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

Havelin, Ronan

2013-11-01

http://hdl.handle.net/10379/3797

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

OÉ Gaillimh
NUI Galway

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**


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
## Contents

3 System Design  
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  
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  
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  
6.1 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 128  
6.2 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132

Appendix  
A CUDA Code  

v
# Contents

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

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>A flow chart outlining the organization of this thesis.</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>The molecular imaging matrix.</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Pinhole camera geometry.</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Simple multi-pinhole camera geometry.</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>Parallel-hole collimator.</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>POSEM subset distribution.</td>
<td>33</td>
</tr>
<tr>
<td>2.6</td>
<td>CUDA Implementation of MLEM.</td>
<td>36</td>
</tr>
<tr>
<td>3.1</td>
<td>BazookaSPECT detector.</td>
<td>41</td>
</tr>
<tr>
<td>3.2</td>
<td>Columnar CsI scintillator.</td>
<td>43</td>
</tr>
<tr>
<td>3.3</td>
<td>Microchannel plate.</td>
<td>43</td>
</tr>
<tr>
<td>3.4</td>
<td>XX1332 image intensifier.</td>
<td>45</td>
</tr>
<tr>
<td>3.5</td>
<td>CCD component of the BazookaSPECT detector.</td>
<td>46</td>
</tr>
<tr>
<td>3.6</td>
<td>Photon cluster extraction.</td>
<td>48</td>
</tr>
<tr>
<td>3.7</td>
<td>Frame parsing.</td>
<td>49</td>
</tr>
<tr>
<td>3.8</td>
<td>Fibre-optic taper.</td>
<td>50</td>
</tr>
<tr>
<td>3.9</td>
<td>Light loss from fibre-optic taper.</td>
<td>51</td>
</tr>
<tr>
<td>3.10</td>
<td>Large-area BazookaSPECT detector.</td>
<td>51</td>
</tr>
<tr>
<td>3.11</td>
<td>Objet Geometries Ltd. Connex350™ printer.</td>
<td>52</td>
</tr>
<tr>
<td>3.12</td>
<td>Collimator casing.</td>
<td>55</td>
</tr>
<tr>
<td>3.13</td>
<td>Focused multi-pinhole collimator.</td>
<td>57</td>
</tr>
<tr>
<td>3.14</td>
<td>Geometry of an off-centre focused pinhole aperture.</td>
<td>58</td>
</tr>
<tr>
<td>3.15</td>
<td>SolidWorks rendering of the collimator.</td>
<td>59</td>
</tr>
<tr>
<td>3.16</td>
<td>Detector casing.</td>
<td>60</td>
</tr>
<tr>
<td>3.17</td>
<td>Detector support structure.</td>
<td>60</td>
</tr>
<tr>
<td>3.18</td>
<td>Collimator holder.</td>
<td>61</td>
</tr>
<tr>
<td>3.19</td>
<td>SolidWorks rendering of the SyntheticSPECT assembly.</td>
<td>61</td>
</tr>
<tr>
<td>3.20</td>
<td>Image of the collimator and phantom.</td>
<td>62</td>
</tr>
<tr>
<td>3.21</td>
<td>Image of the SyntheticSPECT system.</td>
<td>63</td>
</tr>
<tr>
<td>3.22</td>
<td>The LabVIEW control interface for SyntheticSPECT.</td>
<td>64</td>
</tr>
<tr>
<td>3.23</td>
<td>LabVIEW interface workflow.</td>
<td>65</td>
</tr>
<tr>
<td>4.1</td>
<td>An integrated projection image.</td>
<td>68</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.2</td>
<td>Total counts recorded for the first system configuration.</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>Total counts recorded for the second system configuration.</td>
<td>70</td>
</tr>
<tr>
<td>4.4</td>
<td>Total counts recorded for the third system configuration.</td>
<td>70</td>
</tr>
<tr>
<td>4.5</td>
<td>Total counts recorded for the fourth system configuration.</td>
<td>71</td>
</tr>
<tr>
<td>4.6</td>
<td>Total counts recorded for the fifth system configuration.</td>
<td>71</td>
</tr>
<tr>
<td>4.7</td>
<td>Total counts recorded for the sixth system configuration.</td>
<td>71</td>
</tr>
<tr>
<td>4.8</td>
<td>Total counts recorded through the centre of the field of view.</td>
<td>72</td>
</tr>
<tr>
<td>4.9</td>
<td>Projection image from a point source in the corner of the field of view.</td>
<td>73</td>
</tr>
<tr>
<td>4.10</td>
<td>Segmented pinhole projections.</td>
<td>74</td>
</tr>
<tr>
<td>4.11</td>
<td>Single pinhole projection.</td>
<td>74</td>
</tr>
<tr>
<td>4.12</td>
<td>2D Gaussian fit is made to every pinhole projection.</td>
<td>76</td>
</tr>
<tr>
<td>4.13</td>
<td>2D Gaussian fit to a pinhole projection.</td>
<td>77</td>
</tr>
<tr>
<td>4.14</td>
<td>Counts required for Gaussian fitting.</td>
<td>78</td>
</tr>
<tr>
<td>4.15</td>
<td>Projection of the centroid of the 2D Gaussian fits.</td>
<td>81</td>
</tr>
<tr>
<td>4.16</td>
<td>Gaussian coefficients of a central pinhole.</td>
<td>82</td>
</tr>
<tr>
<td>4.17</td>
<td>Gaussian coefficients of a peripheral pinhole.</td>
<td>83</td>
</tr>
<tr>
<td>4.18</td>
<td>Effect of splines on the interpolation.</td>
<td>85</td>
</tr>
<tr>
<td>4.19</td>
<td>Multiplexing pinhole projections.</td>
<td>87</td>
</tr>
<tr>
<td>4.20</td>
<td>Percentage of pinhole projections.</td>
<td>88</td>
</tr>
<tr>
<td>4.21</td>
<td>Simulated low-magnification projection data.</td>
<td>90</td>
</tr>
<tr>
<td>4.22</td>
<td>Simulated high-magnification projection data.</td>
<td>91</td>
</tr>
<tr>
<td>4.23</td>
<td>Reconstruction of the simulated data.</td>
<td>91</td>
</tr>
<tr>
<td>4.24</td>
<td>Signal-to-noise ratio as a function of contrast.</td>
<td>92</td>
</tr>
<tr>
<td>4.25</td>
<td>Simulated phantom.</td>
<td>92</td>
</tr>
<tr>
<td>4.26</td>
<td>Simulation results.</td>
<td>95</td>
</tr>
<tr>
<td>4.27</td>
<td>Reconstruction of the simulated hot-rod phantom using only multiplexed data</td>
<td>96</td>
</tr>
<tr>
<td>4.28</td>
<td>Reconstruction of a simulated hot-rod phantom using non-multiplexed and multiplexed data.</td>
<td>97</td>
</tr>
<tr>
<td>4.29</td>
<td>Fourier crosstalk matrix: Normalized MTF.</td>
<td>100</td>
</tr>
<tr>
<td>4.30</td>
<td>Energy resolution.</td>
<td>101</td>
</tr>
<tr>
<td>5.1</td>
<td>Timeline of <em>in vivo</em> mesenchymal stem cell-sodium iodide symporter imaging and therapy experiments.</td>
<td>107</td>
</tr>
<tr>
<td>5.2</td>
<td>BazookaSPECT images of mice.</td>
<td>109</td>
</tr>
<tr>
<td>5.3</td>
<td><em>In vivo</em> radioiodine therapy of breast tumour xenografts.</td>
<td>111</td>
</tr>
<tr>
<td>5.4</td>
<td>Hot-rod phantom schematic.</td>
<td>112</td>
</tr>
<tr>
<td>5.5</td>
<td>Experimental projections of the hot-rod phantom.</td>
<td>114</td>
</tr>
<tr>
<td>5.6</td>
<td>(a) Slices through the reconstruction of the hot-rod phantom.</td>
<td>115</td>
</tr>
<tr>
<td>5.7</td>
<td>(b) Slices through the reconstruction of the hot-rod phantom.</td>
<td>116</td>
</tr>
<tr>
<td>5.8</td>
<td>Line profiles of the reconstructed rods of the hot-rod phantom.</td>
<td>117</td>
</tr>
<tr>
<td>5.9</td>
<td>Reconstruction of the hot-rod phantom with MLEM iterations.</td>
<td>118</td>
</tr>
<tr>
<td>5.10</td>
<td>POSEM reconstruction of the hot-rod phantom using one system configuration.</td>
<td>119</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.11</td>
<td>POSEM reconstruction of the hot-rod phantom using all system configurations.</td>
<td>120</td>
</tr>
<tr>
<td>5.12</td>
<td>Schematic of mouse translated through FOV.</td>
<td>122</td>
</tr>
<tr>
<td>5.13</td>
<td>Projections (1st section of animal) of a mouse injected with $^{99m}$Tc-MDP.</td>
<td>123</td>
</tr>
<tr>
<td>5.14</td>
<td>Projections (2nd section of animal) of a mouse injected with $^{99m}$Tc-MDP.</td>
<td>123</td>
</tr>
<tr>
<td>5.15</td>
<td>Projections (3rd section of animal) of a mouse injected with $^{99m}$Tc-MDP.</td>
<td>124</td>
</tr>
<tr>
<td>5.16</td>
<td>Projections (4th section of animal) of a mouse injected with $^{99m}$Tc-MDP.</td>
<td>124</td>
</tr>
<tr>
<td>5.17</td>
<td>Projections (5th section of animal) of a mouse injected with $^{99m}$Tc-MDP.</td>
<td>125</td>
</tr>
<tr>
<td>5.18</td>
<td>Full-body reconstruction of a mouse injected with $^{99m}$Tc-MDP.</td>
<td>126</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Common PET and SPECT nuclides used for imaging.</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Physical and scintillation properties of scintillators.</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Small-animal SPECT systems.</td>
<td>13</td>
</tr>
<tr>
<td>2.4</td>
<td>Graphics card specifications.</td>
<td>35</td>
</tr>
<tr>
<td>2.5</td>
<td>CUDA vs MATLAB™</td>
<td>38</td>
</tr>
<tr>
<td>3.1</td>
<td>Phosphor screens for use with BazookaSPECT.</td>
<td>44</td>
</tr>
<tr>
<td>3.2</td>
<td>Lens specifications.</td>
<td>45</td>
</tr>
<tr>
<td>3.3</td>
<td>CCD Specifications.</td>
<td>46</td>
</tr>
<tr>
<td>3.4</td>
<td>3D printing materials data sheet.</td>
<td>53</td>
</tr>
<tr>
<td>4.1</td>
<td>System configurations.</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Storage requirements of the system matrix.</td>
<td>84</td>
</tr>
<tr>
<td>4.3</td>
<td>Validation of the coefficient interpolation.</td>
<td>86</td>
</tr>
<tr>
<td>4.4</td>
<td>Simulated test cases.</td>
<td>93</td>
</tr>
<tr>
<td>5.1</td>
<td>$NIS$ and $MRPL19$ expression in animal tissues.</td>
<td>110</td>
</tr>
<tr>
<td>5.2</td>
<td>Acquisition settings for phantom study.</td>
<td>113</td>
</tr>
<tr>
<td>5.3</td>
<td>MLEM and POSEM reconstruction time.</td>
<td>120</td>
</tr>
</tbody>
</table>
Nomenclature

\( \alpha \) 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.

\( f \) The discrete distribution of radioactivity in object space.

\( g \) The discrete measurements in image space.

\( H \) The discrete system matrix that maps the object vector to an image.

\( f \) The continuous distribution of radioactivity in object space.

\( \Gamma \) Ideal tube integrals of a parallel-projection image.

\( \hat{\vartheta} \) An estimate of the parameters of interest.

\( \lambda \) The decay constant.

\( \mu \) The linear attenuation of the collimator material.

\( \nu_{par} \) The geometric sensitivity of a parallel-hole collimator.

\( \phi \) A basis function.

\( \Psi \) Fourier-space equivalent of \( H \).

\( \rho \) 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 \( j^{th} \) 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, $H$, that maps the object vector to an image is discussed in this section. Section 2.6 outlines how $H$ 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.
2. Background and Methods

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 \textit{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 \textit{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
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).

<table>
<thead>
<tr>
<th>Radionuclide</th>
<th>Half-life</th>
<th>Principle γ-rays (keV)</th>
<th>Modality</th>
</tr>
</thead>
<tbody>
<tr>
<td>$^{11}$C</td>
<td>20.39 (min)</td>
<td>511</td>
<td>PET</td>
</tr>
<tr>
<td>$^{13}$N</td>
<td>9.97 (min)</td>
<td>511</td>
<td>PET</td>
</tr>
<tr>
<td>$^{15}$O</td>
<td>2.04 (min)</td>
<td>511</td>
<td>PET</td>
</tr>
<tr>
<td>$^{18}$F</td>
<td>109.77 (min)</td>
<td>511</td>
<td>PET</td>
</tr>
<tr>
<td>$^{99m}$Tc</td>
<td>6.02 (h)</td>
<td>140</td>
<td>SPECT</td>
</tr>
<tr>
<td>$^{111}$In</td>
<td>2.8 (days)</td>
<td>171, 245</td>
<td>SPECT</td>
</tr>
<tr>
<td>$^{129}$I</td>
<td>13.22 (h)</td>
<td>159</td>
<td>SPECT</td>
</tr>
<tr>
<td>$^{131}$I</td>
<td>8.03 (days)</td>
<td>364</td>
<td>SPECT</td>
</tr>
</tbody>
</table>

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 ($\rho$) 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 in 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>
<thead>
<tr>
<th>Scintillator</th>
<th>$\rho$ (g/cm$^3$)</th>
<th>Effective Z</th>
<th>Decay Time (ns)</th>
<th>Photon Yield / keV</th>
<th>Hygroscopic</th>
<th>Peak Emission (nm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaI(Tl)</td>
<td>3.67</td>
<td>50.6</td>
<td>230</td>
<td>38</td>
<td>Y</td>
<td>415</td>
</tr>
<tr>
<td>CsI(Tl)</td>
<td>4.51</td>
<td>54.1</td>
<td>1000</td>
<td>52</td>
<td>N</td>
<td>540</td>
</tr>
<tr>
<td>BGO</td>
<td>7.13</td>
<td>74.2</td>
<td>350</td>
<td>8</td>
<td>N</td>
<td>480</td>
</tr>
<tr>
<td>LSO</td>
<td>7.4</td>
<td>65.5</td>
<td>42</td>
<td>20-30</td>
<td>N</td>
<td>420</td>
</tr>
<tr>
<td>LaBr$_3$</td>
<td>5.29</td>
<td>46</td>
<td>26</td>
<td>63</td>
<td>Y</td>
<td>380</td>
</tr>
</tbody>
</table>

### 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; Nuysts 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; Jaszcak 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.
2. Background and Methods

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

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, $R_0$, is given by

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

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 $d_{OP}$.](image-url)
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

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

(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

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

(2.3)

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

The sensitivity of an ideal pinhole is given by

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

(2.4)

where $\theta$ 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

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

(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

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

(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

![Multi-Pinhole Imaging Diagram]

**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
2. Background and Methods

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 established (Gunter, 1996; Keller, 1967). The geometric sensitivity of a 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.

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.
collimator can be expressed as

\[ \nu_{par} = \frac{Kw^2}{a_e(w + t)}, \]  

(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

\[ a_e = a - \frac{2}{\mu}, \]  

(2.8)

where \( a \) is the physical hole length, and \( \mu \) 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

\[ R_g = \frac{w(a_e + d_1 + d_2)}{a_e}, \]  

(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

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

(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 tomo-
graphic 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 pin-
holes 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 differ-
ent 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. Mag-
nification 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 recon-
struction. 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 \(m^{th}\) detector element records a measurement given by

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

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
\[ \Gamma_j = \int_S d^3r f(r) t_j(r), \quad (2.12) \]
where \( t_j(r) \) is the tube function, defined as unity within the region of the \( j^{th} \) 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
\[ t_j(r) = \sum_{m=1}^{M} B_{jm} h_m(r), \quad (2.13) \]
then the values of \( \Gamma_j \) should be recoverable. Assuming that the tube functions can be synthesized, the estimate for \( \Gamma_j \) is
\[ \hat{\Gamma}_j = \sum_{m=1}^{M} B_{jm} g_m. \quad (2.14) \]
The difference between this estimator’s expected value and the true value of the parameter is
\[
\langle \hat{\Gamma}_j(g) - \Gamma_j(f) \rangle = \sum_{m=1}^{M} B_{jm} \langle g_m \rangle - \Gamma_j(f)
\]
\[
= \left[ \int_S d^3r B_{jm} h_m(r) f(r) \right] - \Gamma_j(f)
\]
\[
= \left[ \int_S d^3r t_j(r) f(r) \right] - \Gamma_j(f)
\]
\[ = 0. \quad (2.15) \]
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 \( n^{th} \) voxel, and \( v_n(r) = 0 \) elsewhere. The corresponding parameters of the distribution are
\[ \psi_n(f) = \int_S d^3r v_n(r) f(r), \quad (2.16) \]
and, similarly to before, these parameters are estimable if the voxel functions can be synthesized from the sensitivity functions

\[ \psi_n(f) = \sum_{m=1}^{M} A_{jm} h_m(r), \]  

(2.17)

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

\[ \hat{\psi}_n(g) = \sum_{m=1}^{M} A_{jm} g_m. \]  

(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)

\[ g_m = \int_S d^3 r f(r) h_m(r) + n_m, \]  

(2.19)

where \( m = 1, \ldots, M \), and \( h_m(r) \) is the contribution to the \( m^{th} \) measurement from the object at point \( r \), and \( n \) is the noise. The measurements \( g_m \) form the elements of a vector \( g \) 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 \( f \) by voxelizing the object. The imaging equation is then written as

\[ g = Hf + n, \]  

(2.20)

where \( H \) is the discrete system matrix that maps the object vector to an image, and the noise, \( n \), is an M-dimensional vector in data space. Iterative reconstruction algorithms require an accurate measurement of the system matrix for image
2. Background and Methods

For pinhole SPECT, $\mathbf{H}$ should contain information about the system geometry, attenuation and scattering of radiation in the object, radiometric factors, pinhole penetration, and detector blur. $\mathbf{H}$ 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

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

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

$$H_{mn} = \int d^3r h_m(r) \phi_n(r), \quad (2.22)$$

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. $\mathbf{H}$ 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)

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

Or, in component form,

$$g_m = \sum_k \Psi_{mk} F_k + n_m, \quad (2.24)$$

where

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

and the Fourier coefficients are given by

$$F_k = \frac{1}{L^2} \int_{-\infty}^{\infty} d^3r f(r) \exp(-2\pi i p_k \cdot r) s(r), \quad (2.26)$$
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, $\Psi$, is equivalent to the operator $H$. 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 $\Psi_{mk}$ 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

$$\beta_{kk'} = \sum_{m=1}^{M} \Psi_{mk}^* \Psi_{mk'}, \quad (2.27)$$

where $k \neq k'$, $\beta_{kk'}$ is a measure of the degree of aliasing or crosstalk between the two different frequencies $\rho_k$ 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

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

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

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

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 $\beta_{kk}$ 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, $H$, 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.
2. Background and Methods

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 \( pr(g|\theta) \) (Barrett and Myers, 2004). \( \theta \) is a \( P \times 1 \) parameter vector, and \( g \) is denoted as a \( M \times 1 \) vector describing random data, where \( g \) is an individual sample taken from \( pr(g|\theta) \). The likelihood function of the parameters to be estimated given a measured data vector is defined as

\[
L(\theta|g) = pr(g|\theta). \tag{2.30}
\]

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

\[
\hat{\theta}_{ML} \equiv \arg \max_{\theta} pr(g|\theta). \tag{2.31}
\]

The \( \arg\max \) function returns the \( \theta \) argument at which \( pr(g|\theta) \) is maximized. This can be written equivalently as

\[
\hat{\theta}_{ML} = \arg \max_{\theta} \ln[pr(g|\theta)]. \tag{2.32}
\]
2. Background and Methods

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

\[
Pr(g|\vartheta) = \prod_{m=1}^{M} \exp(-\bar{g}_m) (\bar{g}_m)^{g_m} / g_m!,
\]

where the mean data vector is

\[
\bar{g} = H\vartheta.
\]

The log-likelihood function is

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

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:

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

The second derivatives are

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

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

\[
\frac{1}{\sum_{m=1}^{M} H_{mn}} \sum_{m=1}^{M} \frac{g_m}{(H\vartheta)_m} H_{mn} = 1.
\]
Both sides are now multiplied by $\vartheta_n$, which we assume is the radiotracer distribution that is the best estimate of the true radiotracer distribution within the object, yielding

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

(2.39)

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

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

(2.40)

where $k$ denotes the $k^{th}$ 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 $\vartheta$ 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

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

(2.41)

where $\hat{f}^{(k)}_n$ is the estimated activity in voxel $n$ after the $k^{th}$ iteration, $h_{mn}$ is an element of $H$ 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 $H$ matrix. The measured data, $g$, 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

\[ \hat{f}_n^{(i+1)}(k) = \frac{\hat{f}_n^{(i)}(k)}{\sum_{m'\in S_i} h_{mn'}} \sum_{m\in S_i} g_m \sum_{n'=1}^N h_{mn'} \hat{f}_{n'}^{(i)}(k) \]  

(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-

![Detector](image)

**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.

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

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.

