5-2006

Iterative Reconstruction of Cone-Beam Micro-CT Data

Thomas Matthew Benson
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Thomas Matthew Benson entitled "Iterative Reconstruction of Cone-Beam Micro-CT Data." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Jens Gregor, Major Professor

We have read this dissertation and recommend its acceptance:

Michael Berry, Charles Collins, Michael Thomason, Jonathan Wall

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Thomas Matthew Benson entitled "Iterative Reconstruction of Cone-Beam Micro-CT Data". I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Jens Gregor
Major Professor

We have read this dissertation
and recommend its acceptance:

Michael Berry

Charles Collins

Michael Thomason

Jonathan Wall

Accepted for the Council:

Anne Mayhew
Vice Chancellor and Dean of
Graduate Studies

(Original signatures are on file with official student records.)

Iterative Reconstruction of Cone-Beam Micro-CT Data

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Thomas Matthew Benson
May, 2006

Acknowledgments

I would like to thank my advisor, Dr. Jens Gregor, for devoting so much time to advising me during the past four years. Without our many engaging consultations, this work would likely not have been completed and certainly would not have been nearly as enjoyable. I would also like to thank my family for their continued support during my undergraduate and graduate studies. Finally, I would like to thank all of the professors that have shared their knowledge with me throughout the years.

This work was supported by the National Institutes of Health under grant number 1 R01 EB00789. The computer equipment was acquired as part of SInRG, a University of Tennessee grid infrastructure grant supported by the National Science Foundation under grant number EIA-9972889.

Abstract

The use of x-ray computed tomography (CT) scanners has become widespread in both clinical and preclinical contexts. CT scanners can be used to noninvasively test for anatomical anomalies as well as to diagnose and monitor disease progression. However, the data acquired by a CT scanner must be reconstructed prior to use and interpretation. A reconstruction algorithm processes the data and outputs a three dimensional image representing the x-ray attenuation properties of the scanned object. The algorithms in most widespread use today are based on filtered backprojection (FBP) methods. These algorithms are relatively fast and work well on high quality data, but cannot easily handle data with missing projections or considerable amounts of noise. On the other hand, iterative reconstruction algorithms may offer benefits in such cases, but the computational burden associated with iterative reconstructions is prohibitive. In this work, we address this computational burden and present methods that make iterative reconstruction of high-resolution CT data possible in a reasonable amount of time. Our proposed techniques include parallelization, ordered subsets, reconstruction region restriction, and a modified version of the SIRT algorithm that reduces the overall run-time. When combining all of these techniques, we can reconstruct a $512 \times 512 \times 1022$ image from acquired micro-CT data in less than thirty minutes.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The primary goal of this dissertation is the implementation and evaluation of iterative reconstruction techniques for x-ray tomography data. In particular, we focus on reconstructing data acquired from high-resolution small animal medical scanners (micro-CT), although many of the developments apply to other CT applications as well. Currently, the most widely employed algorithms for CT reconstruction are the filtered backprojection (FBP) methods, for example, the FDK algorithm introduced by Feldkamp, Davis, and Kress in 1984 [1]. The FBP methods are based on the Fourier Slice Theorem and can be implemented using the Fast Fourier Transform (FFT). On the other hand, iterative algorithms such as expectation maximization (EM) [2,3] have been used extensively for reconstruction of emission data, such as positron emission tomography (PET) and single photon emission tomography (SPECT). The iterative reconstruction problem associated with emission data is typically much smaller in scale than that associated with micro-CT data. As a result of the higher computational burden, iterative reconstruction of CT data has not been widely employed. Thus, the major focus of this dissertation is to address the computational burden associated with iterative reconstructions.

There are indications that iterative algorithms could offer superior results in some circumstances, such as metal artifact reduction [4] and limited or sparse angle reconstruction [5,6]. In this work, we aim to develop techniques to reduce the amount of computation required and address the remaining requirements using a distributed reconstruction framework. While the focus of the work is on the reconstruction phase, we must also partially understand the functioning of the x-ray scanner as well as the nature of the acquired projection data that will form the input of the reconstruction algorithms.

1.1 History

The medical imaging field dates to December 1895 when Wilhelm Roentgen published a radiograph of his wife's hand. The use of planar radiographic imaging for medical purposes expanded and was ultimately extended to transmission computed tomography (CT) by Godfrey Hounsfield with his development of a CT scanner in 1972 at EMI in England [7]. Transmission CT consists of taking radiographic-like images of the object of interest from multiple angles and reconstructing images of the object rather than viewing the radiographs directly. Hounsfield shared the Nobel Prize in Medicine in 1979 with Allan Cormack, who

had developed theoretical foundations for CT a decade before Hounsfield constructed his CT scanner. In the past several decades, there has been significant progress in the x-ray CT field in terms of both hardware and reconstruction techniques. Furthermore, the medical imaging field includes other modalities, such as nuclear medicine, including positron emission tomography (PET) and single photon emission computed tomography (SPECT), ultrasound imaging, and magnetic resonance imaging (MRI). In this dissertation, we focus on iterative reconstruction algorithms for transmission CT data.

1.2 X-ray Tomography

X-ray scanners consist of at least one x-ray source and at least one detector in varying configurations. X-rays are emitted from the source(s), attenuated as they traverse the scanned object, and finally recorded by the detector(s). These detector recordings form the projection data set that will be the input to the iterative reconstruction algorithms. In addition to the projection data, we must also know the scanner geometry in order to perform the reconstruction.

1.2.1 Projection Data

In order to discuss the nature of the projection data, we must first consider the process of x-ray attenuation. In this section, we follow the presentation of x-ray attenuation given by Prince and Links [8]. X-ray attenuation is the loss of strength of an x-ray beam when passing through some object, due primarily to the photoelectric effect and compton scattering. By recording the intensity of an x-ray beam after traversing an object and comparing that intensity to the initial x-ray intensity, we can determine how much x-ray attenuation occurred along the path of the x-ray beam. While attenuation is a statistical process, we assume for this section that the process is deterministic, as is done in [8]. We define a projection ray corresponding to an x-ray source position and detector element as the path taken by the x-rays emitted from the x-ray source and recorded at the detector element.

Assume that we have an x-ray beam consisting of N monoenergetic photons traversing an object with constant attenuation coefficient μ and width Δx. If we assume that the number of photons lost due to attenuation, ΔN, is proportional to μ, N, and Δx, then we have

$$\Delta N = -\mu N \Delta x \tag{1.1}$$

or, written in another way,

$$\frac{\Delta N}{N} = -\mu \Delta x. \tag{1.2}$$

Consider the case of the object width, and thus the attenuation, approaching zero. Then, treating N as a continuous value, we have

$$\frac{dN}{N} = -\mu \, dx \tag{1.3}$$

with solution

$$N = N_0 \exp\left(-\mu \Delta x\right) \tag{1.4}$$

where N_0 is the number of photons emitted from the x-ray source at position $x = 0$. If we consider a three-dimensional object with the attenuation coefficient dependent on the coordinates x, y, and z, then we have

$$N = N_0 \exp\left\{-\int_{ray} \mu(x, y, z)\, ds\right\} \tag{1.5}$$

where *ray* describes the path, parameterized by s, taken through the object.

We can reinterpret this expression in terms of the original x-ray intensity, I_0, and the detected x-ray intensity, I, as

$$I = I_0 \exp\left\{-\int_{ray} \mu(x, y, z)\, ds\right\} \tag{1.6}$$

or, equivalently,

$$\int_{ray} \mu(x, y, z) ds = \ln \frac{I_0}{I}. \tag{1.7}$$

Since I_0 is not accurately known in general, a blank scan, or a scan with no object present, is performed prior to reconstruction and the intensities measured per detector element are assumed to be the original intensities corresponding to that projection ray. Furthermore, there may be some variability in the accuracy of the detector readings. In particular, the detectors may have a nonzero baseline and may thus produce values that are slightly different from what they should be. To help correct for such readout errors, a dark scan is taken, which consists of performing a scan with the x-ray source deactivated or blocked such that no x-rays reach the detector thus yielding baseline readings for the detector elements. We then correct for these errors by subtracting the dark scan reading, I_d, from both the blank scan and normal scan values. In other words, (1.7) becomes

$$\int_{ray} \mu(x, y, z) ds = \ln \frac{I_0 - I_d}{I - I_d}. \tag{1.8}$$

Thus, we can calculate the line integral approximations for each projection ray by correcting and log-normalizing the projection data according to (1.8). There are other possible corrections, such as identifying and handling a bad detector element, that we do not consider in this work. This projection data, either log-normalized or in its original form, characterizes the input to the reconstruction algorithms.

1.2.2 Interpretation as a Linear System

We will interpret a discretization of the line integral representation of the projection data as given in (1.8) as a linear system. In particular, in the linear system $Ax = b$, b is the (potentially corrected and log-normalized) projection data, x is a discrete interpretation of μ and represents the reconstructed image, and the system matrix A is a linear transformation from image space to projection space and thus mimics the line integral operator. By solving this linear system for x, we will obtain a discrete attenuation map of the scanned object. In the case of a mouse, this attenuation map reveals information about the anatomical

structure of the mouse because different anatomical components (bone, water, soft tissue) have different attenuation coefficients.

1.2.3 Scanner Configurations

The most common x-ray scanner beam configurations are parallel-beam, fan-beam, and cone-beam. A parallel-beam scanner acquires data along parallel lines for each projection; see Fig. 1.1. For this geometry, there must either be multiple x-ray sources or some mechanism for translating the source from one position to the next during the acquisition process. A fan-beam scanner, on the other hand, consists of a single x-ray source and multiple detectors (or a single detector with multiple detector elements) and the x-ray source emits x-rays in a "fan" toward the detectors; see Fig. 1.2. By extending from a one-dimensional detector to a two-dimensional detector, the "fan" of x-rays in the fan-beam geometry extends to a cone, yielding the cone-beam geometry. The cone-beam geometry, which is the geometry we consider for the remainder of this dissertation, is shown in Fig. 1.3.

In addition to the x-ray beam configuration, an x-ray scanner requires some mechanism for obtaining projections of the scanned object from multiple view angles in order to provide sufficient data for a reconstruction. In order to accomplish this, either the x-ray source and detector can rotate about the object of interest, or the object itself can be rotated while the x-ray source and detector remain fixed. Additionally, the x-ray source can rotate in several trajectories around the object. The two primary trajectories used are circular orbit and helical orbit. With a circular orbit trajectory the x-ray source and detector rotate in a circle around the object of interest while the object of interest remains fixed. With a helical trajectory, either the x-ray source can rotate in a helical fashion about the object, or the object can be translated (for example, on a moving gantry) while the x-ray source rotates in a circular orbit. In the latter case, the x-ray source rotates in a circular trajectory, but from the perspective of the scanned object, the x-ray source is rotating in a helical trajectory. Most of the data used for this dissertation was acquired from a MicroCAT II (Siemens/CTI Concorde Microsystems), which is a circular orbit cone-beam micro-CT scanner.

1.3 System Models

As shown in Section 1.2.2, the projection data can be modeled as a linear system given by $Ax = b$. However, we must choose some method by which we define the system matrix elements where element a_{ij} quantifies the contribution of voxel j to the attenuation of projection ray i. The model chosen to specify the system matrix elements is known as the system model. In other words, the system matrix is the embodiment of the system model in the form of a matrix. The system model is critical to both the accuracy and the efficiency of the iterative reconstruction procedure. The system model impacts the accuracy of the reconstruction because ultimately the reconstruction generates an approximate solution to the system $Ax = b$ and the solution will only be of high quality if the system matrix itself is accurate. Furthermore, calculating the system matrix elements may consume a substantial percentage of the reconstruction run-time, so the system model also affects run-time performance. Choosing a system model thus involves a trade-off between accuracy

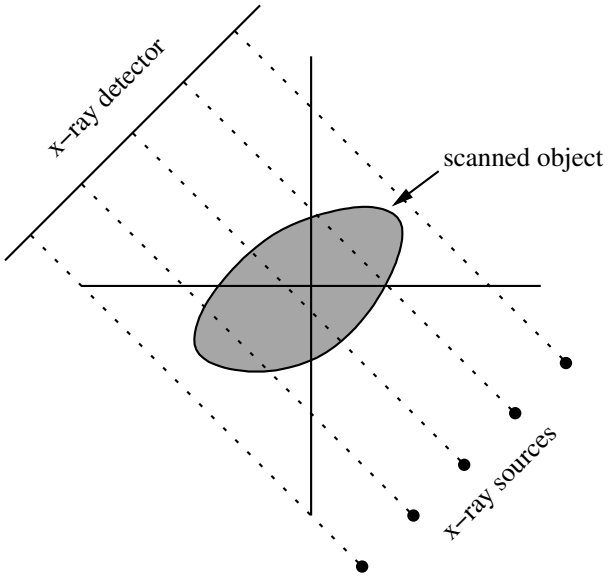

Figure 1.1: Parallel-beam scanner geometry.

