzler, 2005) focused towards the centre of the FOV.

The collimator was designed and fabricated using rapid prototyping (figure 3.12). Unlike the work by Miller et al., 2011, the mold material was not discarded after casting. With rapid-prototyping it is very difficult to maintain the pinhole aperture shape after removing the mold. Miller et al., 2011 created separate platinum pinhole inserts to negate this problem. In this work, many more pinhole apertures were required, and platinum inserts were too costly. By leaving the mold intact, the pinhole apertures maintained their desired profile at the expense of gamma-ray interaction with the mold material.



(A) (B)

(C)

FIGURE 3.12: Several views of the collimator casing. This casing was made with FullCure720^{TM} material. The casing had four open sides to allow the cavity to be cleaned of support material. Three sides of the cavity were then screwed shut. The tungsten epoxy was poured into the cavity through the one remaining open side. The sides were tightly shut because the casing was placed in a centrifuge during the tungsten solidifying process. (a) Front view (wire). The pinhole profiles can be seen within the object, (b) Isotropic view, (c) Top view. The solid pinhole profiles can be seen inside the cavity.

Due to concerns over the rigidity of the 3D printing material, it was decided that the pinhole apertures should be 1 mm diameter; smaller pinholes could break during the cleaning process. The casing also had to be strong enough to withstand the tungsten pouring and solidifying process, which required it to be placed in a centrifuge. The entire casing had a thickness of 18 mm, which included the cavity at the centre where tungsten was poured (figure 3.12); this placed a lower limit on the collimator-to-detector distance.

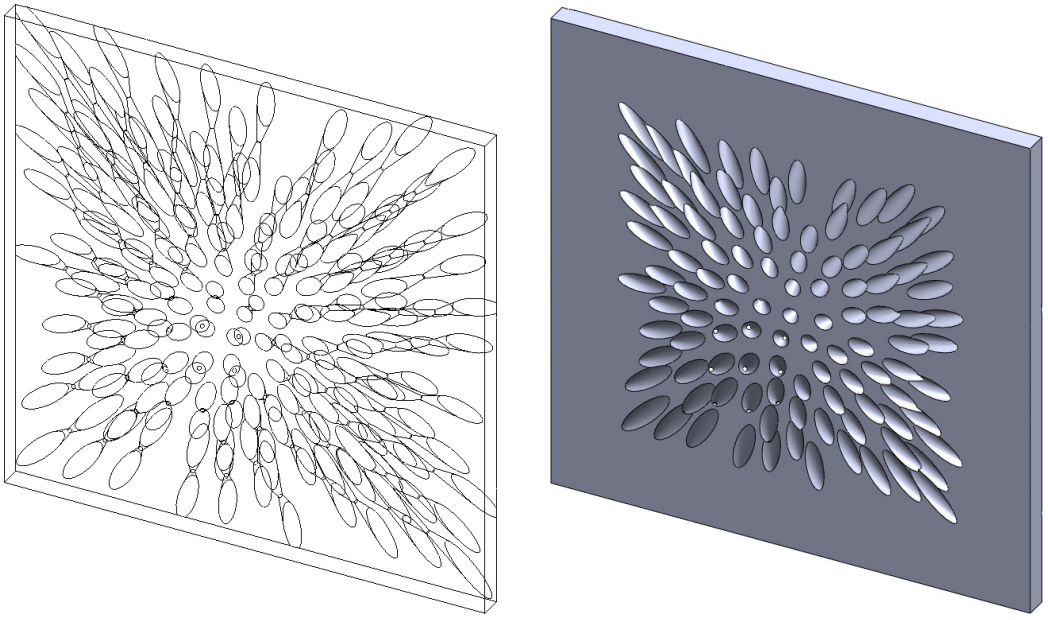
As the collimator-to-detector distance decreases, more non-overlapping pinhole projections fit on the detector surface. Synthetic collimation requires that the pinhole projections are non-overlapping in at least one geometric configuration. The size of the FOV is inversely proportional to the number of non-overlapping pinhole projections that fit on the detector surface. With a lower bound to the collimator-to-detector distance, a FOV set at 25.4 mm and a fixed detector size, the inter-pinhole distance that resulted in non-overlapping pinhole projections was 9 mm. A lower value was initially calculated, although it was increased to 9 mm in anticipation of system construction errors and other experimental tolerances.

It has been shown that irregular pinhole patterns are less likely to result in reconstruction artefacts than regular pinhole patterns (Vunckx et al., 2008b). Therefore, the pinholes were pseudo-randomly positioned, but with the constraint that they should be 9 mm from a neighbouring pinhole. A better solution would be to develop an algorithm that accounts for septal penetration and the location of the pinhole aperture with respect to the central pinhole of the collimator.

The collimator was constructed by first designing a cavity, with the shape of the collimator, within a solid block. Tungsten epoxy was then poured into this cavity, resulting in a tungsten collimator (figure 3.13) encased in FullCure material (figure 3.12).

The opening angles of the pinhole apertures were calculated by projecting a collapsing cone from a central slice that is orthogonal to a line joining the centre of the FOV and the pinhole location (figure 3.14). One needs to find the location of the apex of the cone which results in a knife-edge pinhole at the centre of the collimator. From figure 3.14 it is easy to see that

$$\tan\left(\frac{\alpha}{2}\right) = \frac{r_1 - r_p}{d_1} = \frac{r_1}{d_1 + x}, \tag{3.4}$$



(A) (B)

FIGURE 3.13: The tungsten collimator within the casing of figure 3.12. (a) Isometric view (wire), (b) Isometric view (solid).

where r_1 is the size of the slice through the FOV, r_p is the radius of the pinhole, and d_1 is the distance between the centre of the FOV and the pinhole, and x is the orthogonal distance the apex lies behind the pinhole. Rearranging equation (3.4):

$$x = \frac{d_1 r_p}{r_1 - r_p}. \tag{3.5}$$

Using this value for x and the corresponding value for α, the conical geometry was input to SolidWorks to create one edge of the double-knife-edge pinhole. The other edge of the pinhole was constructed in a similar manner, but the base of the cone was a distance d_1 behind the pinhole along the line connecting the centre of the FOV and the pinhole coordinate. These calculations were repeated for every pinhole location. The collimator and a collection of central slices used to form the apertures are shown in figure 3.15. The collimator had 110 pinhole apertures, but the number of pinhole projections that were detected was reduced to 46 when used with the 10 cm fibre-optic taper.

3.7 SyntheticSPECT

A container for the taper and intensifier was designed on SolidWorks and built using the 3D printer (figure 3.16). This was made of transparent FullCure, but

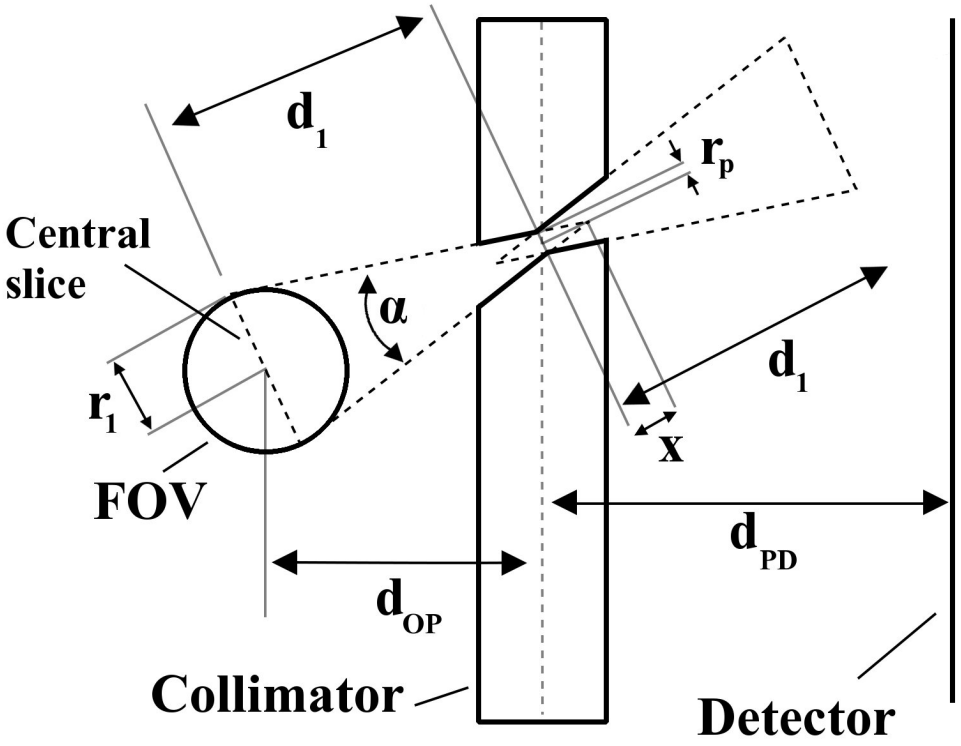
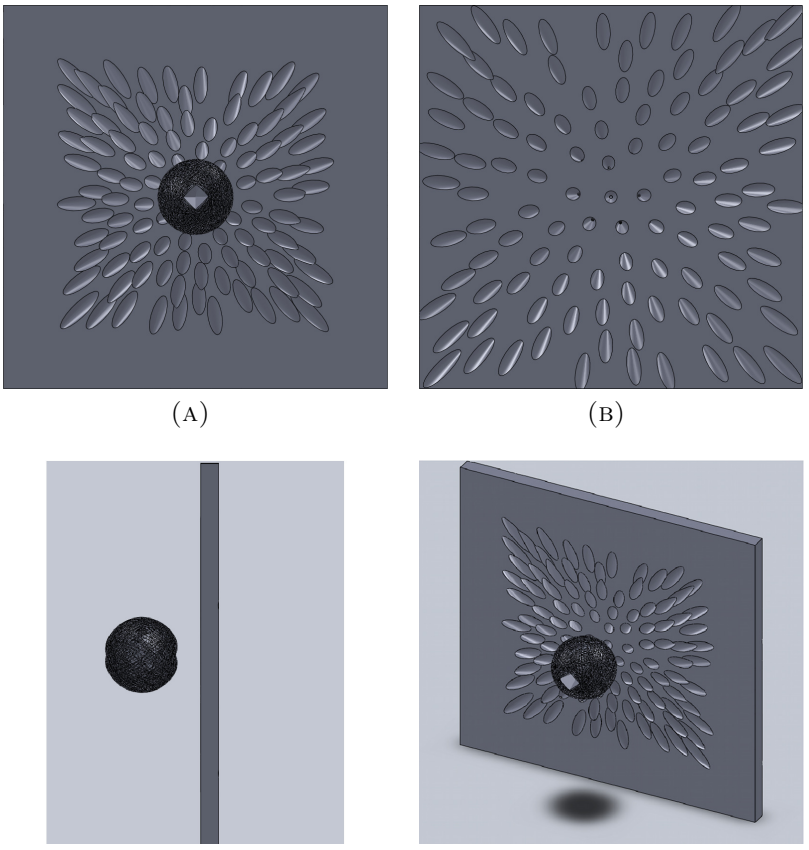


FIGURE 3.14: The double-knife-edge pinhole geometry of an off-centre aperture. The centre of the pinhole aperture is on the central plane of the collimator. The central slice of the FOV, which is used to project the collapsing cone to create the pinhole, is on a plane that is orthogonal to a line between the centre of the FOV and the pinhole aperture. An identical slice is formed a distance d_1 on the detector-side of the collimator to create the double-knife-edge. The opening angle, α, is unique to each pinhole aperture and is calculated using equation (3.4).

was spray painted black to reduce the effect of ambient light on the system. The CCD, image intensifier and fibre-optic taper required a solid support stage to reduce the possibility of movement (figure 3.17). Any movement of the system requires a new system matrix to be measured. The support stage was designed on SolidWorks, and built using 80/20 materials (*80/20, Inc.*). An aluminium plate was custom built and attached to this support structure. This plate held the intensifier, taper and CCD rigidly in place. The collimator was held in place above the support structure and black-box casing with a holder that was built using 3D printing (figure 3.18) with VeroGrey^{TM} material. The collimator was a heavy piece of equipment and it needed to be held rigidly in place. The detector assembly was held in place by the 80/20^{TM} support structure and aluminium plate. The assembly was attached to an optical table (*Thor Labs*) which enabled precise positioning of the equipment. A SolidWorks assembly of all the equipment described in this chapter was designed so that the central pinhole of the collimator



(A) (B)

(C) (D)

FIGURE 3.15: The (a) front view, (b) back view, (c) side view, and (d) isotropic view of the collimator with the approximately spherical FOV visible. This was designed using SolidWorks, and constructed using rapid-prototyping. It was then cast using a tungsten epoxy. The spherical object consists of the central slices of the FOV from which cones are projected to create the focused pinholes (figure 3.14). The sphere is incomplete due to the limited angular sampling of the system, and this is noticeable in (a) and (d) as a bright spot along the z axis of the object.

could be placed directly above the centre of the detector assembly (figure 3.19). This precise equipment positioning removed the need for lateral movement of the collimator, therefore a single-axis Velmex translation stage (*Velmex, Inc.*) was sufficient to control the vertical positioning of the collimator.

A 3-axis motion stage was required to control the positioning of the object within the field of view. This 3-axis stage was required for experimental measurement of the system matrix (chapter 4) and translation of an object through the FOV for extended-object imaging.

Figure 3.20 shows a magnified image of the system during a phantom study. The full system is shown in figure 3.21.

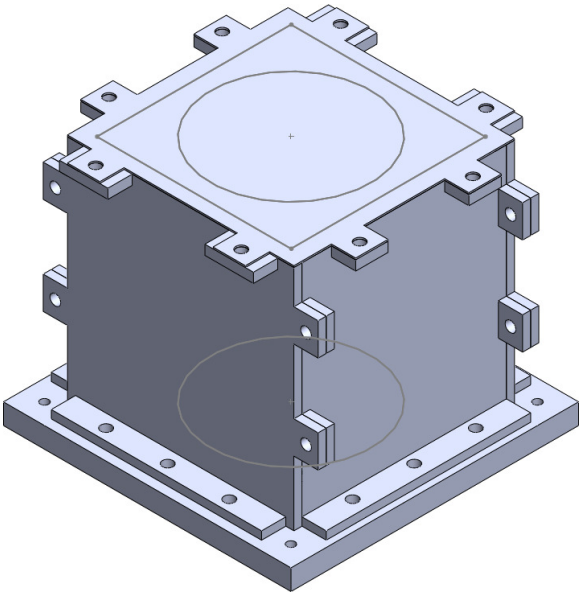


FIGURE 3.16: The image intensifier must be placed in a light-tight casing. This case was built using 3D printing, and was painted black.

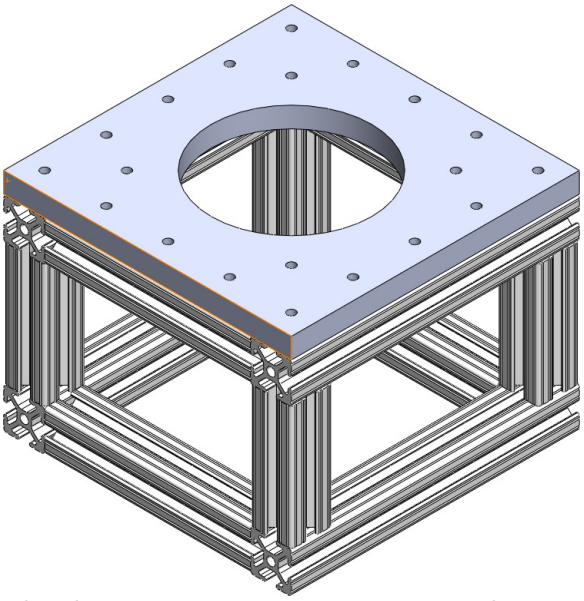


FIGURE 3.17: The detector support structure was designed with 80/20™ material. An aluminium plate was attached on to this support. The plate was designed to hold the intensifier, taper, and CCD rigidly in place.

3.8 System Operation

The system used 4 Velmex translation stages to control the position of the collimator and the object. One stage controlled the z position of the collimator, and the other three stages controlled the position (x,y,z) of the object. The gamma camera remained fixed to the bench. Using a LabVIEW template created at CGRI to control the CCD detector, a custom LabVIEW program was developed

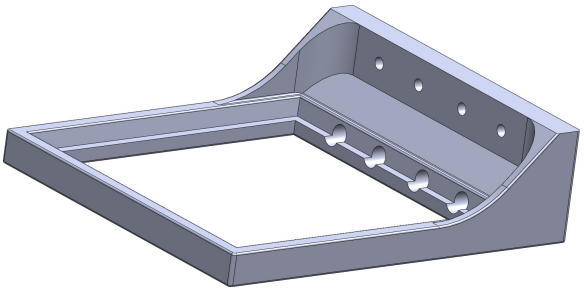


FIGURE 3.18: The collimator was placed on this holder. The holder was then attached to a translation stage that enabled fine adjustments of the detector-collimator distance.

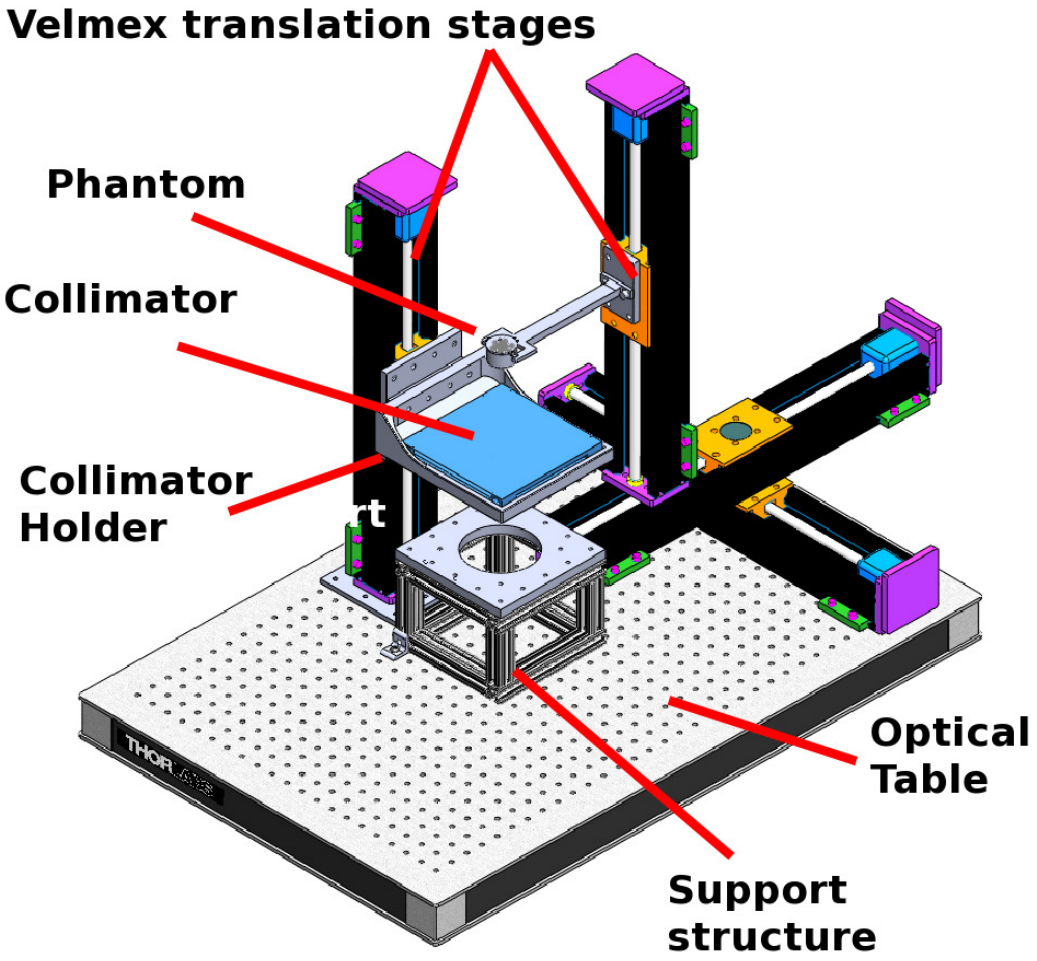


FIGURE 3.19: SolidWorks rendering of the SyntheticSPECT assembly. This assembly was designed to help align the central pinhole of the collimator with the centre of the support structure. The BazookaSPECT detector is not shown in this diagram. The object can be translated in three dimensions, and the collimator can be translated along the z axis only.

to allow the user to carry out a range of imaging tasks (figure 3.22), including

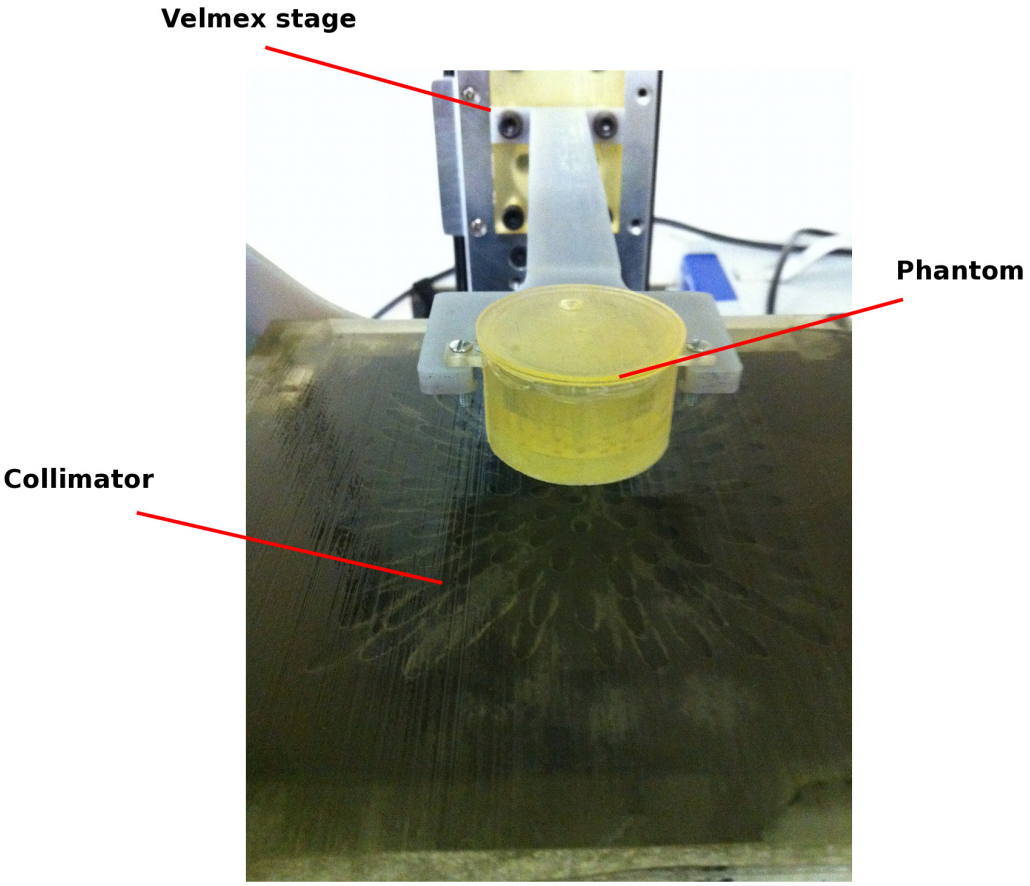


FIGURE 3.20: The focused multi-pinhole collimator and hot-rod phantom. The tungsten collimator is inside a FullCure^{TM} support.

moving the object within the FOV and adjusting the acquisition parameters.

The LabVIEW interface allowed the user to configure all the parameters of the imaging study before image acquisition. These parameters included:

- Acquisition mode: Single-frame, frame parsing or PSF mode.
- Length of acquisition.
- Video mode: All supported video modes for the camera (320 × 240 × 8 bit was usually chosen).
- Frame rate.
- CCD gain.
- CCD shutter time.

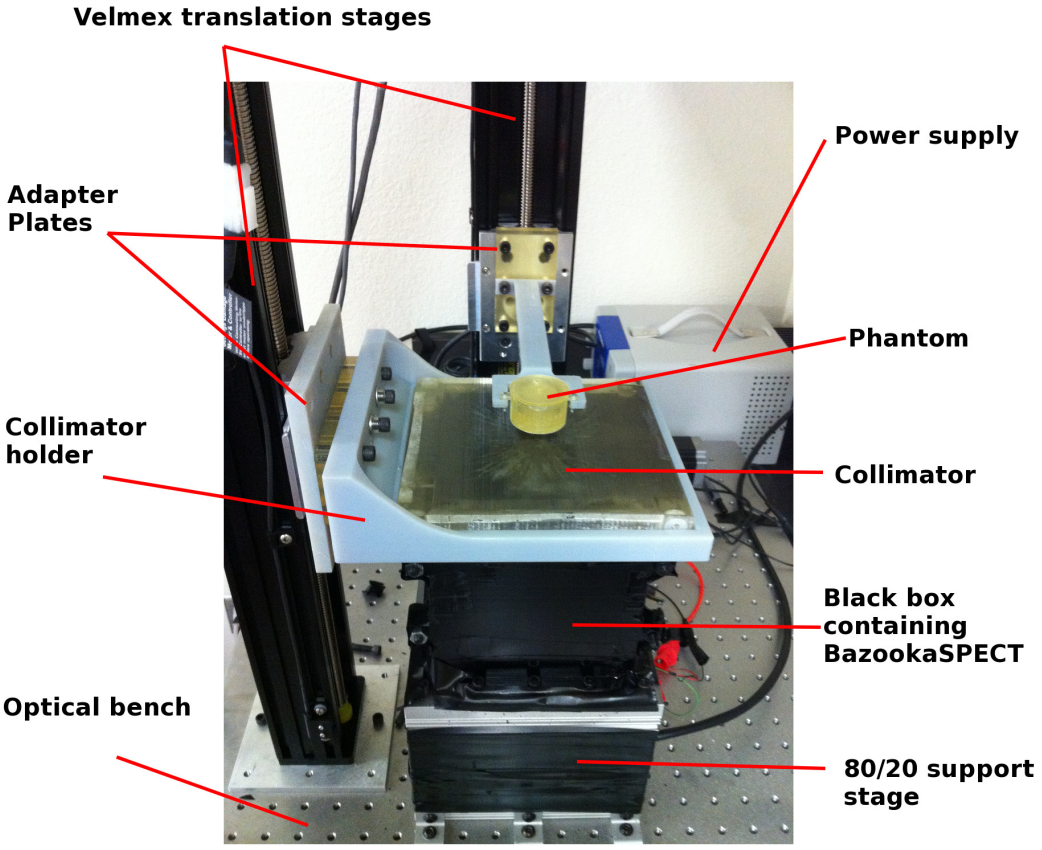


FIGURE 3.21: The SyntheticSPECT during a hot-rod phantom experiment. The parts of the system are labelled in the image. The two translation stages that were used for vertical motion (z axis) are shown. The two stages used for lateral motion (x and y axes) are not shown.

- Signal threshold.
- Cluster radius.
- Radioisotope.

"Single-frame" mode acquired a specific number of frames from the camera. This was used prior to an imaging study to ensure pileup was not occurring on the detector. The images were visually inspected, and the shutter and threshold settings were adjusted accordingly. The "frame parsing" mode acquired images, applied the threshold, detected the clusters, and stored the listmode data (section 3.3). An integrated centroid image was also produced from the listmode data by binning the interaction position to the nearest pixel location. Frame parsing

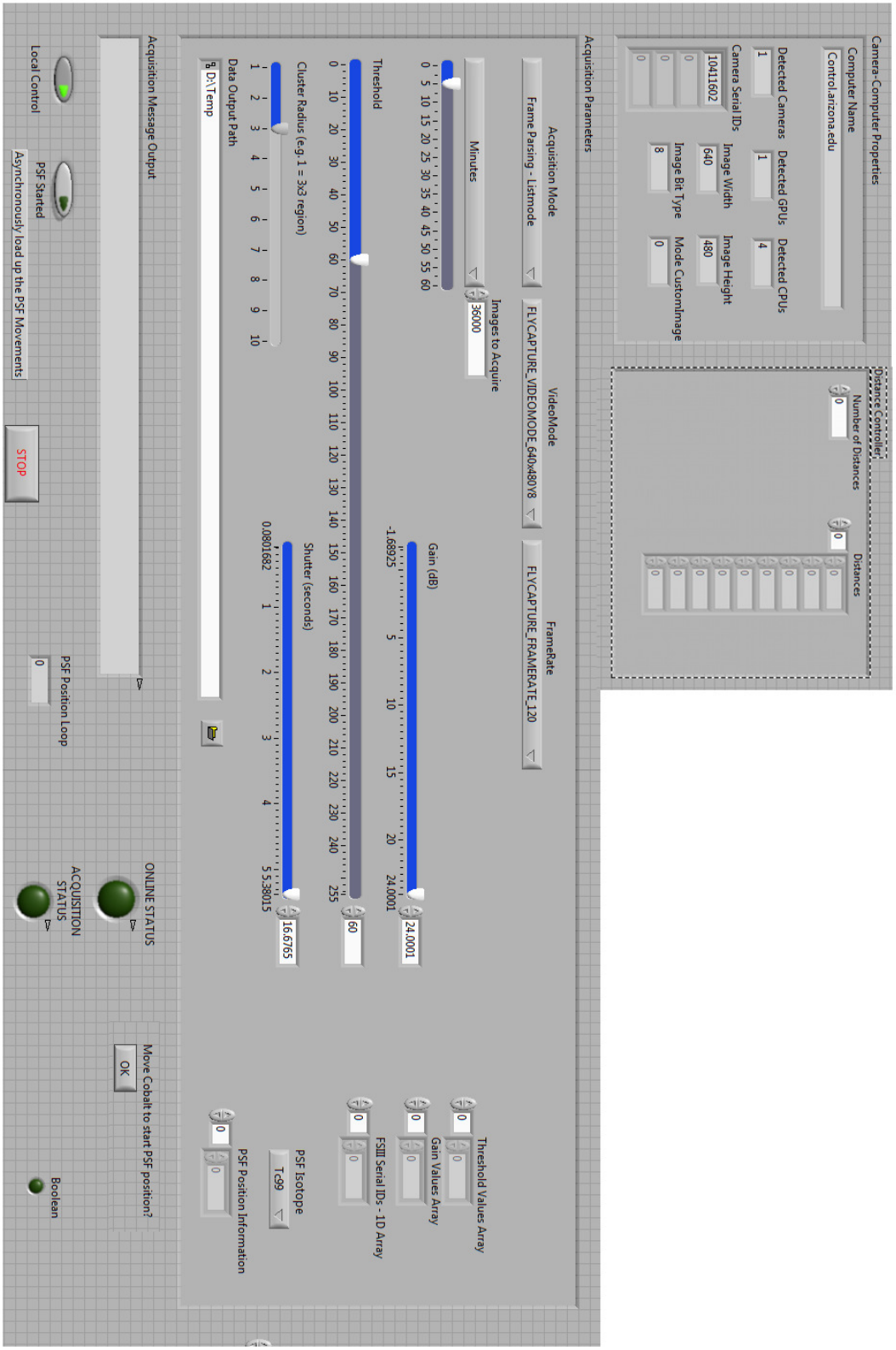


FIGURE 3.22: The LabVIEW control interface for SyntheticSPECT.

was performed for both "synthetic-collimator" and "PSF" modes. In "synthetic-collimator mode" the data acquisition was paused while a separate sub-controller, created by the author, moved the four Velmex stages into the correct position. In "PSF mode" the data acquisition was paused while the object was moved to its next position within the three dimensional calibration grid. The 'PSF mode" code implemented a sub-controller pre-written by the collaborators at CGRI. The LabVIEW workflow is shown in figure 3.23.

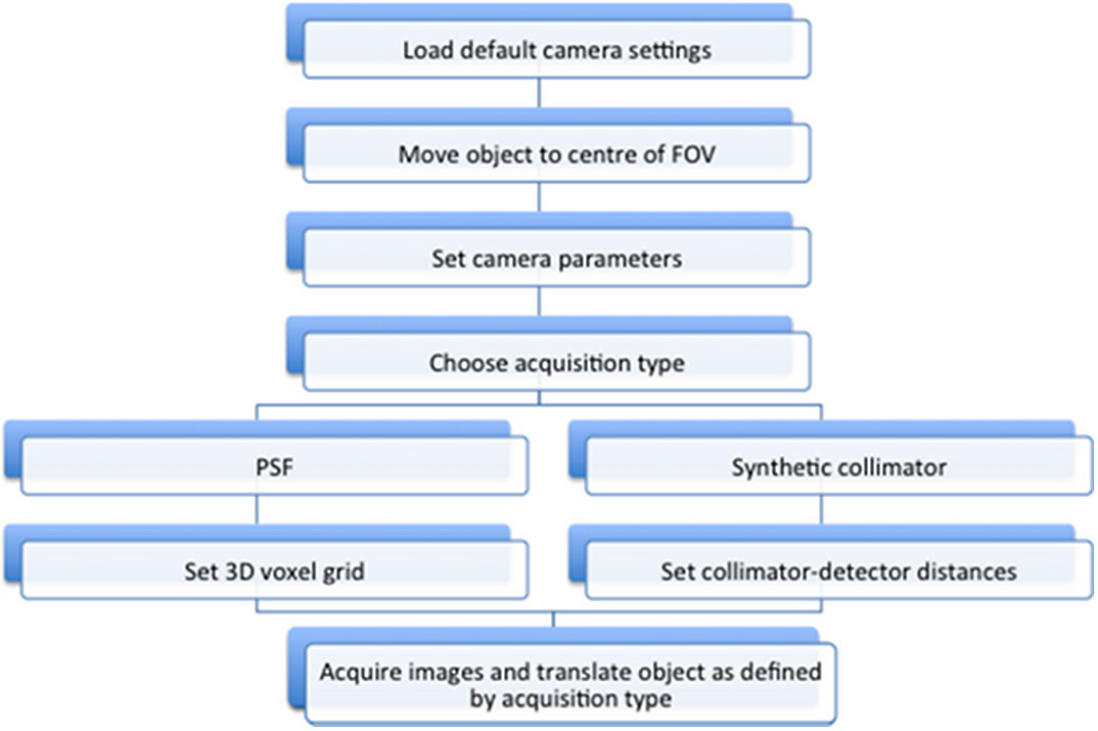


FIGURE 3.23: The LabVIEW interface workflow. The default camera settings were loaded at startup. The object was then moved to the centre of the FOV. The required camera settings were then selected. The acquisition type was then chosen. If "PSF" was chosen then the PSF sub-controller was loaded and prompted the user to input the parameters of the 3D voxel grid. If "Synthetic Collimator" was chosen then the synthetic collimator was loaded; this required all the collimator-detector distances. Both of these sub-controllers signalled the main controller when a movement was complete and image acquisition could begin.

CHAPTER 4

Performance Characterization of SyntheticSPECT

The performance of a SPECT system is usually represented by characteristics such as spatial resolution, sensitivity, and energy resolution. In this work, the spatial resolution was estimated using a hot-rod phantom and through calculation of the Fourier crosstalk matrix. The relative sensitivity of each of the system configurations was calculated, but the peak sensitivity of the system was not estimated. The energy resolution of the BazookaSPECT detector was known to be very poor and so it was not thoroughly investigated.

Section 4.1 details the method used to measure the system matrix (section 2.5) and calibrate the system. The system matrix was experimentally measured on a sparsely-sampled grid. It was assumed that each pinhole projection can be approximated by a 2D Gaussian distribution. This distribution contains information on the probability of detection of a photon emitted from a voxel in object space. Details of the Gaussian fitting are given in section 4.2. Section 4.3 outlines the algorithm used to associate projections that were transmitted through the same pinhole but originated from different grid positions. Section 4.4 describes the method used to interpolate the system matrix. Section 4.6 describes the multiplexing of the system. As the magnification of the system increases, the amount of projection overlap increases. The level of multiplexing was not optimized in this work. Section 4.6 outlines simulations performed using the interpolated system matrices. These simulations validated the reconstruction

algorithm, the interpolation of the system matrix, and the concept of synthetic collimation. Section 4.7 discusses the Fourier crosstalk matrix and how it may be used to characterize a system. Possible sources of error are discussed in each section.

4.1 Calibration of the Imaging System Matrix

An imaging system can be described by

$$\mathbf{g} = \mathbf{H}\mathbf{f} + \mathbf{n}, \tag{4.1}$$

where $\mathbf{H}$ is the discrete system matrix that maps the object vector, $\mathbf{f}$, to an image, $\mathbf{g}$, and the noise, $\mathbf{n}$, is a M-dimensional vector in data space. The object, $\mathbf{f}$, is decomposed as a set of N voxels ($[f_1, \ldots, f_N]^t$), and the image as M data bins, therefore $\mathbf{H}$ is a $M \times N$ matrix. As mentioned in section 2.6, the are many methods to experimentally measure the system matrix (Beque et al., 2003; Defrise et al., 2008; Furenlid et al., 2004; Have et al., 2008; Miller et al., 2012b; Rowe et al., 1993). The most comprehensive method is to exhaustively measure the response of the system to a point source positioned at the center of each voxel. The resolution and FOV of the reconstructed image depends on the sampling of the object space.

SyntheticSPECT implements synthetic collimation for image formation and reconstruction. Synthetic collimation requires several projection images of the object, each taken at a different system magnification, for the reconstruction. The system matrix needs to be determined for every system configuration, and experimental measurement of several high-resolution system matrices is impractical. The system matrix can be measured by translating a point source through a sparsely-sampled grid in object space and recording the system response at each voxel, for each system configuration. A high-resolution system matrix can then be interpolated from these sparsely-sampled data.

In this work, the CCD data were binned to a 320×240 array of pixels, but not all of these pixels imaged the intensifier. The voxel function, f_n, was mimicked by stepping the point source through a 3D grid of measurement points in object space. For each photon interaction in the scintillator there was a certain amount

of light spread, and the light was detected as a cluster of pixels. A frame-parsing algorithm extracted and processed each cluster (Miller et al., 2009). A centroid calculation was performed on the cluster to estimate the 2D interaction location. This interaction location was then binned to the nearest pixel. The integration of these pixel interactions over the acquisition time formed a projection image (figure 4.1).

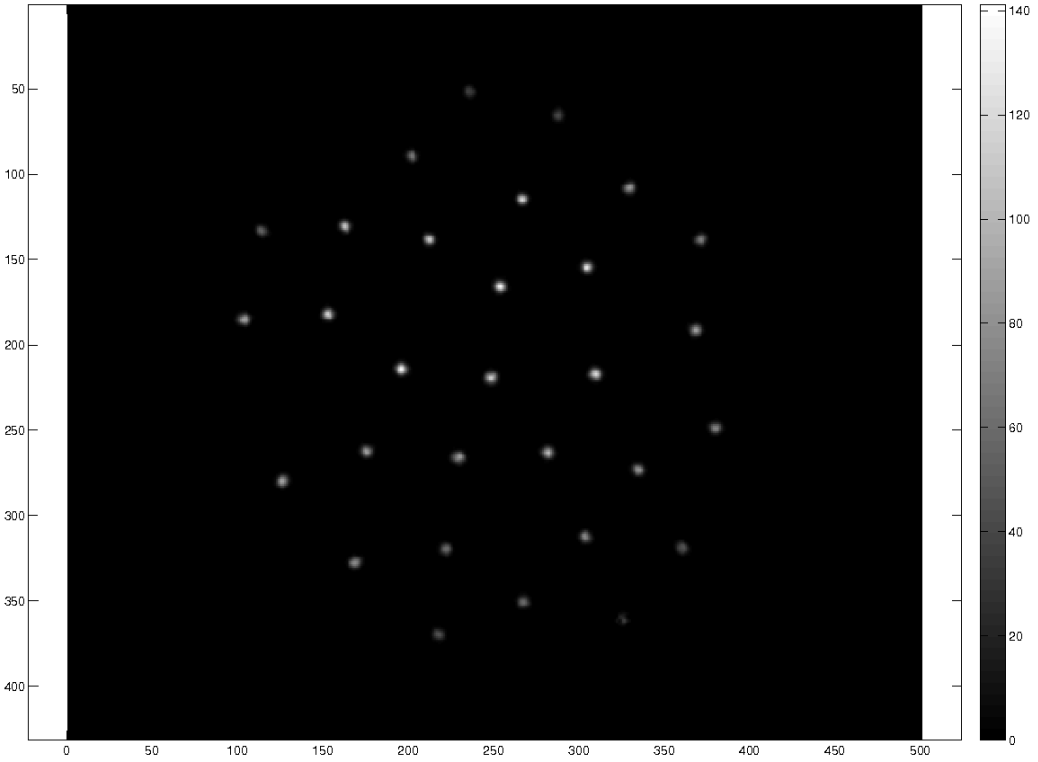


FIGURE 4.1: An integrated projection image. A point source was positioned somewhere in the FOV and data were recorded for a certain time. The individual photon interactions were binned to the nearest pixel, and a projection image was formed by integrating the frames. This is a $640 \times 480 \times 8$ bit image that has been cropped to the area around the active pixels. $320 \times 240 \times 8$ bit images were primarily used for data acquisition to increase the number of counts per pixel.

The mean system response was measured by recording many counts for each voxel position. In systems previously built at CGRI, it was found that > 2500 counts should be recorded for each voxel position for reliable modelling. The sensitivity of a pinhole system increases as the source is moved closer to the pinhole aperture. Therefore, the acquisition time for the system response function may be adjusted accordingly. The acquisition time was 140 seconds at voxel positions farthest from the collimator, and it was 20 seconds at locations nearest

the collimator. The projections were then normalized to an equal time period. To accurately estimate the 2D interaction locations of the gamma rays, data must be acquired without pileup on the detector. It was necessary to find a frame rate, or exposure time, that was suitable when the point source was at each position within the FOV. The source was first placed at a position in the FOV closest to the collimator, and the exposure time was then adjusted until pileup had a low probability of occurring.

TABLE 4.1: System configurations used in this work. The object-pinhole distance remained constant, and the magnification of the system was controlled by adjusting the pinhole-detector distance.

System Configuration	Object - Pinhole Distance, d_{OP} (mm)	Pinhole - Detector Distance, d_{PD} (mm)	Magnification
1	23	9	0.4
2	23	17	0.8
3	23	25	1.1
4	23	33	1.5
5	23	36	1.6
6	23	46	2.0

The system matrix must be measured for each system configuration (d_{OP} : d_{PD} combination, table 4.1). The response was measured on a $13 \times 13 \times 13$ cubic grid of points, with spacings of 2.5 mm, and interpolated to a grid of $97 \times 97 \times 97$ points, corresponding to cubic voxels with a side-length of 0.3125 mm. This grid was greater than the expected common FOV, but it was measured to enable more accurate interpolation of the response at the edges of the FOV. A system matrix with 0.5 mm voxels, corresponding to a grid of $61 \times 61 \times 61$ points, was also interpolated. The interpolation method is discussed in section 4.4.

The normalized total counts recorded at each voxel location are shown in figures 4.2-4.7. These figures show that the number of counts detected increased as the source was moved closer to the collimator. The number of events detected also reduced as the magnification of the system increased, due to the reduction of the solid angle subtended by the detector. Figure 4.8 shows the events detected at each system configuration when the source was at the centre of the FOV.

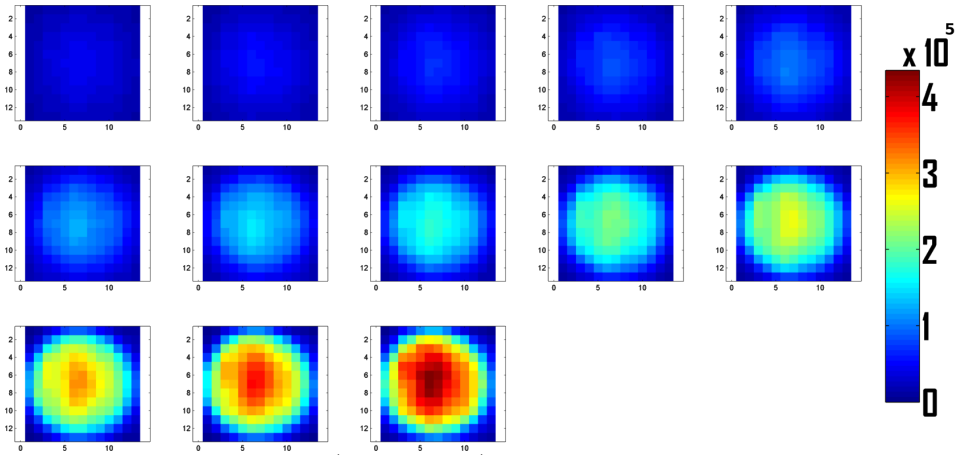


FIGURE 4.2: Total counts (normalized) recorded at each voxel position for the first system configuration. Each image is a slice along the z axis. The first slice is farthest from the collimator, and the last slice is nearest the collimator.

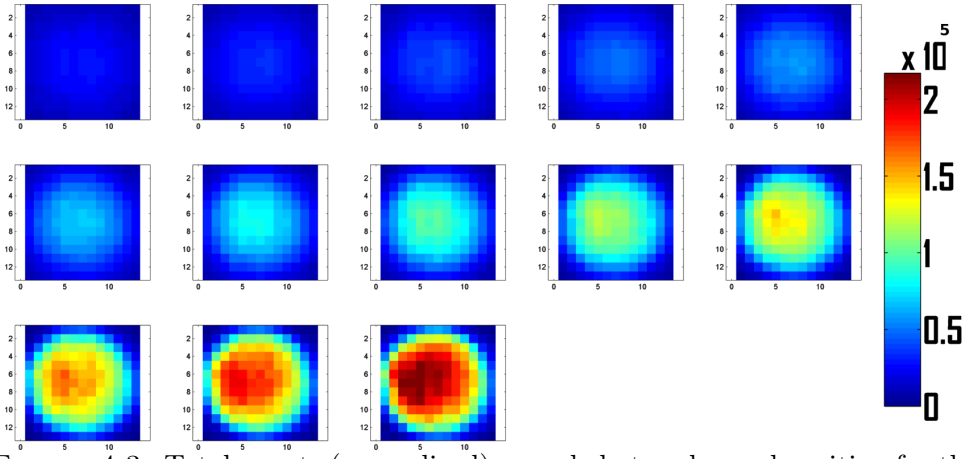


FIGURE 4.3: Total counts (normalized) recorded at each voxel position for the second system configuration. Each image is a slice along the z axis. The first slice is farthest from the collimator, and the last slice is nearest the collimator.

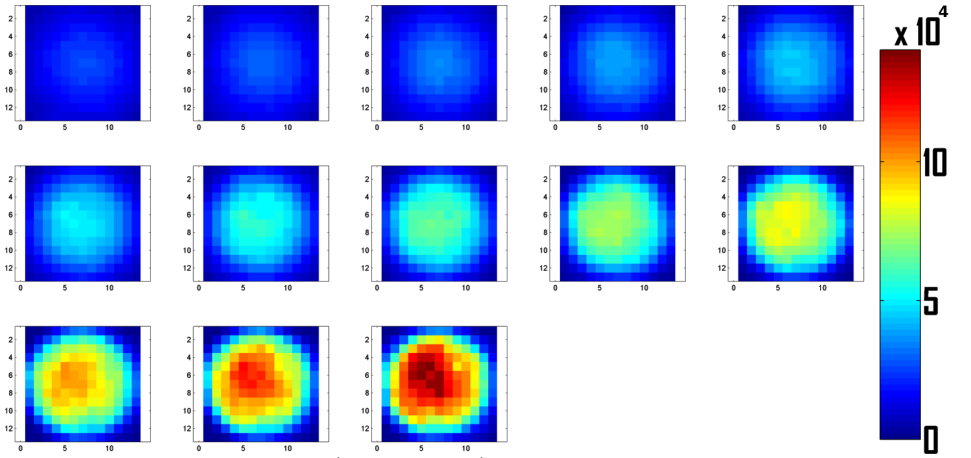


FIGURE 4.4: Total counts (normalized) recorded at each voxel position for the third system configuration. Each image is a slice along the z axis. The first slice is farthest from the collimator, and the last slice is nearest the collimator.

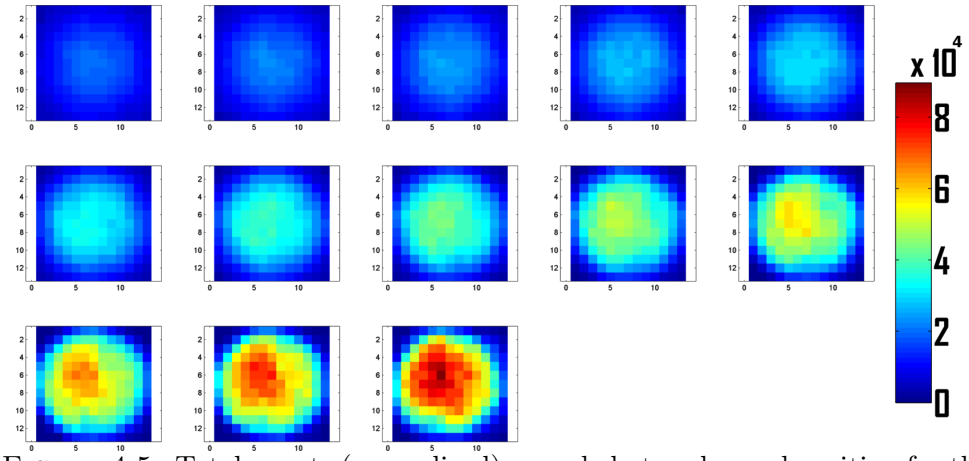


FIGURE 4.5: Total counts (normalized) recorded at each voxel position for the fourth system configuration. Each image is a slice along the z axis. The first slice is farthest from the collimator, and the last slice is nearest the collimator.

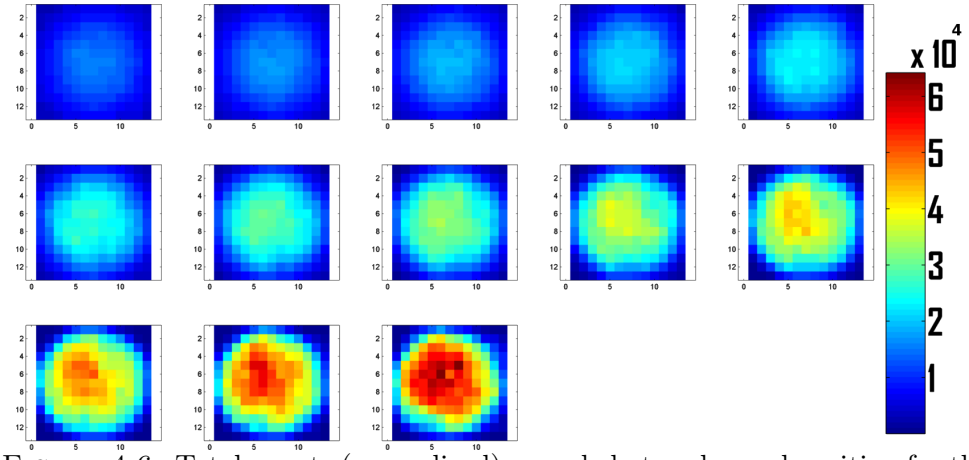


FIGURE 4.6: Total counts (normalized) recorded at each voxel position for the fifth system configuration. Each image is a slice along the z axis. The first slice is farthest from the collimator, and the last slice is nearest the collimator.