<table>
<thead>
<tr>
<th>Device Name</th>
<th>GeForce GTX 580</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Clock</td>
<td>772 MHz</td>
</tr>
<tr>
<td>Memory Clock</td>
<td>4008 MHz</td>
</tr>
<tr>
<td>Shader Clock</td>
<td>1544 MHz</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>512</td>
</tr>
<tr>
<td>Global Memory</td>
<td>3072 MB DDR5</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>48 kB</td>
</tr>
<tr>
<td>No. of Multiprocessors</td>
<td>16</td>
</tr>
<tr>
<td>Maximum Threads per Block</td>
<td>1024</td>
</tr>
</tbody>
</table>

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 $3\sigma$ 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™. 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 \( 3\sigma \) 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\textsuperscript{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\textsuperscript{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 $3\sigma$ radius of the centroid of each pinhole fit.
- Integrate the detector response from each voxel location.

The MATLAB\textsuperscript{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\textsuperscript{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>
<thead>
<tr>
<th>Implementation</th>
<th># voxels in object space</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB\textsuperscript{TM}</td>
<td>$61 \times 61 \times 61$</td>
<td>387</td>
</tr>
<tr>
<td>MATLAB\textsuperscript{TM}</td>
<td>$97 \times 97 \times 97$</td>
<td>1608</td>
</tr>
<tr>
<td>CUDA</td>
<td>$61 \times 61 \times 61$</td>
<td>&lt; 0.01</td>
</tr>
<tr>
<td>CUDA</td>
<td>$97 \times 97 \times 97$</td>
<td>&lt; 0.01</td>
</tr>
</tbody>
</table>

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).
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
3. System Design

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.
3. System Design

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 (Proxivation, Inc. Wang and Blackburn, 2000).

<table>
<thead>
<tr>
<th>Type</th>
<th>Composition</th>
<th>Light emission</th>
<th>Decay time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Range (nm)</td>
<td>Max (nm)</td>
</tr>
<tr>
<td>P20</td>
<td>ZnCdS:Ag</td>
<td>460 - 660</td>
<td>530</td>
</tr>
<tr>
<td>P43</td>
<td>Gd₂O₂S:Tb</td>
<td>360 - 680</td>
<td>545</td>
</tr>
<tr>
<td>P46</td>
<td>Y₅Al₃O₁₂:Ce</td>
<td>490 - 620</td>
<td>530</td>
</tr>
<tr>
<td>P47</td>
<td>Y₂SiO₅:Ce,Tb</td>
<td>370 - 480</td>
<td>400</td>
</tr>
</tbody>
</table>

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.
3. System Design

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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lens</td>
<td>Fujinon DF6HA-1B</td>
</tr>
<tr>
<td>Focal length</td>
<td>6 mm</td>
</tr>
<tr>
<td>Iris range</td>
<td>F1.2 - F16</td>
</tr>
<tr>
<td>Focus &amp; iris operation</td>
<td>Manual</td>
</tr>
<tr>
<td>Focusing range</td>
<td>$\infty \sim 0.1$ m</td>
</tr>
<tr>
<td>Back focal distance</td>
<td>11.44 mm</td>
</tr>
<tr>
<td>Exit pupil position</td>
<td>-46</td>
</tr>
<tr>
<td>Filter thread</td>
<td>M27 $\times$ 0.5</td>
</tr>
<tr>
<td>Mount</td>
<td>C</td>
</tr>
<tr>
<td>Mass</td>
<td>55 g</td>
</tr>
</tbody>
</table>

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 $1024 \times 1024$ 5.5 $\mu m$ pixels, and operates at up to 200 fps.

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

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Grasshopper Express</th>
<th>Dragonfly Express</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Kodak KAI-01050 CCD 1/2&quot;, 5.5 $\mu m$</td>
<td>Kodak KAI-0340DM/C 1/3&quot;, 7.4 $\mu m$</td>
</tr>
<tr>
<td>Shutter</td>
<td>Global</td>
<td>Global</td>
</tr>
<tr>
<td>Resolution</td>
<td>$1024 \times 1024$</td>
<td>$640 \times 480$</td>
</tr>
<tr>
<td>Gain</td>
<td>24 dB</td>
<td>30 dB</td>
</tr>
<tr>
<td>Digital Interface</td>
<td>IEEE-1394-b</td>
<td>IEEE-1394-b</td>
</tr>
<tr>
<td>Transfer Rate</td>
<td>100 MB/s</td>
<td>100 MB/s</td>
</tr>
<tr>
<td>Dimensions</td>
<td>$44 \text{ mm} \times 29 \text{ mm} \times 58 \text{ mm}$</td>
<td>$63.5 \text{ mm} \times 50 \text{ mm} \times 13 \text{ mm}$</td>
</tr>
<tr>
<td>Camera Control</td>
<td>FlyCapture SDK</td>
<td>FlyCapture SDK</td>
</tr>
</tbody>
</table>
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 \( \sim 350 \) 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.
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 $\sim 100 \, \mu m$ 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).
3. System Design

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 \((D_{\text{min}})\) 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

\[
NA = n \sin(\theta_{\text{max}}) = \sqrt{n_1^2 - n_2^2},
\]

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

\[
NA_{\text{eff}} = NA \times \frac{D_{\text{min}}}{D},
\]

where \(D_{\text{min}}\) 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.

\[\text{Figure 3.8: An example of a demagnifying fibre-optic taper. The image on the left is demagnified to produce the image on the right.}\]
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

$$Sp\text{-BW} = \frac{\text{Area of Detector}}{\text{Area of PSF}},$$  \hspace{1cm} (3.3)
3. System Design

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 $\sim 200$ 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\textsuperscript{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\textsuperscript{TM} printer (Objet Geometries Ltd). The printer has a 35 × 35 × 20 cm\textsuperscript{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.](image)

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).

<table>
<thead>
<tr>
<th>Material Name</th>
<th>Material Type</th>
<th>Tensile Strength (MPa)</th>
<th>Modulus of Elasticity (MPa)</th>
<th>Flexural Strength (MPa)</th>
<th>Flexural Modulus (MPa)</th>
<th>Polymerized Density (g/cm³)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FullCure720</td>
<td>Transparent</td>
<td>50-65</td>
<td>2000-3000</td>
<td>80-100</td>
<td>2700-3200</td>
<td>1.18</td>
</tr>
<tr>
<td>VeroGrey</td>
<td>Rigid &amp; Opaque</td>
<td>50-65</td>
<td>2000-3000</td>
<td>75-110</td>
<td>2200-3200</td>
<td>1.17</td>
</tr>
<tr>
<td>TangoPlus</td>
<td>Rubber-like</td>
<td>0.8 - 1.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.12</td>
</tr>
</tbody>
</table>

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 $\sim 18$ 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.

Figure 3.12: Several views of the collimator casing. This casing was made with FullCure720™ 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

$$\tan \left( \frac{\alpha}{2} \right) = \frac{r_1 - r_p}{d_1} = \frac{r_1}{d_1 + x},$$  \hspace{1cm} (3.4)
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):

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

Using this value for $x$ and the corresponding value for $\alpha$, 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
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™ 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™ 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
3. System Design

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.
3. System Design

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\textsuperscript{TM} 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
3. System Design

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
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.

![LabVIEW interface workflow](image_url)

**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

\[ g = Hf + n, \]  

(4.1)

where \( H \) is the discrete system matrix that maps the object vector, \( f \), to an
image, \( g \), and the noise, \( n \), is a M-dimensional vector in data space. The object,
\( f \), is decomposed as a set of \( N \) voxels \( ([f_1, \ldots, f_N]^t) \), and the image as \( M \) data
bins, therefore \( H \) is a \( M \times N \) matrix. As mentioned in section 2.6, there 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 impracti-
cal. 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 \( \times \) 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
4. Performance Characterization of Synthetic SPECT

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 × 480 × 8 bit image that has been cropped to the area around the active pixels. 320 × 240 × 8 bit images were primarily used for data acquisition to increase the number of counts per pixel.](image)

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.

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

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.
4. Performance Characterization of SyntheticSPECT

**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.
4. Performance Characterization of SyntheticSPECT

[Graph showing counts over slice number for different configurations.]

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
4. Performance Characterization of Synthetic SPECT

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™ function \textit{"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).
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.

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™ 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 \( (\sigma_x) \), the \( y \)-width \( (\sigma_y) \), and the correlation coefficient \( (\rho) \). 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\textsuperscript{TM} ("fmincon"). The pixel response function can be represented by

\[
f(x, y) = \frac{A}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \times \exp\left\{ -\frac{1}{2(1-\rho^2)} \left[ \frac{(x-x)^2}{\sigma_x^2} + \frac{(y-y)^2}{\sigma_y^2} - \frac{2\rho(x-x)(y-y)}{\sigma_x\sigma_y} \right] \right\}. \tag{4.2}
\]

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:

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

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

\[
-\ln(pr(g|\theta)) = \sum_{m=1}^{M} \frac{1}{2} \ln[(2\pi\tilde{g}_m(\theta))] + \frac{(g_m - \tilde{g}_m(\theta))^2}{2\tilde{g}_m(\theta)}. \tag{4.4}
\]

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

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

\[
\text{Total Counts} = A \times 2\pi \times \sigma_x \times \sigma_y \times \sqrt{1-\rho^2}. \tag{4.5}
\]

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.
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
4. Performance Characterization of SyntheticSPECT

![Figure 4.13](image)

**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.
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.

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 $\sigma_x$ and $\sigma_y$, 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 (\(\sigma_x, \sigma_y\)), correlation coefficients (\(\rho\)). These 3D arrays were then interpolated to the desired grid size using MATLAB\textsuperscript{TM}’s 3D interpolation function (“interp3”). Interpolation using this function requires a complete 3D grid of data; voxels with no measured values will result in errors.
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.
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...
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) \( \sigma_x \), (c) \( \sigma_y \), (d) \( \rho \), (e) Total signal under Gaussian. All images are normalized to the same range, (f) Total signal under Gaussian. This is the same data as (e) but all images are individually normalized to help visualize the distribution.

MATLAB\textsuperscript{TM} function “interp3”, 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\textsuperscript{TM} 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>
<thead>
<tr>
<th>Voxel size (mm)</th>
<th># voxels</th>
<th># system configurations</th>
<th># matrix elements</th>
<th>Size (GB) with projections</th>
<th>Size (GB) with coefficients</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>61 × 61 × 61</td>
<td>6</td>
<td>1 × 10^{11}</td>
<td>408</td>
<td>0.7</td>
</tr>
<tr>
<td>0.3125</td>
<td>97 × 97 × 97</td>
<td>6</td>
<td>4 × 10^{11}</td>
<td>1.6 × 10^{3}</td>
<td>2.9</td>
</tr>
</tbody>
</table>

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.
4. Performance Characterization of SyntheticSPECT

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™ 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 9th position (amplitude = 0), and the data between the 8th and 9th 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 (Velmax 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.

<table>
<thead>
<tr>
<th>System Configuration</th>
<th>Root-Mean-Square Difference between $C_e$ and $C_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\bar{x}$ (pixels)</td>
</tr>
<tr>
<td>1</td>
<td>0.1446</td>
</tr>
<tr>
<td>2</td>
<td>0.2587</td>
</tr>
<tr>
<td>5</td>
<td>0.4330</td>
</tr>
<tr>
<td>6</td>
<td>0.5648</td>
</tr>
</tbody>
</table>

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.

![Images of the number of overlapping pinhole projections at each system configuration.](image)

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.
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, $f_s(r)$, against a background, $f_b(r)$, 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...
4. Performance Characterization of Synthetic SPECT

\[ H_i \]

is

\[ pr(g|H_i) = \prod_{m=1}^{M} \exp(-\bar{g}_{im}) \frac{\bar{g}_{im}^n}{g_m^1}, \quad (4.6) \]

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

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

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

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

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.

89
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 $16 \times 16 \times 2 \text{ mm}^3$ uniform reservoir below the rods. Simulation 1 was designed to validate the efficacy of combining multiplexed and non-multiplexed
4. Performance Characterization of Synthetic SPECT

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 $\simeq 1.5, 1.6, 2.0$.

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.
The SNR increases linearly with contrast in the object, and using more projection images increases the SNR of the data. Simulation 2 was designed to test the depth resolution of the system when combining multiplexed and non-multiplexed data.

**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.

**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.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>System Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1 &amp; 6</td>
</tr>
<tr>
<td>4</td>
<td>1, 2, 3, 4, 5 &amp; 6</td>
</tr>
</tbody>
</table>

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

$$g_m = \int_S h_m(r)f(r)dr,$$

(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, $f$, can be represented by a Fourier series

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

(4.10)
where

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

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

$$\rho_k = \frac{k}{L}, \quad (4.12)$$

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

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

where

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

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

$$\beta_{kk'} = \sum_{m=1}^{M} \Psi_{mk}^{*} \Psi_{mk'}, \quad (4.15)$$

where $M$ is the total number of detector elements. The diagonal element, $\beta_{kk}$, 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 $\beta_{kk}$ 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

$$\text{MTF}^2_{eq} = |\rho_k| \beta_{kk}. \quad (4.16)$$

If $\beta_{kk} = 0$, the basis function $\Phi_k(r)$ makes no contribution to the data and cannot 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

$$MTF_{eq}(j) = A_0 \exp\left(-\frac{j^2}{2\sigma^2}\right),$$

(4.17)

and its Fourier transform as

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

(4.18)

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

$$\text{Spatial Resolution} = \frac{4\ln(2)}{\pi} \frac{1}{\text{FWHM}[MTF_{eq}(j)]}.$$  

(4.19)

The Fourier lattice has a grid of spacing $1/L$, and a maximum frequency of $1/(2\Delta H)$ in each direction, where $\Delta H$ 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
4. Performance Characterization of Synthetic SPECT

![Graphs](image)

Figure 4.29: The normalized MTF<sub>eq</sub> 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<sub>eq</sub> data (dots). The centre of each plot deviates from an approximate Gaussian due to |ρ_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.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{spectra.png}
\caption{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 $5 \times 5$ region around the centroid of each cluster.}
\end{figure}

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
4. Performance Characterization of SyntheticSPECT

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 × 61 × 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 ($t_{1/2}$) 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

$$N = N_0 e^{-\lambda t}, \quad (5.1)$$

and

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

where $N_0$ is the initial activity, $\lambda$ is the decay constant, and $t$ is the time that had elapsed.
\(^{99m}\text{Tc}\) binds to many biologically-active substances that make it suitable for different imaging applications. The phantom studies used \(^{99m}\text{Tc}\)-pertechnetate. The animal studies in NUIG also used \(^{99m}\text{Tc}\)-pertechnetate, and the animal study in CGRI used \(^{99m}\text{Tc}\)-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. \(^{99m}\text{Tc}\)-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 \(^{99m}\text{Tc}\)-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.
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.

<table>
<thead>
<tr>
<th>Tissue</th>
<th>Intravenous AdNIS</th>
<th>Intravenous MSC-NIS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>hNIS Day 3 Day 7</td>
<td>hMRPL19 Day 3 Day 7</td>
</tr>
<tr>
<td>Tumour</td>
<td>- - + +</td>
<td>++ ++ ++ ++</td>
</tr>
<tr>
<td>Heart</td>
<td>+ - - -</td>
<td>+ + ++ + +</td>
</tr>
<tr>
<td>Lungs</td>
<td>+ - - -</td>
<td>+ + ++ + +</td>
</tr>
<tr>
<td>Liver</td>
<td>+ - - -</td>
<td>+ - - - -</td>
</tr>
<tr>
<td>Spleen</td>
<td>- - - -</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>Kidneys</td>
<td>- - - -</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>Stomach</td>
<td>- - - -</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>Small Intestine</td>
<td>- - - -</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>Large Intestine</td>
<td>- - - -</td>
<td>+ + + + + +</td>
</tr>
</tbody>
</table>

- 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).
5. Imaging with BazookaSPECT and SyntheticSPECT

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 $16 \times 16 \times 2$ 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$^\text{TM}$, and printed with a 3D printer using FullCure 720$^\text{TM}$ material.

The phantom was soaked in a bath of Kodak PhotoFlo$^\text{TM}$ 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™ 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Frame Parsing</td>
</tr>
<tr>
<td>Time</td>
<td>20 mins</td>
</tr>
<tr>
<td>Video Mode</td>
<td>$320 \times 240 \times 8$-bit</td>
</tr>
<tr>
<td>Frame Rate</td>
<td>200 FPS</td>
</tr>
<tr>
<td>CCD Gain</td>
<td>Max</td>
</tr>
<tr>
<td>CCD Shutter Time</td>
<td>2 ms</td>
</tr>
<tr>
<td>Signal Threshold</td>
<td>7</td>
</tr>
<tr>
<td>Cluster Radius</td>
<td>5</td>
</tr>
<tr>
<td>Isotope</td>
<td>$^{99m}$Tc</td>
</tr>
</tbody>
</table>

5.2.2 MLEM Reconstruction

The phantom was reconstructed on the $97 \times 97 \times 97$ 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
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
5. Imaging with BazookaSPECT and SyntheticSPECT

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.

The 3D 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.
5. Imaging with BazookaSPECT and SyntheticSPECT

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.
5. Imaging with BazookaSPECT and SyntheticSPECT

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 \( \sim 1.4 \) 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
5. Imaging with BazookaSPECT and SyntheticSPECT

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.

![Images of reconstructed phantoms with different iterations](image)

**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.
5. Imaging with BazookaSPECT and SyntheticSPECT

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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>No. of voxels</th>
<th>Time (1 iteration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLEM</td>
<td>$61 \times 61 \times 61$</td>
<td>31 s</td>
</tr>
<tr>
<td>POSEM</td>
<td>$61 \times 61 \times 61$</td>
<td>30 s*</td>
</tr>
</tbody>
</table>

*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 × 30 × 30 mm³ 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 ⁹⁹mTc-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
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.
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.

