Implementation and Analysis of Katsevich Reconstruction for Helical Scan CT

Tregidgo, Henry F.J.

2013

MIMS EPrint: **2016.30**

ISSN 1749-9097

IMPLEMENTATION AND ANALYSIS OF KATSEVICH RECONSTRUCTION FOR HELICAL SCAN CT

A dissertation submitted to the University of Manchester
for the degree of Master of Science
in the Faculty of Engineering and Physical Sciences

2013

Henry F.J. Tregidgo
School of Mathematics

Contents

Word count 8970

List of Figures

Abstract

In performing X-ray Computed Tomography two major constraints are the required acquisition time and X-ray dose for the scans. One method of reducing these is to take the scan by moving source and detector in one continuous helix relative to the object, rather than taking several separate circular scans. This Dissertation examines two implementations of the derivatives required for the Katsevich reconstruction algorithm for helical cone beam micro-CT, both the original implementation suggested by Noo et al.[2003] and an alternate formulation proposed by Katsevich[2011]. A new bound on a parameter used by the second formulation is suggested and methods for dealing with practical difficulties when reconstructing real scan data are proposed.

Declaration

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual Property Statement

Acknowledgements

I would like to thank Will Thompson and my supervisor Bill Lionheart for all the discussions and guidance they have provided during the project. I thank my Industrial collaborators, especially TC for providing forward projections, scan data and ideas. Finally, I want to thank my friends and family for their support.

Chapter 1

Introduction

Non-destructive and non-invasive testing have found uses from Medicine to Materials science, Biological research to resource exploration. Our ability to investigate the unseen structures of an object can give great insights into how an object or system should work, and any flaws that could be disadvantageous.

A frequently used tool in Medical, Biological and Material settings is X-ray Computed Tomography (CT). By measuring the attenuation of X-rays passing through an object at different angles, it is possible to build up a picture of the structure it contains. This is equivalent to taking line integrals through the domain of the object and trying to recover the underlying attenuation function. Therefore, in Mathematical terms, this is an Inverse Problem.

Classical CT consists of reconstructing a single slice through the region of interest. This is done by modelling the X-ray transform as analogous to the 2D Radon transform, which can be inverted using Filtered Backprojection algorithms (FBP). In three dimensions the problem of reconstructing an object can be significantly more difficult. One option is to take successive slices through the object and reconstruct one row at a time. However, taking circular scans of every plane of voxels in an object is prohibitive.

We would like to find algorithms and source curves which are equivalent to having data for every point on the cylinder surrounding our region of interest. This can be considered as solving Fritz John's equation [5], an ultrahyperbolic PDE in 4 variables. The solution to this equation can provide all the values on the convex hull of some 3 dimensional boundary surface in a similar way to solving a 2D Dirichlet problem with boundary data measured along a curve. The problem is finding on which surfaces boundary data will be sufficient to uniquely determine a solution.

One option, which appears to work surprisingly well, is to take successive cone beam slices through the object and reconstruct one small cylinder at a time. This is done by assuming that the X-rays coming from a source are parallel within some angle of the central plane and reconstructing in a 2D framework using techniques such as the Feldkamp-Davis-Kress Algorithm (FDK) [3]. However, taking multiple small cone angle scans is again prohibitive and reconstructing planes outside a few degrees of cone angle produces artefacts. Therefore, we look for source curves which satisfy the Kirillov-Tuy completeness condition [10, 16]. This is a condition on the shape of the source curve, which determines whether all points in the region of interest can be exactly reconstructed.

One source curve which satisfies this condition is the helix. There have been several inversion algorithms proposed for helical, or spiral [6, 7, 9], scan CT including rebinning algorithms, FDK-like algorithms, quasi exact and exact algorithms. Each of which is based on a different technique and inversion formula.

Rebinning algorithms estimate the 2D data which would be achieved at a slice through the helix and then reconstruct using techniques for 2D scans. FDK algorithms rely on a generalisation of the Feldkamp algorithm to the helical geometry [3]. These are quite prone to artefacts which cannot easily be predicted. Quasi exact methods build on Grangeat's theory for cone beam reconstruction using the 3D Radon transform [4]. Finally, there are several exact reconstruction algorithms. The one of most interest for this project is the Katsevich inversion formula, as it is of the Filtered Backprojection type and can, therefore, be more easily implemented with existing techniques.

In practice, only medical CT machines and the security scanners based on medical CT machines implement Helical Scan. For most other purposes Circular Scan CT is used. This limits the number of planes which can be reconstructed for each scan due to artefacts at larger cone angles. As there is generally less concern about the X-ray dosage or timescale for these scans, more detail can be gained by taking a new scan with the required plane as the central slice. It would, however, be desirable to implement the Katsevich reconstruction algorithm due its ability to scan accurately at high cone angles and completely reconstruct an image from a shorter scan time.

Work has already been done of the implementation of the Katsevich reconstruction algorithm [12, 18], however this has mostly been examined from a medical imaging standpoint. There are several differences between medical CT machines and micro-CT machines used for materials testing. As noted above both the X-ray dosage and timescales of scans are largely increased for micro-CT, while the size and number of elements on the detector are not constrained by the design needing to accommodate a patient lying down. Therefore the available code is designed to work for a detector with 138×16 detector pixels, while the machines I will be using have square detectors with 512×512 pixels.

Through the course of this project I will aim to modify the available code [18] to accommodate the changed detector geometry used in micro-CT, while evaluating the efficiency and accuracy of the code. I will also have to handle the inherent problems found in taking real scan data, for example the problem of beam hardening. There are also practical problems such as defining the reconstructable area for a given dataset, adjusting the scan geometry to fully take advantage of the helical source trajectory and compensating for the different helices used.

In addition to these aims I will update the code to take advantage of the new formulation of the derivative step proposed by Katsevich [9]. This will require an analysis of how to derive this form of the derivative, as well as its strengths and limitations.

Chapter 2

Katsevich Setup

As the Katsevich reconstruction algorithm is of the filtered backprojection type it is first necessary to understand *backprojection.* For backprojection a ray or cone is first defined by source position and area or point on the filtered dataset. Values from this area of the projection are given a weighting and added to the voxels intersected by the line or cone. By doing this for a sufficient range of source positions, a reconstructed image of the target object can be produced.

There are two separate formulations of the Katsevich reconstruction algorithm. The first, proposed in 2002 [7], requires two separate functions to be backprojected and has been shown to be equivalent to the formulation proposed in 2004 [8]. This second algorithm requires only one backprojection, making it much more efficient and easily implemented. As such, the second formulation is the focus of this dissertation.

To prove that the formula is an exact inversion of the cone beam transform, Katsevich [8] shows that it is equivalent to performing the integration

$$\frac{1}{(2\pi)^3}\int_{\mathbb{R}^3} \mathbf{B}(\mathbf{x},\xi)\tilde{f}(\xi)e^{\mathrm{i}\xi\cdot\mathbf{x}}\,\mathrm{d}\xi. \tag{2.1}$$

Here $\tilde{f}$ is the Fourier transform

$$\tilde{f} = \mathcal{F}(f) = \int_{\mathbb{R}^3} f(\mathbf{x})e^{-\mathrm{i}\xi\cdot\mathbf{x}}\,\mathrm{d}x,$$

of the desired underlying function $f(\mathbf{x})$. As $\mathbf{B}$ is a function which can be shown to be equal to 1 *almost everywhere,* we obtain the Inverse Fourier transform and return our target function f.

A full proof of this Theorem is beyond the scope of this dissertation and can be found in the original paper. Instead, I will concentrate on the implementation of this scheme, both for numerical simulations and real scan data. As such I will now introduce some concepts and notation.

2.1 Detector Geometry

For this project I will be using notation adapted from Wunderlich [18] and Noo et al. [12] for the detector geometry. Here the position of the x-ray source along the helical source curve is taken to be

$$\mathbf{y}(s) = [R\cos(s), R\sin(s), P\frac{s}{2\pi}]^T, \tag{2.2}$$

where R is the radius of the helix, s is the angle through which the source has moved from the starting position and P is the distance the source moves in the z direction per full turn, known as the *Pitch*. In some places in the code this pitch has been described per radian rather than per turn, giving the parameter $h = P/(2\pi)$. It is also important to know the perpendicular distance from the detector to the source, D.

Having defined the source positions and basic geometry, it is now important to define local detector coordinates for storage of the detector data. Hence the coordinates (u, v, w) are used. Here u and w are the horizontal and vertical distances from the line passing perpendicularly through the centre of the detector plate, v is the distance from the rotation axis perpendicular to the detector plate. The unit vectors for this coordinate system are

$$\begin{aligned}
\mathbf{e_u} &= [-\sin(s), \cos(s), 0]^T, \\
\mathbf{e_v} &= [-\cos(s), -\sin(s), 0]^T, \\
\mathbf{e_w} &= [0, 0, 1]^T,
\end{aligned}$$

which are shown in fig. 2.1.

With this geometry, we can now define the data measured at each element of the detector as function g To do this, first we must define the direction vector

$$\boldsymbol{\theta}(s, u, w) = \frac{1}{\sqrt{u^2 + D^2 + w^2}}(u\mathbf{e_u}(s) + D\mathbf{e_v}(s) + w\mathbf{e_w}(s)), \tag{2.3}$$

which is the unit vector in the direction of the ray passing through $\mathbf{y}(s)$ and intersecting the detector at coordinates (u, w). From this we can construct the cone beam transform

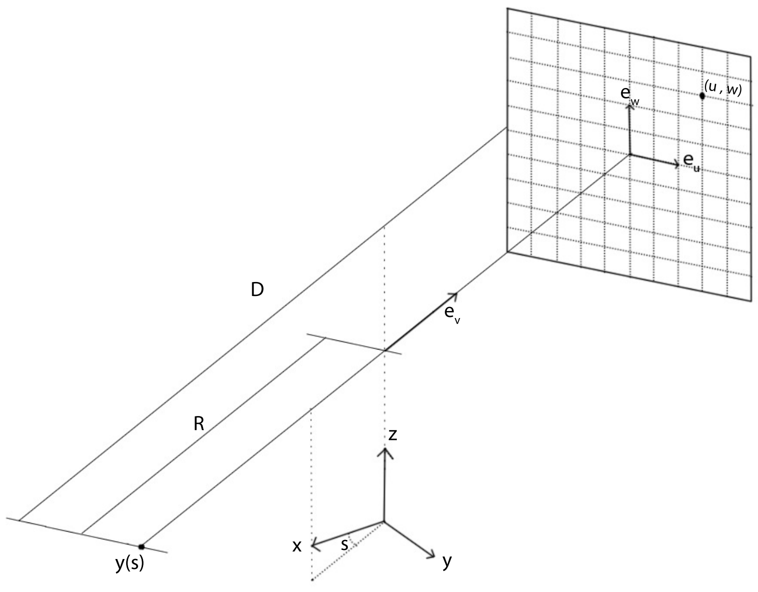


Figure 2.1: Illustration of flat detector geometry.

of the object,

$$g(s,u,w) = \int_0^{+\infty} f(\mathbf{y}(s) + t\boldsymbol{\theta})\,\mathrm{d}t, \tag{2.4}$$

where $f(\mathbf{x})$ is the underlying attenuation function at point $\mathbf{x}$.

2.2 Background Information

Before implementing the Katsevich formula it is important to understand the *Tam-Danielson window*, π*-lines* and κ*-planes.*

The *Tam-Danielson window* (TD window) is the area on the detector plane bounded above and below by the line inscribed by rays from the source position $\mathbf{y}(s)$ that intersect the source helix again on their way to the detector. On a flat detector this region is bounded above by the curve

$$w_{top}(u) = \frac{P}{2\pi RD}(u^2 + D^2)(\pi/2 - \arctan(u/D)), \tag{2.5}$$

and below by the curve

$$w_{bottom}(u) = -\frac{P}{2\pi RD}(u^2 + D^2)(\pi/2 + \arctan(u/D)). \tag{2.6}$$

This region is illustrated in fig. 2.4.

A π-line passing through a point $\mathbf{x}$ in the field of view is defined as a line intersecting the source helix at two points $\mathbf{y}(s_b)$ and $\mathbf{y}(s_t)$ separated by less than one helical turn. For a given $\mathbf{x}$ in the field of view, π-lines have been shown to be unique [1]. Using

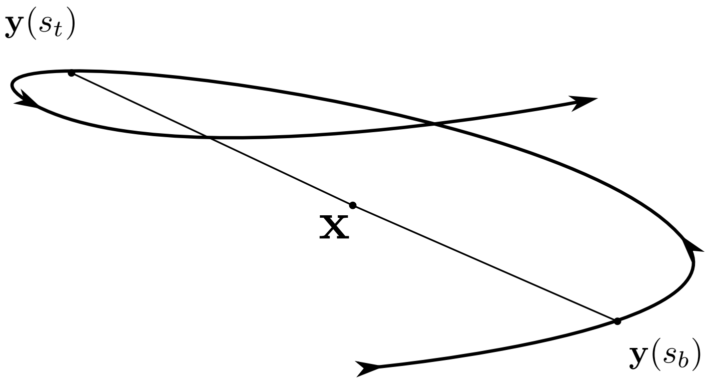


Figure 2.2: Illustration of π-line on a helix.

this concept it has been shown that for source positions $\mathbf{y}(s)$, $s \in I_{PI}(\mathbf{x}) = (s_b, s_t)$, if the projection of $\mathbf{x}$ onto the detector lies within the Tam-Danielson window then $f(\mathbf{x})$ may be reconstructed exactly [15, 1].

A κ-plane, $\mathcal{K}(s, \psi)$, is defined as a 2D plane intersecting the helix at three points, $\mathbf{y}(s)$, $\mathbf{y}(s+\psi)$ and $\mathbf{y}(s+2\psi)$ specified by the angle $\psi \in (-\pi, \pi)$. Taking the normal

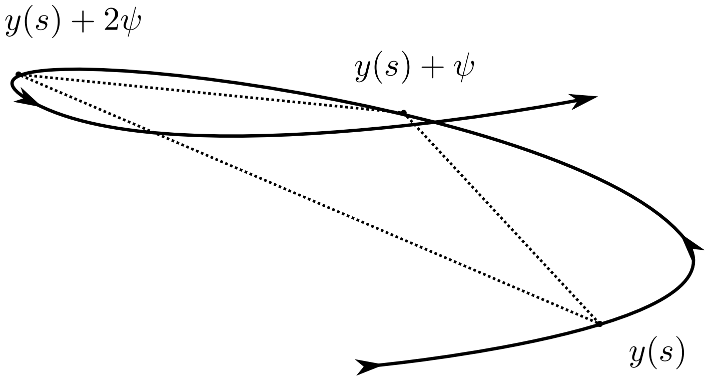


Figure 2.3: Illustration points defining a κ-plane.

to this plane as

$$\boldsymbol{\eta}(s, \psi) = \frac{(\mathbf{y}(s+\psi) - \mathbf{y}(s)) \times (\mathbf{y}(s+2\psi) - \mathbf{y}(s))}{\|(\mathbf{y}(s+\psi) - \mathbf{y}(s)) \times (\mathbf{y}(s+2\psi) - \mathbf{y}(s))\|} \operatorname{sgn}(\psi), \tag{2.7}$$

and a vector $\boldsymbol{\theta}$ parallel to the plane, we can describe any direction $\boldsymbol{\beta}$ along the plane as

$$\boldsymbol{\beta}(s, \gamma) = \cos(\gamma)\boldsymbol{\theta} + \sin(\gamma)(\boldsymbol{\theta} \times \boldsymbol{\eta}(s, \psi)). \tag{2.8}$$

Combining the definition of κ-planes with π-line intervals, we find that there exists a unique κ-plane, $\mathcal{K}(s, \psi)$, $s \in I_{PI}$, with minimum value of $|\psi|$ passing through each point in the field of view. This plane is such that $s_b \leq s + 2\psi \leq s_t$ and its intersection with the detector plane, or κ-line, is used for the 1D Hilbert transform filtering step in Katsevich's formula. These κ-lines are also shown in fig. 2.4.

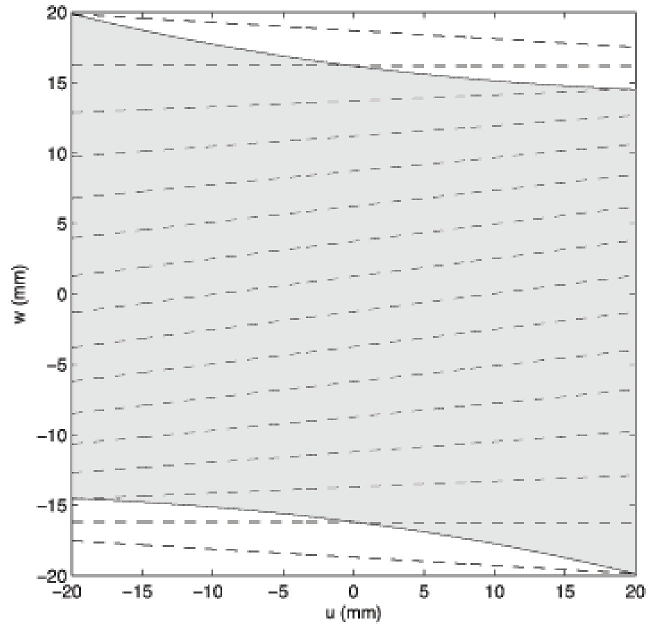


Figure 2.4: Illustration of the TD Window (shaded region) and κ-lines on a square detector.

Chapter 3

Implementing Katsevich

Having defined the above geometrical tools we can now examine the Katsevich inversion formula,

$$f(\mathbf{x}) = -\frac{1}{2\pi}\int_{I_{PI}(\mathbf{x})} \frac{1}{\|\mathbf{x}-\mathbf{y}(s)\|} g^F\left(s, \frac{\mathbf{x}-\mathbf{y}(s)}{\|\mathbf{x}-\mathbf{y}(s)\|}\right) \mathrm{d}s. \tag{3.1}$$

This formula involves backprojecting filtered data $g^F(s, \boldsymbol{\theta})$, to reconstruct the value of the underlying function at voxel position $\mathbf{x}$ in the FOV. The filtered function g^F is calculated as

$$g^F(s, \boldsymbol{\theta}) = \int_0^{2\pi} k_H(\sin\gamma) g'(s, \cos(\gamma)\boldsymbol{\theta} + \sin(\gamma)(\boldsymbol{\theta}\times\mathbf{m}(s,\psi))\,\mathrm{d}\gamma. \tag{3.2}$$

Here k_H is the kernel of the Hilbert transform

$$k_H(t) = -\int_{-\infty}^{\infty} \mathrm{i}\,\mathrm{sgn}(\xi) e^{\mathrm{i}2\pi\xi t}\,\mathrm{d}\xi = \frac{1}{\pi t},$$

$g'(s, \boldsymbol{\theta})$ is the partial derivative with respect to s of the measured $g(s, \boldsymbol{\theta})$ and $\mathbf{m}$ is the normal to the κ-plane $\mathcal{K}(s, \psi)$, with minimum value of $|\psi|$, which is parallel to $\boldsymbol{\theta}$. By considering a fixed κ-plane, this can be shown to be a convolution with the Hilbert kernel along the κ-line associated with that plane.