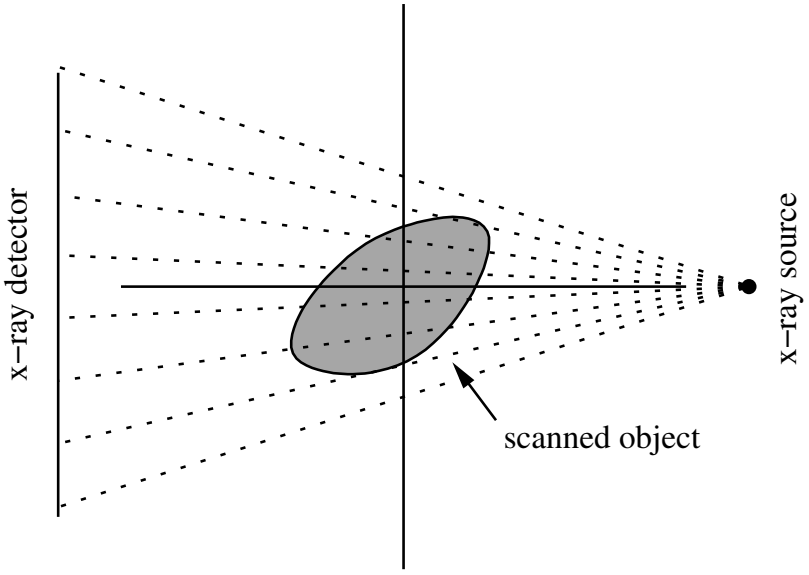

Figure 1.2: Fan-beam scanner geometry.

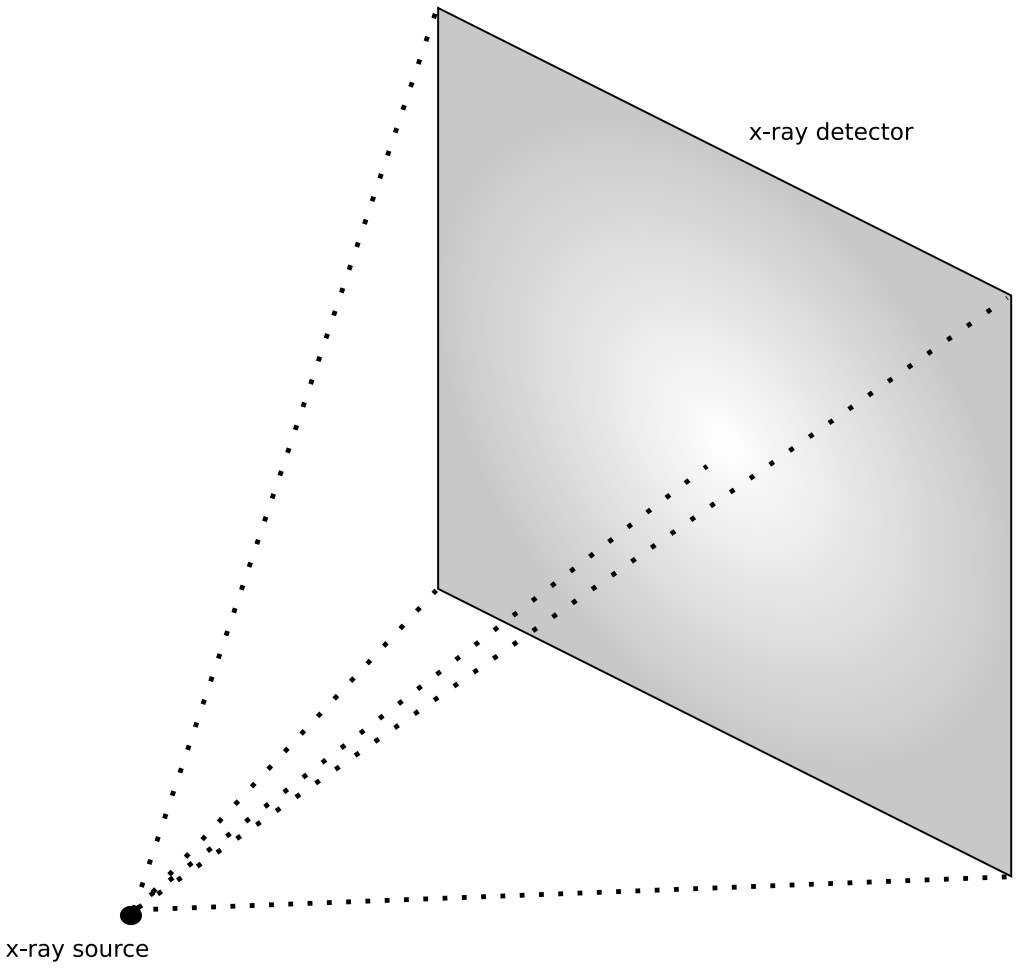

Figure 1.3: Cone-beam scanner geometry.

and efficiency: the model must be accurate enough to yield acceptable reconstructions, but should be computationally simple enough to facilitate an efficient implementation.

We consider several system models in these terms in Chapter 2. In particular, we introduce system models based on line intersections, trilinear interpolation, and volumetric intersection. The line intersection based system model, which we introduce in Section 2.1, computes the length of intersection between a projection ray and each voxel. This model is simple to implement and is very fast, but we will demonstrate that it generates ring artifacts in the reconstructions, although those artifacts may be acceptable in certain applications. On the other hand, the trilinear interpolation model represents the projection ray as a series of equidistant points and assigns the interpolation coefficients corresponding to the eight voxel centers nearest a given point to the associated voxel. We introduce the trilinear interpolation based model in Section 2.2. This model is also fairly simple to implement and requires more computation than the line intersection model, but eliminates the ring artifacts in the reconstruction. Finally, the volumetric approaches introduced in Section 2.3 model the projection ray as a polytope and compute, or approximate, the volume of the intersection between a projection ray polytope and each voxel. These models have a more complex implementation, at least as we have currently implemented them, and are considerably slower than the other models, but they also eliminate the ring artifact prevalent in the line intersection model based reconstructions. We present a more thorough comparison of these models in Section 2.4 and ultimately identify trilinear interpolation as representing a reasonable blend of accuracy and computational efficiency.

1.4 Iterative Reconstruction Algorithms

Once we have chosen and implemented a system model, and thus have a concrete linear system given by $Ax = b$, then the image reconstruction problem is to solve the system for x. In Chapter 3, we discuss iterative methods of solving the linear system. In the medical imaging literature, the most popular iterative reconstruction algorithms can be classified as either algebraic or statistical. The former class of algorithms involves applying algebraic techniques to the linear system while the latter class interprets the process of attenuation in a statistical context and solves the problem in that context, typically by applying gradient ascent methods to a Taylor expansion of a likelihood function.

In this work, we focus on the algebraic reconstruction methods. In particular, we introduce the algebraic reconstruction technique (ART), simultaneous iterative reconstruction technique (SIRT), and simultaneous algebraic reconstruction technique (SART) in Sections 3.2, 3.3, and 3.4, respectively. We relate these algorithms to classical iterative techniques for linear systems, such as the Gauss-Seidel and Jacobi iterations. Additionally, in Section 3.5 we present a parallel SIRT algorithm (PSIRT) that reduces the reconstruction run-time in a distributed computing environment. While we focus on reconstruction of micro-CT data in this dissertation, these iterative reconstruction techniques apply to linear systems in general, including those resulting from reconstruction data corresponding to other imaging modalities.

1.5 Implementation

Due to the computation and memory burdens imposed by iterative reconstruction of large CT data sets, the implementation of the iterative reconstruction framework is critical. In particular, the implementation must control memory consumption such that the reconstruction can run in the memory available on given hardware while minimizing the run-time. In order to reduce the run-time, we parallelize the reconstruction framework using threads to distribute the work locally to available processors on a single node and message passing (MPI) to distribute the work to multiple nodes. This approach reduces the run-time of the iterative reconstruction by distributing the work, but it does not reduce the amount of work that must be performed to complete an iterative reconstruction. Furthermore, distributing the workload to several nodes introduces the requirement of frequent interprocessor communications, performed using MPI in our case, which negatively impacts the final run-time. We present these parallel computing issues in Section 4.1. In Section 4.2 we discuss ordered subsets, a technique that effectively reduces the total amount of work that must be performed by reducing the number of total iterations required for an acceptable reconstruction. While ordered subsets reduces the amount of computational work required, it also increases the relative cost of the interprocessor communications.

The combination of parallel computing and ordered subsets addresses the computational burden of an iterative reconstruction, but does little to reduce the memory requirements. Thus, in order to decrease the memory consumption, as well as the amount of data communicated, we present compressing the image data when only reconstructing a region of interest in Section 4.3. We then address two methods of determining such regions of interest in Sections 4.4 and 4.5. The first such method, discussed in Section 4.4, restricts the reconstruction to a geometry defined support region. Then, in Section 4.5, we present a heuristic, data-driven technique that utilizes the projection data to identify voxels likely to contain an object with nonnegligible attenuation characteristics. The reconstruction region of interest is then restricted to these identified voxels. In addition to reducing the memory requirements and amount of data communicated, these region restriction approaches also significantly reduce the amount of computation because we do not need to compute system matrix elements corresponding to the voxels outside of the region of interest. Finally, we further decrease the amount of data communicated for a reconstruction using the parallel SIRT algorithm presented in Section 3.5.

1.6 Results

In Chapter 5 we discuss the reconstruction results obtained using all of the concepts introduced in this dissertation as well as their associated timings. We demonstrate that the techniques addressed in Chapter 4 very effectively decrease the overall run-time. In particular, in Section 5.1 we present the benefits of parallelizing the reconstruction framework. We then quantify the benefits of implementing ordered subsets in Section 5.2 and present the advantages of restricting the reconstruction to a particular region in Section 5.3. In Section 5.4 we consider the impact of implementing the parallel SIRT algorithm. In light of all of these results, we revisit the issue of parallel computing in Section 5.5 in terms

of varying the number of reconstruction nodes. Finally, we present reconstructions using several of the discussed techniques in Section 5.6.

1.7 Conclusions and Future Work

In Chapter 6 we present our concluding remarks, which are based primarily on the results obtained in Chapter 5, and discuss directions for future research. In particular, we observe that the techniques discussed throughout this dissertation very effectively reduce the overall run-time of an iterative reconstruction.

Chapter 2

System Models

As stated in the introduction, the main goal of the iterative image reconstruction algorithms presented in this work is to approximately solve the linear system

$$Ax = b \tag{2.1}$$

where the vector b corresponds to the projection data acquired from the x-ray scanner. However, in order to do so, we must choose some method of determining the elements of A. We refer to A as the system matrix, its elements as the system matrix elements, and the mathematical formulation defining those elements as the system model. In this chapter, we introduce several system models, along with their advantages and disadvantages. The ultimate goal in choosing a system model is to select a model that is accurate enough to yield reconstructions of acceptable quality, but that also requires as little computation as possible.

Most generally, the system matrix element a_{ij} quantifies the relative contribution of the jth voxel to the attenuation of the ith projection ray. Thus, each projection ray, which in turn corresponds to a single detector element value, yields one row in the system matrix. Similarly, each image voxel corresponds to a column of the system matrix. We will always use the term projection ray to describe the radiological path associated with an x-ray source position and detector element pair, although in some of the models this radiological path is volumetric rather than ray based.