Figure 5.14: (a) - (d) Projections from the second FOV volume at 4 different system configurations.
5. Imaging with BazookaSPECT and SyntheticSPECT

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

Figure 5.16: (a) - (d) Projections from the fourth 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).
5. Imaging with BazookaSPECT and SyntheticSPECT

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. Imaging with BazookaSPECT and SyntheticSPECT

5.3 Conclusions

The BazookaSPECT system successfully demonstrated its ability to track MSCs \emph{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.
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, $\mathbf{H}$, 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
6. Conclusions and Future Work

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.
6. Conclusions and Future Work

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.
CUDA Code

A.1 Program Headers

```c
#define _USE_MATH_DEFINES
#include <stdio.h>
#include <cuda.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <ctime>
#include <string>
#include <sstream>
#include <stdexcept>
#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
```
A. CUDA Code

```c
#define CUDA_CALL(x) {cudaError_t cuda_error_ = (x); if (cuda_error_) printf(#x " returned \"%s\"\n", cudaGetErrorString(cuda_error_));}
```

A.2 Utility Functions

```c
/* Convert an int to a string. */
std::string IntToStr(int n) {
    std::stringstream result;
    result << n;
    return result.str();
}

/* Convert a string to an int. */
int StrToInt(const std::string& s) {
    int result;
    std::stringstream ss(s);
    ss >> result;
    if (!ss) throw std::invalid_argument("StrToInt");
    return result;
}

/* Check CUDA device for errors. */
void syncDeviceAndErrorCheck() {
    cudaDeviceSynchronize();
    cudaError_t err = cudaGetLastError();
    if (err != cudaSuccess) {
        printf("Last Error: \%s\n", cudaGetErrorString(err));
        int stop;
        scanf("%d", &stop);
    }
}

/* Set float values in a CUDA array to zero. */
__global__ void fl_setToZeroCUDA(float* image, int N) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) 
        image[idx] = 0.0f;
}

/* Invert a CUDA array. */
__global__ void fl_invertValueCUDA(float* image, int N) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
```
if (idx < N)
{
    if (image[idx] > 0.0f)
        image[idx] = 1.0f / image[idx];
    else
        image[idx] = 0.0f;
}

/*Apply a mask to a CUDA array*/
__global__ void fl_applyMaskCUDA(float *image, int *mask, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N)
    {
        if (mask[idx] == 0)
            image[idx] = 0.0f;
    }
}

/*Multiply two matrices*/
__global__ void fl_multiplyMatricesCUDA(float *matrix1, float *matrix2, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N)
    {
        matrix1[idx] = matrix1[idx] * matrix2[idx];
    }
}

/*Multiply two matrices with a weight*/
__global__ void fl_addMatricesWithWeightsCUDA(float *matrix1, float *matrix2, float *currentValue, int voxelIndex, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    extern __shared__ float shared_currentValue[];
    if (threadIdx.x == 0)
    {
        shared_currentValue[0] = currentValue[voxelIndex];
    }
    __syncthreads();
    if (idx < N)
    {
        matrix1[idx] += matrix2[idx] * shared_currentValue[0];
    }
}

/*Divide two matrices*/
__global__ void fl_divideMatricesCUDA(float *matrixResult, float *matrix1, float *matrix2, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N)
A. CUDA Code

```c
if (matrix2[idx] > 0.0f)
  matrixResult[idx] = matrix1[idx] / matrix2[idx];
else
  matrixResult[idx] = 0.0f;
}

/* Set a CUDA array to a value. */
__global__ void fl_setToValueCUDA(float *images_d, float value, int *mask, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < N)
  {
    if (mask[idx] == 1)
      images_d[idx] = value;
  }
}

/* Sum arrays. Very inefficient method. */
__global__ void mlem_sumImages(float *detectorSum, float *images_d, int imageSize, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < N)
  {
    float val = 0.0f;
    for (int i = 0; i < imageSize; i++)
    {
      val += images_d[idx * imageSize + i];
    }
    detectorSum[idx] += val;
  }
}

/* Efficient addition of arrays, used with 'getSumArrayGPUtoGPUWithMultiplication'. */
/* This is efficient reduction code adapted from code published by NVIDIA. */
-template <unsigned int blockSize>
__global__ void reduce6WithMultiplicationMultipleImages(float *g_idata, float *input2, float *g_odata, unsigned int n) {
  extern __shared__ float sdata[];
  unsigned int tid = threadIdx.x;
  unsigned int i = blockIdx.x * (blockSize * 2) + tid;
  unsigned int gridSize = blockSize * 2 * blockDim.x;
  sdata[tid] = 0.0f;
  while (i < n)
  {
    sdata[tid] += g_idata[i] * input2[i];
    if (i + blockSize < n)
      sdata[tid] += g_idata[i + blockSize] * input2[i + blockSize];
    i += gridSize;
  }
```

-138-
A. CUDA Code

```c
154 } _syncthreads();
155 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) _global_data[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 while (N > 1)
178 {
179 numBlocks = (N + threadsPerBlock*2 -1) / (threadsPerBlock*2);
180 dim3 dimGrid(numBlocks, 1);
181 if (numBlocks==1)
182 {
183 reduce6MultipleImages<512><<dimGrid, dimBlock, sizeof(float) * threadsPerBlock>>>(input_d, &output_d[index], N);
184 N = numBlocks;
185 }
186 else{
187 reduce6WithMultiplicationMultipleImages<512><<dimGrid, dimBlock, sizeof(float) * threadsPerBlock>>>(input_d, input2_d, input_d, N);
188 N = numBlocks;
189 }
190 }
191
192 /*Set an int array to zero.*/
193 void resetToZero(int *H, int num)
194 {
195 for (int i=0; i<num; i++)
196 {
197 H[i] = 0;
198 }
199 }
```

-139-
A. CUDA Code

```c
/* Reset a float array to zero. */
void resetToZero(float *H, int num)
{
    for (int i=0; i<num; i++)
        H[i] = 0.0f;
}

/* Set a float array to a value. */
void resetToValue(float *H, int num, float value)
{
    for (int i=0; i<num; i++)
        H[i] = value;
}

/* Set an int array to zero. */
void i-resetToValue(int *H, int num, int value)
{
    for (int i=0; i<num; i++)
        H[i] = value;
}

/* Multiplication for POSEM algorithm. */
__global__ void osem_weightedMultiplication(float *proj_d, float *data_d, float *currentValue, int *pixelIndices, int N)
{
    int idx = blockIdx.x*blockDim.x+threadIdx.x;
    extern __shared__ float shared_currentValue[];
    if (threadIdx.x ==0)
        shared_currentValue[0] = currentValue[0];
    __syncthreads();
    if (idx < N)
        proj_d[pixelIndices[idx]] += data_d[idx]*shared_currentValue[0];
}

/* Sum a block for POSEM. */
__global__ void osem_sumBlocks(float *sensitivity, float *data, int blockSize, int N)
{
    int idx = blockIdx.x*blockDim.x+threadIdx.x;
    if (idx < N)
    {
        float val=0.0f;
        for (int i=0; i<blockSize; i++)
            val += data[idx*blockSize + i];
    }
```
A. CUDA Code

```c
sensitivity[idx] += val;
```

```c
/* Update the current estimate of the activity distribution on the GPU. Used for POSEM and MLEM. */
__global__ void mlem_updateEstimateCUDA(float *currentPositionEstimate, float *sensitivityProj, float *detectorSum, int voxelSpaceSize)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < voxelSpaceSize)
    {
    }
}
```

A.3 Masks

```c
/* Apply a mask to an int array. */
void i_applyCircularMask(int *mask)
{
    /* Center found using centroiding. */
    int xCenter = 174;
    int yCenter = 134;
    cout << "Warning: Mask Applied. " << endl;
    for (int i = 0; i < IMAGE_DIMX * IMAGE_DIMY; i++)
    {
        int y = (int)floor((float)i / (float)IMAGE_DIMX);
        int x = i % IMAGE_DIMX;
        if (sqrt((float)((float)pow((float)(x - xCenter), 2) + (float)pow((float)(y - yCenter), 2))) > 79.0f)
        {
            mask[i] = 0;
        }
        
        /* Deadspots. */
        if (sqrt((float)((float)pow((float)(x - 116), 2) + (float)pow((float)(y - 123), 2))) < 8.0f)
        {
            mask[i] = 0;
        }
        if (sqrt((float)((float)pow((float)(x - 104), 2) + (float)pow((float)(y - 258), 2))) < 8.0f)
        {
            mask[i] = 0;
        }
    }
    
    /* Apply a mask to a float array. */
```

-141-
A. CUDA Code

```c

void fl_applyCircularMask(float *mask)
{
    int xCenter = 174;
    int yCenter = 134;
    cout << "Warning: Mask Applied. " << endl;
    for (int i = 0; i < IMAGE_DIMX*IMAGE_DIMY; i++)
    {
        int y = (int)floor((float)i/(float)IMAGE_DIMX);
        int x = i%IMAGE_DIMX;
        if (sqrt((float)(pow((float)(x - xCenter),2) + (float)pow((float)(y-yCenter),2))) > 79.0f)
        {
            mask[i] = 0.0f;
        }
        /* Deadspots. */
        if (sqrt((float)(pow((float)(x - 116),2) + (float)pow((float)(y-123),2))) < 8.0f)
        {
            mask[i] = 0.0f;
        }
        if (sqrt((float)(pow((float)(x - 104),2) + (float)pow((float)(y-258),2))) < 8.0f)
        {
            mask[i] = 0.0f;
        }
    }
}

A.4 Index the Coefficients

/*Sort and index the coefficient files. Very inefficient.*/
int getSize(FILE *fid)
{
    int numFloats = 0;
    int numCoeff = 0;
    for (int i = 0; i < DIMX; i++)
    {
        for (int j = 0; j < DIMY; j++)
        {
            for (int k = 0; k < DIMZ; k++)
            {
                fread(&numCoeff, sizeof(int), 1, fid);
                numFloats = numFloats + numCoeff*6;
                fseek(fid, numCoeff*6* sizeof(float), SEEK_CUR);
            }
        }
    }
    return numFloats;
}
```

---

-142-
A. CUDA Code

```c
void loadCoefficients(FILE *fid, int *numberOfCoefficients, float *gaussianCoefficients)
{
    int numCoeff = 0;
    int gaussianFilePosition = 0;
    int numberOfFilePosition = 0;
    for (int i = 0; i < DIMX; i++)
    {
        for (int j = 0; j < DIMY; j++)
        {
            for (int k = 0; k < DIMZ; k++)
            {
                fread(&numCoeff, sizeof(int), 1, fid);
                numberOfCoefficients[numberOfFilePosition] = numCoeff;

                fread(&gaussianCoefficients[gaussianFilePosition], 1, sizeof(float)*6*numCoeff, fid);
                numberOfFilePosition += 6 * numCoeff;
                gaussianFilePosition += 6 * numCoeff;
            }
        }
    }
}

int getFilePathPosition(int *numberOfCoefficients, int index)
{
    int filePathPosition = 0;
    for (int i = 0; i < index; i++)
    {
        filePathPosition += numberOfCoefficients[i];
    }
    return filePathPosition * 6;
}

void getGaussPosition(int *gaussPosition, int *numberOfCoefficients, int offset)
{
    int index = 0;
    for (int i = 0; i < DIMX; i++)
    {
        for (int j = 0; j < DIMY; j++)
        {
            for (int k = 0; k < DIMZ; k++)
            {
                gaussPosition[index] = getFilePathPosition(numberOfCoefficients, index) + offset;
                index++;
            }
        }
    }
}
```
A.5 Gaussian Kernels

```c
/* Used for the MLEM algorithm. */

/* CUDA kernel to calculate Gaussian response. */
__device__ float gaussKernelOther(float *sharedGaussianCoefficient, 
    float xCent, float yCent)
{
    float a = ((xCent - (sharedGaussianCoefficient[1])) * (xCent - (sharedGaussianCoefficient[1]))) / ((sharedGaussianCoefficient[3]) * (sharedGaussianCoefficient[3]));
    float b = -(2.0f * sharedGaussianCoefficient[5]) * (xCent - (sharedGaussianCoefficient[1])) * (yCent - (sharedGaussianCoefficient[2])) / (sharedGaussianCoefficient[3] * sharedGaussianCoefficient[4]);
    float z = a + b + c;
    float tempVal = sharedGaussianCoefficient[0] * (expf(-z / (2.0f * (1.0f - (sharedGaussianCoefficient[5] * sharedGaussianCoefficient[5])))));
    return tempVal;
}

/* Used for the POSEM algorithm. */
/* CUDA kernel to calculate Gaussian response. */
__device__ float gaussKernelOther2(float *sharedGaussianCoefficient, 
    float xSq, float ySq, float xCent, float yCent)
{
    float a = (xSq) / ((sharedGaussianCoefficient[3]) * (sharedGaussianCoefficient[3]));
    float b = -(2.0f * sharedGaussianCoefficient[5]) * (xCent - (sharedGaussianCoefficient[1])) * (yCent - (sharedGaussianCoefficient[2])) / (sharedGaussianCoefficient[3] * sharedGaussianCoefficient[4]);
    float c = (ySq) / (sharedGaussianCoefficient[4] * sharedGaussianCoefficient[4]);
    float z = a + b + c;
    float tempVal = sharedGaussianCoefficient[0] * (expf(-z / (2.0f * (1.0f - (sharedGaussianCoefficient[5] * sharedGaussianCoefficient[5])))));
    return tempVal;
}

/* Project the Gaussian distribution. */
/* Used for MLEM Implementation 1 */
__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)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
}
if (idx < N)
{
    float x[6];
    int num_sigma = 3;
    int currentIndex = startVoxel + idx;
    int gaussIndex = gaussPosition_d[currentIndex] -
    gaussPositionOffset;
    int imageIndex = idx * imageSize;

    /*Loop through each group of 6 coefficients.*/
    for (int i = 0; i < numberOfCoefficients_d[currentIndex]; i++)
    {
        /*Loop through each of the coefficients. (Amplitude, mx, my, sx, sy, rho).*/
        for (int counter = 0; counter < 6; counter++)
        {
            x[counter] = gaussianCoefficients_d[gaussIndex + i * 6 +
            counter];
            if (counter == 1 || counter == 2)
                x[counter] = x[counter] - 1.0f; /*Convert from MATLAB coordinates to C.*/
        }
    }
    /*All the coefficients are loaded for a single projection.*/
    for (int yCounter = floor(x[2] - num_sigma * x[4]); yCounter <= ceil(x
    [2] + num_sigma * x[4]); yCounter++)
    {
        if ((yCounter >= 0) && (yCounter < IMAGE_DIMY))
        {
            for (int xCounter = floor(x[1] - num_sigma * x[3]); xCounter <=
            ceil(x[1] + num_sigma * x[3]); xCounter++)
            {
                if ((xCounter >= 0) && (xCounter < IMAGE_DIMX))
                {
                    images_d[imageIndex + (IMAGE_DIMX) * (yCounter) +
                    xCounter] += gaussKernelOther(x, (float)xCounter, (float)yCounter);
                }
            }
        }
    }
}

/*Used for Implementation 2 of MLEM (mlem_recon2)/
__global__ void createPSFFast2(float *gaussianCoefficients_d, float *image_d, int N, float *currentPositionEstimate, int index)
{
    extern __shared__ float sharedGaussianCoefficient[];
    if (threadIdx.x < 6)
    {
        if (threadIdx.x == 0)
        {
            sharedGaussianCoefficient[0] = currentPositionEstimate[index] *
            gaussianCoefficients_d[blockIdx.x * 6 + threadIdx.x];
        }
        else if (threadIdx.x == 1 || threadIdx.x == 2)
        {

    
}
sharedGaussianCoefficient[threadIdx.x] =
gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x] - 1.0f;
}  
else{
    sharedGaussianCoefficient[threadIdx.x] =
gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x];
}  
syncThreads();

if (threadIdx.x < N && threadIdx.y < N)
{
    int y = sharedGaussianCoefficient[2] - (N/2.0f) + threadIdx.y +
0.5f;
    int x = sharedGaussianCoefficient[1] - (N/2.0f) + threadIdx.x +
0.5f;
    if (x >= 0 && x < IMAGE_DIMX && y >= 0 && y < IMAGE_DIMY)
    {
        image_d[IMAGE_DIMX*y + x] += gaussKernelOther(
sharedGaussianCoefficient, x, y);
    }
}

__global__ void createPSFFast2WithH(float *gaussianCoefficients_d, 
float *image_d, int N)
{
    extern __shared__ float sharedGaussianCoefficient[];
    if (threadIdx.x < 6)
    {
        if (threadIdx.x == 1 || threadIdx.x == 2)
        {
            sharedGaussianCoefficient[threadIdx.x] =
gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x] - 1.0f;
        }
        else{
            sharedGaussianCoefficient[threadIdx.x] =
gaussianCoefficients_d[blockIdx.x*6 + threadIdx.x];
        }
    }
    syncThreads();

    if (threadIdx.x < N && threadIdx.y < N)
    {
        int y = sharedGaussianCoefficient[2] - (N/2.0f) + threadIdx.y +
0.5f;
        int x = sharedGaussianCoefficient[1] - (N/2.0f) + threadIdx.x +
0.5f;
        if (x >= 0 && x < IMAGE_DIMX && y >= 0 && y < IMAGE_DIMY)
        {
            image_d[IMAGE_DIMX*y + x] += gaussKernelOther(
sharedGaussianCoefficient, x, y);
        }
    }
A. CUDA Code

A.6 MLEM

A.6.1 Functions

/*This function adds up all the pixels in each image to get the response of the detector to each voxel. Response will have the same size as numberOfImages. N is the numberOfImages. Very inefficient implementation.*/
```c
__global__ void voxelResponse(float *response_d, float *images_d, int *mask, int imageSize, int numberOfVoxelsToProcess, int N)
{
    int idx = blockIdx.x*blockDim.x+threadIdx.x;
    if (idx < N)
    {
        response_d[idx] = 0.0f;
        for (int i=0; i<imageSize; i++)
        {
            if (mask[i] == 1)
            {
                response_d[idx] += images_d[idx*imageSize + i];
            }
        }
    }
}

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)
{
    int i=0;
    /* Projections to hold on GPU. */
    float *proj_d;
    CUDA_CALL(cudaMalloc((void**)&proj_d, sizeof(float)*imageSize*numberOfImages));
    for (int imC=0; imC<numberOfImages; imC++)
        fl_setToZeroCUDA<<<(int)ceil((float)imageSize/512.0f),512>>>&proj_d[imC*imageSize], imageSize);

    /* Coefficients. */
    int numberOfVoxelsToProcessTemp = 0;
    int *numberOfCoefficients_d;
    float *gaussianCoefficients_d; /*Memory not declared yet.*/
    int *gaussPosition_d;
    CUDA_CALL(cudaMalloc((void**)&numberOfCoefficients_d, sizeof(int)*voxelSpaceSize));
    CUDA_CALL(cudaMalloc((void**)&gaussPosition_d, sizeof(int)*voxelSpaceSize));

    /* Masks. */
    int *mask_h;
    mask_h = (int*)malloc(sizeof(int)*imageSize);
    i_resetToValue(mask_h, imageSize, 1);
    i_applyCircularMask(mask_h);
    int *mask;
    CUDA_CALL(cudaMalloc((void**)&mask, sizeof(int)*imageSize));
    CUDA_CALL(cudaMemcpy(mask, mask_h, sizeof(int)*imageSize, cudaMemcpyHostToDevice));
```

A. CUDA Code