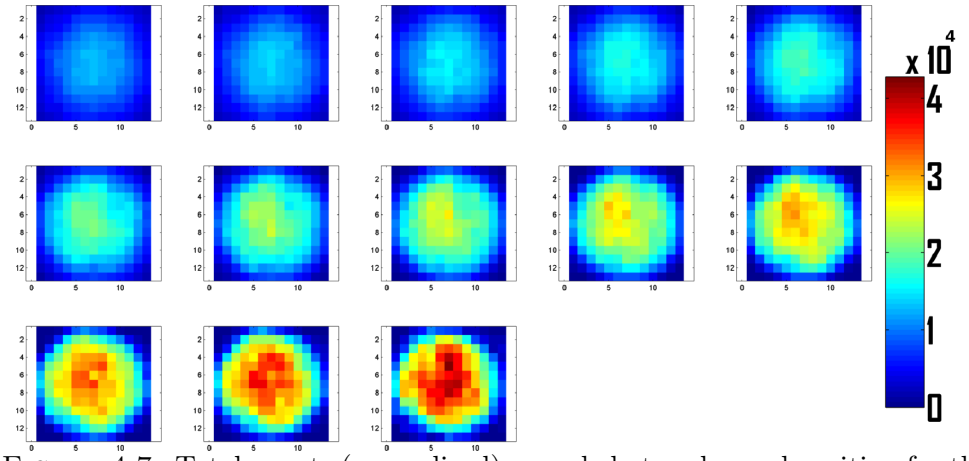


FIGURE 4.7: Total counts (normalized) recorded at each voxel position for the sixth system configuration. Each image is a slice along the z axis. The first slice is farthest from the collimator, and the last slice is nearest the collimator.

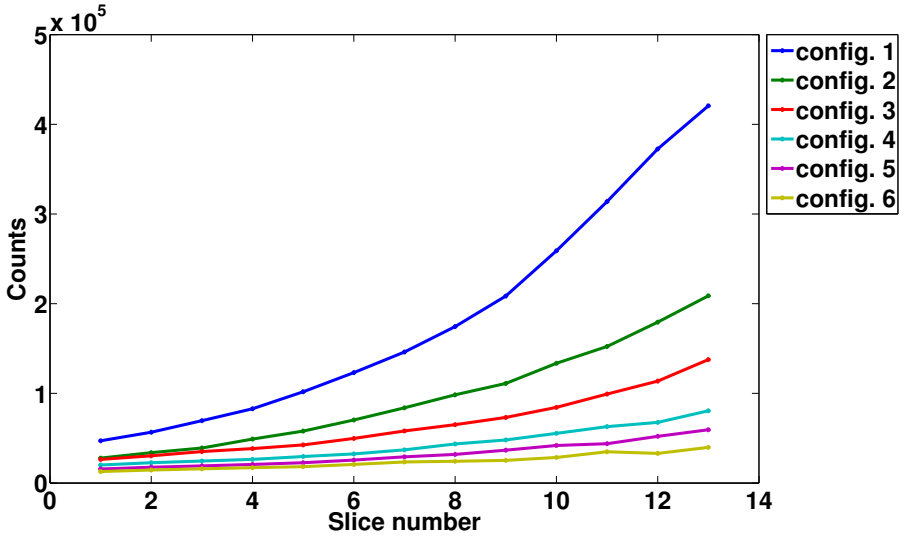


FIGURE 4.8: Total counts (normalized to 100 seconds) recorded at each voxel position at the centre of the FOV, at each slice, for each system configuration. Slice 1 is farthest from the collimator.

4.2 Pinhole Identification and Gaussian Fitting

It is difficult to interpolate using the raw projection images. Instead, a 2D elliptical Gaussian distribution was fitted to the pinhole projections, and their coefficients were interpolated (Chen et al., 2005). Miller et al., 2012b demonstrated that a 2D elliptical Gaussian distribution was a suitable model to fit to the pinhole-projection data acquired by a BazookaSPECT detector. In this section, it is demonstrated that the 2D elliptical Gaussian distribution is suitable for the SyntheticSPECT system and the multi-pinhole collimator. Alternative distributions, such as a combination of a 2D Gaussian and a heavy tailed function such as an exponential, may better describe the septal penetration and collimator scatter response, although they are also likely to increase the computational time and were not analyzed in his work (Chun et al., 2013).

In SyntheticSPECT, each projection image had up to forty-six pinhole projections. Each of these pinhole projections had to be identified, segmented and associated with the correct aperture before interpolation.

Figure 4.9 shows a typical PSF image for a single voxel. The pinhole projections were first identified in each raw projection image using a thresholding algorithm. The image pixels that were below the threshold were set to zero. The resulting image was then filtered with a median filter to remove hot pixels. This

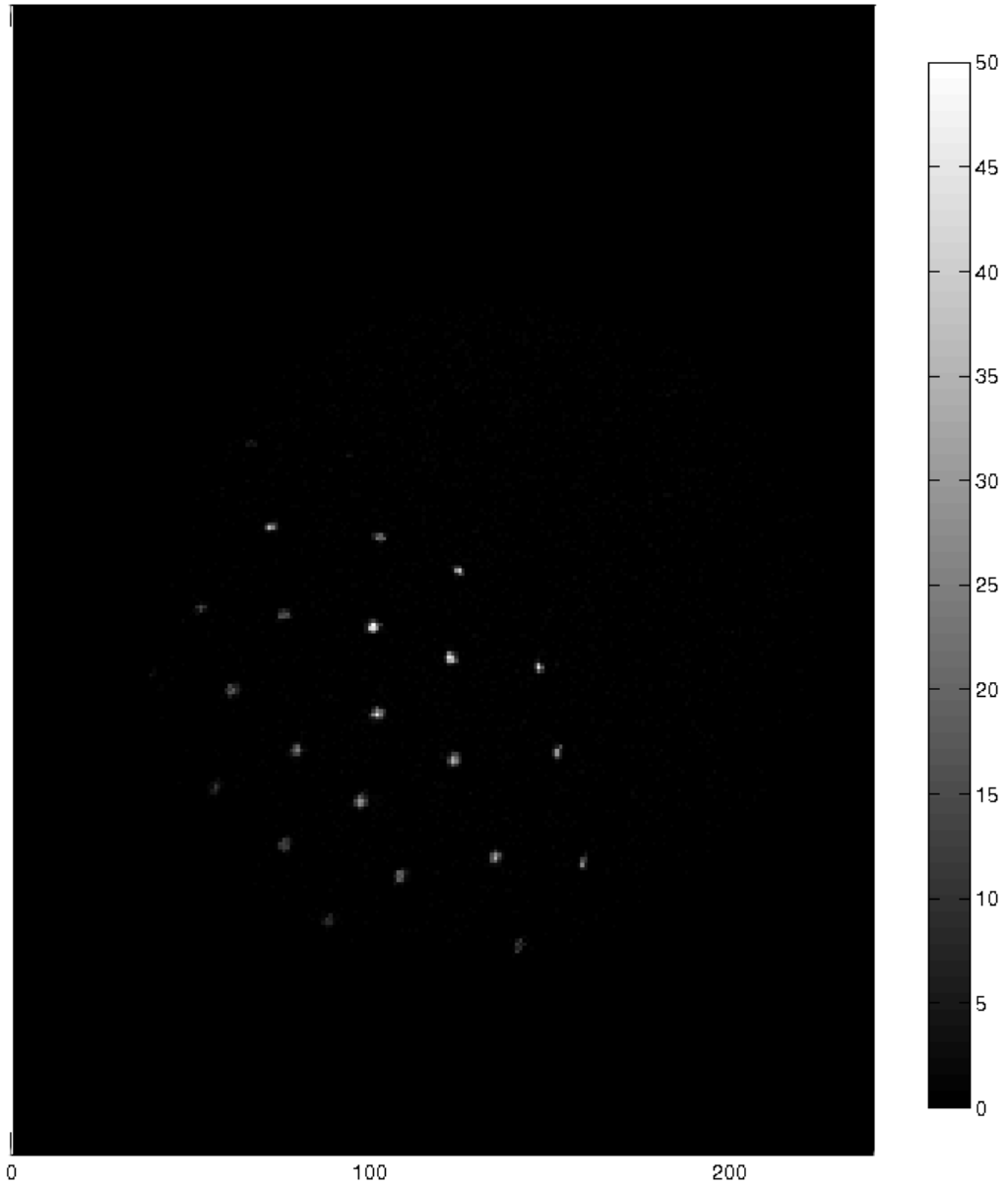


FIGURE 4.9: A typical integrated projection image for one voxel position. The voxel was in the corner of the FOV, and only the pinhole apertures that projected this corner were imaged. There are 21 pinhole projections in this image.

processed image was used for finding the pinhole projections, but the raw image was used for fitting a distribution. The pinholes were 1 mm diameter, and a typical pinhole projection was distributed over several pixels. Clusters of pixels were found using the MATLAB^{TM} function "*bwconncomp*". This function finds connected components in a binary image. The clusters, and a small region of pixels surrounding them, were individually isolated in the raw projection image (figure 4.10).

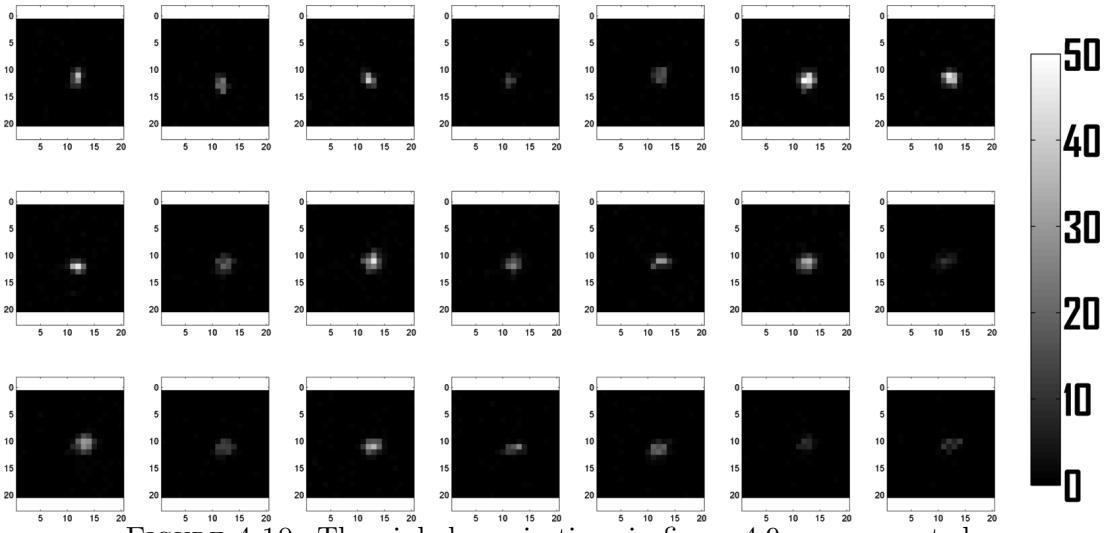
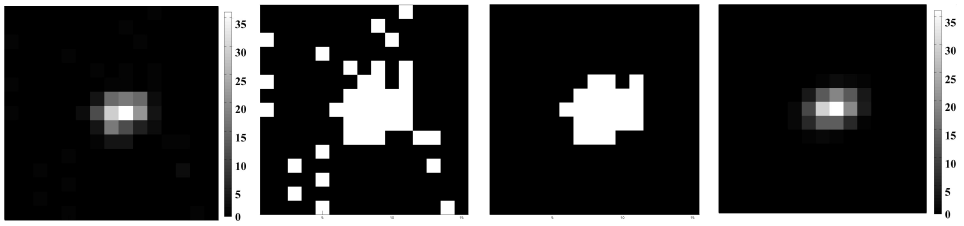


FIGURE 4.10: The pinhole projections in figure 4.9 are segmented.

The centroids of the clusters were then calculated, and the neighbourhood around each cluster was re-centred (figure 4.11). Fig. 4.11a shows a single cluster from figure 4.10 and its binary equivalent before (figure 4.11b) and after processing (figure 4.11c). The pixel values within the clusters were unaffected by any processing. The binary equivalent is shown to highlight noise in the data that could adversely affect the Gaussian fitting algorithm. Figure 4.11d shows the reprojection of the Gaussian coefficients fitted to the projection in figure 4.11a.



(A) (B) (C) (D)

FIGURE 4.11: (a) The pinhole projection from image 17 in figure 4.10. (b) the binary image of (a). (c) Some noise was removed from the edge of the cluster. (d) The PSF was reprojected using the Gaussian coefficients fitted to (a).

To get an initial estimate of the cluster size, every pixel below a certain threshold in this neighbourhood was set to zero. The cluster was then outlined using the MATLAB^{TM} function "*bwselect*". It was possible that the values below the threshold at the edge of the cluster contributed to the signal rather than the noise, therefore the region defined by the "*bwselect*" function was expanded by one pixel in all directions. The centroid of this cluster was re-calculated, and the window was re-centred.

The coefficients that characterize a 2D elliptical Gaussian distribution are the amplitude (A), the x coordinate ($\bar{x}$), the y coordinate ($\bar{y}$), the x-width (σ_x), the y-width (σ_y), and the correlation coefficient (ρ). Estimates, and their upper and lower bounds, were made of these coefficients, and a 2D Gaussian fit was made to the cluster using a constrained non-linear optimization function in MATLAB™ ("$fmincon$"). The pixel response function can be represented by

$$f(x,y) = \frac{A}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \times exp\Big\{ -\frac{1}{2(1-\rho^2)}\Big(\frac{(x-\bar{x})^2}{\sigma_x^2} + \frac{(y-\bar{y})^2}{\sigma_y^2} - \frac{2\rho(x-\bar{x})(y-\bar{y})}{\sigma_x\sigma_y}\Big)\Big\}. \quad (4.2)$$

The constraint used in the fitting procedure is a Gaussian approximation to Poisson data. The number of photons detected in a detector element obeys Poisson statistics (equation 2.33). Using a Gaussian approximation to the Poisson probability (Barrett and Swindell, 1981) gives the following probability density function:

$$\mathrm{pr}(\mathbf{g}|\theta) = \prod_{m=1}^{M} (2\pi\bar{g}_m(\theta))^{-\frac{1}{2}} \times exp\left[\frac{(g_m - \bar{g}_m(\theta))^2}{2\bar{g}_m(\theta)}\right], \quad (4.3)$$

where g_m are the measured counts in the detector elements, and $\bar{g}_m$ are the mean counts determined by the fitting procedure. Taking the natural log of equation 4.3 gives

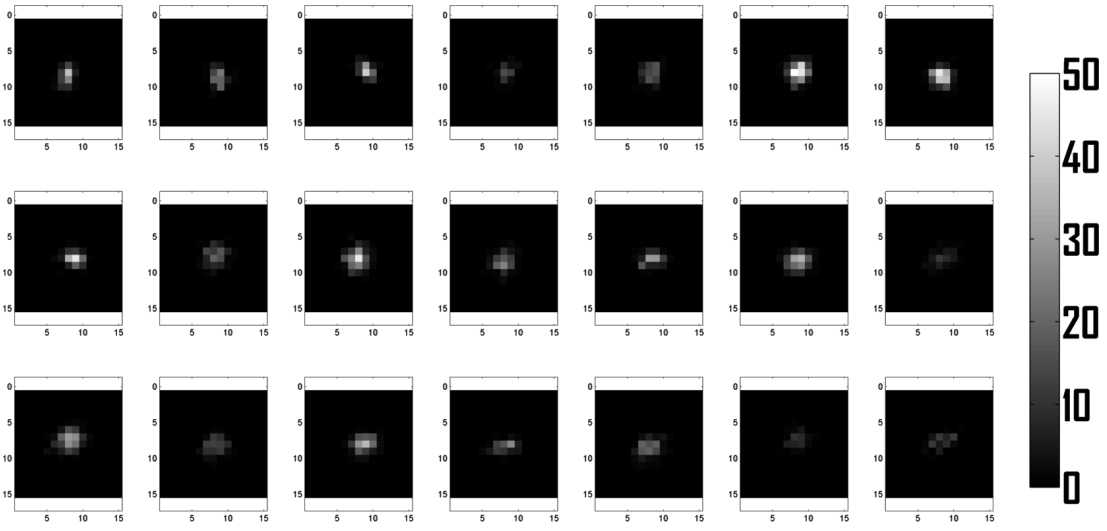
$$-\ln(\mathrm{pr}(\mathbf{g}|\theta)) = \sum_{m=1}^{M} \frac{1}{2}\ln[(2\pi\bar{g}_m(\theta))] + \frac{(g_m - \bar{g}_m(\theta))^2}{2\bar{g}_m(\theta)}. \quad (4.4)$$

The maximum likelihood estimate of the coefficients that fit the distribution will minimize $-\ln(\mathrm{pr}(\mathbf{g}|\theta))$.

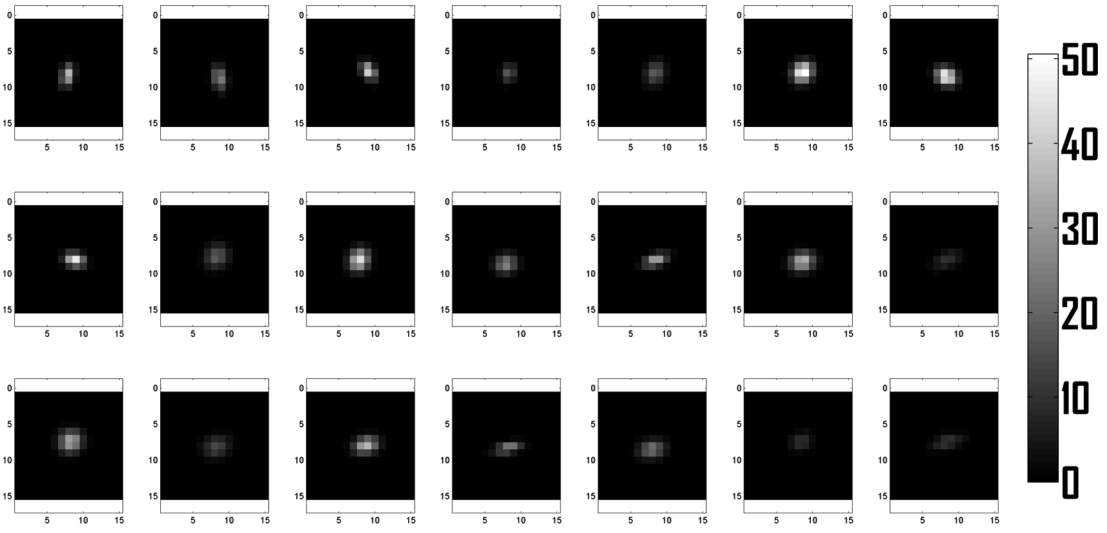
The number of counts in the distribution can be preserved by applying a correction factor determined by the total signal in the raw, and fitted clusters. The total counts in the fitted distribution can be found by reprojecting the fit, or analytically using

$$\textbf{Total Counts} = A \times 2\pi \times \sigma_x \times \sigma_y \times \sqrt{(1-\rho^2)}. \quad (4.5)$$

Figure 4.12a shows the pinhole projections from figure 4.10 after noise processing. A Gaussian distribution was fit to each of these clusters and their reprojections are shown in figure 4.12b.



(A)



(B)

FIGURE 4.12: (a) The neighbourhood around the projections in figure 4.10 were reduced, and noise was removed from the edges of the clusters. (b) 2D Gaussian fits of the clusters in (a).

To further examine the Gaussian model, a cluster in the projection image, detected when the point source is at the centre of the FOV, was isolated and processed (figure 4.13). Figure 4.13a shows the original projection image with a cluster highlighted with a white box. Several mesh views of this cluster are shown in 4.13c. Figure 4.13b shows the reprojection of the Gaussian fits, and the same cluster is also highlighted in the reprojection. The Gaussian fit of the raw cluster data is shown in figure 4.13d.

The Gaussian approximation has been examined when the source is at the edge of the FOV and at the centre of the FOV. The Gaussian model appears to be a good approximation for the cluster distribution and was used for every pinhole

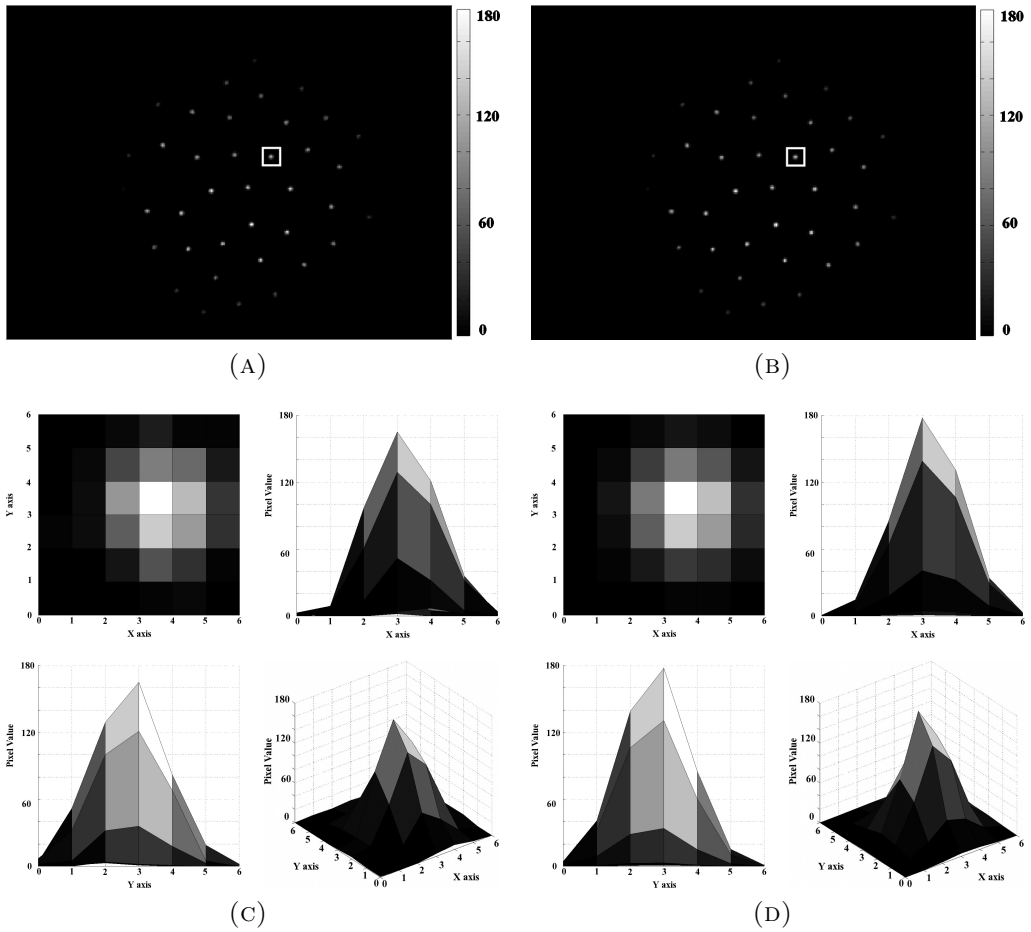
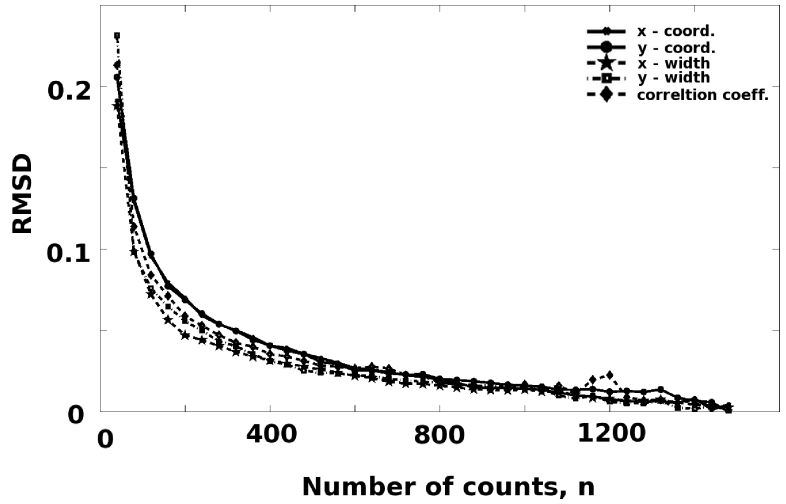


FIGURE 4.13: (a) A raw projection image, acquired when a point source is placed at the centre of the FOV, is processed and a Gaussian fit is made to each cluster. (b) The Gaussian fits are reprojected. A single cluster is highlighted in each image with a small white box. (c) The magnified image of the cluster in (a), and several mesh views of the cluster. (d) The magnified image of the cluster in (b), and several mesh views of the cluster.

projection. The coefficients of the Gaussian fits then needed to be associated with their respective pinhole apertures.

4.2.1 Effect of the Number of Counts on the Fitting Procedure

The number of photons transmitted through each pinhole aperture for a single point-source position was different due to the different source-pinhole distances. The variance of the counts detected in each pixel is inversely proportional to the number of counts detected in each pixel. Figure 4.14 shows the effect of increased counts on the Gaussian distribution fit to each pinhole projection; the coefficients of the Gaussian distributions at different numbers of counts, n, were compared



(A)

FIGURE 4.14: The effect of the number of counts in a pinhole-projection cluster on the fitting of a 2D elliptical Gaussian distribution. A 2D elliptical Gaussian distribution was made to each cluster for different numbers of counts. The RMSD was calculated by comparing the coefficients of each fit after n counts were acquired with the coefficients of each fit after the total, N, counts were acquired.

to the coefficients of the distribution when N counts had been detected. The total number of counts, N, transmitted through each aperture is different at each voxel due to sensitivity differences across the FOV.

The root-mean-square difference (RMSD) between the coefficients of the Gaussian fits reduced to < 0.1 pixels after around 100 counts were detected in a cluster. The fitting procedure was performed on pixelized data, it is likely that an improved result could be achieved using subpixel estimation of the gamma-ray interaction positions.

4.3 Pinhole Tracking

The number of pinhole projections detected at each voxel location ranged from zero to forty-six. These pinholes had to be identified, and associated with each other before interpolation could be performed. The locations of the pinhole apertures were fixed, but the parameters of the detected projections changed as the point source was translated through the 3D voxel grid in object space. It was necessary to match a pinhole aperture with its corresponding projection for each voxel position before interpolation could be performed. This labelling was done on a voxel-by-voxel basis starting with the "base voxel", the voxel from which

the most pinhole projections were detected. The projections from the base voxel were individually labelled, and the voxel was noted as processed. This labelling of the projections from the base voxel was analogous to labelling of the actual apertures. As each new voxel was processed, it was determined which apertures were associated with each of the projections from that voxel. This was a trivial task when the centroid positions of the projections through each aperture varied slowly between voxels. When a few voxels had been processed it was possible to extrapolate the parameters of the pinhole projections from adjacent, unprocessed voxels. A threshold was used to compare extrapolated parameters with measured parameters in an unprocessed voxel. If the threshold was met, then the pinhole projection were accurately labelled. If it was not met, it was possible that the cluster was transmitted through a previously unlabelled aperture. A pinhole projection was flagged for manual inspection if its measured location was different from the estimated position by a large amount (> 10 pixels).

This algorithm was suitable for data in which the inter-voxel locations of the pinhole projections varied slightly. It was difficult to find a threshold level that identified each pinhole accurately in data with much projection overlap. This problem was overcome by taking two low-magnification data sets and extrapolating the parameters of the pinhole projections in higher-magnification data. Before extrapolation was performed, the pinhole projections were identified and associated between each voxel and between each data set. Due to the pinhole geometry, any pinhole projection visible in high-magnification data should have been present in low-magnification data. There were some deviations between expected and measured positions (~ 20 pixels), most likely due to lens distortion, but misidentification was not found to be a problem, upon inspection.

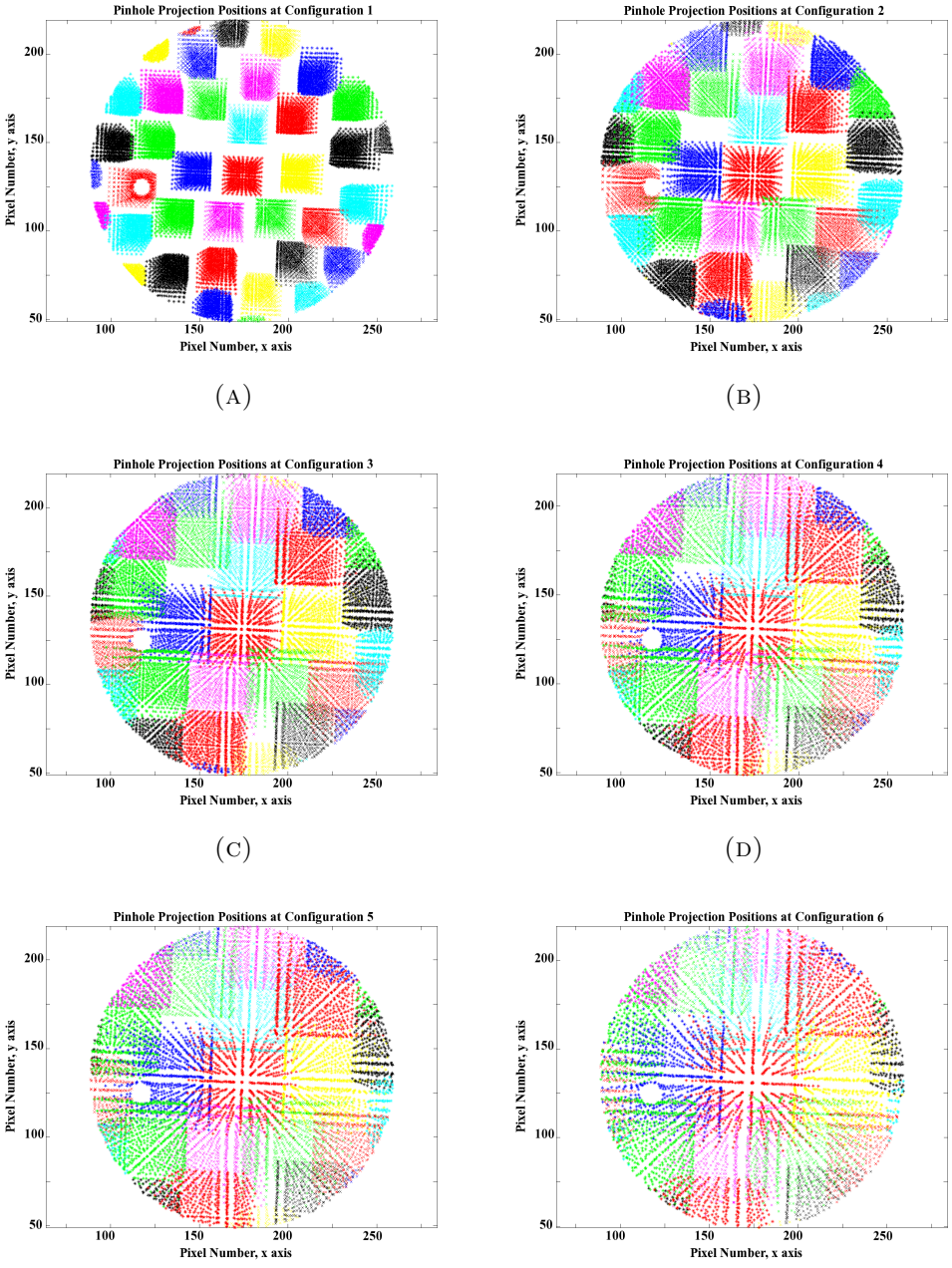
Each pinhole aperture had a voxel-space grid assigned to it that stored the coefficients of the projection through that pinhole from every voxel location. The threshold implemented in the association algorithm was used as a first attempt to identify the pinhole projections. Some pinhole projections were very weak, or had very low values for σ_x and σ_y, and were removed by the thresholding. The omission of these pinhole projections in the final system matrix could have resulted in errors in the interpolation and extrapolation, which are performed in the next section. It was often not possible to determine whether a cluster was a weak pinhole projection, or noise attributable to scatter. When the pinhole projections with strong signals were identified and tracked, it was possible to extrapolate these

data to predict the locations of weak pinhole projections. The positions of the weak pinhole projections were estimated in a shell, with a thickness of one voxel, outside of the measured data within the voxel-space grid. The distance threshold between predicted locations and the potential pinhole data was set to 5 pixels. Figure 4.15 shows the centroid positions of the Gaussian fits for all experimentally measured PSF locations within the range -12.5 mm to +12.5 mm on each axis. The PSF was measured in 13 steps ranging from -15 mm to +15 mm but these peripheral regions are not shown here. They are omitted to help visualize the areas where no overlap occurs in the low magnification configuration. Figure 4.15 shows the increased overlap of the FOV on the detector as the magnification increased. The number of pinhole projections that were detected was reduced as the magnification increased. Effectively, the sensitivity reduced as the resolution increased. A large range of colours was chosen so that any mislabelled pinhole projections would be noticeable. From looking at the centroid projections, there appear to be no errors in pinhole labelling. A finer system matrix could now be interpolated.

Before interpolation was performed, it was important to investigate the data being processed. The 3D grids of coefficients corresponding to the central pinhole and an oblique pinhole are shown in figures 4.16 and 4.17. The changes in the values of the coefficients were smooth between voxels. It should also be noted from figures 4.16 and 4.17 that data were not acquired through both pinholes for every voxel position. This was because the system response was measured in a region outside the FOV common to every pinhole.

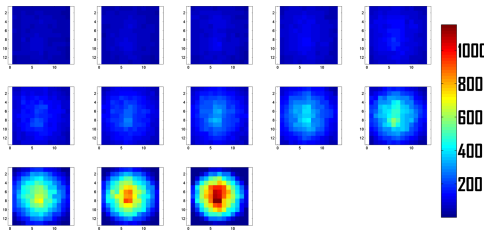
4.4 Interpolation of the System Matrix

With the pinholes identified, their coefficients were arranged into three-dimensional arrays corresponding to the PSF measurement grid for each pinhole (i.e., arrays of amplitudes (A), centroid positions ($\bar{x}$,$\bar{y}$), x- and y- spreads (σ_x,σ_y), correlation coefficients (ρ)). These 3D arrays were then interpolated to the desired grid size using MATLAB^{TM}'s 3D interpolation function ("*interp*3"). Interpolation using this function requires a complete 3D grid of data; voxels with no measured values will result in errors.

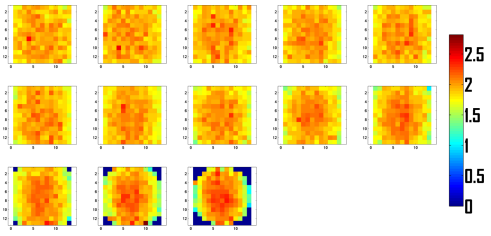


(A) (B) (C) (D) (E) (F)

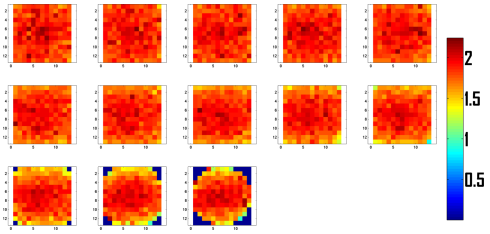
FIGURE 4.15: Centroid positions of the Gaussian coefficients fitted to the experimental pinhole projections for magnifications: (a) 0.4, (b) 0.8, (c) 1.1, (d) 1.5, (e) 1.6, (f) 2.0. The pinholes have been identified, and their detector positions are plotted. A range of colours is used to distinguish between the different pinhole projections. The PSF voxel grid can clearly be seen in the projections. The amount of projection overlap is proportional to the magnification. A small circular insensitive area of the intensifier can clearly be seen in the same location in all images. These images show the overlap of the FOV ranging from -25 mm to + 25 mm. The system matrix was measured in a voxel space of -30 mm to +30 mm but it is not shown here.



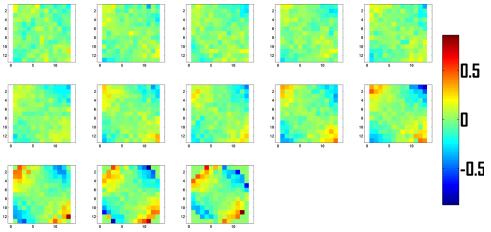
(A) Amplitude



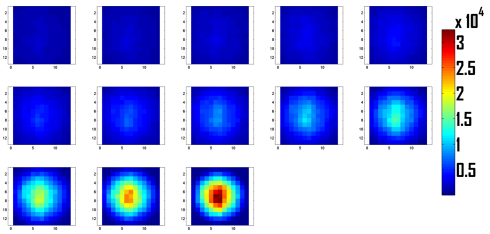
(B) σ_x



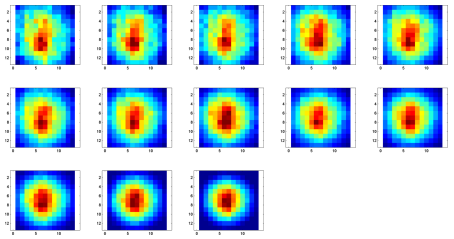
(C) σ_y



(D) ρ



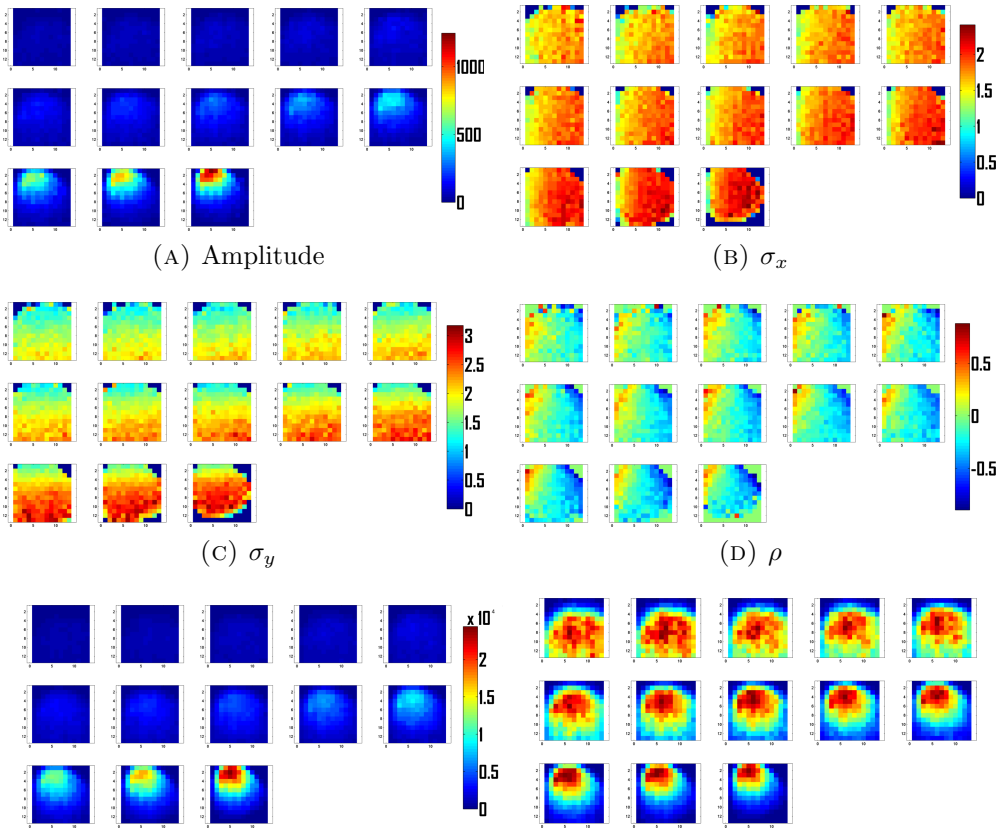
(E) Total signal under Gaussian. All images are normalized to the same range.



(F) Total signal under Gaussian. This is the same data as 4.16e but all images are individually normalized.

FIGURE 4.16: The coefficients of the Gaussian fits to the central pinhole at all voxel locations. The images in each figure correspond to slices of the FOV through the axis perpendicular to the collimator. The first image in each figure is farthest from the collimator, and the last image is nearest to the collimator. (a) Amplitude, (b) σ_x, (c) σ_y, (d) ρ, (e) Total signal under Gaussian. All images are normalized to the same range, (f) Total signal under the Gaussian. This is the same data as (e) but all images are individually normalized to help visualize the distribution.

It was determined that a pinhole projection was not detected from a voxel if a pinhole aperture had an amplitude of zero in the corresponding voxel in its 3D grid of coefficients. Adjacent voxels may have had a non-zero amplitude for the same pinhole aperture. The difficulty was how to determine where the pinhole projection amplitude dropped to zero in the space in which interpolation was being performed. Linear interpolation was initially attempted, but it was found to be unsuitable due to the distribution of the data. The effect of linear interpolation was also seen in the reconstruction of the object; a linear grid corresponding to the original data points attracted most of the counts when implementing the reconstruction algorithm. Cubic spline interpolation, using the



(A) Amplitude

(B) σ_x

(C) σ_y

(D) ρ

(E) Total signal under Gaussian. All images are normalized to the same range.

(F) Total signal under Gaussian. This is the same data as 4.17e but all the images are individually normalized.

FIGURE 4.17: The coefficients of the Gaussian fits to a peripheral pinhole at all voxel locations. The images in each figure correspond to slices of the FOV through the axis perpendicular to the collimator. The first image in each figure is farthest from the collimator and the last image is nearest to the collimator. (a) Amplitude, (b) σ_x, (c) σ_y, (d) ρ, (e) Total signal under Gaussian. All images are normalized to the same range, (f) Total signal under the Gaussian. This is the same data as (e) but all images are individually normalized to help visualize the distribution.

MATLAB™ function "*interp*3", was subsequently used due to the smoother fit. Each coefficient array of each pinhole aperture was interpolated separately. The first grid to be interpolated was the amplitude data. As previously mentioned, the data to be interpolated consisted of incomplete 3D grids of coefficients. In each 3D coefficient grid, the values in the voxels, from which no PSFs were measurable, were extrapolated from the voxels from which responses were measured. This was done using the MATLAB™ functions "*csaps*" and "*fnxtr*". Any voxels whose corresponding amplitude values were zero or negative were ignored in the final interpolated grids.

To demonstrate the effect of the function "$fnxtr$", a line profile through the FOV was selected for a single pinhole. It was known that the amplitude of the pinhole projection was zero at some locations within the FOV but the pinhole had a smooth system response. A spline was fitted separately to each of the coefficients (figure 4.18). The range on the x-axis is small to help visualize the difference between the data sets. A deviation between the fits was observed in each case except for the amplitude, because its value in the last voxel was set to zero and interpolation was performed. The effect of these under- and over-estimations of values might have been negligible in an isolated case, but these errors in interpolation could have lead to artefacts in the reconstructed image when multiple pinholes projected the FOV.

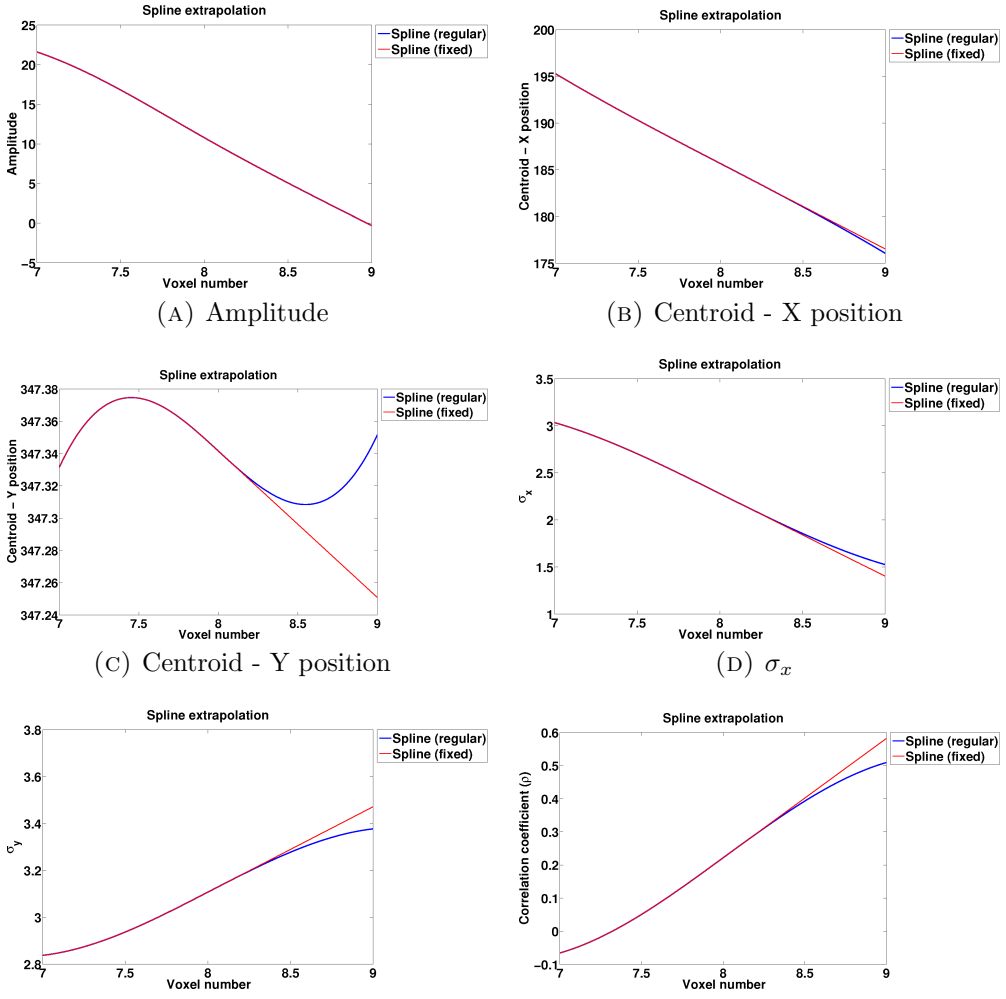
Table 4.2 shows the memory requirements for the two interpolated system matrices. Storing the Gaussian coefficients of the PSFs required $< 0.2\%$ of the memory required to store the raw projection data. The Gaussian approximation to the pinhole response will become increasingly important as the number of pixels in CCDs increases in the future. Higher-resolution sensors will require much more memory to store the raw projection data because of the number of pixels.

TABLE 4.2: Storage requirements of the system matrices with different resolutions. The storage requirements for the projections were calculated using 320 $\times$ 240 $\times$ 32-bit images.

Voxel size (mm)	# voxels	# system configurations	# matrix elements	Size (GB) with projections	Size (GB) with coefficients
0.5	$61 \times 61 \times 61$	6	1×10^{11}	408	0.7
0.3125	$97 \times 97 \times 97$	6	4×10^{11}	1.6×10^{3}	2.9

4.4.1 Validation with Experimental Data

The low-resolution calibration data were measured on a grid with spacings of 2.5 mm along the three axes. A small, finely-sampled system matrix was also measured along a line through the centre of the FOV for four of the six system configurations. These high-resolution measurements had spacings of ~ 0.16 mm.



(A) Amplitude (B) Centroid - X position

(C) Centroid - Y position (D) σ_x

(E) σ_y (F) ρ

FIGURE 4.18: The effect of regular spline extrapolation (blue), and adjusted spline extrapolation (red), on the coefficients of the pinhole fit. They were adjusted using the MATLAB^{TM} function "$fnxtr$". Data were taken for voxel numbers (x-axis) between 1 and 9, but only voxels 7-9 are shown here to help visualize the difference. No pinhole projection was detected at the 9_{th} position (amplitude = 0), and the data between the 8_{th} and 9_{th} voxels needed to be estimated for interpolation at the edge of the data.

Due to time constraints, four-times fewer counts were recorded for each voxel position in the high-resolution measurements than in the low-resolution calibration data. Gaussian fits were made to the pinhole projections in these high-resolution data, and are denoted by C_e. A high-resolution system matrix was interpolated from the low-resolution calibrated data. The interpolated coefficients are denoted by C_i. The differences between C_e and C_i for every pinhole detected in the experimental and interpolated data are an indication of the accuracy of the coefficient interpolation.

Every pinhole projection in the interpolated data was matched with its corresponding pinhole projection in the high-resolution experimental data for the system configurations examined. The RMSD between coefficients of the experimental data and the interpolated data are shown in table 4.3. Low RMSD values indicate strong agreement between the experimentally measured data and the interpolated data. The RMSD between the coefficients is lowest for the low-magnification data, most likely due to higher signal-to-noise ratio in these data. The RMSD increases with system magnification, likely due to the fewer counts detected in each pixel of the pinhole-projection clusters in the small, finely-sampled H matrix.

Repositioning errors of the source within the FOV may also contribute to the RMSD values. The motion-controller documentation indicates that the motion stages have a positioning repeatability resolution of 4 μm (*Velmex, Inc.*). The position of the source within its holder may also change slightly between measurements. These potential errors were not examined in this work, but may affect the performance of the system.

TABLE 4.3: For every pinhole projection, the Gaussian coefficients of the finely-sampled, experimentally-measured data, C_e, were compared with the coefficients of the interpolated data, C_i. The RMSD between the coefficients of every pinhole projection for four system configurations were calculated.

System Configuration	Root-Mean-Square Difference between C_e and C_i				
	$\bar{x}$ (pixels)	$\bar{y}$ (pixels)	σ_x (pixels)	σ_y (pixels)	ρ_y
1	0.1446	0.1570	0.0860	0.0850	0.1232
2	0.2587	0.2205	0.1091	0.1083	0.1353
5	0.4330	0.3485	0.2627	0.2151	0.1798
6	0.5648	0.4003	0.3905	0.3342	0.2108

4.5 Multiplexing

Multiplexing is the overlapping of projections on the detector. The measured PSFs were projected onto the detector, and the amount of pinhole projection overlap was calculated (figure 4.19). There was some overlap of pinhole projections at the edge of the FOV in the first system configuration. The effect of this small overlap was minimized due to the number of different pinhole projections and the pseudo-random pattern in which they were arranged. There may be several object distributions that result in the same projection image, but the

number of possibilities is reduced when images from each system configuration are used in the reconstruction algorithm. The amount of multiplexing increased with magnification in the system. There were up to five pinholes projecting to the same detector element in the final system configuration. The percentage of pixel multiplexing for each system configuration is shown in figure 4.20. In some system configurations there were some detector pixels that detected no signal from the FOV. These pixels were not counted when determining the percentage coverage in figure 4.20.