In order to implement this formula, it must be split up into several filtering steps and a backprojection as shown in the following sections.

3.1 Derivatives

The first step in filtering the data is to estimate the partial derivative of the data with constant direction vector

$$g_1(s,u,w) = \frac{\partial}{\partial s} g(s,\boldsymbol{\theta}) = \lim_{\epsilon \to 0} \frac{g(s+\epsilon,\boldsymbol{\theta}) - g(s,\boldsymbol{\theta})}{\epsilon}, \tag{3.3}$$

where $\boldsymbol{\theta}$ is defined as in eq. (2.3).

Taking accurate derivatives is essential for two reasons. First, errors in this early step will be propagated through the remaining steps and may become large errors or artefacts when the Hilbert transform is applied to the data. Second, the magnitude of the derivatives defines the resolution achievable with this reconstruction method. High derivatives mark edges within the FOV and help to define the objects being scanned.

Thinking about the attenuation values geometrically, as a ray is moved through a smooth object the values will drop when the line's intersection shortens. As the line loses its intersection with the object, the values will tend to zero. For any smooth object there is a lower limit to the length of the intersection and so the line will become tangent to the edge at some point. This means that the desired derivative will become infinite when the ray vector $\boldsymbol{\theta}$ is tangential to an objects edge. However, it is not possible to capture these infinite values in a discrete setting.

To find approximations to these derivatives I have used two methods. The first, implemented in the original Wunderlich [18] code, uses central differences and the chain rule

$$g_1(s,u,w) = \left(\frac{\partial g}{\partial s} + \frac{\partial u}{\partial s}\frac{\partial g}{\partial u} + \frac{\partial w}{\partial s}\frac{\partial g}{\partial w} \right), \tag{3.4}$$

as recommended by Noo et al. [12]. To implement this, averaged central differences are taken giving the value of the derivative at the mid points of the discretisation in each dimension. So

$$\begin{aligned} g_1(s_{k+1/2}, u_{i+1/2}, w_{j+1/2}) \simeq & \sum_{m=i}^{i+1} \sum_{n=j}^{j+1} \frac{g(s_{k+1}, u_m, w_n) - g(s_k, u_m, w_n)}{4\Delta s} \\ & + \left(\frac{u_{i+1/2}^2 + D^2}{D} \right) \sum_{p=k}^{k+1} \sum_{n=j}^{j+1} \frac{g(s_p, u_{i+1}, w_n) - g(s_p, u_i, w_n)}{4\Delta u} \\ & + \left(\frac{u_{i+1/2} w_{j+1/2}}{D} \right) \sum_{p=k}^{k+1} \sum_{m=i}^{i+1} \frac{g(s_p, u_m, w_j) - g(s_p, u_m, w_{j+1})}{4\Delta w}. \end{aligned} \tag{3.5}$$

It has been stated that the formulation of the derivative in eq. (3.5) produces better results under testing than taking a direct central difference between two views using eq. (3.3) [12]. However, as we need the derivative along specific filtering lines for the Hilbert transform step, we will have to interpolate in the w direction. This means that the derivative, which has already been averaged between two detector rows is further interpolated in this direction, reducing the resolution in w.

To combat this problem Katsevich [9] suggests splitting up the the derivative into two parts. By treating the variables u and w as functions of s and $\boldsymbol{\theta}$ we can rewrite the derivative as

$$\begin{aligned} g_1(s, U(s,\boldsymbol{\theta}), W(s,\boldsymbol{\theta})) &= \frac{\mathrm{d}}{\mathrm{d}s} g(s, U(s,\boldsymbol{\theta}), W(s,\boldsymbol{\theta})), \\ &= \frac{\mathrm{d}}{\mathrm{d}s}\, g(s, U(s,\boldsymbol{\theta}), w)|_{w=W(s,\boldsymbol{\theta})} + g'_w(s, U(s,\boldsymbol{\theta}), W(s,\boldsymbol{\theta}))W'_s(s,\boldsymbol{\theta}). \end{aligned} \tag{3.6}$$

In this formulation only the second function has been averaged between detector rows. Therefore, it is possible to calculate each part separately and recombine in the rebinning step so that less resolution has been lost in w.

Now the two components of the derivative must be calculated. I will denote the derivative keeping w constant as g_{1us} and the partial derivative with respect to w as g_{1w}. To calculate g_{1us} Katsevich [9] recommends taking a central difference using two interpolated points corresponding to $g(s_i - \delta, U(s_i - \delta, \boldsymbol{\theta}), w)$ and $g(s_i + \delta, U(s_i + \delta, \boldsymbol{\theta}), w)$. These two values can be found by interpolating over the 4 closest discretisation points as shown in fig. 3.1.

Katsevich [9] recommends deriving the interpolation weightings for each of the surrounding values from specific restrictions. However, an alternate derivation is possible by transforming this domain onto a reference square as shown in fig. 3.2 and using bilinear basis functions [2]. The bilinear basis functions on this reference element are

$$\begin{aligned} a_1(\xi,\eta) &= \frac{(1-\xi)(1-\eta)}{4}, & a_2(\xi,\eta) &= \frac{(1+\xi)(1-\eta)}{4}, \\ a_3(\xi,\eta) &= \frac{(1+\xi)(1+\eta)}{4}, & a_4(\xi,\eta) &= \frac{(1-\xi)(1+\eta)}{4}. \end{aligned} \tag{3.7}$$

In these basis functions ξ and η are coordinates for the reference element. Therefore, before the basis functions can be evaluated the coordinates for the difference point must be found. To do this, note that the relative position of the point in s is $+\delta$ and

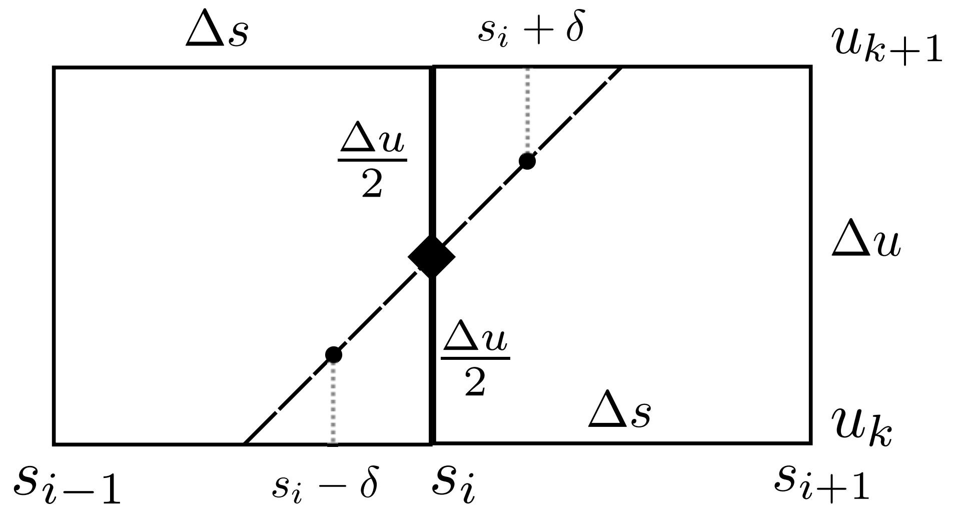


Figure 3.1: Diagram for computing s derivative of g as a function of s and u

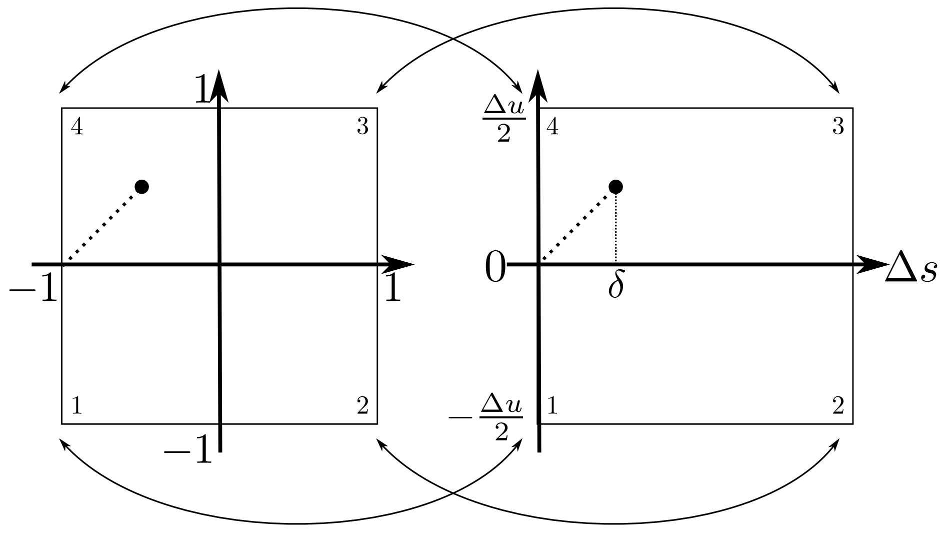


Figure 3.2: Diagram showing transformation between the reference element and interpolation domain for calculation of the values used when taking the difference.

linearise the u value. Taking $U(s_i + \delta, \boldsymbol{\theta}) \simeq u_{j+0.5} + \delta U'_s|_{s=s_i, U=u_{j+0.5}}$, the reference element coordinates are

$$\xi = \left(\frac{2\delta}{\Delta s} - 1\right), \qquad \eta = \frac{2U'_s\delta}{\Delta u}. \tag{3.8}$$

These coordinates give the weightings

$$\begin{aligned} a_1 &= \frac{1}{4}\left(\frac{2\delta}{\Delta s} - 2\right)\left(\frac{2U'_s\delta}{\Delta u} - 1\right), & a_2 &= -\frac{1}{4}\left(\frac{2\delta}{\Delta s}\right)\left(\frac{2U'_s\delta}{\Delta u} - 1\right), \\ a_3 &= \frac{1}{4}\left(\frac{2\delta}{\Delta s}\right)\left(\frac{2U'_s\delta}{\Delta u} + 1\right), & a_4 &= -\frac{1}{4}\left(\frac{2\delta}{\Delta s} - 2\right)\left(\frac{2U'_s\delta}{\Delta u} + 1\right). \end{aligned} \tag{3.9}$$

Using the weightings in eq. (3.9) and the rotational symmetry of the two difference points, the derivative g_{1us} can be calculated as

$$\begin{aligned} g_{1us}(s_i, u_{k+0.5}, w_j) &= \frac{\mathrm{d}}{\mathrm{d}s}\, g(s_i, U(s_i, \boldsymbol{\theta}), w_j)|_{U(s_i,\boldsymbol{\theta})=u_{k+0.5}}, \\ &\simeq \frac{g(s_i + \delta, U(s_i + \delta, \boldsymbol{\theta}), w_j) - g(s_i - \delta, U(s_i - \delta, \boldsymbol{\theta}), w_j)}{2\delta}, \\ &\simeq \frac{1}{2\delta}\Bigg[a_3(g_{i+1,k+1,j} - g_{i-1,k,j}) + a_2(g_{i+1,k,j} - g_{i-1,k+1,j}) \\ &\qquad + (a_4 - a_1)(g_{i,k+1,j} - g_{i,k,j}) \Bigg]. \end{aligned} \tag{3.10}$$

Taking $r = a_3/\delta$ we can reformulate g_{1us} as

$$\begin{aligned} g_{1us}(s_i, u_{k+0.5}, w_j) \simeq \frac{1}{2}\Bigg[& r(g(s_{i+1}, u_{k+1}, w_j) - g(s_{i-1}, u_k, w_j)) \\ & + \left(\frac{1}{\Delta s} - r\right)(g(s_{i+1}, u_k, w_j) - g(s_{i-1}, u_{k+1}, w_j)) \\ & + \left(\frac{1}{\Delta s} + \frac{2U'_s}{\Delta u} - 2r\right)(g(s_i, u_{k+1}, w_j) - g(s_i, u_k, w_j))\Bigg], \end{aligned} \tag{3.11}$$

where U'_s is the partial derivative of $U(s, \boldsymbol{\theta})$ with respect to s at the point $(s_i, u_{k+0.5})$, which is the same formulation as suggested by Katsevich [9]. It is also suggested that r may take any value in the range $0 \leq r \leq 1/\Delta s$, so r has been taken to be one for simplicity in my implementation of the code.

Now to find the second component of the derivative we have to take an averaged central difference in w,

$$\begin{aligned} g_{1w}(s_i, u_{k+0.5}, w_{j+0.5}) &= W'_s g'_w(s_i, u_{k+0.5}, w_{j+0.5}) \\ &\simeq W'_s \frac{(g_{i,k,j+1} + g_{i,k+1,j+1}) - (g_{i,k,j} + g_{i,k+1,j})}{2\Delta w}. \end{aligned}$$

Here W'_s is the partial derivative at the point $(s_i, u_{k+0.5}, w_{j+0.5})$ of $W(s, \boldsymbol{\theta})$ with respect to s, and this derivative is stored in a separate file to be recombined in the forward rebinning step.

These derivatives are calculated by code similar to that found in appendix A.4.

3.2 Length Correction Weighting

Correcting for the increased Euclidean distance of detector elements further from the centre of the detector plate, the length correction

$$g_2(s,u,w) = \frac{D}{\sqrt{u^2 + D^2 + w^2}} g_1(s,u,w) \tag{3.12}$$

is applied when working with the derivatives directly. However, when using Katsevich's formulation of the derivative the same correction must be applied. Hence I have decided to apply this same correction to both parts of the Katsevich derivative, giving

$$\begin{aligned} g_{2us}(s,u,w) &= \frac{D}{\sqrt{u^2 + D^2 + w^2}} g_{1us}(s,u,w), \\ g_{2w}(s,u,w) &= \frac{D}{\sqrt{u^2 + D^2 + w^2}} g_{1w}(s,u,w). \end{aligned} \tag{3.13}$$

3.3 Forward Height Rebinning

As the Hilbert transform filtering step is performed along Kappa lines we want to place the data into the coordinates (s, u, ψ), where ψ labels which kappa line the data is measured on. This is done by taking

$$g_3(s,u,\psi) = g_2(s,u,w_\kappa(u,\psi)), \tag{3.14}$$

such that

$$w_\kappa(u,\psi) = \frac{DP}{2\pi R}\left(\psi + \frac{\psi}{\tan\psi}\frac{u}{D}\right). \tag{3.15}$$

In order to approximate the required values, the code first discretises the range of ψ such that $\psi \in (-\pi/2 - \alpha_m, \pi/2 + \alpha_m)$, where α_m is the half fan angle. I have written code to ensure that the ψ domain is discretised so that the maximum κ-line separation is no greater than the width of a detector row [12]. This is done by equating the differential of the w_k from eq. (3.15) with the detector element width d_w divided by $\Delta\psi$ in the top left corner of the detector using

$$\frac{\mathrm{d}w_k}{\mathrm{d}\psi}(u = u_{\min}, \psi = \pi/2 + \alpha_m) \simeq \frac{d_w}{\Delta\psi}. \tag{3.16}$$

Once the range of ψ has been discretised using the code in appendix B.2, the values of w_κ are calculated using eq. (3.15). These values are then used for one dimensional interpolation to find the required values.

This interpolation is performed slightly differently for each of the two difference schemes implemented in the code. When using the original Noo implementation, the value of $g_3(s, u, \psi)$ is calculated by interpolating linearly on the detector plate between $g_2(s, u, w_j)$ and $g_2(s, u, w_{j+1})$, where $w_j \leq w_\kappa \leq w_{j+1}$. This gives

$$g_3(s, u, \psi) = (1 - c)g_2(s, u, w_j) + cg_2(s, u, w_{j+1}), \tag{3.17}$$

where

$$c = \frac{w_\kappa - w_j}{\Delta w}.$$

As noted above this is an interpolation in w using two values which have already lost resolution due to the implicit averaging process used when taking the central difference. This further reduces the possible resolution in the vertical direction.

This step is why the Katsevich formulation of the derivative has been suggested. In order to reduce resolution loss Katsevich [9] recommends linearly interpolating the function g_{1us} and adding on a nearest neighbour contribution from g_{1w}. This gives

$$g_{3alt} = (1 - c)g_{2us}(s, u, w_j) + cg_{2us}(s, u, w_{j+1}) + g_{2w}(s, u, \mu),$$

where c is as above and μ is whichever of w_j and w_{j+1} is closer to w_κ. Katsevich states that, as the contributions from g_{2w} are generally smaller than those from g_{2us} and the values in g_{2us} have only been interpolated once in w the loss of resolution is reduced.

The code ensures there are an even number of filtering lines as $\psi = 0$ causes a divide by zero error when calculating the values of w_κ. An alternate option may be to use the symmetry of the kappa lines to only explicitly calculate the lines for $\psi > 0$ and rotate these to find the remaining values.

3.4 1D Hilbert Transform

This step is to take a 1D Hilbert transform in u at constant ψ

$$g_4(s,u,\psi) = \int_{-\infty}^{\infty} k_H(u-u')g_3(s,u',\psi)\,\mathrm{d}u'. \tag{3.18}$$

The kernel, k_H of this transform is formally expressed as

$$k_H(t) = -\int_{-\infty}^{\infty} \mathrm{i}\,\mathrm{sgn}(\xi)e^{\mathrm{i}2\pi\xi t}\,\mathrm{d}\xi.$$

However, in practice we wish to have a cut off frequency and so the kernel is approximated as

$$\begin{aligned} k_H(t) &\simeq -\int_{-b}^{b} \mathrm{i}\,\mathrm{sgn}(\xi)e^{\mathrm{i}2\pi\xi t}\,\mathrm{d}\xi, \\ &= \frac{1}{\pi t}[1-\cos(2\pi bt)]. \end{aligned} \tag{3.19}$$

Now discretising the Hilbert transform so that $t_n = n\Delta t$, the Nyquist condition tells us that $b \leq b_{max} = 1/(2\Delta t)$ giving the discrete kernel

$$\begin{aligned} k_H[n] = k_H(n\Delta t) &= \frac{1}{\pi n\Delta t}[1-\cos(\pi n)], \\ &= \begin{cases} \frac{2}{\pi n\Delta t} & \text{if } n \text{ is odd,} \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{3.20}$$

The code also windows the Hilbert kernel in the frequency domain to reduce ringing artefacts. This is done by convolving the inverse Fourier transform of a Hamming window, win$[n]$ with the kernel $k_H[n]$ to get the modified kernel $k_W[n]$. Using this modified kernel and discretising $u_n = n\Delta u$ eq. (3.18) becomes the discrete convolution formula

$$g_4(s,u_n,\psi) = \Delta u \sum_{l=l_{min}}^{l_{max}} k_W[n-l]g_3(s,u_l,\psi), \tag{3.21}$$

which is implemented in the code using FFTs.