```c
/*Send g to the device.*/
float *g_d;
CUDA_CALL(cudaMalloc((void**)&g_d, sizeof(float)*imageSize*numberOfImages));
CUDA_CALL(cudaMemcpy(g_d, g, sizeof(float)*imageSize*numberOfImages, cudaMemcpyHostToDevice));

float *H_d;
CUDA_CALL(cudaMalloc((void**)&H_d, sizeof(float)*imageSize*numberOfImages));

for (int imageNumber=0;imageNumber<numberOfImages;imageNumber++)
{
  i=0;
  /*Get offset for the coefficient file.*/
  int gaussPositionOffset = gaussPosition_h[imageNumber*voxelSpaceSize];

  /*Declare memory for coefficients now. This changes in each loop.*/
  CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d, sizeof(float)*numFloats[imageNumber]));

  /*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));

  numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;
  while (i < voxelSpaceSize)
  {
    /*Make sure it doesn’t go out of bounds.*/
    if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
    {
      numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
    }
    /*Set all projection images to zero.*/
    for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
      fl_setToZeroCUDA<<<(int)cei((float)imageSize/512.0f),512>>>(images_d[imC*imageSize], imageSize);

    /*Create the projection image for a number of voxels.*/
    createPSFFast<<<(int)cei(numberOfVoxelsToProcessTemp),128>>>(numberOfCoefficients_d, gaussianCoefficients_d, gaussPosition_d, images_d, imageSize, i, imageNumber, voxelSpaceSize, gaussPositionOffset, numberOfVoxelsToProcessTemp);

    /*Multiply the projection images by the current estimate and add them all together.*/
    for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
```
A. CUDA Code

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

for (int imC=0; imC < numberOfVoxelsToProcessTemp; imC++)
fl_setToZeroCUDA<<<(int) ceil((float) imageSize / 512.0f), 512
        &images_d[imC*imageSize], imageSize);  
createPSSFast<<<(int) ceil((float) numberOfVoxelsToProcessTemp / (float)(128)), 128>>>(
        &H_d[imageNumber*imageSize], imageData_d[imC*imageSize], &images_d, imageSize, i, imageNumber, voxelSpaceSize,  
        gaussPositionOffset, numberOfVoxelsToProcessTemp);

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);

mlem_sumImages<<<(int) ceil((float) numberOfVoxelsToProcessTemp / (float)(128)), 128>>>(
        &detectorSum_d[i], images_d, imageSize, numberOfVoxelsToProcessTemp);
    i = i + numberOfVoxelsToProcessTemp;
}

CUDA_CALL(cudaFree(gaussianCoefficients_d));

// Free the variables */
cudaFree(gaussPosition_d);
cudaFree(numberOfCoefficients_d);
cudaFree(proj_d);
cudaFree(H_d);
cudaFree(mask);
free(mask_h);
}

/* Implementation 2 of MLEM -> Cluster-by-Cluster method */
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)
{
    int i = 0;
syncDeviceAndErrorCheck();
    for (int imC = 0; imC < numberOfImages; imC++)
        fl_setToZeroCUDA<<<(int) ceil((float) imageSize / 512.0f), 512
            >&proj_d[imC*imageSize], imageSize);
    int numberOfVoxelsToProcessTemp = 0;
    float *gaussianCoefficients_d;
    
    dim3 bl_Size;
    bl_Size.x = 11; bl_Size.y = 11;
for (int imageNumber=0;imageNumber<numberOfImages;imageNumber++)
{
    i = 0;
    int gaussPositionOffset = gaussPosition_h[imageNumber*voxelSpaceSize];
    CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d, sizeof(float)*numFloats[imageNumber]));
    CUDA_CALL(cudaMemcpy(gaussianCoefficients_d, gaussianCoefficients_h[gaussPosition_h[imageNumber*voxelSpaceSize]], sizeof(float)*numFloats[imageNumber], cudaMemcpyHostToDevice));

    numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;
    while (i < voxelSpaceSize)
    {
        if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
        {
            numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
        }
        for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
        {
            int gaussIndex = gaussPosition_h[imageNumber*voxelSpaceSize + i+imC] - gaussPositionOffset;
            int numberOfCoeff = numberOfCoefficients_h[imageNumber*voxelSpaceSize + i + imC];
            dim3 num_bl;
            num_bl.x = numberOfCoeff;
            num_bl.y = 1;
            if (numberOfCoeff > 0)
            {
                createPSFFast2<<<num_bl, bl_Size, 6*sizeof(float)>>>(gaussianCoefficients_d[gaussIndex], &proj_d[imageNumber*imageSize], bl_Size.x, currentPositionEstimate_d, i+imC);
            }
            i = i + numberOfVoxelsToProcessTemp;
        }
    }
     CUDA_CALL(cudaFree(gaussianCoefficients_d));
}

for (int j=0;j<numberOfImages;j++)
fl_applyMaskCUDA<<<(int)ceil(float(imageSize)/(float(512))) ,512>>>(&proj_d[j*imageSize],mask,imageSize);

// Divide g by proj_d and send the result to H_d
for (int j=0;j<numberOfImages;j++)
fl_divideMatricesCUDA<<<(int)ceil((float)imageSize/512.0f) ,512>>>(&H_d[j*imageSize],&g_d[j*imageSize],&proj_d[j*imageSize], imageSize);
fl_setToZeroCUDA<<<(int)ceil((float)voxelSpaceSize/512.0f),512 >>>(detectorSum_d, voxelSpaceSize);

for (int imageNumber = 0;imageNumber<numberOfImages;imageNumber++)
{
int gaussPositionOffset = gaussPosition_h[imageNumber*voxelSpaceSize];

CUDA_CALL(cudaMalloc((void**)&gaussianCoefficients_d, sizeof(float)*numFloats[imageNumber]));
CUDA_CALL(cudaMemcpy(gaussianCoefficients_d, gaussianCoefficients_h[gaussPosition_h[imageNumber*voxelSpaceSize]], sizeof(float)*numFloats[imageNumber], cudaMemcpyHostToDevice));

i = 0;
numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;
while (i < voxelSpaceSize)
{
    if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
    {
        numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
    }
    for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
    {

        fl_setToZeroCUDA<<<(int)ceil((float)imageSize/512.0f),512>>>(&images_d[0*imageSize], imageSize);

        int gaussIndex = gaussPosition_h[imageNumber*voxelSpaceSize + i+imC] - gaussPositionOffset;
        int numberOfCoeff = numberOfCoefficients_h[imageNumber*voxelSpaceSize + i + imC];
        dim3 num_bl;
        num_bl.x = numberOfCoeff;
        num_bl.y = 1;
        if (numberOfCoeff > 0)
            createPSFFast2WithH<<<num_bl, bl_Size,6*sizeof(float)>>>(&gaussianCoefficients_d[gaussIndex],&images_d[0*imageSize], bl_Size .x);

        getSumArrayGPUtoGPUWithMultiplication(&images_d[0*imageSize],&H_d[imageNumber*imageSize], detectorSum_d,i+imC, imageSize);

    }
    i = i + numberOfVoxelsToProcessTemp;
}
CUDA_CALL(cudaFree(gaussianCoefficients_d));

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)
{
/*Diagnostic Output*/
FILE *foutH;
foutH = fopen("H_proj.Speedtest.dat", "wb");
float *response_d;
cudaMalloc((void**)&response_d, sizeof(float)*numberOfVoxelsToProcess);
flsetToZeroCUDA<<<(int)ceil((float)numberOfVoxelsToProcess/512),512>>>(response_d, numberOfVoxelsToProcess);
int numberOfVoxelsToProcessTemp = 0;
/
/*Declare the temporary variables for the GPU!*/
int *numberOfCoefficients_d;
float *gaussianCoefficients_d;
int *gaussPosition_d;
cudaMalloc((void**)&numberOfCoefficients_d, sizeof(int)*voxelSpaceSize);
cudaMalloc((void**)&gaussPosition_d, sizeof(int)*voxelSpaceSize);

/*Mask.*/
int *mask_h;
mask_h = (int*)malloc(imageSize*sizeof(int));
int *mask_d;
cudaMalloc((void**)&mask_d, sizeof(int)*imageSize);
i.resetToValue(mask_h, imageSize, 1);
if (APPLYMASK == 1)
{
i.applyCircularMask(mask_h);
}
CUDA_CALL(cudamempy(mask_d, mask_h, sizeof(int)*imageSize, cudamempyHostToDevice));
cout << " Variables are declared. " << endl;

clock_t start, end;
for (int k=0;k<numberOfImages;k++)
{
	/*Get offset for the coefficient file.*/
	int gaussPositionOffset = gaussPosition_h[k*voxelSpaceSize];
	CUDA_CALL(cudamalloc((void**)&gaussianCoefficients_d, sizeof(float)*numFloats[k]));
	CUDA_CALL(cudamempy(gaussianCoefficients_d,&gaussianCoefficients_h[gaussPosition_h[k*voxelSpaceSize]], sizeof(float)*numFloats[k],cudamempyHostToDevice));
	CUDA_CALL(cudamempy(numberOfCoefficients_d,&numberOfCoefficients_h[k*voxelSpaceSize], sizeof(int)*voxelSpaceSize,cudamempyHostToDevice));
	CUDA_CALL(cudamempy(gaussPosition_d,&gaussPosition_h[k*voxelSpaceSize], sizeof(int)*voxelSpaceSize,cudamempyHostToDevice));
	/*Diagnostic Output.*/
	cout << " gaussPositionOffset: " << gaussPositionOffset << ", numFloats: " << numFloats[k] << endl;/

int i=0;
A. CUDA Code

```c
start = clock();
numberOfVoxelsToProcessTemp = numberOfVoxelsToProcess;

while (i < voxelSpaceSize)
{
    if (i+numberOfVoxelsToProcess >= voxelSpaceSize)
    {
        numberOfVoxelsToProcessTemp = voxelSpaceSize - i;
    }
    for (int imC=0;imC<numberOfVoxelsToProcessTemp;imC++)
    {
        fl_setToZeroCUDA<<<(int) ceil((float)imageSize/512.0f),512
            (&images_d[imC*imageSize], imageSize);
        createPSFFast<<<(int) ceil((float)numberOfVoxelsToProcessTemp/128),128
            (numberOfCoefficients_d, gaussianCoefficients_d,
             gaussPosition_d, images_d, imageSize, i, k, voxelSpaceSize,
             gaussPositionOffset, numberOfVoxelsToProcessTemp);
        for (int n=0;n<numberOfVoxelsToProcessTemp;n++)
        {
            fl_applyMaskCUDA<<<(int) ceil((float)imageSize/(float)(512)),512
                (&images_d[n*imageSize], mask_d, imageSize);
        }
        fl_setToZeroCUDA<<<(int) ceil((float)numberOfVoxelsToProcessTemp/128),128
            (response_d, numberOfVoxelsToProcessTemp);
        mlem_sumImages<<<(int) ceil((float)numberOfVoxelsToProcessTemp/128),128
            (response_d, images_d, imageSize, numberOfVoxelsToProcessTemp);
        CUDA_CALL(cudaMemcpy(&sensitivity[(voxelSpaceSize * k) + i],
            response_d, sizeof(float)*numberOfVoxelsToProcessTemp,
            cudaMemcpyDeviceToHost));
        i = i + numberOfVoxelsToProcessTemp;
    }
    Voxel position check.*/
    cout << "i: " << i << " , VoxelSpaceSize: " << voxelSpaceSize << endl;
    end=clock();
    cout << "sensitivity: " << (double)(end - start)/CLOCKS_PER_SEC << endl;
    cudaFree(gaussianCoefficients_d);
    cout << "H(" << k+1 << ")/" << numberOfImages << " ) Complete." << endl;
}
cout << "Complete" <<endl;
fwrite(sensitivity, voxelSpaceSize*numberOfImages, sizeof(float), foutH);
fclose(foutH);
cudaFree(response_d);
cudaFree(numberOfCoefficients_d);
cudaFree(gaussPosition_d);
cudaFree(mask_d);
free(mask_h);
```
void invertSensitivity(float *sensitivity, int voxelSpaceSize, int numberOfImages)
{
    cout << "Invert Sensitivity." << endl;
    float tempMax = 0.0f;
    for (int i = 0; i < voxelSpaceSize; i++)
    {
        for (int j = 0; j < numberOfImages; j++)
        {
            if (sensitivity[j * voxelSpaceSize + i] > tempMax)
            {
                sensitivity[j * voxelSpaceSize + i] = (1.0f / sensitivity[j * voxelSpaceSize + i]);
            }
            else
            {
                sensitivity[j * voxelSpaceSize + i] = 0.0f;
            }
        }
    }
    float tmpSum = 0.0f;
    for (int ii = 0; ii < voxelSpaceSize * numberOfImages; ii++)
    {
        tmpSum += sensitivity[ii];
    }
    printf("Sensitivities loaded, Sum = %0.2f
", tmpSum);
}

A.6.2 MLEM Main

int main(int argc, char *argv[])
{
    int voxelSpaceSize = DIMX*DIMY*DIMZ;
    int numberOfImages = 0;
    const int NoI = NUMBEROFIMAGES;
    int stopper = 0;

    /*Open the coefficient file.*/
    FILE *fn[6];

    /*97*97*97 cubic grid.*/
    fn[0] = fopen("E:\PSF\H\gaussParams_00mm_082012_v3_PE_raw_cubic.dat", "rb");
    fn[1] = fopen("E:\PSF\H\gaussParams_08mm_082012_v3_PE_raw_cubic.dat", "rb");
    fn[2] = fopen("E:\PSF\H\gaussParams_16mm_082012_v3_PE_raw_cubic.dat", "rb");
    fn[3] = fopen("E:\PSF\H\gaussParams_24mm_082012_v3_PE_raw_cubic.dat", "rb");

    /*...*/
fn[4] = fopen("E:\PSF\H\gaussParams_27mm_082012_v3\PE\raw\cubic\dat" , "rb")
fn[5] = fopen("E:\PSF\H\gaussParams_37mm_082012_v3\PE\raw\cubic\dat" , "rb")

/∗61*61 cubic grid. ∗/
fn[0] = fopen("E:\PSF\H\gaussParams_00mm_082012_v3\PE\raw\cubic\Halfmm.dat" , "rb")
fn[1] = fopen("E:\PSF\H\gaussParams_08mm_082012_v3\PE\raw\cubic\Halfmm.dat" , "rb")
fn[2] = fopen("E:\PSF\H\gaussParams_16mm_082012_v3\PE\raw\cubic\Halfmm.dat" , "rb")
fn[3] = fopen("E:\PSF\H\gaussParams_24mm_082012_v3\PE\raw\cubic\Halfmm.dat" , "rb")
fn[4] = fopen("E:\PSF\H\gaussParams_27mm_082012_v3\PE\raw\cubic\Halfmm.dat" , "rb")
fn[5] = fopen("E:\PSF\H\gaussParams_37mm_082012_v3\PE\raw\cubic\Halfmm.dat" , "rb"); ∗/

/∗ Output Root. */
FILE *fileOut;
string rootOut;
rootOut. assign("E:\Phantom Study\6-21-12\Run3\"");

/∗ Projection data. */
FILE *fil1;
fil1 = fopen("E:\TEMP\PSFproj_rods_reservoir.dat" , "rb");

printf("File... open\n");
int h1[3];
/∗ Read the 3 int header of the coefficient files. */
printf("File... read\n");
for (int fileCnt =0; fileCnt < numberOfImages; fileCnt++)
{
    fread(&h1[0] , sizeof(int) , 3 , fn[fileCnt])
    printf("%d,%d,%d\n",h1[0],h1[1],h1[2]);
}
int imageSize = IMAGE_DIMX*IMAGE_DIMY;

int *imageNumbers;
imageNumbers=(int *)malloc(numberOfImages* sizeof(int));

for (int i=0;i<numberOfImages;i++)
{
    imageNumbers[i] = i;
    printf("imageNumbers[%d] = %d\n" , i , imageNumbers[i]);
}

float *g;
g=(float *)malloc(numberOfImages*imageSize* sizeof(float));
resetToZero(g,numberOfImages*imageSize);

/∗ Read in the 3 int headers to the projection data. */
int tmp[3];
fread(&tmp[0] , sizeof(int) , 3 , fil1);
printf("Image Header : %d %d %d\n Imagesize = %d , numberOfImages = %d\n",tmp[0],tmp[1],tmp[2],imageSize ,numberOfImages);

float *H;
H=(float *)malloc(imageSize*sizeof(float));
resetToZero(H,imageSize);

//Read in the projection images.*/
float *g_tmp;
g_tmp=(float *)malloc(numberOfImages*imageSize*sizeof(float));
read(g_tmp, sizeof(float), numberOfImages*imageSize , fil1);

float sumcheck=0.0f; /*Diagnostic Variable.*/
for ( int i=0;i<numberOfImages;i++)
{
    printf(" Loading Image : %d ... .",imageNumbers[i]);
    for ( int j=0;j<imageSize;j++)
    {
        g[i*imageSize + j] = g_tmp[j + ((imageNumbers[i])*imageSize)];
        sumcheck = sumcheck + g[i*imageSize + j];
    }
    
    if (APPLYMASK == 1)
    {
        fl_applyCircularMask(&g[i*imageSize]);
        printf("sumcheck = %.4f\n",sumcheck);
        sumcheck = 0.0f;
    }
}
cout << " All images loaded. 
" << endl;

/*Diagnostic Output.*/
FILE *foutH;
foutH = fopen("g.dat","wb");
fwrite(g,imageSize*numberOfImages,sizeof(float),foutH);close(foutH);

/*Allocate memory to store all of the coefficients.*/
int *numFloats;
numFloats = (int *)malloc(sizeof(int)*NoI);
for ( int fileCnkt=0;fileCnkt<numberOfImages;fileCnkt++)
{
    numFloats[fileCnkt] = getSize(fn[fileCnkt]);
    printf(" File Number: %d , Count: %d\n",fileCnkt ,numFloats[fileCnkt]);
    rewind(fn[fileCnkt]);
    fread(&h1[0],sizeof(int),3,fn[fileCnkt]);
    printf("%d,%d,%d\n",h1[0],h1[1],h1[2]);
}

int totalFloats = 0;
for ( int fileCnkt=0;fileCnkt<numberOfImages;fileCnkt++)
{
    totalFloats += numFloats[fileCnkt];
}
A. CUDA Code

```cpp
# A. CUDA Code

cout << "Total Floats: " << totalFloats << endl;

/*@ Host Memory */
printf("Allocate Host Memory.\n");
int *numberOfCoefficients_h;
numberOfCoefficients_h = (int *)malloc(sizeof(int)*voxelSpaceSize*
    numberOfImages);
float *gaussianCoefficients_h;
    gaussianCoefficients_h = (float *)malloc(sizeof(float)*totalFloats);
int *gaussPosition_h;
    gaussPosition_h = (int *)malloc(sizeof(int)*voxelSpaceSize*
        numberOfImages);

for (int k=0;k<numberOfImages;k++)
    
    /* Read in the coefficients */
    int filePosTemp=0;
    for (int j=0;j<k;j++)
    {
        filePosTemp += numFloats[j];
    }
    cout << "FILEPOS: " << filePosTemp << endl;
    loadCoefficients(fn[k],&numberOfCoefficients_h[k*voxelSpaceSize ],
        &gaussianCoefficients_h[filePosTemp]);
    printf("Coefficients Loaded.\n");

    getGaussPosition(&gaussPosition_h[k*voxelSpaceSize ],
        &numberOfCoefficients_h[k*voxelSpaceSize ],filePosTemp);
    printf("Gauss Positions Loaded.\n");
    rewind(fn[k]);
    fread(&h1[0], sizeof(int),3,fn[k]);