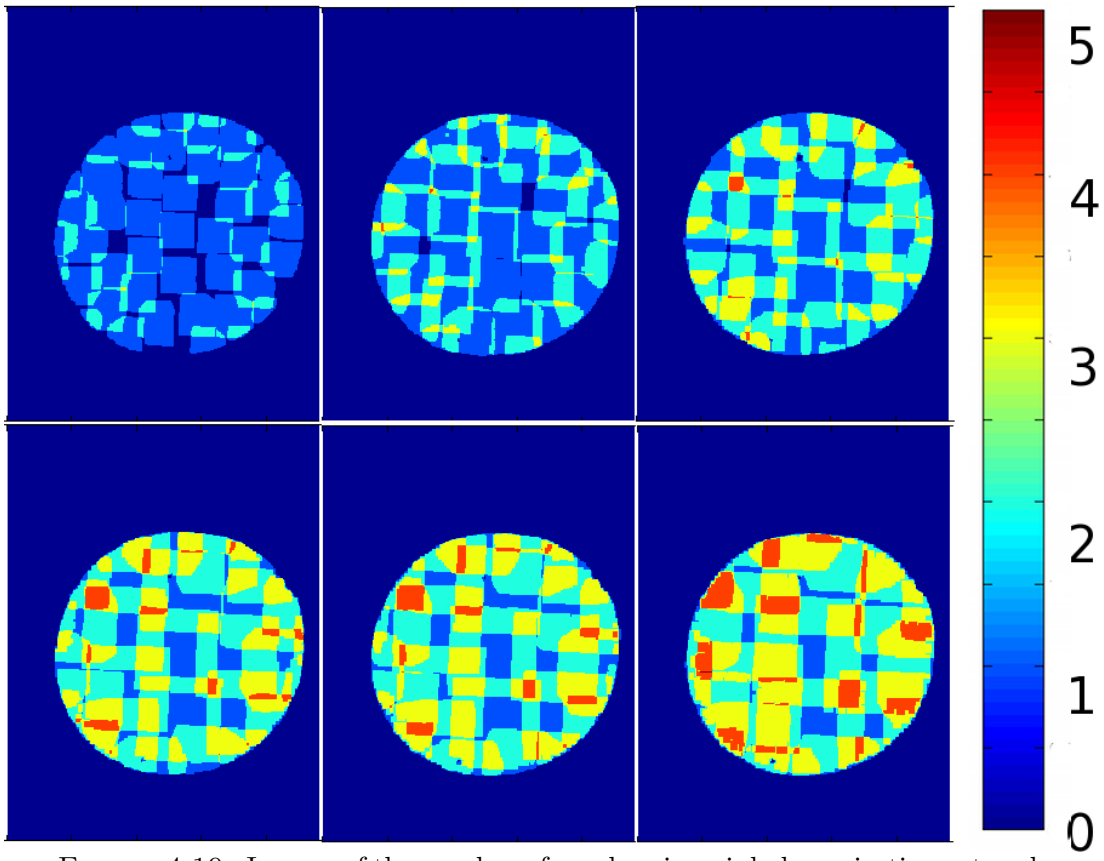


FIGURE 4.19: Images of the number of overlapping pinhole projections at each system configuration. The number of pinholes projecting to the same detector element ranges from 0 to 5 at each system configuration.

Figures 4.19 and 4.20 show that even at the second system configuration (magnification = 0.8) approximately 50% of the detector pixels detected gamma rays through at least two pinhole apertures. This percentage increased to approximately 90% in the last system configuration.

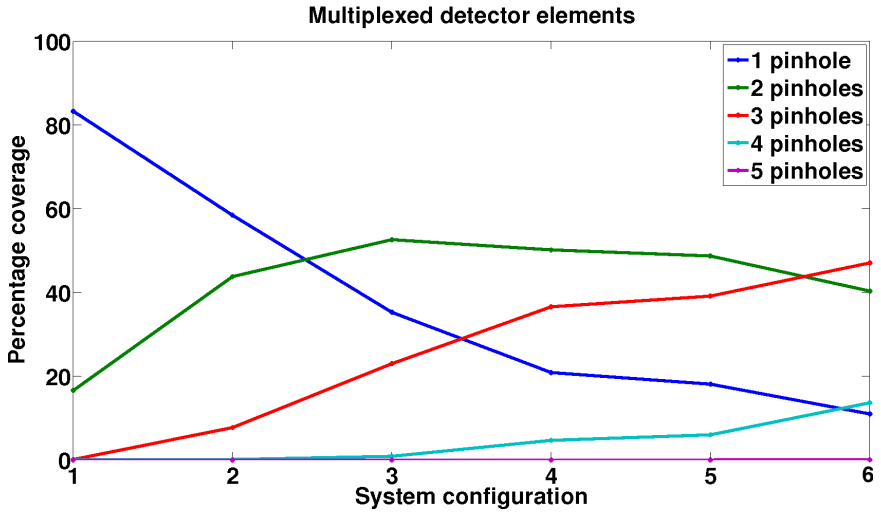


FIGURE 4.20: The percentage of pinhole-projection overlap on detector elements in each system configuration. This was calculated using the coarse PSF measurements and will slightly underestimate the amount of overlap. This graph does not account for the detector pixels that detect no signal from any pinhole projection.

4.6 Simulations using the System Matrix

As described in Barrett and Myers, 2004, the capability of detecting a signal at a specific location in a noisy background can be used to assess the performance of an imaging system (Chen, 2006). The SKE/BKE (signal-known-exactly/background-known-exactly) detection task can be used to assess the performance. Although unrealistic, this task was used as a basic evaluation of the imaging system. The images were generated by forward projecting a simulated phantom through the system matrix **H**. The phantom used in this simulation was similar to the one used in the experimental phantom experiments (section 5.2.1), but with more rods. The rods had diameters of 1.0, 1.2, 1.4, 1.6, 1.8 and 2.0 mm. The distance between the centres of the rods in each section was equal to twice the diameter of the rods in that section. This phantom was placed in the FOV against a flat background. The test was to determine the presence of a signal, $f_s(\mathrm{r})$, against a background, $f_b(\mathrm{r})$, in images generated by a linear imaging system. There were two hypothesis: the signal was absent (H_1), and the signal was present (H_2). The presence of the signal did not alter the background. The task was the discrimination between two exactly-specified signals in an image with only Poisson noise, the conditional probability density function under hypothesis

H_i is

$$pr(\mathbf{g}|H_i) = \prod_{m=1}^{M} exp(-\bar{g}_{im})\frac{\bar{g}_{im}^{g_m}}{g_m!}, \tag{4.6}$$

where $\bar{g}_{im}$ is the mean response of the m_{th} detector pixel when H_i is true. The ideal observer's test statistic is

$$\lambda(g) = \sum_{m=1}^{M} g_m ln\frac{\bar{g}_{2m}}{\bar{g}_{1m}}. \tag{4.7}$$

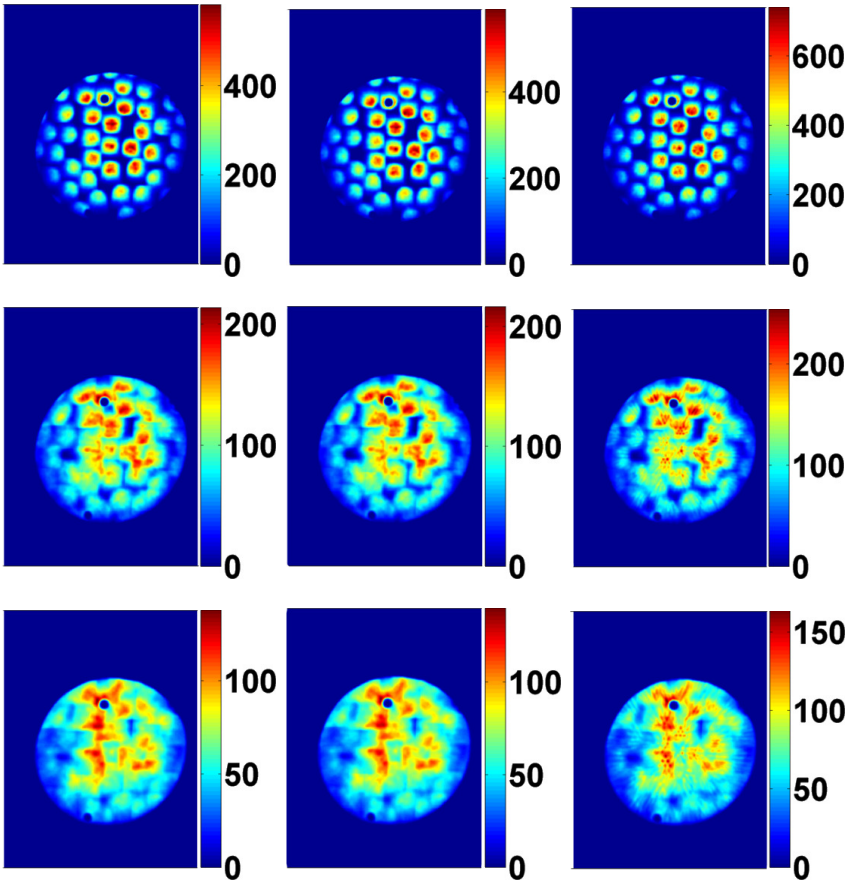
The signal-to-noise ratio associated with this test statistic is

$$SNR^2 = \frac{\left[\sum_{m=1}^{M}(\bar{g}_{2m} - \bar{g}_{1m})ln(\frac{\bar{g}_{2m}}{\bar{g}_{1m}})\right]^2}{\frac{1}{2}\sum_{m=1}^{M}(\bar{g}_{2m} + \bar{g}_{1m})ln^2(\frac{\bar{g}_{2m}}{\bar{g}_{1m}})}. \tag{4.8}$$

There was a uniform background A_b within the FOV. The signal contrast is defined as $C = A_s/A_b$, where A_s is the signal activity. The signal activity was within each rod defined by the voxelized hot-rod phantom. The values of A_b and A_s were adjusted so that approximately 10^7 counts were acquired for each simulation. The phantom was projected through the system matrix with a voxel size of 0.3125 mm (figures 4.21 and 4.22) and the phantom was reconstructed using the system matrix with a voxel size of 0.5 mm.

The synthetic-collimator images that were generated (figure 4.23) show that some of the rods can be reconstructed at each signal contrast. These reconstructions validate the reconstruction algorithm used to produce the high-resolution images.

Figure 4.24 shows that the SNR increased as data from higher magnifications were used. As the magnification of the system was increased, the sensitivity was reduced because fewer pinhole projections were incident on the detector. The relative increase in SNR reduces with each system configuration due to the reduction in sensitivity.

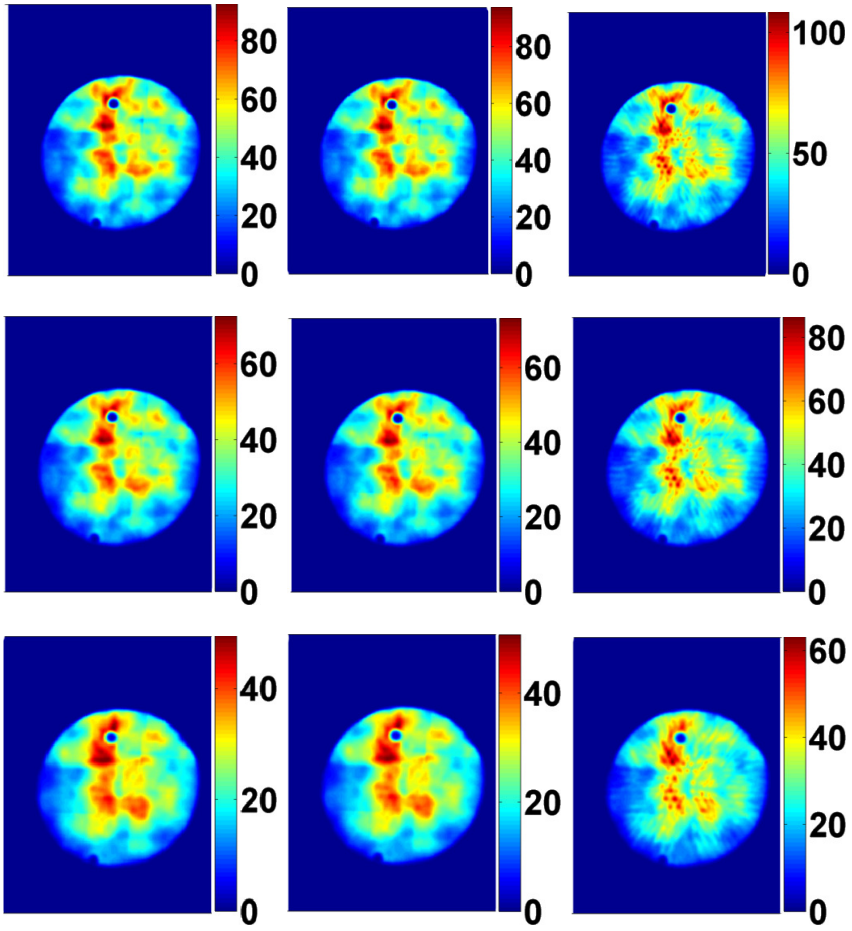


(A) Contrast = 0.1b (B) Contrast = 0.2 (C) Contrast = 0.5

FIGURE 4.21: Simulated projection data. A hot-rod phantom was projected through the system matrix at each system configuration. Objects with contrasts: (column (a)) 0.1, (column(b)) 0.2, and (column (c)) 0.5 were simulated. The projections in each column correspond to system magnifications of $\simeq$ 0.4, 0.8, 1.1.

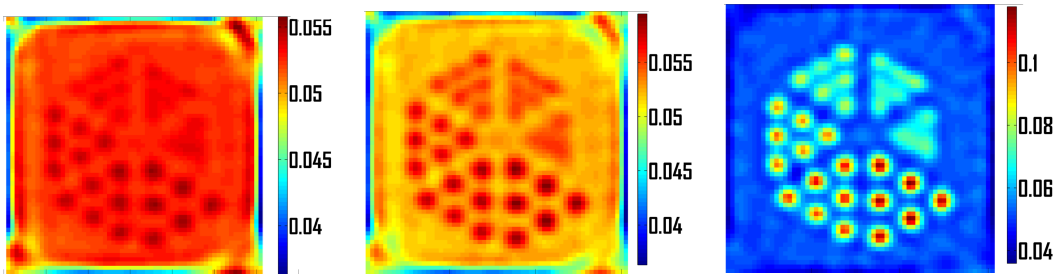
4.6.1 Reconstructions with Multiplexed Data

The effect of combining multiplexed and non-multiplexed data is demonstrated using simulated hot-rod phantom data. Two phantoms were simulated to validate the reconstruction algorithm. In both phantoms, the diameter of the rods were 1.0, 1.2, 1.4, 1.6, 1.8 and 2.0 mm, and the distance between the centers of adjacent rods was equal to twice the diameter of the rods in that section (Figure 4.25). In the first simulation, a background activity was simulated in addition to the activity in the rods. In the second simulation, the activity was restricted to the rods and a $16 \times 16 \times 2$ mm^3 uniform reservoir below the rods. Simulation 1 was designed to validate the efficacy of combining multiplexed and non-multiplexed



(A) Contrast = 0.1 (B) Contrast = 0.2 (C) Contrast = 0.5

FIGURE 4.22: Simulated projection data. A hot-rod phantom was projected through the system matrix at each system configuration. Objects with contrasts: (column (a)) 0.1, (column(b)) 0.2, and (column (c)) 0.5 were simulated. The projections in each column correspond to system magnifications of $\simeq$ 1.5, 1.6, 2.0.



(A) Object contrast = 0.1 (B) Object contrast = 0.2 (C) Object contrast = 0.5

FIGURE 4.23: Synthetic-collimator images reconstructed using the simulated projections. The voxel activity in the field of view is estimated and is integrated along the axis perpendicular to collimator face.

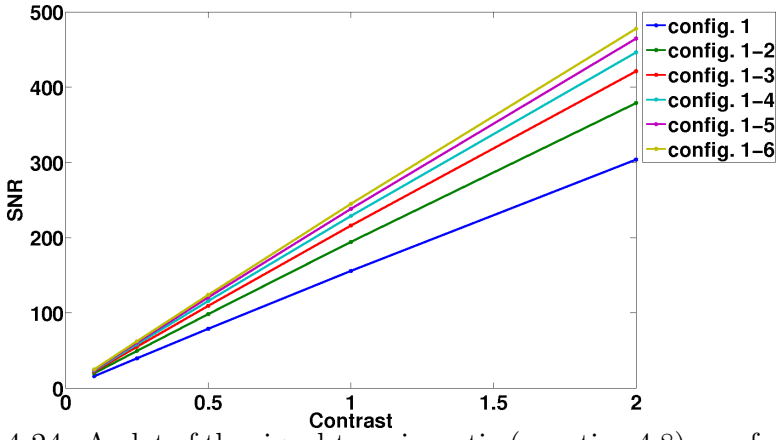


FIGURE 4.24: A plot of the signal-to-noise ratio (equation 4.8) as a function of contrast in the object. The SNR increases linearly with contrast in the object, and using more projection images increases the SNR of the data.

data. Simulation 2 was designed to test the depth resolution of the system when combining multiplexed and non-multiplexed data.

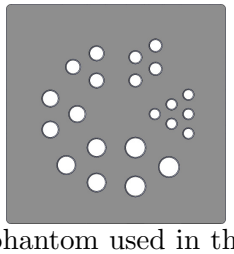


FIGURE 4.25: Image of the phantom used in the simulations. The diameters of the rods were 1.0, 1.2, 1.4, 1.6, 1.8, 2.0 mm, and the separation between the centres of the rods in each section was equal to twice the diameter of those rods. A uniform background activity within the dark grey region was also simulated for Simulation 1. The contrast between the signal in the rod and the background activity was 0.5.

4.6.1.1 Simulation 1

The phantom was voxelized to a 3D grid with cubic voxels of side-length 0.3125 mm. An interpolated system matrix with this resolution was used to estimate the projection data for six system configurations. These projection data were then used to reconstruct the object on a 61 × 61 × 61 grid with cubic voxels of side-length 0.5 mm. The phantom had a uniform background, A_b, within the FOV. The phantom was also surrounded by a region with no activity. Each voxel within the rods had an activity, A_s, in addition to the background activity. The signal contrast was 0.5. The values of A_s and A_b in each Test Case (table 4.4) were adjusted so that a total of $\sim 1.5 \times 10^7$ counts were acquired. The test cases were based on the system configurations (table 4.1) and were selected to demonstrate

the effect of using only non-multiplexed data (Test Case 1), only multiplexed data (Test Case 2), and a combination of non-multiplexed and multiplexed data (Test Cases 3 and 4).

TABLE 4.4: Simulated test cases to demonstrate the effect of combining multiplexed and non-multiplexed data. The projection data from the different system configurations (table 4.1) were used with an interpolated system matrix to reconstruct the hot-rod phantom. The total number of counts for each test case was normalized to $\sim 1.5 \times 10^7$ counts.

Test Case	System Configurations
1	1
2	6
3	1 & 6
4	1, 2, 3, 4, 5 & 6

For each test case in table 4.4, the hot-rod phantom was reconstructed using 400 iterations of the MLEM algorithm and post-smoothed with a Gaussian filter (figure 4.26). Using an nVidia GTX 580 GPU, each iteration of the MLEM algorithm took approximately five seconds per projection image.

The hot-rod phantom was reconstructed for each Test Case with, and without, Poisson noise added to the projection data (Figure 4.26). The noise-less projection data were used to verify the reconstruction algorithm, and the projection data with Poisson noise were used to examine the effect of noise on the synthetic-collimator image.

In the noise-free data, combining multiplexed data with the non-multiplexed data clearly improves the spatial resolution of the system (figures 4.26a - 4.26h). In each of the test cases that used non-multiplexed data, the phantom was clearly outlined and some of the rods were resolved. Figures 4.26b and 4.26j (Test Case 2) show that multiplexed data cannot be used in isolation to reconstruct an object, even in a noise-free system. The lowest magnification system configuration enabled reconstruction of the larger rods and the general outline of the phantom (figure 4.26a). The general outline of the phantom is not noticeable in figure 4.26b due to the extreme values in the corners of the FOV, although some of the rods appear to be resolved in the centre of the reconstruction (figure 4.26f), but multiplexing artefacts are still evident. When the high-magnification multiplexed data were combined with the low-magnification data, the smaller rods were clearly resolved (figures 4.26c and 4.26g). The 1.2 mm rods appear to be resolved when all

the data from each system configuration were used in the reconstruction (figures 4.26d and 4.26h).

Poisson noise appears to have had a major effect on the synthetic-collimator images for each test case (figures 4.26i - 4.26p). While the phantom was clearly outlined, and the largest rods were resolved, the background activity was very prominent. As expected, the multiplexing effects were visible in the reconstruction of Test Case 2 (figures 4.26j and 4.26n). Due to the low counts-per-pixel for system configuration 6, the Poisson noise appears to reduce the effectiveness of combining high-resolution multiplexed data with the low-magnification data (figure 4.26o), although some smaller rods were more clearly defined. Figures 4.26l and 4.26p show that, as predicted, using data from as many system configurations as possible produces the best estimates of the tube integrals and results in improved spatial resolution.

4.6.1.2 Simulation 2

The projection data were simulated using the same voxelization as Simulation 1. The reconstruction algorithm was stopped after only 5 iterations of the MLEM algorithm because the objectives of the simulation were achieved.

Figure 4.27 shows every second slice (0.5 mm thickness) of the reconstructed phantom when only the highest magnification projection data were used in the MLEM reconstruction (5 iterations). Some of the larger rods appear to be reconstructed in some central slices, but due to the noise elsewhere in the slices one can be not be certain that these are accurate representations. The effects of multiplexing are immediately obvious in the slices. Each of the slices are locally normalized; a global value range could not be found that demonstrated the effect of multiplexing while retaining good image contrast in each of the slices.

Figure 4.28 shows every second slice (0.5 mm thickness) of the reconstructed phantom when data from all the system configurations were used in the MLEM reconstruction (5 iterations). The rods are clearly reconstructed in many of the slices. The small reservoir is also outlined, but it is not very well reconstructed; a small uniform activity distribution is difficult to reconstruct using limited-angle tomography. The rods in the slices farthest from the collimator are not well resolved. The lateral resolution was expected to be quite poor (> 2 mm) in the

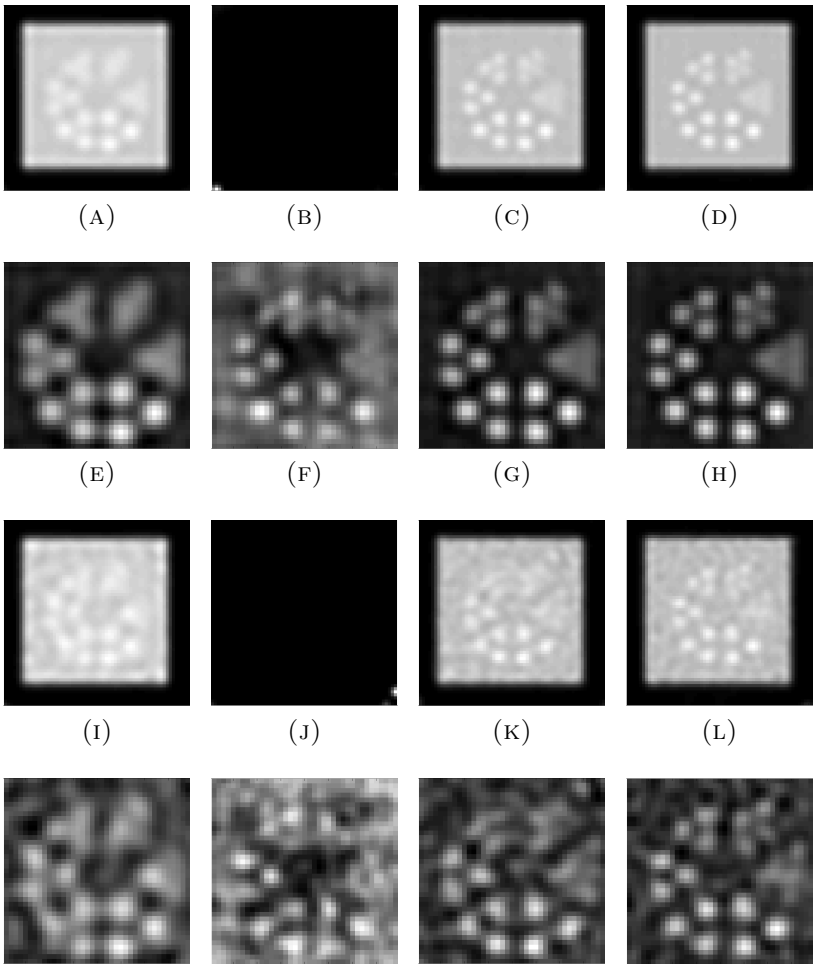


FIGURE 4.26: The synthetic-collimator images for the four test cases of Simulation 1: (Column 1) Test Case 1, (Column 2) Test Case 2, (Column 3) Test Case 3 and (Column 4) Test Case 4. The object, a hot-rod phantom with uniform background, was reconstructed using MLEM and the activities in the voxels were summed along the axis perpendicular to the collimator face. (a) - (d) Synthetic-collimator images of the FOV reconstructed using noise-free projections. (e) - (h) The edges of (a) - (d) have been removed to highlight the contrast between the rods and the background. (i) - (l) Synthetic-collimator images of the FOV reconstructed using projections with Poisson noise. (m) - (p) The edges of (i) - (l) have been removed to highlight the contrast between the rods and the background.

farthest slices. The rods should not actually be visible in the first few slices but have been partially reconstructed due to the poor resolution of the system along the z axis. Errors at the edge of the reconstruction could also be caused by small amounts of unintended multiplexing in the "non-multiplexed" data.

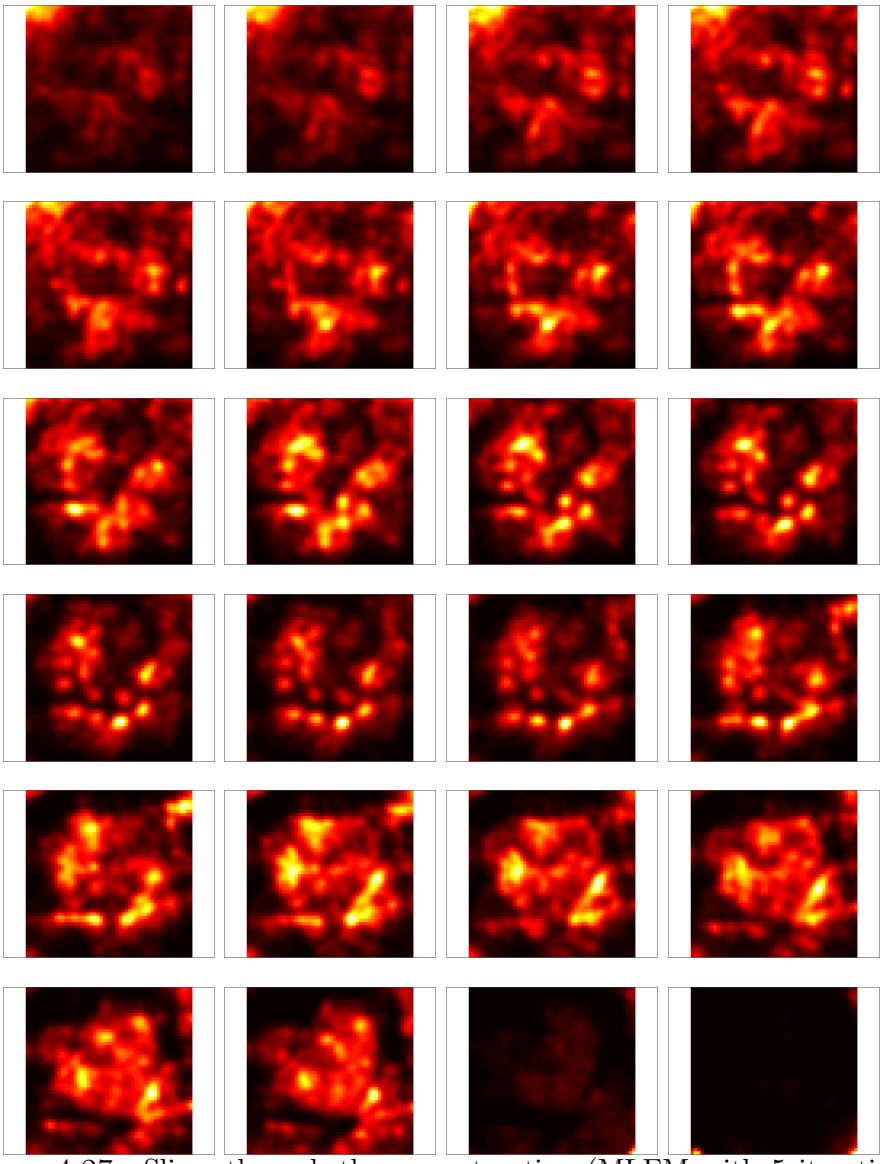


FIGURE 4.27: Slices through the reconstruction (MLEM with 5 iterations) of the simulated hot-rod phantom using only the sixth system configuration (magnification $\simeq 2.0$), which has many overlapping projections. The effects of multiplexing are immediately obvious. These images are all locally normalized; a global colour range could not be found that demonstrated the effect of multiplexing while retaining good image contrast in each of the slices.

4.7 Fourier Crosstalk Matrix and Spatial Resolution

The Fourier crosstalk matrix fully describes all deterministic properties of the imaging system and provides a summary measure of resolution of the hardware alone, averaged over the FOV. This section provides a short review and mathematical background of the Fourier crosstalk matrix. A more detailed discussion

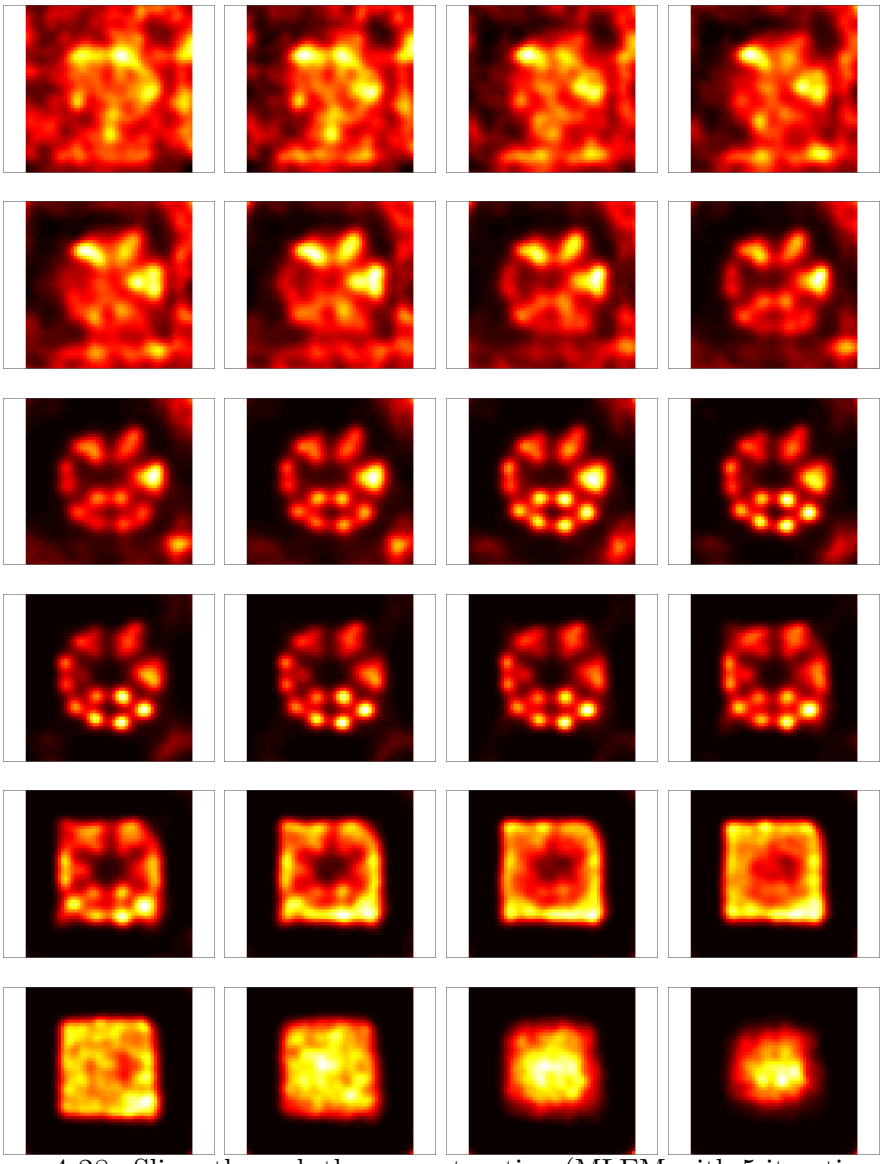


FIGURE 4.28: Slices through the reconstruction (MLEM with 5 iterations) of a simulated hot-rod phantom using all the system configurations (both non-multiplexed and multiplexed data). The phantom appears to have been accurately reconstructed. These images are all locally normalized so they can be compared with those in figure 4.27.

can be found in Barrett et al., 1995. From equation 4.1, the m_{th} detector measurement is given by

$$g_m = \int_S h_m(r) f(r) dr, \tag{4.9}$$

where S is the region defined by the support function $S(r)$, whose value is 1 if r $\in$ S and 0 otherwise. The object, $\mathbf{f}$, can be represented by a Fourier series

$$f(r) = \sum_{k=-\infty}^{\infty} F_k \Phi_k(r), \tag{4.10}$$

where

$$\Phi_k(r) = e^{2\pi i \rho_k \cdot r} S(r), \tag{4.11}$$

is the k_{th} Fourier basis function. This vector index **k** spans an infinite set of integers (k_x, k_y, k_z), and the wavevector ρ_k has the values

$$\rho_k = \frac{k}{L}, \tag{4.12}$$

where L is the width of the cubic region enclosing the space. This gives

$$g_m = \sum_{k=-\infty}^{\infty} F_k \Psi_{mk}, \tag{4.13}$$

where

$$\Psi_{mk} = \int_S h_m(r) e^{2\pi i \rho_k \cdot r} dr. \tag{4.14}$$

The detector sensitivity function, $h_m(r)$, describes the response of the m_{th} detector to a point source at r. An element of the Fourier crosstalk matrix is defined by

$$\beta_{kk'} = \sum_{m=1}^{M} \Psi^*_{mk} \Psi_{mk'}, \tag{4.15}$$

where M is the total number of detector elements. The diagonal element, β_{kk}, is the squared norm of the data when the object is a single Fourier-series component, and it quantifies how strongly a 3D spatial frequency contributes to the data. For tomographic systems it has been observed that β_{kk} falls off approximately as the reciprocal of the spatial frequency (Barrett et al., 1995; Kim et al., 2006). A factor of $|\rho_k|$ is included to get a summary measure of the spatial resolution of the hardware, and the equivalent MTF^2 is given by

$$\mathrm{MTF}^2_{\mathrm{eq}} = |\rho_k| \, \beta_{kk}. \tag{4.16}$$

If $\beta_{kk} = 0$, the basis function Φ_k(r) makes no contribution to the data and can not be recovered by post-processing. It is found that the width of the MTF^2_{eq}, defined by its full-width half-maximum (FWHM), is a direct measure of the response of the system hardware to the Fourier components which fill the FOV. The resulting width is in units of spatial-frequency and is approximately Gaussian, so its Fourier transform is also approximately Gaussian. The FWHM of the resulting space-domain function along the three axes gives a measure of the 3D spatial resolution

of the system, and can be used to compare tomographic systems. A Gaussian-distributed MTF_{eq} along the j axis can be written as

$$\mathrm{MTF}_{eq}(j) = A_0 exp(-\frac{j^2}{2\sigma^2}), \tag{4.17}$$

and its Fourier transform as

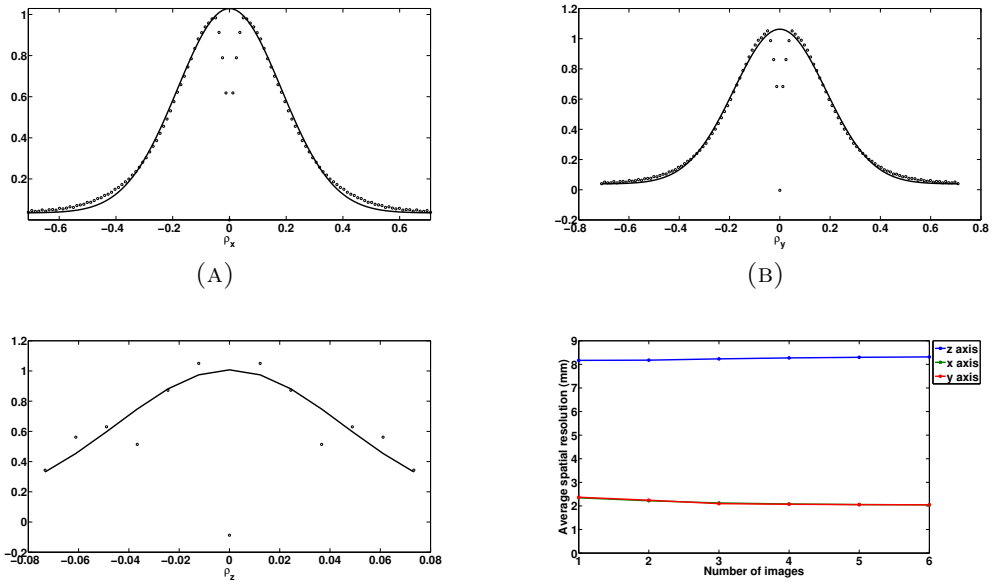
$$F\{MTF_{eq}(j)\} = A_0\sqrt{2\pi\sigma^2}exp(-2\pi^2\sigma^2\epsilon^2). \tag{4.18}$$

The FWHM of $\mathrm{MTF}_{eq}(j)$ is $\sigma\sqrt{8ln(2)}$, and that of $F\{\mathrm{MTF}_{eq}(j)\}$ is $\sqrt{2ln(2)/\pi^2}/\sigma$. The spatial resolution of the system along the j axis is

$$\text{Spatial Resolution} = \frac{4ln(2)}{\pi}\frac{1}{\mathrm{FWHM}[\mathrm{MTF}_{eq}(j)]}. \tag{4.19}$$

The Fourier lattice has a grid of spacing 1/L, and a maximum frequency of $1/(2\triangle H)$ in each direction, where $\triangle H$ denotes the sample spacing. The Fourier lattice should be as finely-sampled as possible, and have the maximum frequency possible. This is achieved by finely sampling the object space when measuring **H**. The effect of sampling is investigated by using two different voxel sizes (0.3125 mm, and 0.5 mm) in calculating the Fourier crosstalk.

The Fourier crosstalk matrix was calculated using the "cufft" library in CUDA. Due to memory requirements, it was a complicated function to evaluate computationally. The sensitivity of each detector element needed to be calculated individually and transformed to Fourier space. The plots of the MTF_{eq}, when all six system configurations are used, are shown in figure 4.29. The average spatial resolution, as each new system configuration was included in the calculation, is shown in figure 4.29d. In each of the plots, the central few data points were omitted from the fitting procedure. A Gaussian distribution was fit to the data points, and the FWHM was calculated. The baseline of the data was accounted for when estimating the FWHM of the Gaussian fit. The results of the fit are shown in figure 4.29. The average spatial resolutions along the x and y axes ($\sim$ 2 mm) were very similar, as expected, and resulted in values close to that predicted for lateral resolution by traditional pinhole theory (Accorsi and Metzler, 2005). As this was a limited-angle tomography system, it was expected that the spatial resolution in the z axis should be lower than in the x and y axes. It was found that the average spatial resolution along the z axis was approximately 8



(A) (B) (C) (D)

FIGURE 4.29: The normalized $\mathrm{MTF_{eq}}$ along the: (a) x axis, (b) y axis, and (c) z axis. These were calculated using the 97 × 97 × 97 cubic grid; the results with the 61 × 61 × 61 grid are very similar. A Gaussian function (solid line) has been fitted to the MTF_{eq} data (dots). The centre of each plot deviates from an approximate Gaussian due to $|\rho_k|$, and was ignored in the fitting procedure. The average spatial resolution of the system is shown in (d), where the x axis denotes the number of system configurations used in the calculation. The average lateral resolution of the system improved as data from high-magnification configurations were included in the calculation but the resolution along the z axis did not change.

mm. As noted earlier, these values represent the average resolution over the FOV without considering the reconstruction algorithm. By using a statistical iterative reconstruction algorithm, such as MLEM, it is possible to get better resolution at certain slices through an object than is estimated by the Fourier crosstalk matrix. Results of a hot-rod phantom imaging study are presented in section 5.2.1 to confirm this.

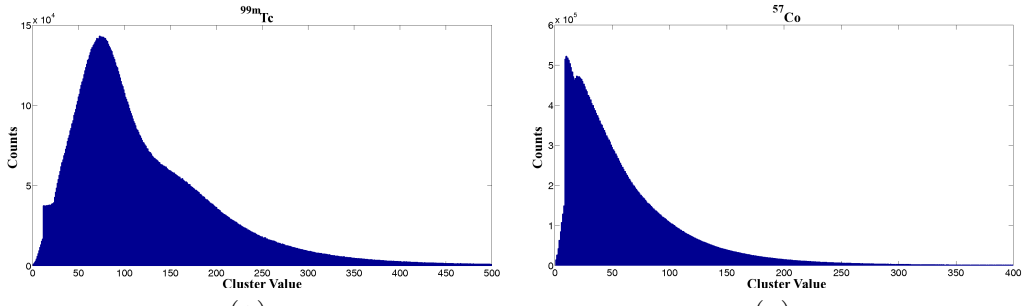
4.8 Energy Resolution

The energy resolution is examined using the PSF data and the phantom data (figure 4.30). For the PSF measurements, a ^{57}Co (122 keV) source was translated through the FOV. The hot-rod phantom measurements used a ^{99m}Tc (140 keV) source. Individual events were extracted from the listmode data, and the total sum within each cluster was used to generate the energy histograms. Energy

resolution was not expected to be good due to the short exposure time of each frame and the 8-bit imaging mode used. An estimate of the gamma-ray interaction parameters is improved with each scintillation photon detected (Barrett et al., 2009). In this work, the short exposure time may have resulted in only a fraction of the light emitted in each scintillation being detected. The difference in exposure times between the ^{57}Co work and the ^{99m}Tc work is the reason for the different spectra. The energies of the primary gamma rays of both sources are very similar so it was expected that the spectra would be similar, but this was not the case. Due to high activity of the ^{57}Co source, the exposure time for each frame was reduced. This reduced exposure time reduced the number of scintillation photons acquired in each cluster.

The CCD was operated in 8-bit mode for this work. In this mode, a pixel could only store one of 256 different values for a given frame. The more *bits* used to store the recorded values, the larger the range of the values that can be recorded. It is possible that a larger range of values would result in improved discrimination of energy levels.

In this work, every detected event was used to estimate the activity distribution in the object. The data were not filtered using a photo-peak window.



(A) (B)

FIGURE 4.30: SyntheticSPECT energy spectra acquired using a thin Lanex scintillator using (a) ^{99m}Tc (140 keV), and (b)^{57}Co (122 keV). The histograms were generated by integrating the signal within a 5 × 5 region around the centroid of each cluster.

4.9 Conclusion

The SyntheticSPECT imaging system was calibrated by recording the system response to an isotropically emitting point source moving through the FOV in

a 3D grid pattern. The system response was measured for six different system configurations. A projection image was recorded for each voxel position in object space. The pinhole projections in each image were identified and associated with their respective pinhole apertures. The algorithm developed for the pinhole identification and tracking is applicable to any multi-pinhole system, and there is no upper bound to the number of pinholes that can be tracked. A 2D elliptical Gaussian fit was made to each of the projections, and the coefficients of the fits were stored. The coefficients corresponding to each pinhole aperture were grouped together and were used to interpolate the system matrix. Forty-six pinholes projected the FOV onto the detector. The FOV common to each pinhole was smaller than the measured volume. Spline interpolation and extrapolation of the coefficients was done at the edges of the FOV where the pinhole responses were not detected.

The interpolation was validated with high-resolution calibration data. This validation may be improved by increasing the number of counts acquired in the high-resolution data. Two interpolated system matrices with different resolutions were generated. The interpolated system matrices were used to determine the multiplexing in the projection data for each system configuration. The system matrices were also used to validate the concept of the synthetic collimator through simulations. The simulations demonstrated the concept of synthetic collimation and the efficacy of combining multiplexed and non-multiplexed data. Multiplexed data could only be used to improve the spatial resolution of the reconstruction if they were combined with non-multiplexed data.

From the simulations, the lateral spatial resolution of the system appears to be approximately 1.2 - 1.4 mm. This agrees well with pinhole collimation theory (section 2.4.1). The resolution may be improved by reducing the pinhole diameter or increasing the magnification of the system. Although becoming increasingly common in preclinical SPECT systems, sub-millimeter resolution was not the objective of this work. The objective of SyntheticSPECT was to synthesize a parallel-projection image.

The Fourier crosstalk matrix was also calculated using the system matrix. The crosstalk matrix provides a summary measure of the average spatial resolution of the system over the entire field of view. The resolution of the synthetic collimator system varied with axial position in the FOV (z) because of the limited-angle tomography. The average lateral resolution of the imaging system was estimated

to be 2 mm, and the resolution in the z axis was estimated to be 8 mm. The peak system resolution should be better than that estimated by the Fourier crosstalk matrix.

CHAPTER 5

Imaging with BazookaSPECT and SyntheticSPECT

Two imaging systems were built during the course of this work. The first imaging system, BazookaSPECT, was used for the work at the National University of Ireland Galway (NUIG). This system was used to investigate a novel therapy for breast cancer. The second imaging system, SyntheticSPECT, was designed and built by the author at the Center for Gamma-Ray Imaging (CGRI), Department of Radiology, University of Arizona, Tucson, AZ, USA. In this chapter, BazookaSPECT will refer to the system at NUIG, and SyntheticSPECT system will refer to the system at CGRI.

Section 5.1 outlines the results of the collaboration with the Regenerative Medicine Institute (REMEDI), NUIG. For this work, the software and hardware for the BazookaSPECT system were provided to the author by the collaborators at CGRI, but the design of the experimental imaging work undertaken with the system was attributable to the author.

Section 5.2.1 presents results of hot-rod phantom studies carried out on the SyntheticSPECT system. One characteristic of a hot-rod phantom is the distance between the centre of the rods in each section of the phantom is twice the diameter of the rods in that section. The phantom should not be confused with a Derenzo phantom. The distance between the centre of the rods in each section of a Derenzo phantom is three times the diameter of the rods in that section. The hot-rod

phantom is more difficult to reconstruct for this reason. Further details of the hot-rods used in the imaging studies are given later in this chapter.

Section 5.2.4 presents results from an initial small-animal imaging study conducted with SyntheticSPECT. A full-body image was acquired by translating the mouse through the FOV of the detector.

The data were reconstructed with MLEM and POSEM algorithms. The data acquired with the BazookaSPECT were reconstructed using a computational system matrix and MLEM code written by Dr. Don W. Wilson. This system matrix assumed a perfect pinhole, and was calculated based on the geometry of the system. The system matrix for SyntheticSPECT was experimentally measured by the author and it has been extensively discussed in chapter 4. The SyntheticSPECT system matrix was validated experimentally with a hot-rod phantom.

The SyntheticSPECT system used two different system matrices. Reconstructions were initially performed using a system matrix with a voxel size of 0.3125 mm. This was chosen as it was recommended that the voxel size should be at least half the size of the smallest object that was expected to be resolvable. Initial studies showed that this high voxel resolution was unnecessary, and that a voxel resolution of 0.5 mm would suffice. This corresponds to a voxel-space of 61 $\times$ 61 $\times$ 61 voxels. This was one quarter the size of the higher-resolution voxel space, and so reconstructions were much less computationally expensive.

Technetium-99m (^{99m}Tc) was used for every experiment in this chapter. This isotope has a half-life ($t_{\frac{1}{2}}$) of 6 hours, and emits gamma rays at 140 keV. The decay time was accounted for during the experiments by extending the acquisition time of certain images, or in the reconstruction step by adjusting the counts recorded in the projection images. The projection counts, or the acquisition time, were adjusted based on the formula for radioactive decay

$$N = N_0 e^{-\lambda t}, \tag{5.1}$$

and

$$\lambda = \frac{ln(2)}{t_{\frac{1}{2}}}, \tag{5.2}$$

where N_0 is the initial activity, λ is the decay constant, and t is the time that had elapsed.

^{99m}Tc binds to many biologically-active substances that make it suitable for different imaging applications. The phantom studies used ^{99m}Tc-pertechnetate. The animal studies in NUIG also used ^{99m}Tc-pertechnetate, and the animal study in CGRI used ^{99m}Tc-MDP. The radionuclide was typically administered to the animal 2-3 hours prior to imaging. This was sufficient time for the tissues to accumulate the pharmaceutical. ^{99m}Tc-MDP is used as a skeletal-imaging agent, and is widely used in diagnosing and monitoring a variety of bone diseases. Pertechnetate is useful because it can be substituted for iodine in the sodium iodide symporter (NIS) channel. This channel was the focus of the work undertaken by REMEDI (Dwyer et al., 2011).

5.1 BazookaSPECT

The BazookaSPECT imaging system was used to investigate the mesenchymal stem cell-mediated delivery of the sodium iodide symporter. The author contributed to the design of the imaging section of the experiment and to the data processing and analyses. The handling of the small animals and the verification of the imaging data were carried out by the collaborators at REMEDI.

The system consisted of a rotational stage, a collimator with a 1-mm keel-edge pinhole aperture, and a BazookaSPECT detector. At NUIG, the maximum permissible dose of ^{99m}Tc-pertechnetate given to a small animal was 74 MBq. The small animals used in this work were injected with this dose at the beginning of each imaging session. The author monitored the decay of the isotope and adjusted the image acquisition parameters accordingly. The parameters controlled by the author were the image magnification, the field of view, the acquisition time and the rotation steps. The author also used the projection data to reconstruct the object using an MLEM algorithm (provided by Dr. Don Wilson at CGRI) and a computational system matrix. The reconstructed images were analyzed by the author to determine the activity in the regions of interest.

Mesenchymal stem cells (MSCs) are nonhematopoietic multi-potent cells that can be isolated and expanded from a number of sources, including bone marrow and adipose tissue (Bieback et al., 2008; Digirolamo et al., 1999). MSCs play a primary role in tissue regeneration and they have the ability to migrate to the site of multiple tumour types (Spaeth et al., 2008). MSCs have the ability to bypass

host immune surveillance which makes them an attractive vehicle for tumour-targeted delivery of therapeutic agents. This is promising for metastatic disease in particular, as many cancer-related deaths occur as a result of complications of metastasis. The ability to dynamically image MSC migration and engraftment in real time to confirm tumour targeting is essential to understanding of MSC-mediated delivery of therapeutic agents. The timeline of the *in vivo* experiments is given in figure 5.1.