There are two typical classes of system models to choose from: voxel-driven and ray-driven. Voxel-driven approaches, which in the two-dimensional case are termed pixel-driven approaches, iterate over the voxels and calculate the system matrix column corresponding to each voxel. Typically, these calculations depend on some interpretation of the footprint of a voxel on the detector (see, e.g., [9]). On the other hand, ray-driven approaches iterate over the projection rays and calculate the system matrix row corresponding to that projection ray. We will introduce several ray-driven approaches in this chapter. Additionally, there is an approach termed distance-driven introduced by De Man and Basu [10, 11] that does not iterate over voxels or detector elements, but instead iterates over calculated intercepts. We do not consider voxel-driven or distance-driven methods in this work.

We use several methods to compare the system models introduced in this chapter. The first method of comparison we employ is an analysis of the voxel support generated by each system model for the scanner geometry corresponding to the tiny hole phantom

data set in Appendix A.2. Algebraically, we define the support for voxel j as the sum of the jth column of the system matrix. Thus, the voxel support values are equivalent to backprojecting a projection data set of constant ones. The intuitive interpretation of voxel support for a geometry based system model is the amount of data support for a given voxel that is available in the projection data. Voxels outside of the detected paths of the x-ray cone-beam will have no support, so there will be a sharp change in support values at these boundaries. However, it seems reasonable to expect that any two neighboring voxels that are both within a detected region of the x-ray cone-beam would have similar support values. Large variations in the voxel support values in those regions supported by the scanner geometry would generate reconstruction artifacts during the backprojection operation. Thus, we will examine the voxel support generated by the various system models to determine if such reconstruction issues are likely to arise.

In addition to the voxel support analysis, in Section 2.4 we also compare reconstructions of the tiny hole phantom data set described in Appendix A.2 using the presented system models. We compare the reconstructions for visual quality and also compare the time required to perform the system model calculations using our current implementation. Throughout this chapter, when we refer to the computational efficiency of a system model, we are only referencing the efficiency of our current implementation as opposed to a more general complexity metric.

2.1 Line Intersection Model

One of the simplest and fastest system models is based upon line intersections. Define projection ray i as the line segment joining the x-ray source, which we assume to be a point source, and some point in the corresponding detector element, typically the center. Then the line intersection system model stipulates that element a_{ij} of the system matrix is the intersection length between projection ray i and image voxel j. Thus, the row sum defined by $\sum_{j=1}^{N} a_{ij}$ is the length of the intersection of projection ray i with the voxel space. A two-dimensional example is shown in Fig. 2.1.

2.1.1 Implementation

As with most system models, the efficiency of the line intersection method depends greatly upon the implementation. A brute force approach would iterate over all of the voxels for each projection ray and calculate the intersection length of the projection ray with each voxel. While this approach is simple, it would be exceedingly slow since a single projection ray does not intersect the vast majority of voxels in a three-dimensional image. Siddon introduced a fast method of calculating this "exact radiological path" [12] and Jacobs et al. published a modification that further accelerated the algorithm [13]. This approach is based on representing the projection ray parametrically and iteratively updating the parameters to identify each intersected voxel and calculate the intersection length. Thus, the intersected voxels are determined in a straightforward manner and no computational time is expended on nonintersected voxels.

By using the modified Siddon's algorithm, we can quickly generate system matrix elements for a given projection ray. However, in our current implementation, we compute

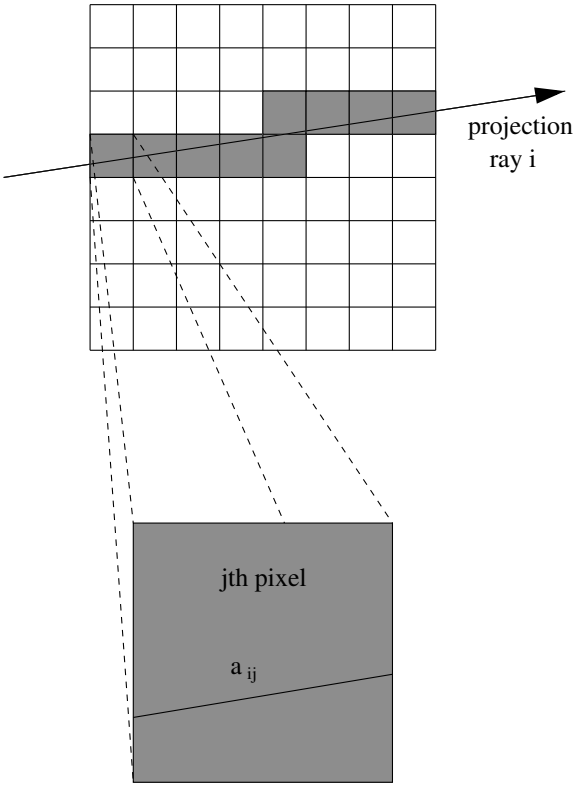

Figure 2.1: An example of the line intersection model for a single projection ray in the two-dimensional case.

a full row and then use that row in the algorithm, so we must temporarily store it. The most obvious data structure to employ is simply an image sized array that represents a single row in the system matrix. However, since the vast majority of the entries in each row are zero, this approach is not efficient in terms of memory consumption. In fact, for the mouse data set described in Appendix A.3, the image arrays consume approximately one gigabyte of memory when using single precision floating point numbers to represent each system matrix element, which can quickly become prohibitive. Furthermore, if multiple threads are computing system matrix elements for different rows, then we would have to store multiple image-sized arrays.

Thus, as an alternative to the direct storage approach, we store two arrays in a class: one of system matrix values and one of voxel indices corresponding to the system matrix values. The arrays grow dynamically (doubling in size each time the current size is exceeded), but a static variable in the class records the maximum row size used thus far and all new rows are created with that size. Therefore, the class very quickly adapts to an appropriate size and very few memory reallocations occur. However, this approach also requires a boundary check each time a new element is inserted into a row, which occurs very frequently. In terms of run-time performance, this boundary check seems to have a negligible impact. We use this data structure to store system matrix elements computed by all of the subsequent system models as well.

2.1.2 Voxel Support

While the line intersection model is fast, simple to implement, and generates smaller system matrices than most other models, it unfortunately introduces ring artifacts when used for circular orbit cone-beam reconstructions. This phenomenon was reported by Zeng and Gullberg [14] who observed that "[t]he ring artifacts are due to unequal weighting as the ray passes from voxel to voxel". Another issue affecting such models is uneven sampling due to the divergent projection rays, which was addressed by Mueller et al. [5]. We will further characterize these artifacts in terms of voxel support and explore some potential solutions.

Inconsistency issues in the voxel support may or may not be evident in the reconstructed images, but it would obviously be beneficial to remove the inconsistencies when feasible. We will demonstrate inconsistencies in the line intersection based system matrix that can be observed in the voxel support values. After noting these inconsistencies, we will illustrate how they can generate rings in the reconstructed images.

Consider the voxel support values for the line intersection based system model associated with the scanner geometry used to acquire the tiny hole phantom data set in Appendix A.2. Figures 2.2, 2.3, and 2.4 show several transaxial slices of voxel support values interpreted as gray level images for view angle zero with line plots through $y = 200$ for each slice. There are two problems in these transaxial support values: the curved lines that occur in all slices, and the vertical bands that occur in some slices. Similar patterns were observed by De Man and Basu [10].

However, these same curved artifacts also exist in the coronal slices. The vertical band artifacts in the transaxial slices are then due to the intersection of a transaxial slice with the curved artifacts in the coronal slices. The width of the vertical bands depends upon the point of intersection of the transaxial slice with the curved artifacts in the coronal slices. If the curved artifacts in the coronal slice are flatter with respect to the transaxial

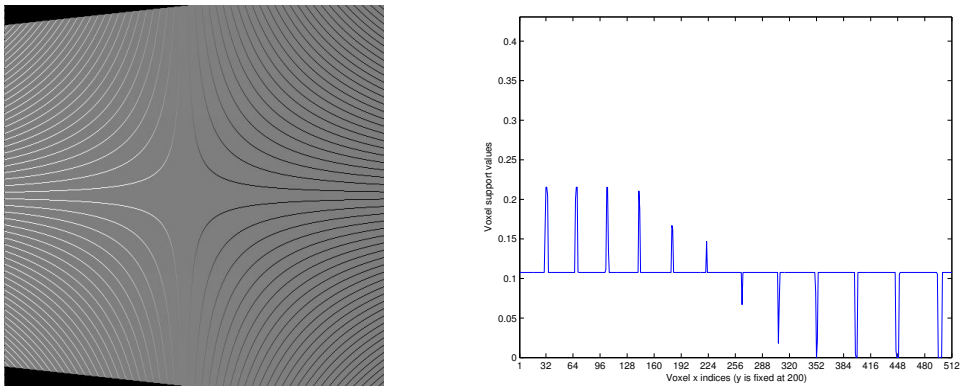

Figure 2.2: Left: Transaxial slice 127 of the line intersection based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

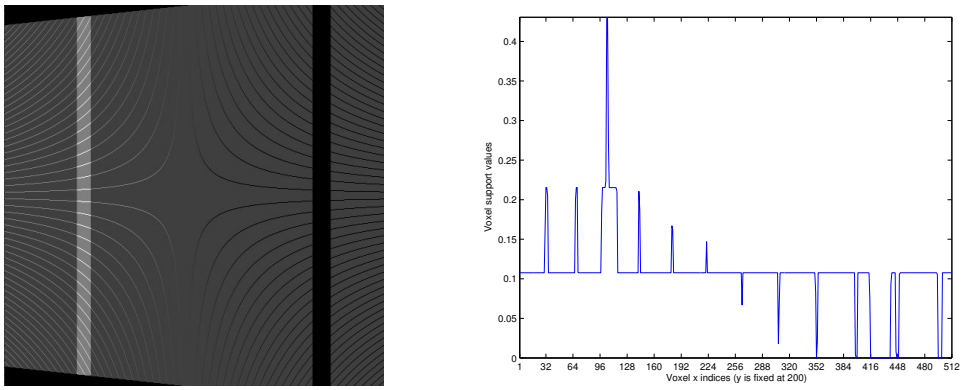

Figure 2.3: Left: Transaxial slice 120 of the line intersection based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

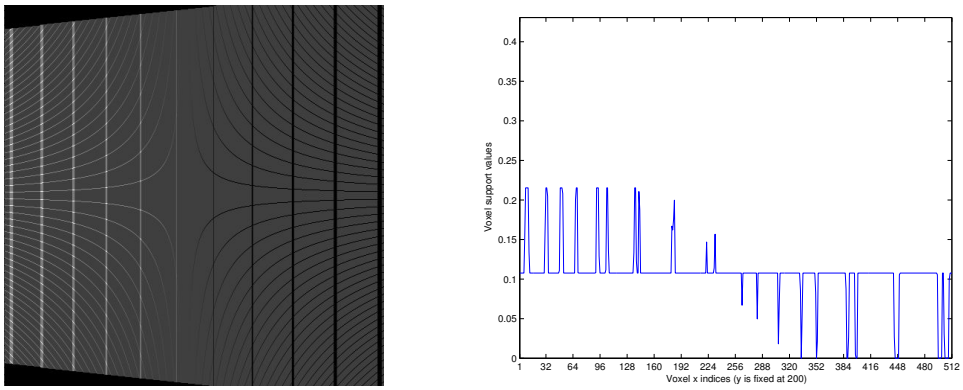

Figure 2.4: Left: Transaxial slice 80 of the line intersection based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

direction at the point of intersection with the transaxial slice, then the vertical band will have a larger width. Furthermore, the coronal curve artifacts shift with respect to the view angle, which rotates the vertical band artifacts into sloped band artifacts. These band artifacts are those that ultimately manifest themselves as ring artifacts in the voxel support values resulting from considering all projections, and also potentially as ring artifacts in the reconstructed images. We will first demonstrate the existence of the ring artifacts and then address how the vertical bands rotate into sloped bands for other projection view angles and thus generate the ring artifacts.