3.5 Backward Height Rebinning

For each height position on the detector we only want to backproject from the kappa line with lowest absolute value which passes through this point. Therefore we set

$$g^F(s,u,w) = g_5(s,u,w) = g_4(s,u,\hat{\psi}(u,w)), \tag{3.22}$$

where $\hat{\psi}(u,w)$ is the angle of smallest absolute value satisfying

$$w = \frac{DP}{2\pi R}\left(\psi + \frac{\psi}{\tan\psi}\frac{u}{D}\right).$$

This is not an injective mapping and the value of $\hat{\psi}(u,w)$ will only be used to define grid points for interpolation, so finding $\hat{\psi}(u,w)$ explicitly may not be the most efficient course of action [12].

As the backprojection is only required to run on an area slightly larger than the TD window, we can use the uniqueness of κ-lines passing through each point in the TD window to define an interpolation scheme. By definition, inside the TD window

$$g_5(s,u,w_\kappa(u,\psi)) = g_4(s,u,\psi),$$

while outside the TD window κ-lines defined by increasing values of ψ may only cross each other in the top right hand and bottom left quarters of the detector plate.

In the top right quarter of the detector plate $w_\kappa(u,\psi)$ increases monotonically with ψ until the cross over point where it becomes monotonically decreasing. Similarly in the bottom left quarter of the detector $w_\kappa(u,\psi)$ decreases monotonically as ψ becomes more negative, until the cross over point. Therefore, if we iterate through the ψ values to find ψ_j and ψ_{j+1} such that

$$w_\kappa(u_i,\psi_j) \leq w \leq w_\kappa(u_i,\psi_{j+1}), \tag{3.23}$$

we can take the linear interpolant in w to get the correct approximation to g_5.

The original code provided by Wunderlich [18] performed this step by iterating through every κ-line to find the correct ψ interval for each individual grid point. However, as noted above the function increases and decreases monotonically within the regions required for backprojection. Therefore, in the right half of the detector plane, I have redefined the starting point for the search at (u_i, w_{j+1}) to be the interval found for (u_i, w_j). Similarly, in the left half of the detector plane, I have redefined the starting point for the search at (u_i, w_{j-1}) to be the interval found for (u_i, w_j). This ensures that the number of comparisons along each line $u = u_i$ is kept to $O(L)$, where L is the total number of κ-lines.

3.6 Backprojection

Having used the code in appendix A.9 to find the π-line intervals for the backprojection the code completes the reconstruction of $f(\mathbf{x})$ by

$$f(\mathbf{x}) = \frac{1}{2\pi}\int_{s_b}^{s_t} \frac{1}{v^*(s,\mathbf{x})} g_5(s, u^*(s,\mathbf{x}), w^*(s,\mathbf{x}))\,\mathrm{d}s. \tag{3.24}$$

Here the values of u, v and w are given by

$$\begin{aligned} v^*(s,\mathbf{x}) &= R - x\cos(s) - y\sin(s), && (3.25)\\ u^*(s,\mathbf{x}) &= \frac{D}{v^*(s,\mathbf{x})}(-x\sin(s) + y\cos(s)), && (3.26)\\ w^*(s,\mathbf{x}) &= \frac{D}{v^*(s,\mathbf{x})}\left(z - \frac{P}{2\pi}s\right). && (3.27) \end{aligned}$$

In the code, this has been discretised as

$$f(\mathbf{x}) \simeq \sum_k \frac{\rho(s_k,\mathbf{x})\Delta s}{2\pi v^*(s_k,\mathbf{x})} g_5(s, u^*(s_k,\mathbf{x}), w^*(s_k,\mathbf{x})). \tag{3.28}$$

The function $\rho(s,\mathbf{x})$ has been added here to smooth the transition near the end of the π-line interval. Due to the discrete nature of our s values, it is unlikely that the end values can be accurately specified, so to avoid artefacts ρ has been specified as

$$\rho(s,\mathbf{x}) = \begin{cases} 0 & \text{if } s \le s_b - \Delta s,\\ (1+d_b)^2/2 & \text{if } s_b - \Delta s \le s \le s_b,\\ \frac{1}{2} + d_b - d_b^2/2 & \text{if } s_b \le s \le s_b + \Delta s,\\ 1 & \text{if } s_b + \Delta s \le s \le s_t - \Delta s,\\ \frac{1}{2} + d_t - d_t^2/2 & \text{if } s_t - \Delta s \le s \le s_t,\\ (1+d_t)^2/2 & \text{if } s_t \le s \le s_t + \Delta s,\\ 0 & \text{if } s_t + \Delta s \le s, \end{cases} \tag{3.29}$$

where

$$d_b = \tfrac{s - s_b(\mathbf{x})}{\Delta s} \quad d_t = \tfrac{s_t(\mathbf{x}) - s}{\Delta s}.$$

It has also been necessary to interpolate the values of g_5 to evaluate $g^F(s, u^*, w^*)$ needed for the backprojection step. In the code, this has been implemented using nearest neighbour interpolation.

Chapter 4

Numerical Results

Before using my modified code on real scan data, I first tested it on two different numerical phantom images. The first of these is the phantom I have named *Reduced Ball.* This phantom is based on the *3D Shepp Logan* head phantom, which has been modified so that the ellipsoids are replaced by spheres with a uniform attenuation coefficient of 1. The reason for using this image was to more accurately simulate the situations arising in materials scanning as opposed to medical imaging, while increasing the efficiency of my forward projection function.

The second phantom, *9 Planes of Spheres*, was set up to test geometry similar to the real scan setup, and to compare against reconstructions already being done using circular scan methods.

Parameters for all reconstructions may be found in appendix C

4.1 Reduced Ball Phantom

4.1.1 Intermediate Steps

While not generally studied, the intermediate datasets produced in completing a Katsevich reconstruction can be informative when trying to understand how the algorithm works. The differences in these images produced by using a different formulation of the derivative, or changing specific parameters, also show the sensitivities present in this inversion method.

The first step in modifying the view shown in fig. 4.1, as mentioned in section 3.1, is to take the partial derivative of the function g with respect to s at constant ray direction. Figure 4.2 shows the result of this derivative using the original chain rule implementation while fig. 4.3 shows the two required derivatives in the Katsevich formulation.

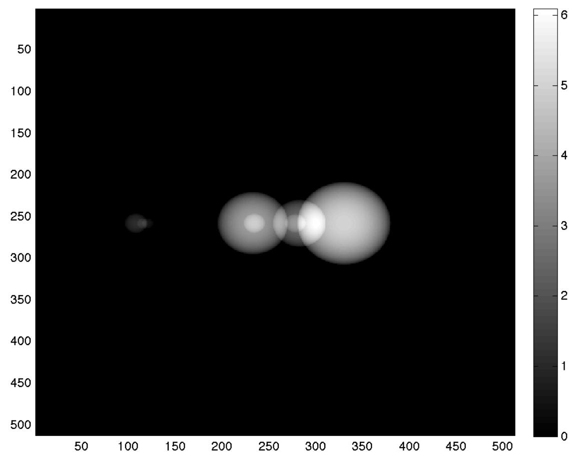


Figure 4.1: Single view of the projected scan data g for the Reduced Ball phantom.

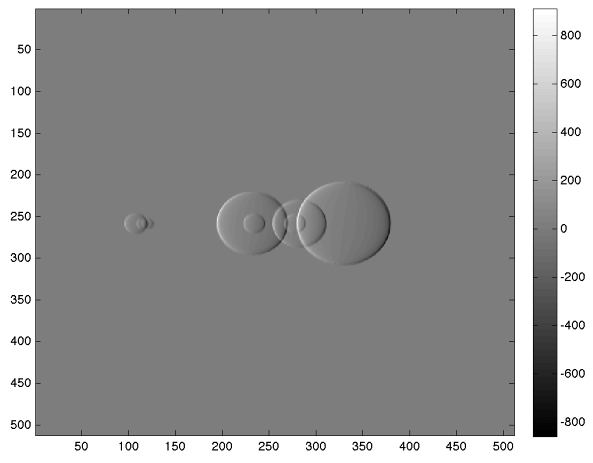


Figure 4.2: Single view of derivative data g_1 taken using Noo derivatives.

Looking at figs. 4.2 and 4.3 we can see several similarities between g_1 and g_{1us}, such as the general position of high positive and negative values on opposite edges of the spheres. Here sharp discontinuities are captured by both derivatives at the spheres' edges, indicating that singular support is being preserved. This implies that

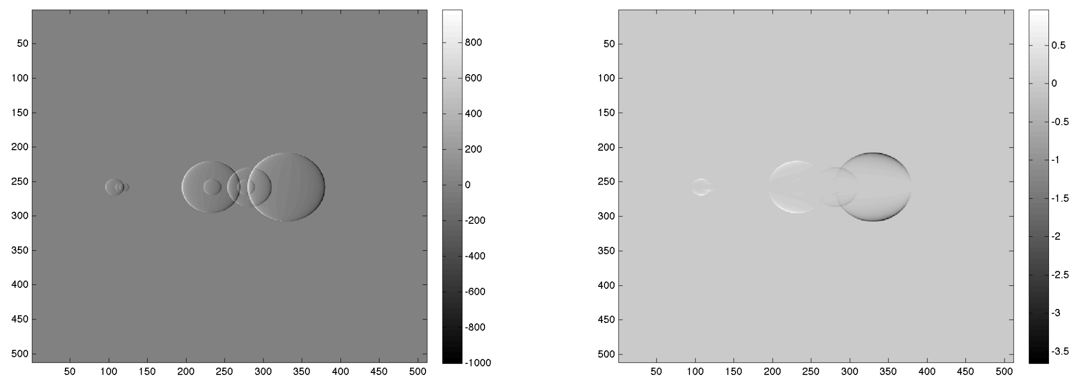


Figure 4.3: Single views of derivative data g_{1us} on the left and g_{1w} on the right taken using Katsevich's derivative formulation.

both formulations of the discrete derivative are equivalent to some pseudo-differential operator.

Looking more closely at the scaling, it can be noted that g_{1us} captures gradients which are larger in magnitude than g_1, meaning the highest gradients are less spread out and hopefully providing higher resolution reconstructions. Figure 4.3 also appears to validate the observation that the values in g_{1w} are generally much lower than those in g_{1us} [9].

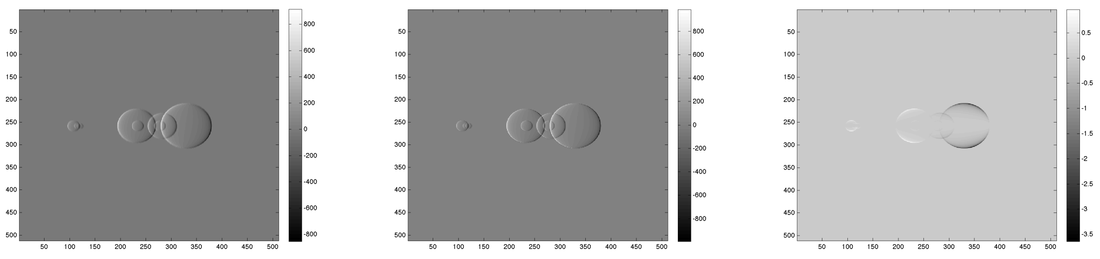


Figure 4.4: Single views of length corrected derivatives g_2 left, g_{2us} centre and g_{2w} right.

The next step is to correct the derivative values for source to pixel length producing the views in fig. 4.4, which are then interpolated into (s, u, ψ) coordinates to produce figs. 4.5 and 4.6. These images appear slightly tilted due to the positioning of the κ-lines on the detector plate effectively stretching and compressing different parts of the image. Again the values produced using the Katsevich derivative are larger and more concentrated towards the edge of the spheres than with Noo's formulation.

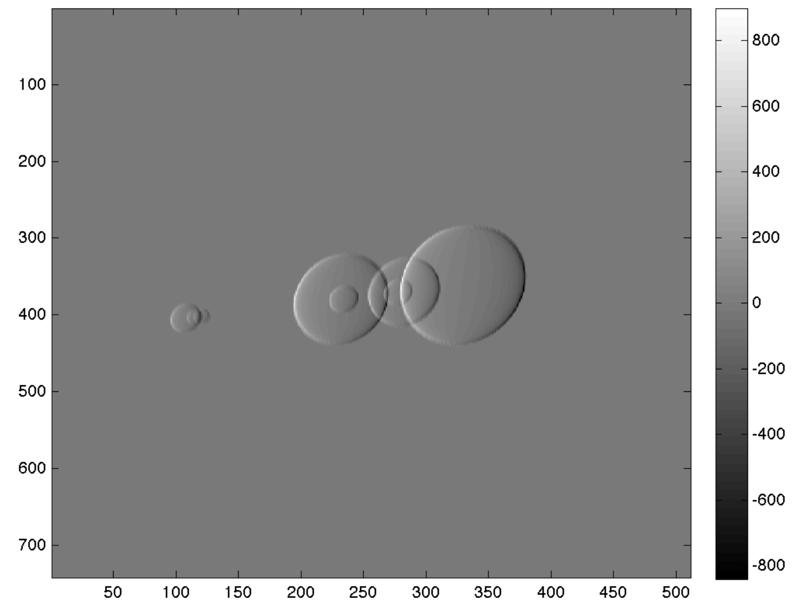


Figure 4.5: Single view of Noo formulation derivatives rebinned to (s, u, ψ) coordinates giving g_3.

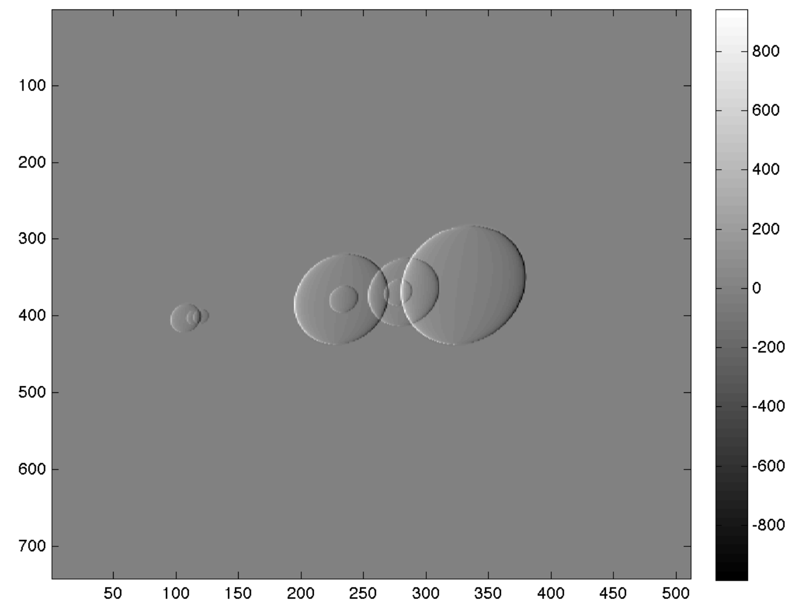


Figure 4.6: Single view of Katsevich formulation derivatives rebinned to (s, u, ψ) coordinates giving g_{3alt}.

Having rebinned to the correct coordinate system, the 1D Hilbert transform can be applied. Other than taking the derivatives, this is the step which produces the most noticeable change in the data as shown in figs. 4.7 and 4.8. These figures show the filtered data required for backprojection. However, the values are given in the wrong coordinate system and include some values from κ-lines which are not needed. Therefore, the next step is to interpolate necessary values to a regular grid, which can be used for backprojection.

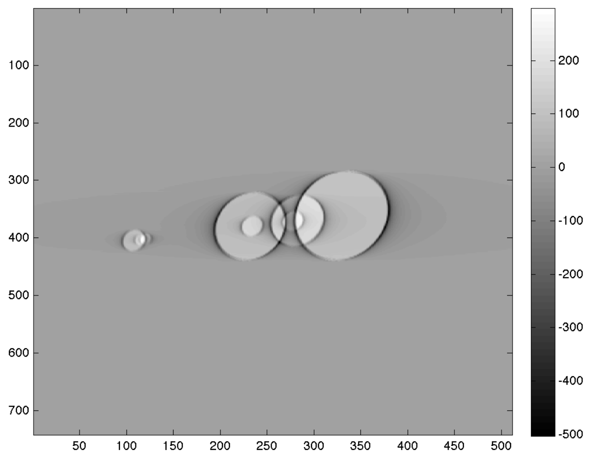


Figure 4.7: Single view of Hilbert transform data g_4 produced using Noo formulation derivatives.

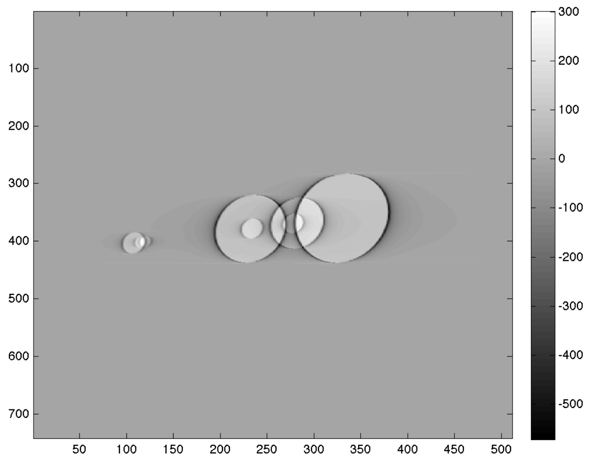


Figure 4.8: Single view of Hilbert transform data g_{4alt} produced using Katsevich formulation derivatives.

Finally, interpolating back to a regular grid gives the views shown in figs. 4.9 and 4.10. Again the values produced using the Katsevich formulation of the derivative seem to be larger making the image appear sharper. However, this method also appears to produce a more noticeable streak of small negative numbers either side of the spheres' projections. These features of the filtered data may explain why the reconstructions using Katsevich's formulation of the derivative appear to have higher resolution but also show much more noticeable artefacts.

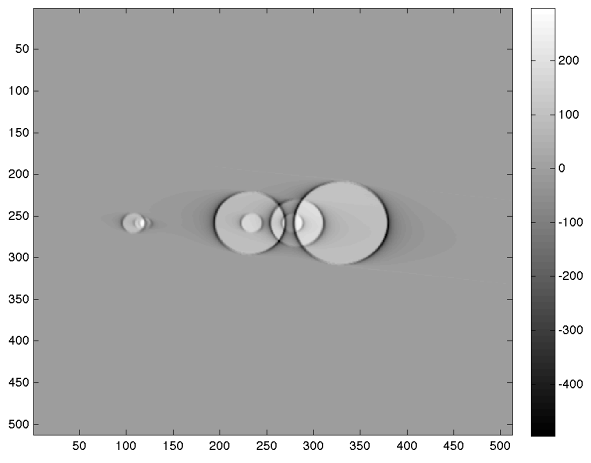


Figure 4.9: Single view of filtered dataset g_5 produced using Noo formulation derivatives.