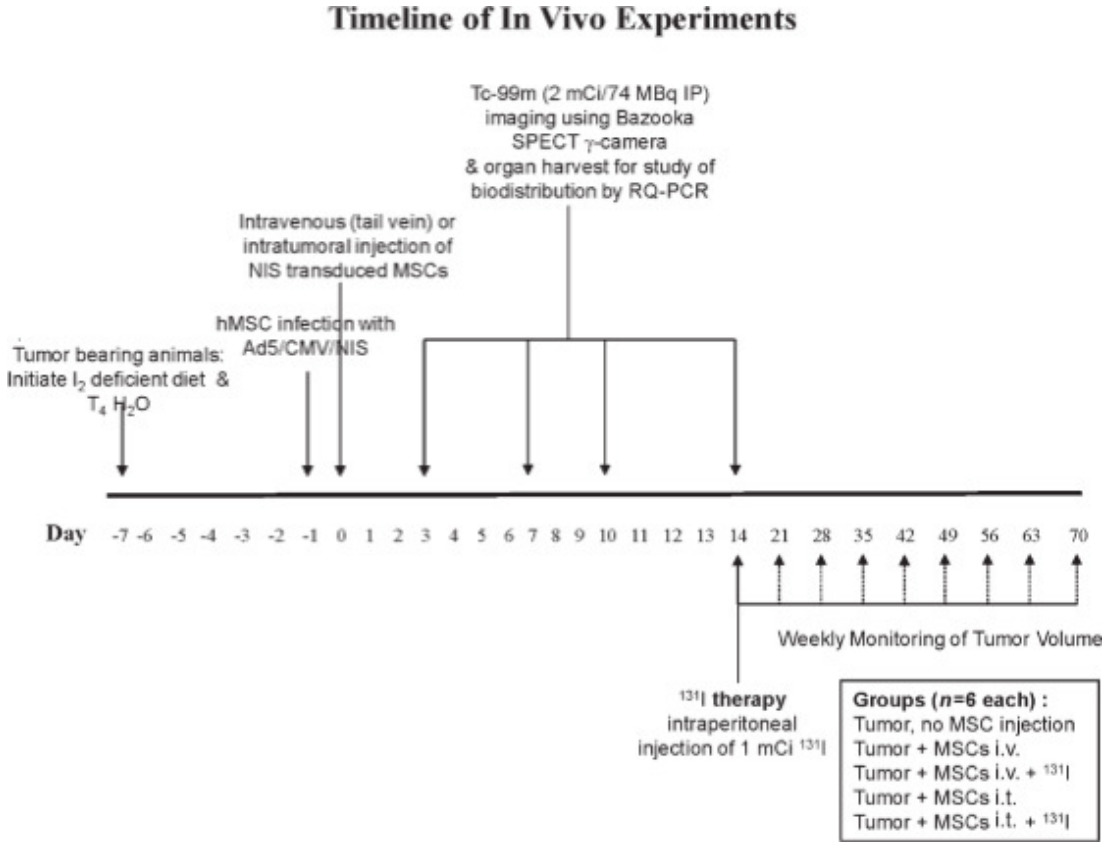


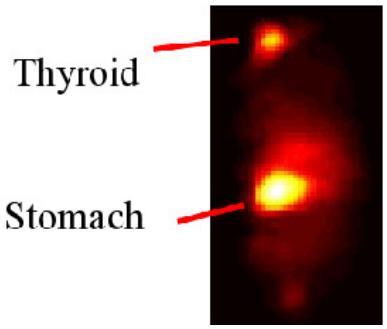
FIGURE 5.1: Schematic representation showing timeline of *in vivo* mesenchymal stem cell-sodium iodide symporter imaging and therapy experiments. Abbreviations: CMV, cytomegalovirus; hMSC, human mesenchymal stem cell; i.t., intratumoral; i.v., intravenous; MSC, mesenchymal stem cell; NIS, sodium iodide symporter; RQ-PCR, relative quantitative-polymerase chain reaction. (Reproduced from Dwyer et al., 2011.)

The ability of the thyroid gland to concentrate iodide is based on expression of NIS and forms the basis for radiolabelled iodide imaging and treatment of thyroid disease. This enabled non-invasive imaging of uptake of ^{99m}Tc or ^{123}I prior to administration of a therapeutic dose of ^{131}I. A major problem of using NIS expression for imaging is the lack of tumour targeting. This was overcome by using MSCs as cellular vehicles. NIS-transduced MSCs allowed for non-invasive imaging to track MSC engraftment at the tumour site.

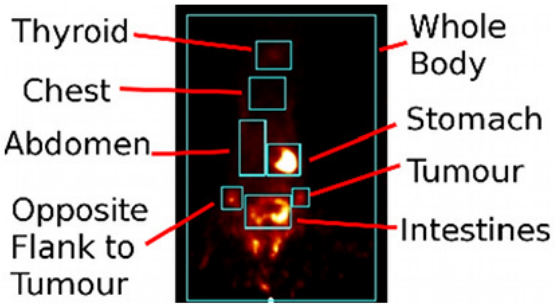
The objective of this work was to introduce functional NIS expression into MSCs and determine whether systemic administration to tumour-bearing animals would result in sufficient engraftment to allow for radionuclide imaging. 3-14 days after administration of MSC-NIS, tumour-bearing animals were given an intraperitoneal injection (i.p.) injection of 2 mCi/74 MBq ^{99m}Tc-pertechnetate. After 1 hour the animals were anaesthetized with a ketamine/xylazine cocktail and images were acquired with the BazookaSPECT. Tomographic data were taken of the control mouse, and single pinhole-projection images were taken of the animals with tumours. SPECT data were acquired from 30 images in 12 degree steps over a full rotation, and the volume data were reconstructed with 0.5 mm voxels. The acquisition time for the first image was two minutes, with each subsequent acquisition time adjusted to account for radioactive decay. The planar images were 15 minute acquisitions. The thyroid and stomach, which were known to uptake the radioisotope used in this study, were used as anatomical markers. The location of these objects can be seen in figure 5.2. Whole-body images were required in order to acquire data on the tumour and the thyroid. Due to the size of the mice in this study, the centre of rotation of each mouse was approximately 60 mm from the pinhole. The magnification was adjusted to fill as much of the detector as possible.

In the control mouse that had received no injection of MSC-NIS, a pattern of uptake correlating with native expression was observed, with ^{99m}Tc concentrated in the thyroid/salivary gland and stomach (figure 5.2a). Following intravenous (i.v.) administration of MSC-NIS, an image of the animal intestines was observed on day 3 (figure 5.2b), with some diffuse uptake also seen in the chest area and a weak signal from the right flank tumour. On Day 14, uptake of tracer was visible at the site of the tumour with a significant reduction in accumulation in non-target tissue observed (figure 5.2c). Based on region-of-interest (ROI) readings, levels of tracer accumulation in areas of interest were expressed as a percentage of total administered dose. Levels of accumulation in the thyroid and stomach remained similar at day 3 (1% thyroid, 14.7% stomach) and day 14 (2.3% thyroid, 15.9% stomach). At day 3, 18% of tracer was accumulated in intestines, and had reduced to non-significant levels (<1%) by day 14. Accumulation at the tumour site was estimated at 1.2% on day 3, increasing to 9.4% by day 14 (figure 5.2c).

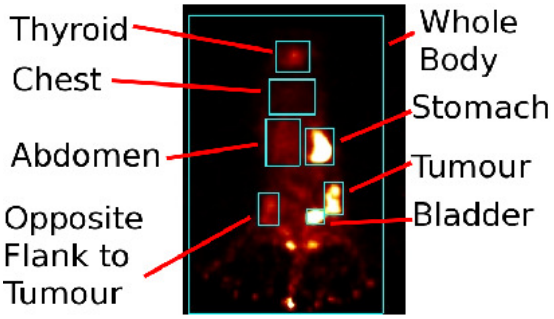
Approximately 3-7 days after i.v. injection of MSC-NIS or adenovirus alone (AdNIS), animals were sacrificed and organs were harvested for detection of NIS



(A)



(B)



(C)

FIGURE 5.2: Each mouse was imaged approximately 2 hours after injection of 74 MBq ^{99m}Tc. Three images of mice used in the study: (a) SPECT data were acquired of the control mouse. Uptake is concentrated in the thyroid/salivary gland and stomach. This image shows a single slice through the centre of the mouse. (b) On Day 3, a single projection image was taken. The intestines are observed, and a weak image of the right flank tumour is visible. (c) On Day 14, a single projection image was taken. Uptake of tracer was visible at the site of the tumour with a significant reduction in accumulation in non-target tissue observed.

expression by relative quantitative-PCR (RQ-PCR) (table 5.1). The species specific human endogenous control *MRPL19* was included as an indicator for the presence of engrafted MSCs in tissues. *MRPL19* was detected in tumours established using human cells but was not detected in any other tissue when virus alone was administered. There was no detectable human-NIS (hNIS) expression at the tumour site at day 7. 3 days after administration of MSC-NIS, hNIS was

detected in tumour, heart, lung, liver, and large intestine tissue. MRPL19 was also detected in these tissues, indicating the presence of engrafted human MSCs. At day 7 the level of NIS/MRPL19 was depleted in the heart, lungs and liver, while the level was maintained at the tumour site.

TABLE 5.1: Detection of human *NIS* and *MRPL19* expression in animal tissues 3 and 7 days after intravenous administration of either naked virus (Ad-NIS) or adenovirus infected mesenchymal stem cell-sodium iodide symporter.

	Intravenous AdNIS				Intravenous MSC-NIS			
	hNIS		*hMRPL19*		*hNIS*		*hMRPL19*	
Tissue	Day 3	Day 7	Day 3	Day 7	Day 3	Day 7	Day 3	Day 7
Tumour	-	-	+	+	++	++	++	++
Heart	+	-	-	-	++	+	++	+
Lungs	+	-	-	-	++	+	++	+
Liver	+	-	-	-	+	-	-	-
Spleen	-	-	-	-	-	-	-	-
Kidneys	-	-	-	-	-	-	-	-
Stomach	-	-	-	-	-	-	-	-
Small Intestine	-	-	-	-	-	-	-	-
Large Intestine	-	-	-	-	+	+	+	+

-,Not detectable; +, low levels of gene expression detected; ++, more robust gene expression detected.
Abbreviations: AdNIS, adenovirus-sodium iodide symporter; hNIS, human sodium iodide symporter; MSC, mesenchymal stem cell; NIS, sodium iodide symporter.

Investigation of MSC-NIS biodistribution following systemic infusion revealed engraftment in the lungs, heart, intestine, and tumour within days of administration. At later time points, detection of engraftment persisted in the tumour while it was reduced elsewhere. The RQ-PCR data agreed well with the imaging data. The imaging data at day 14 confirmed engraftment of MSC and functional NIS expression at the tumour site, and reduced levels of tracer in non-target tissues. This provided grounds to proceed to ^{131}I therapy. The results of the different treatments are shown in figure 5.3. In animals that received an i.v. injection of MSC-NIS without treatment, the mean tumour volume was the largest. In animals that received intratumoral MSC-NIS the tumour volume indicated no effect of ^{131}I therapy. In animals that received i.v. MSC-NIS injection followed by ^{131}I therapy, the tumour size was approximately 75% smaller than the control tumours. This effect could be improved with a repeat dose of MSC-NIS and ^{131}I as it has been shown that tumour irradiation increases MSC engraftment (Klopp et al., 2007).

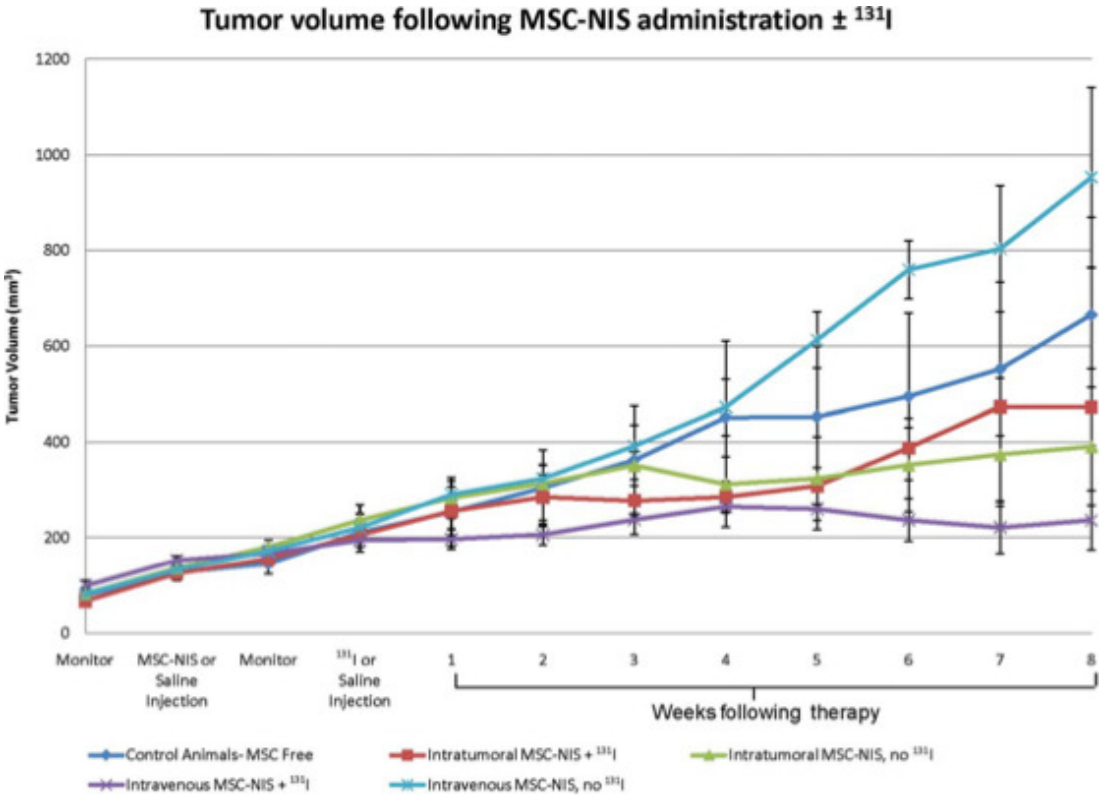


FIGURE 5.3: *In vivo* radioiodine therapy of MDA-MB-231 breast tumour xenografts 14 days after injection of mesenchymal stem cell (MSC) sodium iodide symporter (NIS). Animals received an intratumoral or intravenous injection of NIS-expressing MSCs followed by an i.p. dose of either ^{131}I or saline 14 days later. Abbreviations: MSC, mesenchymal stem cell; NIS, sodium iodide symporter. (Reproduced from Dwyer et al., 2011.)

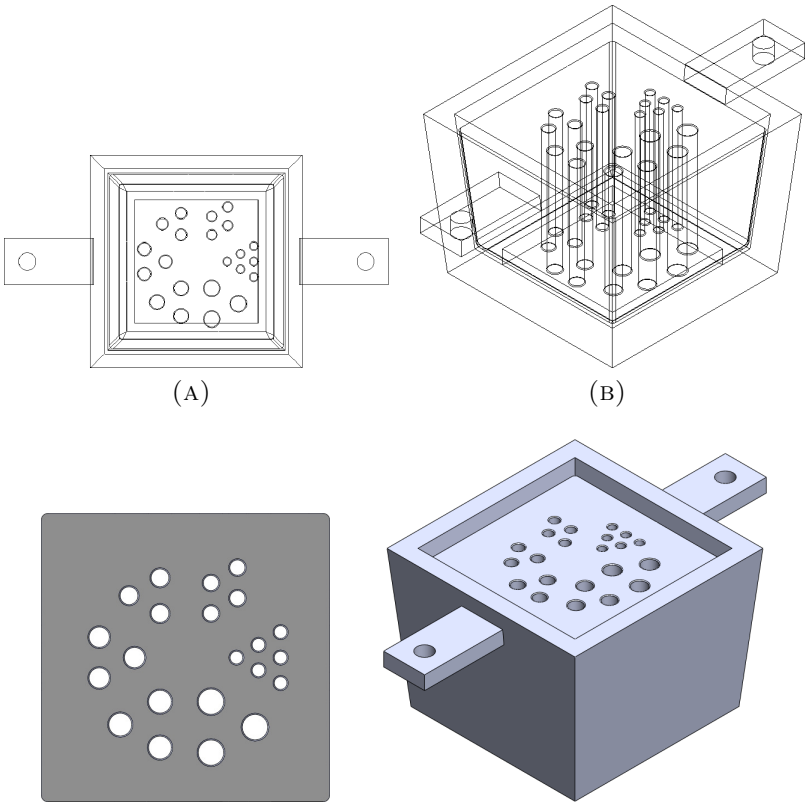
5.2 SyntheticSPECT

The SyntheticSPECT system was designed to implement the concept of synthetic collimation. This method of collimation would be ideal for REMEDI's application: detection of stem-cell engraftment in tumours on the flank of a small animal. The system was calibrated (section 4.1) and the reconstruction algorithm was validated with simulated projection data (section 4.6). The system is validated experimentally in this section. The work in this section was carried out independently by the author.

5.2.1 Hot-Rod Phantom

It is common to reconstruct a hot-rod phantom to demonstrate the performance of a SPECT system (Funk et al., 2006; Have et al., 2008). A hot-rod phantom

with six sections containing capillaries with diameters 1.0, 1.2, 1.4, 1.6, 1.8 and 2.0 mm was imaged (figure 5.4). The distance between the rods in each section was equal to the diameter of the rods in that section. There was also a 16 × 16 × 2 mm^3 reservoir at the base of the phantom directly below the rods. The rods were 15 mm in length. The phantom was designed using SolidWorks™, and printed with a 3D printer using FullCure 720™ material.



(A) (B) (C) (D)

FIGURE 5.4: Images of the phantom used in this study. The diameters of the rods are 1.0, 1.2, 1.4, 1.6, 1.8, 2.0 mm, and the separation between the centres of the rods in each section is equal to twice the diameter of those rods. There is also a small 16 × 16 × 2 mm^3 reservoir at the base of the rods.

The phantom was soaked in a bath of Kodak PhotoFlo™ solution to reduce surface tension. Even with this solution, it was difficult to ensure that the radioactive solution flowed into each of the rods of the phantom and had uniform distribution.

The centre of the phantom was placed approximately at the centre of the FOV, with the rods oriented perpendicular to the detector and the reservoir situated between the rods and the detector. The object-to-pinhole distance (d_{OP}) was kept constant during image acquisition. The pinhole-to-detector distances (d_{PD})

were 9, 17, 25, 33, 36, and 46 mm, corresponding to the system configurations used earlier for the calibration (table 4.1, section 4.1). This minimum distance is difficult to measure due to mechanical constraints, but using the SolidWorks^{TM} design it was estimated to be $\simeq$ 9 mm. Taking the centre of the FOV to be 23 mm from the pinholes, and the minimum d_{PD} to be 9 mm, the data were taken with magnifications of approximately 0.43, 0.78, 1.13, 1.48, 1.61 and 2.04 (figure 5.5). The acquisition parameters for the phantom study are shown in table 5.2. The acquisition time for the first image was 20 minutes and the acquisition time for each subsequent image was adjusted for isotope decay. The combined counts in the six images was $\sim 10^7$.

TABLE 5.2: The SyntheticSPECT detector settings for the phantom study.

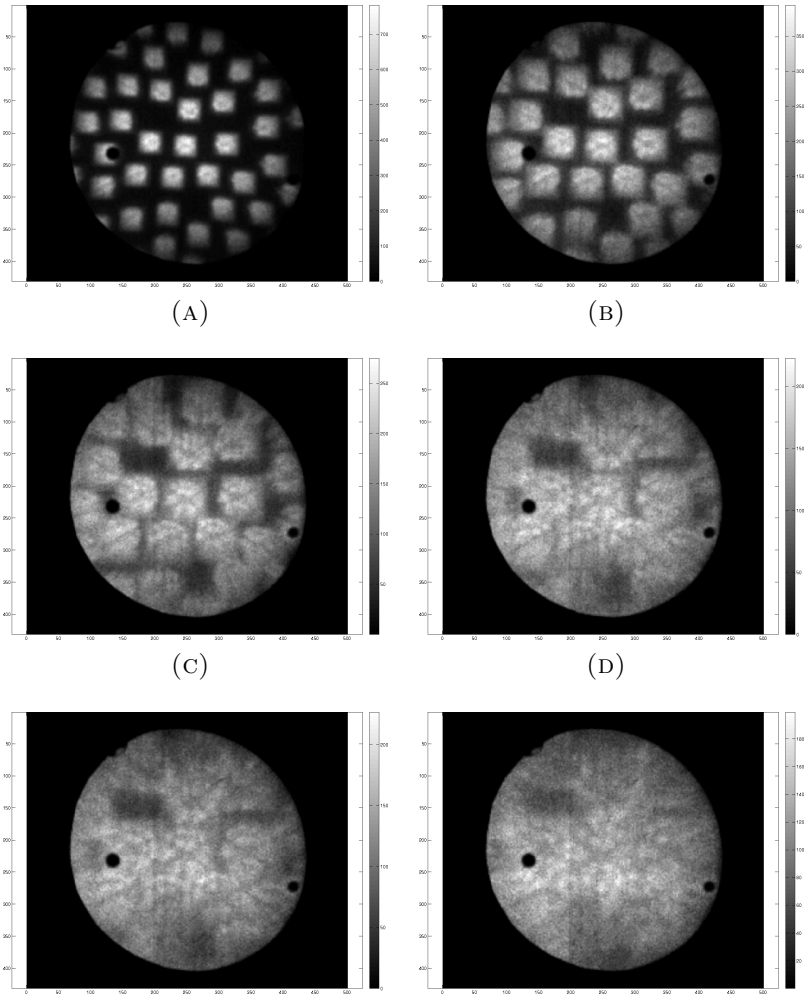
Parameter	Setting
Mode	Frame Parsing
Time	20 mins
Video Mode	320 × 240 × 8-bit
Frame Rate	200 FPS
CCD Gain	Max
CCD Shutter Time	2 ms
Signal Threshold	7
Cluster Radius	5
Isotope	^{99m}Tc

5.2.2 MLEM Reconstruction

The phantom was reconstructed on the 97 × 97 × 97 cubic grid using 400 iterations of the MLEM algorithm, and post-smoothed with a Gaussian filter. The number of iterations used in MLEM is slightly arbitrary and subjective. One approach is to calculate the difference in likelihood between one iteration and the next. The iterations are stopped if this difference falls below a predetermined threshold, set by the user. In this work, the algorithm was stopped after visual inspection of the reconstructed image.

The edge of the FOV has been removed due to excess build-up of estimated activity, as is common with MLEM. Some slices of the reconstructed object within the FOV are shown in figures 5.6 and 5.7.

Each image in figures 5.6 and 5.7 is a slice (thickness = 0.3125 mm) of the object at different distances from the collimator. The first slices (top left) were



(A) (B) (C) (D) (E) (F)

FIGURE 5.5: A hot-rod phantom was imaged at six geometric configurations corresponding to magnifications of: (a) 0.4, (b) 0.8, (c) 1.1, (d) 1.5, (e) 1.6 and (f) 2.0.

farthest from the collimator, and the last slices (lower right) were nearest to the collimator. The rods were not entirely filled, although some activity was visible in the smaller rods in the slices farthest from the detector. This was due to these rods being used as injection sites for the ^{99m}Tc, and some activity may have attached to the walls of the rods. The 1.4 mm rods, and some of the 1.2 mm rods, were resolved in several of the slices, although they were not clearly resolvable in the 3D synthetic-collimator image (figure 5.8).

The 1.6 mm rods were the smallest diameter rods that were resolved in the 3D synthetic-collimator image. It appears that one of the rods in the 1.6 mm set did not completely fill, and it was visible in only a small number of slices close to the collimator. This is the reason why it did not show a strong signal in the 3D

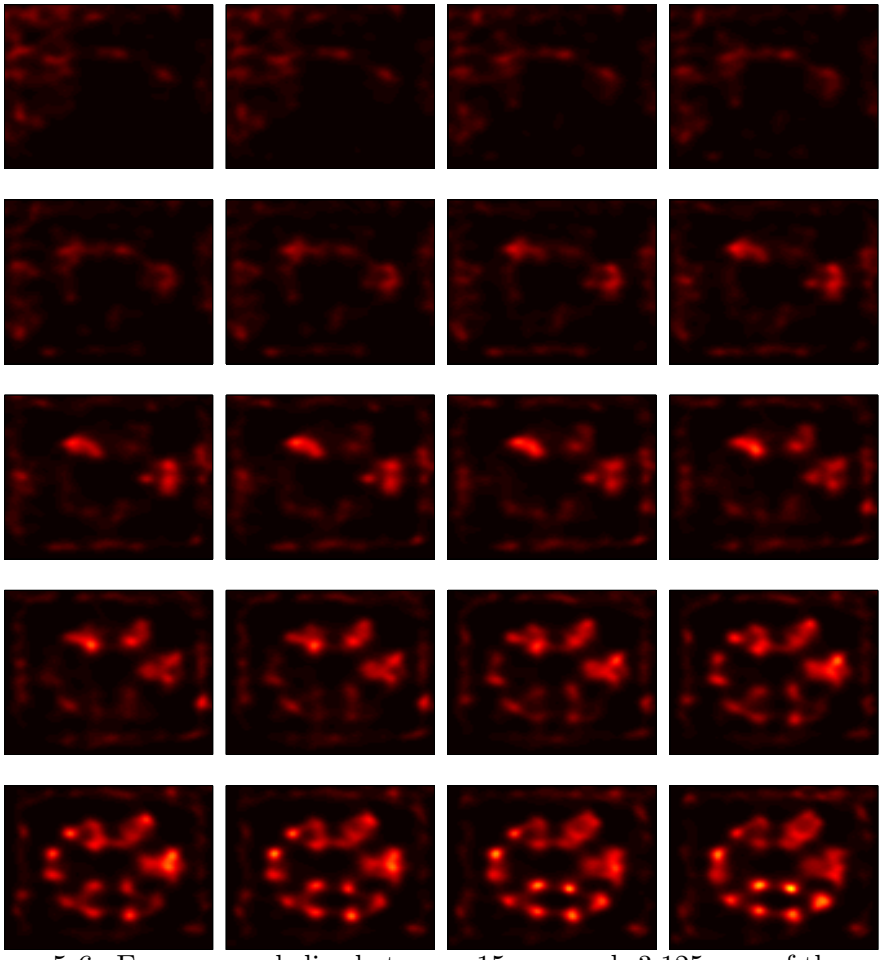


FIGURE 5.6: Every second slice between -15 mm and -3.125 mm of the reconstruction of the phantom with 400 iterations of the MLEM algorithm. The slice thickness is 0.3125 mm, and the intensities are globally normalized. The first slice is furthest from the collimator, and the last is nearest the collimator. The diameter of the rods range from 1.0 mm to 2.0 mm. The distance between the centre of each rod in each section is equal to twice the diameter of the rod. The rods are partially filled, and so they are not visible in each slice. The injection sites, where some activity may have adhered to the walls of the rods, can be seen in the first few slices. The rest of the slices of the reconstruction are shown in figure 5.7.

synthetic-collimator image. The 1.8 mm and 2.0 mm rods were clearly resolvable in most of the slices throughout the FOV. The reservoir of activity at the base of the phantom was clearly seen, although uniform activity was not estimated accurately. It is known that large uniform objects are difficult to reconstruct in limited-angle tomography.

The 3D synthetic-collimator image, after a range of iterations, is shown in figure 5.9. The larger rods were identifiable almost immediately, with the smaller rods becoming resolvable as the iterations increased.

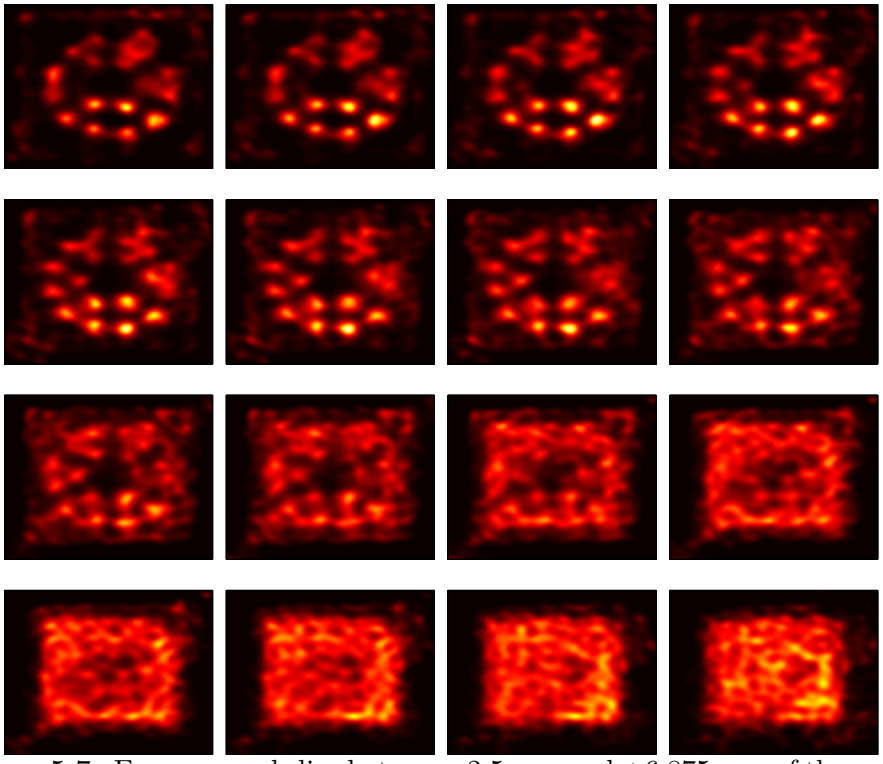


FIGURE 5.7: Every second slice between -2.5 mm and +6.875 mm of the reconstruction of the hot-rod phantom with 400 iterations of the MLEM algorithm. The slice thickness is 0.3125 mm, and the intensities are globally normalized. The first slice is furthest from the collimator, and the last is nearest the collimator. The diameter of the rods range from 1.0 mm to 2.0 mm. The distance between the centre of each rod in each section is equal to twice the diameter of the rod. The activity from the small reservoir becomes dominant in the last number of slices.

The experimental results agree with the results of the simulations in section 4.6. The simulated projection data were generated from discretized point-source data incorporating the 2D elliptical Gaussian model of the pinhole projections. The reconstruction using these data indicates the upper limit to the spatial resolution of the system using the model. The successful experimental reconstruction of a continuous hot-rod phantom validates the 2D elliptical Gaussian model and demonstrates the efficacy of synthetic collimation.

The experimental reconstruction demonstrates accurate estimation of the tube integrals of a parallel-projection image. The hot-rod phantom, with rods oriented perpendicular to the detector, was the most appropriate demonstration of synthetic collimation. A flank tumour, isolated from extraneous sources of emission, would be readily identified in the synthetic-collimator image.

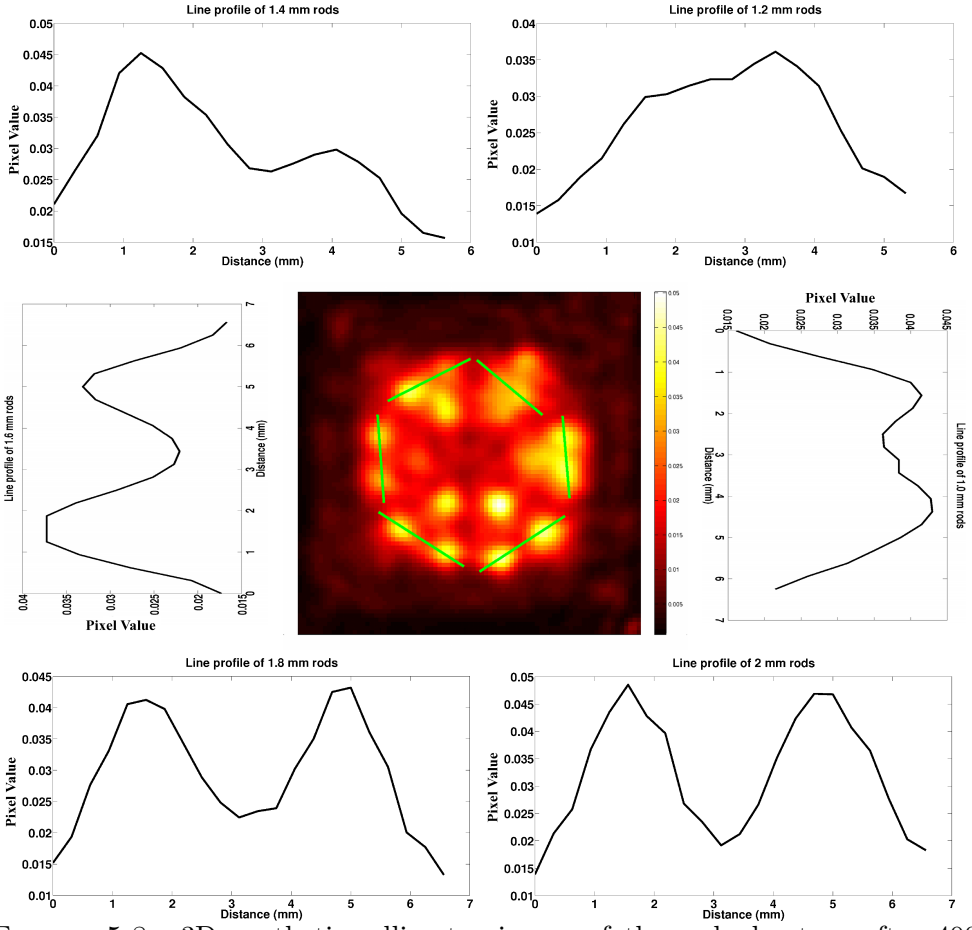
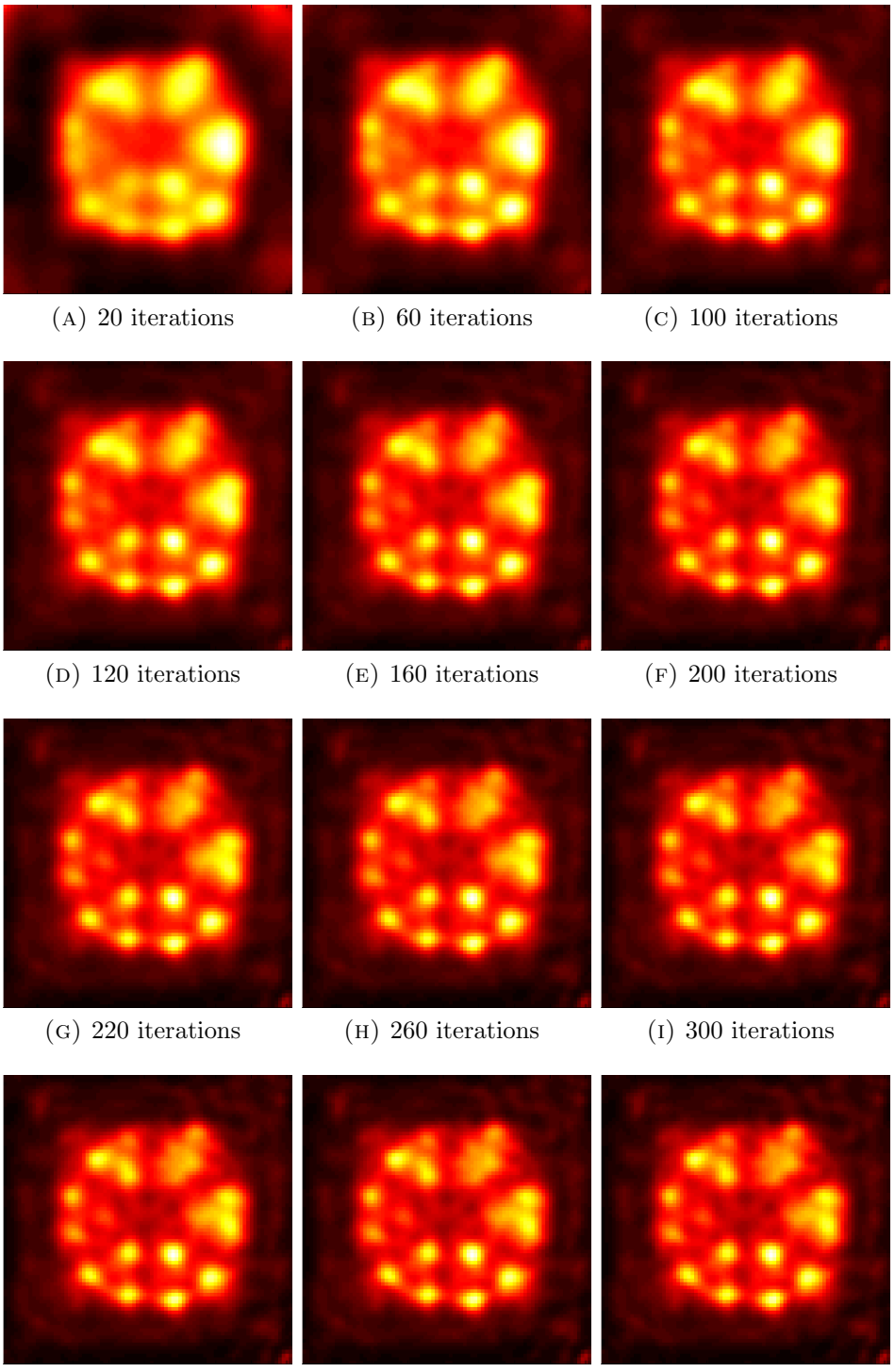


FIGURE 5.8: 3D synthetic-collimator image of the rod phantom after 400 iterations of MLEM. It was formed by summing the reconstructed voxels along the z axis, with z defined as the direction perpendicular to the collimator face. The line profiles for some of the rods are shown next to a green line highlighting the profile. The profiles were selected by joining the centroid of activity within each rod with the centroid of a neighbouring rod.

5.2.3 POSEM Reconstruction

The POSEM algorithm was also used to reconstruct the phantom due to the computational time of the MLEM algorithm. The MLEM results indicated that the smallest rods that could be resolved were ~ 1.4 mm diameter. The voxel size of 0.3125 mm was smaller than required. The reconstructions in this section were done with a voxel size of 0.5 mm. This corresponded to a voxel-space of $61 \times 61 \times 61$ voxels. This was one quarter the size of the voxel space reconstructed in the previous section, and so reconstructions were much less computationally expensive. The phantom was reconstructed using the first system configuration (figure 5.10), and using all the system configurations (figure 5.11). One iteration



(A) 20 iterations (B) 60 iterations (C) 100 iterations

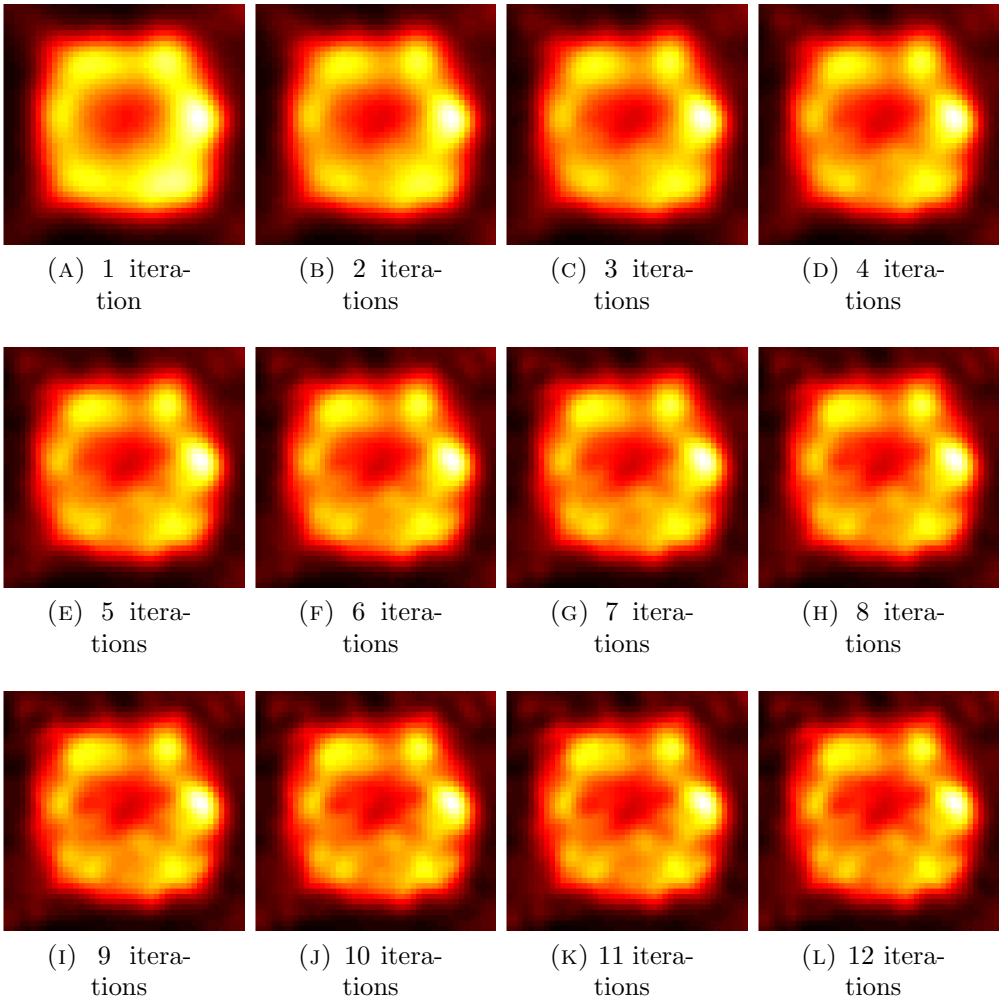
(D) 120 iterations (E) 160 iterations (F) 200 iterations

(G) 220 iterations (H) 260 iterations (I) 300 iterations

(J) 320 iterations (K) 360 iterations (L) 400 iterations

FIGURE 5.9: The phantom was reconstructed with the MLEM algorithm and smoothed with a Gaussian filter. The images displayed are the 3D synthetic-collimator images after different iterations (labelled below each image). These were produced by integrating the activities in the voxels along the axis perpendicular to collimator face. The diameter of the rods range from 1.0 mm to 2.0 mm. The distance between the centre of each rod in each section is equal to twice the diameter of the rod.

of the POSEM algorithm is approximately equivalent to sixteen iterations of the MLEM algorithm because of the acceleration factor used.



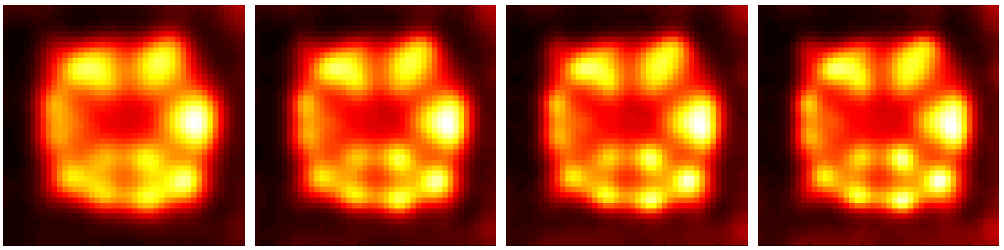
(A) 1 iteration (B) 2 iterations (C) 3 iterations (D) 4 iterations

(E) 5 iterations (F) 6 iterations (G) 7 iterations (H) 8 iterations

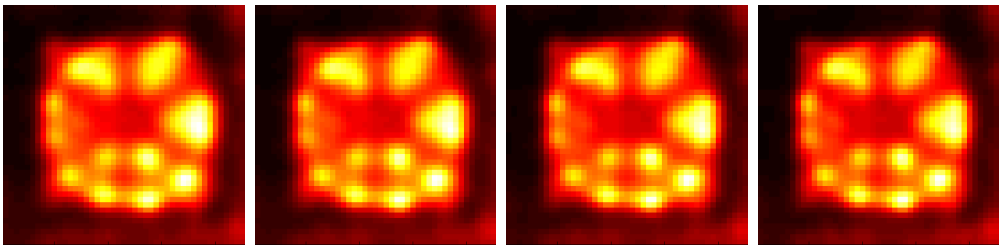
(I) 9 iterations (J) 10 iterations (K) 11 iterations (L) 12 iterations

FIGURE 5.10: The phantom was reconstructed with the POSEM algorithm. The images displayed are the 3D synthetic-collimator images after different iterations (labelled below each image). The data set with system magnification of 0.4 was used for this reconstruction. These were produced by summing the activities in the voxels along the axis perpendicular to collimator face.

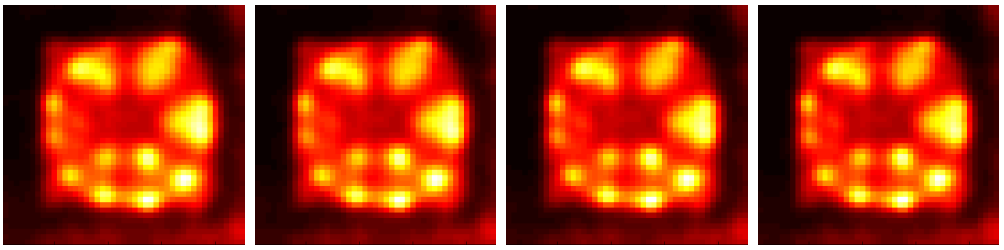
The POSEM algorithm successfully reconstructed the hot-rod phantom when data from all the system configurations were used. The effect of using higher-magnification system configurations is evident from figures 5.10 and 5.11. The reconstructions improved with each new system configuration included in the reconstruction. The POSEM algorithm produced similar 3D synthetic-collimator images to the MLEM algorithm, but in less time (table 5.3). These results validate the POSEM algorithm for use with synthetic collimation. The acceleration factor was not as large as expected. This poor performance was likely due to the extra operations required to access specific indices within the H matrix.



(A) 1 iteration (B) 2 iterations (C) 3 iterations (D) 4 iterations



(E) 5 iterations (F) 6 iterations (G) 7 iterations (H) 8 iterations



(I) 9 iterations (J) 10 iterations (K) 11 iterations (L) 12 iterations

FIGURE 5.11: The phantom was reconstructed with the POSEM algorithm. The images displayed are the 3D synthetic-collimator images after different iterations (labelled below each image). The data sets with system magnifications of 0.4, 0.8, 1.1, 1.5, 1.6, and 2.0 were used for this reconstruction. These were produced by summing the activities in the voxels along the axis perpendicular to collimator face.

TABLE 5.3: The reconstruction times for the MLEM and POSEM algorithms when all system configurations were used. The acceleration factor was not as high as predicted because of optimized MLEM code.

Algorithm	No. of voxels	Time (1 iteration)
MLEM	$61 \times 61 \times 61$	31 s
POSEM	$61 \times 61 \times 61$	30 s*

*POSEM is an accelerated algorithm and is normalized here to the equivalent of 1 MLEM iteration.

5.2.4 Small-Animal Imaging

It has been demonstrated that the SyntheticSPECT system can reconstruct an object within the FOV. Reconstruction of a small animal was a much more difficult challenge. The body of the small animal would likely extend beyond the

FOV for which the system matrix was measured. The body would have to be translated through the FOV to acquire a full-body dataset.

A typical method of acquiring these data is to image the body in small overlapping sections. These sections can then be reconstructed simultaneously, or individually and "stitched" together. In each of the sections there would be excess counts detected in some detector elements due to gamma rays originating from outside the common FOV of the collimator. The MLEM algorithm would attribute these extra counts to the edge of the reconstructed FOV, but they would originate from well outside the FOV. The pinholes were focused to a common FOV, which reduced the amount of gamma rays detected from outside the FOV. The focused pinholes at the edge of the collimator would detect photons originating outside the common FOV, along the x or y axis, depending on the orientation of the small animal. A solution to this would be to measure the system response for a much larger area and in areas that project through only a small number of the oblique pinholes. This would be extremely time consuming and computationally expensive and so this method was not implemented.

The approach taken in this work was to translate the small animal through the FOV in small increments. Data were taken at four different system configurations for each position. The system matrix was measured for a $30 \times 30 \times 30$ mm^3 FOV. The small-animal was translated through the FOV in 26 mm increments (figure 5.12). The entire body was reconstructed at the same time, with the overlapping sections, using all the data available.

4.58 mCi of ^{99m}Tc-MDP in 0.16 ml was injected into a 44g female imprinting control region (ICR) mouse. The mouse was sacrificed two hours after injection. Imaging of the mouse was done approximately six hours post-sacrifice because too many counts were detected in every frame, even at the highest frame rate. After six hours, the activity was sufficiently low for a low probability of pileup on the detector. A full-body imaging study was performed by placing the animal in the FOV and translating it through the FOV, for each system configuration. Five semi-overlapping datasets were acquired of the small animal (figures 5.13-5.17). The radiotracer would not be localized at a single point within the mouse and it was expected that out-of-field activity may affect the reconstruction.

The size of the mouse ($>$ 12 cm in length) posed a significant challenge. The activity in the area of the bladder was outside the shielding provided by the

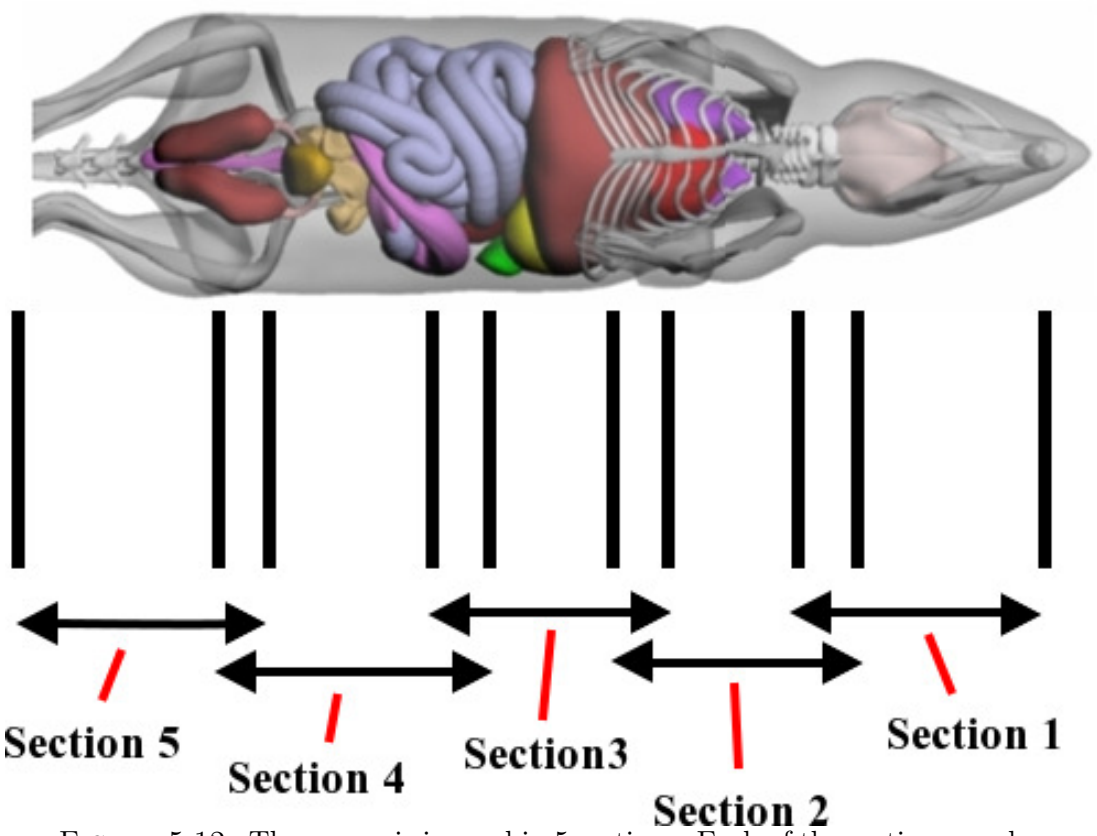
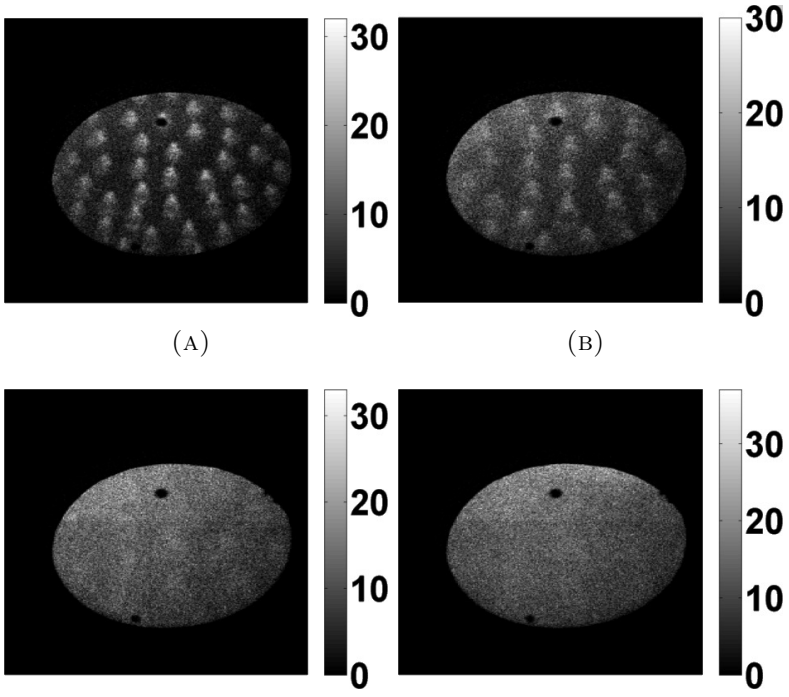


FIGURE 5.12: The mouse is imaged in 5 sections. Each of the sections overlap slightly with its neighbouring sections. Each of the sections corresponds to a $30 \times 30 \times 30$ mm^3 volume. (Reproduced from *Carl E Ravin Advanced Imaging Laboratories*.)

collimator when the high-magnification system configurations were used. When the detector was adjacent to the collimator this was not noticed because the activity in the periphery of the mouse was not detected. The activity was detected as the collimator was moved further from the detector for the high-magnification system configurations. The effect of this activity can be seen as a general increase in intensity in the corner of some of the projection images (figures 5.13-5.17). It was not possible to account for this in the reconstruction step, and high-resolution reconstructions were not possible. The low-magnification projection data were used to reconstruct the activity distribution within the animal.