    
    
    clock_t start,end;

    /* If the image size is 320x240. The coefficients are stored for a 
    640x480 image. */
    for (int j=0;j<totalFloats;j=j+6)
    {
        if (IMAGE_DIMX == 320)
        {
            gaussianCoefficients_h[j+1] = gaussianCoefficients_h[j+1]/2.0f ;
            gaussianCoefficients_h[j+2] = gaussianCoefficients_h[j+2]/2.0f ;
            gaussianCoefficients_h[j+3] = gaussianCoefficients_h[j+3]/2.0f ;
            gaussianCoefficients_h[j+4] = gaussianCoefficients_h[j+4]/2.0f ;
        }
    }
```
/* The closest voxels to the collimator have unreliable data due to excess pileup. */
/* Set their intensities to zero. */
if (FILTER_PSF == 1)
{
    int counter = 0;
    int singleCounter = 0;
    for (int j = 0; j < numberOfImages; j++)
    {
        int voxelCounter = 0;
        for (int x = 0; x < DIMX; x++)
        {
            for (int y = 0; y < DIMY; y++)
            {
                for (int z = 0; z < DIMZ; z++)
                {
                    if (z >= DIMZ - 7)
                    {
                        counter = 0;
                        for (int k = 0; k < numberOfCoefficients_h[j * voxelSpaceSize + voxelCounter]; k++)
                        {
                            gaussianCoefficients_h[gaussPosition_h[j * voxelSpaceSize + voxelCounter] + counter] = 0.0f;
                            counter = counter + 6;
                        }
                        voxelCounter++;
                        singleCounter++;
                    }
                }
            }
        }
    }
}

/* How many voxels can be processed at a time. 3 GB GPU. */
printf("Allocating to GPU.\n");
int numberOfVoxelsToProcess;
if (IMAGE_DIMX == 640)
{
    numberOfVoxelsToProcess = 1536;
}
else if (IMAGE_DIMX == 320)
{
    numberOfVoxelsToProcess = 6144;
}

float *sensitivityProj;
sensitivityProj = (float *) malloc(numberOfImages * voxelSpaceSize * sizeof(float));
float *sensitivityProj_d;
CUDA_CALL(cudaMalloc((void**)&sensitivityProj_d, sizeof(float) * numberOfImages * voxelSpaceSize));
resetToValue(sensitivityProj, numberOfImages * voxelSpaceSize, 0.0f);
float *detectorSum_h;
detectorSum_h=(float *)malloc(voxelSpaceSize*sizeof(float));
float *detectorSum_d;
CUDA_CALL(cudaMalloc((void**)&detectorSum_d,sizeof(float)*voxelSpaceSize));

/*! Initialize to zero. */
resetToZero(detectorSum_h,voxelSpaceSize);
/*! Copy across to GPU. */
CUDA_CALL(cudAMemcpY(detectorSum_d,detectorSum_h,sizeof(float)*voxelSpaceSize,cudAMemcpYHostToDevice));
cout << "Complete" << endl;

/*! How much space for images is available on the device? */
size_t avail;
size_t total;
cudaMemGetInfo(&avail,&total);
cout << "Total: " << total << "\nAvailable: " << avail << endl;

/*! Declare images on the CPU and GPU. */
float *images_h;
images_h = (float*)malloc(sizeof(float)*numberOfVoxelsToProcess*imageSize);
float *images_d;
CUDA_CALL(cudaMalloc((void**)&images_d,sizeof(float)*numberOfVoxelsToProcess*imageSize));
/*! Initialize to zero. */
resetToZero(images_h,numberOfVoxelsToProcess*imageSize);
/*! Copy across to GPU. */
CUDA_CALL(cudAMemcpY(images_d,images_h,sizeof(float)*numberOfVoxelsToProcess*imageSize,cudAMemcpYHostToDevice));

/*! Much memory left after that transfer? */
cudaMemGetInfo(&avail,&total);
cout << "Total: " << total << "\nAvailable: " << avail << endl;

printf("Begin GPU Processing?: (Write: 1, Process: 0) ");
float *mlem_detectorSum_h;
float *mlem_detectorSum_d;

/*! ALGORITHM */
MLEM = 0;
POSEM = 1;
IMAGE_OSEM = 2; // Not implemented.
LANDWEBER = 3; // Not implemented.
/*

/*! Sensitivity of each voxel. */
totalSensitivity(sensitivityProj,imageNumbers,numFloats,
numberOfImages,numberOfCoefficients_h,gaussianCoefficients_h,
gaussPosition_h,images_h,images_d,imageSize,
numberOfVoxelsToProcess,voxelSpaceSize);
cout << "Exit sensitivity function." << endl;
float *mlem_sensitivityProj;
mlem_detectorSum_h=(float*)malloc(voxelSpaceSize*sizeof(float));
CUDA Code

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

CUDA_CALL(cudaMemcpy(currentPositionEstimate_d, currentPositionEstimate_h, sizeof(float)*voxelSpaceSize, cudaMemcpyHostToDevice));

for (int iterations=estimateOffset; iterations<=stopper; iterations++)
{
    cout << "Iteration: " << iterations << endl;
    start = clock();

    /* Start MLEM.*/

    /* Implementation 1. Basic implementation of MLEM concepts. Mainly used for learning and debugging.*/
    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);

    /* Implementation 2. Uses optimized reduction code and is more memory efficient.*/
    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);

    /* Update estimate.*/
    mlem_updateEstimateCUDA<<<(int)ceil((float)voxelSpaceSize/512.0f),512>>>(currentPositionEstimate_d, mlem_sensitivityProj_d, mlem_detectorSum_d, voxelSpaceSize);

    end = clock();
    cout << "1 Iteration takes: " << (double)(end - start) / CLOCKS_PER_SEC << endl;

    /* Output reconstruction.*/
    if ((iterations == 0) || (iterations == 1) || (iterations == 2) || (iterations == 3) || (iterations % 5 == 0))
    {
        CUDA_CALL(cudaMemcpy(currentPositionEstimate_h, currentPositionEstimate_d, sizeof(float)*voxelSpaceSize, cudaMemcpyDeviceToHost));
        cout << "***Writing. ***" << endl;
        string newFile;
        newFile.append(rootOut);
        newFile.append("reconstruction");
        newFile.append(IntToStr(iterations));
        newFile.append(".dat");
        cout << newFile << endl;
        fileOut = fopen(newFile.c_str(),"wb");
A. CUDA Code

    fwrite(currentPositionEstimate_h, sizeof(float), voxelSpaceSize, fileOut);
    fclose(fileOut);
}

free(numberOfCoefficients_h);
free(gaussianCoefficients_h);
free(detectorSum_h);

if(ALGORITHM == 0) /* mlem */
{
    free(mlem_detectorSum_h);
    cudaFree(mlem_detectorSum_d);
    free(mlem_sensitivityProj);
}

cudaFree(images_d);
cudaFree(detectorSum_d);
free(gaussPosition_h);
free(images_h);
free(g_tmp);
free(H);
free(g);
return 0;

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

/* Get the POSEM image mask. */
void createPosemMask(int *mask, int subsetNumber)
{
    int subsetSize = 4;
    int *miniMask;
    miniMask = (int*)malloc(sizeof(int)*subsetSize*subsetSize);

    /* Below is the order suggested in the POSEM paper. */
    miniMask[0] = 9;
    miniMask[1] = 13;
    miniMask[2] = 1;
    miniMask[3] = 5;
    miniMask[4] = 0;
A. CUDA Code

```c
miniMask[5] = 4;
miniMask[6] = 8;
miniMask[7] = 12;
miniMask[8] = 6;
miniMask[9] = 10;
miniMask[10] = 14;
miniMask[11] = 2;
miniMask[12] = 15;
miniMask[13] = 3;
miniMask[14] = 7;
miniMask[15] = 11;

int numPixX = IMAGE_DIMX/subsetSize;
int numPixY = IMAGE_DIMY/subsetSize;

for (int x=0;x<numPixX;x++)
{
    for (int y=0;y<numPixY;y++)
    {
        for (int i=0;i<subsetSize;i++)
        {
            for (int j=0;j<subsetSize;j++)
            {
                int index = subsetSize*IMAGE_DIMX*y + x*subsetSize + j*
IMAGE_DIMX + i;
                if (miniMask[j*subsetSize + i] == subsetNumber)
                {
                    mask[index] = 1;
                }
            }
        }
    }
}

/*Number of pixels in each POSEM subset.*/
void getNumberOfPixelsInEachSubset(int *mask, int *numberOfPixels, int imageSize)
{
    int count=0;
    for (int i=0;i<16;i++)
    {
        for (int idx=0;idx<imageSize;idx++)
        {
            if (mask[i*imageSize + idx]==1)
                count++;
        }
        numberOfPixels[i] = count;
        count=0;
    }
}

/*Indices of the pixels.*/
void getPixelIndices(int *pixelIndices, int *mask, int imageSize)
{
    int currentIndex = 0;
    int subsetCount = 0;
    for (int i=0;i<16;i++)
    {
        subsetCount++;
        if (mask[i*imageSize + currentIndex] == 1)
            currentIndex++;
    }
    currentIndex = 0;
    for (int i=0;i<16;i++)
    {
        pixelIndices[i] = currentIndex;
        currentIndex++;
    }
    for (int i=imageSize*16;i<imageSize*imageSize;i++)
    {
        pixelIndices[i] = -1;
    }
}
```

---

-165-
A.8 Fourier Crosstalk Matrix

A.8.1 Functions

```c
/* Fourier Crosstalk Matrix */

__global__ void fl2_setToZeroCUDA(float2 *image, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N)
    {
        image[idx].x = 0.0f;
        image[idx].y = 0.0f;
    }
}

void fl2_resetToZero(float2 *H, int num)
{
    for (int i = 0; i < num; i++)
    {
        H[i].x = 0.0f;
        H[i].y = 0.0f;
    }
}

int countActivePixels(int *mask)
{
    int cnt = 0;
    for (int i = 0; i < IMAGE_DIMX * IMAGE_DIMY; i++)
    {
        if (mask[i] == 1)
            cnt++;
    }
    return cnt;
}
```
void convertIndexToXY (int *mask, int2 *coord )
{
    int cnt = 0;
    int index = 0;
    for (int y = 0; y < IMAGE_DIMY; y++)
    {
        for (int x = 0; x < IMAGE_DIMX; x++)
        {
            if (mask[index] == 1)
            {
                coord[cnt].x = x;
                coord[cnt].y = y;
                cnt++;
            }
            index++;
        }
    }
}

_global_ void sumAllFFT (float *total, float2 *image, int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N)
    {
        total[idx] += image[idx].x * image[idx].x + image[idx].y * image[idx].y;
    }
}

_device_ float gaussKernel (float *sharedGaussianCoefficient, float xCent, float yCent)
{
    float a = ( (xCent - (sharedGaussianCoefficient[1]-1.0f)) * (xCent - (sharedGaussianCoefficient[1]-1.0f)) ) / ( (sharedGaussianCoefficient[3])*(sharedGaussianCoefficient[3]));
    float b = -(2.0f*sharedGaussianCoefficient[5]) * (xCent - (sharedGaussianCoefficient[1]-1.0f)) * (yCent - (sharedGaussianCoefficient[2]-1.0f)) / (sharedGaussianCoefficient[3]*sharedGaussianCoefficient[4]);
    float c = ((yCent - (sharedGaussianCoefficient[2]-1.0f)) * (yCent - (sharedGaussianCoefficient[2]-1.0f)) ) / (sharedGaussianCoefficient[4]*sharedGaussianCoefficient[4]);
    float z = a + b + c;
    float tempVal = sharedGaussianCoefficient[0] * ( _expf( -z / (2.0f * (1.0f-(sharedGaussianCoefficient[5]*sharedGaussianCoefficient[5])))));
    return tempVal;
}
```c
__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) {
    int idx = blockIdx.x*blockDim.x+threadIdx.x;
    extern __shared__ float2 sharedCoord[];

    if (threadIdx.x < numberOfPixels) {
        sharedCoord[threadIdx.x].x = (float)coord[threadIdx.x].x;
        sharedCoord[threadIdx.x].y = (float)coord[threadIdx.x].y;
    }__syncthreads();

    if (idx < 61) {
        float x[6];
        int currentIndex = startVoxel + idx;
        int gaussIndex = gaussPosition_d[currentIndex] - gaussPositionOffset;
        int arraySize = 256*256*256;
        for (int i=0;i<numberOfCoefficients_d[currentIndex];i++) {
            for (int counter=0;counter<6;counter++)
                x[counter] = gaussianCoefficients_d[gaussIndex + i*6 + counter];
        }
        for (int j=0;j<numberOfPixels;j++)
            array_d[indexAdj + idx + j*arraySize].x += gaussKernel(x, sharedCoord[j].x, sharedCoord[j].y);
    }
}

void getArrayForFFT(int *numberOfCoefficients_d, float *gaussianCoefficients_d, int *gaussPosition_d, float2 *array_d, int2 *coord, int pixelStart, int numberOfPixels, int gaussPositionOffset) {
    int currentVoxel=0;
    int numberOfVoxelsToProcessTemp = 64;
    int block_size = 32;
    int n_blocks = (int)ceil((float)numberOfVoxelsToProcessTemp/(float)(block_size));

    for (int k=0;k<256;k++) {
        for (int j=0;j<256;j++)
            if (k<61 && j<61)
```
A. CUDA Code

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

A.8.2 Crosstalk Main

```c
int main(void)
{
  size_t avail;
  size_t total;
  cudaMemGetInfo(&avail, &total);
  std::cout << "Total: " << total << "\nAvailable: " << avail << std::endl;

  /*Build the Mask and get the Active Pixels */
  int *mask;
  mask = (int *)malloc(sizeof(int)*IMAGE_DIMX*IMAGE_DIMY);
  i_ResetToValue(mask, IMAGE_DIMX*IMAGE_DIMY, 1);
  i_ApplyCircularMask(mask);
  int activePixels = countActivePixels(mask);
  std::cout << "There are " << activePixels << " active pixels. " << std::endl;

  int2 *coord;
  coord = (int2 *)malloc(sizeof(int2)*activePixels);
  convertIndexToXY(mask, coord);
  std::cout << "Pixel 1: " << coord[0].x << ", " << coord[0].y << std::endl;
  std::cout << "Pixel End: " << coord[activePixels-1].x << ", " << coord[activePixels-1].y << std::endl;

  int2 *coord_d;
  cudaMalloc((void **)&coord_d, sizeof(int2)*activePixels);
  cudaMemcpy(coord_d, coord, sizeof(int2)*activePixels, cudaMemcpyHostToDevice);

  FILE *pixOut;
  pixOut = fopen("E:\TEMP\activePixels.dat", "wb");
  fwrite((coord, sizeof(int2)*activePixels), pixOut);
  fclose(pixOut);

  //Load all the H matrices.
  int numberOfImages = 6;
  int voxelSpaceSize = DIMX*DIMY*DIMZ;
  FILE *fn[6];
```

–169–
A. CUDA Code

```c
/* fn[0] = fopen("E:\PSF\H\gaussParams_00mm_082012_v3_PE_raw_cubic.dat","rb");
 fn[1] = fopen("E:\PSF\H\gaussParams_08mm_082012_v3_PE_raw_cubic.dat","rb");
 fn[2] = fopen("E:\PSF\H\gaussParams_16mm_082012_v3_PE_raw_cubic.dat","rb");
 fn[3] = fopen("E:\PSF\H\gaussParams_24mm_082012_v3_PE_raw_cubic.dat","rb");
 fn[4] = fopen("E:\PSF\H\gaussParams_27mm_082012_v3_PE_raw_cubic.dat","rb");
 fn[5] = fopen("E:\PSF\H\gaussParams_37mm_082012_v3_PE_raw_cubic.dat","rb"); */

fn[0] = fopen("E:\PSF\H\gaussParams_00mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
fn[1] = fopen("E:\PSF\H\gaussParams_08mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
fn[2] = fopen("E:\PSF\H\gaussParams_16mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
fn[3] = fopen("E:\PSF\H\gaussParams_24mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
fn[4] = fopen("E:\PSF\H\gaussParams_27mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");
fn[5] = fopen("E:\PSF\H\gaussParams_37mm_082012_v3_PE_raw_cubic_Halfmm.dat","rb");

printf(" File . . . open\n");
int h1[3];
printf(" File . . . read\n");
for (int fileCnt=0;fileCnt<numberOffImages;fileCnt++)
{
    fread(&h1[0],sizeof(int),3,fn[fileCnt]);
    printf("%d,%d,%d\n",h1[0],h1[1],h1[2]);
}

/* Allocate memory to store all of the coefficients. */
int *numFloats;
numFloats = (int *) malloc(sizeof(int)*numberOffImages);
for (int fileCnt=0;fileCnt<numberOffImages;fileCnt++)
{
    numFloats[fileCnt] = getSize(fn[fileCnt]);
    printf(" File Number: %d, Count: %d\n",fileCnt, numFloats[fileCnt]);
    rewind(fn[fileCnt]);
    fread(&h1[0],sizeof(int),3,fn[fileCnt]);
    printf("%d,%d,%d\n",h1[0],h1[1],h1[2]);
}

int totalFloats = 0;
for (int fileCnt=0;fileCnt<numberOffImages;fileCnt++)
{
    totalFloats += numFloats[fileCnt];
}
std::cout << "Total Floats: " << totalFloats << std::endl;

printf(" Allocate Memory.\n");
int *numberOfCoefficients_h;
```
A. CUDA Code

81  numberOfCoefficients_h = (int *)malloc(sizeof(int)*voxelSpaceSize*numberOfImages);
82  float *gaussianCoefficients_h;
83  gaussianCoefficients_h = (float *)malloc(sizeof(float)*totalFloats*numOfImages);
84  int *gaussPosition_h;
85  gaussPosition_h = (int *)malloc(sizeof(int)*voxelSpaceSize*numberOfImages);
86  cudaMemGetInfo(& avail,& total);
87  std::cout << " Total: " << total << " Available: " << 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  int counter = 0;
120  int singleCounter=0;
121  for (int j=0;j<numberOfImages;j++)
122  {
123    int voxelCounter = 0;
124    for (int x=0;x<DIMX;x++)
125    {
126      
127    }
for (int y=0;y<DIMY;y++)
{
    for (int z=0;z<DIMZ;z++)
    {
        if (z>=DIMZ-4)
        {
            counter = 0;
            for (int k=0;k<numberCoefficients_h[j*voxelSpaceSize+voxelCounter];k++)
            {
                gaussianCoefficients_h[gaussPosition_h[j*voxelSpaceSize+voxelCounter]+counter] = 0.0f;
                counter = counter + 6;
            }
            voxelCounter++;
            singleCounter++;
        }
    }
}

int NX=256,NY=256,NZ=256;
/*Declare the array to hold the results of the mth pixel for every object position.*/
int numberOfPixels=10;
float2 *array1_h;
array1_h = (float2 *)malloc(sizeof(float2)*NX*NY*NZ*numberOfPixels);

/*Declare the variable that will hold the raw data and the FFT. This will get overwritten in every loop.*/
float2 *array1_d;
cudaMalloc((void**)&array1_d, sizeof(float2)*NX*NY*NZ*numberOfPixels);

/*Create the array that will store the sum of 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.*/
float *fftSum_h;
fftSum_h = (float *)malloc(sizeof(float)*NX*NY*NZ);
fl_setToZero(fftSum_h,NX*NY*NZ);

float *fftSum_d;
cudaMalloc((void**)&fftSum_d, sizeof(float)*NX*NY*NZ);
int block_size = 512;
int n_blocks = (int)ceil(float(NX*NY*NZ)/(float(block_size)));
fl_setToZeroCUDA<<<n_blocks , block_size>>>(fftSum_d,NX*NY*NZ);

/* Create a 3D FFT plan. */
cufftHandle plan;
cufftPlan3d(&plan, NX, NY, NZ, CUFFT_C2C);
A. CUDA Code