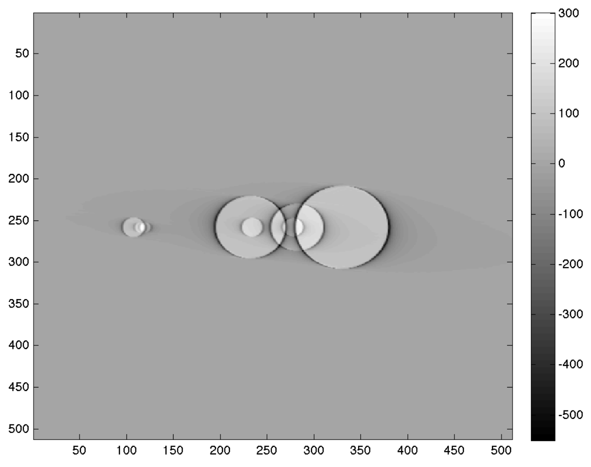


Figure 4.10: Single view of filtered dataset g_{5alt} produced using Katsevich formulation derivatives.

4.1.2 Reconstructions

Figures 4.11 and 4.12 show the central slices of reconstructions produced using the Noo and Katsevich formulations of the derivative respectively. Comparing the two, they appear to be quite similar at this scaling. Both reconstructions are able to show all the spheres, however, the scaling of fig. 4.12 shows that the Katsevich derivative produces a higher maximum value in the overlap region. As the highest value present in the analytic phantom should be 2, this higher value implies that the Katsevich form of the derivative does allow for greater resolution images.

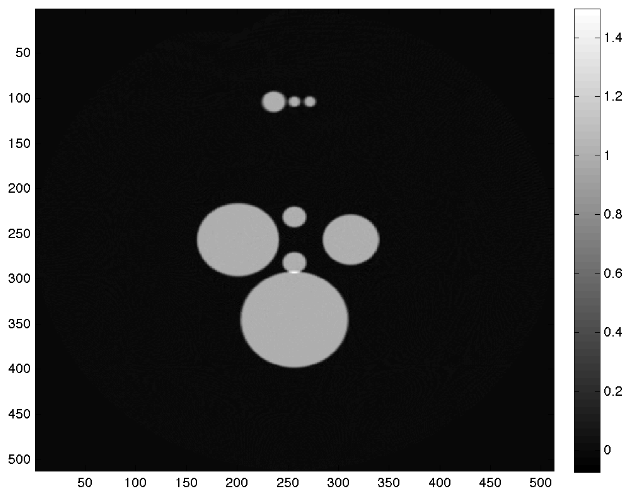


Figure 4.11: Centre slice of Reduced Ball phantom reconstructed with Noo derivatives.

To compare these reconstruction methods more accurately, I have rescaled the images to pick up the discretisation noise and artefacts. The resulting images can be seen in figs. 4.13 and 4.14. Looking more closely at these rescaled images, we can see similar swirling noise patterns and streak like artefacts in both reconstructions. However, the artefacts produced using the Katsevich derivatives appear to have a greater magnitude.

Investigating side views of the reconstructions, it becomes more obvious how the artefacts are situated throughout the volume. Diagonal artefact lines can be seen coming off from the tops and bottoms of the spheres. These become quite prominent at the point where they intersect the spheres, especially when using the Katsevich formulation of the derivative.

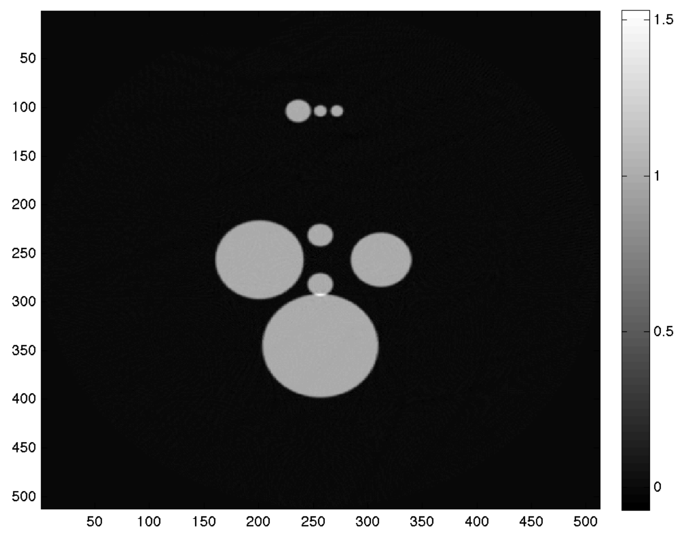


Figure 4.12: Centre slice of Reduced Ball phantom reconstructed with Katsevich derivatives.

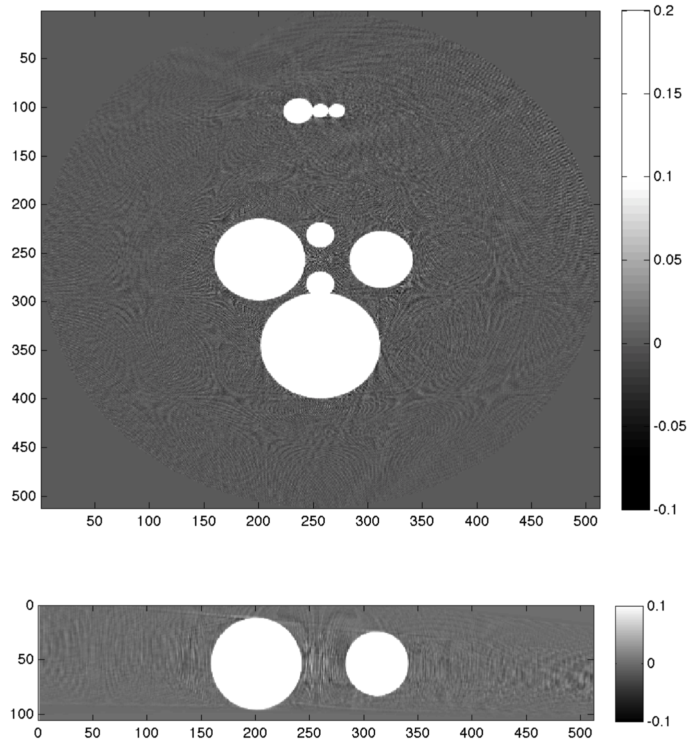


Figure 4.13: Slices through Reduced Ball phantom reconstructed with Noo derivatives, rescaled to show noise and artefacts.

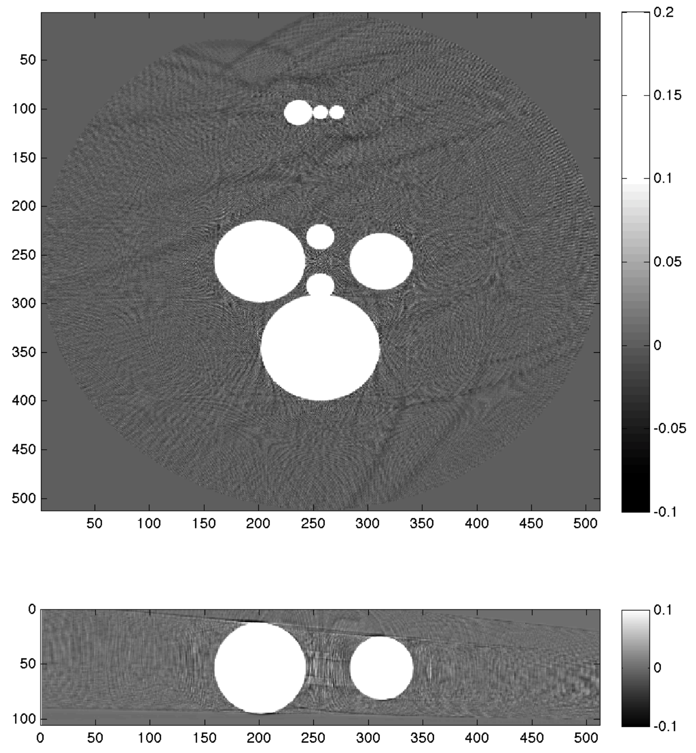


Figure 4.14: Slices through Reduced Ball phantom reconstructed with Katsevich derivatives, rescaled to show noise and artefacts.

Figure 4.15 shows a slice through the top of the largest sphere in the image taken from the Katsevich derivative reconstruction. This image highlights the problem posed by the streak artefacts. As the artefact intersects the top of the sphere it makes a large difference to the values, impacting image quality.

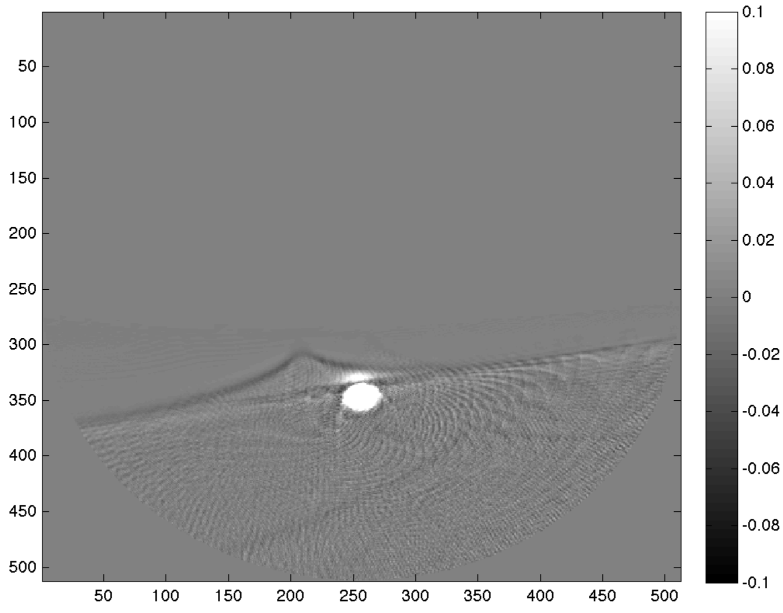


Figure 4.15: Slice through top of Reduced Ball phantom reconstructed with Katsevich derivatives, rescaled to show significant artefacts.

The line profile in fig. 4.16 shows exactly how much of a difference it makes to use an alternate formulation for the derivative. This line profile is taken from column 256 in the image in fig. 4.15 and focuses on the region affected by the artefacts. The Katsevich derivative gives a closer value to the analytic phantom and gives a sharper image but the smoother image of the Noo reconstruction reduces the impact of artefacts.

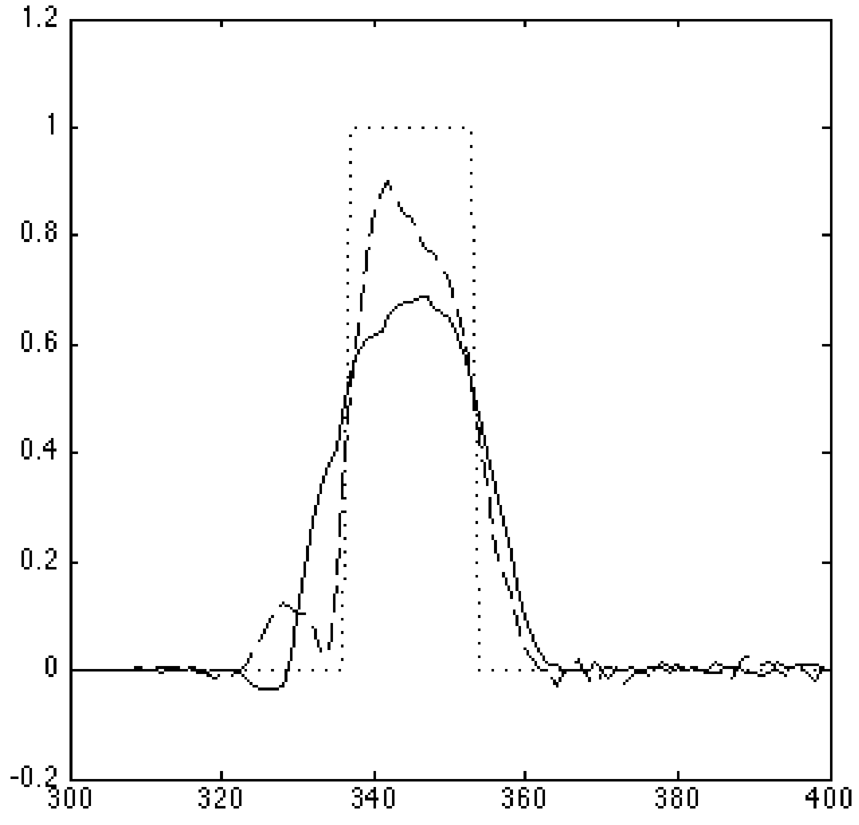


Figure 4.16: Line profile though top slice of the largest ball in the Reduced Ball phantom. Solid line refers to Noo derivative reconstruction, dashed line refers to Katsevich derivative reconstruction and dotted is the analytic phantom.

4.2 Planes of Spheres Phantom

The second phantom I used was Planes of Spheres phantom. As with the Reduced Ball phantom the spheres all have a uniform attenuation coefficient of 1, with overlapping regions which have coefficient 2. Figures 4.17 and 4.18 show reconstructed slices of this phantom using Noo and Katsevich formulations of the derivative respectively. Again there appear to be few differences in the two reconstructions at first glance. However, the Katsevich formulation of the derivatives does still seem to capture the overlap regions more accurately giving values closer to 2 in these areas.

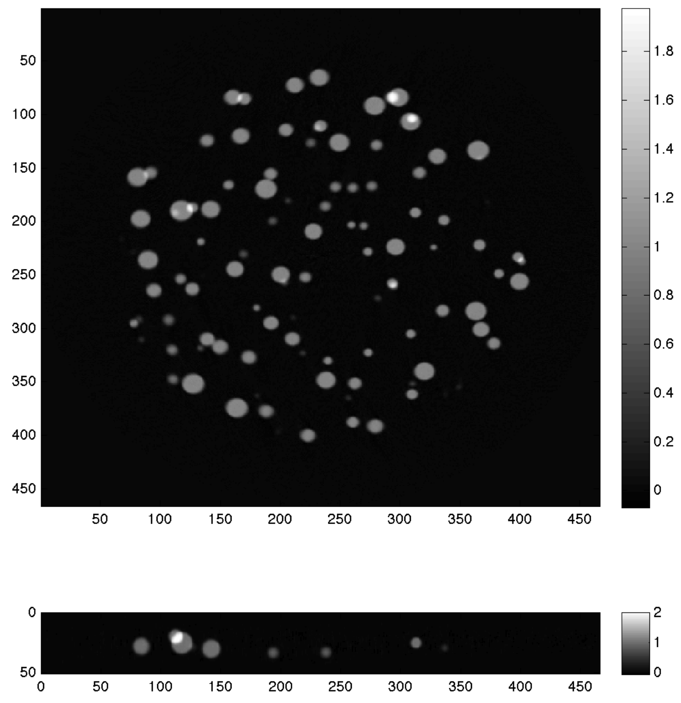


Figure 4.17: Vertical and Horizontal slices of the Planes of Spheres phantom reconstructed using Noo derivatives.

The scaled views also show the same kind of streak artefacts present in the Reduced Ball phantom, although there doesn't seem to be as much difference in these two images. This reduction in the effects of the artefacts may be due to the reduced pitch used for this simulation. It is also worth noting that for circular scan reconstructions of this phantom, high cone angles produce large areas of negative values between spheres.

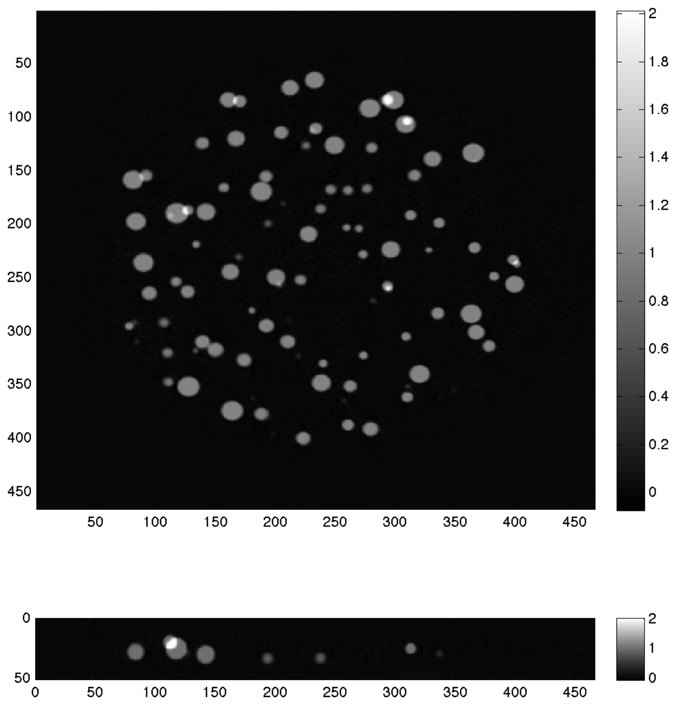
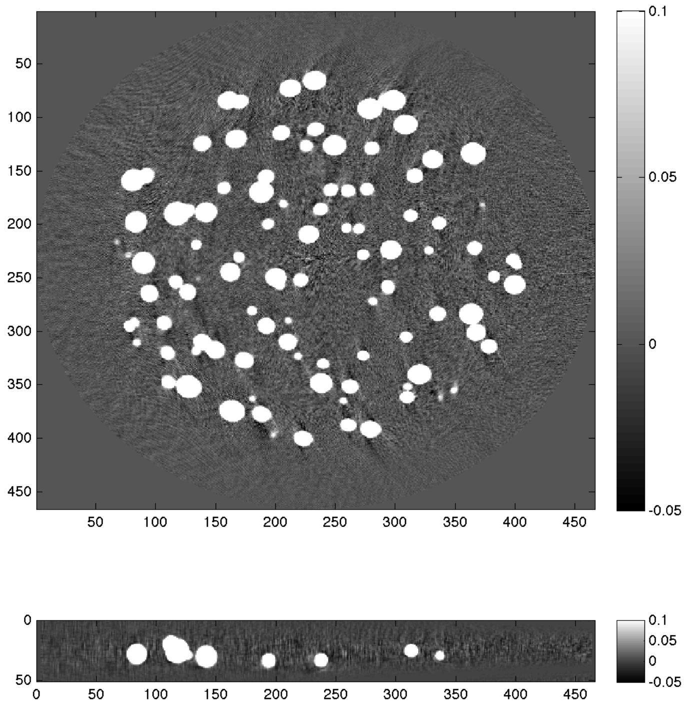
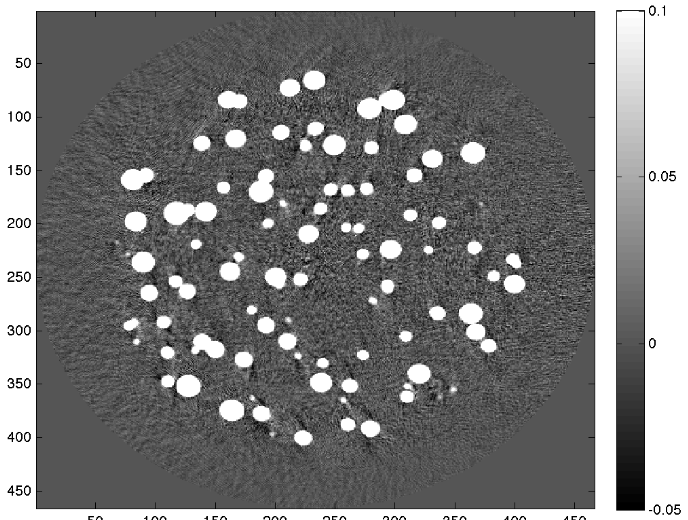
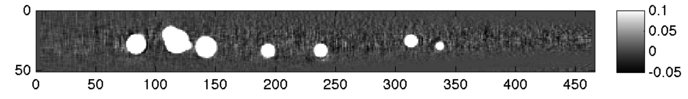


Figure 4.18: Vertical and Horizontal slices of the Planes of Spheres phantom reconstructed using Katsevich derivatives.