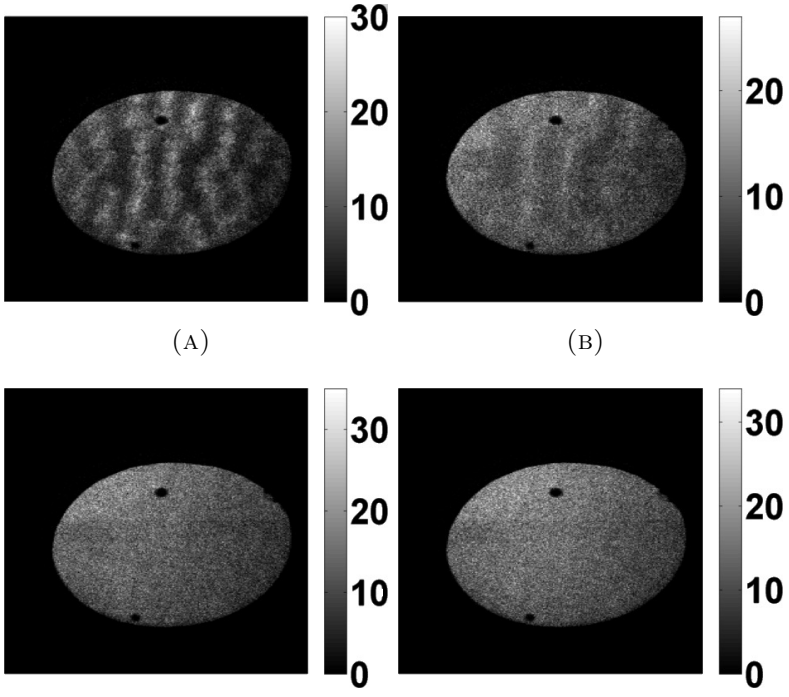
A low-resolution reconstruction is shown in figure 5.18, but this image does not take full advantage of the synthetic collimator. The overlapping regions had a large amount of excess counts in the reconstruction, most likely because of out-of-field activity. These regions have been partially masked and smoothed with a Gaussian filter because a solution was not found for these excess counts. The skull, kidneys and bladder were visible in the masked synthetic-collimator image.



(A) (B)

(C) (D)

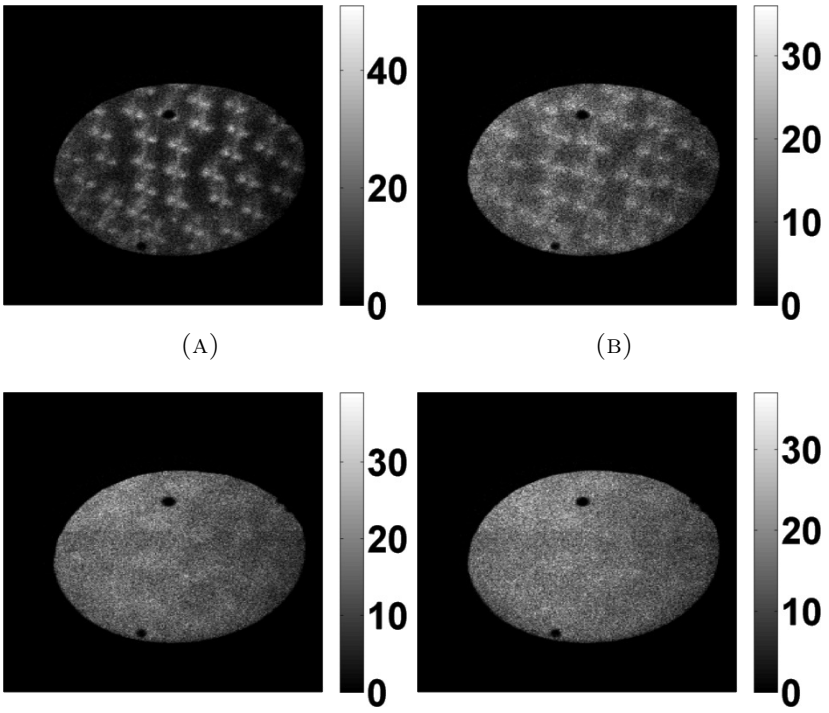
FIGURE 5.13: A mouse is injected with ^{99m}Tc and translated through the FOV in 26 mm steps. (a) - (d) Projections from the first FOV volume (skull) at 4 different system configurations. The low-resolution images were used in the reconstruction of figure 5.18.



(A) (B)

(C) (D)

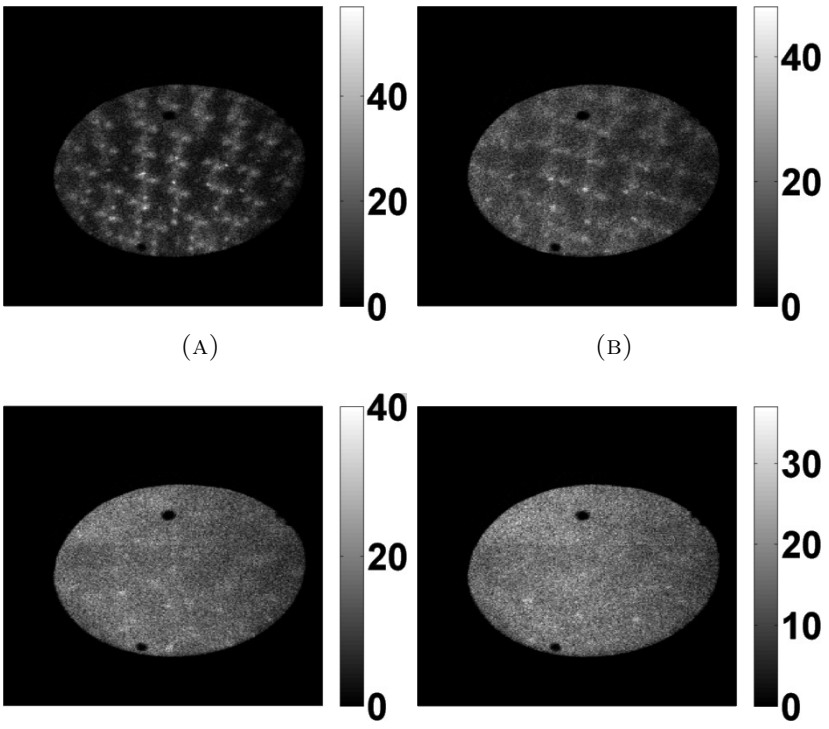
FIGURE 5.14: (a) - (d) Projections from the second FOV volume at 4 different system configurations.



(A) (B)

(C) (D)

FIGURE 5.15: (a) - (d) Projections from the third FOV volume at 4 different system configurations.



(A) (B)

(C) (D)

FIGURE 5.16: (a) - (d) Projections from the fourth FOV volume at 4 different system configurations.

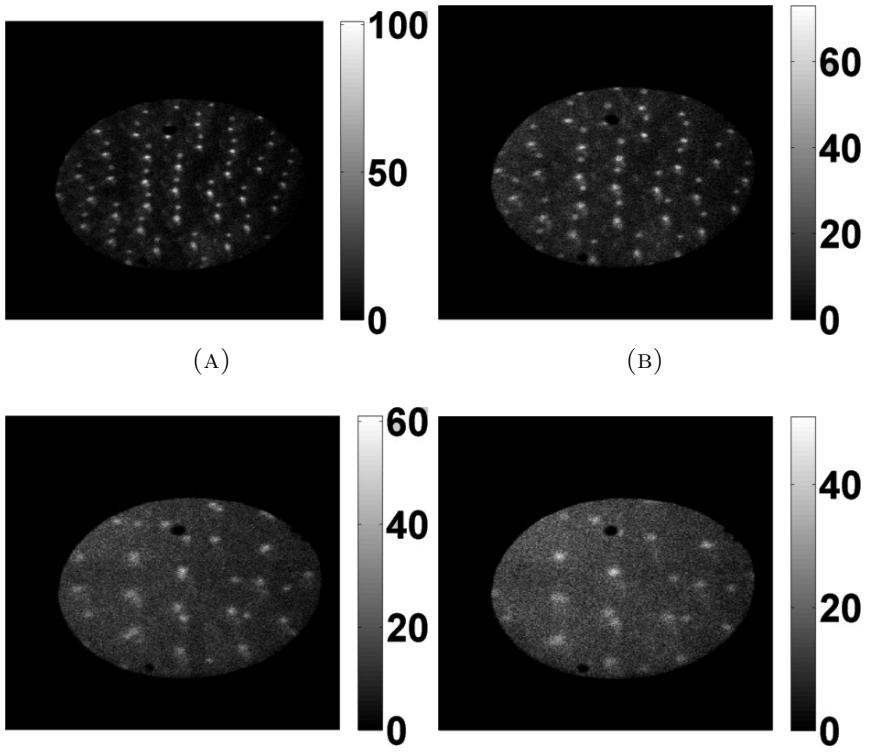


FIGURE 5.17: (a) - (d) Projections from the fifth FOV volume at 4 different system configurations.

The SyntheticSPECT system was not designed for a large mouse, as used in this study. When the head of the mouse was being imaged, the collimator did not shield the detector from the tail end of the mouse. Unfortunately it was not possible to perform another imaging study with a smaller, more suitable mouse. The system design could be adjusted to enable imaging of larger animals; a larger collimator could be fabricated or separate shielding could be attached to the detector. The reconstruction would also be improved by measuring the system response for a larger FOV. The object translation steps and pattern can also be optimized for whole-body imaging. A spiral bed-trajectory has been shown to improve the performance of a whole-body SPECT system (Vaissier et al., 2012).

FIGURE 5.18: A full-body image of 44g female ICR mouse that was injected with 4.58 mCi ^{99m}Tc-MDP in 0.16 ml. This is the first animal image from the SyntheticSPECT system. The mouse was over 12 cm in length. The animal was translated through the FOV in 26 mm increments (figures 5.13 - 5.17). The whole body was reconstructed simultaneously, but there were still excess counts in the overlapping regions. For this image, the boundaries where the different FOVs overlap have been partially masked and smoothed with a Gaussian filter. The image contrast is increased to enhance the signal in the centre of the animal. The activity is heavily concentrated in the head, kidneys and bladder of the animal.

5.3 Conclusions

The BazookaSPECT system successfully demonstrated its ability to track MSCs *in vivo*. A computational system matrix, developed by Dr. Wilson at CGRI, was used for the BazookaSPECT. This system matrix was not optimal as it did not account for many of physical characteristics of the system. An experimentally measured system matrix would improve the reconstructions.

The SyntheticSPECT system was validated experimentally with a hot-rod phantom. The reconstructions of hot-rod phantom showed that some depth information could be retrieved using a single tomographic angle. 1.4 mm rods were resolved in some of the slices of the reconstructed volume, but 1.6 mm rods were resolved in the 3D synthetic-collimator image. This difference is likely due to poor resolution in the z axis and blurring caused by under- and over-lying activity.

The POSEM algorithm was shown to have similar results to the MLEM algorithm. The computational time was not reduced as much as expected, likely due suboptimal implementation of the algorithm and the priority given to the MLEM implementation. The effect of using several system configurations was also demonstrated with POSEM. Higher-resolution reconstructions were acquired when all the system configurations were used.

The small-animal imaging study demonstrated that further work must be done to account for activity outside the measured FOV. Improved shielding must also be put in place before conducting further small-animal imaging studies.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

Major achievements:

- Designed the imaging section of the experiment to investigate the mesenchymal stem cell-mediated delivery of the sodium iodide symporter.
- Designed and constructed the first SPECT imaging system that implements synthetic collimation using a single tomographic angle.
- Designed a novel multi-pinhole collimator using 3D modelling software (SolidWorks) and fabricated it using 3D printing.
- Formulated an innovative calibration method for a multi-pinhole collimator. A radioactive point source was translated through the field of view in a sparsely-sampled grid and the system response was recorded at each position. A 2D elliptical Gaussian model was fitted to each pinhole-projection image. A finely-sampled system response was interpolated by accurate identification of pinhole-aperture projection patterns.
- Reconstructed the object in the field of view using maximum-likelihood estimation-maximization. Implemented this reconstruction algorithm using C and CUDA. Achieved a >1000× performance improvement compared to a MATLAB implementation.

- Validated the system through simulations and experimental reconstruction of a hot-rod phantom. Published results and suggested avenues for further research based on the synthetic-collimation concept. This concept could provide system designers with a cost-effective way to improve the performance of future commercial SPECT systems.

The objective of this research was to develop a high-resolution small-animal imaging system. Through a collaboration with the Center for Gamma-ray Imaging (CGRI), The University of Arizona, Tucson, AZ, USA, a prototype gamma-ray detector, called BazookaSPECT, was developed at NUI Galway. This system was used in collaboration with the Regenerative Medicine Institute, NUI Galway to investigate the mesenchymal stem cell-mediated delivery of the sodium iodide symporter.

The BazookaSPECT detector, collimator, and acquisition and reconstruction software were provided by the collaborators at CGRI. The author's contribution to the work was the design of the imaging work, given the constraints of the system, and the data processing. Full-body images of the small animals were required because of a lack of anatomical information in the data. With full-body imaging it is often easier to distinguish anatomical objects within the animal due to the strong uptake of radiotracer in the thyroid and the stomach. A system geometry was determined that resulted in whole-body imaging of the small animal, but at the expense of sensitivity and resolution. After much experimentation, some small-animal images were acquired that validated a novel therapy for breast cancer.

An imaging system should be customized for a particular application. The future application objectives at REMEDI could not be achieved by a single small detector. To achieve the high resolution and sensitivity required, many detectors, or a much larger detector would be needed. The image intensifier used in BazookaSPECT had a small active area (<50 mm). Larger image intensifiers are available, although they are often prohibitively expensive, and so an alternative solution was sought. The collaborators at CGRI were investigating the use of a fibre-optic taper to increase the active area of a BazookaSPECT detector. During a visit to CGRI, I developed a large-area BazookaSPECT detector using a fibre-optic taper with an opening diameter of 100 mm. Due to light loss, a double-stack MCP, instead of the XX1332 image intensifier, was required to amplify the scintillation light.

It is possible to use this large-area detector in several configurations depending on the application. It could be used to highly magnify a small field of view, or as a planar imager for mice and rats. Construction of a high-resolution planar imager was one of the goals of my work, and I successfully developed an imaging system, SyntheticSPECT, to achieve this objective.

Researchers at CGRI had shown through simulation that a synthetic collimator could be used to synthesize parallel-projection images. It was believed that the large-area BazookaSPECT detector would be ideal to demonstrate the potential of the synthetic collimator because of its large active area and high-resolution detector. This collimation technique allows for a greater range of applications than a parallel-hole collimator, typically used to produce parallel-projection images. The system hardware design, collimator fabrication and software were developed by the author.

The first challenge in this project was to fabricate a collimator that was suitable for the task. A novel pinhole-fabrication method using 3D printing was developed. The method developed by the author was much lower cost and more readily applicable than other 3D printing methods. The 3D printing material was left within the collimator after casting with tungsten epoxy. Removing the material damages the pinhole-aperture geometry and recasting of the pinholes is often required, increasing the fabrication cost. This fabrication method allowed the development of a large, focused multi-pinhole collimator that would have been difficult to fabricate with traditional machining methods. This collimator was designed for a 160 mm fibre-optic taper, but unfortunately the taper was not available for use. The collimator had one hundred and ten focused pinholes, but only forty-six pinholes projected the FOV onto the 100 mm taper. Using this collimation technique, it was demonstrated that pinhole-projection overlap could be used to maximize the active detector area while not sacrificing image resolution. It was shown that acquiring data with overlapping projections increased the system resolution, provided the data were used in conjunction with non-overlapping projection data.

The system matrix, **H**, was measured on a sparsely-sampled grid encompassing the FOV common to all the focused pinholes. Motion control for a 3-axis translation stage was integrated into the image acquisition software to measure this system matrix. An algorithm was developed to identify and track all the individual pinhole projections on the detector as the point source was translated

through the FOV. The pinhole response was modelled as a 2D elliptical Gaussian distribution. The high-resolution system matrix was interpolated from these Gaussian coefficients. The interpolated system matrix was validated through comparison with a small, finely-sampled system matrix and with a hot-rod phantom experiment.

Fast GPU-based MLEM and POSEM reconstruction algorithms were developed. These algorithms generated the system matrix on-the-fly from coefficient data extracted from measured PSF data. Implementation on a GPU enabled the projection of the entire H matrix in less than a second rather than in minutes, as is typical with CPU based approaches. These reconstruction methods were validated through simulation and experimentally with a hot-rod phantom.

A small-animal imaging study was also performed with SyntheticSPECT. The small animal was translated through the FOV in increments smaller than the measured FOV. These sections were then reconstructed simultaneously using MLEM. The reconstruction showed a lot of activity in the overlapping sections; an indication of an unresolved error in the reconstruction step. This was likely due to out-of-field activity but a solution was not found. The skull and other parts of the animal were reconstructed, but high-resolution images were not acquired because of out-of-field activity.

Synthetic collimation advantages:

- An interpretable parallel-projection image can be synthesized from pinhole-projection data.
- Synthetic collimation makes more efficient use of the detector than typical collimation methods.
- The system is low cost compared to a typical multi-detector SPECT system. A single detector and collimator are required, but rotation of the animal or the system is not required.
- With a stack of thin detectors, multiplexed and non-multiplexed data could be simultaneously acquired. This could lead to more efficient SPECT system designs.
- Image reconstruction is very fast and a diagnostic image can be reconstructed in seconds.

- The signal-to-noise ratio of the projection data is higher because there is no movement or rotation of the object. Longer acquisition times are possible.

Synthetic collimation disadvantages:

- Poor depth resolution is achieved. Activity in each voxel is not independently quantifiable.
- The system requires a large-area detector.
- The collimator was custom designed using 3D printing. This technology may not be available to many groups.

6.2 Future Work

There are many ways that the BazookaSPECT and SyntheticSPECT systems may be improved. The implementation of the algorithms used for image formation and image reconstruction can be improved upon.

The single-channel BazookaSPECT system at NUIG will be expanded to an eight-channel system. The increase in the number of detectors will improve the sensitivity and spatial resolution of the system, depending on the configuration. For further MSC investigations, the small animal will be translated through a small FOV to provide a high-resolution full-body image. Sufficient shielding will be in place to reduce the potential for out-of-field activity. Researchers at REMEDI are also interested in imaging the knee joint; a task that is suited to this system configuration.

The design of the multi-pinhole collimator for SyntheticSPECT was not optimized in this work. The minimum inter-pinhole distance was kept constant, but perhaps it could be varied to maximize the active detector area in all projections. The inter-pinhole distances could be changed as a function of distance from the central pinhole aperture to minimize multiplexing for the peripheral pinhole-aperture projections and maximize the active area of the detector.

There have been rapid advancements in the field of 3D printing during the course of this work. It is currently possible to directly print (rather than cast) parts using tungsten and other materials. This direct-printing technique uses

much denser materials, allowing much thinner collimators to be designed. The resolution of the printing will continue to improve, enabling smaller-diameter pinholes to be fabricated.

The work presented here demonstrated the efficacy of synthetic collimation. The next improvement to SyntheticSPECT could be to demonstrate its high-resolution capability by fabricating a multi-pinhole collimator with smaller pinholes. The optimal system configurations for synthetic collimation were not determined in this work. Six system configurations were used, and although it was shown that each configuration improves the estimate of the activity distribution, the optimal combination of data is unclear.

A thin Kodak Lanex scintillator is currently used in SyntheticSPECT. A thicker scintillator such as columnar CsI or $LaBr_3$ could improve the detection efficiency and energy resolution of the system. Currently, every event detected is used to reconstruct the image, but improved energy resolution could lead to better event discrimination. The depth-of-interaction effects would need to be modelled if a thick scintillator is used. Implementation of a maximum-likelihood model for the gamma-ray interaction in the scintillator would improve the spatial resolution and energy resolution of the system. Although GPUs are extremely fast, real time maximum-likelihood estimation for every event is not yet feasible at high count rates because of their iterative nature.

For a given pinhole diameter and object-collimator distance, the number of events detected may be increased by improving detector efficiency with a thicker scintillator, by increasing the activity of the source, or by increasing the acquisition time. Increasing the frame rate of the CCD will reduce the probability of cluster overlap in each frame. The uncertainty in the 2D Gaussian fitting may be reduced by increasing the number of events detected for each voxel position. Improved Gaussian fitting will result in a smoother, more accurate interpolation of the system matrix. The results presented here show the importance of high counts-per-pixel for each system configuration.

It has been demonstrated that a 4:1 fibre-optic taper can be effectively used to increase the area of the BazookaSPECT detector. The maximum taper minification that can be used should be determined. The limiting factor will likely be light loss due to the minification process, but image intensifiers with increased gain, which would mitigate this light loss, are also under development.

A single detector was used in this work. The magnification of the system was adjusted by changing the collimator-detector distance. Synthetic collimation would be optimally implemented using a stack of thin detectors along a single axis. In a stacked configuration, a series of detectors could simultaneously acquire non-multiplexed and multiplexed data. There could be increased scatter due to multiple detectors, but the potentially deleterious effects could be mitigated with a comprehensive measurement of the system matrix.

APPENDIX A

CUDA Code

A.1 Program Headers

```
#define _USE_MATH_DEFINES
#include <stdio.h>
#include <cuda.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <ctime>
#include <cstring>
#include <sstream>
#include <stdexcept>
#include <string>
#include <cufft.h>

#include <curand_kernel.h>
using namespace std;

#define DIMX 97 /*or 61*/
#define DIMY 97 /*or 61*/
#define DIMZ 97 /*or 61*/
#define IMAGE_DIMX 320
#define IMAGE_DIMY 240
#define APPLYMASK 1
#define LOADESTIMATE 0
#define NUMBEROFIMAGES 6
#define ADD_POISSON 0
#define DECAY_CORRECT_PSF 0
#define ALGORITHM 0 /* MLEM = 0, POSEM = 1, Image OSEM = 2,
    LANDWEBER = 3*/
#define FILTER_PSF 1
#define NOISE_PROJECTION 0
```

```
31 #define CUDACALL(x) {cudaError_t cuda_error__ = (x); if (
    cuda_error__) printf(#x " returned \"%s\"\n", cudaGetErrorString(
    cuda_error__));}
```

A.2 Utility Functions

```
1  /*Convert an int to a string.*/
2  std::string IntToStr( int n )
3  {
4    std::stringstream result;
5    result << n;
6    return result.str();
7  }
8
9  /*Convert a string to an int.*/
10 int StrToInt( const std::string& s )
11 {
12   int result;
13   std::stringstream ss( s );
14   ss >> result;
15   if (!ss) throw std::invalid_argument( "StrToInt" );
16   return result;
17 }
18
19 /*Check CUDA device for errors.*/
20 void syncDeviceAndErrorCheck()
21 {
22   cudaDeviceSynchronize();
23   cudaError_t err = cudaGetLastError();
24   if (err!= cudaSuccess)
25   {
26     printf("Last Error: %s\n",cudaGetErrorString(err));
27     int stop;
28     scanf("%d",&stop);
29   }
30 }
31
32 /*Set float values in a CUDA array to zero.*/
33 __global__ void fl_setToZeroCUDA(float *image,int N)
34 {
35   int idx = blockIdx.x*blockDim.x+threadIdx.x;
36
37   if (idx < N)
38   {
39     image[idx] = 0.0f;
40   }
41 }
42
43 /*Invert a CUDA array.*/
44 __global__ void fl_invertValueCUDA(float *image,int N)
45 {
46   int idx = blockIdx.x*blockDim.x+threadIdx.x;
47
```

```
48    if (idx < N)
49    {
50      if (image[idx]>0.0f)
51        image[idx] = 1.0f/image[idx];
52      else
53        image[idx] = 0.0f;
54    }
55  }
56
57  /*Apply a mask to a CUDA array.*/
58  __global__ void fl_applyMaskCUDA(float *image,int *mask,int N)
59  {
60    int idx = blockIdx.x*blockDim.x+threadIdx.x;
61
62    if (idx < N)
63    {
64      if (mask[idx]==0)
65        image[idx] = 0.0f;
66    }
67  }
68
69  /*Multiply two matrices.*/
70  __global__ void fl_multiplyMatricesCUDA(float *matrix1,float *
      matrix2,int N)
71  {
72    int idx = blockIdx.x*blockDim.x+threadIdx.x;
73
74    if (idx < N)
75    {
76      matrix1[idx] = matrix1[idx]*matrix2[idx];
77    }
78  }
79
80  /*Multiply two matrices with a weight.*/
81  __global__ void fl_addMatricesWithWeightsCUDA(float *matrix1,float *
      matrix2,float *currentValue,int voxelIndex,int N)
82  {
83    int idx = blockIdx.x*blockDim.x+threadIdx.x;
84    extern __shared__ float shared_currentValue[];
85    if (threadIdx.x ==0)
86    {
87      shared_currentValue[0] = currentValue[voxelIndex];
88    }
89    __syncthreads();
90    if (idx < N)
91    {
92      matrix1[idx] += matrix2[idx]*shared_currentValue[0];
93    }
94  }
95
96  /*Divide two matrices.*/
97  __global__ void fl_divideMatricesCUDA(float *matrixResult,float *
      matrix1,float *matrix2,int N)
98  {
99    int idx = blockIdx.x*blockDim.x+threadIdx.x;
100
101   if (idx < N)
```

```
102   {
103     if (matrix2[idx]>0.0f)
104       matrixResult[idx] = matrix1[idx]/matrix2[idx];
105     else
106       matrixResult[idx] = 0.0f;
107
108   }
109 }
110
111 /*Set a CUDA array to a value.*/
112 __global__ void fl_setToValueCUDA(float *images_d,float value,int *
    mask,int N)
113 {
114   int idx = blockIdx.x*blockDim.x+threadIdx.x;
115
116   if (idx < N)
117   {
118     if (mask[idx] == 1)
119       images_d[idx] = value;
120   }
121 }
122
123 /*Sum arrays. Very inefficient method.*/
124 __global__ void mlem_sumImages(float *detectorSum,float *images_d,
    int imageSize,int N)
125 {
126   int idx = blockIdx.x*blockDim.x+threadIdx.x;
127
128   if (idx < N)
129   {
130     float val=0.0f;
131     for (int i=0;i<imageSize;i++)
132     {
133       val += images_d[idx*imageSize + i];
134     }
135     detectorSum[idx] += val;
136   }
137 }
138
139 /*Efficient addition of arrays, used with '
    getSumArrayGPUtoGPUWithMultiplication'.*/
140 /*This is efficient reduction code adapted from code published by
    NVIDIA.*/
141 template <unsigned int blockSize>
142 __global__ void reduce6WithMultiplicationMultipleImages(float *
    g_idata,float *input2, float *g_odata, unsigned int n) {
143 extern __shared__ float sdata[];
144 unsigned int tid = threadIdx.x;
145 unsigned int i = blockIdx.x*(blockSize*2) + tid;
146 unsigned int gridSize = blockSize*2*gridDim.x;
147 sdata[tid] = 0.0f;
148 while (i < n)
149 {
150   sdata[tid] += g_idata[i]*input2[i];
151   if(i+blockSize < n)
152     sdata[tid] += g_idata[i+blockSize]*input2[i+blockSize];
153   i += gridSize;
```

```
154 }
155 __syncthreads();
156 if (blockSize >= 512) { if (tid < 256) { sdata[tid] += sdata[tid +
    256]; } __syncthreads(); }
157 if (blockSize >= 256) { if (tid < 128) { sdata[tid] += sdata[tid +
    128]; } __syncthreads(); }
158 if (blockSize >= 128) { if (tid <  64) { sdata[tid] += sdata[tid +
     64]; } __syncthreads(); }
159 if (tid < 32)
160 {
161   volatile float *smem = sdata;
162   if (blockSize >=  64) smem[tid] += smem[tid + 32];
163 if (blockSize >=  32) smem[tid] += smem[tid + 16];
164 if (blockSize >=  16) smem[tid] += smem[tid +  8];
165 if (blockSize >=   8) smem[tid] += smem[tid +  4];
166 if (blockSize >=   4) smem[tid] += smem[tid +  2];
167 if (blockSize >=   2) smem[tid] += smem[tid +  1];
168 }
169 if (tid == 0) g_odata[blockIdx.x] += sdata[0];
170 }
171
172 void getSumArrayGPUtoGPUWithMultiplication(float *input_d,float *
    input2_d, float *output_d,int index,int N)
173 {
174   const unsigned int threadsPerBlock = 512;
175   dim3 dimBlock(threadsPerBlock,1);
176   int numBlocks = (N + threadsPerBlock*2 -1) / (threadsPerBlock*2);
177
178   while (N > 1)
179   {
180     numBlocks = (N + threadsPerBlock*2 -1) / (threadsPerBlock*2);
181     dim3 dimGrid(numBlocks,1);
182
183     if (numBlocks==1)
184     {
185       reduce6MultipleImages<512><<<dimGrid,dimBlock,sizeof(float)*
    threadsPerBlock>>>(input_d, &output_d[index], N);
186       N = numBlocks;
187     }
188     else{
189   reduce6WithMultiplicationMultipleImages<512><<<dimGrid,dimBlock,
    sizeof(float)*threadsPerBlock>>>(input_d,input2_d, input_d, N);
190     N = numBlocks;
191
192     }
193   }
194 }
195
196 /*Set an int array to zero.*/
197 void i_resetToZero(int *H,int num)
198 {
199   for (int i=0;i<num;i++)
200   {
201     H[i] = 0;
202   }
203 }
204
```

```
205 /*Reset a float array to zero.*/
206 void resetToZero(float *H,int num)
207 {
208   for (int i=0;i<num;i++)
209   {
210     H[i] = 0.0f;
211   }
212 }
213
214 /*Set a float array to a value.*/
215 void resetToValue(float *H,int num,float value)
216 {
217   for (int i=0;i<num;i++)
218   {
219     H[i] = value;
220   }
221 }
222
223 /*Set an int array to zero.*/
224 void i_resetToValue(int *H,int num,int value)
225 {
226   for (int i=0;i<num;i++)
227   {
228     H[i] = value;
229   }
230 }
231
232 /*Multiplication for POSEM algorithm.*/
233 __global__ void osem_weightedMultiplication(float *proj_d,float *
      data_d, float *currentValue,int *pixelIndices,int N)
234 {
235   int idx = blockIdx.x*blockDim.x+threadIdx.x;
236   extern __shared__ float shared_currentValue[];
237   if (threadIdx.x ==0)
238   {
239     shared_currentValue[0] = currentValue[0];
240   }
241   __syncthreads();
242   if (idx < N)
243   {
244     proj_d[pixelIndices[idx]] += data_d[idx]*shared_currentValue[0];
245   }
246 }
247
248 /*Sum a block for POSEM.*/
249 __global__ void osem_sumBlocks(float *sensitivity,float *data,int
      blockSize,int N)
250 {
251   int idx = blockIdx.x*blockDim.x+threadIdx.x;
252
253   if (idx < N)
254   {
255     float val=0.0f;
256     for (int i=0;i<blockSize;i++)
257     {
258       val += data[idx*blockSize + i];
259     }
```

```
260       sensitivity[idx] += val;
261     }
262 }
263
264 /*Update the current estimate of the activity distribution on the
      GPU. Used for POSEM and MLEM.*/
265 __global__ void mlem_updateEstimateCUDA(float *
      currentPositionEstimate,float *sensitivityProj,float *detectorSum
      ,int voxelSpaceSize)
266 {
267   int idx = blockIdx.x*blockDim.x+threadIdx.x;
268
269   if (idx < voxelSpaceSize)
270   {
271     currentPositionEstimate[idx] = currentPositionEstimate[idx] *
      sensitivityProj[idx] * detectorSum[idx];
272   }
273 }
```

A.3 Masks

```
1 /*Apply a mask to an int array.*/
2 void i_applyCircularMask(int *mask)
3 {
4   /*Center found using centroiding.*/
5   int xCenter = 174;
6   int yCenter = 134;
7   cout << "Warning: Mask Applied. " << endl;
8   for (int i=0;i<IMAGE_DIMX*IMAGE_DIMY;i++)
9   {
10    int y = (int)floor((float)i/(float)IMAGE_DIMX) ;
11    int x = i%IMAGE_DIMX;
12    if (sqrt((float)((float)pow((float)(x - xCenter),2) + (float)pow
      ((float)(y-yCenter),2))) > 79.0f)
13    {
14      mask[i] = 0;
15    }
16
17    /*Deadspots.*/
18    if (sqrt((float)((float)pow((float)(x - 116),2) + (float)pow((
      float)(y-123),2))) < 8.0f)
19    {
20      mask[i] = 0;
21    }
22    if (sqrt((float)((float)pow((float)(x - 104),2) + (float)pow((
      float)(y-258),2))) < 8.0f)
23    {
24      mask[i] = 0;
25    }
26  }
27 }
28
29 /*Apply a mask to a float array.*/
```

```
30 void fl_applyCircularMask(float *mask)
31 {
32   int xCenter = 174;
33   int yCenter = 134;
34   cout << "Warning: Mask Applied. " << endl;
35   for (int i=0;i<IMAGE_DIMX*IMAGE_DIMY;i++)
36   {
37     int y = (int)floor((float)i/(float)IMAGE_DIMX) ;
38     int x = i%IMAGE_DIMX;
39
40     if (sqrt((float)((float)pow((float)(x - xCenter),2) + (float)pow
     ((float)(y-yCenter),2))) > 79.0f)
41     {
42       mask[i] = 0.0f;
43     }
44
45     /*Deadspots.*/
46     if (sqrt((float)((float)pow((float)(x - 116),2) + (float)pow((
     float)(y-123),2))) < 8.0f)
47     {
48       mask[i] = 0.0f;
49     }
50     if (sqrt((float)((float)pow((float)(x - 104),2) + (float)pow((
     float)(y-258),2))) < 8.0f)
51     {
52       mask[i] = 0.0f;
53     }
54   }
55 }
```

A.4 Index the Coefficients

```
1 /*Sort and index the coefficient files. Very inefficient.*/
2 int getSize(FILE *fid)
3 {
4   int numFloats = 0;
5   int numCoeff = 0;
6
7   for (int i=0;i<DIMX;i++)
8   {
9     for (int j=0;j<DIMY;j++)
10    {
11      for (int k=0;k<DIMZ;k++)
12      {
13        fread(&numCoeff,sizeof(int),1,fid);
14        numFloats = numFloats + numCoeff*6;
15        fseek(fid,numCoeff*6*sizeof(float),SEEK_CUR);
16      }
17    }
18  }
19  return numFloats;
20 }
21
```

```
22 void loadCoefficients(FILE *fid, int *numberOfCoefficients,float *
     gaussianCoefficients)
23 {
24   int numCoeff = 0;
25
26   int gaussianFilePosition = 0;
27   int numberFilePosition = 0;
28   for (int i=0;i<DIMX;i++)
29   {
30     for (int j=0;j<DIMY;j++)
31     {
32       for (int k=0;k<DIMZ;k++)
33       {
34         fread(&numCoeff,sizeof(int),1,fid);
35         numberOfCoefficients[numberFilePosition] = numCoeff;
36
37         fread(&gaussianCoefficients[gaussianFilePosition],1,sizeof(
     float)*6*numCoeff,fid);
38         numberFilePosition++;
39         gaussianFilePosition = gaussianFilePosition + 6*numCoeff;
40       }
41     }
42   }
43 }
44
45 int getFilePosition(int *numberOfCoefficients,int index)
46 {
47   int filePosition = 0;
48   for (int i=0;i<index;i++)
49   {
50     filePosition = filePosition + numberOfCoefficients[i];
51   }
52   filePosition = filePosition * 6;
53   return filePosition;
54 }
55
56 void getGaussPosition(int *gaussPosition,int *numberOfCoefficients,
     int offset)
57 {
58   int index = 0;
59   for (int i=0;i<DIMX;i++)
60   {
61     for (int j=0;j<DIMY;j++)
62     {
63       for (int k=0;k<DIMZ;k++)
64       {
65         gaussPosition[index] = getFilePosition(numberOfCoefficients,
     index) + offset;
66         index++;
67       }
68     }
69   }
70 }
```

A.5 Gaussian Kernels

```
1  /*Used for the MLEM algorithm.*/
2
3  /*CUDA kernel to calculate Gaussian response.*/
4  __device__ float gaussKernelOther(float *sharedGaussianCoefficient,
       float xCent,float yCent)
5  {
6    float a = ((xCent - (sharedGaussianCoefficient[1]))*(xCent - (
       sharedGaussianCoefficient[1]))) / ((sharedGaussianCoefficient[3])
       *(sharedGaussianCoefficient[3]));
7    float b = -(2.0f*sharedGaussianCoefficient[5])*(xCent - (
       sharedGaussianCoefficient[1]))*(yCent - (
       sharedGaussianCoefficient[2])) / (sharedGaussianCoefficient[3]*
       sharedGaussianCoefficient[4]);
8    float c = ((yCent - (sharedGaussianCoefficient[2]))*(yCent - (
       sharedGaussianCoefficient[2]))) / (sharedGaussianCoefficient[4]*
       sharedGaussianCoefficient[4]);
9    float z = a + b + c;
10   float tempVal = sharedGaussianCoefficient[0]*(__expf(-z / (2.0f
       *(1.0f-(sharedGaussianCoefficient[5]*sharedGaussianCoefficient
       [5])))));
11
12   return tempVal;
13 }
14
15 /*Used for the POSEM algorithm.*
16 /*CUDA kernel to calculate Gaussian response. */
17 __device__ float gaussKernelOther2(float *sharedGaussianCoefficient,
       float xSq,float ySq,float xCent,float yCent)
18 {
19   float a = (xSq) / ((sharedGaussianCoefficient[3])*(
       sharedGaussianCoefficient[3]));
20   float b = -(2.0f*sharedGaussianCoefficient[5])*(xCent - (
       sharedGaussianCoefficient[1]))*(yCent - (
       sharedGaussianCoefficient[2])) / (sharedGaussianCoefficient[3]*
       sharedGaussianCoefficient[4]);
21   float c = (ySq) / (sharedGaussianCoefficient[4]*
       sharedGaussianCoefficient[4]);
22   float z = a + b + c;
23   float tempVal = sharedGaussianCoefficient[0]*(__expf(-z / (2.0f
       *(1.0f-(sharedGaussianCoefficient[5]*sharedGaussianCoefficient
       [5])))));
24
25   return tempVal;
26 }
27
28 /*Project the Gaussian distribution.*/
29 /*Used for MLEM Implementation 1*/
30 __global__ void createPSFFast(int *numberOfCoefficients_d,float *
       gaussianCoefficients_d,int *gaussPosition_d,float *images_d,int
       imageSize,int startVoxel,int imageNumber,int voxelSpaceSize,int
       gaussPositionOffset,int N)
31 {
32   int idx = blockIdx.x*blockDim.x+threadIdx.x;
33
```

```
34   if (idx < N)
35   {
36     float x[6];
37     int num_sigma = 3;
38     int currentIndex = startVoxel + idx;
39     int gaussIndex = gaussPosition_d[currentIndex] -
     gaussPositionOffset;
40     int imageIndex = idx*imageSize;
41
42     /*Loop through each group of 6 coefficients.*/
43     for (int i=0;i<numberOfCoefficients_d[currentIndex];i++)
44     {
45       /*Loop through each of the coefficients. (Amplitude, mx, my,sx
     ,sy,rho).*/
46       for (int counter=0;counter<6;counter++)
47       {
48         x[counter] = gaussianCoefficients_d[gaussIndex + i*6 +
     counter];
49         if (counter==1 || counter == 2)
50           x[counter] = x[counter] - 1.0f; /*Convert from MATLAB
     coordinates to C.*/
51       }
52       /*All the coefficients are loaded for a single projection.*/
53       for (int yCounter=floor(x[2]-num_sigma*x[4]);yCounter<=ceil(x
     [2] + num_sigma*x[4]);yCounter++)
54       {
55         if ((yCounter>=0) && (yCounter<IMAGE_DIMY))
56         {
57           for (int xCounter=floor(x[1]-num_sigma*x[3]);xCounter<=
     ceil(x[1] + num_sigma*x[3]);xCounter++)
58           {
59             if((xCounter >=0) && (xCounter < IMAGE_DIMX))
60             {
61               images_d[imageIndex + (IMAGE_DIMX)*(yCounter) +
     xCounter] += gaussKernelOther(x,(float)xCounter,(float)yCounter);
62             }
63           }
64         }
65       }
66     }
67   }
68 }
69
70 /*Used for Implementation 2 of MLEM (mlem_recon2)*/
71 __global__ void createPSFFast2(float *gaussianCoefficients_d,float *
     image_d,int N,float *currentPositionEstimate,int index)
72 {
73   extern __shared__ float sharedGaussianCoefficient[];
74   if (threadIdx.x < 6)
75   {
76     if (threadIdx.x == 0)
77     {
78       sharedGaussianCoefficient[0] = currentPositionEstimate[index]*
     gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x];
79     }
80     else if (threadIdx.x == 1 || threadIdx.x == 2)
81     {
```

```
82        sharedGaussianCoefficient[threadIdx.x] =
     gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x] - 1.0f;
83      }
84      else{
85        sharedGaussianCoefficient[threadIdx.x] =
     gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x];
86      }
87    }
88    __syncthreads();
89
90    if (threadIdx.x < N && threadIdx.y < N)
91    {
92      int y = sharedGaussianCoefficient[2] - (N/2.0f) + threadIdx.y +
     0.5f;
93      int x = sharedGaussianCoefficient[1] - (N/2.0f) + threadIdx.x +
     0.5f;
94
95      if (x >= 0 && x <IMAGE_DIMX && y>=0 && y<IMAGE_DIMY)
96      {
97        image_d[IMAGE_DIMX*y + x] += gaussKernelOther(
     sharedGaussianCoefficient,x,y);
98      }
99    }
100
101 }
102
103 /*Used for Implementation 2 of MLEM (mlem_recon2). Similar to above.
     */
104 __global__ void createPSFFast2WithH(float *gaussianCoefficients_d,
     float *image_d,int N)
105 {
106   extern __shared__ float sharedGaussianCoefficient[];
107   if (threadIdx.x < 6)
108   {
109     if (threadIdx.x == 1 || threadIdx.x == 2)
110     {
111       sharedGaussianCoefficient[threadIdx.x] =
     gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x] - 1.0f;
112     }
113     else{
114       sharedGaussianCoefficient[threadIdx.x] =
     gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x];
115     }
116   }
117   __syncthreads();
118
119   if (threadIdx.x < N && threadIdx.y < N)
120   {
121     int y = sharedGaussianCoefficient[2] - (N/2.0f) + threadIdx.y +
     0.5f;
122     int x = sharedGaussianCoefficient[1] - (N/2.0f) + threadIdx.x +
     0.5f;
123
124     if (x >= 0 && x <IMAGE_DIMX && y>=0 && y<IMAGE_DIMY)
125     {
126       image_d[IMAGE_DIMX*y + x] += gaussKernelOther(
     sharedGaussianCoefficient,x,y);
```

```
127       }
128     }
129 }
130
131 /*Create the PSF using shared memory. Used in POSEM*/
132 __global__ void createPSFFast3(int *pixelIndices,int
      numberOfCoefficients,float *gaussianCoefficients_d,float *image_d
      ,int N)
133 {
134   int idx = blockIdx.x*blockDim.x+threadIdx.x;
135   extern __shared__ float sharedGaussianCoefficient[];
136
137   int coeffPerThread = ceil((float)numberOfCoefficients*6/blockDim.x
      );
138   if (threadIdx.x*coeffPerThread < numberOfCoefficients*6)
139   {
140     for (int i=0;i<coeffPerThread;i++)
141     {
142       if (threadIdx.x*coeffPerThread+i < numberOfCoefficients*6)
143         sharedGaussianCoefficient[threadIdx.x*coeffPerThread+i] =
      gaussianCoefficients_d[threadIdx.x*coeffPerThread+i];
144     }
145   }
146   __syncthreads();
147
148   if (idx < N)
149   {
150     int y = (int)floor((float)pixelIndices[idx]/(float)IMAGE_DIMX) ;
151     int x = pixelIndices[idx]%IMAGE_DIMX;
152
153     /*Loop through each group of 6 coefficients.*/
154     for (int i=0;i<numberOfCoefficients;i++)
155     {
156       float xSq = (x - (sharedGaussianCoefficient[i*6+1]))*(x - (
      sharedGaussianCoefficient[i*6+1]));
157       float ySq = (y - (sharedGaussianCoefficient[i*6+2]))*(y - (
      sharedGaussianCoefficient[i*6+2]));
158       if (sqrtf(xSq+ySq)<10.0f)
159         image_d[idx] += gaussKernelOther2(&sharedGaussianCoefficient
      [i*6],xSq,ySq,x,y);
160     }
161   }
162 }
```