```c
int *numberOfCoefficients_d;
int *gaussPosition_d;
float *gaussianCoefficients_d;

CUDA_CALL(cudamalloc((void**)&numberOfCoefficients_d,sizeof(int)* voxelSpaceSize));
CUDA_CALL(cudamalloc((void**)&gaussPosition_d,sizeof(int)* voxelSpaceSize));
FILE *fout[6];
fout[0] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_1im.dat","wb");
fout[1] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_2im.dat","wb");
fout[2] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_3im.dat","wb");
fout[3] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_4im.dat","wb");
fout[4] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_5im.dat","wb");
fout[5] = fopen("E:\\TEMP\\cudaFFTtest2_Halfmm_6im_6im.dat","wb");
std::cout << "Files are open and crosstalk is now being calculated ." << std::endl;
for (int imageNumber=0;imageNumber<6;imageNumber++)
{
    cudaMemGetInfo(&avail,&total);
    std::cout << "Total: " << total << "\nAvailable: " << avail << "\n", Required: " << numFloats[imageNumber]*4 << std::endl;
    numberOfPixels = 10;
    int gaussPositionOffset = gaussPosition_h[imageNumber* voxelSpaceSize];
    CUDA_CALL(cudamalloc((void**)&gaussianCoefficients_d,sizeof(float)*numFloats[imageNumber]));
    CUDA_CALL(cudamemcpy(gaussianCoefficients_d,&
    gaussianCoefficients_h[gaussPosition_h[imageNumber*voxelSpaceSize]],&sizeof(float)*numFloats[imageNumber],cudaMemcpyHostToDevice));
    CUDA_CALL(cudamemcpy(numberOfCoefficients_d,&
    numberOfCoefficients_h[imageNumber*voxelSpaceSize],&sizeof(int)*voxelSpaceSize,cudaMemcpyHostToDevice));
    CUDA_CALL(cudamemcpy(gaussPosition_d,&gaussPosition_h[imageNumber* voxelSpaceSize],&sizeof(int)*voxelSpaceSize,cudaMemcpyHostToDevice));
}

syncDeviceAndErrorCheck();
int currentPixelNumber=0;

//How many iterations are required for the whole pixel list?
start = clock();
int loopsRequired = (int)ceil((float)activePixels / (float)numberOfPixels);
block_size = 512;
n_blocks = (int)ceil((float)(NX*NY*NZ)/(float)(block_size)));
for (int repeat=0;repeat<loopsRequired;repeat++)
{
    if (currentPixelNumber + numberOfPixels > activePixels)
    numberOfPixels = activePixels - currentPixelNumber;
    
    /*Set the FFT arrays to zero.*/
    for (int i=0;i<numberOfPixels;i++)
```
A. CUDA Code

```c
fl2_setToZeroCUDA<<<n_blocks, block_size>>>(array1_d[i*Nx*Ny*Nz],Nz*Ny*Nz);

getArrayForFFT(numberOfCoefficients_d, gaussianCoefficients_d, gaussianPosition_d, array1_d, coord_d, currentPixelNumber, numberOfPixels, gaussPositionOffset);

currentPixelNumber+=numberOfPixels;

/* Transform the first signal in place. */
for (int i=0;i<numberOfPixels;i++)
cufftExecC2C(plan, (cufftComplex *)&array1_d[i*Nx*Ny*Nz], (cufftComplex *)&array1_d[i*Nx*Ny*Nz], CUFFT_FORWARD);

/* Need to sum the voxels in the FFTs. */
for (int i=0;i<numberOfPixels;i++)
sumAllFFTs<<<n_blocks, block_size>>>(fftSum_d,&array1_d[i*Nx*Ny*Nz],Nx*Ny*Nz);
}
cudaFree(gaussianCoefficients_d);

CUDA_CALL(cudaMemcpy(fftSum_h,fftSum_d,sizeof(float)*Nx*Ny*Nz,cudaMemcpyDeviceToHost));
fwrite(fftSum_h,sizeof(float),Nx*Ny*Nz,fout[imageNumber]);
fclose(fout[imageNumber]);
end = clock();
std::cout << "Image Number: " << imageNumber << ", Iteration takes: " << (double)(end - start)/CLOCKS_PER_SEC << std::endl;
}

/* Destroy the cuFFT plan. */
cudaFree(fftSum_d);
cudaFree(coord_d);
cufftDestroy(plan);
cudaFree(numberOfCoefficients_d);
cudaFree(gaussianPosition_d);
free(array1_h);
free(fftSum_h);
free(gaussianCoefficients_h);
free(gaussPosition_h);
free(numberOfCoefficients_h);
free(coord);
free(numFloats);
cudaFree(array1_d);
return 0;
```

-174-
APPENDIX B

MATLAB Code

B.1 Main MATLAB Script

```matlab
% 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];
umVoxels = 13*13*13;
dim = [13 13 13];

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

% 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);
```
fileIn = fileOut;

%%This function tracks and sorts the pinholes. It also converts them into coefficients that can be used with [640 480] images.
%%Do this for each of the system configurations. Name them p_0 and p_8 or similar.
p = pinholeTrackingv6(fileIn,dim);

%%Determine pinhole matches between PSF datasets 00mm and 08mm%%
These matches will be used to extrapolate the positions of the pinholes in higher magnification datasets. These positions are then compared to experimentally measured PSFs to sort them.

pinhole = manualPinholeMatching(p0,p8,[13 13 13]);

%%Extrapolate the projection positions in high-magnification data.
filename_Coeff = 'E:\PSF\New Intensifier\6-22-12-24mm\gaussianParams_normalized_24mm.dat';
[p24 distances1] = extrapolateHighMagPinholeCoordinates(
    filename_Coeff,p0,p8,320,24,pinhole,[13 13 13],20);

filename_Coeff = 'E:\PSF\New Intensifier\6-22-12-27mm\gaussianParams_normalized_27mm.dat';
[p27 distances1] = extrapolateHighMagPinholeCoordinates(
    filename_Coeff,p0,p8,320,27,pinhole,[13 13 13],25);

filename_Coeff = 'E:\PSF\New Intensifier\6-22-12-37mm\gaussianParams_normalized_37mm.dat';
[p37 distances2] = extrapolateHighMagPinholeCoordinates(
    filename_Coeff,p0,p8,320,37,pinhole,[13 13 13],35);

%%RUN PSF_weakPinholeDetection here.%%
%%This is a separate .m file, not a function.%%
%%This function will detect weak projections at the edge of the field of view.

%%The outputs of the PSF_weakPinholeDetection function are loaded now.
load 8-20-12-p0mm_PE_v3_complete.mat
pS{1} = p;
load 8-20-12-p8mm_PE_v3_complete.mat
pS{2} = p;
load 8-20-12-p16mm_PE_v3_complete.mat
pS{3} = p;
load 8-20-12-p24mm_PE_v3_complete.mat
pS{4} = p;
load 8-20-12-p27mm_PE_v3_complete.mat
pS{5} = p;
load 8-20-12-p37mm_PE_v3_complete.mat
pS{6} = p;
endings = ['00'; '08'; '16'; '24'; '27'; '37'];
B. MATLAB Code

for n=1:6
    coeffInterp = interpolateinterpWithMask_x3_simple(pS{n},0.1,[-15 15];2.5,0.5,[13 13],'cubic');
    fileout = [ 'E:\PSF\H\gaussParams_' endings(n,:) ' mm_082012_v3_PE_raw_cubic_Halfmm.dat' ];
    writeCoeffToFile(coeffInterp,fileout,size(coeffInterp));
end

%%These files can be used to reconstruct an object using the MLEM or POSEM algorithms.

B.2 MATLAB Functions

B.2.1 Gaussian Fitting

function gaussianFitting(fileIn, fileOut, numVoxels, imSize, padsize, thresholdPeak, thresholdCluster)
    %thresholdCluster is the value that 3 pixels inside a cluster must have to be classified as potential signal and not noise. This should be adjusted depending on the data.
    fn = fopen(fileIn,'r');
    header = fread(fn,3,'int');
    fout = fopen(fileOut,'w');
    fwrite(fout,header,'int');
    for n=1:numVoxels
        image1 = fread(fn,imSize(1)*imSize(2),'float32');
        Gauss_Params = [ ];
        if max(image1(:)) > thresholdPeak
            imTmp = reshape(image1,[imSize(1) imSize(2)]);
            [Gauss_Params res1]= Gaussian2D_master_6params_Simple_v3(imTmp,padsize,thresholdCluster);
        end
        fwrite(fout,size(Gauss_Params,2),'int');
        if size(Gauss_Params,2) > 0
            fwrite(fout,Gauss_Params,'float32');
        end
    end
    fclose(fn);
    fclose(fout);
end

% Gaussian2D_master_6params_Simple_v3.m – To fit a 2-D gaussian.
# B. MATLAB Code

```matlab
function [Gauss_Params residuals] = Gaussian2D_master_6params_Simple_v3(imTmp, padsize, threshold)
% padsize must be an even number!
residuals = []; fval = [];
offsetNeg = padsize / 2;
offsetPos = padsize / 2 - 1;
imTmp = imTmp';
immed = imTmp;
immed(immed < threshold) = 0;
imTmp1 = padarray(imTmp, [padsize padsize]);
imTmp1med = padarray(immed, [padsize padsize]);
clear immed imTmp
cc = bwconncomp(imTmp1med);
% Found a bug in this code when a cluster has values below the threshold in
% the middle of the blob so it splits it in two. This happened for a
% single cluster out of thousands.
% Need to get the distances between all the blobs found in the
% connected components and concatenate the pixelIdxList from both clusters.
cc = removeDuplicateClusters(cc, 15, size(imTmp1med));
numPixels = cellfun(@(x) size(x, 1), cc.PixelIdxList);
ind = find(numPixels >= 3);
counter = 1;
for p = 1:size(ind, 2)
    currentPixel = cc.PixelIdxList{ind(p)};
    [R C] = ind2sub(size(imTmp1med), currentPixel(1));
    pixel = [R(1) C(1)];
    g = imTmp1(pixel(1)-offsetNeg:pixel(1)+offsetPos, pixel(2)-offsetNeg:pixel(2)+offsetPos);
    gMed = imTmp1med(pixel(1)-offsetNeg:pixel(1)+offsetPos, pixel(2)-offsetNeg:pixel(2)+offsetPos);
    % Just in case g is only non-zero in a section of Nx1
    % pixels. (This would break 'moments.m')
    if sum(sum(g, 1) > 0) < 2 || sum(sum(g, 2) > 0) < 2
        g(round(size(g, 1)/2), round(size(g, 2)/2)) = g(round(size(g, 1)/2), round(size(g, 2)/2)) + .1;
        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;
```

---

-178-
B. MATLAB Code

```matlab
[mx, my, var_x, var_y, covar_xy, ~, ~, ~, ~] = moments(g);
mx = mx+pixel(2)-(offsetNeg + 1);
my = my+pixel(1)-(offsetNeg + 1);
dim_x=7;
dim_y=7;
mxT = floor(mx + 0.5);
myT = floor(my + 0.5);
g = imTemp1(myT-dim_y:myT+dim_y,mxT-dim_x:mxT+dim_x);
gTemp = g;

gTemp(g<=2) = 0;
gTemp(gTemp<=(max(gTemp(:))*0.1)) = 0;
[r c] = find(gTemp == max(gTemp(:)));
mask = 1*bwselect(gTemp,c(1),r(1),8);
mask = increaseBoundary(mask,sqrt(2));
g = g.*mask;

if sum(sum(g,1)>0) < 2 || sum((sum(g,2)>0)) < 2
    g(r+1,c) = g(r+1,c) + 0.01;
g(r,c+1) = g(r,c+1) + 0.01;
g(r-1,c) = g(r-1,c) + 0.01;
g(r,c-1) = g(r,c-1) + 0.01;
g(r,c) = g(r,c) + 0.01;
end
clear mask c r

[sx, sy, var_x, var_y, covar_xy, ~, ~, ~, ~] = moments(g);
sx = sqrt(var_x);
sy = sqrt(var_y);
rho_xy = covar_xy/sx/sy;
max_pixel = max(g(:));

offset_percentage = 0.5;
A_lb = 1e-10;
A_ub = 1000*max_pixel;
xm_lb = mx-abs(mx+offset_percentage);
xm_ub = mx+abs(mx+offset_percentage);
ym_lb = my-abs(my+offset_percentage);
ym_ub = my+abs(my+offset_percentage);
sx_lb = sx-abs(sx+offset_percentage);
sx_ub = sx+10*abs(sx+offset_percentage);
if sx_ub < 5
    sx_ub = 5;
end
sy_lb = sy-abs(sy+offset_percentage);
sy_ub = sy+10*abs(sy+offset_percentage);
if sy_ub < 5
    sy_ub = 5;
end
```

% Upper and Lower Bounds with fmincon

```matlab
```

–179–
B. MATLAB Code

```matlab
x = [max_pixel mx my sx sy rho_xy];

if sx_lb<=0 || sx_lb > 1
    sx_lb=0.1;
end
if sy_lb<=0 || sy_lb > 1
    sy_lb=0.1;
end

rho_xy_lb=-0.99;
rho_xy_ub=0.99;

if(sx == 0 || sy == 0 || max_pixel == 0)
    continue;
end

sum_prev=sum(g(:));
dim_y_old = dim_y;
dim_x_old = dim_x;

dim_y = size(g,1);
dim_x = size(g,2);

g=g(:);

[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', 10000, 'MaxIter', 10000, 'Display', 'notify-detailed'),g,dim_y, dim_x);

Gauss_Params(2,counter) = Gauss_Params(2,counter) + mxT-(dim_x_old +1);
Gauss_Params(3,counter) = Gauss_Params(3,counter) + myT-(dim_y_old +1);

%%Have limits been reached?
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
    disp('Limits have been detected');
    Gauss_Params(:,counter)
end
if sum(isreal(Gauss_Params(:))) == 0
    disp('ERROR: Complex Number Detected');
end