50
100
150
200
250
300
350
400
450
50
100
150
200
250
300
350
400
450
0.1
0.05
0
-0.05


0
50
0
50
100
150
200
250
300
350
400
450
0.1
0.05
0
-0.05

Chapter 5

Real Scans

After testing my code on the numerical phantoms shown in chapter 4, I was able to start work reconstructing real scan data. I have reconstructed scans of two samples, a sample of platinum dust in suspension and a sample of two sapphire balls held in styrofoam.

5.1 Differences From Simulations

Using real scan data introduces several additional sources of error to a reconstruction algorithm. When modelling a phantom for numerical testing there are many simplifications made and aspects of the physics involved may be overlooked. This has been explained very well by Nuyts et al. [13] in the context of modeling for iterative reconstruction methods. In this section I will try to summarize some of the points made in Nuyts' paper which can also be applied to direct inversion formulas.

Several of the problems with real scan data occur at the detector plate. For example detector pixels can sometimes affect each other in a process known as *crosstalk*, which can cause blurring between pixels in a single view. A similar blurring effect can be seen when the energy provided to a detector pixel does not dissipate properly between views causing *afterglow*. Additionally the design of the detector can impact the images taken. The scans used for the reconstructions in this dissertation were taken using a machine which uses scintillators to convert X-rays to other forms of light which can be optically magnified. Converting the X-rays is essentially a random process and there is a small possibility that the X-ray will not be absorbed but rather scatter or pass through the detector. Finally, there is also the possibility of electronic readout noise being added to the output of the detector.

In addition to noise effects produced at the detector there are several which originate at the X-ray source. Other than the fact that the source itself has finite size rather than being a point, the main difficulty with this part of the process is producing an X-ray spectrum with a constant distribution of energies. As the production of X-rays within the source is governed by a random process the energies of the photons produced are random variables. This is in addition to the fact that in practice the current supplied to the source is not constant adding more variation.

Errors can also originate within the field of view itself. One problem that can occur is *scattering* of X-rays. When X-rays are deflected from there original path rather than being absorbed they may create cupping or streak artefacts. This effect can also be joined by *beam hardening*, which is caused by non-uniform attenuation of X-rays at different energies. Rather than being an entirely linear process the absorption of photons is dependent on their energy producing cupping and shadow artefacts. This has been partially addressed in this dissertation for one sample reconstruction.

There are also problems more specific to the Katsevich reconstruction algorithm. This is a theoretically exact inversion formula and only uses the minimum required data meaning that errors in parameter measurements do not create blurring but instead can cause significant errors in the reconstruction [17].

5.2 Platinum Dust Sample

The first real scan reconstructed was of platinum dust in suspension. This sample was chosen as it contains many elements which are at the resolution limit of the detector. Figures 5.1 and 5.2 show reconstructions using both Noo and Katsevich formulation derivatives respectively. As with the last numerical phantom there appears to be little difference using either derivative for this pitch value. However, in the rescaled images in figs. 5.3 and 5.4, there do appear to be large streak artefacts as seen before. These are also joined by slight starburst artefacts similar to those seen in other reconstruction methods.

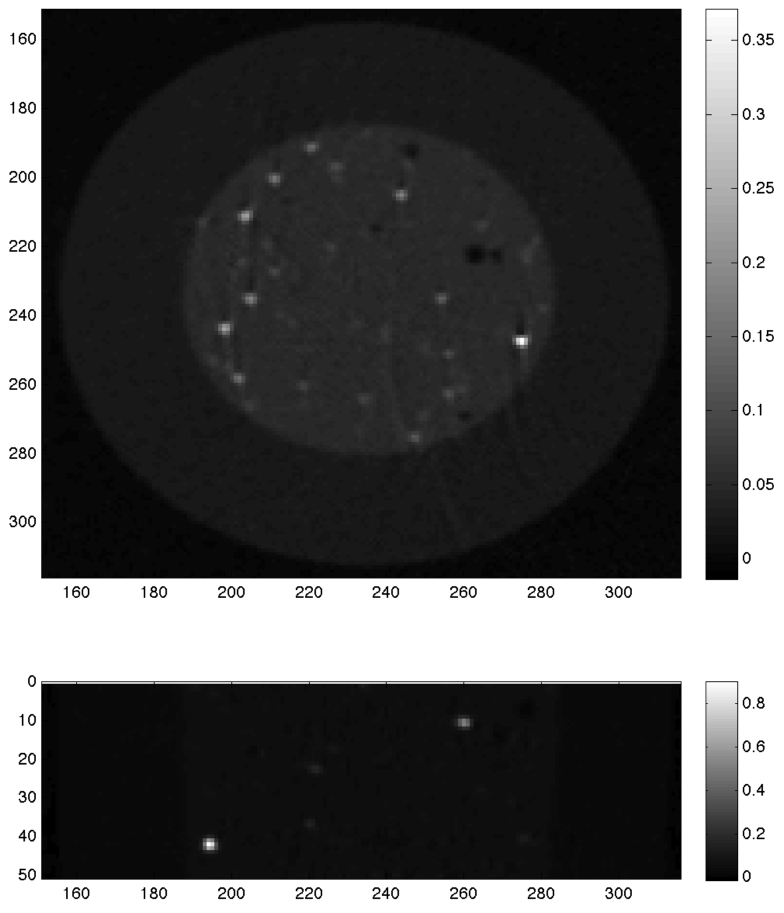


Figure 5.1: Horizontal and vertical slices of reconstructed image of platinum dust in suspension reconstructed using Noo derivatives.

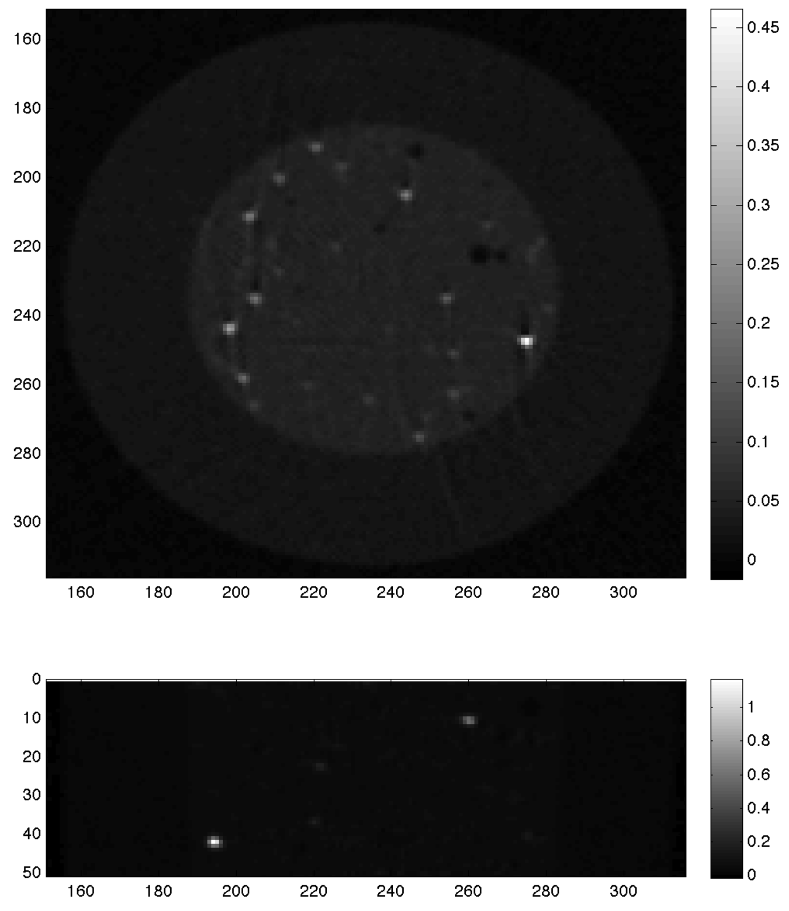


Figure 5.2: Horizontal and vertical slices of reconstructed image of platinum dust in suspension reconstructed using Katsevich derivatives.

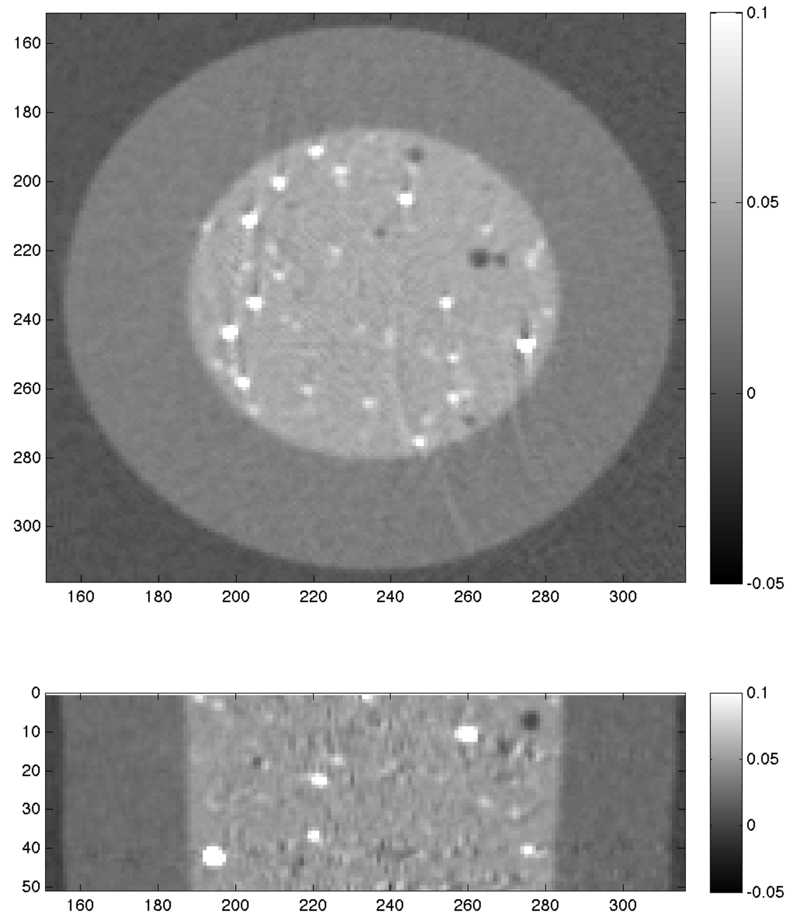


Figure 5.3: Horizontal and vertical slices of reconstructed image of platinum dust in suspension reconstructed using Noo derivatives rescaled to show details and artefacts.

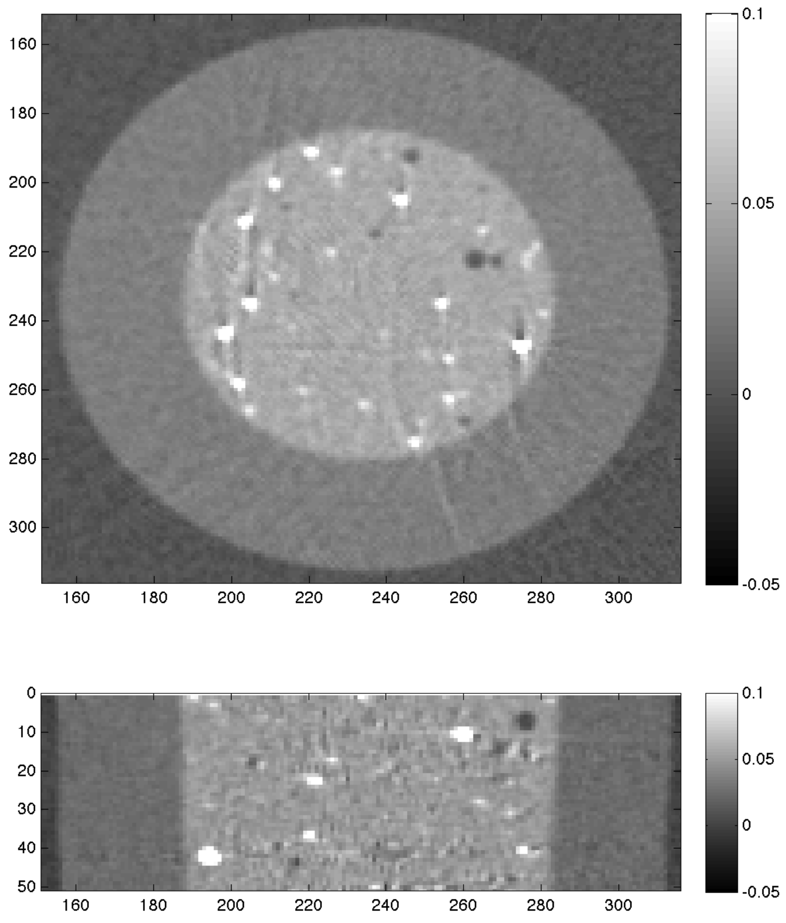


Figure 5.4: Horizontal and vertical slices of reconstructed image of platinum dust in suspension reconstructed using Katsevich derivatives rescaled to show details and artefacts.

5.3 Sapphire Balls

One of the problems with using the platinum sample was the difficulty in removing artefacts not specific to the reconstruction algorithm. A new sample was chosen to check the effect of beam hardening on the Katsevich reconstruction algorithm. The new target object consisted of two sapphire balls held in styrofoam, more closely resembling the original reduced ball phantom and allowing for an empirical beam hardening correction to be calculated. This was done by examining circular scan data of this object and noting values produced where the balls passed in front of each other to find the correction function

$$g_{BH}(s,u,w) = g(s,u,w)(1 + 0.33[g(s,u,w)]^2). \tag{5.1}$$

Three different reconstructions were performed based on this target sample, each using the Noo formulation of the derivative. First the original scan data, then data corrected using the beam hardening correction in eq. (5.1) and finally a numerical simulation of the sample to gauge additional effects on the real scan. Views from the scan data for each of these three reconstructions can be seen in fig. 5.5. Figure 5.6 shows a comparison of the scan data with and without beam hardening correction. As shown in this figure, the correction increases the measured attenuation values to compensate for the non-linear absorption of X-rays.

Figure 5.7 shows the three different reconstructions scaled to their own maximum and minimum values. At these scalings the reconstructions look mostly very similar. The only difference easily visible is that the beam hardening correction increases the maximum values to be closer to those present in the simulation. In order to see the other differences in the images it is necessary to rescale.

Figures 5.8 to 5.10 show the same slices rescaled to show noise and negative artefacts. Looking at fig. 5.8 streak like artefacts can be seen. These are similar to those seen in previous reconstructions but they are made worse by a region of beam hardening artefacts between the two balls. The correction used for fig. 5.9 appears to have brought this area much closer to the image expected from the simulation shown in fig. 5.10. However, it is worth noting that the same curved streak artefacts are present in all three reconstructions to varying extents.

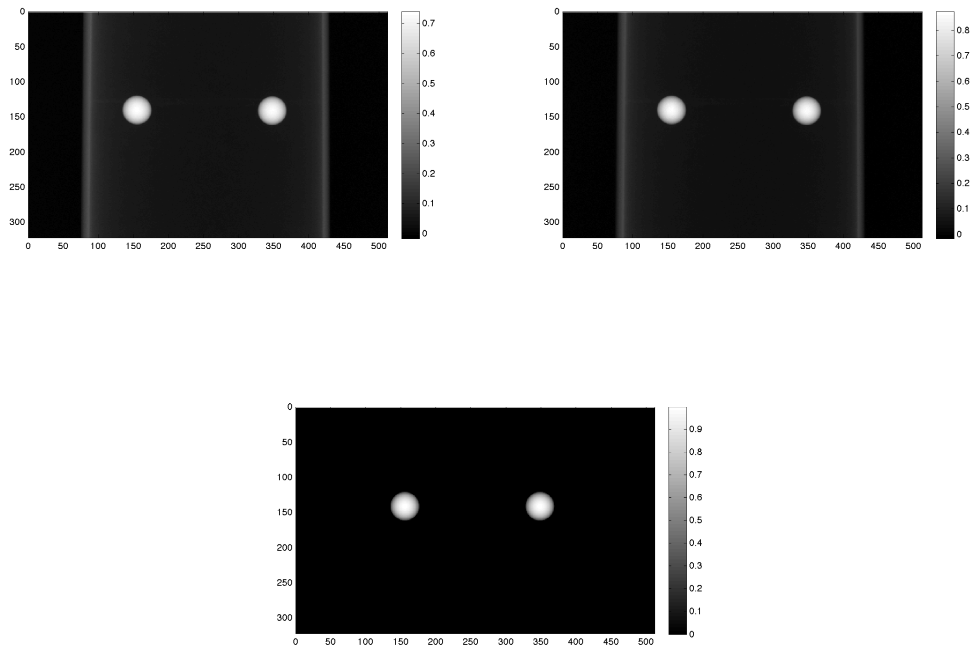


Figure 5.5: Single views of the scan data for the three Sapphire Ball reconstructions. Original scan top left, beam hardening corrected data top right and simulated data bottom.

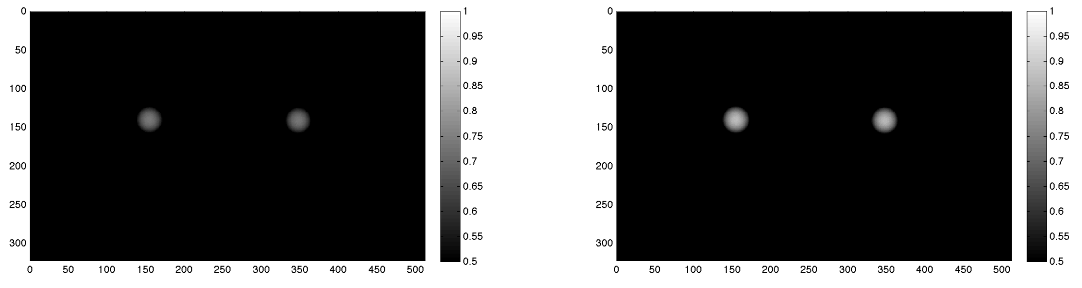


Figure 5.6: Single views from the original scan and beam hardening corrected data rescaled for comparison.