A.6 MLEM

A.6.1 Functions

```
1 /*This function adds up all the pixels in each image to get the
      response of the detector to each voxel.
2 Response will have the same size as numberOfImages.
3 N is the numberOfImages. Very inefficient implementation. */
```

```
4  __global__ void voxelResponse(float *response_d,float *images_d,int
       *mask,int imageSize,int numberOfVoxelsToProcess,int N)
5  {
6    int idx = blockIdx.x*blockDim.x+threadIdx.x;
7
8    if (idx < N)
9    {
10     response_d[idx] = 0.0f;
11     for (int i=0;i<imageSize;i++)
12     {
13       if (mask[i] == 1)
14       {
15         response_d[idx] += images_d[idx*imageSize + i];
16       }
17     }
18
19   }
20 }
21
22 /*MLEM Project and Backproject.*/
23 void mlem_recon(float *sensitivity_h,float *detectorSum_d,float *
       currentPositionEstimate_d,int *numFloats,int numberOfImages,float
        *g,int *numberOfCoefficients_h,float *gaussianCoefficients_h,int
        *gaussPosition_h,float *images_h,float *images_d,int imageSize,
       int numberOfVoxelsToProcess,int voxelSpaceSize)
24 {
25
26   int i=0;
27
28   /*Projections to hold on GPU.*/
29   float *proj_d;
30   CUDA_CALL(cudaMalloc((void**)&proj_d,sizeof(float)*imageSize*
       numberOfImages));
31   for (int imC=0;imC<numberOfImages;imC++)
32     fl_setToZeroCUDA<<<(int)ceil((float)imageSize/512.0f),512 >>>(&
       proj_d[imC*imageSize], imageSize);
33
34   /*Coefficients.*/
35   int numberOfVoxelsToProcessTemp = 0;
36   int *numberOfCoefficients_d;
37   float *gaussianCoefficients_d; /*Memory not declared yet.*/
38   int *gaussPosition_d;
39   CUDA_CALL(cudaMalloc((void**)&numberOfCoefficients_d,sizeof(int)*
       voxelSpaceSize));
40   CUDA_CALL(cudaMalloc((void**)&gaussPosition_d,sizeof(int)*
       voxelSpaceSize));
41
42   /*Masks.*/
43   int *mask_h;
44   mask_h = (int *) malloc(sizeof(int)*imageSize);
45   i_resetToValue(mask_h,imageSize,1);
46   i_applyCircularMask(mask_h);
47   int *mask;
48   CUDA_CALL(cudaMalloc((void**)&mask,sizeof(int)*imageSize));
49   CUDA_CALL(cudaMemcpy(mask,mask_h,sizeof(int)*imageSize,
       cudaMemcpyHostToDevice));
50
```

```
51  /*Send g to the device.*/
52  float *g_d;
53  CUDA_CALL(cudaMalloc((void**)&g_d,sizeof(float)*imageSize*
     numberOfImages));
54  CUDA_CALL(cudaMemcpy(g_d,g,sizeof(float)*imageSize*numberOfImages,
     cudaMemcpyHostToDevice));
55
56  float *H_d;
57  CUDA_CALL(cudaMalloc((void**)&H_d,sizeof(float)*imageSize*
     numberOfImages));
58
59  for (int imageNumber=0;imageNumber<numberOfImages;imageNumber++)
60  {
61    i=0;
62    /*Get offset for the coefficient file.*/
63    int gaussPositionOffset = gaussPosition_h[imageNumber*
     voxelSpaceSize];
64
65    /*Declare memory for coefficients now. This changes in each loop
     .*/
66    CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d,sizeof(
     float)*numFloats[imageNumber]));
67
68    /*Copy the data from the host to the device.  */
     CUDA_CALL(cudaMemcpy(gaussianCoefficients_d,&
     gaussianCoefficients_h[gaussPosition_h[imageNumber*voxelSpaceSize
     ]],sizeof(float)*numFloats[imageNumber],cudaMemcpyHostToDevice));
69    CUDA_CALL(cudaMemcpy(numberOfCoefficients_d,&
     numberOfCoefficients_h[imageNumber*voxelSpaceSize],sizeof(int)*
     voxelSpaceSize,cudaMemcpyHostToDevice));
70  CUDA_CALL(cudaMemcpy(gaussPosition_d,&gaussPosition_h[imageNumber*
     voxelSpaceSize],sizeof(int)*voxelSpaceSize,cudaMemcpyHostToDevice
     ));
71
72    numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;
73    while (i < voxelSpaceSize)
74    {
75      /*Make sure it doesn't go out of bounds.*/
76      if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
77      {
78        numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
79      }
80      /*Set all projection images to zero.*/
81      for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
82        fl_setToZeroCUDA<<<(int)ceil((float)imageSize/512.0f),512
     >>>(&images_d[imC*imageSize], imageSize);
83
84    /*Create the projection image for a number of voxels.*/
        createPSFFast<<<(int)ceil(float(numberOfVoxelsToProcessTemp)
     /((128.0f))),128>>>(numberOfCoefficients_d,gaussianCoefficients_d
     ,gaussPosition_d,images_d,imageSize,i,imageNumber,voxelSpaceSize,
     gaussPositionOffset,numberOfVoxelsToProcessTemp);
85
86      /*Multiply the projection images by the current estimate and
     add them all together.*/
87      for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
```

```
88          fl_addMatricesWithWeightsCUDA<<<(int)ceil((float)imageSize
    /512.0f),512,sizeof(float) >>>(&proj_d[imageNumber*imageSize],&
    images_d[imC*imageSize],currentPositionEstimate_d,i+imC,imageSize
    );
89
90          i = i + numberOfVoxelsToProcessTemp;
91        }
92        CUDA_CALL(cudaFree(gaussianCoefficients_d));
93      }
94
95      /*Mask the data.*/
96      for (int j=0;j<numberOfImages;j++)
97        fl_applyMaskCUDA<<<(int)ceil(float(imageSize)/(float(512)))
    ,512>>>(&proj_d[j*imageSize],mask,imageSize);
98
99  /*      Diagnostic Test!
100     float *proj_h;
101     proj_h = (float *)malloc(sizeof(float)*imageSize);
102     CUDA_CALL(cudaMemcpy(proj_h,proj_d,sizeof(float)*imageSize,
        cudaMemcpyDeviceToHost));
103     FILE *out;
104     out = fopen("E:\\TEMP\\NEWMLEMPROJ.dat","ab");
105     fwrite(proj_h,imageSize,sizeof(float),out);
106     fclose(out);
107 */
108
109     /* Divide g by proj_d and send the result to H_d*/
110     for (int j=0;j<numberOfImages;j++)
111       fl_divideMatricesCUDA<<<(int)ceil((float)imageSize/512.0f)
    ,512>>>(&H_d[j*imageSize],&g_d[j*imageSize],&proj_d[j*imageSize],
    imageSize);
112
113     fl_setToZeroCUDA<<<(int)ceil((float)voxelSpaceSize/512.0f),512
    >>>(detectorSum_d, voxelSpaceSize);
114
115 /*Similar to above.*/
116     for (int imageNumber = 0;imageNumber<numberOfImages;imageNumber++)
117     {
118
119       int gaussPositionOffset = gaussPosition_h[imageNumber*
    voxelSpaceSize];
120       CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d,sizeof(
    float)*numFloats[imageNumber]));
121       CUDA_CALL(cudaMemcpy(gaussianCoefficients_d,&
    gaussianCoefficients_h[gaussPosition_h[imageNumber*voxelSpaceSize
    ]],sizeof(float)*numFloats[imageNumber],cudaMemcpyHostToDevice));
122       CUDA_CALL(cudaMemcpy(numberOfCoefficients_d,&
    numberOfCoefficients_h[imageNumber*voxelSpaceSize],sizeof(int)*
    voxelSpaceSize,cudaMemcpyHostToDevice));
123       CUDA_CALL(cudaMemcpy(gaussPosition_d,&gaussPosition_h[
    imageNumber*voxelSpaceSize],sizeof(int)*voxelSpaceSize,
    cudaMemcpyHostToDevice));
124
125       i=0;
126       numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;
127       while (i < voxelSpaceSize)
128       {
```

```
129         if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
130         {
131           numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
132         }
133 
134         for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
135           fl_setToZeroCUDA<<<(int)ceil((float)imageSize/512.0f),512
      >>>(&images_d[imC*imageSize], imageSize);
136     createPSFFast<<<(int)ceil(float(numberOfVoxelsToProcessTemp)/(
      float(128))),128>>>(numberOfCoefficients_d, gaussianCoefficients_d
      , gaussPosition_d, images_d, imageSize, i, imageNumber, voxelSpaceSize,
      gaussPositionOffset, numberOfVoxelsToProcessTemp);
137 
138         for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
      fl_multiplyMatricesCUDA<<<(int)ceil(float(imageSize)/(512.0f))
      ,512>>>(&images_d[imC*imageSize],&H_d[imageNumber*imageSize],
      imageSize);
139 
140     mlem_sumImages<<<(int)ceil(float(numberOfVoxelsToProcessTemp)/(
      float(128))),128>>>(&detectorSum_d[i], images_d, imageSize,
      numberOfVoxelsToProcessTemp);
141         i = i + numberOfVoxelsToProcessTemp;
142       }
143 
144       CUDA_CALL(cudaFree(gaussianCoefficients_d));
145     }
146 
147     /*Free the variables.*/
148     cudaFree(gaussPosition_d);
149     cudaFree(numberOfCoefficients_d);
150     cudaFree(proj_d);
151     cudaFree(H_d);
152     cudaFree(mask);
153     free(mask_h);
154 
155 }
156 
157 /*Implementation 2 of MLEM -> Cluster-by-Cluster method*/
158 void mlem_recon2(float *H_d, int *numberOfCoefficients_d, int *
      gaussPosition_d, float *proj_d, int *mask, float *sensitivity_h,
      float *detectorSum_d, float *detectorSum_h, float *
      currentPositionEstimate_d, int *numFloats, int numberOfImages, float
       *g_d, int *numberOfCoefficients_h, float *gaussianCoefficients_h,
      int *gaussPosition_h, float *images_h, float *images_d, int
      imageSize, int numberOfVoxelsToProcess, int voxelSpaceSize)
159 {
160   int i=0;
161   syncDeviceAndErrorCheck();
162   for (int imC=0;imC<numberOfImages;imC++)
163     fl_setToZeroCUDA<<<(int)ceil((float)imageSize/512.0f),512 >>>(&
      proj_d[imC*imageSize], imageSize);
164 
165   int numberOfVoxelsToProcessTemp = 0;
166   float *gaussianCoefficients_d;
167 
168   dim3 bl_Size;
169   bl_Size.x = 11; bl_Size.y = 11;
```

```
170
171   for (int imageNumber=0;imageNumber<numberOfImages;imageNumber++)
172   {
173     i=0;
174     int gaussPositionOffset = gaussPosition_h[imageNumber*
      voxelSpaceSize];
175     CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d,sizeof(
      float)*numFloats[imageNumber]));
176         CUDA_CALL(cudaMemcpy(gaussianCoefficients_d,&
      gaussianCoefficients_h[gaussPosition_h[imageNumber*voxelSpaceSize
      ]],sizeof(float)*numFloats[imageNumber],cudaMemcpyHostToDevice));
177
178     numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;
179     while (i < voxelSpaceSize)
180     {
181       if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
182       {
183         numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
184       }
185
186       for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
187       {
188         int gaussIndex = gaussPosition_h[imageNumber*voxelSpaceSize
      + i+imC] - gaussPositionOffset;
189         int numberOfCoeff = numberOfCoefficients_h[imageNumber*
      voxelSpaceSize + i + imC];
190         dim3 num_bl;
191         num_bl.x = numberOfCoeff;
192         num_bl.y = 1;
193         if (numberOfCoeff > 0)
194           createPSFFast2<<<num_bl,bl_Size,6*sizeof(float)>>>(&
      gaussianCoefficients_d[gaussIndex],&proj_d[imageNumber*imageSize
      ],bl_Size.x,currentPositionEstimate_d,i+imC);
195
196       }
197       i = i + numberOfVoxelsToProcessTemp;
198     }
199
200     CUDA_CALL(cudaFree(gaussianCoefficients_d));
201   }
202
203   for (int j=0;j<numberOfImages;j++)
204     fl_applyMaskCUDA<<<(int)ceil(float(imageSize)/(float(512)))
      ,512>>>(&proj_d[j*imageSize],mask,imageSize);
205
206   // Divide g by proj_d and send the result to H_d
207   for (int j=0;j<numberOfImages;j++)
208     fl_divideMatricesCUDA<<<(int)ceil((float)imageSize/512.0f)
      ,512>>>(&H_d[j*imageSize],&g_d[j*imageSize],&proj_d[j*imageSize],
      imageSize);
209
210   fl_setToZeroCUDA<<<(int)ceil((float)voxelSpaceSize/512.0f),512
      >>>(detectorSum_d, voxelSpaceSize);
211
212   for (int imageNumber = 0;imageNumber<numberOfImages;imageNumber++)
213   {
214
```

```
215     int gaussPositionOffset = gaussPosition_h[imageNumber*
        voxelSpaceSize];
216     CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d,sizeof(
        float)*numFloats[imageNumber]));
217     CUDA_CALL(cudaMemcpy(gaussianCoefficients_d,&
        gaussianCoefficients_h[gaussPosition_h[imageNumber*voxelSpaceSize
        ]],sizeof(float)*numFloats[imageNumber],cudaMemcpyHostToDevice));
218
219     i=0;
220     numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;
221     while (i < voxelSpaceSize)
222     {
223       if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
224       {
225         numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
226       }
227       for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
228       {
229
230       fl_setToZeroCUDA<<<(int)ceil((float)imageSize/512.0f),512
        >>>(&images_d[0*imageSize], imageSize);
231
232
233         int gaussIndex = gaussPosition_h[imageNumber*voxelSpaceSize
        + i+imC] - gaussPositionOffset;
234         int numberOfCoeff = numberOfCoefficients_h[imageNumber*
        voxelSpaceSize + i + imC];
235         dim3 num_bl;
236         num_bl.x = numberOfCoeff;
237         num_bl.y = 1;
238         if (numberOfCoeff > 0)
239         createPSFFast2WithH<<<num_bl,bl_Size,6*sizeof(float)>>>(&
        gaussianCoefficients_d[gaussIndex],&images_d[0*imageSize],bl_Size
        .x);
240
241         getSumArrayGPUtoGPUWithMultiplication(&images_d[0*imageSize
        ],&H_d[imageNumber*imageSize], detectorSum_d,i+imC, imageSize);
242
243       }
244
245       i = i + numberOfVoxelsToProcessTemp;
246     }
247
248     CUDA_CALL(cudaFree(gaussianCoefficients_d));
249   }
250 }
251
252
253 void totalSensitivity(float *sensitivity,int *imageNumbers,int *
      numFloats,int numberOfImages,int *numberOfCoefficients_h,float *
      gaussianCoefficients_h,int *gaussPosition_h,float *images_h,float
       *images_d,int imageSize,int numberOfVoxelsToProcess,int
      voxelSpaceSize)
254 {
255
256   /*Diagnostic Output.*/
257   FILE *foutH;
```

```
258   foutH = fopen("H_proj_Speedtest.dat","wb");
259
260   float *response_d;
261   cudaMalloc((void**)&response_d,sizeof(float)*
       numberOfVoxelsToProcess);
262   fl_setToZeroCUDA<<<(int)ceil((float)numberOfVoxelsToProcess/512)
       ,512 >>>(response_d, numberOfVoxelsToProcess);
263
264   int numberOfVoxelsToProcessTemp = 0;
265
266   /*Declare the temporary variables for the GPU!*/
267   int *numberOfCoefficients_d;
268   float *gaussianCoefficients_d;
269   int *gaussPosition_d;
270   cudaMalloc((void**)&numberOfCoefficients_d,sizeof(int)*
       voxelSpaceSize);
271   cudaMalloc((void**)&gaussPosition_d,sizeof(int)*voxelSpaceSize);
272
273   /*Mask.*/
274   int *mask_h;
275   mask_h = (int*)malloc(imageSize*sizeof(int));
276
277   int *mask_d;
278   cudaMalloc((void**)&mask_d,sizeof(int)*imageSize);
279   i_resetToValue(mask_h,imageSize,1);
280   if (APPLYMASK == 1)
281   {
282     i_applyCircularMask(mask_h);
283   }
284   CUDA_CALL(cudaMemcpy(mask_d,mask_h,sizeof(int)*imageSize,
       cudaMemcpyHostToDevice));
285   cout << "Variables are declared." << endl;
286
287
288   clock_t start,end;
289   for (int k=0;k<numberOfImages;k++)
290   {
291     /*Get offset for the coefficient file.*/
292     int gaussPositionOffset = gaussPosition_h[k*voxelSpaceSize];
293
294     CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d,sizeof(
       float)*numFloats[k]));
295     CUDA_CALL(cudaMemcpy(gaussianCoefficients_d,&
       gaussianCoefficients_h[gaussPosition_h[k*voxelSpaceSize]],sizeof(
       float)*numFloats[k],cudaMemcpyHostToDevice));
296     CUDA_CALL(cudaMemcpy(numberOfCoefficients_d,&
       numberOfCoefficients_h[k*voxelSpaceSize],sizeof(int)*
       voxelSpaceSize,cudaMemcpyHostToDevice));
297     CUDA_CALL(cudaMemcpy(gaussPosition_d,&gaussPosition_h[k*
       voxelSpaceSize],sizeof(int)*voxelSpaceSize,cudaMemcpyHostToDevice
       ));
298
299     /*Diagnostic  Output.*/
300     /*cout << "gaussPositionOffset: " << gaussPositionOffset << ".
       numFloats: " << numFloats[k] << endl;*/
301
302     int i=0;
```

```
303       start = clock();
304       numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;
305
306       while (i < voxelSpaceSize)
307       {
308         if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
309         {
310           numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
311         }
312
313         for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
314           fl_setToZeroCUDA<<<(int)ceil((float)imageSize/512.0f),512
      >>>(&images_d[imC*imageSize], imageSize);
315       createPSFFast<<<(int)ceil((float)numberOfVoxelsToProcessTemp
      /128),128>>>(numberOfCoefficients_d, gaussianCoefficients_d,
      gaussPosition_d, images_d, imageSize, i, k, voxelSpaceSize,
      gaussPositionOffset, numberOfVoxelsToProcessTemp);
316
317         for (int n=0;n<numberOfVoxelsToProcessTemp;n++)
318       fl_applyMaskCUDA<<<(int)ceil(float(imageSize)/(float(512)))
      ,512>>>(&images_d[n*imageSize], mask_d, imageSize);
319
320 fl_setToZeroCUDA<<<(int)ceil((float)numberOfVoxelsToProcessTemp/128)
      ,128 >>>(response_d, numberOfVoxelsToProcessTemp);
321
322         mlem_sumImages<<<(int)ceil(float(numberOfVoxelsToProcessTemp)
      /128),128>>>(response_d, images_d, imageSize,
      numberOfVoxelsToProcessTemp);
323
324       CUDA_CALL(cudaMemcpy(&sensitivity[(voxelSpaceSize * k) + i],
      response_d, sizeof(float)*numberOfVoxelsToProcessTemp,
      cudaMemcpyDeviceToHost));
325         i = i + numberOfVoxelsToProcessTemp;
326       }
327
328     /*Voxel position check.*/
329     cout << "i: " << i << ", VoxelSpaceSize: " << voxelSpaceSize <<
       endl;
330     end=clock();
331     cout << "sensitivity: " << (double)(end - start)/CLOCKS_PER_SEC <<
        endl;
332     cudaFree(gaussianCoefficients_d);
333     cout << "H("<< k+1 << "/" << numberOfImages << ") Complete." <<
       endl;
334     }
335
336     cout << "Complete" <<endl;
337     fwrite(sensitivity, voxelSpaceSize*numberOfImages, sizeof(float),
       foutH);
338     fclose(foutH);
339
340
341     cudaFree(response_d);
342     cudaFree(numberOfCoefficients_d);
343     cudaFree(gaussPosition_d);
344     cudaFree(mask_d);
345     free(mask_h);
```

```
346
347 }
348
349 void invertSensitivity(float *sensitivity, int voxelSpaceSize, int
    numberOfImages)
350 {
351   cout <<"Invert Sensitivity." << endl;
352
353   float tempMax = 0.0f;
354   for (int i = 0; i < voxelSpaceSize; i++){
355     for (int j=0;j<numberOfImages;j++)
356     {
357       if (sensitivity[j*voxelSpaceSize + i] > tempMax){
358         sensitivity[j*voxelSpaceSize + i] = (1.0f/sensitivity[j*
    voxelSpaceSize +  i]);
359       }
360       else{
361         sensitivity[j*voxelSpaceSize + i] = 0.0f;
362       }
363     }
364   }
365
366   float tmpSum=0.0f;
367   for (int ii = 0; ii < voxelSpaceSize*numberOfImages; ii++){
368     tmpSum += sensitivity[ ii];
369   }
370   printf("Sensitivities loaded, Sum = %0.2f\n",tmpSum);
371 }
```

A.6.2 MLEM Main

```
1 int main(int argc, char *argv[])
2 {
3
4   int voxelSpaceSize = DIMX*DIMY*DIMZ;
5   int numberOfImages=0;
6   numberOfImages = NUMBEROFIMAGES;
7
8   const int NoI = NUMBEROFIMAGES;
9   int stopper = 0;
10
11  /*Open the coefficient file.*/
12  FILE *fn[6];
13
14  /*97*97*97 cubic grid.*/
15  fn[0] = fopen("E:\\PSF\\H\\gaussParams_00mm_082012_v3_PE_raw_cubic
    .dat","rb");
16  fn[1] = fopen("E:\\PSF\\H\\gaussParams_08mm_082012_v3_PE_raw_cubic
    .dat","rb");
17  fn[2] = fopen("E:\\PSF\\H\\gaussParams_16mm_082012_v3_PE_raw_cubic
    .dat","rb");
18  fn[3] = fopen("E:\\PSF\\H\\gaussParams_24mm_082012_v3_PE_raw_cubic
    .dat","rb");
```

```
19  fn[4] = fopen("E:\\PSF\\H\\gaussParams_27mm_082012_v3_PE_raw_cubic
     .dat","rb");
20  fn[5] = fopen("E:\\PSF\\H\\gaussParams_37mm_082012_v3_PE_raw_cubic
     .dat","rb");
21
22  /*61*61*61 cubic grid.
23  fn[0] = fopen("E:\\PSF\\H\\
     gaussParams_00mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
24  fn[1] = fopen("E:\\PSF\\H\\
     gaussParams_08mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
25  fn[2] = fopen("E:\\PSF\\H\\
     gaussParams_16mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
26  fn[3] = fopen("E:\\PSF\\H\\
     gaussParams_24mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
27  fn[4] = fopen("E:\\PSF\\H\\
     gaussParams_27mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
28  fn[5] = fopen("E:\\PSF\\H\\
     gaussParams_37mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");*/
29
30  /*Output Root.*/
31  FILE *fileOut;
32  string rootOut;
33  rootOut.assign("E:\\Phantom Study\\6-21-12\\Run3\\");
34
35  /*Projection data.*/
36  FILE *fil1;
37  fil1 = fopen("E:\\TEMP\\PSFproj_rods_reservoir.dat","rb");
38
39
40  printf("File...open\n");
41  int h1[3];
42  /*Read the 3 int header of the coefficient files.*/
43  printf("File...read\n");
44  for (int fileCnt=0;fileCnt<numberOfImages;fileCnt++)
45  {
46    fread(&h1[0],sizeof(int),3,fn[fileCnt]);
47    printf("%d,%d,%d\n",h1[0],h1[1],h1[2]);
48  }
49  int imageSize = IMAGE_DIMX*IMAGE_DIMY;
50
51  int *imageNumbers;
52  imageNumbers=(int *)malloc(numberOfImages*sizeof(int));
53
54  for (int i=0;i<numberOfImages;i++)
55  {
56    imageNumbers[i] = i;
57    printf("imageNumbers[%d] = %d\n",i,imageNumbers[i]);
58  }
59
60  float *g;
61  g=(float *)malloc(numberOfImages*imageSize*sizeof(float));
62  resetToZero(g,numberOfImages*imageSize);
63
64  /*Read in the 3 int headers to the projection data.*/
65  int tmp[3];
66  fread(&tmp[0], sizeof(int), 3, fil1);
67
```

```
68   printf("Image Header: %d %d %d\n Imagesize = %d, numberOfImages =
     %d\n",tmp[0],tmp[1],tmp[2],imageSize,numberOfImages);
69
70   float *H;
71   H=(float *)malloc(imageSize*sizeof(float));
72   resetToZero(H,imageSize);
73
74   /*Read in the projection images.*/
75   float *g_tmp;
76   g_tmp=(float *)malloc(numberOfImages*imageSize*sizeof(float));
77   fread(g_tmp, sizeof(float),numberOfImages*imageSize, fil1);
78
79   float sumcheck=0.0f; /*Diagnostic Variable.*/
80   for (int i=0;i<numberOfImages;i++)
81   {
82     printf("Loading Image : %d ...",imageNumbers[i]);
83     for (int j=0;j<imageSize;j++)
84     {
85       g[i*imageSize + j] = g_tmp[j + ((imageNumbers[i])*imageSize)];
86       sumcheck = sumcheck + g[i*imageSize + j];
87     }
88
89     if (APPLYMASK == 1)
90     {
91       fl_applyCircularMask(&g[i*imageSize]);
92     }
93     printf("sumcheck = %.4f!\n",sumcheck);
94     sumcheck = 0.0f;
95   }
96
97   cout << "All images loaded." << endl;
98
99   /*Diagnostic Output.*/
100  FILE *foutH;
101  foutH = fopen("g.dat","wb");
102  fwrite(g,imageSize*numberOfImages,sizeof(float),foutH);
103  fclose(foutH);
104
105
106  /*Allocate memory to store all of the coefficients.*/
107  int *numFloats;
108  numFloats = (int *)malloc(sizeof(int)*NoI);
109  for (int fileCnt=0;fileCnt<numberOfImages;fileCnt++)
110  {
111    numFloats[fileCnt] = getSize(fn[fileCnt]);
112    printf("File Number: %d, Count: %d\n",fileCnt,numFloats[fileCnt
     ]);
113    rewind(fn[fileCnt]);
114    fread(&h1[0],sizeof(int),3,fn[fileCnt]);
115    printf("%d,%d,%d\n",h1[0],h1[1],h1[2]);
116  }
117
118  int totalFloats = 0;
119  for (int fileCnt=0;fileCnt<numberOfImages;fileCnt++)
120  {
121    totalFloats += numFloats[fileCnt];
122  }
```

```
123   cout << "Total Floats: " << totalFloats << endl;
124
125   /*Host Memory.*/
126   printf("Allocate Host Memory.\n");
127   int *numberOfCoefficients_h;
128   numberOfCoefficients_h = (int *)malloc(sizeof(int)*voxelSpaceSize*
       numberOfImages);
129   float *gaussianCoefficients_h;
130   gaussianCoefficients_h = (float *)malloc(sizeof(float)*totalFloats
       );
131   int *gaussPosition_h;
132   gaussPosition_h = (int *)malloc(sizeof(int)*voxelSpaceSize*
       numberOfImages);
133
134   for (int k=0;k<numberOfImages;k++)
135   {
136     /*Read in the coefficients.*/
137     int filePosTemp=0;
138     for (int j=0;j<k;j++)
139     {
140       filePosTemp += numFloats[j];
141     }
142     cout << "FILEPOS: " << filePosTemp << endl;
143
144     loadCoefficients(fn[k],&numberOfCoefficients_h[k*voxelSpaceSize
       ],&gaussianCoefficients_h[filePosTemp]);
145
146     printf("Coefficients Loaded.\n");
147
148     getGaussPosition(&gaussPosition_h[k*voxelSpaceSize],&
       numberOfCoefficients_h[k*voxelSpaceSize],filePosTemp);
149     printf("Gauss Positions Loaded.\n");
150     rewind(fn[k]);
151     fread(&h1[0],sizeof(int),3,fn[k]);
152   }
153
154
155   clock_t start,end;
156
157   /*If the image size is 320x240. The coefficients are stored for a
       640x480 image.*/
158   for (int j=0;j<totalFloats;j=j+6)
159   {
160     if (IMAGE_DIMX == 320)
161     {
162       gaussianCoefficients_h[j+1] = gaussianCoefficients_h[j+1]/2.0f
       ;
163       gaussianCoefficients_h[j+2] = gaussianCoefficients_h[j+2]/2.0f
       ;
164       gaussianCoefficients_h[j+3] = gaussianCoefficients_h[j+3]/2.0f
       ;
165       gaussianCoefficients_h[j+4] = gaussianCoefficients_h[j+4]/2.0f
       ;
166     }
167   }
168
```

```
169   /*The closest voxels to the collimator have unreliable data due to
        excess pileup. */
170   /*Set their intensities to zero.*/
171   if (FILTER_PSF == 1)
172   {
173     int counter = 0;
174     int singleCounter=0;
175     for (int j=0;j<numberOfImages;j++)
176     {
177       int voxelCounter = 0;
178
179       for (int x=0;x<DIMX;x++)
180       {
181         for (int y=0;y<DIMY;y++)
182         {
183           for (int z=0;z<DIMZ;z++)
184           {
185             if (z>=DIMZ-7)
186             {
187               counter = 0;
188               for (int k=0;k<numberOfCoefficients_h[j*voxelSpaceSize
      +voxelCounter];k++)
189               {
190                 gaussianCoefficients_h[gaussPosition_h[j*
      voxelSpaceSize+voxelCounter]+ counter] = 0.0f;
191                 counter = counter + 6;
192               }
193
194             }
195             voxelCounter++;
196             singleCounter++;
197           }
198         }
199       }
200     }
201   }
202
203   /*How many voxels can be processed at a time. 3 GB GPU.*/
204   printf("Allocating to GPU.\n");
205   int numberOfVoxelsToProcess;
206   if (IMAGE_DIMX == 640)
207   {
208     numberOfVoxelsToProcess = 1536;
209   }
210   else if(IMAGE_DIMX == 320)
211   {
212     numberOfVoxelsToProcess = 6144;
213   }
214
215   float *sensitivityProj;
216   sensitivityProj=(float *)malloc(numberOfImages*voxelSpaceSize*
       sizeof(float));
217   float *sensitivityProj_d;
218   CUDA_CALL(cudaMalloc((void**)&sensitivityProj_d,sizeof(float)*
       numberOfImages*voxelSpaceSize));
219   resetToValue(sensitivityProj,numberOfImages*voxelSpaceSize,0.0f);
220
```

```
221   float *detectorSum_h;
222   detectorSum_h=(float *)malloc(voxelSpaceSize*sizeof(float));
223   float *detectorSum_d;
224   CUDA_CALL(cudaMalloc((void**)&detectorSum_d,sizeof(float)*
       voxelSpaceSize));
225
226   /*Initialize to zero.*/
227   resetToZero(detectorSum_h,voxelSpaceSize);
228   /*Copy across to GPU.*/
229   CUDA_CALL(cudaMemcpy(detectorSum_d,detectorSum_h,sizeof(float)*
       voxelSpaceSize,cudaMemcpyHostToDevice));
230   cout << "Complete" << endl;
231
232   /*How much space for images is available on the device?*/
233   size_t avail;
234   size_t total;
235   cudaMemGetInfo(&avail,&total);
236   cout << "Total: "<< total <<"\nAvailable: "<< avail << endl;
237
238   /*Declare images on the CPU and GPU.*/
239   float *images_h;
240   images_h = (float *)malloc(sizeof(float)*numberOfVoxelsToProcess*
       imageSize);
241   float *images_d;
242   CUDA_CALL(cudaMalloc((void**)&images_d,sizeof(float)*
       numberOfVoxelsToProcess*imageSize));
243   /*Initialize to zero.*/
244   resetToZero(images_h,numberOfVoxelsToProcess*imageSize);
245   /*Copy across to GPU.*/
246     CUDA_CALL(cudaMemcpy(images_d,images_h,sizeof(float)*
       numberOfVoxelsToProcess*imageSize,cudaMemcpyHostToDevice));
247
248   /*Much memory left after that transfer?*/
249   cudaMemGetInfo(&avail,&total);
250   cout << "Total: "<< total <<"\nAvailable: "<< avail << endl;
251
252   printf("Begin GPU Processing?: (Write:1, Process:0)");
253   float *mlem_detectorSum_h;
254   float *mlem_detectorSum_d;
255
256   /*ALGORITHM
257   MLEM = 0;
258   POSEM = 1;
259   IMAGE OSEM = 2; //Not implemented.
260   LANDWEBER = 3; //Not implemented.
261   */
262
263 /*Sensitivity of each voxel. */
264     totalSensitivity(sensitivityProj,imageNumbers,numFloats,
       numberOfImages,numberOfCoefficients_h,gaussianCoefficients_h,
       gaussPosition_h,images_h,images_d,imageSize,
       numberOfVoxelsToProcess,voxelSpaceSize);
265   cout << "Exit sensitivity function." << endl;
266   float *mlem_sensitivityProj;
267
268   mlem_detectorSum_h=(float *)malloc(voxelSpaceSize*sizeof(float));
```

```
269  CUDA_CALL(cudaMalloc((void**)&mlem_detectorSum_d,sizeof(float)*
       voxelSpaceSize));
270  resetToZero(mlem_detectorSum_h,voxelSpaceSize);
271    /*Copy across to GPU.*/
272      CUDA_CALL(cudaMemcpy(mlem_detectorSum_d,mlem_detectorSum_h,
       sizeof(float)*voxelSpaceSize,cudaMemcpyHostToDevice));
273    mlem_sensitivityProj=(float *)malloc(voxelSpaceSize*sizeof(float
       ));
274    resetToZero(mlem_sensitivityProj,voxelSpaceSize);
275
276    for (int i=0;i<numberOfImages;i++)
277    {
278      for (int j=0;j<voxelSpaceSize;j++)
279      {
280        mlem_sensitivityProj[j] += sensitivityProj[i*voxelSpaceSize
       + j];
281      }
282    }
283
284  invertSensitivity(mlem_sensitivityProj, voxelSpaceSize, 1);
285  float *mlem_sensitivityProj_d;
286  CUDA_CALL(cudaMalloc((void**)&mlem_sensitivityProj_d,sizeof(float)
       *voxelSpaceSize));
287  CUDA_CALL(cudaMemcpy(mlem_sensitivityProj_d,mlem_sensitivityProj,
       sizeof(float)*voxelSpaceSize,cudaMemcpyHostToDevice));
288
289  float *currentPositionEstimate_h;
290  currentPositionEstimate_h=(float *)malloc(voxelSpaceSize*sizeof(
       float));
291  float *currentPositionEstimate_d;
292  CUDA_CALL(cudaMalloc((void**)&currentPositionEstimate_d,sizeof(
       float)*voxelSpaceSize));
293  /*Initialize to zero.*/
294  resetToValue(currentPositionEstimate_h,voxelSpaceSize,0.01f);
295
296  /*Copy across to GPU.*/
297  CUDA_CALL(cudaMemcpy(currentPositionEstimate_d,
       currentPositionEstimate_h,sizeof(float)*voxelSpaceSize,
       cudaMemcpyHostToDevice));
298
299
300  stopper = 400;
301  int voxelCounter = 0;
302  for (int x=0;x<DIMX;x++)
303  {
304    for (int y=0;y<DIMY;y++)
305    {
306      for (int z=0;z<DIMZ;z++)
307      {
308        if (z>=DIMZ-7)
309        {
310          currentPositionEstimate_h[voxelCounter]=0.0f;
311        }
312        voxelCounter++;
313      }
314    }
315  }
```

```
316   int estimateOffset = 0;
317   CUDA_CALL(cudaMemcpy(currentPositionEstimate_d,
      currentPositionEstimate_h,sizeof(float)*voxelSpaceSize,
      cudaMemcpyHostToDevice));
318   for (int iterations=estimateOffset;iterations<=stopper;iterations
      ++)
319   {
320     cout << "Iteration: " << iterations << endl;
321
322     start = clock();
323
324       /*Start MLEM.*/
325
326 /*Implementation 1. Basic implementation of MLEM concepts. Mainly
      used for learning and debugging.*/
327     mlem_recon(mlem_sensitivityProj,mlem_detectorSum_d,
      currentPositionEstimate_d,numFloats, numberOfImages,g,
      numberOfCoefficients_h,gaussianCoefficients_h,gaussPosition_h,
      images_h,images_d, imageSize, numberOfVoxelsToProcess,
      voxelSpaceSize);
328
329 /*Implementation 2. Uses optimized reduction code and is more memory
      efficient.*/
330 mlem_recon2(H_d,numberOfCoefficients_d,gaussPosition_d,proj_d,mask,
      mlem_sensitivityProj,mlem_detectorSum_d,mlem_detectorSum_h,
      currentPositionEstimate_d,numFloats, numberOfImages,g_d,
      numberOfCoefficients_h,gaussianCoefficients_h,gaussPosition_h,
      images_h,images_d, imageSize, numberOfVoxelsToProcess,
      voxelSpaceSize);
331
332       /*Update estimate.*/
333     mlem_updateEstimateCUDA<<<(int)ceil((float)voxelSpaceSize/512.0f
      ),512 >>>(currentPositionEstimate_d,mlem_sensitivityProj_d,
      mlem_detectorSum_d, voxelSpaceSize);
334
335 end = clock();
336     cout << "1 Iteration takes: " << (double)(end - start)/
      CLOCKS_PER_SEC << endl;
337
338   /*Output reconstruction.*/
339     if ((iterations == 0) || (iterations == 1) || (iterations == 2)
      ||(iterations == 3) || (iterations % 5 == 0))
340     {
341           CUDA_CALL(cudaMemcpy(currentPositionEstimate_h,
      currentPositionEstimate_d,sizeof(float)*voxelSpaceSize,
      cudaMemcpyDeviceToHost));
342       cout << "***Writing. ***"<< endl;
343       string newFile;
344       newFile.append(rootOut);
345
346
347       newFile.append("reconstruction_");
348
349       newFile.append(IntToStr(iterations));
350       newFile.append(".dat");
351       cout << newFile << endl;
352       fileOut = fopen(newFile.c_str(),"wb");
```

```
353         fwrite(currentPositionEstimate_h,sizeof(float),voxelSpaceSize,
      fileOut);
354         fclose(fileOut);
355       }
356
357     }
358
359     free(numberOfCoefficients_h);
360     free(gaussianCoefficients_h);
361     free(detectorSum_h);
362
363     if (ALGORITHM == 0)/*mlem*/
364     {
365       free(mlem_detectorSum_h);
366       cudaFree(mlem_detectorSum_d);
367       free(mlem_sensitivityProj);
368     }
369     cudaFree(images_d);
370     cudaFree(detectorSum_d);
371     free(gaussPosition_h);
372     free(images_h);
373     free(g_tmp);
374     free(H);
375     free(g);
376
377     return 0;
378 }
```

A.7 POSEM

The POSEM implementation was very similar to Implementation 2 of MLEM but not all pixels were processed in each iteration. The code for the POSEM masks is included here but the main function is not because of its similarity to MLEM above.

A.7.1 Functions

```
1  /*Get the POSEM image mask.*/
2  void createPosemMask(int *mask,int subsetNumber)
3  {
4    int subsetSize = 4;
5    int *miniMask;
6    miniMask = (int *)malloc(sizeof(int)*subsetSize*subsetSize);
7
8    /*Below is the order suggested in the POSEM paper. */
9    miniMask[0] = 9;
10   miniMask[1] = 13;
11   miniMask[2] = 1;
12   miniMask[3] = 5;
13   miniMask[4] = 0;
```

```
14    miniMask[5] = 4;
15    miniMask[6] = 8;
16    miniMask[7] = 12;
17    miniMask[8] = 6;
18    miniMask[9] = 10;
19    miniMask[10] = 14;
20    miniMask[11] = 2;
21    miniMask[12] = 15;
22    miniMask[13] = 3;
23    miniMask[14] = 7;
24    miniMask[15] = 11;
25    int numPixX = IMAGE_DIMX/subsetSize;
26    int numPixY = IMAGE_DIMY/subsetSize;
27
28    for (int x=0;x<numPixX;x++)
29    {
30      for (int y=0;y<numPixY;y++)
31      {
32        for (int i=0;i<subsetSize;i++)
33        {
34          for (int j=0;j<subsetSize;j++)
35          {
36            int index = subsetSize*IMAGE_DIMX*y + x*subsetSize + j*
     IMAGE_DIMX + i;
37            if (miniMask[j*subsetSize + i] == subsetNumber)
38            {
39              mask[index] = 1;
40            }
41          }
42        }
43      }
44    }
45  }
46
47  /*Number of pixels in each POSEM subset.*/
48  void getNumberOfPixelsInEachSubset(int *mask,int *numberOfPixels,int
       imageSize)
49  {
50    int count=0;
51    for (int i=0;i<16;i++)
52    {
53      for (int idx=0;idx<imageSize;idx++)
54      {
55        if (mask[i*imageSize + idx]==1)
56          count++;
57      }
58      numberOfPixels[i] = count;
59      count=0;
60    }
61  }
62
63  /*Indices of the pixels.*/
64  void getPixelIndices(int *pixelIndices,int *mask,int imageSize)
65  {
66    int currentIndex = 0;
67    int subsetCount = 0;
68    for (int i=0;i<16;i++)
```

```
69   {
70     int offset = i*320*240/16;
71     subsetCount = 0;
72     currentIndex = 0;
73     for (int m=0;m<imageSize;m++)
74     {
75       if (mask[i*imageSize + m] == 1)
76       {
77         pixelIndices[offset + subsetCount] = currentIndex;
78         subsetCount++;
79       }
80       currentIndex++;
81     }
82   }
83 }
```

A.8 Fourier Crosstalk Matrix

A.8.1 Functions

```
1  /*Fourier Crosstalk Matrix.*/
2
3  __global__ void fl2_setToZeroCUDA(float2 *image,int N)
4  {
5    int idx = blockIdx.x*blockDim.x+threadIdx.x;
6
7    if (idx < N)
8    {
9      image[idx].x = 0.0f;
10     image[idx].y = 0.0f;
11   }
12 }
13
14 void fl2_resetToZero(float2 *H,int num)
15 {
16   for (int i=0;i<num;i++)
17   {
18     H[i].x = 0.0f;
19     H[i].y = 0.0f;
20   }
21 }
22
23 int countActivePixels(int *mask)
24 {
25   int cnt=0;
26   for (int i=0;i<IMAGE_DIMX*IMAGE_DIMY;i++)
27   {
28     if (mask[i]==1)
29       cnt++;
30   }
31   return cnt;
32 }
```

```
33
34 void convertIndexToXY(int *mask,int2 *coord)
35 {
36   int cnt=0;
37   int index=0;
38   for (int y=0;y<IMAGE_DIMY;y++)
39   {
40     for (int x=0;x<IMAGE_DIMX;x++)
41     {
42       if (mask[index]==1)
43       {
44         coord[cnt].x = x;
45         coord[cnt].y = y;
46         cnt++;
47       }
48       index++;
49     }
50   }
51 }
52
53 __global__ void sumAllFFT(float *total,float2 *image,int N)
54 {
55   int idx = blockIdx.x*blockDim.x+threadIdx.x;
56
57   if (idx < N)
58   {
59     total[idx] += image[idx].x*image[idx].x + image[idx].y*image[idx
     ].y;
60
61   }
62 }
63
64 __device__ float gaussKernel(float *sharedGaussianCoefficient,float
     xCent,float yCent)
65 {
66   float a = ((xCent - (sharedGaussianCoefficient[1]-1.0f))*(xCent -
     (sharedGaussianCoefficient[1]-1.0f))) / ((
     sharedGaussianCoefficient[3])*(sharedGaussianCoefficient[3]));
67   float b = -(2.0f*sharedGaussianCoefficient[5])*(xCent - (
     sharedGaussianCoefficient[1]-1.0f))*(yCent - (
     sharedGaussianCoefficient[2]-1.0f)) / (sharedGaussianCoefficient
     [3]*sharedGaussianCoefficient[4]);
68   float c = ((yCent - (sharedGaussianCoefficient[2]-1.0f))*(yCent -
     (sharedGaussianCoefficient[2]-1.0f))) / (
     sharedGaussianCoefficient[4]*sharedGaussianCoefficient[4]);
69   float z = a + b + c;
70   float tempVal = sharedGaussianCoefficient[0]*(__expf(-z / (2.0f
     *(1.0f-(sharedGaussianCoefficient[5]*sharedGaussianCoefficient
     [5])))));
71
72     return tempVal;
73
74 }
75
```

```
76 __global__ void createPSFFast(int *numberOfCoefficients_d,float *
     gaussianCoefficients_d,int *gaussPosition_d,float2 *array_d,int2
     *coord,int indexAdj,int startVoxel,int numberOfPixels,int
     gaussPositionOffset,int N)
77 {
78   int idx = blockIdx.x*blockDim.x+threadIdx.x;
79   extern __shared__ float2 sharedCoord[];
80
81   if (threadIdx.x < numberOfPixels)
82   {
83       sharedCoord[threadIdx.x].x = (float)coord[threadIdx.x].x;
84       sharedCoord[threadIdx.x].y = (float)coord[threadIdx.x].y;
85   }
86   __syncthreads();
87
88   if (idx < 61)
89   {
90     float x[6];
91     int currentIndex = startVoxel + idx;
92     int gaussIndex = gaussPosition_d[currentIndex] -
     gaussPositionOffset;
93     int arraySize = 256*256*256;
94     for (int i=0;i<numberOfCoefficients_d[currentIndex];i++)
95     {
96       for (int counter=0;counter<6;counter++)
97       {
98         x[counter] = gaussianCoefficients_d[gaussIndex + i*6 +
     counter];
99       }
100      for (int j=0;j<numberOfPixels;j++)
101      {
102        array_d[indexAdj + idx + j*arraySize].x += gaussKernel(x,
     sharedCoord[j].x,sharedCoord[j].y);
103      }
104    }
105  }
106 }
107
108
109 void getArrayForFFT(int *numberOfCoefficients_d,float *
     gaussianCoefficients_d,int *gaussPosition_d,float2 *array_d,int2
     *coord,int pixelStart,int numberOfPixels,int gaussPositionOffset)
110 {
111   int currentVoxel=0;
112   int numberOfVoxelsToProcessTemp = 64;
113   int block_size = 32;
114   int n_blocks = (int)ceil(float(numberOfVoxelsToProcessTemp)/(float
     (block_size)));
115
116   for (int k=0;k<256;k++)
117   {
118     for (int j=0;j<256;j++)
119     {
120       if (k<61 && j<61)
```

```
121         {       createPSFFast<<<n_blocks,block_size,numberOfPixels*
    sizeof(float2)>>>(numberOfCoefficients_d,gaussianCoefficients_d,
    gaussPosition_d,array_d,&coord[pixelStart],k*256*256 + j*256,
    currentVoxel, numberOfPixels, gaussPositionOffset,
    numberOfVoxelsToProcessTemp);
122           currentVoxel+=61;
123         }
124       }
125     }
126 }
```

A.8.2 Crosstalk Main

```
1  int main(void)
2  {
3
4    size_t avail;
5    size_t total;
6    cudaMemGetInfo(&avail,&total);
7    std::cout << "Total: "<< total <<"\nAvailable: "<< avail << std::
      endl;
8
9
10   /*Build the Mask and get the Active Pixels */
11   int *mask;
12   mask = (int *)malloc(sizeof(int)*IMAGE_DIMX*IMAGE_DIMY);
13   i_resetToValue(mask,IMAGE_DIMX*IMAGE_DIMY,1);
14   i_applyCircularMask(mask);
15   int activePixels = countActivePixels(mask);
16   std::cout << "There are " << activePixels << " active pixels. " <<
       std::endl;
17
18          int2 *coord;
19   coord = (int2 *)malloc(sizeof(int2)*activePixels);
20   convertIndexToXY(mask,coord);
21   std::cout << "Pixel 1: " << coord[0].x << ", " << coord[0].y <<
      std::endl;
22   std::cout << "Pixel End: " << coord[activePixels-1].x << ", " <<
      coord[activePixels-1].y << std::endl;
23   int2 *coord_d;
24   cudaMalloc((void**)&coord_d, sizeof(int2)*activePixels);
25   cudaMemcpy(coord_d,coord,sizeof(int2)*activePixels,
      cudaMemcpyHostToDevice);
26   FILE *pixOut;
27   pixOut = fopen("E:\\TEMP\\activePixels.dat","wb");
28   fwrite(coord,sizeof(int2),activePixels,pixOut);
29   fclose(pixOut);
30
31
32   //Load all the H matrices.
33   int numberOfImages = 6;
34
35   int voxelSpaceSize = DIMX*DIMY*DIMZ;
36   FILE *fn[6];
```

```
37  /*fn[0] = fopen("E:\\PSF\\H\\
     gaussParams_00mm_082012_v3_PE_raw_cubic.dat","rb");
38  fn[1] = fopen("E:\\PSF\\H\\gaussParams_08mm_082012_v3_PE_raw_cubic
     .dat","rb");
39  fn[2] = fopen("E:\\PSF\\H\\gaussParams_16mm_082012_v3_PE_raw_cubic
     .dat","rb");
40  fn[3] = fopen("E:\\PSF\\H\\gaussParams_24mm_082012_v3_PE_raw_cubic
     .dat","rb");
41  fn[4] = fopen("E:\\PSF\\H\\gaussParams_27mm_082012_v3_PE_raw_cubic
     .dat","rb");
42  fn[5] = fopen("E:\\PSF\\H\\gaussParams_37mm_082012_v3_PE_raw_cubic
     .dat","rb");*/
43
44  fn[0] = fopen("E:\\PSF\\H\\
     gaussParams_00mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
45  fn[1] = fopen("E:\\PSF\\H\\
     gaussParams_08mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
46  fn[2] = fopen("E:\\PSF\\H\\
     gaussParams_16mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
47  fn[3] = fopen("E:\\PSF\\H\\
     gaussParams_24mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
48  fn[4] = fopen("E:\\PSF\\H\\
     gaussParams_27mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
49  fn[5] = fopen("E:\\PSF\\H\\
     gaussParams_37mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
50
51        printf("File...open\n");
52  int h1[3];
53  printf("File...read\n");
54  for (int fileCnt=0;fileCnt<numberOfImages;fileCnt++)
55  {
56    fread(&h1[0],sizeof(int),3,fn[fileCnt]);
57    printf("%d,%d,%d\n",h1[0],h1[1],h1[2]);
58  }
59
60  /*Allocate memory to store all of the coefficients.*/
61  int *numFloats;
62  numFloats = (int *)malloc(sizeof(int)*numberOfImages);
63  for (int fileCnt=0;fileCnt<numberOfImages;fileCnt++)
64  {
65    numFloats[fileCnt] = getSize(fn[fileCnt]);
66    printf("File Number: %d, Count: %d\n",fileCnt,numFloats[fileCnt
     ]);
67    rewind(fn[fileCnt]);
68    fread(&h1[0],sizeof(int),3,fn[fileCnt]);
69    printf("%d,%d,%d\n",h1[0],h1[1],h1[2]);
70  }
71
72  int totalFloats = 0;
73  for (int fileCnt=0;fileCnt<numberOfImages;fileCnt++)
74  {
75    totalFloats += numFloats[fileCnt];
76  }
77  std::cout << "Total Floats: " << totalFloats << std::endl;
78
79  printf("Allocate Memory.\n");
80  int *numberOfCoefficients_h;
```

```
81  numberOfCoefficients_h = (int *)malloc(sizeof(int)*voxelSpaceSize*
     numberOfImages);
82  float *gaussianCoefficients_h;
83  gaussianCoefficients_h = (float *)malloc(sizeof(float)*totalFloats
     );//*numberOfImages);
84  int *gaussPosition_h;
85  gaussPosition_h = (int *)malloc(sizeof(int)*voxelSpaceSize*
     numberOfImages);
86  cudaMemGetInfo(&avail,&total);
87  std::cout << "Total: "<< total <<"\nAvailable: "<< avail << std::
     endl;
88
89        for (int k=0;k<numberOfImages;k++)
90  {
91    /*Read in the coefficients.*/
92    int filePosTemp=0;
93    for (int j=0;j<k;j++)
94    {
95      filePosTemp += numFloats[j];
96    }
97    std::cout << "FILEPOS: " << filePosTemp << std::endl;
98    loadCoefficients(fn[k],&numberOfCoefficients_h[k*voxelSpaceSize
     ],&gaussianCoefficients_h[filePosTemp]);
99    printf("Coefficients Loaded.\n");
100   getGaussPosition(&gaussPosition_h[k*voxelSpaceSize],&
     numberOfCoefficients_h[k*voxelSpaceSize],filePosTemp);
101   printf("Gauss Positions Loaded.\n");
102   rewind(fn[k]);
103   fread(&h1[0],sizeof(int),3,fn[k]);
104 }
105 for (int i=0;i<numberOfImages;i++)
106   fclose(fn[i]);
107 clock_t start,end;
108
109 for (int j=0;j<totalFloats;j=j+6)
110 {
111   if (IMAGE_DIMX == 320)
112   {
113     gaussianCoefficients_h[j+1] = gaussianCoefficients_h[j+1]/2.0f
     ;
114     gaussianCoefficients_h[j+2] = gaussianCoefficients_h[j+2]/2.0f
     ;
115     gaussianCoefficients_h[j+3] = gaussianCoefficients_h[j+3]/2.0f
     ;
116     gaussianCoefficients_h[j+4] = gaussianCoefficients_h[j+4]/2.0f
     ;
117   }
118 }
119
120 int counter = 0;
121 int singleCounter=0;
122 for (int j=0;j<numberOfImages;j++)
123 {
124   int voxelCounter = 0;
125
126   for (int x=0;x<DIMX;x++)
127   {
```

```
128       for (int y=0;y<DIMY;y++)
129       {
130         for (int z=0;z<DIMZ;z++)
131         {
132           if (z>=DIMZ-4)
133           {
134             counter = 0;
135             for (int k=0;k<numberOfCoefficients_h[j*voxelSpaceSize+
    voxelCounter];k++)
136             {
137               gaussianCoefficients_h[gaussPosition_h[j*
    voxelSpaceSize+voxelCounter]+ counter] = 0.0f;
138               counter = counter + 6;
139             }
140           }
141           voxelCounter++;
142           singleCounter++;
143         }
144       }
145     }
146   }
147
148
149   int NX=256,NY=256,NZ=256;
150   /*Declare the array to hold the results of the mth pixel for every
        object position.*/
151   int numberOfPixels=10;
152
153   float2 *array1_h;
154   array1_h = (float2 *)malloc(sizeof(float2)*NX*NY*NZ*numberOfPixels
    );
155
156         /*Declare the variable that will hold the raw data and the
    FFT. This will get overwritten in every loop.*/
157   float2 *array1_d;
158   cudaMalloc((void**)&array1_d, sizeof(float2)*NX*NY*NZ*
    numberOfPixels);
159
160   /*Create the array that will store the sum off all these FFTs. It
    should be NX*NY*NZ. In every iteration of 50 pixels (or whatever)
        the voxels of all the FFT array will be summed with the same
    voxels in the other arrays.*/
161   float *fftSum_h;
162   fftSum_h = (float *)malloc(sizeof(float)*NX*NY*NZ);
163   fl_resetToZero(fftSum_h,NX*NY*NZ);
164
165   float *fftSum_d;
166   cudaMalloc((void**)&fftSum_d, sizeof(float)*NX*NY*NZ);
167   int block_size = 512;
168   int n_blocks = (int)ceil(float(NX*NY*NZ)/(float(block_size)));
169   fl_setToZeroCUDA<<<n_blocks, block_size>>>(fftSum_d,NX*NY*NZ);
170
171
172   /* Create a 3D FFT plan. */
173   cufftHandle plan;
174   cufftPlan3d(&plan, NX, NY, NZ, CUFFT_C2C);
175
```

```
176   int *numberOfCoefficients_d;
177   int *gaussPosition_d;
178   float *gaussianCoefficients_d;
179
180   CUDA_CALL(cudaMalloc((void**)&numberOfCoefficients_d,sizeof(int)*
       voxelSpaceSize));
181   CUDA_CALL(cudaMalloc((void**)&gaussPosition_d,sizeof(int)*
       voxelSpaceSize));
182   FILE *fout[6];
183   fout[0] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_1im.dat","wb");
184   fout[1] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_2im.dat","wb");
185   fout[2] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_3im.dat","wb");
186   fout[3] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_4im.dat","wb");
187   fout[4] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_5im.dat","wb");
188   fout[5] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_6im.dat","wb");
189   std::cout << "Files are open and crosstalk is now being calculated
       ." << std::endl;
190   for (int imageNumber=0;imageNumber<6;imageNumber++)
191   {
192     cudaMemGetInfo(&avail,&total);
193     std::cout << "Total: "<< total <<"\nAvailable: "<< avail  << ",
       Required: " << numFloats[imageNumber]*4 << std::endl;
194   numberOfPixels = 10;
195   int gaussPositionOffset = gaussPosition_h[imageNumber*
       voxelSpaceSize];
196   CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d,sizeof(float)
       *numFloats[imageNumber]));
197   CUDA_CALL(cudaMemcpy(gaussianCoefficients_d,&
       gaussianCoefficients_h[gaussPosition_h[imageNumber*voxelSpaceSize
       ]],sizeof(float)*numFloats[imageNumber],cudaMemcpyHostToDevice));
198   CUDA_CALL(cudaMemcpy(numberOfCoefficients_d,&
       numberOfCoefficients_h[imageNumber*voxelSpaceSize],sizeof(int)*
       voxelSpaceSize,cudaMemcpyHostToDevice));
199   CUDA_CALL(cudaMemcpy(gaussPosition_d,&gaussPosition_h[imageNumber*
       voxelSpaceSize],sizeof(int)*voxelSpaceSize,cudaMemcpyHostToDevice
       ));
200
201   syncDeviceAndErrorCheck();
202   int currentPixelNumber=0;
203
204   //How many iterations are required for the whole pixel list?
205   start = clock();
206   int loopsRequired = (int)ceil((float)activePixels / (float)
       numberOfPixels);
207
208   block_size = 512;
209   n_blocks = (int)ceil(float(NX*NY*NZ)/(float(block_size)));
210
211   for (int repeat=0;repeat<loopsRequired;repeat++)
212   {
213     if (currentPixelNumber + numberOfPixels > activePixels)
214       numberOfPixels = activePixels - currentPixelNumber;
215
216   /*Set the FFT arrays to zero.*/
217
218   for (int i=0;i<numberOfPixels;i++)
```

```
219       fl2_setToZeroCUDA<<<n_blocks, block_size>>>(&array1_d[i*NX*NY*NZ
          ],NX*NY*NZ);
220
221     getArrayForFFT(numberOfCoefficients_d, gaussianCoefficients_d,
          gaussPosition_d, array1_d, coord_d, currentPixelNumber,
          numberOfPixels, gaussPositionOffset);
222     currentPixelNumber+=numberOfPixels;
223
224     /* Transform the first signal in place. */
225     for (int i=0;i<numberOfPixels;i++)
226       cufftExecC2C(plan, (cufftComplex *) &array1_d[i*NX*NY*NZ], (
          cufftComplex *) &array1_d[i*NX*NY*NZ], CUFFT_FORWARD);
227
228     /*Need to sum the voxels in the FFTs.*/
229     for (int i=0;i<numberOfPixels;i++)
230       sumAllFFT<<<n_blocks, block_size>>>(fftSum_d,&array1_d[i*NX*NY*NZ
          ],NX*NY*NZ);
231
232     }
233     cudaFree(gaussianCoefficients_d);
234
235     CUDA_CALL(cudaMemcpy(fftSum_h, fftSum_d, sizeof(float)*NX*NY*NZ,
          cudaMemcpyDeviceToHost));
236     fwrite(fftSum_h, sizeof(float),NX*NY*NZ, fout[imageNumber]);
237     fclose(fout[imageNumber]);
238     end = clock();
239     std::cout << "Image Number: " <<  imageNumber << ", Iteration
          takes: " << (double)(end - start)/CLOCKS_PER_SEC << std::endl;
240
241     }
242
243     /* Destroy the cuFFT plan. */
244     cudaFree(fftSum_d);
245     cudaFree(coord_d);
246     cufftDestroy(plan);
247     cudaFree(numberOfCoefficients_d);
248     cudaFree(gaussPosition_d);
249
250     free(array1_h);
251     free(fftSum_h);
252     free(gaussianCoefficients_h);
253     free(gaussPosition_h);
254     free(numberOfCoefficients_h);
255     free(coord);
256     free(numFloats);
257     cudaFree(array1_d);
258
259     return 0;
260 }
```