Figures 2.5, 2.6, and 2.7 contain transaxial support slices with line plots through $y = 200$ when considering all 720 projections. We see that the near central transaxial slice in Fig. 2.5 exhibits approximately even support over the supported region with no clear ring artifacts. On the other hand, the non central transaxial slices in Figs. 2.6 and 2.7 exhibit obvious ring artifacts consistent with the location of the vertical bands present in projection zero. Thus, we would expect the backprojection operation to generate ring artifacts in the reconstructed transaxial slices as well. We explore the effects on an actual reconstruction further in Section 2.4 where we compare reconstructions performed using the various system models. In addition, we consider several variations of the line intersection based model in the following section in order to determine the voxel support resulting from those models.

In order to demonstrate the existence of the sloped bands in rotated projection views, we also calculated the voxel support values corresponding to a projection view with the x-ray source rotated 45 degrees. The results are shown in Figs. 2.8, 2.9, and 2.10 for transaxial slice 120 and coronal slices 200 and 300. Figure 2.8 demonstrates the rotation of the bands in the transaxial slices that ultimately causes the emergence of the ring artifacts in the transaxial slices of the voxel support. We see the cause for the sloping of the bands in Figs. 2.9 and 2.10: the curved patterns still exist in the coronal slices, but shift from one slice to the next, so intersecting a stack of coronal slices with a transaxial slice generates a sloped band in the transaxial slice.

2.1.3 Multiple Line Intersection Model

There are several variations of the line intersection based model that may reduce the support based ring artifacts presented in Section 2.1.2. In this section, we consider varying the number of projection rays per detector element as well as the terminal point selection of the projection rays in the detector elements. When using multiple projection rays per detector element, we sum the system matrix elements from each projection ray and compute the average values to get the final system matrix elements. We give three possibilities of point selection in Fig. 2.11. These possibilities include using the center of the detector element (which we considered previously), the corners of the detector element, and points packed evenly within the detector element. The evenly packed points are chosen such that distances between any two neighboring points are equal whether the neighboring point is in the same detector element or a neighboring element. We also consider randomly selecting some number of points per detector element. While the points could be chosen in any number of ways, we constrain ourselves to these variations.

However, none of the variations that we tested seemed to offer a significantly improved system model in terms of voxel support. In Figs. 2.12, 2.13, and 2.14 we show support values corresponding to projection view zero for transaxial slice 80 using three different variations

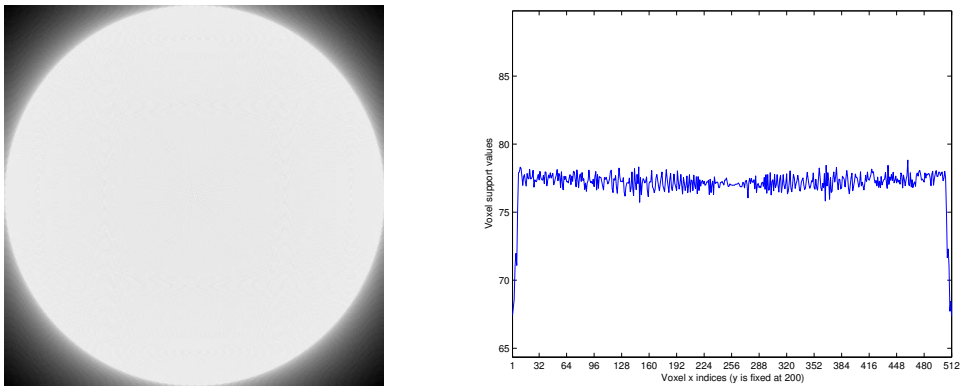

Figure 2.5: Left: Transaxial slice 127 of the line intersection based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

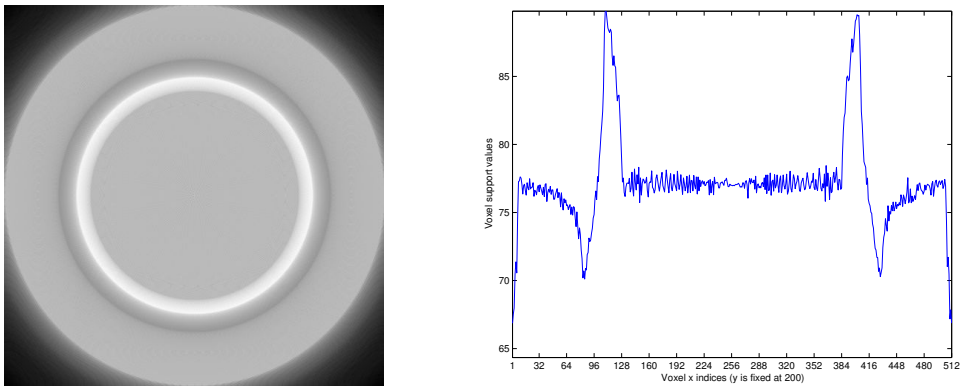

Figure 2.6: Left: Transaxial slice 120 of the line intersection based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

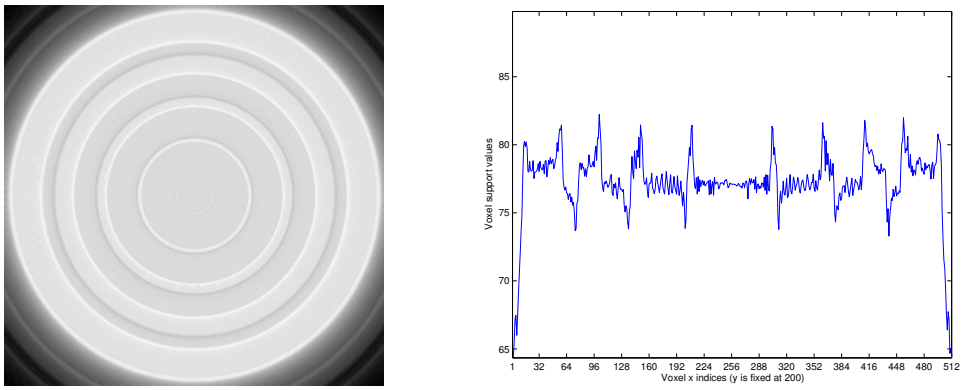

Figure 2.7: Left: Transaxial slice 80 of the line intersection based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

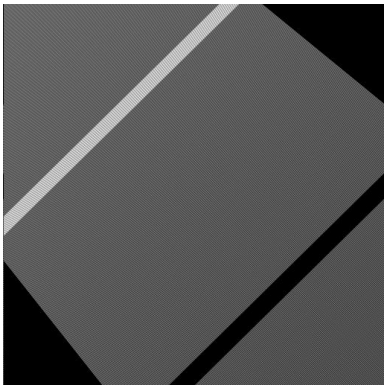

Figure 2.8: Transaxial slice 120 for a projection corresponding to a 45 degree view angle.

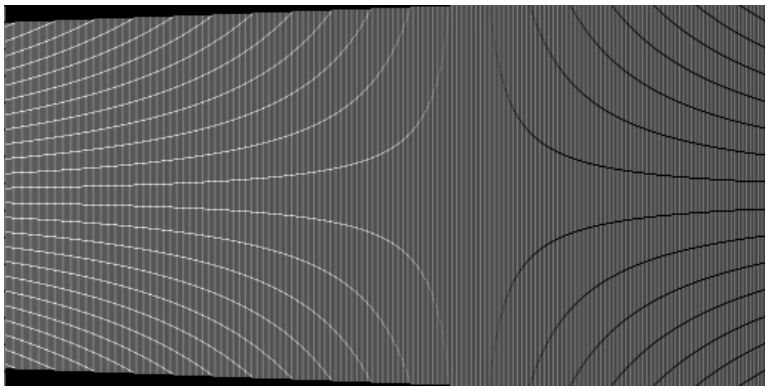

Figure 2.9: Coronal slice 200 for a projection corresponding to a 45 degree view angle.

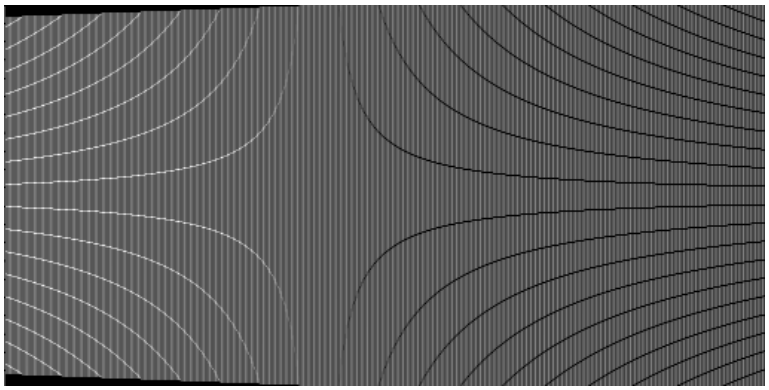

Figure 2.10: Coronal slice 300 for a projection corresponding to a 45 degree view angle.

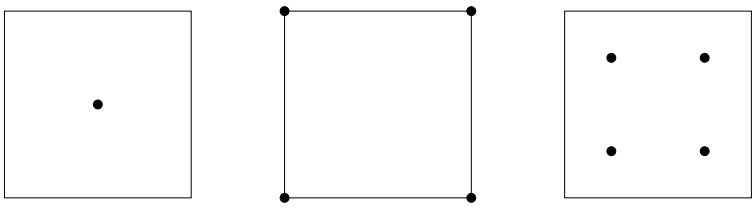

Figure 2.11: Several approaches for choosing points in the detector element by which we define our projection rays. The filled dots denote the points. The approaches include choosing the central point (left), the four corners of the element (center), and equally spaced points within the element (right). There are many other possibilities.

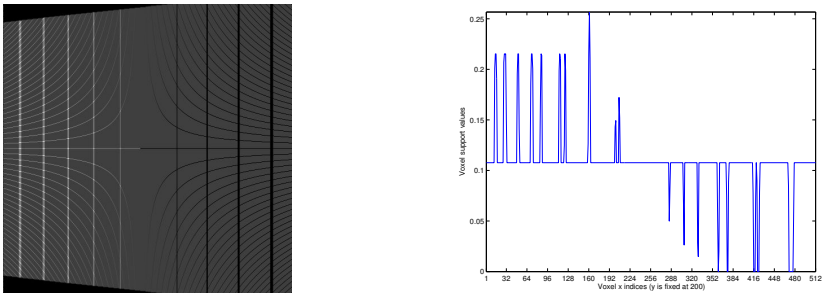

Figure 2.12: Left: Transaxial slice 80 of the line intersection based system matrix column sums corresponding to view angle zero and using the four corners of each detector element as the projection ray terminal points. Right: Line profile with $y = 200$.

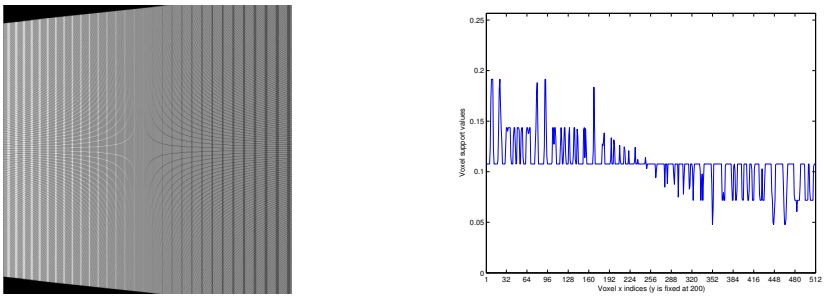

Figure 2.13: Left: Transaxial slice 80 of the line intersection based system matrix column sums corresponding to view angle zero and using nine equally distanced points per detector element as the projection ray terminal points. Right: Line profile with $y = 200$.