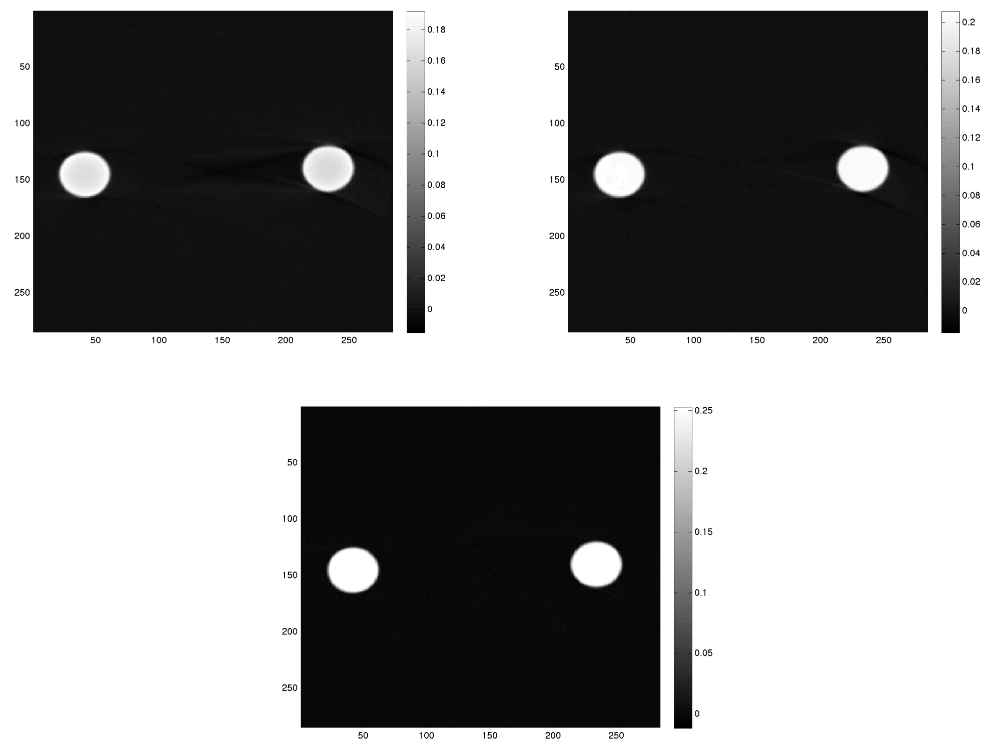


Figure 5.7: Slices from the three Sapphire Sphere reconstructions. Original scan top left, beam hardening corrected top right and numerical simulation bottom.

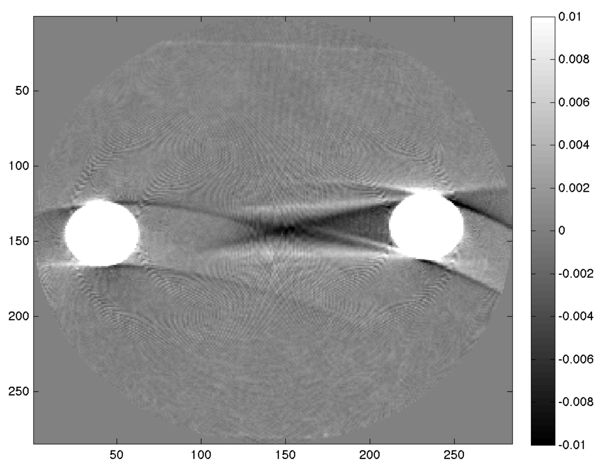


Figure 5.8: Scaled image of uncorrected Sapphire Ball reconstruction.

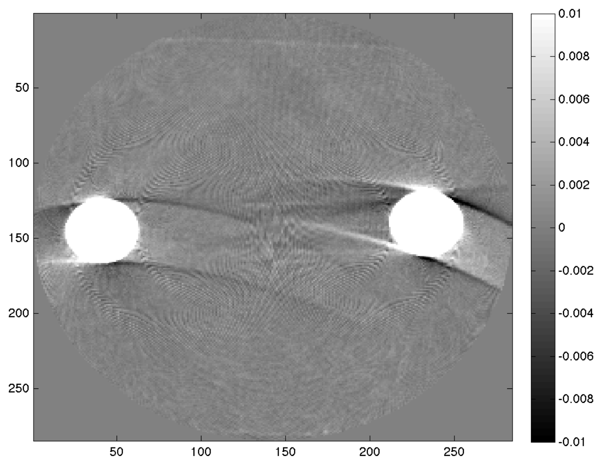


Figure 5.9: Scaled image of beam hardening corrected Sapphire Ball reconstruction.

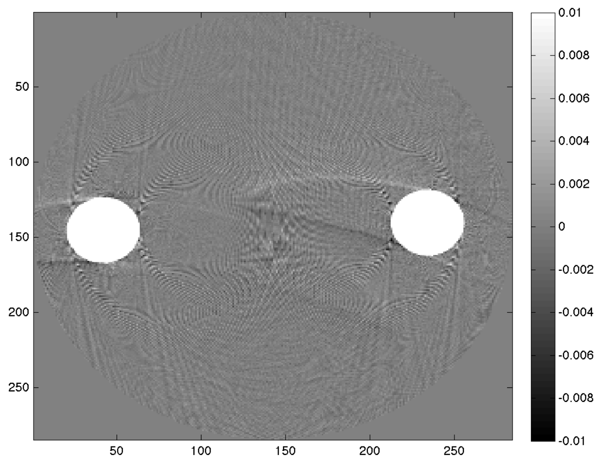


Figure 5.10: Scaled image of numerical Sapphire Ball reconstruction.

Chapter 6

Additional Observations and Improvements

Over the course of this project I have had to make several observations and write additional support code to help reconstruct both the numerical phantoms and the real scan samples. In this chapter I will outline some of the most interesting additional observations and achievements.

6.1 Overscan

One of the main selling points of the Katsevich algorithm is its efficiency. When taking circular scan images of an object, additional redundant views must be taken to account for the approximate nature of the inversion or provide a consistent system of equations for iterative methods. By taking a helical scan of an object and using an exact algorithm we avoid the need for additional redundant data to be collected.

However there are still limitations on the minimum range of scans which must be taken to reconstruct any given slice. While the reconstruction of any given voxel requires data from at most $180°$ plus the fan angle, this domain of dependence is not necessarily symmetrical around the voxel. So for a fan angle of $40°$, while one voxel in a layer may require data from $-110°$ to $+110°$ another may require data from $-120°$ to $+100°$. This makes determining the range of views required for a given reconstruction non-trivial.

The method I have implemented to determine this range is to calculate the surface traced out by all the Pi-Lines intersecting the FOV above a given source position, as shown in fig. 6.1. This process is equivalent to calculating the top of the TD window

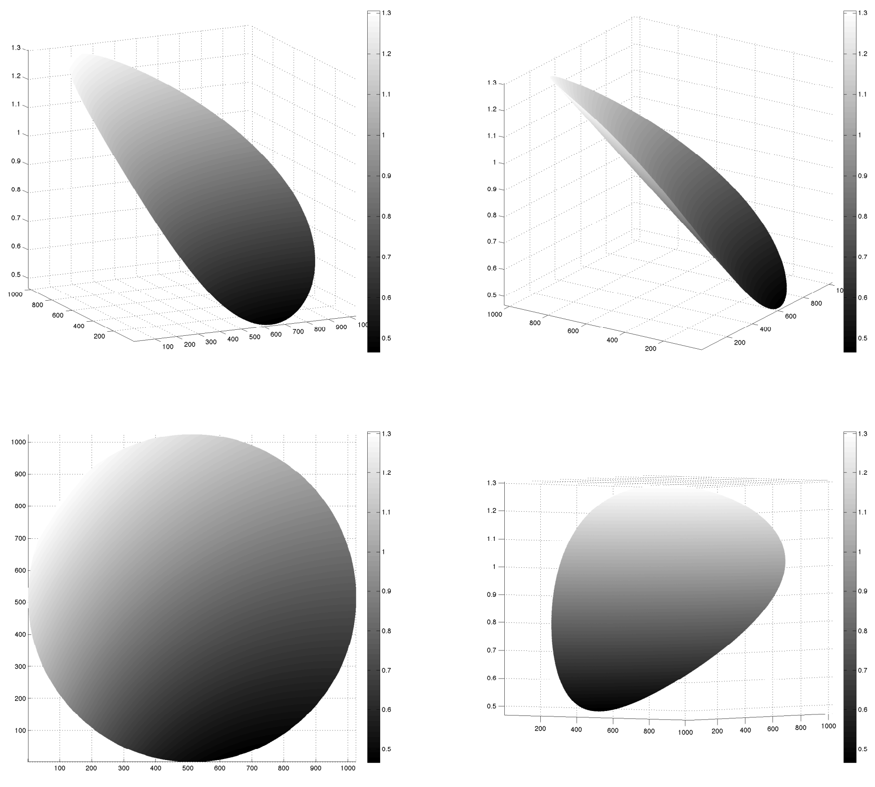


Figure 6.1: Minimum reconstructable heights in FOV for data starting at a given helical source position

when placing the detector within the radius of the FOV. As such this surface can be given, in a similar way to eqs. (2.5) and (2.6) by the equation

$$w_{surf}(u,v) = \frac{P}{2\pi RD}(u^2+v^2)(\pi/2 - \arctan(u/v)). \tag{6.1}$$

This has been implemented in the code shown in appendix B.3. Due to the symmetry of the helix, this surface gives the shape of both the top and bottom of the reconstructable volume.

To determine the required overscan for a given cylindrical volume I have implemented code which produces the surface shown in fig. 6.1 and takes the maximum value. The overscan at each end of the volume can then be calculated as

$$\text{Overscan} = \max_{u,v\in\text{FOV}}\left(\frac{2\pi w_{surf}(u,v)}{P}\right). \tag{6.2}$$

6.2 Scan Direction and Helix Type

Unlike with circular scan CT, the direction the source moves in plays an important role in reconstructing helical scan CT. With the addition of motion in the vertical direction there is the problem of whether the source is moving up or down and whether it is moving along a left or right handed helix. These details impact several steps in the reconstruction and pose a problem for maintaining a code base.

Reconstructing data taken on the same helix type, either right handed or left handed, is relatively simple. Due to the discrete nature of the scans, reconstructing data taken with the source moving down is equivalent to running the reconstruction for the source moving up with the views taken in reverse order. Therefore to move between the two you can just reverse the order of the views and run the same code.

Moving between left and right handed helices is more difficult. Changing between these two helices changes the geometry of both the kappa lines and TD window on the detector plate and can cause problems to reconstruction. Ideally we would like code to be able to reconstruct both left and right handed helices.

The easiest way to get around this problem is through taking reflections of the data, helix and sample. When a right handed helix is reflected in a plane it becomes a left handed helix with the same radius and pitch. All that is needed is to show that projections taken from the reflected helical source path of the reflected object are equivalent to a simple reflection of the scan views. With this we can reconstruct the reflected object and transform back to obtain the required reconstruction.

Notice reflections are isomorphic up to a rotation. Therefore, reflecting in the plane defined by the source position and the line running vertically down the centre of the detector plate is equivalent to the same reflection for any other source position. This means we can perform the reconstruction by reflecting our views in the vertical central line, reconstructing on the type of helix we have code for and then reflecting the resulting image back into place.

As the real scans were provided for the source position moving down the opposite helix type to that used in the code, both of these transformations were used to produce results during this project.

6.3 Streak Artefacts

One of the most common artefacts found in my reconstructions is the curved streak artefact, usually coming from the top and bottom of the objects in question. These streaks tend to be negative attenuation coefficients sweeping from the top and bottom of the objects and have reached values in the order of 10% of the attenuation coefficient of the object.

The position and angling of these artefacts suggests that it may have something to do with the interpolation steps to and from the filtering lines. When interpolating along a vertical edge the linear interpolation does not cross a discontinuity and can relatively accurately estimate the value of the derivative at these points. However, at the top of the projection the interpolation is acting between the very high magnitude derivatives of the objects edge and the zero values outside. This discontinuity may cause fill in which could be spread by the Hilbert transform step.

This may explain the difference in the level of artefacts seen between the reconstructions performed with both Katsevich's and Noo's versions of the derivative. As the Katsevich derivative retains greater spacial resolution by providing higher derivatives towards the edges of objects it also produces higher values along edges not tangential to the interpolation direction. This means that although there is a sharper picture with less of a halo effect there is also greater fill in near edges normal to the filtering direction.

If this interpolation is the cause of the artefacts, reducing the detector pixel size should produce improvements as a smaller proportion of the filtering line elements will experience fill in. A more sophisticated interpolation scheme may also produce improvements if it can take into account the *Hölder Continuity* of the data being interpolated.

Some form of basis functions may also provide improvements. In finite elements gradients are calculated, not through finite differences as with our code, but by adding the gradients of the basis functions at any given point. This may be an interesting way of taking the required derivative along the filtering lines. If an appropriate basis function can be found, combining finite elements and finite differences it could be possible to find the derivative at the required filtering points and cut out the interpolation step entirely.

One method to test this hypothesis may be to produce a modified forward projection. There exist forward projectors with the capability to project rays close enough to each other to take the required derivative directly. It would also be possible to take these derivative projections on the filtering lines themselves so as to avoid any interpolation before the Hilbert transform step.

6.4 Norms

One thing to keep in mind when discussing the errors of reconstructions is the method with which we quantify errors in the data. One useful method of comparing reconstruction algorithms is to compare line profiles, taking a single line to more easily view noise and gradients. This allows for visual comparison of specific interest areas and gives an idea of resolution and noise effects. However, these are qualitative measures and highly susceptible to bias when choosing both the slice and line to be profiled.

Another popular method of quantifying error is to take the L^2 norm. This form of error norm gives a more accurate and balanced accounting of the error in an image, however, it does not provide us with data about how well the image captures edges or specifics from the geometry of our object.

A more accurate method is recommended by Natterer [11], namely to examine the geometry of smooth objects with defined edges and take the norm for the Sobolev space these kinds of objects are likely to inhabit. Doing this generally involves taking the norm of a fractional Sobolev space as the smooth discontinuities in our images tend to lead to pictures which are *Hölder Continuous* to some fractional power. This is possibly the most accurate and objective method for determining the validity of a reconstruction method. Again there are drawbacks though, including the difficulty in implementing a fractional Sobolev space norm as well as the fact that it may be more than required when comparing the reconstructions to discrete approximations of the underlying continuous image.

This last point prompts the question of what our error norm is required to evaluate. For example, the first measure of any reconstruction is usually how good it looks to a human observer. This is measured by how easy it is for the human eye to detect edges, how smooth constant regions appear and the presence of noticeable artefacts. Therefore, if we are trying to quantify how good the reconstruction will look to a human observer the obvious norm to use would be one which mimics the human brain's edge detection filtering mechanisms as closely as possible.

Another way to choose the norm may be to instead look at the purpose of the scan. With the large number of reconstruction methods available the purpose of the scan can often determine the type of scan done as well as the reconstruction method. For scans of time dependent or moving phenomena helical scan is usually chosen over circular for its acquisition time, while the ability of iterative methods to resolve anisotropic voxel grids may recommend it for specific purposes. Within practical uses of X-ray CT it may be more useful to quantify the accuracy of certain features within the image, rather than evaluate how close an image is likely to be to an exact reconstruction.

6.5 Katsevich Derivative Bounds

When examining the derivation of the derivative proposed by Katsevich [9] a problem arrises with the proposed bounds on his parameter r. In this paper the bounds given are derived purely from the relations between the weightings used in the interpolation step rather than the full expressions of the basis functions themselves. When looking at these values as functions of δ as shown in eq. (3.9) it becomes apparent that

$$\begin{aligned} r &= \frac{a_3}{\delta}, \\ &= \frac{1}{2\Delta s}\left(\frac{2U_s'\delta}{\Delta u} + 1\right). \end{aligned} \tag{6.3}$$

Looking at r as a function of δ we must first find the acceptable bounds on delta for this to represent interpolation rather than extrapolation. From fig. 6.2 it can bee seen that there are four different scenarios to consider. When $|U_s'\Delta s| > \Delta u/2$ we must restrict delta such that the interpolation point remains between $\pm\Delta u/2$. This gives $0 \le \delta < \Delta u/(2|U_s'|)$, so by plugging these limits in to eq. (6.3), for positive U_s', r must be in the range

$$\frac{1}{2\Delta s} \le r < \frac{1}{\Delta s}. \tag{6.4}$$

Similarly, for negative U_s', r must be in the range

$$0 < r \le \frac{1}{2\Delta s}. \tag{6.5}$$

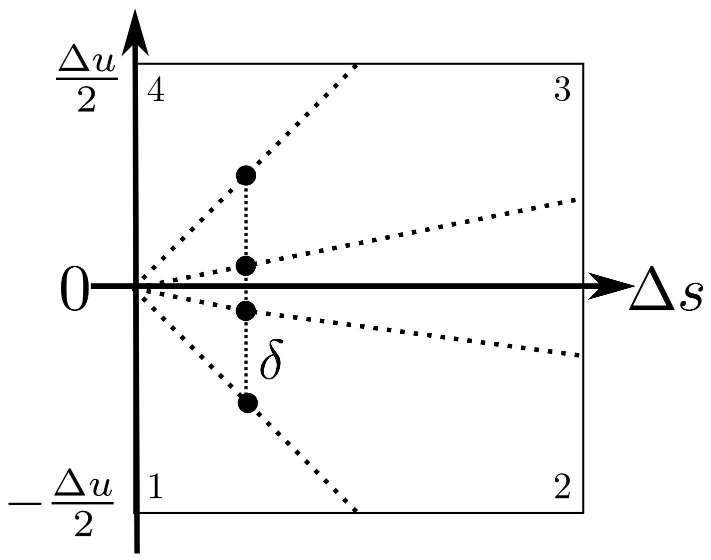


Figure 6.2: Diagram showing the four possible scenarios for different values of U'_s.

The other two situations are covered by the case where $|U'_s \Delta s| \leq \Delta u/2$. This gives $0 \leq \delta < \Delta s$, so by plugging these limits in to eq. (6.3), for positive U'_s, r must be in the range

$$\frac{1}{2\Delta s} \leq r \leq \frac{U'_s}{\Delta u} + \frac{1}{2\Delta s}. \tag{6.6}$$

Similarly, for negative U'_s, r must be in the range

$$\frac{U'_s}{\Delta u} + \frac{1}{2\Delta s}. \leq r \leq \frac{1}{2\Delta s}. \tag{6.7}$$

This shows that, while the bound suggested by Katsevich [9] is technically correct, it is not restrictive enough in all cases and some choices of r may lead to extrapolation rather than interpolation.