APPENDIX B

MATLAB Code

B.1 Main MATLAB Script

```
%%Master Pinhole Fitting, Interpolation & Writing

%%Compile the Centroids
fileIn_root = 'E:\PSF\New Intensifier\6-22-12-08mm\';%%The folder which contains the numbered folders with all the PSF information.
name = 'centroids_08mm.dat';
fileOut = [fileIn_root name];
numberRange = [0 2196];
imSize = [320 240];

%%This just reads in the centroid files and writes them into a single file.
im_centroid = compileCentroidsSimple(fileIn_root,fileOut,numberRange, imSize);

%%Read in the centroid files and fit Gaussian distributions.
fileIn = fileOut;%%The centroid file created above.
nameGauss = 'gaussianParams_08mm_PE_v1.dat';
fileOut = [fileIn_root nameGauss];
numVoxels = 13*13*13;
dim = [13 13 13];

padsize=20;

%%Gaussian fitting to the projections
gaussianFitting(fileIn, fileOut,numVoxels,imSize,padsize,10,6);

fileIn = fileOut;

%%Need to normalize the gaussian fits due to the varying acquisition times used in the PSF scan.
fileOut = [fileIn_root 'gaussianParams_normalized_08mm_PE_v1.dat'];
normalizeGaussian(fileIn,fileOut,dim);
```

```
30 fileIn = fileOut;
31
32 %%This function tracks and sorts the pinholes. It also converts them
       into coefficients that can be used with [640 480] images.
33 %%Do this for each of the system configurations. Name them p_0 and
     p_8 or similar.
34 p = pinholeTrackingv6(fileIn,dim);
35
36
37 %%Determine pinhole matches between PSF datasets 00mm and 08mm.%%
       These matches will be use to extrapolate the positions of the
       pinholes in higher magnification datasets. These positions are
       then compared to experimentally measured PSFs to sort them.
38
39 pinhole = manualPinholeMatching(p0,p8,[13 13 13]);
40
41
42 %%Extrapolate the projection positions in high-magnification data.
43 filename_Coeff = 'E:\PSF\New Intensifier\6-22-12-24mm\
     gaussianParams_normalized_24mm.dat';
44 [p24 distances1] = extrapolateHighMagPinholeCoordinates(
     filename_Coeff,p0,p8,320,24,pinhole,[13 13 13],20);
45
46 filename_Coeff = 'E:\PSF\New Intensifier\6-22-12-27mm\
     gaussianParams_normalized_27mm.dat';
47 [p27 distances1] = extrapolateHighMagPinholeCoordinates(
     filename_Coeff,p0,p8,320,27,pinhole,[13 13 13],25);
48
49 filename_Coeff = 'E:\PSF\New Intensifier\6-22-12-37mm\
     gaussianParams_normalized_37mm.dat';
50 [p37 distances2] = extrapolateHighMagPinholeCoordinates(
     filename_Coeff,p0,p8,320,37,pinhole,[13 13 13],35);
51
52
53 %%RUN PSF_weakPinholeDetection here.
54 %%This is a separate .m file, not a function.
55 %%This function will detect weak projections at the edge of the
       field of view.
56
57
58 %%The outputs of the PSF_weakPinholeDetection function are loaded
       now.
59 load 8-20-12-p0mm_PE_v3_complete.mat
60 pS{1} = p;
61 load 8-20-12-p8mm_PE_v3_complete.mat
62 pS{2} = p;
63 load 8-20-12-p16mm_PE_v3_complete.mat
64 pS{3} = p;
65 load 8-20-12-p24mm_PE_v3_complete.mat
66 pS{4} = p;
67 load 8-20-12-p27mm_PE_v3_complete.mat
68 pS{5} = p;
69 load 8-20-12-p37mm_PE_v3_complete.mat
70 pS{6} = p;
71
72 endings = ['00';'08';'16';'24';'27';'37'];
73
```

```
74 for n=1:6
75
76 coeffInterp = interpolateinterpnWithMask_x3_simple(pS{n},0,1,[-15
       15],2.5,0.5,[13 13 13],'cubic');
77
78 fileout = ['E:\PSF\H\gaussParams_' endings(n,:) '
       mm_082012_v3_PE_raw_cubic_Halfmm.dat'];
79 writeCoeffToFile(coeffInterp,fileout,size(coeffInterp));
80 end
81
82 %%These files can be used to reconstruct an object using the MLEM or
       POSEM algorithms.
```

B.2 MATLAB Functions

B.2.1 Gaussian Fitting

```
1  function gaussianFitting(fileIn, fileOut,numVoxels,imSize,padsize,
       thresholdPeak,thresholdCluster)
2  %%thresholdCluster is the value that 3 pixels inside a cluster must
       have to
3  %%be classified as potential signal and not noise. This should be
       adjusted depending on the data.
4
5  fn = fopen(fileIn,'r');
6  header = fread(fn,3,'int');
7  fout = fopen(fileOut,'w');
8  fwrite(fout,header,'int');
9
10 for n=1:numVoxels
11     image1 = fread(fn,imSize(1)*imSize(2),'float32');
12
13     Gauss_Params = [];
14     if max(image1(:)) > thresholdPeak
15         imTmp = reshape(image1,[imSize(1) imSize(2)]);
16         [Gauss_Params res1]= Gaussian2D_master_6params_Simple_v3(
       imTmp,padsize,thresholdCluster);
17
18     end
19
20     fwrite(fout,size(Gauss_Params,2),'int');
21     if size(Gauss_Params,2) > 0
22         fwrite(fout,Gauss_Params,'float32');
23     end
24 end
25
26 fclose(fn);
27 fclose(fout);
28
29
30 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
31 %% Gaussian2D_master_6params_Simple_v3.m - To fit a 2-D gaussian.
```

```
32 %%
33 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
34 function [Gauss_Params residuals] =
       Gaussian2D_master_6params_Simple_v3(imTmp, padsize, threshold)
35 %padsize must be an even number!
36 residuals=[];
37 fval = [];
38
39 offsetNeg = padsize/2;
40 offsetPos = padsize/2 - 1;
41 imTmp = imTmp';
42
43 im1med = imTmp;
44 im1med(im1med<threshold)=0;
45 imTmp1 = padarray(imTmp,[padsize padsize]);
46 imTmp1med = padarray(im1med,[padsize padsize]);
47
48 clear im1med imTmp
49 cc = bwconncomp(imTmp1med);
50 %%Found a bug in this code when a cluster has values below the
       threshold in
51 %%the middle of the blob so it splits it in two. This happened for a
        single cluster out of thousands.
52 %%Need to get the distances between all the blobs found in the
       connected
53 %%components and concatenate the pixelIdxList from both clusters.
54
55 cc = removeDuplicateClusters(cc,15,size(imTmp1med));
56 numPixels = cellfun(@(x)size(x,1),cc.PixelIdxList);
57 ind = find(numPixels >= 3);
58
59 counter = 1;
60
61 for p = 1:size(ind,2)
62     currentPixel = cc.PixelIdxList{ind(p)};
63     [R C] = ind2sub(size(imTmp1med),currentPixel(1));
64     pixel = [R(1) C(1)];
65
66
67     %%Get the list, select the first pixel.
68     %%Take a 20x20 area around it.
69     %%Get the center of it and coefficients.
70     %%Set the 20x20 grid to 0 in the main image.
71
72     g = imTmp1(pixel(1)-offsetNeg:pixel(1)+offsetPos,pixel(2)-
       offsetNeg:pixel(2)+offsetPos);
73     gMed = imTmp1med(pixel(1)-offsetNeg:pixel(1)+offsetPos,pixel(2)-
       offsetNeg:pixel(2)+offsetPos);
74
75     %%Just in case g is only non-zero in a section of Nx1
76     %%pixels. (This would break 'moments.m')
77
78     if sum(sum(g,1)>0) < 2 || sum((sum(g,2)>0)) < 2
79         g(round(size(g,1)/2),round(size(g,2)/2)) = g(round(size(g,1)
       /2),round(size(g,2)/2)) +.1;
80         g(round(size(g,1)/2)-1,round(size(g,2)/2)-1) = g(round(size(
       g,1)/2)-1,round(size(g,2)/2)-1) + .1;
```

```
81      end
82
83      [mx, my, var_x, var_y, covar_xy, ~, ~, ~, ~, ~] = moments(g);
84
85      mx = mx+pixel(2)-(offsetNeg + 1);
86      my = my+pixel(1)-(offsetNeg + 1);
87
88      dim_x=7;
89      dim_y=7;
90
91      mxT = floor(mx + 0.5);
92      myT = floor(my + 0.5);
93      g = imTmp1(myT-dim_y:myT+dim_y,mxT-dim_x:mxT+dim_x);
94      gTemp = g;
95
96      gTemp(g<=2) = 0;
97      gTemp(gTemp<=(max(gTemp(:))*0.1)) = 0;
98      [r c] = find(gTemp == max(gTemp(:)));
99      mask = 1*bwselect(gTemp,c(1),r(1),8);
100     mask = increaseBoundary(mask,sqrt(2));
101     g = g.*mask;
102
103     if sum(sum(g,1)>0) < 2 || sum((sum(g,2)>0)) < 2
104         g(r+1,c) = g(r+1,c) + 0.01;
105         g(r,c+1) = g(r,c+1) + 0.01;
106         g(r-1,c) = g(r-1,c) + 0.01;
107         g(r,c-1) = g(r,c-1) + 0.01;
108         g(r,c) = g(r,c) + 0.01;
109     end
110     clear mask c r
111     [mx, my, var_x, var_y, covar_xy, ~, ~, ~, ~, ~] = moments(g);
112
113     sx = sqrt(var_x);
114     sy = sqrt(var_y);
115     rho_xy = covar_xy/sx/sy;
116     max_pixel = max(g(:));
117
118
119     %%Upper and Lower Bounds with fmincon
120 offset_percentage = 0.5;
121     A_lb = 1e-10;
122     A_ub = 1000*max_pixel;
123     xm_lb = mx-abs(mx*offset_percentage);
124     xm_ub = mx+abs(mx*offset_percentage);
125     ym_lb = my-abs(my*offset_percentage);
126     ym_ub = my+abs(my*offset_percentage);
127
128     sx_lb = sx-abs(sx*offset_percentage);
129     sx_ub = sx+10*abs(sx*offset_percentage);
130     if sx_ub < 5
131         sx_ub  = 5;
132     end
133     sy_lb = sy-abs(sy*offset_percentage);
134     sy_ub = sy+10*abs(sy*offset_percentage);
135     if sy_ub < 5
136         sy_ub  = 5;
137     end
```

```
138
139     x = [max_pixel mx my sx sy rho_xy];
140
141
142     if sx_lb<=0 || sx_lb > 1
143         sx_lb=0.1;
144     end
145     if sy_lb<=0 || sy_lb > 1
146         sy_lb=0.1;
147     end
148
149         rho_xy_lb=-0.99;
150         rho_xy_ub=0.99;
151
152     if(sx == 0 || sy == 0 || max_pixel == 0)
153         continue;
154     end
155
156     sum_pre=sum(g(:));
157     dim_y_old = dim_y;
158     dim_x_old = dim_x;
159
160     dim_y = size(g,1);
161     dim_x = size(g,2);
162
163     g=g(:);
164
165     [Gauss_Params(1:6,counter), fval(counter), exitflag(counter)] =
    fmincon('GaussFit',x,[],[],[],[],[A_lb xm_lb ym_lb sx_lb sy_lb
    rho_xy_lb],[A_ub xm_ub ym_ub sx_ub sy_ub rho_xy_ub],[],optimset('
    Algorithm', 'sqp','TolX',1e-10,'TolFun',1e-10,'MaxFunEvals'
    ,100000,'MaxIter', 10000, 'Display', 'notify-detailed'),g,dim_y,
    dim_x);
166
167     Gauss_Params(2,counter) = Gauss_Params(2,counter) + mxT-(
    dim_x_old +1);
168     Gauss_Params(3,counter) = Gauss_Params(3,counter) + myT-(
    dim_y_old +1);
169
170
171     %%Have limits been reached?
172     if Gauss_Params(1,counter) == A_ub || Gauss_Params(4,counter) ==
     sx_lb || Gauss_Params(4,counter) == sx_ub || Gauss_Params(5,
    counter) == sy_lb || Gauss_Params(5,counter) == sy_ub ||
    Gauss_Params(6,counter) == rho_xy_lb || Gauss_Params(6,counter)
    == rho_xy_ub
173
174         disp('Limits have been detected');
175         Gauss_Params(:,counter)
176     end
177     if sum(isreal(Gauss_Params(:))) == 0
178         disp('ERROR: Complex Number Detected');
179     end
180
181
182   %%Renormalize the gaussian.
183   %%Analytical:
```

```
184   %%sum_post=Gauss_Params(1,counter)*2*pi*Gauss_Params(4,counter)*
       Gauss_Params(5,counter)*sqrt(1-Gauss_Params(6,counter)^2);
185   %%Numerical:
186   sum_post = sum(sum(generatePSFfromGaussianCoefficients([
       Gauss_Params(1,counter); ...
187           (Gauss_Params(2,counter) - padsize)  ;(Gauss_Params(3,
       counter) - padsize) ;...
188           Gauss_Params(4,counter) ;Gauss_Params(5,counter); ...
189           Gauss_Params(6,counter)]  ,2,2)));
190
191   Gauss_Params(1,counter) = Gauss_Params(1,counter)/sum_post*sum_pre
       ;
192
193       imTmp1med(myT-offsetNeg:myT+offsetPos,mxT-offsetNeg:mxT+
       offsetPos) = 0;
194
195   counter = counter + 1;
196 end
197
198 residuals = fval;
199
200 type = exist('Gauss_Params');
201 if type == 1
202     Gauss_Params(2,:) = Gauss_Params(2,:) - padsize ;
203     Gauss_Params(3,:) = Gauss_Params(3,:) - padsize ;
204
205     %Remove NaNs caused by insufficient data.
206     ind = isnan(Gauss_Params(2,:));
207     ind = find(ind == 1);
208     Gauss_Params(:,ind) = [];
209
210 else
211     Gauss_Params = [];
212 end
213
214
215 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
216 %% GaussFit.m
217 %%
218 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
219 function LS = GaussFit(x,g,dim_y, dim_x)
220
221 %A = x(1)
222 %mx = x(2);
223 %my = x(3);
224 %sigma_x = x(4);
225 %sigma_y = x(5);
226 %rho_xy = x(6)
227
228 [Y,X] = meshgrid(1:dim_y,1:dim_x);
229 X = X(:);
230 Y = Y(:);
231
232 %%2D Gaussian
233 a = ((X-x(2)).^2)./(x(4).^2);
234 b = -(2.*x(6).*(X-x(2)).*(Y-x(3)))./(x(4).*x(5));
235 c = ((Y-x(3)).^2)./(x(5).^2) ;
```

```
236 z = a + b + c;
237 gm_bar = x(1).*exp(-z./(2*(1-x(6).^2)));
238
239 gm_bar = reshape(gm_bar,dim_x,dim_y)';
240
241 gm_bar = gm_bar(:);
242
243
244 g_new = g;
245 gm_bar_new = gm_bar;
246
247
248 %%This blocks against very low values causing a large value in the
    following equations.
249 ind = find(gm_bar_new>=1/(2*pi));
250
251 if size(ind,1)>0
252     LS = sum( 0.5*log(2*pi*(gm_bar_new(ind(:)) )) + ( ((g_new(ind(:)
    ) -gm_bar_new(ind(:)))).^2)./((2*gm_bar_new(ind(:)))) ) );
253 else
254     LS = 10^6;
255 end
256
257 ind = find(gm_bar_new < 1/(2*pi));
258 if size(ind,1) > 0
259     LS = LS + sum((g_new(ind(:)) - gm_bar_new(ind(:))).^2);
260 end
261
262
263 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
264 %% Moments.m - Brian W. Miller
265 %%
266 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
267 function [mean_x, mean_y, var_x, var_y, covar_xy, skew_x, skew_y,
    kurt_x, kurt_y, eccentricity] = moments(clust)
268
269     %Local Centroid
270         area = sum(sum(clust));
271         [rows,cols] = size(clust);
272         x = ones(rows,1)*[1:cols];      % Matrix with each pixel set
    to its x coordinate
273         y = [1:rows]'*ones(1,cols);     %    "      "      "      "      "  "
        "  y     "
274         mean_x = sum(sum(clust.*x))/area;
275         mean_y = sum(sum(clust.*y))/area;
276
277     %2nd Central Moment - Variance
278         m20 = sum(sum(clust.*(x-mean_x).^2))/area;
279         m02 = sum(sum(clust.*(y-mean_y).^2))/area;
280         m11 = sum(sum(clust.*(y-mean_y).*(x-mean_x)))/area;
281         var_x = m20;
282         var_y = m02;
283         covar_xy = m11;
284
285     %3rd Central Moment
286         m30 = sum(sum(clust.*(x-mean_x).^3))/(area * m20^(3/2));
287         m03 = sum(sum(clust.*(y-mean_y).^3))/(area*m02^(3/2));
```

```
288         m21 = sum(sum(clust.*(x-mean_x).*(x-mean_x).*(y - mean_y)));
289         m12 = sum(sum(clust.*(x-mean_x).*(y-mean_y).*(y - mean_y)));
290         skew_x = m30;
291         skew_y = m03;
292
293       %4th Central Moment
294         m40 = sum(sum(clust.*(x-mean_x).^4))/(area * m20^2) - 3;
295         m04 = sum(sum(clust.*(y-mean_y).^4))/(area * m02^2) - 3;
296         m31 = sum(sum(clust.*(x-mean_x).^3.*(y - mean_y)));
297         m13 = sum(sum(clust.*(y-mean_y).^3.*(x - mean_x)));
298         kurt_x = m40;
299         kurt_y = m04;
300
301       %Eccentricity
302         eccentricity = ((m02-m02)^2 + 4*m11*m11);
303
304 %     ** Interpretation of spatial moments **
305 %
306 % * order 0  = TOTAL MASS [units: concentration, density, etc.]
307 % * order 1  = location of CENTRE OF MASS in x and y from 0,0 [units
      : L]
308 % * order 2  = VARIANCE around centroid in x and y [units: L^2]
309 % * order 3  = coeff. of skew_yNESS (symmetry) in x and y [units: n/
      a]
310 %                   --> =0  : SYMMETRIC distribution
311 %                   --> <0  : Distribution asymmetric to the LEFT
312 %                             (tail extends left of centre of mass)
313 %                   --> >0  : Distribution asymmetric to the RIGHT
314 %                             (tail extends right of centre of mass)
315 % * order 4  = KURTOSIS (flatness) in x and y [units: n/a]
316 %                   --> =0   : Gaussian (NORMAL) distribution
317 %                   --> <0   : Distribution FLATTER than normal
318 %                   --> >0   : Distribution MORE PEAKED than normal
319 %                   --> <-1.2: BclustODAL (or multclustodal)
      distribution
320 %
321 % ** Parameters derived from 2nd moments ** (from Awcock (1995)
322 %       "Applied clustage Processing")
323 %
324 %   * ELONGATION (ECCENTRICITY) = Ratio of longest to shortest
325 %     distance vectors rom the object's centroid to its boundaries
326 %   * ORIENTATION = For elongated objects, describes the
327 %     orientation (in degrees) of the "long" direction with
328 %     respect to horizontal (x axis)
329
330 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
331 %% Normalize Gaussian.m
332 %%
333 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
334 function normalizeGaussian(fileIn,fileOut,numVoxels)
335
336 fn = fopen(fileIn,'r');
337 header = fread(fn,3,'int');
338 fout = fopen(fileOut,'w');
339 fwrite(fout,header,'int');
340
341 for i=1:numVoxels(3)
```

```
342     for j=1:numVoxels(2)
343         for k=1:numVoxels(1)
344 
345             numberOfCoefficients = fread(fn,1,'int');
346             Gauss_Params = fread(fn,[6 numberOfCoefficients],'
    float32');
347             if size(Gauss_Params,2)>0
348             switch k
349                 case 1
350                     factor = 140;
351                 case 2
352                     factor = 130;
353                 case 3
354                     factor = 120;
355                 case 4
356                     factor = 110;
357                 case 5
358                     factor = 100;
359                 case 6
360                     factor = 90;
361                 case 7
362                     factor = 80;
363                 case 8
364                     factor = 70;
365                 case 9
366                     factor = 60;
367                 case 10
368                     factor = 50;
369                 case 11
370                     factor = 40;
371                 case 12
372                     factor = 30;
373                 case 13
374                     factor = 20;
375             end
376 
377             Gauss_Params(1,:) = Gauss_Params(1,:)*100/factor;
378             end
379             fwrite(fout,size(Gauss_Params,2),'int');
380             if size(Gauss_Params,2) > 0
381                 fwrite(fout,Gauss_Params,'float32');
382             end
383 
384         end
385     end
386 end
387 
388 fclose(fn);
389 fclose(fout);
```

B.2.2 Pinhole Tracking

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %% Pinhole Tracking _v6.m
```

```
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function p = pinholeTrackingv6(filename,dim)
coef = getCoefficients(filename,13);
%%Example:
%%coef = getCoefficients('E:\PSF\New Intensifier\6-22-12-00mm\
    gaussianParams_normalized_00mm_PE_v3.dat',13);

xSize=320;
if xSize == 320
    coef2 = coef;
    for i=1:dim(3)
        for j=1:dim(2)
            for k=1:dim(1)
                if isempty(coef2{k,j,i})~=1
                    coef2{k,j,i}(2,:) = 2*coef2{k,j,i}(2,:);
                    coef2{k,j,i}(3,:) = 2*coef2{k,j,i}(3,:);
                    coef2{k,j,i}(4,:) = 2*coef2{k,j,i}(4,:);
                    coef2{k,j,i}(5,:) = 2*coef2{k,j,i}(5,:);
                end
            end
        end
    end
end

clear p
p(dim(1),dim(2),dim(3),100) = struct('value',[],'posX',[],'posY',[],
    'sx',[],'sy',[],'tilt',[]);

%%Load in these as the base pinholes. This is at the back, center of
    the FOV.
i=floor((dim(3)/2) + 0.5);
j=floor((dim(2)/2) + 0.5);
k=1;
start = [k j i];
m1 = coef2{k,j,i};

for n=1:size(m1,2)
    p(k,j,i,n).value = m1(1,n);
    p(k,j,i,n).posX = m1(2,n);
    p(k,j,i,n).posY = m1(3,n);
    p(k,j,i,n).sx = m1(4,n);
    p(k,j,i,n).sy = m1(5,n);
    p(k,j,i,n).tilt = m1(6,n);
end

done = zeros(dim(1),dim(2),dim(3));
done(k,j,i) = 1;

currentMaxPinholeNumber = 0;

projectedX = zeros(dim(1),dim(2),dim(3),100);
projectedY = zeros(dim(1),dim(2),dim(3),100);

for voxels = 1:(dim(1)*dim(2)*dim(3))-1

    %%Predict where everything will be based on everything we know!
```

```
57
58 %%  This step uses 'inpaintn.m', although any extrapolation code
       could work here.
59 %%  Written by: Damien Garcia.
60 %%  Source: Mathworks File Exchange
61 %%  INPAINTN Inpaint over missing data in N-D array
62 %%   Y = INPAINTN(X) replaces the missing data in X by extra/
       interpolating
63 %%   the non-missing elements. The non finite values (NaN or Inf) in
        X are
64 %%   considered as missing data. X can be any N-D array.
65
66 %%   Important note:
67 %%   ---------------
68 %%  INPAINTN uses an iterative process that converges toward the
       solution.
69 %%  Y = INPAINTN(X,N) uses N iterations. By default, N = 100. If you
70 %%  estimate that INPAINTN did not totally converge, increase N:
71 %%  Y = INPAINTN(X,1000);
72
73     for pinNumber = 1:100
74         clear valueI posXI posYI sxI syI tiltI posX0 posY0 doneX
75
76
77         [posX0 posY0 doneX] = getVoxelPinholeMap(p,pinNumber,dim);
78         if sum(isnan(posX0(:))) < dim(1)*dim(2)*dim(3);
79
80             clear match willPaint
81             %%Find the indices of all voxels that are adjacent to a
       non-empty voxel.
82             match = findNearestIndicesForFixer(doneX,2*sqrt(2));
83             willPaint = fillDone(match,doneX);
84
85             posX0 = inpaintn(posX0,100).*willPaint;
86             posY0 = inpaintn(posY0,100).*willPaint;
87
88         end
89         projectedX(:,:,:,pinNumber) = posX0;
90         projectedY(:,:,:,pinNumber) = posY0;
91     end
92
93     [k2 j2 i2] = getNextVoxelToProcess(done,start,dim);
94     m2 = coef2{k2,j2,i2};
95     thresh = 20;
96     newPin = 1;
97     for count = 1:size(m2,2)
98         clear dist;
99         dist = reshape(sqrt((m2(2,count) - projectedX(k2,j2,i2
       ,1:100)).^2 + (m2(3,count) - projectedY(k2,j2,i2,1:100)).^2),[1
       100]);
100         ind = find(dist == min(dist));
101
102         if dist(ind) < thresh
103             p(k2,j2,i2,ind).value = m2(1,count);
104             p(k2,j2,i2,ind).posX = m2(2,count);
105             p(k2,j2,i2,ind).posY = m2(3,count);
106
```

```
107                 p(k2,j2,i2,ind).sx = m2(4,count);
108                 p(k2,j2,i2,ind).sy = m2(5,count);
109                 p(k2,j2,i2,ind).tilt = m2(6,count);
110             else
111                 emptyPinhole = find(isnan(dist) == 1);
112                 emptyPinhole=emptyPinhole(newPin)
113                 newPin=newPin+1;
114                 p(k2,j2,i2,emptyPinhole).value = m2(1,count);
115                 p(k2,j2,i2,emptyPinhole).posX = m2(2,count);
116                 p(k2,j2,i2,emptyPinhole).posY = m2(3,count);
117 
118                 p(k2,j2,i2,emptyPinhole).sx = m2(4,count);
119                 p(k2,j2,i2,emptyPinhole).sy = m2(5,count);
120                 p(k2,j2,i2,emptyPinhole).tilt = m2(6,count);
121             end
122         end
123         done(k2,j2,i2) = 1;
124 end
125 
126 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
127 %% Manual Pinhole Matching
128 %% This function loads of 2 images for the user to match projections
        .
129 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
130 function pinhole = manualPinholeMatching(p0,p8,dim)
131 %Match the 00mm with 08mm data set.
132 %Need to match up the pinhole numbers.
133 
134 
135 %Display the image corresponding to coeff{1,8,8} for both data sets.
136 %Click a pinhole in the right (2nd) image and the corresponding
        pinhole in
137 %the left (1st) image.
138 %Find the nearest pinholes to the clicked location.
139 %These are now matched. Moved them into another p structure.
140 
141 donePin = zeros(dim(1),dim(2),dim(3),100);
142 for i=1:dim(3)
143     for j=1:dim(2)
144         for k=1:dim(1)
145             for p=1:100
146                 if isempty(p8(k,j,i,p).value) ~= 1
147                     donePin(k,j,i,p) = 1;
148                 end
149             end
150         end
151     end
152 end
153 
154 done = ones(dim(1),dim(2),dim(3));
155 
156 index = [1 floor(dim(2)/2 + 0.5) floor(dim(3)/2 + 0.5)];
157 for p=1:60
158     %% i=7;j=7;k=1;
159     %%Find a pinhole projection that still hasn't been associated.
160 
161 
```

```
162     %%Are there any pinholes that haven't been indexed yet?
163     %%This is recorded by the donePin array.
164     %%If it is 1, it means that this pinhole has not been indexed.
165     %%If donePin(k,j,i,1:100) is all 0, then it means that every
        pinhole for
166     %%this voxel has been indexed.
167
168     for i=1:dim(3)
169         for j=1:dim(2)
170             for k=1:dim(1)
171                 cnt=0;
172                 for n=1:100
173                     if donePin(k,j,i,n) == 1
174                         %%This pinhole has not been associated!
175                         done(k,j,i) = 0;
176                         cnt=0;
177                     else
178                         cnt=cnt+1;
179                     end
180                 end
181                 if cnt==100
182                     done(k,j,i)=1;
183                 else
184                     done(k,j,i)=0;
185                 end
186             end
187         end
188     end
189
190     if exist('index') == 1
191             %%Do I need to change image?
192     %%Default is Yes!
193     %%Check through each pinhole for this voxel and if they are not
        already associated with another pinhole (donePin == 0) then we
        don't change this image.
194
195     change = 1;
196     for n=1:100
197         if donePin(index(1),index(2),index(3),n) == 1
198             change = 0;
199         end
200     end
201
202     %%If we need to change the image.
203     %%Find the nearest voxel with a pinhole that hasn't been
        associated.
204     if change == 1
205         number = 0;
206         index = zeros(1,3);
207         distanceFromCenter = 35;
208         clear index
209         for i2=1:dim(3)
210             for j2=1:dim(2)
211                 for k2=1:dim(1)
212                     if done(k2,j2,i2) ~= 1
213                         [k j i d] = findNearestIndices(done,k2,j2,i2
        ,1,1.6);
```

```
214                         tempD = ((k2 - 1)^2 + (j2-floor(dim(2)/2 +
    0.5))^2 + (i2-floor(dim(3)/2 + 0.5))^2)^0.5;
215                         if size(k,2) >= number && d <=1.6 && tempD <
     distanceFromCenter
216                             distanceFromCenter = tempD;
217
218                             number = size(k,2);
219                             index = [k2 j2 i2];
220                         end
221                     end
222                 end
223             end
224         end
225     end
226     end
227     if exist('index') == 1
228         k=index(1);
229         j=index(2);
230         i=index(3);
231         if done(k,j,i) == 0
232
233             ktemp=k;jtemp=j;itemp=i;
234             accept = 0;
235             while accept == 0
236                 im0 = zeros(640,480);
237                 [ktemp jtemp itemp]
238                 for n=1:100
239                     if isempty(p0(ktemp,jtemp,itemp,n).value) ~= 1
240                         clear c
241                         c(:,n) = [p0(ktemp,jtemp,itemp,n).value;p0(
    ktemp,jtemp,itemp,n).posX;p0(ktemp,jtemp,itemp,n).posY;p0(ktemp,
    jtemp,itemp,n).sx;p0(ktemp,jtemp,itemp,n).sy;p0(ktemp,jtemp,itemp
    ,n).tilt];
242                         im = generatePSFfromGaussianCoefficients(c
    ,2,1);
243                         im0 = im0 + im;
244                     end
245                 end
246
247                 im8 = zeros(640,480);
248                 for n=1:100
249                     if isempty(p8(k,j,i,n).value) ~= 1
250                         clear c
251                         c(:,n) = [p8(k,j,i,n).value;p8(k,j,i,n).posX
    ;p8(k,j,i,n).posY;p8(k,j,i,n).sx;p8(k,j,i,n).sy;p8(k,j,i,n).tilt
    ];
252                         im = generatePSFfromGaussianCoefficients(c
    ,2,1);
253                         im8 = im8 + im;
254                     end
255                 end
256
257                 h1 = subplot(1,2,1),imagesc(im0);
258                 h2 = subplot(1,2,2),imagesc(im8);
259                 [y2 x2] = getpts(h2);
260                 [y1 x1] = getpts(h1);
261
```

```
262
263
264     %%If a pinhole projection isn't in the second data set for some
        reason then the user can click a variable amount of times to
        determine the direction in 3D space they want to navigate to find
         the next likely voxel with the projection.
265
266                 if (size(x2,1) == 1)
267                     accept=1;
268                 end
269                 if size(x2,1) == 2
270                     ktemp=ktemp-1;
271                     ktemp = checkForOutOfBounds(ktemp,1,dim(1));
272                 elseif size(x2,1) == 3
273                     ktemp=ktemp+1;
274                     ktemp = checkForOutOfBounds(ktemp,1,dim(1));
275                 elseif size(x2,1) == 4
276                     jtemp=jtemp-1;
277                     jtemp = checkForOutOfBounds(jtemp,1,dim(2));
278                 elseif size(x2,1) == 5
279                     jtemp=jtemp+1;
280                     jtemp = checkForOutOfBounds(jtemp,1,dim(2));
281                 elseif size(x2,1) == 6
282                     itemp=itemp-1;
283                     itemp = checkForOutOfBounds(itemp,1,dim(3));
284                 elseif size(x2,1) == 7
285                     itemp=itemp+1;
286                     itemp = checkForOutOfBounds(itemp,1,dim(3));
287                 elseif size(x1,1) > 1
288                     %repeat.
289                 end
290
291             end
292
293             %Find the closest pinhole associated with each of these.
294             dist1(1:100) = NaN;
295             dist2(1:100) = NaN;
296             for m=1:100
297                 if isempty(p0(ktemp,jtemp,itemp,m).value) ~= 1
298                     dist1(m) = sqrt((x1 - p0(ktemp,jtemp,itemp,m).
    posX)^2 + (y1 - p0(ktemp,jtemp,itemp,m).posY)^2);
299                 end
300                 if isempty(p8(k,j,i,m).value) ~= 1
301                     dist2(m) = sqrt((x2 - p8(k,j,i,m).posX)^2 + (y2
    - p8(k,j,i,m).posY)^2);
302                 end
303             end
304
305             %Find the minimum distance in p0
306             ind1 = find(dist1 == min(dist1(:)));
307             ind2 = find(dist2 == min(dist2(:)));
308             pinhole(p,1:2) = [ind1 ind2];
309
310             %Remove the projections of these pinholes in both p0 and
     p8 but keep them slightly visible to allow association by a
    person.
311             for i=1:dim(3)
```

```
312                     for j=1:dim(2)
313                         for k=1:dim(1)
314                             donePin(k,j,i,ind2) = 0;
315                             if isempty(p0(k,j,i,ind1).value)~= 1
316                                 p0(k,j,i,ind1).value = 10;
317                                 p0(k,j,i,ind1).sx = 1;
318                                 p0(k,j,i,ind1).sy = 1;
319                             end
320                             if isempty(p8(k,j,i,ind2).value)~= 1
321                                 p8(k,j,i,ind2).value = 10;
322                                 p8(k,j,i,ind2).sx = 1;
323                                 p8(k,j,i,ind2).sy = 1;
324                             end
325                         end
326                     end
327                 end
328             end
329         end
330 end
```

B.2.3 Extrapolation of Pinhole Projections

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %% Extrapolate Pinhole Projections
3  %%
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  function [new_pDistance distances] =
       extrapolateHighMagPinholeCoordinates(filename_Coeff,p_0,p_8,xSize
       ,distance_PD,pinhole,dim,threshold)
7
8  new_p0(dim(1),dim(2),dim(3),100) = struct('value',[],'posX',[],'posY
       ',[],'sx',[],'sy',[],'tilt',[]);
9  new_p8(dim(1),dim(2),dim(3),100) = struct('value',[],'posX',[],'posY
       ',[],'sx',[],'sy',[],'tilt',[]);
10 new_pDistance(dim(1),dim(2),dim(3),100) = struct('value',[],'posX'
       ,[],'posY',[],'sx',[],'sy',[],'tilt',[]);
11
12
13 for i=1:dim(3)
14     for j=1:dim(2)
15         for k=1:dim(1)
16             for p = 1:size(pinhole,1)
17                 new_p0(k,j,i,p) = p_0(k,j,i,pinhole(p,1));
18                 new_p8(k,j,i,p) = p_8(k,j,i,pinhole(p,2));
19             end
20         end
21     end
22 end
23
24 projectedX = zeros(dim(1),dim(2),dim(3),100);
25 projectedY = zeros(dim(1),dim(2),dim(3),100);
26 coef = getCoefficients(filename_Coeff,dim);
27 coef2 = coef;
```

```
28 if xSize == 320
29     for i=1:dim(3)
30         for j=1:dim(2)
31             for k=1:dim(1)
32                 if isempty(coef2{k,j,i})~=1
33                     coef2{k,j,i}(2,:) = 2*coef2{k,j,i}(2,:);
34                     coef2{k,j,i}(3,:) = 2*coef2{k,j,i}(3,:);
35                     coef2{k,j,i}(4,:) = 2*coef2{k,j,i}(4,:);
36                     coef2{k,j,i}(5,:) = 2*coef2{k,j,i}(5,:);
37                 end
38             end
39         end
40     end
41 end
42
43 for pinNumber = 1:100
44     clear valueI posXI posYI sxI syI tiltI
45
46     %Load in all the information about a particular pinhole for 00mm
       and
47     %08mm
48
49     [posX0 posY0 doneY] = getVoxelPinholeMap(new_p0,pinNumber,dim);
50     [posX8 posY8 doneX] = getVoxelPinholeMap(new_p8,pinNumber,dim);
51
52   if sum(isnan(posX8(:))) < dim(1)*dim(2)*dim(3)-1;
53
54         clear match
55         %Find all voxels that are adjacent to a non-empty voxel
56         match = findNearestIndicesForFixer(doneX,3*sqrt(2));
57         willPaint = fillDone(match,doneX);
58
59         posX0 = inpaintn(posX0,1000).*willPaint;
60         posY0 = inpaintn(posY0,1000).*willPaint;
61         posX8 = inpaintn(posX8,1000).*willPaint;
62         posY8 = inpaintn(posY8,1000).*willPaint;
63         xx = 0:1;
64         for i=1:dim(3)
65             for j=1:dim(2)
66                 for k=1:dim(1)
67                     if distance_PD > 8
68                         if isnan(posX0(k,j,i)) ~= 1 && isnan(posX8(k
       ,j,i)) ~= 1
69                             a(1) = posX0(k,j,i);
70                             a(2) = posX8(k,j,i);
71                             projectedX(k,j,i,pinNumber) = interp1(xx
       ,a,(distance_PD/8),'linear','extrap');
72                             a(1) = posY0(k,j,i);
73                             a(2) = posY8(k,j,i);
74                             projectedY(k,j,i,pinNumber) = interp1(xx
       ,a,(distance_PD/8),'linear','extrap');
75                         end
76                     else
77                         if isnan(posX0(k,j,i)) ~= 1
78                             projectedX(k,j,i,pinNumber) = posX0(k,j,
       i);
```

```
79                                  projectedY(k,j,i,pinNumber) = posY0(k,j,
    i);
80                              end
81                          end
82                      end
83                  end
84              end
85          end
86 end
87
88 distances = zeros(dim(1),dim(2),dim(3),100);
89 cnt=1;
90 for i=1:dim(3)
91     for j=1:dim(2)
92         for k=1:dim(1)
93             m2 = coef2{k,j,i};
94             num=0;
95             for count = 1:size(m2,2)
96                 clear dist;
97
98                 dist = reshape(sqrt((m2(2,count) - projectedX(k,j,i
    ,1:100)).^2 + (m2(3,count) - projectedY(k,j,i,1:100)).^2),[1
    100]);
99                 ind = find(dist == min(dist));
100                distances(k,j,i,ind) = dist(ind);
101
102                %This projection probably belongs to this pinhole!
103
104                if dist(ind) < threshold
105
106                    if isempty(new_pDistance(k,j,i,ind).value)== 1
107                        new_pDistance(k,j,i,ind).value = m2(1,count)
    ;
108                        new_pDistance(k,j,i,ind).posX = m2(2,count);
109                        new_pDistance(k,j,i,ind).posY = m2(3,count);
110                        new_pDistance(k,j,i,ind).sx = m2(4,count);
111                        new_pDistance(k,j,i,ind).sy = m2(5,count);
112                        new_pDistance(k,j,i,ind).tilt = m2(6,count);
113                    end
114                else
115                    disp('warning');
116                    [k j i]
117                end
118            end
119
120        end
121    end
122 end
123
124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
125 %% Extrapolate Weak Pinhole Projections
126 %%
127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
128 %This function uses previous data which matched strong pinhole
    projections.
129 %This data is processed using the 'Master' m file.
```

```
130 %Traditional code formatting is relaxed here due to the number of '
      for loops' in this function.
131
132 clear
133 for systemSetup = 1:6
134 clear p
135 switch systemSetup
136 case 1
137 %Read in the old data which is prefectly matched. Associate it with
      the new
138 %data.
139 load 8-20-12-p0mm_PE_v1_complete.mat
140 case 2
141 load 8-20-12-p8mm_PE_v1_complete.mat
142 case 3
143 load 8-20-12-p16mm_PE_v1_complete.mat
144 case 4
145 load 8-20-12-p24mm_PE_v1_complete.mat
146 case 5
147 load 8-20-12-p27mm_PE_v1_complete.mat
148 case 6
149 load 8-20-12-p37mm_PE_v1_complete.mat
150 end
151
152 for repeat=1:6
153 switch systemSetup
154 case 1
155 fn = fopen('E:\PSF\New Intensifier\6-22-12-00mm\centroids_00mm.dat',
      'r');
156 h = fread(fn,3,'int');
157 fileOut = ['E:\PSF\New Intensifier\6-22-12-00mm\' '
      gaussianParams_normalized_00mm_PE_v2_complete.dat'];
158 case 2
159 fn = fopen('E:\PSF\New Intensifier\6-22-12-08mm\centroids_08mm.dat',
      'r');
160 h = fread(fn,3,'int');
161 fileOut = ['E:\PSF\New Intensifier\6-22-12-08mm\' '
      gaussianParams_normalized_08mm_PE_v2_complete.dat'];
162 case 3
163 fn = fopen('E:\PSF\New Intensifier\6-22-12-16mm\centroids_16mm.dat',
      'r');
164 h = fread(fn,3,'int');
165 fileOut = ['E:\PSF\New Intensifier\6-22-12-16mm\' '
      gaussianParams_normalized_16mm_PE_v2_complete.dat'];
166 case 4
167 fn = fopen('E:\PSF\New Intensifier\6-22-12-24mm\centroids_24mm.dat',
      'r');
168 h = fread(fn,3,'int');
169 fileOut = ['E:\PSF\New Intensifier\6-22-12-24mm\' '
      gaussianParams_normalized_24mm_PE_v2_complete.dat'];
170 case 5
171 fn = fopen('E:\PSF\New Intensifier\6-22-12-27mm\centroids_27mm.dat',
      'r');
172 h = fread(fn,3,'int');
173 fileOut = ['E:\PSF\New Intensifier\6-22-12-27mm\' '
      gaussianParams_normalized_27mm_PE_v2_complete.dat'];
174 case 6
```

```
175 fn = fopen('E:\PSF\New Intensifier\6-22-12-37mm\centroids_37mm.dat',
    'r');
176 h = fread(fn,3,'int');
177 fileOut = ['E:\PSF\New Intensifier\6-22-12-37mm\' '
    gaussianParams_normalized_37mm_PE_v2_complete.dat'];
178
179     end
180
181
182     %%Load in all the data and predict pinhole locations!
183     %%Declare the Variables.
184     clear pos* sx sy posX* posY* sx*
185     posXAll{100} = [];
186     posYAll{100} = [];
187     sxAll{100} = [];
188     syAll{100} = [];
189
190
191     %%Isolate each pinhole. Inpaintn the data.
192     exists = zeros(13,13,13,100);
193     for pinhole=1:100
194     posX = NaN(13,13,13);
195     posY = NaN(13,13,13);
196     sx = NaN(13,13,13);
197     sy = NaN(13,13,13);
198     accept = 0;
199     for i=1:13
200     for j=1:13
201     for k=1:13
202     if isempty(p(k,j,i,pinhole).value)~= 1
203     posX(k,j,i) = p(k,j,i,pinhole).posX;
204     posY(k,j,i) = p(k,j,i,pinhole).posY;
205     sx(k,j,i) = p(k,j,i,pinhole).sx;
206     sy(k,j,i) = p(k,j,i,pinhole).sy;
207     exists(k,j,i,pinhole) = 1;
208     accept = 1;
209     end
210     end
211     end
212     end
213     if accept == 1
214     posX = fillVolBox(posX);
215     posY = fillVolBox(posY);
216     sx = fillVolBox(sx);
217     sy = fillVolBox(sy);
218
219     [posX0 posY0 doneX] = getVoxelPinholeMap(p,pinhole,[13 13 13])
    ;
220     match = findNearestIndicesForFixer(doneX,sqrt(2));
221     willPaint = fillDone(match,doneX);
222
223     posX = posX.*willPaint;
224     posY = posY.*willPaint;
225     sx = sx.*willPaint;
226     sy = sy.*willPaint;
227     posXAll{pinhole} = posX;
228     posYAll{pinhole} = posY;
```

```
229         sxAll{pinhole} = sx;
230         syAll{pinhole} = sy;
231         end
232         end
233         clear a* A* G* done* match posX posX0 posY posY0 sx sy vol*
    will*
234         %All the pinhole locations are now predicted.
235         %Loop through all the centroid images.
236         %Check all the predicted locations.
237         %Have they already been accounted for?
238         %Yes? - move on.
239         %No? - fit a Gaussian to the area.
240         cnt=1;
241         for i=1:13
242         for j=1:13
243         for k=1:13
244         imR = reshape(fread(fn,320*240,'float32'),[320 240]);
245
246         imR = imR';
247         imR2=imR;
248
249         %Remove all already known pinholes from the image.
250         for pin = 1:50
251         if exists(k,j,i,pin) == 1
252         %This pinhole projection from this voxel has already
253         %been recorded.
254         posX = posXAll{pin};
255         posX = posX/2;
256         posY = posYAll{pin};
257         posY = posY/2;
258         sx = sxAll{pin};
259         sx = sx/2;
260         sy = syAll{pin};
261         sy = sy/2;
262         if sum(isnan(posX(:))) ~= numel(posX)
263         if posY(k,j,i)~= 0
264         imR(floor(posY(k,j,i)-12 + 0.5):floor(posY(k,j,i)+12 + 0.5),
    floor(posX(k,j,i)-12 + 0.5):floor(posX(k,j,i)+12 + 0.5)) = 0;
265         end
266         end
267         end
268         end
269
270         %All the known pinhole projections have now been removed so
271         %anything left is either noise or a weak source.
272         for pin = 1:50
273         if exists(k,j,i,pin) == 1
274         %Do nothing here.
275         else
276         %The pinhole projection has not been recorded. It may
277         %be very feint or it may not be present.
278
279         posX = posXAll{pin};
280         posX = posX/2;
281         posY = posYAll{pin};
282         posY = posY/2;
283         sx = sxAll{pin};
```

```
284         sx = sx/2;
285         sy = syAll{pin};
286         sy = sy/2;
287         if sum(isnan(posX(:))) ~= numel(posX) && posY(k,j,i)~= 0
288         %Extract the neighborhood as predicted.
289         %Check if neighborhood bounds will be outside the
290         %image.
291         padsize = 8;
292         offsetNeg = padsize/2;
293         offsetPos = padsize/2 - 1;
294
295         skip = 0;
296         if floor(posY(k,j,i)-offsetNeg + 0.5) < 1 || floor(posY(k,j,i)
        +offsetPos + 0.5)> size(imR,1) || floor(posX(k,j,i)-offsetNeg +
        0.5) < 1 || floor(posX(k,j,i)+offsetPos + 0.5)> size(imR,2)
297         offsetNeg = 6;
298         offsetPos = 5;
299         end
300         if floor(posY(k,j,i)-offsetNeg + 0.5) < 1 || floor(posY(k,j,i)
        +offsetPos + 0.5)> size(imR,1) || floor(posX(k,j,i)-offsetNeg +
        0.5) < 1 || floor(posX(k,j,i)+offsetPos + 0.5)> size(imR,2)
301         skip = 1;
302         end
303
304         if skip == 0
305         neighborhood = imR(floor(posY(k,j,i)-offsetNeg + 0.5):floor(
        posY(k,j,i)+offsetPos + 0.5),floor(posX(k,j,i)-offsetNeg + 0.5):
        floor(posX(k,j,i)+offsetPos + 0.5));
306         %This neighborhood will most likely be all zeros or
307         %have some noise. It may also have some weak signal.
308
309         gTemp = neighborhood;
310         gTemp(gTemp<=2) = 0;
311         gTemp(gTemp<=(max(gTemp(:))*0.1)) = 0;
312         [r c] = find(gTemp == max(gTemp(:)));
313         if size(r,1) < 10 %Probably an image with central point.
314         %If there are multiple max values, which is the most
        appropriate
315         %location to pick? -> Select the indice with the largest
        cluster value surrounding it.
316         bestInd = bestMax(imR,r,c,posY(k,j,i),posX(k,j,i),offsetNeg);
317         pixel = [floor(posY(k,j,i)-offsetNeg + 0.5)+r(bestInd)-1 floor
        (posX(k,j,i)-offsetNeg + 0.5)+c(bestInd)-1];
318         gTemp = imR(pixel(1)-offsetNeg:pixel(1)+offsetPos,pixel(2)-
        offsetNeg:pixel(2)+offsetPos);
319         gTemp(gTemp<=2) = 0;
320         gTemp(gTemp<=(max(gTemp(:))*0.1)) = 0;
321         [r c] = find(gTemp == max(gTemp(:)));
322         bestInd = bestMax(imR,r,c,pixel(1),pixel(2),offsetNeg);
323
324         if sum(sum(gTemp,1)>0) < 2 || sum((sum(gTemp,2)>0)) < 2
325         gTemp = imagePadSmallValues(gTemp,r(bestInd),c(bestInd));
326         end
327         [mx, my, var_x, var_y, covar_xy, ~, ~, ~, ~] = moments(
        gTemp);
328         mx = mx+pixel(2)-(offsetNeg + 1);
329         my = my+pixel(1)-(offsetNeg + 1);
```

```
330         mxT = floor(mx + 0.5);
331         myT = floor(my + 0.5);
332         g = imR(myT-offsetNeg:myT+offsetPos,mxT-offsetNeg:mxT+
    offsetPos);
333         gTemp = g;
334         gTemp(gTemp<=2) = 0;
335         gTemp(gTemp<=(max(gTemp(:))*0.1)) = 0;
336         [r c] = find(gTemp == max(gTemp(:)));
337 
338         %If there is a valid signal in the neighborhood it will
339         %have certain characteristics.
340 
341         %There should only be small number of max values.
342         %If r or c have too many elements it means that there
343         %is no signal in the neighborhood. It is unlikely that
344         %a real signal would be spread uniformly over large
345         %number of pixels.
346         if size(r,1) <= 10
347         bestInd = bestMax(imR,r,c,myT,mxT,offsetNeg);
348         %Isolate the area around the cluster.
349         mask = 1*bwselect(gTemp,c(bestInd),r(bestInd),8);
350         nonZeros = find(mask>0);
351         %If the cluster size is larger than 1 then it could
352         %be a signal.
353         if size(nonZeros,1) >= 2 || max(max(gTemp.*mask))>3
354         mask = increaseBoundary(mask,sqrt(2));
355         g = g.*mask;
356 
357         %This g has a weak cluster. Fit a Gaussian to it.
358 
359         gTemp = g;
360         [r c] = find(gTemp == max(gTemp(:)));
361 
362         if sum(sum(gTemp,1)>0) < 2 || sum((sum(gTemp,2)>0)) < 2
363 
364         gTemp = imagePadSmallValues(gTemp,r(1),c(1));
365 
366         end
367 
368         [mxt, myt, var_xt, var_yt, covar_xyt, ~, ~, ~, ~, ~] = moments
    (gTemp);
369         sxt = sqrt(var_xt);
370         syt = sqrt(var_yt);
371         rho_xyt = covar_xyt/sxt/syt;
372         max_pixelt = max(g(:));
373         offset_percentage = 0.5;
374 
375         offset_percentage_Center = 0.5;
376         %Upper and Lower Bounds with fmincon
377         %A_lb = 1;
378         A_lb = 1e-10;
379         A_ub = 1000*max_pixelt;
380         xm_lb = mxt-abs(mxt*offset_percentage_Center);
381         xm_ub = mxt+abs(mxt*offset_percentage_Center);
382         ym_lb = myt-abs(myt*offset_percentage_Center);
383         ym_ub = myt+abs(myt*offset_percentage_Center);
384         sx_lb = sxt-abs(sxt*offset_percentage);
```

```
385         sx_ub = sxt+10*abs(sxt*offset_percentage);
386         if sx_ub < 3
387         sx_ub  = 5;
388         end
389         sy_lb = syt-abs(syt*offset_percentage);
390         sy_ub = syt+10*abs(syt*offset_percentage);
391         if sy_ub < 3
392         sy_ub  = 5;
393         end
394
395         x = [max_pixelt mxt myt sxt syt rho_xyt];
396
397         if sx_lb<=0
398         sx_lb=0.1;
399         end
400         if sy_lb<=0
401         sy_lb=0.1;
402         end
403         rho_xy_lb=-0.99;
404         rho_xy_ub=0.99;
405
406         dim_x = size(g,2);
407         dim_y = size(g,1);
408         g=g(:);
409
410         [Gauss_Params, fval, exitflag] = fmincon('GaussFit',x
    ,[],[],[],[],[A_lb xm_lb ym_lb sx_lb sy_lb rho_xy_lb],[A_ub xm_ub
     ym_ub sx_ub sy_ub rho_xy_ub],[],optimset('Algorithm', 'sqp','
    TolX',1e-10,'TolFun',1e-10,'MaxFunEvals',100000,'MaxIter', 10000,
     'Display', 'notify-detailed'),g,dim_y, dim_x);
411
412         %Have limits been reached?
413
414         if sum(isreal(Gauss_Params(:))) == 0
415         disp('ERROR: Complex Number Detected');
416         pause
417         disp('Continue');
418         end
419
420         Gauss_Params(2) = Gauss_Params(2) + mxT-(offsetNeg +1);
421         Gauss_Params(3) = Gauss_Params(3) + myT-(offsetNeg +1);
422
423         %This is to renormalize the gaussian.
424         sum_pre = sum(g(:));
425         sum_post = sum(sum(generatePSFfromGaussianCoefficients(
    Gauss_Params',2,2)));
426
427       Gauss_Params(1) = Gauss_Params(1)/sum_post*sum_pre;
428       %Normalize it to Acquisition time.
429       Gauss_Params(1) = Gauss_Params(1)*100/(10*(15-k));
430       p(k,j,i,pin).value = Gauss_Params(1);
431       p(k,j,i,pin).posX = 2*Gauss_Params(2);
432       p(k,j,i,pin).posY = 2*Gauss_Params(3);
433       p(k,j,i,pin).sx = 2*Gauss_Params(4);
434       p(k,j,i,pin).sy = 2*Gauss_Params(5);
435       p(k,j,i,pin).tilt = Gauss_Params(6);
436       exists(k,j,i,pin) = 1;
```

```
437
438       imR( floor(Gauss_Params(3)-offsetNeg + 0.5):floor(Gauss_Params
      (3)+offsetPos +0.5),floor(Gauss_Params(2)-offsetNeg + 0.5):floor(
      Gauss_Params(2)+offsetPos + 0.5)) = 0;
439       end
440       end
441       end
442       end
443       end
444       end
445       end
446       end
447       end
448       end
449
450       fclose(fn);
451
452       end
453
454       writePtoFileAsCoeff(fileOut,p);
455       switch systemSetup
456       case 1
457       save('8-20-12-p0mm_PE_v2_complete.mat','p');
458       case 2
459       save('8-20-12-p8mm_PE_v2_complete.mat','p');
460       case 3
461       save('8-20-12-p16mm_PE_v2_complete.mat','p');
462       case 4
463       save('8-20-12-p24mm_PE_v2_complete.mat','p');
464       case 5
465       save('8-20-12-p27mm_PE_v2_complete.mat','p');
466       case 6
467       save('8-20-12-p37mm_PE_v2_complete.mat','p');
468 end
469 end
470
471 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
472 %% Extrapolate Values in a Volume
473 %%
474 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
475 function posX = fillVolBox(posX)
476 %This function pads a 3D array and inpaints it.
477
478 volBoxPosX = NaN(size(posX,1)+4,size(posX,2)+4,size(posX,3)+4);
479 %Put the data in the middle
480 volBoxPosX(3:size(volBoxPosX,1)-2,3:size(volBoxPosX,2)-2,3:size(
      volBoxPosX,3)-2) = posX;
481 %Inpaint the box
482 volBoxPosX = inpaintn(volBoxPosX,1000);
483 %Extract the data.
484 posX = volBoxPosX(3:size(volBoxPosX,1)-2,3:size(volBoxPosX,2)-2,3:
      size(volBoxPosX,3)-2);
```