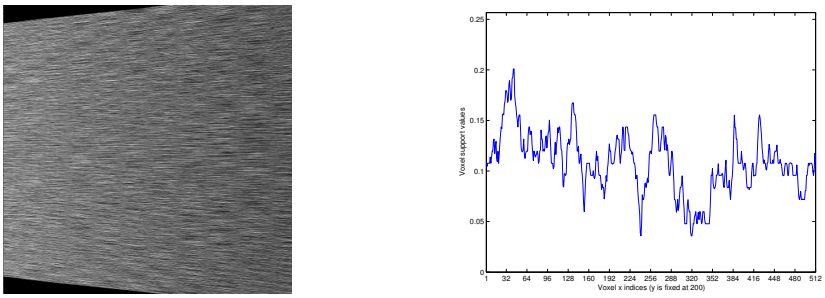

Figure 2.14: Left: Transaxial slice 80 of the line intersection based system matrix column sums corresponding to view angle zero and using nine random points per detector element as the projection ray terminal points. Right: Line profile with $y = 200$.

of the line intersection based system model. Also shown are the the profiles corresponding to $y = 200$ for each of the transaxial slices. The variations include using the four corners of each detector element, nine equally spaced points within each detector element, and nine random points within each detector element. Using the four corners and nine packed points still results in bands in the voxel support, although the patterns are slightly different from when using only the central line. Randomizing the projection ray points yields a rather chaotic support profile that would very likely lead to poor reconstructions. Furthermore, the computational cost increases with the number of projection rays per detector element, so the multiple line approach increases the total run-time significantly as the number of projection rays increases. Thus, these variations in the line intersection model do not seem to generate good enough results to justify the increase in the cost of computing the system model several times per detector element.

2.2 Interpolation Models

The interpolation model considered here models the projection ray as a group of equidistant points lying along a line based projection ray. For each of these equidistant sample points, we identify the eight nearest voxel centers and calculate the interpolation coefficients corresponding to the voxel centers. We then assign these interpolation coefficients to the appropriate voxels represented by the voxel center. This approach distributes support values to neighboring voxels which mitigates the artifacts associated with the line intersection based method. Trilinear interpolation has been employed by others, e.g. Wang [15], and its two-dimensional analogue, bilinear interpolation, has been used in the two-dimensional case by Andersen and Kak [16]. While we choose to use equidistant sample points, another approach by Zeng and Gullberg chose the sample points as the midpoints of the intersections of the projection ray with each voxel [14].

Interpolation is a technique typically used to generate new data points from known data points. For example, given two points x_1 and x_2, the midpoint is $m = (x_1 + x_2)/2$. If we have some function f with known values $f(x_1)$ and $f(x_2)$, then we can approximate $f(m)$ using linear interpolation as $f(m) = (f(x_1) - f(x_2))/2$. More generally, if $x_1 < x_2$, then any point $x \in [x_1, x_2]$ can be represented by $x = x_1 + t(x_2 - x_1)$ for some $0 \leq t \leq 1$ where t is the interpolation coefficient. We can loosely interpret t as defining the fractional contributions made by the functional values at x_1 and x_2 to yield $f(x)$: $f(x)$ is composed of $(1 - t)$ parts of $f(x_1)$ and t parts of $f(x_2)$. In other words, we can approximate $f(x)$ as

$$f(x) = (1 - t)f(x_1) + tf(x_2). \tag{2.2}$$

Linear interpolation extends to bilinear interpolation in two dimensions and trilinear interpolation in three dimensions. Figure 2.15 depicts the situation for bilinear interpolation in a square grid. In this figure, the four reference points are given by (x_i, y_j), (x_i, y_{j+1}), (x_{i+1}, y_j), and (x_{i+1}, y_{j+1}) and the interpolated point is (x, y). Let f represent the function defined for all pixel centers that maps a pixel center to the value of its associated voxel.

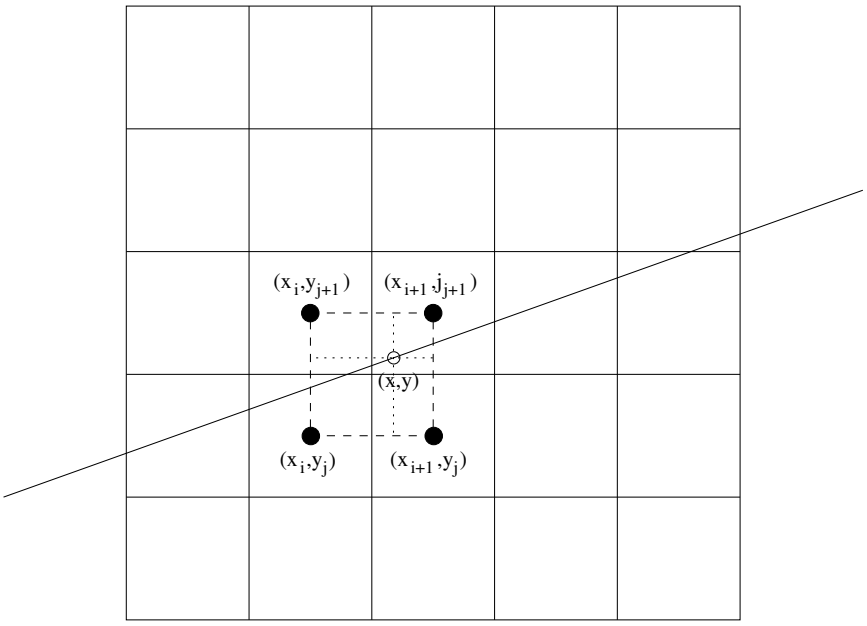

Figure 2.15: Depiction of the process of bilinear interpolation.

Table 2.1: The eight points representing voxel centers and the associated interpolation coefficients used in trilinear interpolation.

Voxel Center	Interpolation Coefficient
(x_i, y_j, z_k)	$(1-t)(1-u)(1-v)$
(x_{i+1}, y_j, z_k)	$t(1-u)(1-v)$
(x_i, y_{j+1}, z_k)	$(1-t)u(1-v)$
(x_{i+1}, y_{j+1}, z_k)	$tu(1-v)$
(x_i, y_j, z_{k+1})	$(1-t)(1-u)v$
(x_{i+1}, y_j, z_{k+1})	$t(1-u)v$
(x_i, y_{j+1}, z_{k+1})	$(1-t)uv$
$(x_{i+1}, y_{j+1}, z_{k+1})$	tuv

Then we can approximate $f(x, y)$ as

$$\begin{aligned} f(x,y) &= (1-t)(1-u)f(x_i, y_j) + t(1-u)f(x_{i+1}, y_j) + \quad (2.3) \\ &\quad (1-t)uf(x_i, y_{j+1}) + tuf(x_{i+1}, y_{j+1}). \quad (2.4) \end{aligned}$$

In this case, $f(x, y)$ is composed of $(1-t)(1-u)$ parts of $f(x_i, y_j)$, $t(u-1)$ parts of $f(x_{i+1}, y_j)$, etc. When we extend to trilinear interpolation, we get eight reference points and the associated interpolation coefficients shown in Table 2.1. For each of the equidistant points along the projection ray, we identify the eight nearest voxel centers, calculate the interpolation coefficients as shown in Table 2.1, and assign those interpolation coefficients to the appropriate voxel. Since a voxel center may be one of the eight nearest voxel centers for multiple sample points along a single projection ray, the system matrix value associated with that voxel is the sum of all such interpolation coefficients.

2.2.1 Implementation

Since the system matrix can become very large, it is important to minimize the amount of computation required to calculate the system matrix elements. Thus, the implementation of the system model is quite important. In this section, we present several details that affect the efficiency of the trilinear interpolation implementation, one of which will have implications in the algorithm implementation as well.

The first implementation detail to consider is the storage of the computed trilinear interpolation coefficients. Since consecutive sample points may generate interpolation coefficients for the same voxel, all interpolation coefficients corresponding to a given voxel must be summed to yield the true system matrix value for that voxel. This requirement can be implemented using several different data structures. For example, we could simply allocate an image sized array of floating point numbers and add interpolation coefficients as they are generated. However, as discussed in Section 2.1.1, this is not practical due to memory consumption. A more elegant approach would be to use an associative array or lookup table to store data only for voxels which have nonzero values. This could be done, for example, by using the `std::map` container in C++. While this approach works well

for a single thread, performance suffers significantly when using multiple threads*. Even if insertion was faster with multiple threads, accessing the data in the algorithm code would not be straightforward and would likely be slower than necessary. Instead, we use the same approach discussed in Section 2.1.1 where we allocate one array for voxel indices and another for system matrix values. Each time we generate an interpolation coefficient and associated voxel index, we append the information to the arrays, increasing their size if needed as described previously. Thus, we do not sum the interpolation coefficients generated for a given voxel and instead store partial sums. This is sufficient for most purposes and does not require much additional memory, although it can potentially limit us in the algorithm implementation; see Section 3.3.3 for an example of one such limitation.

There are other considerations that significantly affect the performance of the trilinear interpolation code. For example, a direct implementation would require repeated integer truncations, which are quite slow, in order to calculate the voxel indices. We avoid these repeated truncations by truncating once and then updating the index values using loops. When assigning interpolation coefficients to neighboring voxels, it is important to verify that the neighboring voxels are valid, so boundary testing may take an inordinate amount of time. When considering a full image, this is only a concern on the edges, but it will be complicated by the region restriction methods introduced in Sections 4.4 and 4.5. Thus, for a fixed pair of y and z voxel indices, we store the minimum and maximum valid x voxel indices for which all of the neighbors of the voxel identified by x, y, and z are also valid. As we will discuss in Chapter 4, all voxels between the minimum and maximum valid indices in the x direction are assumed valid. Therefore, we can test a voxel to determine if it is within this safe region, and if so, then we can insert values for all of the neighboring voxels without additional boundary checks. This replaces eight boundary checks by a single check in most cases and significantly reduces the total number of boundary checks performed.

2.2.2 Voxel Support

We will consider the voxel support values corresponding to projection zero and all projections as we did for the line intersection based system model. In Figs. 2.16, 2.17, and 2.18, we show the voxel support values for several transaxial slices and the associated profiles through $y = 200$. Although the curved patterns still exist in the trilinear interpolation support, they are much less prevalent than in the line intersection based models. This is likely due to the fact that the sample points are still chosen along the projection ray and thus share similarities with the line intersection based model, but the interpolation procedure distributes values to nearby voxels and prevents the sharp difference in values noticeable when using line intersection lengths.

However, the support values are now higher closer to the x-ray source due to the higher sampling near the x-ray source. There are some vertical bands as we observed in the line intersection based support, but the bands correspond only to higher support rather than higher support closer to the x-ray source and lower support farther from the x-ray source as is the case in the line intersection based model. Furthermore, the bands are much less pronounced and do not generate obvious ring artifacts as we show shortly.

*This is true with gcc as of the writing of this document. For more information, see http://gcc.gnu.org/bugzilla/show_bug.cgi?id=13823. Other compilers or `std::map` implementations may not exhibit this behavior.