%%Renormalize the gaussian.
%%Analytical:
```

---180---
% sum_post = Gauss_Params(1, counter) * 2 * pi * Gauss_Params(4, counter) *
  Gauss_Params(5, counter) * sqrt(1 - Gauss_Params(6, counter)^2);

% Numerical:
sum_post = sum(sum(generatePSFfromGaussianCoefficients([*
  Gauss_Params(2, counter) - padsize); (Gauss_Params(3, 
  counter) - padsize); ... 
  Gauss_Params(4, counter); Gauss_Params(5, counter); ... 
  Gauss_Params(6, counter), 2, 2));

Gauss_Params(1, counter) = Gauss_Params(1, counter) / sum_post * sum_pre 
   ;
   imTmp1med(myT-offsetNeg:myT+offsetPos,mT-offsetNeg:mT+ 
offsetPos) = 0;
   counter = counter + 1;
end
residuals = fval;

type = exist('Gauss_Params');
if type == 1
  Gauss_Params(2,:) = Gauss_Params(2,:) - padsize ;
  Gauss_Params(3,:) = Gauss_Params(3,:) - padsize ;
  %Remove NaNs caused by insufficient data.
  ind = isnan(Gauss_Params(2,:));
  ind = find(ind == 1);
  Gauss_Params(:,ind) = [];
else
  Gauss_Params = [];
end

%%%%%%%%%%%%%%%%%%%%%%%%%% GaussFit.m
%%%%%%%%%%%%%%%%%%%%%%%%%%
function LS = GaussFit(x,g,dim_y, dim_x)

% A = x(1)
% mx = x(2);
% my = x(3);
% sigma_x = x(4);
% sigma_y = x(5);
% rho_xy = x(6)

[Y,X] = meshgrid(1:dim_y,1:dim_x);
X = X(:);
Y = Y(:);

% 2D Gaussian
a = ((X-mx(2)).^2)/(x(4).^2);
b = -(2.*x(6)).*(X-mx(2)).*(Y-my(3)))./(x(4).*x(5));
c = ((Y-my(3)).^2)/(x(5).^2);
\[
z = a + b + c;
\]
\[
gm\_bar = x(1) \cdot \exp(-z/(2*(1-x(6)))^2);
\]
\[
gm\_bar = \text{reshape}(gm\_bar, \text{dim}\_x, \text{dim}\_y)';
\]
\[
gm\_bar = gm\_bar(:, :);
\]
\[
g\_new = g;
\]
\[
gm\_bar\_new = gm\_bar;
\]
\[
\% This blocks against very low values causing a large value in the following equations.
\]
\[
\text{ind} = \text{find}(gm\_bar\_new >= 1/(2*\pi));
\]
\[
\text{if size}(\text{ind}, 1) > 0
\]
\[
LS = \text{sum}(0.5*\log(2*\pi*(gm\_bar\_new(\text{ind}(:)) + (g\_new(\text{ind}(:)) - gm\_bar\_new(\text{ind}(:))).^2)/(2*gm\_bar\_new(\text{ind}(:)))))
\]
\[
\text{else}
\]
\[
LS = 10^6;
\]
\[
\text{end}
\]
\[
\text{ind} = \text{find}(gm\_bar\_new < 1/(2*\pi));
\]
\[
\text{if size}(\text{ind}, 1) > 0
\]
\[
LS = LS + \text{sum}((g\_new(\text{ind}(:)) - gm\_bar\_new(\text{ind}(:))).^2);
\]
\[
\text{end}
\]

\[
\% Moments.m - Brian W. Miller
\]
\[
\% function [mean\_x, mean\_y, var\_x, var\_y, covar\_xy, skew\_x, skew\_y, kurt\_x, kurt\_y, eccentricity] = moments(clust)
\]
\[
\% Local Centroid
\]
\[
\text{area} = \text{sum}(\text{sum}(\text{clust}));
\]
\[
[\text{rows}, \text{cols}] = \text{size}(\text{clust});
\]
\[
\text{x} = \text{ones}(\text{rows}, 1)*[1: \text{cols}]; \quad \% \text{Matrix with each pixel set to its } x \text{ coordinate}
\]
\[
\text{y} = [1: \text{rows}]' * \text{ones}(1, \text{cols}); \quad \% " " " " " " " " \\
\text{mean}\_x = \text{sum}(\text{sum}(\text{clust}.*\text{x}))/\text{area};
\]
\[
\text{mean}\_y = \text{sum}(\text{sum}(\text{clust}.*\text{y}))/\text{area};
\]
\[
\% 2nd Central Moment - Variance
\]
\[
\text{m00} = \text{sum}(\text{sum}(\text{clust}.*(\text{x}-\text{mean}\_x).^2))/\text{area};
\]
\[
\text{m02} = \text{sum}(\text{sum}(\text{clust}.*(\text{y}-\text{mean}\_y).^2))/\text{area};
\]
\[
\text{m11} = \text{sum}(\text{sum}(\text{clust}.*(\text{y}-\text{mean}\_y).*(\text{x}-\text{mean}\_x)))/\text{area};
\]
\[
\text{var}\_x = \text{m00};
\]
\[
\text{var}\_y = \text{m02};
\]
\[
\text{covar}\_xy = \text{m11};
\]
\[
\% 3rd Central Moment
\]
\[
\text{m03} = \text{sum}(\text{sum}(\text{clust}.*(\text{y}-\text{mean}\_y).^3))/(\text{area} * \text{m00}^3 / \text{m02}^3);
\]
\[
\text{m03} = \text{sum}(\text{sum}(\text{clust}.*(\text{y}-\text{mean}\_y).^3))/(\text{area} * \text{m02}^3 / \text{m02}^3);
\]
m21 = sum(sum(clust.*(x-mean_x).*(y-mean_y)));  

m12 = sum(sum(clust.*(x-mean_x).*(y-mean_y)));  

skew_x = m30;  

skew_y = m03;  

% 4th Central Moment  

m40 = sum(sum(clust.*(x-mean_x).^4))/(area*m20^2) - 3;  

m04 = sum(sum(clust.*(y-mean_y).^4))/(area*m02^2) - 3;  

m31 = sum(sum(clust.*(x-mean_x).^3.*(y-mean_y)));  

m13 = sum(sum(clust.*(y-mean_y).^3.*(x-mean_x)));  

kurt_x = m40;  

kurt_y = m04;  

% Eccentricity  

eccentricity = ((m02-m02)^2 + 4*m11*m11);  

% ** Interpretation of spatial moments **  

% * order 0 = TOTAL MASS [units: concentration, density, etc.]  
% * order 1 = location of CENTRE OF MASS in x and y from 0,0 [units: L]  
% * order 2 = VARIANCE around centroid in x and y [units: L^2]  
% * order 3 = coeff. of skewNESS (symmetry) in x and y [units: n/a]  
% ---> =0 : SYMMETRIC distribution  
% ---> <0 : Distribution asymmetric to the LEFT  
% ---> >0 : Distribution asymmetric to the RIGHT  
% * order 4 = KURTOSIS (flatness) in x and y [units: n/a]  
% ---> =0 : Gaussian (NORMAL) distribution  
% ---> <0 : Distribution FLATTER than normal  
% ---> >0 : Distribution MORE PEAKED than normal  
% ---> <-1.2: BclustODAL (or multiclustodal) distribution  

% ** Parameters derived from 2nd moments ** (from Awcock (1995) "Applied clustage Processing")  

% * ELONGATION (ECCENTRICITY) = Ratio of longest to shortest  
% distance vectors rom the object’s centroid to its boundaries  
% * ORIENTATION = For elongated objects, describes the  
% orientation (in degrees) of the ”long” direction with  
% respect to horizontal (x axis)  

function normalizeGaussian(fileIn, fileOut, numVoxels)  

fn = fopen(fileIn, ’r’);  
header = fread(fn, 3, ’int’);  
fout = fopen(fileOut, ’w’);  
fwrite(fout, header, ’int’);  

for i=1:numVoxels(3)
for j=1:numVoxels(2)
    for k=1:numVoxels(1)
        numberOfCoefficients = fread(fn,1,'int');
        Gauss_Params = fread(fn,[6 numberOfCoefficients],'float32');
        if size(Gauss_Params,2)>0
            switch k
                case 1
                    factor = 140;
                case 2
                    factor = 130;
                case 3
                    factor = 120;
                case 4
                    factor = 110;
                case 5
                    factor = 100;
                case 6
                    factor = 90;
                case 7
                    factor = 80;
                case 8
                    factor = 70;
                case 9
                    factor = 60;
                case 10
                    factor = 50;
                case 11
                    factor = 40;
                case 12
                    factor = 30;
                case 13
                    factor = 20;
            end
            Gauss_Params(1,:) = Gauss_Params(1,:) * 100 / factor;
        end
        fwrite(fout,size(Gauss_Params,2),'int');
        if size(Gauss_Params,2) > 0
            fwrite(fout,Gauss_Params,'float32');
        end
    end
end
fclose(fn);
fclose(fout);

B.2.2 Pinhole Tracking
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}{2,:})
                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
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];
ml = coef2{k,j,i};

for n=1:size(ml,2)
p(k,j,i,n).value = ml(1,n);
p(k,j,i,n).posX = ml(2,n);
p(k,j,i,n).posY = ml(3,n);
p(k,j,i,n).sx = ml(4,n);
p(k,j,i,n).sy = ml(5,n);
p(k,j,i,n).tilt = ml(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!
%% This step uses 'inpaintn.m', although any extrapolation code could work here.
%% Written by: Damien Garcia.
%% Source: Mathworks File Exchange
%% INPAINTN Inpaint over missing data in N-D array
%% Y = INPAINTN(X) replaces the missing data in X by extra/interpolating
%% the non-missing elements. The non finite values (NaN or Inf) in X are
%% considered as missing data. X can be any N-D array.

Important note:

%% INPAINTN uses an iterative process that converges toward the solution.
%% Y = INPAINTN(X,N) uses N iterations. By default, N = 100. If you estimate that INPAINTN did not totally converge, increase N:
%% Y = INPAINTN(X,1000);

for pinNumber = 1:100
    clear valueI posXI posYI sxI syI tI tiltI posX0 posY0 doneX

    [posX0 posY0 doneX] = getVoxelPinholeMap(p,pinNumber,dim);
    if sum(isnan(posX0(:))) < dim(1)*dim(2)*dim(3);
        clear match willPaint
        %Find the indices of all voxels that are adjacent to a non-empty voxel.
        match = findNearestIndicesForFixer(doneX,2*sqrt(2));
        willPaint = fillDone(match,doneX);
        posX0 = inpaintn(posX0,100).*willPaint;
        posY0 = inpaintn(posY0,100).*willPaint;
    end

    projectedX(:,:,pinNumber) = posX0;
    projectedY(:,:,pinNumber) = posY0;
end

[k2 j2 i2] = getNextVoxelToProcess(done,start,dim);
m2 = coef2{k2,j2,i2};
thresh = 20;
newPin = 1;
for count = 1:size(m2,2)
    clear dist;
    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]);
    ind = find(dist == min(dist));
    if dist(ind) < thresh
        p(k2,j2,i2,ind).value = m2(1,count);
        p(k2,j2,i2,ind).posX = m2(2,count);
        p(k2,j2,i2,ind).posY = m2(3,count);
    end
end
B. MATLAB Code

```matlab
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     p(k2, j2, i2, emptyPinhole).sx = m2(4, count);
118     p(k2, j2, i2, emptyPinhole).sy = m2(5, count);
119     p(k2, j2, i2, emptyPinhole).tilt = m2(6, count);
120   end
121   end
122   done(k2, j2, i2) = 1;
123 end
124 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
125 % Manual Pinhole Matching
126 % This function loads 2 images for the user to match projections
127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
128 function pinhole = manualPinholeMatching(p0, p8, dim)
129   %Match the 00mm with 08mm data set.
130   %Need to match up the pinhole numbers.
131   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
132   donePin = zeros(dim(1), dim(2), dim(3), 100);
133   for i = 1:dim(3)
134     for j = 1:dim(2)
135       for k = 1:dim(1)
136         for p = 1:100
137           if isempty(p0(k, j, i, p).value) ~= 1
138             donePin(k, j, i, p) = 1;
139           end
140         end
141       end
142     end
143   end
144   done = ones(dim(1), dim(2), dim(3));
145   index = [1 floor(dim(2)/2 + 0.5) floor(dim(3)/2 + 0.5)];
146   for p = 1:60
147     if i == 7; j == 7; k == 1;
148       % Find a pinhole projection that still hasn’t been associated.
149   end
```

–187–
Are there any pinholes that haven’t been indexed yet?
This is recorded by the donePin array.
If it is 1, it means that this pinhole has not been indexed.
If donePin(k,j,i,1:100) is all 0, then it means that every pinhole for this voxel has been indexed.

for i = 1:dim(3)
    for j = 1:dim(2)
        for k = 1:dim(1)
            cnt = 0;
            for n = 1:100
                if donePin(k,j,i,n) == 1
                    This pinhole has not been associated!
                    done(k,j,i) = 0;
                    cnt = 0;
                else
                    cnt = cnt + 1;
                end
            end
            if cnt == 100
                done(k,j,i) = 1;
            else
                done(k,j,i) = 0;
            end
        end
    end
end

if exist('index') == 1
    Do I need to change image?
    Default is Yes!
    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.

    change = 1;
    for n = 1:100
        if donePin(index(1),index(2),index(3),n) == 1
            change = 0;
        end
    end

    If we need to change the image.
    Find the nearest voxel with a pinhole that hasn’t been associated.
    if change == 1
        number = 0;
        index = zeros(1,3);
        distanceFromCenter = 35;
        clear index
        for i2 = 1:dim(3)
            for j2 = 1:dim(2)
                for k2 = 1:dim(1)
                    if done(k2,j2,i2) <= 1
                        [k j i d] = findNearestIndices(done,k2,j2,i2,1,1.6);
B. MATLAB Code

```matlab
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
distanceFromCenter = tempD;
216 number = size(k,2);
217 index = [k2 j2 i2];
218 end
219 end
220 end
221 end
222 end
223 if exist('index') == 1
224 k = index(1);
225 j = index(2);
226 i = index(3);
227 if done(k,j,i) == 0
228 ktemp = k; jtemp = j; itemp = i;
229 accept = 0;
230 while accept == 0
231 im0 = zeros(640,480);
232 [ktemp jtemp itemp] for n=1:100
233 if isempty(p0(ktemp,jtemp,itemp,n).value) ~= 1
234 clear c
235 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];
236 im = generatePSFfromGaussianCoefficients(c(:,2,1);
237 im0 = im0 + im;
238 [y1 x1] = gptps(h1);
239 im0 = im0 + im;
240 end
241 im8 = zeros(640,480);
242 for n=1:100
243 if isempty(p8(k,j,i,n).value) ~= 1
244 clear c
245 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];
246 im = generatePSFfromGaussianCoefficients(c(:,2,1);
247 im8 = im8 + im;
248 end
249 end
250 h1 = subplot(1,2,1), imagesc(im0);
251 h2 = subplot(1,2,2), imagesc(im8);
252 [y2 x2] = gptps(h2);
253 [y1 x1] = gptps(h1);
```
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.

```matlab
if (size(x2,1) == 1)
    accept=1;
end
if size(x2,1) == 2
    ktemp=ktemp-1;
    ktemp = checkForOutOfBounds(ktemp,1,dim(1));
elseif size(x2,1) == 3
    ktemp=ktemp+1;
    ktemp = checkForOutOfBounds(ktemp,1,dim(1));
elseif size(x2,1) == 4
    jtemp=jtemp-1;
    jtemp = checkForOutOfBounds(jtemp,1,dim(2));
elseif size(x2,1) == 5
    jtemp=jtemp+1;
    jtemp = checkForOutOfBounds(jtemp,1,dim(2));
elseif size(x2,1) == 6
    itemp=itemp-1;
    itemp = checkForOutOfBounds(itemp,1,dim(3));
elseif size(x2,1) == 7
    itemp=itemp+1;
    itemp = checkForOutOfBounds(itemp,1,dim(3));
elseif size(x1,1) > 1
    %repeat.
end

%Find the closest pinhole associated with each of these.
dist1(1:100) = NaN;
dist2(1:100) = NaN;
for m=1:100
    if isempty(p0(ktemp,jtemp,itemp,m).value) ~= 1
        dist1(m) = sqrt(((x1 - p0(ktemp,jtemp,itemp,m).posX)^2 + (y1 - p0(ktemp,jtemp,itemp,m).posY)^2));
    end
    if isempty(p8(k,j,i,m).value) ~= 1
        dist2(m) = sqrt(((x2 - p8(k,j,i,m).posX)^2 + (y2 - p8(k,j,i,m).posY)^2));
    end
end

%Find the minimum distance in p0
ind1 = find(dist1 == min(dist1(:)));
ind2 = find(dist2 == min(dist2(:)));
pinhole(p,1:2) = [ind1 ind2];

%Remove the projections of these pinholes in both p0 and p8 but keep them slightly visible to allow association by a person.
for i=1:dim(3)
```

```matlab
%```
B. MATLAB Code

for j = 1:dim(2)
    for k = 1:dim(1)
        donePin(k, j, i, ind2) = 0;
        if isempty(p0(k, j, i, ind1).value) == 1
            p0(k, j, i, ind1).value = 10;
            p0(k, j, i, ind1).sx = 1;
            p0(k, j, i, ind1).sy = 1;
        end
        if isempty(p8(k, j, i, ind2).value) == 1
            p8(k, j, i, ind2).value = 10;
            p8(k, j, i, ind2).sx = 1;
            p8(k, j, i, ind2).sy = 1;
        end
    end
end
end
end
end
end

B.2.3 Extrapolation of Pinhole Projections

function [new_pDistance distances] =
    extrapolateHighMagPinholeCoordinates(filename_Coeff, p0, p8, xSize,
                                            distance_PD, pinhole, dim, threshold)
new_p0(dim(1), dim(2), dim(3), 100) = struct('value', [], 'posX', [], 'posY', [], 'sx', [], 'sy', [], 'tilt', []);
new_p8(dim(1), dim(2), dim(3), 100) = struct('value', [], 'posX', [], 'posY', [], 'sx', [], 'sy', [], 'tilt', []);
new_pDistance(dim(1), dim(2), dim(3), 100) = struct('value', [], 'posX', [], 'posY', [], 'sx', [], 'sy', [], 'tilt', []);
for i = 1:dim(3)
    for j = 1:dim(2)
        for k = 1:dim(1)
            for p = 1:size(pinhole, 1)
                new_p0(k, j, i, p) = p_0(k, j, i, pinhole(p, 1));
                new_p8(k, j, i, p) = p_8(k, j, i, pinhole(p, 2));
            end
        end
    end
end
projectedX = zeros(dim(1), dim(2), dim(3), 100);
projectedY = zeros(dim(1), dim(2), dim(3), 100);
coef = getCoefficients(filename_Coeff, dim);
coef2 = coef;
if xSize == 320
    for i = 1:dim(3)
        for j = 1:dim(2)
            for k = 1:dim(1)
                if isempty(coef2{k,j,i})
                    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

for pinNumber = 1:100
    clear valueI posXI posYI sxI syI tiltI

    % Load in all the information about a particular pinhole for 00mm and 08mm
    [posX0 posY0 doneY] = getVoxelPinholeMap(new_p0,pinNumber,dim);
    [posX8 posY8 doneX] = getVoxelPinholeMap(new_p8,pinNumber,dim);

    if sum(isnan(posX8(:))) < dim(1)*dim(2)*dim(3) - 1;
        clear match
        % Find all voxels that are adjacent to a non-empty voxel
        match = findNearestIndicesForFixer(doneX,3*sqrt(2));
        willPaint = fillDone(match,doneX);
        posX0 = inpaintn(posX0,1000).*willPaint;
        posY0 = inpaintn(posY0,1000).*willPaint;
        posX8 = inpaintn(posX8,1000).*willPaint;
        posY8 = inpaintn(posY8,1000).*willPaint;
        xx = 0:1;
        for i = 1:dim(3)
            for j = 1:dim(2)
                for k = 1:dim(1)
                    if distance_PD > 8
                        if isnan(posX0(k,j,i)) \= 1 \&\& isnan(posX8(k,j,i)) \= 1
                            a(1) = posX0(k,j,i);
                            a(2) = posX8(k,j,i);
                            projectedX(k,j,i,pinNumber) = interp1(xx,a,(distance_PD/8),’linear’,’extrap’);
                            a(1) = posY0(k,j,i);
                            a(2) = posY8(k,j,i);
                            projectedY(k,j,i,pinNumber) = interp1(xx,a,(distance_PD/8),’linear’,’extrap’);
                        end
                        else
                            if isnan(posX0(k,j,i)) \= 1
                                projectedX(k,j,i,pinNumber) = posX0(k,j,i);
                            end
                    end
                end
            end
        end
    end

end
B. MATLAB Code

```matlab
projectedY(k, j, i, pinNumber) = posY0(k, j, i);
end
end
end
end
end
end
end
distances = zeros(dim(1),dim(2),dim(3),100);
cnt = 1;
for i = 1:dim(3)
    for j = 1:dim(2)
        for k = 1:dim(1)
            m2 = coef2{k, j, i};
            num = 0;
            for count = 1:size(m2, 2)
                clear dist;
                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]);
                ind = find(dist == min(dist));
                distances(k, j, i, ind) = dist(ind);
            end
        end
    end
end

% This projection probably belongs to this pinhole!
if dist(ind) < threshold
    if isempty(new_pDistance(k, j, i, ind).value) == 1
        new_pDistance(k, j, i, ind).value = m2(1, count);
        new_pDistance(k, j, i, ind).posX = m2(2, count);
        new_pDistance(k, j, i, ind).posY = m2(3, count);
        new_pDistance(k, j, i, ind).sx = m2(4, count);
        new_pDistance(k, j, i, ind).sy = m2(5, count);
        new_pDistance(k, j, i, ind).tilt = m2(6, count);
    end
else
    disp('warning');
    [k j i]
end
end
end
end
end
end
end
end
end
```

%% Extrapolate Weak Pinhole Projections

%%

%% This function uses previous data which matched strong pinhole projections.
%% This data is processed using the 'Master' m file.
%Traditional code formatting is relaxed here due to the number of 'for loops' in this function.

clear
for systemSetup = 1:6
    clear
    switch systemSetup
        case 1
            %Read in the old data which is perfectly matched. Associate it with the new data.
            load 8-20-12-p0mm_PE_v1_complete.mat
        case 2
            load 8-20-12-p8mm_PE_v1_complete.mat
        case 3
            load 8-20-12-p16mm_PE_v1_complete.mat
        case 4
            load 8-20-12-p24mm_PE_v1_complete.mat
        case 5
            load 8-20-12-p27mm_PE_v1_complete.mat
        case 6
            load 8-20-12-p37mm_PE_v1_complete.mat
    end
for repeat = 1:6
    switch systemSetup
        case 1
            fn = fopen('E:\PSF\New Intensifier\6-22-12-00mm\centroids_00mm.dat', 'r');
            h = fread(fn, 3, 'int');
            fileOut = ['E:\PSF\New Intensifier\6-22-12-00mm', ' gaussianParams_normalized_00mm_PE_v2_complete.dat'];
        case 2
            fn = fopen('E:\PSF\New Intensifier\6-22-12-08mm\centroids_08mm.dat', 'r');
            h = fread(fn, 3, 'int');
            fileOut = ['E:\PSF\New Intensifier\6-22-12-08mm', ' gaussianParams_normalized_08mm_PE_v2_complete.dat'];
        case 3
            fn = fopen('E:\PSF\New Intensifier\6-22-12-16mm\centroids_16mm.dat', 'r');
            h = fread(fn, 3, 'int');
            fileOut = ['E:\PSF\New Intensifier\6-22-12-16mm', ' gaussianParams_normalized_16mm_PE_v2_complete.dat'];
        case 4
            fn = fopen('E:\PSF\New Intensifier\6-22-12-24mm\centroids_24mm.dat', 'r');
            h = fread(fn, 3, 'int');
            fileOut = ['E:\PSF\New Intensifier\6-22-12-24mm', ' gaussianParams_normalized_24mm_PE_v2_complete.dat'];
        case 5
            fn = fopen('E:\PSF\New Intensifier\6-22-12-27mm\centroids_27mm.dat', 'r');
            h = fread(fn, 3, 'int');
            fileOut = ['E:\PSF\New Intensifier\6-22-12-27mm', ' gaussianParams_normalized_27mm_PE_v2_complete.dat'];
        case 6
            % Add cases for other system setups if needed
    end
fn = fopen( 'E:\PSF\New Intensifier\6-22-12-37mm\centroids_37mm.dat' , 'r');
h = fread(fn,3,'int');
fileOut = [ 'E:\PSF\New Intensifier\6-22-12-37mm\', gaussianParams_normalized_37mm_PE_v2_complete.dat ];

end

% Load in all the data and predict pinhole locations!
% Declare the Variables.
clear pos* sx sy posX* posY* sx*
posXAll{100} = [];
posYAll{100} = [];
sxAll{100} = [];
syAll{100} = [];