B.2.4 Interpolation of Pinhole Projections

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Interpolate Pinhole Projections
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [coeffInterpN2 p1] = interpolateinterpnWithMask_x3_simple(p
    ,filter,paint,range,inc,newInc,dim,interpolationMethod,varargin)

[x1 y1 z1] = ndgrid(range(1):inc:range(2),range(1):inc:range(2),
    range(1):inc:range(2));
[x4 y4 z4] = ndgrid(range(1):newInc:range(2),range(1):newInc:range
    (2),range(1):newInc:range(2));


interpDim = [size(range(1):newInc:range(2),2) size(range(1):newInc:
    range(2),2) size(range(1):newInc:range(2),2)];
mask = ones(640,480);

coeffInterpN2{interpDim(1),interpDim(2),interpDim(3)} = [];
p1(interpDim(1),interpDim(2),interpDim(3),100) = struct('value',[],'
    posX',[],'posY',[],'sx',[],'sy',[],'tilt',[]);

for pinNumber = 1:100
    clear valueI posXI posYI sxI syI tilt done valueI2 posXI2 posYI2
    sxI2 syI2 tiltI2
    [posX0 posY0 done] = getVoxelPinholeMap(p,pinNumber,dim);

    if paint == 1
        valueI(1:dim(1),1:dim(2),1:dim(3)) = NaN;
        posXI(1:dim(1),1:dim(2),1:dim(3)) = NaN;
        posYI(1:dim(1),1:dim(2),1:dim(3)) = NaN;
        sxI(1:dim(1),1:dim(2),1:dim(3)) = NaN;
        syI(1:dim(1),1:dim(2),1:dim(3)) = NaN;
        tiltI(1:dim(1),1:dim(2),1:dim(3)) = NaN;
    else
        valueI(1:dim(1),1:dim(2),1:dim(3)) = 0;
        posXI(1:dim(1),1:dim(2),1:dim(3)) = 0;
        posYI(1:dim(1),1:dim(2),1:dim(3)) = 0;
        sxI(1:dim(1),1:dim(2),1:dim(3)) = 0;
        syI(1:dim(1),1:dim(2),1:dim(3)) = 0;
        tiltI(1:dim(1),1:dim(2),1:dim(3)) = 0;
    end

    accept = zeros(dim(1),dim(2),dim(3));
    for i=1:dim(3)
        for j=1:dim(2)
            for k=1:dim(1)
                if isempty(p(k,j,i,pinNumber).value) ~= 1
                    valueI(k,j,i) = p(k,j,i,pinNumber).value;
                    posXI(k,j,i) = p(k,j,i,pinNumber).posX;
                    posYI(k,j,i) = p(k,j,i,pinNumber).posY;
                    sxI(k,j,i) = p(k,j,i,pinNumber).sx;
                    syI(k,j,i) = p(k,j,i,pinNumber).sy;
                    tiltI(k,j,i) = p(k,j,i,pinNumber).tilt;
                    accept(k,j,i) = 1;
```

```
49                  end
50              end
51          end
52      end
53      if sum(accept(:)) > 0
54
55          if filter~=0
56              if filter == 1 || filter == 2
57                  if pinNumber==1
58                      disp('Median & Smooth Filters Selected');
59                  end
60                      valueI = filterVolBox(valueI,filter,1);
61                      posXI = filterVolBox(posXI,filter,1);
62                      posYI = filterVolBox(posYI,filter,1);
63                      sxI = filterVolBox(sxI,filter,1);
64                      syI = filterVolBox(syI,filter,1);
65                      tiltI = filterVolBox(tiltI,filter,1);
66              end
67          end
68
69          clear done2
70          done2 = interpn(x1,y1,z1,done,x4,y4,z4,'linear');
71
72          valueI = fillArrayInPaint(valueI);
73          posXI = fillArrayInPaint(posXI);
74          posYI = fillArrayInPaint(posYI);
75          sxI = fillArrayInPaint(sxI);
76          syI = fillArrayInPaint(syI);
77          tiltI = fillArrayInPaint(tiltI);
78          valueI(isnan(valueI(:))==1)=0;
79
80          done2(done2>0) = 1;
81
82          valueI2 = interpn(x1,y1,z1,valueI,x4,y4,z4,
    interpolationMethod,0).*done2;
83          posXI2 = interpn(x1,y1,z1,posXI,x4,y4,z4,interpolationMethod
    ,0).*done2;
84          posYI2 = interpn(x1,y1,z1,posYI,x4,y4,z4,interpolationMethod
    ,0).*done2;
85          sxI2 = interpn(x1,y1,z1,sxI,x4,y4,z4,interpolationMethod,0)
    .*done2;
86          syI2 = interpn(x1,y1,z1,syI,x4,y4,z4,interpolationMethod,0)
    .*done2;
87          tiltI2 = interpn(x1,y1,z1,tiltI,x4,y4,z4,interpolationMethod
    ,0).*done2;
88
89          %Remove out of bound values.
90          ind = find(valueI2<0);
91          if size(ind,1)>0
92              valueI2(ind(:))=0;
93              posXI2(ind(:))=0;
94              posYI2(ind(:))=0;
95              sxI2(ind(:))=0;
96              syI2(ind(:))=0;
97              tiltI2(ind(:))=0;
98          end
```

```
99          ind = [find(tiltI2<=-1 | tiltI2>=1); find(sxI2<0); find(syI2
    <0)];
100         if size(ind,1)>0
101             valueI2(ind(:))=0;
102             posXI2(ind(:))=0;
103             posYI2(ind(:))=0;
104             sxI2(ind(:))=0;
105             syI2(ind(:))=0;
106             tiltI2(ind(:))=0;
107         end
108
109
110         for i=1:interpDim(3)
111             for j=1:interpDim(2)
112                 for k=1:interpDim(1)
113                     if valueI2(k,j,i) >= 0.1
114                         if floor(posYI2(k,j,i) + 0.5) > 0 && (
    floor(posXI2(k,j,i) + 0.5)) > 0 && floor(posYI2(k,j,i) + 0.5)
    <480 && (floor(posXI2(k,j,i) + 0.5)) < 640
115                             if mask(floor(posXI2(k,j,i) + 0.5),
    floor(posYI2(k,j,i) + 0.5)) == 1
116                                 if isempty(coeffInterpN2{k,j,i})
     == 1
117                                     coeffInterpN2{k,j,i} = [
    valueI2(k,j,i);posXI2(k,j,i);posYI2(k,j,i);sxI2(k,j,i);syI2(k,j,i
    );tiltI2(k,j,i)];
118                                 else
119                                     coeffInterpN2{k,j,i} = [
    coeffInterpN2{k,j,i} [valueI2(k,j,i);posXI2(k,j,i);posYI2(k,j,i);
    sxI2(k,j,i);syI2(k,j,i);tiltI2(k,j,i)]];
120                                 end
121                             end
122                         end
123                     end
124                 end
125             end
126         end
127     end
128  end
129  clear valueI posXI posYI sxI syI tiltI
130 end
131
132
133 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
134 %% Extrapolate Values in a Volume
135 %%
136 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
137 function array = fillArrayInPaint(array)
138
139 done = ones(size(array));
140 %Create a 'done' matrix
141 ind = find(array==0 | isnan(array)==1);
142 if size(ind,1)>0
143     done(ind(:))=0;
144     %Find the ideal starting point.
145     %This point will have as many values as possible surrounding it,
    not
```

```
146     %necessarily the largest values.
147     [k j i] = returnLargestNeighborhood(array);
148     start = [k j i];
149 
150     counter=0;
151     continueLoop=1;
152     while continueLoop>0 && size(ind,1)>counter
153         [k2 j2 i2] = getNextVoxelToProcess(done,start,size(done));
154         [value continueLoop] = returnAveragePredictedValueLinear(
    array,k2,j2,i2);
155 
156         if continueLoop>0
157             array(k2,j2,i2)=value;
158         end
159         done(k2,j2,i2)=1;
160         counter=counter+1;
161     end
162 end
163 
164 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
165 %% Extrapolate Values in a Volume 1D
166 %% Called from : fillArrayInPaint.m
167 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
168 function [value cnt] = returnAveragePredictedValueLinear(volMiBox,k,
    j,i)
169 
170 value = 0;
171 cnt=0;
172 
173 for dimension = 1:3
174     switch dimension
175         case 1
176             %Original setup
177             volMi2 = permute(volMiBox,[1 2 3]);
178         case 2
179             %Change axes setup
180             volMi2 = permute(volMi2,[3 1 2]);%3 1 2
181         case 3
182             volMi2 = permute(volMi2,[3 1 2]);%2 3 1
183     end
184     switch dimension
185         case 1
186             im = volMi2(:,j,i);
187             if sum(isnan(im(:))) == size(im,1)*size(im,2) || sum(
    isnan(im(:))==0) <2
188             else
189                 val = local_Interp_v2(im,k);
190                 if val>0
191                     value = value + val;
192                     cnt=cnt+1;
193                 end
194 
195             end
196         case 2
197             im = volMi2(:,k,j);
198             if sum(isnan(im(:))) == size(im,1)*size(im,2) || sum(
    isnan(im(:))==0) <2
```

```
199             else
200                 val = local_Interp_v2(im,i);
201                 if val>0
202                     value = value + val;
203                     cnt=cnt+1;
204                 end
205             end
206
207         case 3
208             im = volMi2(:,i,k);
209             if sum(isnan(im(:))) == size(im,1)*size(im,2)   || sum(
    isnan(im(:))==0) <2
210             else
211                 val = local_Interp_v2(im,j);
212                 if val>0
213                     value = value + val;
214                     cnt=cnt+1;
215                 end
216
217             end
218     end
219 end
220 if cnt>0
221     value = value/cnt;
222 end
223 end
224
225 %%SUBFUNCTIONS - Linear Interpolation of the Arrays of '
    returnAveragePredictedValueLinear.m'
226 function val = local_Interp(im,i)
227 val=0;
228 if i+2<=size(im,1) && i-2>=1
229     runF = 0;
230     if isnan(im(i+1))==0 && isnan(im(i+2))==0 && runF == 0
231         val = interp1([0 1],[im(i+2) im(i+1)],2,'linear','extrap');
232         runF = 1;
233     end
234     if isnan(im(i-1))==0 && isnan(im(i-2))==0 && runF == 0
235         val = interp1([0 1],[im(i-2) im(i-1)],2,'linear','extrap');
236         runF = 1;
237     end
238 elseif i+2<=size(im,1) && i-2<=0
239     runF = 0;
240     if isnan(im(i+1))==0 && isnan(im(i+2))==0 && runF == 0
241         val = interp1([0 1],[im(i+2) im(i+1)],2,'linear','extrap');
242         runF = 1;
243     end
244 elseif i+2>size(im,1) && i-2>=1
245     runF = 0;
246     if isnan(im(i-1))==0 && isnan(im(i-2))==0 && runF == 0
247         val = interp1([0 1],[im(i-2) im(i-1)],2,'linear','extrap');
248         runF = 1;
249     end
250 end
251 end
252
```

```
253 %% Cubic Interpolation of the Arrays of '
    returnAveragePredictedValueLinear.m'
254 function val = local_Interp_v2(im,i)
255 val=0;
256
257 positionP=i;
258 while positionP+1 <= size(im,1) && isnan(im(positionP+1))==0
259     positionP=positionP+1;
260 end
261
262
263 positionN=i;
264 while positionN-1 > 0 && isnan(im(positionN-1))==0
265     positionN=positionN-1;
266 end
267
268 if abs(positionP - i) > 2 || abs(positionN - i) > 2
269 %Which has the most data points: positive or negative direction?
270 if abs(positionP - i) > abs(positionN-i)
271     if sum(isnan(im(i+1:positionP)))>0
272         im(i+1:positionP)
273         pause
274     end
275     %Can put a factor into the csaps function. The default value is
    used
276     %here.
277     s = fnxtr(csaps(i+1:positionP,im(i+1:positionP)));
278     val = fnval(s,i);
279 else
280     s = fnxtr(csaps(positionN:i-1,im(positionN:i-1)));
281     val = fnval(s,i);
282 end
283 else
284     val = 0;
285 end
286 end
```

Bibliography

80/20, Inc. `http://www.8020.net`.

Abex (UK). `http://www.abex.com/`.

Accorsi, R (2007). "MediSPECT: single photon emission computed tomography system for small field of view small animal imaging based on a CdTe hybrid pixel detector". *Nucl. Instrum. Methods Phys. Res.* 571.1-2, p. 44.

Accorsi, R and Metzler, S. D. (2005). "Resolution-effective diameters for asymmetric-knife-edge pinhole collimators". *IEEE Trans. Med. Imaging* 12, pp. 1637–1646.

Accorsi, R, Gasparini, F, and Lanza, R. C. (2001). "A coded aperture for high-resolution nuclear medicine planar imaging with a conventional Anger camera: Experimental results". *IEEE Trans. Nucl. Sci.* 48, pp. 2411–2417.

Acton, P. D. and Kung, H. F. (2003). "Small animal imaging with high resolution single photon emission tomography". *Nuclear Medicine and Biology* 30.8, pp. 889–895.

Anger, H. O. (1952). "Use of a gamma-ray pinhole camera for in vivo studies". *Nature* 170, pp. 200–201.

Anger, H. O. (1958). "Scintillation Camera". *Review of Scientific Instruments* 29.1, pp. 27–33.

Barber, H. B. and Woolfenden, J. M. (2006). "Nuclear Medicine". Ed. by R. E. Henkin. Mosby, St. Louis, pp. 137–153.

Barrett, H. H. (1990a). "Limited angular tomography for the nineties". *Journal of Nuclear Medicine* 31, pp. 1688 –1692.

Barrett, H. H. (1990b). "Objective assessment of image quality: effects of quantum noise and object variability". *J. Opt. Soc. Am. A* 7, pp. 1266–1277.

Barrett, H. H. and Gifford, H. C. (1994). "Cone-beam tomography with discrete data sets". *Phys. Med. Biol.* 39, pp. 451–476.

Barrett, H. H. and Hunter, W. C. J. (2005). "Small-Animal SPECT". Springer. Chap. Detectors for small-animal SPECT: I, Overview of technologies, pp. 9–48.

Barrett, H. H. and Myers, K. J. (2004). *Foundations of image science*. John Wiley and Sons.

Barrett, H. H. and Swindell, W (1981). *Radiological Imaging: The theory of image formation, detection, and processing*. Academic Press.

Barrett, H. H., Denny, J, Wagner, R. F., and Myers, K. J. (1995). "Objective Assessment of image quality II: Fisher information, Fourier crosstalk and figures of merit for task performance". *J. Opt. Soc. Am. A* 12, pp. 834–852.

Barrett, H. H., Denny, J. L., Gifford, H. C., Abbey, C. K., Wagner, R. F., and Myers, K. J. (1996). "Generalized NEQ: Fourier analysis where you would least expect to find it". *SPIE* 2708, pp. 41–52.

Barrett, H. H., Hunter, W. C. J., Miller, B. W., Moore, S. K., Chen, Y, and Furenlid, L. R. (2009). "Maximum-likelihood methods for processing signals from gamma-ray detectors". *IEEE Trans. Nucl. Sci.* 56.3, pp. 725–735.

Beekman, F. J. and Have, F Van der (2007). "The pinhole: gateway to ultra-high-resolution three-dimensional radionuclide imaging". *Eur. J. Nucl. Med. Mol. Imaging* 34.2, pp. 151–161.

Beekman, F. J., Kamphuis, C, King, M. A., Rijk, P. P., and Viergever, M. A. (2001). "Improvement of image resolution and quantitative accuracy in clinical Single Photon Emission Computed Tomography". *Computerized Medical Imaging and Graphics* 25, pp. 135–146.

Beekman, F. J., Have, F van der, Vastenhouw, B, Linden, A. J. A. van der, Rijk, P. P. van, Burbach, J. P. H., and Smidt, M. P. (2005). "U-SPECT-I: a novel System for submillimeter-resolution tomography with radiolabeled molecules in mice". *J. Nucl. Med.* 46.7, p. 1194.

Beque, D, Nuyts, J, Bormans, G, Suetens, P, and Dupont, P (2003). "Characterization of acquisition geometry of pinhole SPECT". *IEEE Trans. Med. Imaging* 22, pp. 599–612.

Beyer, T, Freudenberg, L. S., Czemin, J, and Townsend, D. W. (2011). "The future of hybrid imaging-part 3: PET/MR, small-animal imaging and beyond". *Insights Imaging* 2.3, pp. 235–246.

Bieback, K, Kern, S, Kocaomer, A, Ferlik, K, and Bugert, P (2008). "Comparing mesenchymal stromal cells from different human tissues: bone marrow, adipose tissue and umbilical cord blood". *Biomed. Mater. Eng.* 18.1, S71–76.

Branderhorst, W., Vastenhouw, B., and Beekman, F. J. (2010). "Pixel-based subsets for rapid multi-pinhole SPECT reconstruction". *Phys. Med. Biol.* 55, pp. 2023–2034.

Bruyant, P. P. (2002). "Analytic and iterative reconstruction algorithms in SPECT". *J. Nucl. Med.* 43.10, pp. 1343–1358.

Cao, Z, Bal, G, Accorsi, R, and Acton, P. D. (2005). "Optimal number of pinholes in multi-pinhole SPECT for mouse brain imaging-a simulation study". *Phys. Med. Biol.* 50.19, pp. 4609–4624.

Carl E Ravin Advanced Imaging Laboratories. `http://deckard.mc.duke.edu/`.

Chen, Y. C., Furenlid, L. R., Wilson, D. W., and Barrett, H. H. (2005). "Small-Animal SPECT". Springer. Chap. Calibration of Scintillation Cameras and Pinhole SPECT Imaging Systems, pp. 195–201.

Chen, Y.-C. (2006). "System calibration and image reconstruction for a new small-animal SPECT system". PhD thesis. The University of Arizona.

Cherry, S (2004). "In vivo molecular and genomic imaging: new challenges for imaging physics". *Phys. Med. Biol.* 49, R13–R48.

Cherry, S. R. (2001). "Fundamentals of positron emission tomography and applications in preclinical drug development". *J. Clin. Pharmacol.* 41.5, pp. 482–491.

Chiewitz, O and Hevesy, G (1935). "Radioactive indicators in the study of phosphorous metabolism in rats". *Nature* 136, pp. 754–755.

Chun, S. Y., Fessler, J. A., and Dewaraja, Y. K. (2013). "Correction for collimator-detector response in SPECT using point spread function template". *IEEE Trans. Med. Imaging* 32, pp. 295–305.

Clarkson, E, Barrett, H. H., and Wilson, D. W. (1998). "Synthesizing a parallel-projection image from pinhole data". *SPIE* 3336, pp. 251–259.

Clarkson, E, Wilson, D. W., and Barrett, H. H. (1999). "The synthetic collimator for 2D and 3D imaging". *SPIE* 3659, pp. 107–117.

Defrise, M and Gullberg, G. T. (2006). "Review: Image Reconstruction". *Phys. Med. Biol.* 51.13, pp. 139–154.

Defrise, M, Vanhove, C, and Nuyts, J (2008). "Perturbative refinement of the geometric calibration in pinhole SPECT". *IEEE Trans. Med. Imaging* 27, pp. 204–214.

DiFilippo, F. P. (2008). "Design and performance of a multi-pinhole collimation device for small animal imaging with clinical SPECT and SPECT-CT scanners". *Phys. Med. Biol.* 53(15), pp. 4185–4201.

Digirolamo, C. M., Stokes, D, Colter, D, Phinney, D. G., Class, R, and Prockop, D. J. (1999). "Propogation and senescence of human marrow stromal cells in culture: a simple colony-forming assay identifies samples with the greatest potential to propogate and differentiate". *Br. J. Hamematol.* 107.2, pp. 275–281.

Dorenbos, P (1995). "Non-proportionality in the scintillation response and the energy resolution obtainable with scintillation crystals". *IEEE Trans. Nucl. Sci.* 42.6, pp. 2190–2202.

Dwyer, R. M., Ryan, J., Havelin, R. J., Morris, J. C., Miller, B. W., Liu, Z., Flavin, R., O'Flatharta, C., Foley, M. J., Barrett, H. H., Murphy, J. M., Barry, F. P., O'Brien, T., and Kerin, M. J. (2011). "Mesenchymal stem cell-mediated delivery of the sodium iodide symporter supports radionuclide imaging and treatment of breast cancer". *STEM CELLS* 29, pp. 1149–1157.

Franc, B. L., Acton, P. D., Mari, C, and Hasegawa, B. H. (2008). "Small-animal SPECT and SPECT/CT: important tools for preclinical investigation". *J. Nucl. Med.* 49.10, pp. 1651–1653.

Funk, T, Despres, P, Barber, W. C., Shah, K. S., and Hasegawa, B. H. (2006). "A multipinhole small animal SPECT system with submillimeter spatial resolution". *Med. Phys.* 33.5, pp. 1259–1268.

Furenlid, L. R., Clarkson, E, Marks, D. G., and Barrett, H. H. (2000). "Spatial pileup considerations for pixellated gamma-ray detectors". *IEEE Trans. Nucl. Sci.* 47.4, pp. 1399–1403.

Furenlid, L. R., Wilson, D. W., Chen, Y, Kim, H, Pietraski, P. J., Crawford, M. J., and Barrett, H. H. (2004). "FastSPECT II: a second-generation high-resolution dynamic SPECT imager". *IEEE Trans. Nucl. Sci.* 51.3, p. 631.

Gambhir, S. S., Berman, D. S., Ziffer, J, Nagler, M, Sandler, M, Patton, J, Hutton, B, Sharir, T, and Haim, S. B. (2009). "A novel high-sensitivity rapid-acquisition single-photon cardiac imaging camera". *J. Nucl. Med.* 50, pp. 635–43.

Garcia, E. V., Faber, T. L., and Esteves, F. P. (2011). "Cardiac dedicated ultrafast SPECT cameras: new designs and clinical implications". *J. Nucl. Med.* 52, pp. 210–7.

Gibson, A and Dehghani, H (2009). "Diffuse optical imaging". *Phil. Trans. R. Soc. A* 367, pp. 3055 –3072.

Goertzen, A. L., Jones, D. W., Seidel, J, Li, K, and Green, M. V. (2005). "First results from the high-resolution mouseSPECT annular scintillation camera". *IEEE Trans. Med. Imaging* 24, pp. 863–867.

Golestani, R, Wu, C, Tio, R. A., Zeebregts, C. J., Petrov, A. D., Beekman, F. J., Dierckx, R. A., Boersma, H. H., and Slart, R. H. (2010). "Small-animal SPECT and SPECT/CT: application in cardiovascular research". *Eur. J. Nucl. Med. Mol. Imaging* 37.9, pp. 1766–1777.

Gordon, R, Bender, R, and Herman, G. T. (1970). "Algebraic reconstruction techniques (ART) for the three-dimensional electron microscopy and X-ray photography". *J. Theor. Biol.* 29, pp. 471–481.

Green, P. J. (1990). "Bayesian reconstructions from emission tomography data using a modified EM algorithm". *IEEE Trans. Med. Imaging*, pp. 84–93.

Gruber, G. J., Choong, W. S., Moses, W. W., Derenzo, S. E., Holland, S. E., Pedrali-Noy, M, Krieger, B, Mandelli, E, Meddeler, G, and Wang, W (2002). "A compact 64-pixel CsI(Tl)/Si PIN photodiode imaging module with IC readout". *IEEE Trans. Nucl. Sci.* 49.1, pp. 147–152.

Gunter, D. L. (1996). "Collimator characteristics and design". Ed. by R. E. Henkin. Mosby. Chap. 8, pp. 107–126.

Habraken, J. B. A., Bruin, K. de, Shehata, M, Booij, J, Bennink, R, Berthe, L. F., Eck, S van, and Sokole, E. B. (2001). "Evaluation of high-resolution pinhole SPECT using a small rotating animal". *J. Nucl. Med.* 42, pp. 1863–1869.

Hasegawa, B. H., Brown, J. K., Blankespoor, S. C., Lang, T. F., Reilly, S. M., Liew, S. C., Tsui, B. M. W., and Ramanathan, C (1993). "Object-specific attenuation correction of SPECT with correlated dual-energy X-ray CT". *IEEE Trans. Nucl. Sci.* 40, pp. 1242–1252.

Have, F van der, Vastenhouw, B, Rentmeester, M, and Beekman, F. J. (2008). "System calibration and statistical image reconstruction for ultra-high resolution stationary pinhole SPECT". *IEEE Trans. Med. Imaging* 27, pp. 960–971.

Have, F van der, Vastenhouw, B, Ramakers, R. M., Branderhorst, W, Krah, J. O., Ji, C, Staelens, S. G., and Beekman, F. J. (2009). "U-SPECT-II: An ultra-high-resolution device for molecular small-animal imaging". *J. Nucl. Med.* 50.4, pp. 599–605.

Hawrysz, D. J. and Sevick-Muraca, E. M. (2000). "Developments towards diagnostic breast cancer imaging using near-infrared optical measurements and uorescent contrast agents." *Neoplasia* 2, pp. 388 –417.

Heanue, J. A., Boser, B. E., and Hasegawa, B. H. (1995). "CMOS detector readout electronics for an emission-transmission medical imaging system". *IEEE Trans. Nucl. Sci.* 42.4, pp. 1133–1138.

Herbert, D. J., Saveliev, V, Belcari, N, D'Ascenzo, N, Del Guerra, A, and Golovin, A (2006). "First results of scintillator readout with silicon photomultiplier". *IEEE Trans. Nucl. Sci.* 53.1, pp. 389–394.

Herman, G. T. (1980). *Image reconstruction from projections*. New York: Academic Press.

Hubbell, J. H. and Seltzer, S. M. (1995). "Tables of x-ray mass attenuation coefficients and mass energy-absorption coefficients 1 keV to 20 MeV for elements Z = 1 to 92 and 48 additional substances of dosimetric interest". *NISTIR Report 5632 (Gaithersburg, MD: NIST)*.

Hudson, H. M. and Larkin, R. S. (1994). "Accelerated image reconstruction using ordered subsets of projection data". *IEEE Trans. Med. Imaging* 13.4, p. 601.

Hutton, B. F., Hudson, H. M., and Beekman, F. J. (1997). "A clinical perspective of accelerated statistical reconstruction". *Eur. J. Nucl. Med.* 24, pp. 797–808.

Jaszczak, R. J., Li, J, Wang, H, Zalutsky, M. R., and Coleman, R. E. (1994). "Pinhole collimation for ultra-high-resolution small-field-of-view SPECT". *Phys. Med. Biol.* 39, pp. 425–437.

Jong, M de and Maina, T (2010). "Of mice and humans: Are they the same? Implications in cancer translational research". *J. Nucl. Med.* 51, pp. 501–504.

Keller, E. L. (1967). "Optimum dimensions of parallel-hole, multi-aperture collimators for gamma-ray cameras". *J. Nucl. Med.* 9.6, pp. 233–235.

Kemp, R. A. de, Epstein, F. H., Catana, C, Tsui, B. M. W., and Ritman, E (2010). "Small-animal molecular imaging methods". *J. Nucl. Med.* 51.5, 18S–32S.

Kim, H, Furenlid, L. R., Crawford, M. J., Wilson, D. W., Barber, H. B., Peterson, T, Hunter, W. C. J., Liu, Z, Woolfenden, J. M., and Barrett, H. H. (2006). "SemiSPECT: A small-animal single-photon emission computed tomography (SPECT) imager based on eight cadmium zinc telluride (CZT) detector arrays". *Med. Phys.* 33, pp. 465–474.

Klopp, A. H., Spaeth, E. L., Dembinski, J. L., Woodward, W. A., Munshi, A, Meyn, R. E., Cox, J. D., Andreeff, M, and Marini, F. C. (2007). "Tumor irradiation increases the recruitment of circulating mesenchymal stem cells into the tumor microenvironment". *Cancer Res.* 67.24, pp. 11687–11695.

Knoll, G. F. (1979). *Radiation detection and measurement*. Wiley and Sons, Inc.

Kubo, H, Hattori, K, Ida, C, Iwaki, S, Kabuki, S, Kurosawa, S, Miuchi, K, Nagayoshi, T, Nishimura, H, Okada, Y, Orito, R, Takada, A, Tanimori, T, Tsuchiya, K, and Ueno, K (2007). "Development of a gamma camera based on an 8x8 array of LaBr3(Ce) scintillator pixels coupled to a 64-channel multi-anode PMT". *Nuclear Science Symposium Conference Record* 6, pp. 4569–4573.

Mahmood, S. T., Erlandsson, K, Cullum, I, and Hutton, B. F. (2010). "The potential for mixed multiplexed and non-multiplexed data to improve the reconstruction quality of a multi-slit-slat collimator SPECT system". *Phys. Med. Biol.* 55, pp. 2247–2268.

Mahmood, S. T., Erlandsson, K, Cullum, I, and Hutton, B. F. (2011). "Experimental results from a prototype slit-slat collimator with mixed multiplexed and non-multiplexed data". *Phys. Med. Biol.* 56, pp. 4311–4331.

Mallard, J. R. and Myers, M. J. (1963). "The performance of a gamma camera for the visualisation of radioactive isotopes in vivo". *Phys. Med. Biol.* 8, pp. 165–182.

Massoud, T. F. and Gambhir, S. S. (2003). "Molecular imaging in living subjects: seeing fundamental biological processes in a new light". *Genes & Development* 17, pp. 545–580.

Meikle, S. R., Kench, P, Weisenberger, A. G., Wojcik, R, Smith, M. F., Majewski, S, Eberl, S, Fulton, R. R., Rosenfeld, A. B., and Fulham, M. J. (2002). "A prototype coded aperture detector for small animal SPECT". *IEEE Trans. Nucl. Sci.* 49.5, pp. 2167–2171.

Meikle, S. R., Kench, P, Kassiou, M, and Banati, R. B. (2005). "Small animal SPECT and its place in the matrix of molecular imaging technologies". *Phys. Med. Biol.* 50, R45–R61.

Meng, L. J., Fu, G, Roy, E. J., Suppe, B, and Chen, C. T. (2009). "An Ultra-high resolution SPECT system for I-125 mouse brain imaging studies". *Nucl. Instrum. Methods Phys. Res. A* 600.1, pp. 498–505.

Miller, B. W. (2011). "High-resolution gamma-ray imaging with columnar scintillators and CCD/CMOS sensors, and FASTSPECT III: A third-generation stationary SPECT imagers". PhD thesis. College of Optical Science, University of Arizona.

Miller, B. W., Barber, H. B., Barrett, H. H., Wilson, D. W., and Chen, L (2006). "A low-cost approach to high-resolution, single-photon imaging using columnar scintillators and image intensifiers". *IEEE Nuclear Science Symposium Conference Record*. Vol. 6, pp. 3540–3545.

Miller, B. W., Barber, H. B., Barrett, H. H., Chen, L, and Taylor, S. J. (2007). "Photon-counting gamma camera based on columnar CsI(Tl) optically coupled to a back-illuminated CCD". *Proc. SPIE* 6510, 65100N.

Miller, B. W., Barrett, H. H., Furenlid, L. R., Barber, H. B., and Hunter, R. J. (2008). "Recent advances in BazookaSPECT: Real-time data processing and the development of a gamma-ray microscope". *Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 591.1, pp. 272–275.

Miller, B. W., Barber, H. B., Furenlid, L. R., Moore, S. K., and Barrett, H. H. (2009). "Progress in BazookaSPECT". *Proc. SPIE* 7450, p. 74500C.

Miller, B. W., Moore, J. W., Barrett, H. H., Frye, T, Adler, S, Sery, J, and Furenlid, L. R. (2011). "3D printing in X-ray and Gamma-Ray Imaging: A novel method for fabricating high-density imaging apertures". *Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 659, pp. 262–268.

Miller, B. W., Barber, H. B., Barrett, H. H., Liu, Z, Nagarkar, V. V., and Furenlid, L. R. (2012). "Progress in BazookaSPECT: high-resolution dynamic scintigraphy with large-area imagers". *SPIE Optical Engineering + Applications*. Vol. 8508. International Society for Optics and Photonics, 85080F.

Miller, B. W., Barber, H. B., Barrett, H. H., Liu, Z, Nagarkar, V. V., and Furenlid, L. R. (2012a). "Progress in BazookaSPECT: high-resolution dynamic scintigraphy with large-area imagers". *SPIE Optical Engineering + Applications* 8508, 85080F.

Miller, B. W., Van Holen, R., Barrett, H. H., and Furenlid, L. R. (2012b). "A system calibration and fast iterative reconstruction method for next-generation SPECT imagers". *IEEE Trans. Nucl. Sci.* 59, pp. 1990–1996.

Mitchell, G. S. and Cherry, S. R. (2009). "A high sensitivity small animal SPECT system". *Phys. Med. Biol.* 54.5, pp. 1291–1305.

Mok, G, Wang, Y, and Tsui, B. M. W. (2009). "Quantification of the multiplexing effects in multi-pinhole small animal SPECT: a simulation study". *IEEE Trans. Nucl. Sci.* 56.5, pp. 2636–2643.

Mok, S. P., Wang, Y, and Tsui, B. M. W. (2005). "Design of a novel pinhole collimator system for SPECT imaging of small animals with different sizes". *Proc. IEEE Nuclear Science Symp. Conf. Record* 5, pp. 2649–2652.

Nagarkar, V. V., Gupta, T. K., Miller, S. R., Klugerman, Y, Squillante, M. R., and Entine, G (1998). "Structured CsI(Tl) scintillators for x-ray imaging applications". *IEEE Trans. Nucl. Sci.* 45, pp. 492–496.

Nuyts, J., Vunckx, K., Defrise, M., and Vanhove, C. (2009). "Small animal imaging with multi-pinhole SPECT". *Methods* 48.2, pp. 83–91.

Objet Geometries Ltd. `http://www.objet.com/`.

Peters, L. L., Robledo, R. F., Bult, C. J., Churchill, G. A., Paigen, B. J., and Svenson, K. L. (2007). "The mouse as a model for human biology: a resource guide for complex trait analysis". *Nat. Rev. Genet.* 8, pp. 58–69.

Peterson, T. E. and Furenlid, L. R. (2011). "SPECT detectors: The Anger Camera and beyond". *Phys. Med. Biol.* 56, R145–R182.

Peterson, T. E., Shokouhi, S, Furenlid, L. R., and Wilson, D. W. (2009). "Multi-pinhole SPECT imaging with silicon strip detectors". *IEEE Trans. Nucl. Sci.* 56, pp. 646–652.

Phelps, M. E., Hoffman, E. J., Mullani, N. A., and Ter-Pogossian, M. M. (1975). "Application of annihilation coincidence detection to transaxial reconstruction tomography". *J. Nucl. Med.* 16.3, pp. 210–224.

Photek. `http://www.photek.com/`.

Point Grey Research, Inc. URL: `http://www.ptgrey.com/`.

Proxivision, Inc. URL: `http://www.proxivision.de`.

Qi, J and Leahy, R. M. (2006). "Topical Review: Iterative reconstruction techniques in emission computed tomography". *Phys. Med. Biol.* 51, R541–R578.

Qian, J, Bradley, E. L., Majewski, S, Popov, V, Saha, M. S., Smith, M. F., Weisenberger, A. G., and Welsh, R. E. (2008). "A small-animal imaging system capable of multipinhole circular/helical SPECT and parallel-hole SPECT". *Nucl. Instrum. Methods Phys. Res. A* 594.1, pp. 102–110.

Recchia, F. A. and Lionetti, V (2007). "Animal models of dilated cardiomyopathy for translational research". *Vet. Res. Commum.* 31.1, pp. 35–41.

Rentmeester, M. C. M., Have, F van der, and Beekman, F. J. (2007). "Optimizing multi-pinhole SPECT geometries using an analytical model". *Phys. Med. Biol.* 52.9, p. 2567.

Rogulski, M. M., Barber, H. B., Barrett, H. H., Shoemaker, R. L., and Woolfenden, J. M. (1993). "Ultra-high resolution brain SPECT imaging: simulation results". *IEEE Trans. Nucl. Sci.* 40, pp. 1123–1129.

Roper Scientific, I. (2000). "Fiberoptic tapers in high-resolution scientific imaging - what you need to know when selecting a fiberoptically coupled ccd camera". *Tech. report.*

Rowe, R. K., Aarsvold, J. N., Barrett, H. H., Chen, J. C., Klein, W. P., Moore, B. A., Pang, I. W., Patton, D. D., and White, T. A. (1993). "A stationary hemi-spherical SPECT imager for three-dimensional brain imaging". *Journal of Nuclear Medicine* 34, p. 474.

Rowland, D. J. and Cherry, S. R. (2008). "Small-animal preclinical nuclear medicine instrumentation and methodology". *Semin. Nucl. Med.* 38.3, pp. 209–222.

Schramm, N. U., Ebel, G, Engeland, U, Schurrat, T, Behe, M, and Behr, T. M. (2003). "High-resolution SPECT using multipinhole collimation". *IEEE Trans. Nucl. Sci.* 50.3, p. 315.

Shah, K. S., Farrell, R, Grazioso, R, Myers, R, and Cirignano, L (2001). "Large-area APDs and monlithic APD arrays". *IEEE Trans. Nucl. Sci.* 48.6, pp. 2352–2356.

Shepp, L. A. and Vardi, Y (1982). "Maximum likelihood reconstruction for emission tomography". *IEEE Trans. Med. Imaging* 1.2, pp. 113–122.

Shokouhi, S, Metzler, S. D., Wilson, D. W., and Peterson, T. E. (2009). "Multi-pinhole collimator design for small-object imaging with SiliSPECT: a high-resolution SPECT". *Phys. Med. Biol.* 54.2, pp. 207–225.

Smith, M. F., Majewski, S, and Weisenberger, A. G. (2003). "Optimizing pinhole and parallel hole collimation for scintimammography with compact pixellated detectors". *IEEE Trans. Nucl. Sci.* 50.3, pp. 321–326.

Snider, P and Conway, S. J. (2011). "Probing human cardiovascular congenital disease using transgenic mouse models". *Prog. Mol. Biol. Transl. Sci.* 100, pp. 83–110.

Spaeth, E, Klopp, A, Dembinski, J, Andreeff, M, and Marini, F (2008). "Inflammation and tumor microenvironments: defining the migratory itinerary of mesenchymal stem cells". *Gene Therapy* 15.10, pp. 730–738.

Spanoudaki, V. C. and Ziegler, S. I. (2008). "PET & SPECT instrumentation". *Handb. Exp. Pharmacol.* 185 Pt 1, pp. 53–74.

Suzuki, K, Horiba, I, and Sugie, N (2000). "Fast connected-component labeling based on sequential local operations in the course of forward raster scan followed by backward raster scan". *Pattern Recognition, Proceedings, 15th International Conference* 2, pp. 434–437.

Thor Labs. `http://www.thorlabs.com/`.

Townsend, D. W. (2008). "Multimodality imaging of structure and function". *Phys. Med. Biol.* 53.4, R1–R39.

Tsui, B. M. W., Zhao, X, Frey, E. C., and McCartney, W. H. (1994). "Quantitative single-photon emission computed tomography: basics and clinical considerations". *Semin. Nucl. Med.* 24.1, pp. 38–65.

Tungsten Heavy Powder, Inc. `http://www.tungstenheavypowder.com/`.

Vaissier, P. E. B., Goorden, M. C., Vastenhouw, B, Have, F van der, Ramakers, R. M., and Beekman, F. J. (2012). "Fast Spiral SPECT with Stationary γ-Cameras and Focusing Pinholes". *J. Nucl. Med.* 53, pp. 1292 –1299.

Vastenhouw, B and Beekman, F. J. (2007). "Submillimeter total-body murine imaging with U-SPECT-I". *J. Nucl. Med.* 48.3, pp. 487–493.

Velmex, Inc. `http://www.Velmex.com/`.

Vogel, R. A., Kirch, D, LeFree, M, and Steele, P (1978). "A new method of multiplanar emission tomography using a seven pimhole collimator and an Anger scintillation camera". *J. Nucl. Med.* 19, pp. 648–654.

Vunckx, K, Suetens, P, and Nuyts, J (2008a). "Effect of overlapping projections on reconstruction image quality in multipinhole SPECT". *IEEE Trans. Med. Imaging* 27.7, p. 972.

Vunckx, K, Beque, D, Defrise, M, and Nuyts, J (2008b). "Single and multipinhole collimator design evaluation method for small animal SPECT". *IEEE Trans. Med. Imaging* 27, pp. 36–46.

Wagner, R. F., Brown, D. G., and Metz, C. E. (1981). "On the multiplex advantage of coded source aperture photon imaging". *Proc. SPIE* 314, pp. 72–76.

Wang, J and Blackburn, T. J. (2000). "The AAPM/RSMA Physics tutorial for residents". *Radiographics* 20, pp. 1471–1477.

Watanabe, M, Shimoi, H, Liu, H, and Yoshizawa, Y (1999). "A new compact position-sensitive PMT for scintillation detectors". *IEEE Trans. Nucl. Sci.* 46.3, pp. 354–358.

Weber, D. A. and Ivanovic, M (1999). "Ultra-high-resolution imaging of small animals: implications for preclinical and research studies". *J. Nucl. Cardiol.* 6.3, pp. 332–244.

Weissleder, R (2002). "Scaling down imaging: Molecular mapping of cancer in mice". *Nature Reviews Cancer* 2, pp. 1 –8.

Welch, M. J. and Redvanly, C. S., eds. (2005). *Handbook of Radiopharmaceuticals: Radiochemistry and Applications*. New York: Wiley.

Wilson, D. W., Barrett, H. H., and Clarkson, E. W. (2000). "Reconstruction of two- and three-dimensional images from synthetic-collimator data". *IEEE Trans. Med. Imaging* 19, pp. 412–422.

Wiza, J. L. (1979). "Microchannel plate detectors". *Nucl. Instrum. Methods 162* no. 1-3, pp. 587–601.

Xu, M and Wang, L. V. (2006). "Photoacoustic imaging in biomedicine". *Rev. Sci. Instrum.* 77, p. 041101.

Zeniya, T, Watabe, H, Aoi, T, Kim, K. M., Teramoto, N, Takeno, T, Ohta, Y, Hayashi, T, Mashino, H, Ota, Y, Yamamoto, S, and Iida, H (2006). "Use of a compact pixellated gamma camera for small animal pinhole SPECT imaging". *Ann. Nucl. Med.* 20.6, pp. 409–416.