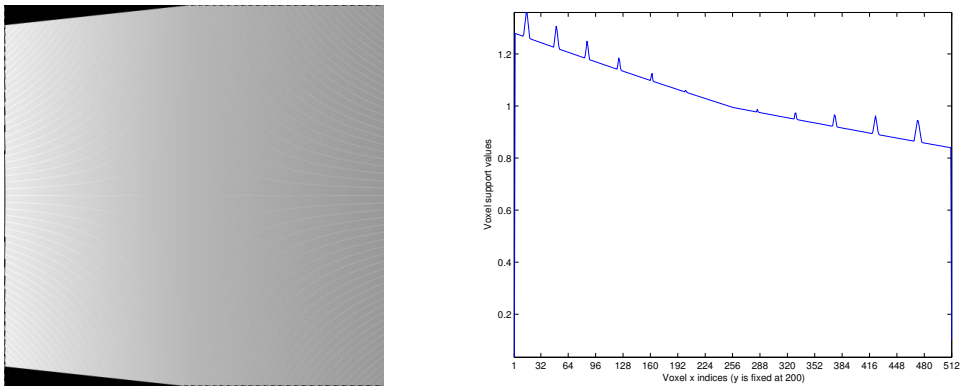

Figure 2.16: Left: Transaxial slice 127 of the trilinear interpolation based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

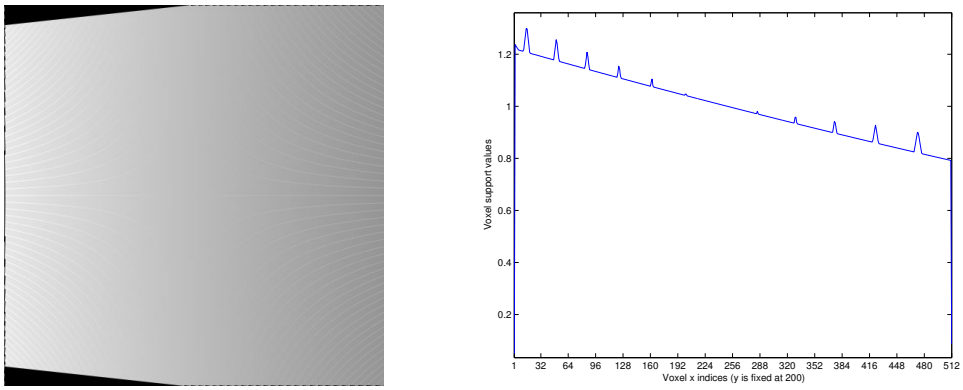

Figure 2.17: Left: Transaxial slice 120 of the trilinear interpolation based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

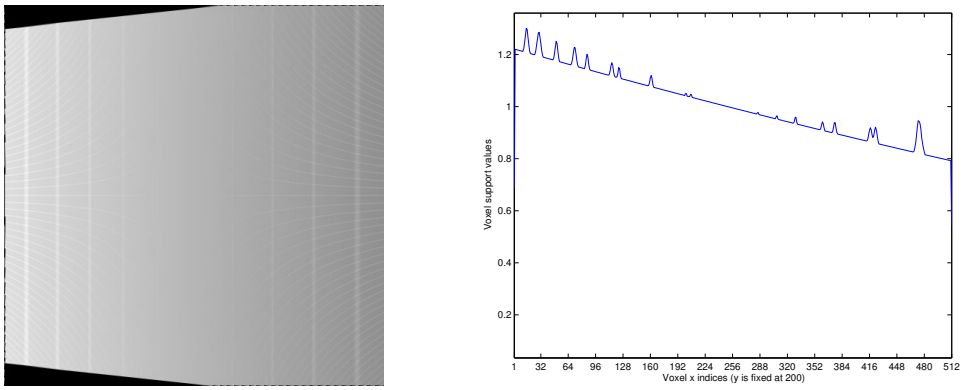

Figure 2.18: Left: Transaxial slice 80 of the trilinear interpolation based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

In Figs. 2.19, 2.20, and 2.21, we show the voxel support corresponding to the same transaxial slices when considering all projection view angles as well as the support profiles through $y = 200$. In these images, there are no clear ring patterns as there were in the line intersection based models, although the support values tend to dip in the center of each transaxial slice. We could compensate for this dipping by applying a weighting factor, but we have not investigated the matter enough to determine if the weighting would improve image quality, so we do not consider it further in this work.

In terms of support value consistency, the trilinear interpolation model is significantly better than the line intersection based models. In particular, the ring artifacts seem to be eliminated, or at least significantly reduced if they are in fact present. In Section 2.4, we will further compare the system models based upon reconstruction quality and computational time.

2.3 Volumetric Models

The volumetric system models considered in this work model the projection ray as a three-dimensional polytope instead of a line. Thus, it is not really a projection "ray", but we maintain the terminology for consistency. This polytope is formed by connecting the x-ray source to the detector element boundaries. In other words, the x-ray source is the apex of the polytope and the detector element is the base. System matrix element a_{ij} is then the volume of the intersection between projection ray i and voxel j. We can either calculate this volume exactly or approximate it in some way.

In order to compute the exact volume of the intersection between a projection ray and a voxel, we first identify the vertices of the polytope resulting from the intersection. While this is theoretically rather simple, it is somewhat complicated by the computer based finite representation of floating point numbers. However, it can be done in the vast majority of cases. Once the vertices, and thus the faces, of the polytope are known, we can compute the volume of the polytope by triangulating the faces, calculating the volume of the tetrahedrons joining the triangulated regions and the centroid, and summing the tetrahedron volumes to obtain the final volume. There is a closed form formula for calculating the volume of tetrahedrons, so that step is rather simple (see, e.g., [17]). Furthermore, since the polytope is the intersection of two convex shapes, namely the projection ray and voxel, it is also convex, and thus its centroid is located inside the polytope. Therefore, the tetrahedrons partition the polytope and the sum of their volumes is the volume of the polytope.

In rare cases, the floating point number representation can become problematic. For example, two or more computed values may be so close together that it is difficult to determine from their floating point representations whether they are distinct vertices or the same vertex. In these cases, we employ the volumetric approximation approach, which we present shortly, to approximate the volume of the intersection as closely as needed. Additionally, while the presented method calculates the volume intersection exactly in theory, in practice there will be roundoff errors in the calculation of the vertices, tetrahedron volume, and final polytope volume, although these errors should be small. While we have an implementation of an exact volumetric system model using these approaches, it is very computationally intensive and thus we do not use it for practical reconstructions.

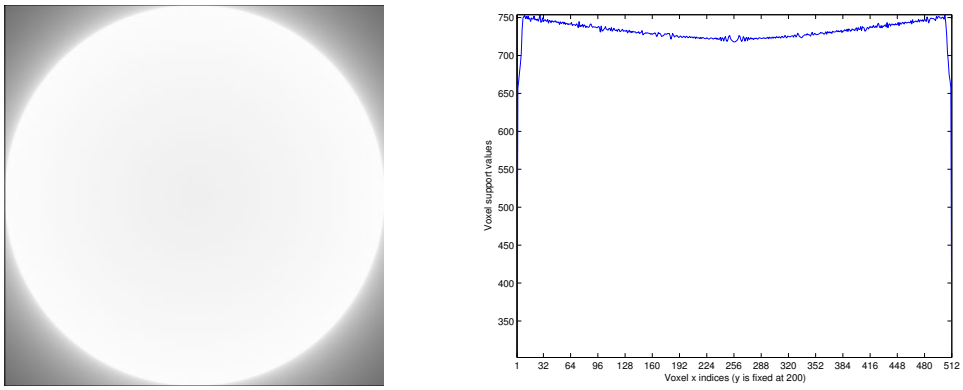

Figure 2.19: Left: Transaxial slice 127 of the trilinear interpolation based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

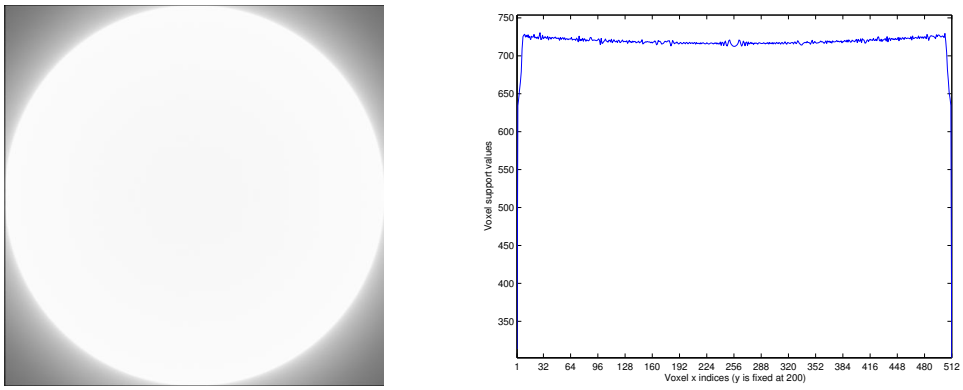

Figure 2.20: Left: Transaxial slice 120 of the trilinear interpolation based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

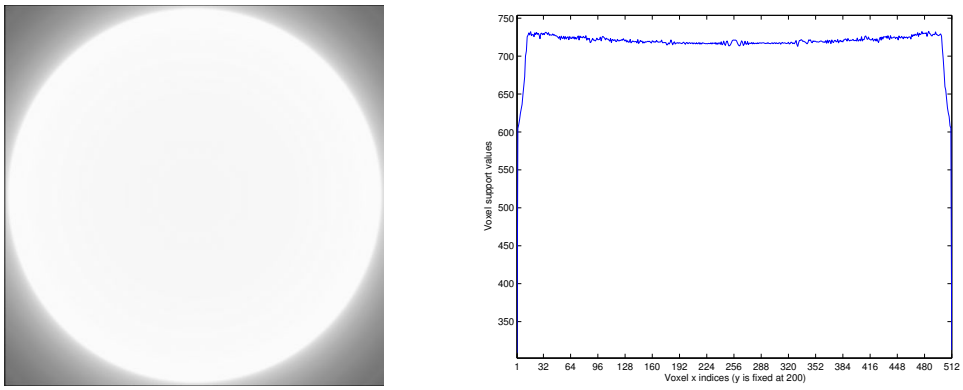

Figure 2.21: Left: Transaxial slice 80 of the trilinear interpolation based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

In addition to the exact volumetric model, we have implemented a volumetric approximation model. The approximation model divides the voxel into some number of subvoxels and determines the number of subvoxel centers contained within the projection ray. The system matrix element a_{ij} is then based on the number of subvoxels of voxel j whose centers are contained within projection ray i. There are many approaches to computing the system matrix element: we can use the integral number of contained subvoxel centers, divide the number of contained subvoxel centers by the total number of subvoxels to obtain a normalized fractional system matrix element, or compute the normalized fractional value and multiply by the voxel volume to obtain a true volumetric approximation. Since the resultant system matrices differ only by a scalar value, the approaches are mathematically very similar.

By increasing the number of subvoxels used, we can obtain increasingly accurate volume approximations. In the case of using only one subvoxel, the model simplifies to the binary model that we discuss in Section 2.3.1. Determining the actual number of subvoxels needed for a system matrix to be accurate enough for a particular usage would require careful consideration.

2.3.1 Binary Models

Binary system models, which assign the value zero or one to each system matrix element, were popular in the past (see, e.g., [18]). In these models, some method is used to determine if voxel j should be considered within projection ray i: if so, then $a_{ij} = 1$, and $a_{ij} = 0$ otherwise. With two-dimensional reconstructions, containment was generally determined by the containment of a pixel center within a strip defining the projection ray. In the fan-beam case, this strip could be defined as the area joining the x-ray source and the detector element boundaries with the detector element as a base and the x-ray source as an apex. In the cone-beam case, this extends to the polytope described in Section 2.3. Therefore, the most natural binary model for cone-beam CT would be to set $a_{ij} = 1$ if the center of voxel j is contained within projection ray i as defined by the polytope. This would not, however, be an accurate approximation to the volumetric intersection between the projection ray and voxel and thus would not likely facilitate high quality reconstructions.

2.3.2 Implementation

As with all of the system models, an efficient implementation is paramount since the code will be executed very frequently. We only discuss the implementation of the volumetric approximation code because the exact volumetric code is complicated by floating point precision issues. Since we do not use this model in practice, we have only optimized the implementation to the point that we can reasonably use it for comparison purposes. However, there is very likely a more efficient approach than we present here.

We divide the volumetric approximation system model computation into two steps. We first identify a list of candidate voxels that may be intersected by the projection ray, and then test each voxel to determine the number of contained subvoxel centers. Of course, we want to make the likelihood that a candidate voxel is intersected by the projection ray as high as possible, although it may not be computationally advantageous to guarantee an intersection. We find our candidate voxels using an octree approach. This approach

proceeds by recursively dividing segments of the voxel space into eighths. We then test each new subdivision for intersection with the projection ray and further subdivide those that are intersected. Thus, we quickly remove from consideration regions that are far from the projection ray and recursively subdivide regions near or inside the projection ray. If we continue this process until every region has either been removed from consideration or only consists of one voxel, then the candidate list will contain exactly those voxels intersected by the projection ray. However, once a region contains eight or fewer voxels, we add all of the voxels to the candidate list as that approach is slightly faster in practice.