% Isolate each pinhole. Inpaint the data.
exists = zeros(13,13,13,100);
for pinhole=1:100
    posX = NaN(13,13,13);
    posY = NaN(13,13,13);
    sx = NaN(13,13,13);
    sy = NaN(13,13,13);
    accept = 0;
    for i=1:13
        for j=1:13
            for k=1:13
                if isempty(p(k,j,i,pinhole).value)~=1
                    posX(k,j,i) = p(k,j,i,pinhole).posX;
                    posY(k,j,i) = p(k,j,i,pinhole).posY;
                    sx(k,j,i) = p(k,j,i,pinhole).sx;
                    sy(k,j,i) = p(k,j,i,pinhole).sy;
                    exists(k,j,i,pinhole) = 1;
                    accept = 1;
                end
            end
        end
    end
if accept == 1
    posX = fillVolBox(posX);
    posY = fillVolBox(posY);
    sx = fillVolBox(sx);
    sy = fillVolBox(sy);

    [posX0 posY0 doneX] = getVoxelPinholeMap(p,pinhole,[13 13 13])
    match = findNearestIndicesForFixer(doneX,sqrt(2));
    willPaint = fillDone(match,doneX);
    posX = posX.*willPaint;
    posY = posY.*willPaint;
    sx = sx.*willPaint;
    sy = sy.*willPaint;
posXAll{pinhole} = posX;
posYAll{pinhole} = posY;

end
sxAll{pinhole} = sx;
syAll{pinhole} = sy;
end
end
clear a* A* G* done* match posX posX0 posY posY0 sx sy vol*

All the pinhole locations are now predicted.
Loop through all the centroid images.
Check all the predicted locations.
Have they already been accounted for?
Yes? – move on.
No? – fit a Gaussian to the area.
cnt = 1;
for i = 1:13
  for j = 1:13
    for k = 1:13
      imR = reshape(fread(fn, 320*240, 'float32'), [320 240]);
      imR = imR';
      imR2 = imR;

      Remove all already known pinholes from the image.
      for pin = 1:50
        if exists(k, j, i, pin) == 1
          This pinhole projection from this voxel has already
          been recorded.
        posX = posXAll{pin};
        posY = posYAll{pin};
        sx = sxAll{pin};
        sy = syAll{pin};
        if sum(isnan(posX(:))) ~= numel(posX)
          posY(k, j, i) = 0
          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;
        end
      end
    end
  end
end

All the known pinhole projections have now been removed so
anything left is either noise or a weak source.
for pin = 1:50
  if exists(k, j, i, pin) == 1
    Do nothing here.
  else
    The pinhole projection has not been recorded. It may
    be very faint or it may not be present.
  end
  posX = posXAll{pin};
  posY = posYAll{pin};
  sx = sxAll{pin};
sx = sx / 2;
sy = syAll{pin};
if sum(isnan(posX(:))) ~= numel(posX) && posY(k,j,i)~= 0
    \%Extract the neighborhood as predicted.
    \%Check if neighborhood bounds will be outside the \%image.
    padsize = 8;
    offsetNeg = padsize / 2;
    offsetPos = padsize / 2 - 1;
    skip = 0;
    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)
        offsetNeg = 6;
        offsetPos = 5;
    end
    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)
        skip = 1;
    end
    if skip == 0
        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));
        \%This neighborhood will most likely be all zeros or \%have some noise. It may also have some weak signal.
        gTemp = neighborhood;
        gTemp(gTemp<=2) = 0;
        gTemp(gTemp<=round(max(gTemp(:))*0.1)) = 0;
        [r c] = find(gTemp == max(gTemp(:)));
        if size(r,1) < 10 \%Probably an image with central point. \%If there are multiple max values, which is the most appropriate
            \%Location to pick? \textbf{\small-> Select the indice with the largest cluster value surrounding it.}
            bestInd = bestMax(imR,r,c,posY(k,j,i),posX(k,j,i),offsetNeg);
            pixel = [floor(posY(k,j,i)-offsetNeg + 0.5)+r(bestInd)-1 floor(posX(k,j,i)-offsetNeg + 0.5)+c(bestInd)-1];
            gTemp = imR(pixel(1)-offsetNeg:pixel(1)+offsetPos,pixel(2)-offsetNeg:pixel(2)+offsetPos);
            gTemp(gTemp<=2) = 0;
            gTemp(gTemp<=round(max(gTemp(:))*0.1)) = 0;
            [r c] = find(gTemp == max(gTemp(:)));
            bestInd = bestMax(imR,r,c,pixel(1),pixel(2),offsetNeg);
            if sum(sum(gTemp,1)>0) < 2 || sum(sum(gTemp,2)>0)) < 2
                gTemp = imagePadSmallValues(gTemp,r(bestInd),c(bestInd));
            end
            [mx, my, var_x, var_y, covar_xy , ~, ~, ~, ~] = moments(gTemp);
            mx = mx+pixel(2)-(offsetNeg + 1);
            my = my+pixel(1)-(offsetNeg + 1);
B. MATLAB Code

```matlab
mxT = floor(mx + 0.5);
myT = floor(my + 0.5);
g = imR(myT-offsetNeg:myT+offsetPos,mxT-offsetNeg:mxT+offsetPos);
gTemp = g;
gTemp(gTemp<=2) = 0;
gTemp(gTemp<=(max(gTemp(:)) * 0.1)) = 0;
[r c] = find(gTemp == max(gTemp(:)));

% If there is a valid signal in the neighborhood it will
% have certain characteristics.

% There should only be small number of max values.
% If r or c have too many elements it means that there
% is no signal in the neighborhood. It is unlikely that
% a real signal would be spread uniformly over large
% number of pixels.
if size(r,1) <= 10
    bestInd = bestMax(imR,r,c,mxT,myT,offsetNeg);
end

% Isolate the area around the cluster.
mask = 1 * bwselect(gTemp,c(bestInd),r(bestInd),8);
nonZeros = find(mask > 0);

% If the cluster size is larger than 1 then it could
% be a signal.
if size(nonZeros,1) >= 2 || max(max(gTemp.*mask)) > 3
    mask = increaseBoundary(mask,sqrt(2));
g = g.*mask;
end

% This g has a weak cluster. Fit a Gaussian to it.

gTemp = g;
[r c] = find(gTemp == max(gTemp(:)));

if sum(sum(gTemp,1) > 0) < 2 || sum(sum(gTemp,2) > 0) < 2
    gTemp = imagePadSmallValues(gTemp,r(1),c(1));
end

[mxt, myt, var_xt, var_yt, covar_xyt, -, -, -, -] = moments(gTemp);
sxt = sqrt(var_xt);
syt = sqrt(var_yt);
rho_xyt = covar_xyt / sxt/syt;
max_pixelt = max(g(:));
offset_percentage = 0.5;

offset_percentage.Center = 0.5;

%A_lb = 1;
A_lb = 1e-10;
A_ub = 1000*max_pixelt;
xm_lb = mxt-abs(mxt*offset_percentage.Center);
xm_ub = mxt+abs(mxt*offset_percentage.Center);
ym_lb = myt-abs(myt*offset_percentage.Center);
ym_ub = myt+abs(myt*offset_percentage.Center);
sx_lb = sxt-abs(sxt*offset_percentage);
```
B. MATLAB Code

```
sx_ub = sxt + 10 * abs(sxt * offset_percentage);
if sx_ub < 3
    sx_ub = 5;
end
sy_lb = syt - abs(syt * offset_percentage);
sy_ub = syt + 10 * abs(syt * offset_percentage);
if sy_ub < 3
    sy_ub = 5;
end

x = [max_pixelt mxt myt sxt syt rho xyt];
if sx_lb <= 0
    sx_lb = 0.1;
end
if sy_lb <= 0
    sy_lb = 0.1;
end
rho_xy_lb = -0.99;
rho_xy_ub = 0.99;

dim_x = size(g, 2);
dim_y = size(g, 1);
g = g(:);

[Gauss_Params, fval, exitflag] = fmincon('GaussFit', x, [], [], [], [], [A_lb xm_lb ym_lb sx_lb syt xyt rho_xy_lb], [A_ub sx_ub ym_ub syub rho_xy_ub], [], optimset('Algorithm', 'sqp', 'TolX', 1e-10, 'TolFun', 1e-10, 'MaxFunEvals', 100000, 'MaxIter', 10000, 'Display', 'notify-detailed'), g, dim_y, dim_x);

% Have limits been reached?
if sum(isreal(Gauss_Params(:))) == 0
    disp('ERROR: Complex Number Detected');
    pause
    disp('Continue');
end

Gauss_Params(2) = Gauss_Params(2) + mxT - (offsetNeg + 1);
Gauss_Params(3) = Gauss_Params(3) + myT - (offsetNeg + 1);

% This is to renormalize the gaussian.
sum_pre = sum(g(:));
sum_post = sum(sum(generatePSFfromGaussianCoefficients(Gauss_Params', 2, 2)));

Gauss_Params(1) = Gauss_Params(1) / sum_post * sum_pre;

% Normalize it to Acquisition time.
Gauss_Params(1) = Gauss_Params(1) * 100 / (10 * (15 - k));
p(k, j, i, pin).value = Gauss_Params(1);
p(k, j, i, pin).posX = 2 * Gauss_Params(2);
p(k, j, i, pin).posY = 2 * Gauss_Params(3);
p(k, j, i, pin).sx = 2 * Gauss_Params(4);
p(k, j, i, pin).sy = 2 * Gauss_Params(5);
p(k, j, i, pin).tilt = Gauss_Params(6);
exists(k, j, i, pin) = 1;
```
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;

fclose(fn);

writePtoFileAsCoeff(fileOut,p);
switch systemSetup
  case 1
    save('8−20−12−p0mm_PE_v2_complete.mat','p');
  case 2
    save('8−20−12−p8mm_PE_v2_complete.mat','p');
  case 3
    save('8−20−12−p16mm_PE_v2_complete.mat','p');
  case 4
    save('8−20−12−p24mm_PE_v2_complete.mat','p');
  case 5
    save('8−20−12−p27mm_PE_v2_complete.mat','p');
  case 6
    save('8−20−12−p37mm_PE_v2_complete.mat','p');
end
end

%%% Extrapolate Values in a Volume
%%% %Put the data in the middle
volBoxPosX = NaN(size(posX,1)+4.size(posX,2)+4.size(posX,3)+4);
volBoxPosX(3:size(volBoxPosX,1)−2,3:size(volBoxPosX,2)−2,3:size(volBoxPosX,3)−2) = posX;
%Inpaint the box
volBoxPosX = inpaintn(volBoxPosX,1000);
%Extract the data.
posX = volBoxPosX(3:size(volBoxPosX,1)−2,3:size(volBoxPosX,2)−2,3:size(volBoxPosX,3)−2);
B. 2.4 Interpolation of Pinhole Projections

function [coeffInterpN2 p1] = interpolateInterpnWithMask_x3_simple(p,
                      filter, paint, range, inc, newInc, dim, interpolationMethod, varargin)

% Interpolate Pinhole Projections
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% [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 syl2 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;
            end
        end
    end
end

–201–
B. MATLAB Code

```matlab
end
end

if sum(accept(:)) > 0
    if filter~=0
        if filter == 1 || filter == 2
            if pinNumber==1
                disp('Median & Smooth Filters Selected');
            end
            valueI = filterVolBox(valueI,filter,1);
            posXI = filterVolBox(posXI,filter,1);
            posYI = filterVolBox(posYI,filter,1);
            sxI = filterVolBox(sxI,filter,1);
            syI = filterVolBox(syI,filter,1);
            tiltI = filterVolBox(tiltI,filter,1);
        end
    end
end

clear done2
done2 = interp3(x1,y1,z1,done,x4,y4,z4,'linear');

valueI = fillArrayInPaint(valueI);
posXI = fillArrayInPaint(posXI);
posYI = fillArrayInPaint(posYI);
sxI = fillArrayInPaint(sxI);
syI = fillArrayInPaint(syI);
tiltI = fillArrayInPaint(tiltI);
valueI(isnan(valueI(:))==1)=0;

done2(done2>0) = 1;

valueI2 = interp3(x1,y1,z1,valueI,x4,y4,z4,interpolationMethod,0).*done2;
posXI2 = interp3(x1,y1,z1,posXI,x4,y4,z4,interpolationMethod,0).*done2;
posYI2 = interp3(x1,y1,z1,posYI,x4,y4,z4,interpolationMethod,0).*done2;
sxI2 = interp3(x1,y1,z1,sxI,x4,y4,z4,interpolationMethod,0).*done2;
syI2 = interp3(x1,y1,z1,syI,x4,y4,z4,interpolationMethod,0).*done2;
tiltI2 = interp3(x1,y1,z1,tiltI,x4,y4,z4,interpolationMethod,0).*done2;

%Remove out of bound values.
ind = find(valueI2<0);
if size(ind,1)>0
    valueI2(ind(:))=0;
posXI2(ind(:))=0;
posYI2(ind(:))=0;
sxI2(ind(:))=0;
syI2(ind(:))=0;
tiltI2(ind(:))=0;
end
```
ind = [find(tiltI2 <= -1 | tiltI2 >= 1); find(sxI2 < 0); find(syI2 < 0)];

if size(ind,1)>0
    valueI2(ind(:)) = 0;
    posXI2(ind(:)) = 0;
    posYI2(ind(:)) = 0;
    sxI2(ind(:)) = 0;
    syI2(ind(:)) = 0;
    tiltI2(ind(:)) = 0;
end

for i = 1:interpDim(3)
    for j = 1:interpDim(2)
        for k = 1:interpDim(1)
            if valueI2(k,j,i) >= 0.1
                if floor(posXI2(k,j,i) + 0.5) > 0 && (floor(posYI2(k,j,i) + 0.5)) > 0 && floor(posYI2(k,j,i) + 0.5) < 480 && (floor(posYI2(k,j,i) + 0.5)) < 640
                    if mask(floor(posXI2(k,j,i) + 0.5), floor(posYI2(k,j,i) + 0.5)) == 1
                        if isempty(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)];
                        else
                            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)];
                        end
                    end
                end
            end
        end
    end
end
end
clear valueI posXI posYI sxI syI tiltI

function array = fillArrayInPaint(array)
done = ones(size(array));

% Create a 'done' matrix
ind = find(array==0 | isnan(array)==1);
if size(ind,1)>0
    done(ind(:)) = 0;
    % Find the ideal starting point.
    % This point will have as many values as possible surrounding it, not

% % Extrapolate Values in a Volume
% %
function array = fillArrayInPaint(array)
%necessarily the largest values.
[k j i] = returnLargestNeighborhood(array);
start = [k j i];

counter = 0;
continueLoop = 1;
while continueLoop > 0 && size(ind,1) > counter
[k2 j2 i2] = getNextVoxelToProcess(done, start, size(done));
[value continueLoop] = returnAveragePredictedValueLinear(array,k2,j2,i2);
    if continueLoop > 0
        array(k2,j2,i2) = value;
    end
    done(k2,j2,i2) = 1;
    counter = counter + 1;
end

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
% Extrapolate Values in a Volume 1D
% Called from : fillArrayInPaint.m
% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %
function [value cnt] = returnAveragePredictedValueLinear(volMiBox,k,j,i)
value = 0;
cnt = 0;
for dimension = 1:3
    switch dimension
    case 1
        %Original setup
        volMi2 = permute(volMiBox,[1 2 3]);
    case 2
        %Change axes setup
        volMi2 = permute(volMi2,[3 1 2]);
    case 3
        volMi2 = permute(volMi2,[3 1 2]);
    end
    switch dimension
    case 1
        im = volMi2(:,j,i);
        if sum(isnan(im(:))) == size(im,1) * size(im,2) || sum(isnan(im(:))) == 0 < 2
            else
                val = local_Interp_v2(im,k);
                if val > 0
                    value = value + val;
                    cnt = cnt + 1;
                end
        end
    case 2
        im = volMi2(:,k,j);
        if sum(isnan(im(:))) == size(im,1) * size(im,2) || sum(isnan(im(:))) == 0 < 2
            else
                val = local_Interp_v2(im,k);
                if val > 0
                    value = value + val;
                    cnt = cnt + 1;
                end
        end
end


```matlab
else
    val = local_interp.v2(im, i);
    if val>0
        value = value + val;
        cnt=cnt+1;
    end
end

case 3
    im = volMi2(:, i, k);
    if sum(isnan(im(:))) == size(im,1)*size(im,2) || sum(isnan(im(:))==0) < 2
        else
            val = local_interp.v2(im, j);
            if val>0
                value = value + val;
                cnt=cnt+1;
            end
        end
    end
end
if cnt>0
    value = value/cnt;
end
end

% SUBFUNCTIONS – Linear Interpolation of the Arrays of ’returnAveragePredictedValueLinear.m’
function val = local_interp(im, i)
    val=0;
    if i+2<=size(im,1) && i-2>=1
        runF = 0;
        if isnan(im(i+1))==0 && isnan(im(i+2))==0 && runF == 0
            val = interp1([0 1], [im(i+2) im(i+1)], 2, ’linear’, ’extrap’);
            runF = 1;
        end
        if isnan(im(i-1))==0 && isnan(im(i-2))==0 && runF == 0
            val = interp1([0 1], [im(i-2) im(i-1)], 2, ’linear’, ’extrap’);
            runF = 1;
        end
    elseif i+2<=size(im,1) && i-2<=0
        runF = 0;
        if isnan(im(i+1))==0 && isnan(im(i+2))==0 && runF == 0
            val = interp1([0 1], [im(i+2) im(i+1)], 2, ’linear’, ’extrap’);
            runF = 1;
        end
    elseif i+2>=size(im,1) && i-2>=1
        runF = 0;
        if isnan(im(i-1))==0 && isnan(im(i-2))==0 && runF == 0
            val = interp1([0 1], [im(i-2) im(i-1)], 2, ’linear’, ’extrap’);
            runF = 1;
        end
    end
end
```

---

B. MATLAB Code
function val = local_interp_v2(im, i)
val = 0;
positionP = i;
while positionP + 1 <= size(im, 1) && isnan(im(positionP + 1)) == 0
    positionP = positionP + 1;
end

positionN = i;
while positionN - 1 > 0 && isnan(im(positionN - 1)) == 0
    positionN = positionN - 1;
end

if abs(positionP - i) > 2 || abs(positionN - i) > 2
    if abs(positionP - i) > abs(positionN - i)
        if sum(isnan(im(i + 1:positionP))) > 0
            im(i + 1:positionP)
            pause
        end
    end
    else
        if abs(positionN - i) > abs(positionP - i)
            if sum(isnan(im(positionN:i - 1))) > 0
                im(positionN:i - 1)
                pause
            end
        end
        else
            val = 0;
        end
    end
end
Bibliography


Bibliography


Point Grey Research, Inc. URL: http://www.ptgrey.com/.

Proxivision, Inc. URL: http://www.proxivision.de.


Bibliography


–214–