It is also worth noting that the choice $r = 1/2\Delta s$ always corresponds to the choice $\delta = 0$. Choosing this value for delta means that the value produced is the average of the directional derivatives of the bilinear interpolations on the regions either side of the point $(s_i, u_{k+0.5})$. This highlights the ability of basis functions to find approximations to derivatives at general points. Using this it may be possible to merge the difference step with the forward rebinning step using trilinear basis functions to approximate the gradient at a general point and find

$$\frac{\partial}{\partial s} g(s, \boldsymbol{\theta}) \simeq \frac{1}{\sqrt{1 + (U'_s)^2 + (W'_s)^2}} \begin{pmatrix} 1 \\ U'_s \\ W'_s \end{pmatrix} \cdot \nabla G, \tag{6.8}$$

where G is the trilinear interpolant of g.

Chapter 7

Conclusions and Further Work

Over the course of this project I made substantial changes to Wunderlich's code. In order to start working on real scan data, the efficiency of the code has had to be increased. Moving from using a 138×16 pixel detector to a 512×512 detector has meant making changes such as: reorganising the code for ease of editing, adding code to save intermediate steps for analysis, restructuring for loops, removing inefficient or redundant code and vectorising operations to take advantage of MATLAB's implicit parallelism.

I have also produced solutions for problems when reconstructing different datasets. For example a method for reconstructing data taken on any helical source curve. Other solutions have resulted in the support files in appendix B, which calculate the maximum allowable pitch, minimum number of filter lines and maximum reconstructable volume.

Sections of this dissertation also touch on analysis of the possible causes for artefacts and different methods for evaluating results. These sections are only a brief outline and should be visited again in the future along with other possible research areas such as: implementing image processing techniques such edge aware interpolation [14] to increase resolution, modifying the smoothing function for backprojection based on whether the backprojector is voxel driven or pixel driven, looking at how to reduce the time required for π-line calculation and implementing code to optimise the parameters used as stated in Varslot et al. [17].

Ultimately I have also been able to perform reconstructions on three numerical simulations and two real samples using two implementations of the Katsevich reconstruction algorithm. However, the most interesting and significant finding in this dissertation may be the implication that taking the wrong bound whilst implementing Katsevich's form of the derivative can cause extrapolation rather than interpolation. Therefore I would recommend research into the effect of using these different bounds as well as the possibility of using trilinear basis functions to calculate the derivatives without a further interpolation step.

Appendix A

Implementation Scripts

I have included the scripts required to reconstruct an image using Katsevich's formulation of the derivative to demonstrate most of the changes I have implemented. I have also made efficiency changes to the Noo formulation code, resulting in code that is mostly similar to these script files.

The support functions I have written can be found in appendix B. However, for support functions `findPI.m` and `PIfun.m` or an implementation of Noo formulation derivatives please see Wunderlich [18].

A.1 `katsetup.m`

```
%% Parameters
%
% This section sets up user options and geometry values for
% use in reconstructing with the Katsevich formula.
%
% The user options determine which optional steps will be
% performed as well as the location and names of save files.
%
% user defined parameters for geometry currently include
% the number of detector elements as well as the shape and
% position of the detector.
%
% Pitch is currently calculated to use close to the maximum
% available area for the TD window while limiting complexity
% of Pi-line calculations and giving room for pixels either
% side of the window.
%
% The number of filter/kappa lines is calculated to ensure
% lines are never more than 1 detector element appart,
  reducing
```

```
% the loss of resolution from interpolating to these lines.
%
% Running this script requires funtions calcPitch.m and
% calcFilterLineNo.m
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% clear all
% gohome

% user options
PIfile        =  false;  % use file for PI-intervals?
calc_phantom =  true;  % calculate the original phantom?
show_phantom =  false;  % plot original phantom?
show_recon    =  false;  % plot reconstruction?
circle =        true;    % only reconstruct inside circle
  inscribed in ROI?
LPfiltering =   false;   % LP filter after Hilbert transform?
pre_interp =    false;   % use preinterpolation option before
  backprojection?
save_steps =    true;   % save intermediary steps

datdirpath = strcat(pwd,'/data'); % the directory in which all
   data files will be saved
PIdirpath  = strcat(datdirpath,'/PIfiles'); % the directory in
   which all PI files will be saved
datfilename = 'FPsaphire'; % general name of scan files
PIfilename ='PIfile_zp'; % general name of PI files


% user defined parameters for geometry
% Z = -0.25;    % which slice to reconstruct?
% M = 512;   % number of detector rows  -- take to be even
N = 512;   % number of detector columns
% height = 40;  % detector height mm
R = 80;  % helical scanning radius mm
D = 112.623;  % source to detector distance mm
%
% min_obj_z_value = -0.46*12; % begining of the scanned object
% max_obj_z_value = -0.04*12; % end of the scanned object

SourcesPerTurn = 720;  % number of source positions per turn (
  even)
% delta_w = height/M; % detector element height
delta_w = 69.337/511;
delta_u = delta_w; % detector element width
delta_s = (2*pi/360)/3;  % stepsize between source positions
```

```
width = N*delta_u;
alpha_m = atan(width/(2*D));  % half fan angle for FOV
r = R*sin(alpha_m);     % FOV radius
ROI = [-r r -r r]; % region of interest


% P = calcPitch(height,R,D,N);   % helical pitch (per turn)
% P = 46.527410053023878;
P = 45.708479106849403;
h = P/(2*pi); % alternate expression for pitch (per radian)
% M = minRows(R,D,P,delta_w,width);
M = 322;
height = M*delta_w;
L = calcFilterLineNo(height,R,D,M,P);   % number kappa-lines
  on detector (even)


% these are testing variables left over from the orriginal
  Wunderlich code
Q = 2;   % pre-interpolate to a Q-times denser grid
mexp = 0;    % phantom will be mexp-1 times differentiable
delta = 0;   % detector shift (usually either 0 or 1/4)
```

A.2 katinit.m

```
%% initialization and preprocessing
%
% This section sets up voxel, detector element and source
% possitions as well as arrays and coordinate systems for
% further calculations.
%
% It also determines the filepath for Piline files based on
% the current geometry for testing purposes.
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

s_range = [-128.50*pi/180,131.50*pi/180];
% interval of s values on helix

s = s_range(1):delta_s:(s_range(2));  % s samples
K = length(s);  % total number of source positions

hx = R*delta_u/D;   % stepsize for x-dim
hy = hx;   % stepsize for y-dim
```

```
hz = hx;

center = [(ROI(1)+ROI(2)), (ROI(3)+ROI(4))]/2;   % center of
  ROI
% radius = min(ROI(2)-ROI(1),ROI(4)-ROI(3))/2; % radius of
  inscribed circle
radius = R*sin(atan(144*delta_u/D));

MX = 2*ceil(radius/hx);   % x-dim of reconstruction matrix
MY = MX;   % y-dim of reconstruction matrix

% x = ROI(1)+hx*(0:MX-1);
x = zeros(1,MX);% range of x-values in reconstruction
x(MX/2+1:MX)=hx/2:hx:hx/2+(MX/2-1)*hx;
x(1:MX/2) = -fliplr(x(MX/2+1:MX));

y = x';    % range of y-values in reconstruction
% z_slices=min_obj_z_value+hz/2:hz:max_obj_z_value;
z_slices=zeros(1,51);
z_slices(27:51)=hz:hz:hz*25;
z_slices(1:25)=-fliplr(z_slices(27:51));
z_slices=z_slices+s(391)*h;

min_obj_z_value = z_slices(1)-(hz/2); % begining of the
  scanned object
max_obj_z_value = -1*min_obj_z_value; % end of the scanned
  object

PIfilepath = strcat(PIdirpath,'/pitch_',num2str(P),...
    '/hradius_',num2str(R),'/hstep_',num2str(delta_s),...
    '/ROI_x_',num2str(x(1)),'_',num2str(x(length(x))),'_y_'
      ,...
    num2str(y(1)),'_',num2str(y(length(y))),...
    '/',num2str(MX),'x',num2str(MY),...
    '/minz_',num2str(min_obj_z_value));


u = zeros(N,1);   % u samples
w = zeros(1,M);   % w samples
for j=1:N,
   u(j) = ((j-0.5)+delta-N/2)*delta_u;
end
for j=1:M,
    w(j) = ((j-0.5)-M/2)*delta_w;
end

SourcePos = zeros(K,3);   % source positions
for i=1:K,
    % the source is currently moving allong an anticlockwise
      helix as long
    % as h is positive
    SourcePos(i,1) = R*cos(s(i));
    SourcePos(i,2) = R*sin(s(i));
    SourcePos(i,3) = h*s(i);
end
% detector coordinate unit vectors
e_u = zeros(3,1); e_v = zeros(3,1); e_w = [0;0;1];
theta = zeros(3,1);  % direction unit vector
```

A.3 katdata.m

```
%% compute/load cone beam data
%
% This section loads or calculates forward projection data
% for use in the reconstruction.
%
% Currently data is calculated for an anticlockwise helix by
% tracing one ray per detector element using the function
% redball_forward.m. This function calculates the intersection
% of a given ray with the analytic phantom and returns an
% attenuation value.
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check for or create a directory to store the scan data
scanfilepath = strcat(datdirpath,'/scanfiles/z_',num2str(
  min_obj_z_value)...
    ,'_',num2str(max_obj_z_value));

if ~(exist(scanfilepath,'dir'))
    mkdir(datdirpath,strcat('scanfiles/z_',num2str(
      min_obj_z_value)...
    ,'_',num2str(max_obj_z_value)))
end

% if there exists a scan file in the designated path load the
  file.
% Otherwise start to calculate a new forward projection.
if exist(strcat(scanfilepath,'/',datfilename,'.mat'),'file')
   disp('loading␣cone-beam␣data')
   load(strcat(scanfilepath,'/',datfilename),'g');
   % add noise
   % noise = rand(K,N,M)*.01;
   % g = g+noise;
else
```

```
    g = zeros(K,N,M);  % array of x-ray data -- variables are(s
      ,u,w)

    disp('computing␣cone-beam␣data')
    fprintf('\ncalculating␣%d␣views\n',K);

    % cm is a value relating to the differentiability of the
      phantom in use.
    % it was evaluated inside the inner for loop which
      drastically increased
    % run time.
    cm = (2^(2*mexp+1))*(gamma(mexp+1)^2)/gamma(2*mexp+2);

    datatime = 0;

    % calculate the attenuation of a single ray hitting each
      detector
    % element at each step along the helix.
    for m=1:K,   % s-loop

        fprintf('calculating␣view␣%d␣of␣%d\n',m,K);
        tic;
        % basis vectors on the detector plane
        e_u = [-sin(s(m));cos(s(m));0];
        e_v = [-cos(s(m));-sin(s(m));0];
        for i=1:N,  % u-loop
            for j=1:M,  % w-loop
                % unit vector in direction of measured ray.
                theta = (u(i)*e_u+D*e_v+w(j)*e_w)/sqrt(u(i)^2+D
                  ^2+w(j)^2);
                % calculate attenuation.
                g(m,i,j) = redball_forward(SourcePos(m,:),theta
                  ,mexp,cm);
            end
        end

        newtime = toc;

        fprintf('timetaken␣for␣view␣=␣%g\n',newtime);

        datatime = datatime + newtime;

        fprintf('total␣time␣taken␣=␣%g\n\n',datatime);

    end
    clear cm newtime
    save(strcat(scanfilepath,'/',datfilename),'g');
end

% clear redundant variables.
clear a theta mexp e_u e_v e_w
```

A.4 altkatdiff.m

```
%% compute derivatives
%
% This section loads or calculates components of derivatives
% with respect to helical step at constant ray dirrection
% using the modified chain rule method given by Katsevich
  2011.
%
% The array g1us gives the derivative with respect to s at
% constant height on the detector. These values are given at
% points half way between orriginal detector columns.
% (ie. ( s , u+0.5 , w) )
%
% The array g1w gives the derivative with respect to w at
% constant helical step. These values are given at points half
% way between detector columns and rows.
% (ie. ( s , u+0.5 , w+0.5) )
%
% The desired derivative will be calculated on the filtering
% lines by linear interpolation at the forward rebinning step.
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('computing␣derivatives')

% shift s,u,and w arrays to agree with derivative computation
s = s(2:K-1);
u = u(1:(length(u)-1))+delta_u/2;
% w = w+delta_w/2;

if exist(strcat(datdirpath,'/intsteps/',datfilename,'/g1us'
  ,...
        '.mat'),'file')
   load(strcat(datdirpath,'/intsteps/',datfilename,'/g1us'));
else

%     a=1;
    b=(1/delta_s);

    g1us = zeros(K-2,N-1,M);  % array of derivatives

    % calculate the s derivative using the formula 2.6 in
      Katsevich's
    % note on computing the derivative at a constant
      dirrection 2011.
    % here the interpolation value r=a3/delta has been taken
      as 1 for
    % simplicity.
    for i=1:(N-1)
        g1us(:,i,:)=((g(3:K,i+1,:)-g(1:K-2,i,:))...
                    +(b - 1)*(g(3:K,i,:)-g(1:K-2,i+1,:))...
                    +(b + 2*(u(i)^2+D^2)/(D*delta_u) - 2)*...
                        (g(2:K-1,i+1,:)-g(2:K-1,i,:)));
    end

    g1us=0.5*g1us;

    % saving the derivatives
    if save_steps
        if ~(exist(strcat(datdirpath,'/intsteps/',datfilename)
          ,'dir'))
            mkdir(strcat(datdirpath,'/intsteps'),datfilename)
        end
        save(strcat(datdirpath,'/intsteps/',datfilename,'/g1us
          '),'g1us');
    end
end


if exist(strcat(datdirpath,'/intsteps/',datfilename,'/g1w',...
        '.mat'),'file')
   load(strcat(datdirpath,'/intsteps/',datfilename,'/g1w'));
else

    g1w  = zeros(K-2,N-1,M-1);

    % calculate the w derivative using the formula 2.7 in
      Katsevich's
    % note on computing the derivative at a constant
      dirrection 2011.
    % essentially takes dirrect difference and linearly
      interpolates in u
    % to find the value at interlaced possitions.
    for i=1:(N-1)
        for j=1:(M-1)
            g1w(:,i,j) = (u(i)*(w(j)+delta_w)/D) * (g(2:K-1,i,
              j+1) + ...
                                g(2:K-1,i+1,j+1) - g(2:K-1,i,j) -
                                  g(2:K-1,i+1,j));
        end
    end

    g1w=g1w/(2*delta_w);

    % saving the derivatives
    if save_steps
```

```
            if ~(exist(strcat(datdirpath,'/intsteps/',datfilename)
              ,'dir'))
                mkdir(strcat(datdirpath,'/intsteps'),datfilename)
            end
            save(strcat(datdirpath,'/intsteps/',datfilename,'/g1w'
              ),'g1w');
        end
    end


    clear g;
```

A.5 altkatlength.m

```
%% length-correction weighting
%
% This section loads or calculates the length correction step.
% Length correction performed seperately on us and w
  derrivatives.
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Length␣correcting')

if exist(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g2us','.mat'),'file')
   load(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g2us'));
else
    % us length correction
    g2us = zeros(size(g1us));
    for i=1:(size(g1us,2)),  % u-loop
       for j=1:(size(g1us,3)),  % w-loop
          g2us(:,i,j) = g1us(:,i,j)*D/sqrt(u(i)^2+D^2+w(j)^2);
       end
    end


    % saving the derivatives
    if save_steps
        if ~(exist(strcat(datdirpath,'/intsteps/',datfilename)
          ,'dir'))
            mkdir(strcat(datdirpath,'/intsteps'),datfilename)
        end
        save(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g2us'),'g2us');
    end
end

clear g1us;


if exist(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g2w','.mat'),'file')
   load(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g2w'));
else

    % w length correction.
    g2w = zeros(size(g1w));
    for i=1:(size(g1w,2)),  % u-loop
       for j=1:(size(g1w,3)),  % w-loop
          g2w(:,i,j) = g1w(:,i,j)*D/sqrt(u(i)^2+D^2+(w(j)+0.5*
            delta_w)^2);
       end
    end


    % saving the derivatives
    if save_steps
        if ~(exist(strcat(datdirpath,'/intsteps/',datfilename)
          ,'dir'))
            mkdir(strcat(datdirpath,'/intsteps'),datfilename)
        end
        save(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g2w'),'g2w');
    end
end

clear g1w;
```

A.6 altkatforwardrebinn.m

```
%% forward height rebinning
%
% This section loads or calculates the desired s derivative
% at heights corresponding to the intersection of detector
% columns with the filtering lines (kappa lines).
%
% This is done by combining linear interpolation along the w
% axis of the detector for the g2us derivative with nearest
% neighbour interpolation for the g2w derivative.
%
% Henry Tregidgo, July 2013
```

```
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('forward␣height␣rebinning')

psi = zeros(L,1); % psi values of L different kappa lines
w_k = zeros(N-1,L); % matrix of w values for the lth kappa
  line at u=u(n)
delta_psi = (pi+2*alpha_m)/(L-1);

for j=1:L,
    psi(j) = -pi/2-alpha_m+(j-1)*delta_psi;
end
for i=1:(N-1),  % u-loop
   w_k(i,:)=D*h/R*(psi+psi./tan(psi)*u(i)/D);
end

if exist(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g3_alt','.mat'),'file')
   load(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g3_alt'));
else
    g3alt = zeros(K-2,N-1,L);      %variables are (s,u,psi)

    if mod(M,2) ~= 0,
        disp('error:␣M␣not␣even!');
    end

    for n=1:(N-1) % u loop
        fprintf('rebinning␣column␣number␣%d\n',n);
        for l=1:L % psi loop

            % position of the w_k value relative to the vector
               w of node
            % points
            posinwvector = (w_k(n,l)-w(1))/delta_w + 1;
            wf = floor(posinwvector);

            % index of lower point for interpolation
            j = max(1,wf);

            if j >= M
                g3alt(:,n,l) = g2us(:,n,M)+g2w(:,n,M-1);
            else
                % calculate nearest neighbour w derivative
                  index
                wr = round(posinwvector-0.5);
                k  = max(1,wr);
                t  = posinwvector - j;

                % combine nearest neighbour w derivative with
                  linear
                % interpolant of us derivative.
                g3alt(:,n,l) = g2w(:,n,k) + (1 - t)*g2us(:,n,j
                  ) + t*g2us(:,n,j+1);

            end
        end
    end


    % saving the derivatives
    if save_steps
        if ~(exist(strcat(datdirpath,'/intsteps/',datfilename)
          ,'dir'))
            mkdir(strcat(datdirpath,'/intsteps'),datfilename)
        end
        save(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g3_alt'),'g3alt');
    end
end

clear g2us
clear g2w
```