After we have a list of candidate voxels, we use dot products to determine whether or not a subvoxel is contained within the polytope. By computing the dot product between a subvoxel center and the normal for a plane corresponding to a polytope side, we can determine on which side of the plane the subvoxel center lies. Thus, if the subvoxel center lies on the "inside" of all of the polytope planes, then that subvoxel center is considered contained within the projection ray. Therefore, we compute these dot products for each subvoxel center for each candidate voxel and assign the number of subvoxel centers of voxel j contained in projection ray i to system matrix element a_{ij}. The loops computing these dot products can be arranged to store partial dot products in order to reduce the amount of computation.

2.3.3 Voxel Support

As with the line intersection and trilinear interpolation based models, we will analyze the voxel support corresponding to projection zero and all projections as a preliminary measure of the model effectiveness. We used a volumetric approximation model with twenty-seven subvoxels per voxel for the results in this section. In Figs. 2.22, 2.23, and 2.24, we show the voxel support for several transaxial slices corresponding to projection view zero and the associated support profiles through $y = 200$. In these figures, we observe that the voxels that are supported by the system geometry have constant support values. This is to be expected since each subvoxel center that is supported by the scanner geometry should be in exactly one projection ray. Thus, computing the voxel support values, or system matrix column sums, is equivalent to summing the number of subvoxels per voxel supported by a projection view. Similarly, we see in Figs. 2.25, 2.26, and 2.27 that the voxel support values for the supported voxels are also constant when considering all projection views. Thus, the volumetric approximation system model produces ideal relative support, although that does not mean that the model itself accurately models the attenuation process. Furthermore, these support results apply to the volumetric approximation model when using any number of subvoxels per voxel, assuming that we always use the same number of subvoxels for each voxel. The results also apply to the exact volumetric model in the ideal case, although the support values would not be constant in practice due to floating point roundoff errors. We will consider the performance of the volumetric system model in actual reconstructions in Section 2.4.

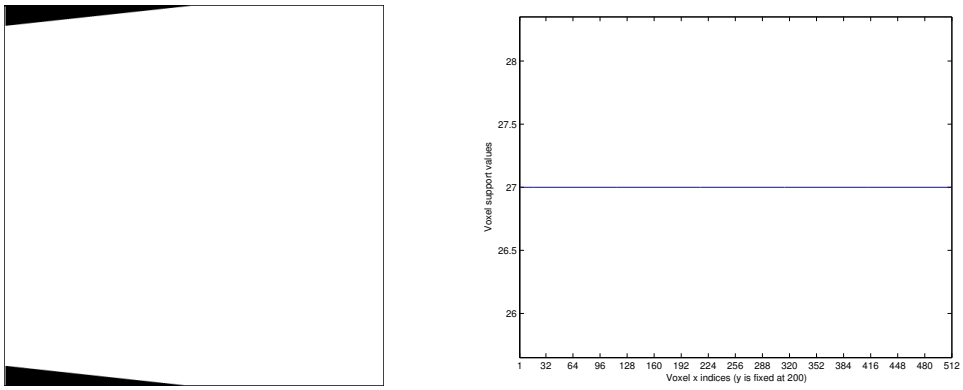

Figure 2.22: Left: Transaxial slice 127 of the volumetric approximation based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

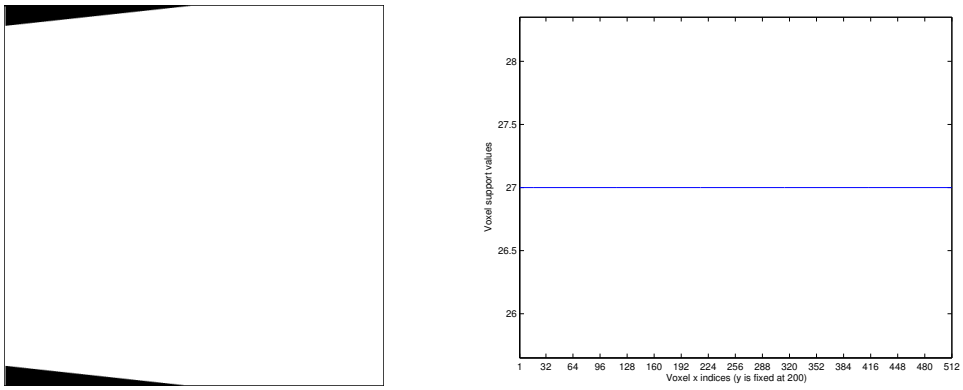

Figure 2.23: Left: Transaxial slice 120 of the volumetric approximation based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

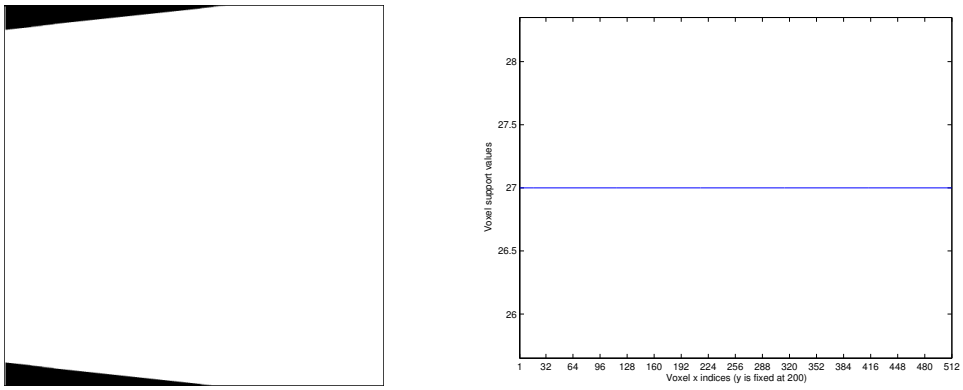

Figure 2.24: Left: Transaxial slice 80 of the volumetric approximation based system matrix column sums corresponding to view angle zero. Right: Line profile with y = 200.

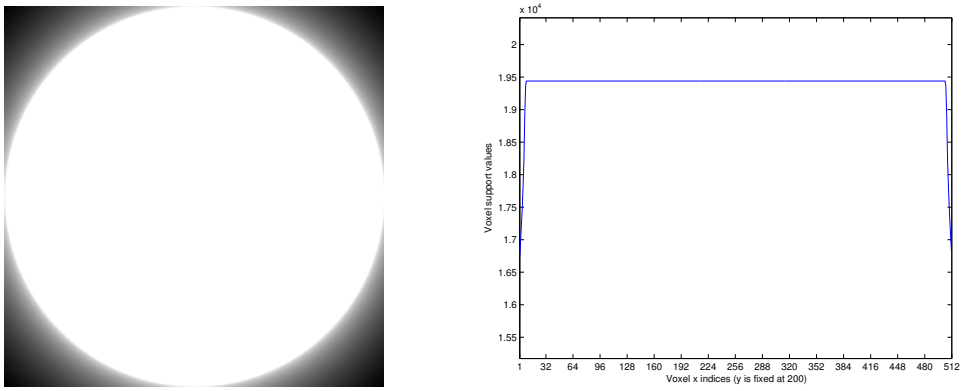

Figure 2.25: Left: Transaxial slice 127 of the volumetric approximation based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

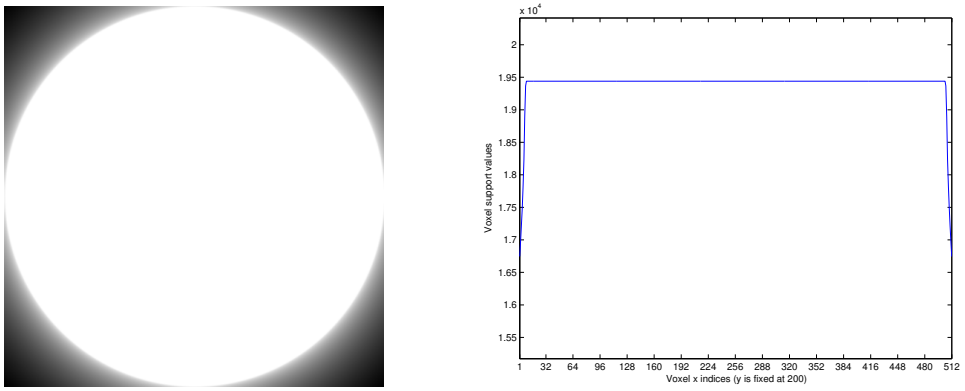

Figure 2.26: Left: Transaxial slice 120 of the volumetric approximation based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

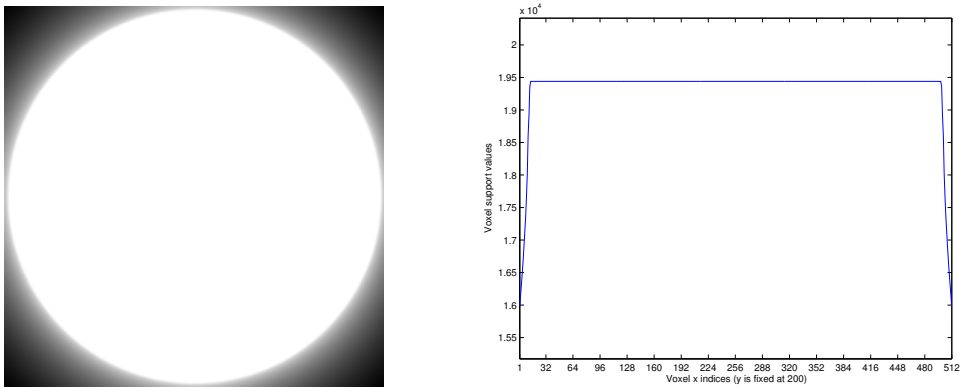

Figure 2.27: Left: Transaxial slice 80 of the volumetric approximation based system matrix column sums corresponding to all 720 view angles. Right: Line profile with y = 200.

2.4 System Model Comparison

While comparing the voxel support values for various system models is useful, ultimately the most important measurements of a system model are the quality of the reconstructions that it facilitates and the computational burden. Thus, we will examine those aspects of the system models in this section. However, it is difficult to directly compare the reconstructions to one another. In particular, all of the linear systems associated with the system models have the same b component, but the system matrix A depends upon the system model. Thus, since the values of A vary, the reconstructed images given by x will be significantly different from one linear system to the next and directly comparing the reconstructed attenuation coefficients will not be useful. Therefore, we will compare the images qualitatively, not quantitatively.

In Fig. 2.28 we show transaxial slices 141 and 181 of reconstructions of the tiny hole phantom using each of the system models. These reconstructions used five iterations of the SIRT algorithm with 48 ordered subsets, all of which will be described in subsequent chapters. Firstly, note that all of the reconstructions are similar, which is to be expected for reasonable system models. However, there are some artifacts visible in the line intersection based reconstructions. In particular, for the line intersection based reconstructions, transaxial slice 141 has an artifact above the drilled holes, which we term "flaring", and transaxial slice 181 has some ring artifacts intersecting the drilled holes. In Figs. 2.29 and 2.30 we zoom the images at the location of the artifacts for the line intersection based reconstructions and do the same for the trilinear interpolation based reconstructions for comparative purposes. Note that all of the reconstructions contain some ring artifacts that seem to be caused by the projection data as they are also present in filtered backprojection reconstructions. However, the additional ring artifacts in the line intersection based reconstruction are due to the rings in the transaxial voxel support. Thus, based on the visual quality of the reconstructions, the trilinear interpolation and volumetric approximation models appear to be superior, although the line intersection model would likely be sufficient for screening purposes or if the ring artifacts are deemed unimportant in a specific case.

The computational burden associated with each of the system models is also very important. Thus, we will compare the run-times of our current implementation of each of the system models. Note that this comparison is specific to our implementation and computing environment; it is entirely possible that much more efficient implementations of the system models are possible. In order to compare the run-times, we performed one iteration of SIRT, but did not perform the updates associated with the algorithm and only calculated the system matrix rows. In the case of the volumetric approximation based system model, we used twenty-seven subvoxels per voxel as in the presented reconstructions. As described in Chapter 4, the reconstruction software is parallelized using threads to distribute the workload to the processors on a given node and using message passing to distribute the workload among nodes. Here, we report the maximum run-time for all of the system matrix computation threads, which corresponds to a partial computation of the system matrix. However, it allows us to compare the amount of time spent computing a substantial number of system matrix rows with each model.