A.7 altkathilbert.m

```
%% 1D hilbert transform in u at constant psi
%
% This section loads or calculates the 1D Hilbert transform
% of the derivatives. This transform is computed along the
% kappa lines defined by the angle psi. As the derivatives
% have been rebinned this is equivalent to taking the
  transform
% allong the 3rd dimension of the array g3alt.
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('computing␣1D␣hilbert␣transforms')

if exist(strcat(datdirpath,'/intsteps/',datfilename,'/g4_alt'
  ,...
        '.mat'),'file')
```

```
    load(strcat(datdirpath,'/intsteps/',datfilename,'/g4_alt'))
       ;
else

    g4alt = zeros(size(g3alt));      %variables are (s,u,psi)

    q = N-1;
    h_ideal = zeros(2*q+1,1); %truncated ideal filter response
        (non-causal)
    for n=(-q):q,
        if mod(n,2) == 0,  % n even
            h_ideal(n+q+1) = 0;
        else    % n odd
            h_ideal(n+q+1) = 2/(pi*n);
        end
    end

    win = real(fftshift(ifft(fftshift(hanning(2*q+1)))));
    kernel = conv(win,h_ideal);
    kernel = kernel(q+1:3*q+1);
    for m=1:(size(g3alt,1)),  % s-loop
       if mod(m,10)==0,
           fprintf('%d␣convolutions␣completed\n',m);
       end
       for j=1:L,  % psi-loop
           g_filt = conv(kernel,g3alt(m,:,j));  % length 3q
           g4alt(m,:,j) = g_filt(q+1:2*q); % take middle q
             samples
       end
    end


    % saving the derivatives
    if save_steps
        if ~(exist(strcat(datdirpath,'/intsteps/',datfilename)
          ,'dir'))
            mkdir(strcat(datdirpath,'/intsteps'),datfilename)
        end
        save(strcat(datdirpath,'/intsteps/',datfilename,'/
          g4_alt'),'g4alt');
    end
end

clear g3alt
```

A.8 altkatbackrebinn.m

```
%% backward height rebinning
%
```

```
% This section loads or calculates the the values of the
% filtered data on a regular grid. This is to more easily
% facilitate the process of backprojection.
%
% As the formula relating psi to w is non-linear and inverting
% may not give the data for the psi with smallest absolute
  value,
% this rebinning step is performed by iterating through psi
% and performing linear interpolation.
%
% By checking the values of w at a given u value for
  successive
% kappa lines we avoid dirrectly calculating the needed psi
% values and can easily ensure we only use the value of psi
% with smallest absolute value.
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('backward␣height␣rebinning')
% use linear interpolation as decribed by Noo et al. rather
  than
% solving the required nonlinear equation
g5alt = zeros(K-2,N-1,M);  % variables are (s,u,w)


if exist(strcat(datdirpath,'/intsteps/',datfilename,...
        '/g5_alt','.mat'),'file')
   load(strcat(datdirpath,'/intsteps/',datfilename,...
        '/g5_alt'));
else
    for i=N/2:(N-1),  % (positive) u-loop
        if(mod(i-N/2,10)==0)
            fprintf('rebinned␣%d␣positive␣u␣columns\n',i-N/2);
        end
        % In this half of the detector plane kappa lines
          should only cross
        % towards the top of the detector so we start from the
           bottom and
        % work upwards.
        psiloopstart=1;
        for j=1:M,   % w-loop
            for l=psiloopstart:(L-1),  %psi-loop
                if (w(j)>= w_k(i,l) && w(j)<= w_k(i,l+1)),
                    c = (w(j) - w_k(i,l))/(w_k(i,l+1) - w_k(i,
                      l));
                    g5alt(:,i,j) = (1-c)*g4alt(:,i,l) + c*
                      g4alt(:,i,l+1);
                    % As less negative values of w should have
                       less
                    % negative values of psi there is no need
                      to
                    % re-check earlier values of psi.
                    psiloopstart=l;
                    break
                end
            end  % psi-loop
        end  % w-loop
    end % u-loop
    for i=1:(N/2-1),  % (negative) u-loop
        if(mod(i,10)==0)
            fprintf('rebinned␣%d␣negative␣u␣columns\n',i);
        end
        % In this half of the detector plane kappa lines
          should only cross
        % towards the bottom of the detector so we start from
          the top and
        % work downwards.
       psiloopstart=L;
      for j=M:-1:1,   % w-loop
         for l=psiloopstart:-1:2, %psi-loop
            if (w(j)>= w_k(i,l-1) && w(j)<= w_k(i,l)),
               c = (w(j) - w_k(i,l-1))/(w_k(i,l) - w_k(i,l-1))
                 ;
               g5alt(:,i,j) = (1-c)*g4alt(:,i,l-1) + c*g4alt
                 (:,i,l);
               psiloopstart=l;
               break;
            end
         end % psi-loop
      end  % w-loop
    end  % u-loop

    % saving the derivatives
    if save_steps
        if ~(exist(strcat(datdirpath,'/intsteps/',datfilename)
          ,'dir'))
            mkdir(strcat(datdirpath,'/intsteps'),datfilename)
        end
        save(strcat(datdirpath,'/intsteps/',datfilename,...
            '/g5_alt'),'g5alt');
    end
end

clear g4alt
```

A.9 katpiline.m

```
%% find PI-line intervals for each x in reconstruction region
%
% looks for the pi line end points for lines intersecting the
% individual pixels. If circle = true it only looks for those
% intersecting the field of view.
%
% If there already exists a pi file for the current slice and
% geometry it will load the file instead of creating a new one
%
% This script requires the functions findPI.m and PIfun.m
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tic
PI = zeros(MX,MY,2);  % variables: (x,y,[s_b s_t])

% the location of the slice in terms of voxel layers
planenumber=round((Z-min_obj_z_value+(hz/2))/hz);

% identification number for filing
planeident=num2str(planenumber);

if ~exist(PIfilepath,'dir')
    mkdir(PIfilepath)
end

if exist(strcat(PIfilepath,'/',PIfilename,planeident,'.mat'),'
  file')
   load(strcat(PIfilepath,'/',PIfilename,planeident));
else
   disp('finding␣PI-line␣intervals␣with␣the␣Champley␣method')
   for i=1:MX,
       if (mod(i,10)==0)
            fprintf('%d␣rows␣completed\n',i);
       end
      parfor j=1:MY,
        if circle,
             if (x(i)^2 + y(j)^2) > radius^2,
                 continue;
             end
        end
          PI(i,j,:) = findPI(P,R,delta_s,[x(i) y(j) Z]);
      end
   end
```

```
    save(strcat(PIfilepath,'/',PIfilename,planeident),'PI');
end

toc
```

A.10 altkatbackproject.m

```
%% backprojection step
% use trap. rule to implement eqtn(73) in Noo et. al.
%
% This section backprojects the data in g5 to build a
% reconstructed slice.
%
% As suggested in Noo et. al. smoothing is applied to the
% edges of the Tam Danielson window to avoid production of
% artefacts. However, this smoothing is done in the s
  direction
% which requires the use of pi line intervals for the
% reconstruction.
%
% Smoothing can be applied to the window in other ways but
% has been found to produce worse results due to dependence
% on regularisation variables.
%
% Henry Tregidgo, July 2013
%
% Based on code by Adam Wunderlich
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic

if ~exist('F','var')
    F = zeros(MX,MY,length(z_slices));  % recontruction array
      (x,y,z)
end

F(:,:,planenumber)=zeros(MX,MY);

u_length=size(g5alt,2);
w_length=size(g5alt,3);

disp(strcat('backprojecting␣slice␣',num2str(planenumber)))
disp('completed␣reconstructing␣x␣column:')
for i=1:MX,   % x-loop
    if mod(i,50)==0,
        disp(i)
    end
    for j=1:MY,   % y-loop
        if circle,
           if (x(i)^2 + y(j)^2) > radius^2,
              continue;
           end
        end
        sb = PI(i,j,1);  % bottom of PI segment
        st = PI(i,j,2);  % top of PI segment
        % find indices of s corresponding to sb and st
        k_sb = (sb-s_range(1))/delta_s+1;
        k_st = (st-s_range(1))/delta_s+1;
        % backproject
        for k=(floor(k_sb)-2):(ceil(k_st)+2),  % integrate over
           PI-line
           vstar = R - x(i)*cos(s(k)) - y(j)*sin(s(k));
           ustar = D*(-x(i)*sin(s(k))+y(j)*cos(s(k)))/vstar;
           wstar = D*(Z-h*s(k))/vstar;

           % find nearest neightbor
           usn = round((ustar-u(1))/delta_u+1);
           usn = max(1,usn);
           usn = min(u_length,usn);
           wsn = round((wstar-w(1))/delta_w+1);
           wsn = max(1,wsn);
           wsn = min(w_length,wsn);
           gi_near = g5alt(k,usn,wsn);
           % determine rho
           % weight the endpoints of the PI-line in a smooth
             fashion
           d_in = (s(k)-sb)/delta_s;
           d_out = (st - s(k))/delta_s;
           if (s(k) <= (sb - delta_s))
              rho = 0;
           elseif ((sb - delta_s) < s(k)) && (s(k) <= sb)
              rho = .5*(1+d_in)^2;
           elseif (sb < s(k)) && (s(k) <= (sb+delta_s))
              rho = .5 + d_in - .5*d_in^2;
           elseif (sb + delta_s < s(k)) && (s(k) <= (st -
             delta_s))
              rho = 1;
           elseif ((st - delta_s) < s(k)) && (s(k) <= st)
              rho = .5 + d_out - .5*d_out^2;
           elseif (st < s(k)) && (s(k) <= (st + delta_s))
              rho = .5*(1+d_out)^2;
           elseif (s(k) > (st + delta_s))
              rho = 0;
           end
           deltaF = rho*delta_s*gi_near/vstar;
           F(i,j,planenumber) = F(i,j,planenumber)+deltaF;
        end  % end k-loop
    end  % end y-loop
end   % end x-loop
```

```
F(:,:,planenumber) = F(:,:,planenumber)./(2*pi);

toc
disp('Done!')
```

Appendix B

Support Files

B.1 calcPitch.m

```
function [ P ] = calcPitch(height,R,D,N)
%CALCPITCH Calculate Pitch for helix using voxel height & TD
  window on a square detector
%
%   height  - height dimesion of square detector panel
%   R       - Radius of Helix
%   D       - Source to detector distance
%   N       - Number of detector rows/columns
%
%   uses formula from Noo et al.
%
%   P = (height * R * D * pi) / ((u^2+D^2)*(pi/2 + alpha))
%
%   where u is the minimum horizontal coordinate and alpha is
  the half fan
%   angle.
%
%   Henry Tregidgo (27 Jun 2013)

% calculate detector element dimensions
step = height/N;

% find voxel dimensions assuming cubic and that detector
  element dimensions
% are a simple magnification.
vox = step*R/D;

% we want there to be extra rows outside the TD window to
  allow for taking
% derivatives and interpolating to kappa lines.
desiredTDHeight = height - 3*step;

% calculate the minimum horizontal coordinate and perform part
   of the
% calculation.
```

```
part1 = desiredTDHeight/((height/2)^2 + D^2);

% calculate half fan angle and max psi angle for kappa lines.
maxPsi = (pi/2)+atan(height/(2*D));

% calculate the maximum Pitch given the constraints.
P = part1*R*D*pi/maxPsi;

% reduce P to be a multiple of voxel height to reduce
  complexity of Pi-Line
% calculations.
temp = P/vox;
temp1 = floor(temp);

P = temp1*vox;

end
```

B.2 calcFilterLineNo.m

```
function [ L ] = calcFilterLineNo( height,R,D,M,P )
%CALCFILTERLINENO calculate number of filter lines needed
%
% calculates the minimum even number of filter lines needed to
   make sure
% that the vertical distance between lines is never greater
  than the
% detector pixel height.
%
%   height  - height dimesion of detector panel
%   R       - Radius of Helix
%   D       - Source to detector distance
%   N       - Number of detector rows
%
%   Henry Tregidgo July 2013

step = height/M;

alpha = atan(height/(2*D));

psi = ((pi/2) + alpha);

a = D*P/(2*pi*R);
b = -(height)/(2*D);

temp  = tan(psi)-psi*sec(psi)*sec(psi);
temp1 = temp/(tan(psi)*tan(psi));

wdiff = a*(1+b*temp1);
```

```
L=2*psi*wdiff/step;

L=ceil(L);

if mod(L,2)
    L=L+1;
else
    L=L+2;
end

end
```

B.3 dependencesurf.m

```
function [ wtop, depsurf ] = dependencesurf( R,P,radfov )
%DEPENDENCESURF return minimum surface for given minimum angle
% code written to draw the surface describing the ends of a
% reconstructable cylinder within a given helical geometry
%
%       R       - radius of the helical path
%       P       - pitch of the helical path
%       x       - vector of node positions in grid within the
%                 mask of the field of view.
%       radfov  - radius of field of view.
%       D       - source to row distance.
%       wtop    - height of piline intersection
%

x=-radfov:radfov/512:radfov;
wtop = zeros(1025);
for i = 1:1025

    D = R + x(i);
    r=x.^2+D^2;
    wtop(i,:)=(P/(2*pi*R*D))*r.*((pi/2)-atan(x/D));

end


depsurf =zeros(size(wtop));

for i =1:1025
for j =1:1025
if ((i-513)^2 + (j-513)^2 < 512^2)
depsurf(i,j)=wtop(i,j);
else
depsurf(i,j)=NaN;
end
```

```
end
end

end
```

Appendix C

Reconstruction Parameters

Table C.1: Parameters for Reduced ball reconstructions. All lengths in mm.

Parameter	Reduced Ball Value
Source to detector distance	80
Source to axis of rotation	50
Pitch per full turn	40.4297
Detector height	40
Source positions	576
Helical step (radians)	0.0123
Detector rows	512
Detector columns	512
Filter lines	742

Table C.2: Parameters for Planes of Spheres reconstructions. All lengths in mm.

Parameter	Planes of Spheres Value
Source to detector distance	77.5
Source to axis of rotation	7.5
Pitch per full turn	3.3649
Detector height	27.436
Source positions	721
Helical step (radians)	0.0087
Detector rows	202
Detector columns	512
Filter lines	194

Table C.3: Parameters for Platinum Dust reconstructions. All lengths in mm.

Parameter	Platinum Dust Values
Source to detector distance	77.5
Source to axis of rotation	7.5
Pitch per full turn	3.3649
Detector height	27.4091
Source possitions	721
Helical step (radians)	0.0087
Detector rows	202
Detector columns	512
Filter lines	194

Table C.4: Parameters for Sapphire Balls reconstructions. All lengths in mm.

Parameter	Sapphire Balls Values
Source to detector distance	112.6
Source to axis of rotation	80
Pitch per full turn	45.7085
Detector height	43.6918
Source possitions	781
Helical step (radians)	0.0058
Detector rows	322
Detector columns	512
Filter lines	372

Bibliography

[1] P. E. Danielsson, P. Edholm, J. Eriksson, and M. Magnusson Seger. Towards exact reconstruction for helical cone-beam scanning of long objects. a new detector arrangement and a new completeness condition. In D. Townsend and P. Kinahan, editors, *Meeting on Fully 3D Image Reconstruction in Radiology and Nuclear Medicine*, pages 141–4, Pittsburgh, PA, 1997.

[2] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics.* Oxford University Press, 2005.

[3] L. A. Feldkamp, L. C. Davis, and J. W. Kress. Practical cone-beam algorithm. *J. Opt. Soc. Am. A*, 1(6):612–619, Jun 1984. doi: 10.1364/JOSAA.1.000612. URL `http://josaa.osa.org/abstract.cfm?URI=josaa-1-6-612`.

[4] P. Grangeat. Mathematical Framework of Cone Beam 3D Reconstruction via the 1st Derivative of the Radon-Transform. *Lecture Notes In Mathematics*, 1497: 66–97, 1991. ISSN 0075-8434.

[5] F. John. The ultrahyperbolic differential equation with four independent variables. *Duke Math. J.*, 4(2):300–322, 1938.

[6] W. Kalender. *Computed tomography: fundamentals, system technology, image quality, applications*, volume 1. Wiley-VCH, 2000.

[7] A. Katsevich. Theoretically exact filtered backprojection-type inversion algorithm for spiral ct. *SIAM Journal on Applied Mathematics*, 62(6):2012–2026, 2002.

[8] A. Katsevich. An improved exact filtered backprojection algorithm for spiral computed tomography. *Advances in Applied Mathematics*, 32(4):681 – 697,

2004. ISSN 0196-8858. doi: 10.1016/S0196-8858(03)00099-X. URL `http://www.sciencedirect.com/science/article/pii/S019688580300099X`.

[9] A. Katsevich. A note on computing the derivative at a constant direction. *Physics in Medicine and Biology*, 56(4):N53, 2011. URL `http://stacks.iop.org/0031-9155/56/i=4/a=N01`.

[10] A. A. Kirillov. On a problem of im gelfrand. *Soviet Math.*, 2:268–269, 1961.

[11] F. Natterer. A sobolev space analysis of picture reconstruction. *SIAM Journal on Applied Mathematics*, 39(3):402–411, 1980.

[12] F. Noo, J. Pack, and D. Heuscher. Exact helical reconstruction using native cone-beam geometries. *Physics in Medicine and Biology*, 48(23):3787, 2003. URL `http://stacks.iop.org/0031-9155/48/i=23/a=001`.

[13] J. Nuyts, B. De Man, J. A. Fessler, W. Zbijewski, and F. J. Beekman. Modelling the physics in the iterative reconstruction for transmission computed tomography. *Physics in Medicine and Biology*, 58, 2013.

[14] Y. Shi, X. Li, B. Yin, and D. Kong. Image interpolation using reproducing kernel filter and texture orientation. In *2006 8TH INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING, VOLS 1-4*, pages 1310–1313, 2006. ISBN 978-0-7803-9736-1.

[15] K. C. Tam, S. Samarasekera, and F. Sauer. Exact cone beam ct with a spiral scan. *Physics in Medicine and Biology*, 43(4):1015, 1998. URL `http://stacks.iop.org/0031-9155/43/i=4/a=028`.

[16] H. Tuy. An inversion formula for cone-beam reconstruction. *SIAM Journal on Applied Mathematics*, 43(3):546–552, 1983. doi: 10.1137/0143035. URL `http://epubs.siam.org/doi/abs/10.1137/0143035`.

[17] T. Varslot, A. Kingston, G. Myers, and A. Sheppard. High-resolution helical cone-beam micro-ct with theoretically-exact reconstruction from experimental data. *Medical Physics*, 38:5459, 2011. URL `http://online.medphys.org/resource/1/mphya6/v38/i10/p5459_s1`.

[18] A. J. Wunderlich. *The Katsevich Inversion Formula for Cone-Beam Computed Tomography*. MSc. dissertation, Oregon State University, 2006.