The maximum thread run-time was 72 seconds for the line intersection based system model, 344 seconds for the trilinear interpolation based system model, and 13343 seconds for the volumetric approximation based system model. Thus, the trilinear interpolation based

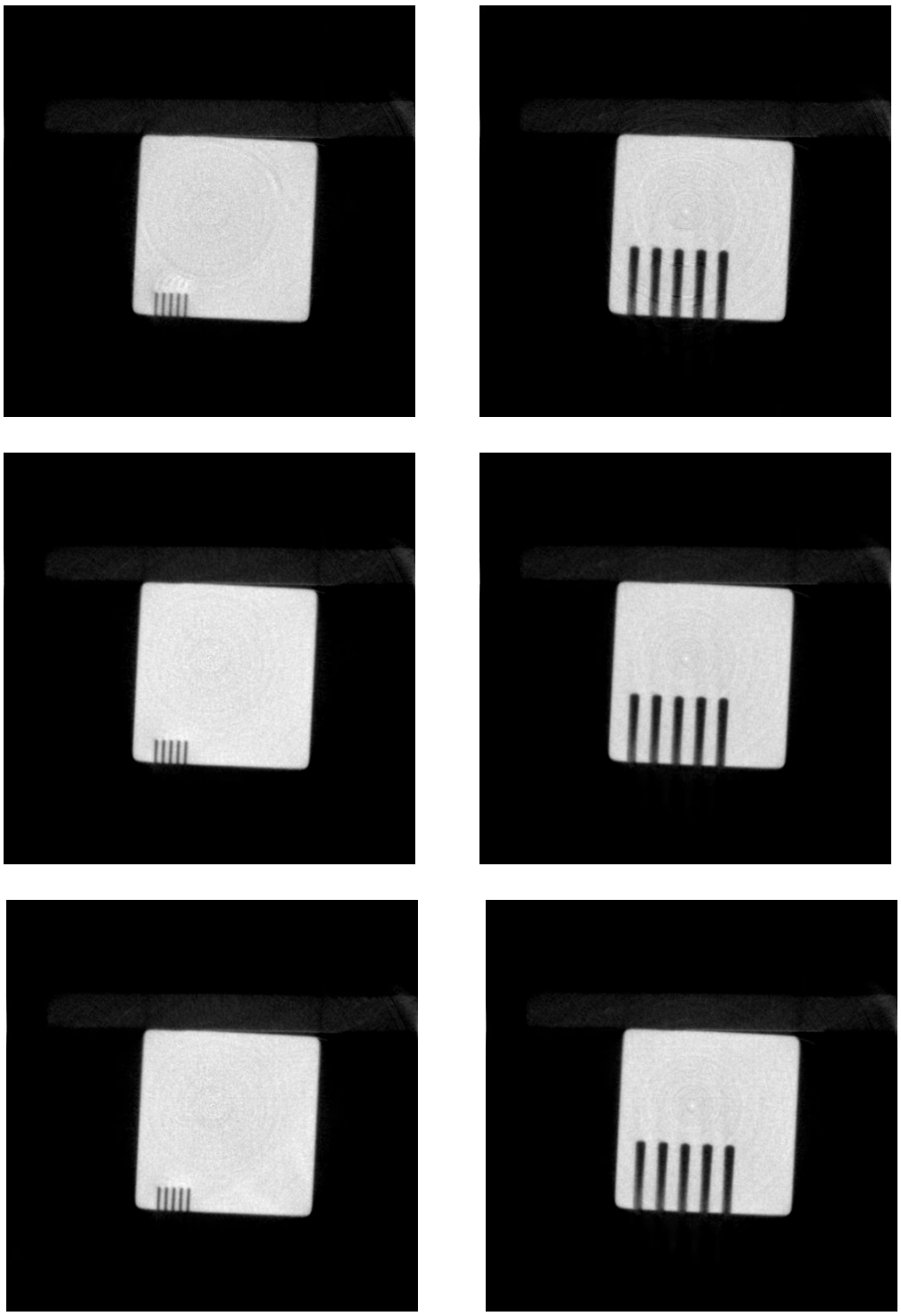

Figure 2.28: Transaxial slices 141 (left) and 181 (right) of the tiny hole phantom using the following system models: line intersections (top), trilinear interpolation (middle), and volumetric approximation (bottom).

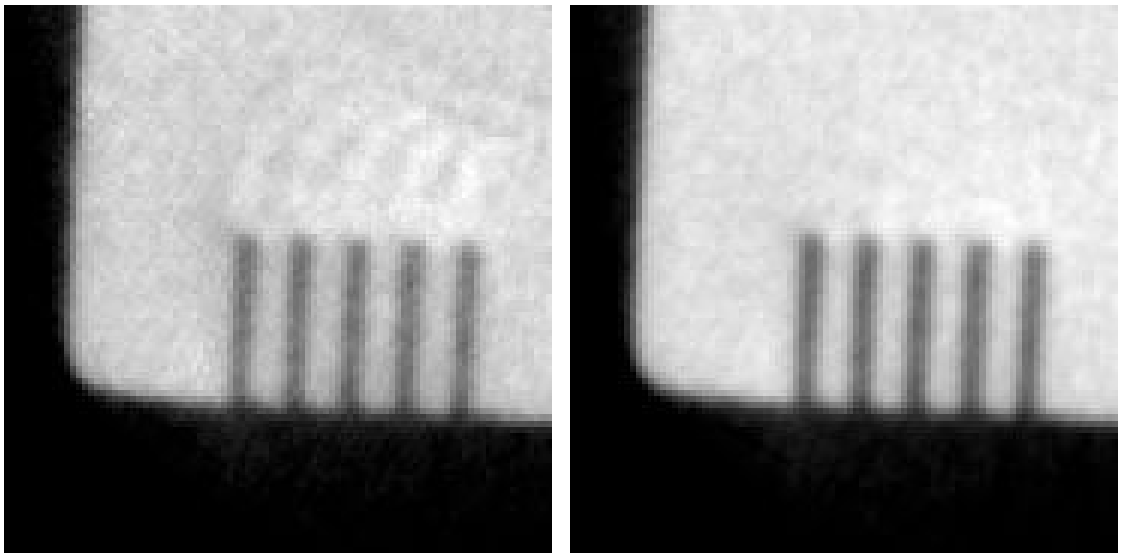

Figure 2.29: Zoomed version of the flaring artifact that is present in the line intersection based reconstruction (left) but not the trilinear interpolation based reconstruction (right).

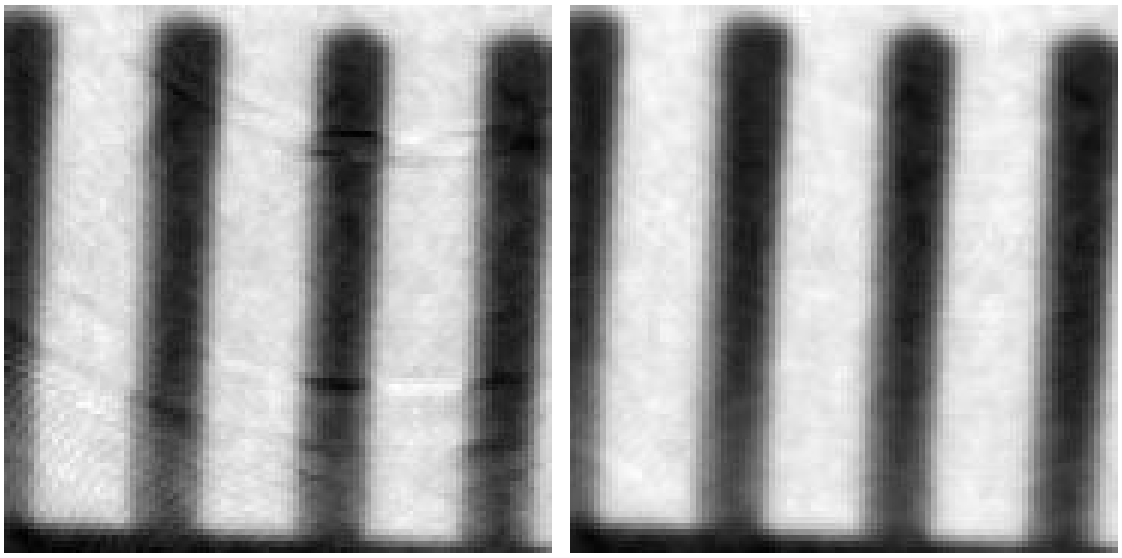

Figure 2.30: Zoomed version of the ring artifacts that are present in the line intersection based reconstruction (left) but not the trilinear interpolation based reconstruction (right).

model takes approximately 4.8 times longer to compute than the line intersection based model, and the volumetric approximation model takes about 185 times longer than the line intersection based model and 39 times longer than the trilinear interpolation based model. Note that these times are only for the system matrix computation. The algorithm updates are identical for each system model, so that aspect of the reconstruction will be nearly identical for each system model, although system models that generate more nonzero system matrix elements will have slightly higher associated algorithm updates. When accounting for other costs such as algorithmic computation and interprocessor communication, a trilinear interpolation based reconstruction actually takes approximately three times longer per iteration than a line intersection based reconstruction.

Based on the run-time, the line intersection based model is clearly the best, while the trilinear interpolation based model is acceptable. The current implementation of the volumetric approximation model, however, is clearly too costly for practical use. Therefore, as a compromise between image quality and computational run-time, we choose the trilinear interpolation based system model for the remainder of this work.

Chapter 3

Algebraic Reconstruction Algorithms

Once a system model has been selected, we effectively have a linear system given by $Ax = b$ where system matrix A is the embodiment of the system model and b represents the projection data. In order to compute an approximate solution to x, which represents the reconstructed image, we use an iterative reconstruction algorithm. In this chapter, we focus on algebraic reconstruction algorithms.

The ultimate goal of the algebraic reconstruction algorithms is to solve the linear system $Ax = b$ for x where A is an $m \times n$ matrix. While direct methods exist for solving such a system, such as applying Gaussian Elimination or Cholesky Factorization to the associated normal equations, the dimensions and potential lack of sparsity of the coefficient matrix for the normal equations given by $A^T A$ prevent these direct methods from being practical. On the other hand, the system matrices given by A are very large, but also very sparse, and the iterative methods presented in this chapter can be implemented using only a single row of the system matrix at a time. Censor refers to methods that use one matrix row at a time as row-action methods [19].

There are a large class of related algebraic algorithms available in the medical imaging literature that can be used to solve a linear system: algebraic reconstruction technique, simultaneous iterative reconstruction technique, simultaneous algebraic reconstruction technique, etc. Most, if not all, of these algorithms existed in some form in the mathematical literature previous to their rediscovery and adoption in medical imaging. In this chapter, we present the algorithms used for iterative reconstruction in medical imaging and relate them to their previous existence in other fields. This is not meant to be an exhaustive presentation of the history and development of these algorithms, but should add some perspective and intuition to their methodologies.

In addition to algebraic reconstruction algorithms, there is a group of statistical reconstruction algorithms as well. The statistical approaches differ from the algebraic approaches in that they attempt to iteratively increase the accuracy of the solution in some statistical context. One of the more successful statistical reconstruction algorithms in the field of medical imaging is the expectation-maximization (EM) algorithm. Shepp and Vardi proposed the EM algorithm for emission tomography in 1982 [2]. In 1984 Lange and Carson

independently derived the same algorithm for emission tomography as well as a complementary algorithm for transmission tomography [3]. While the transmission version of the EM algorithm introduced by Lange and Carson has not been widely used, most likely due to a proliferation of variables that would complicate the implementation, other statistical approaches have been used for transmission tomography. For example, Nuyts et al. proposed a statistical algorithm termed MLTR [20], which is a gradient-ascent based maximum likelihood algorithm for transmission tomography. This algorithm is developed by assuming Poisson-distributed photon counts, writing the log-likelihood resulting from an application of Bayes' Rule (ignoring the prior term), and finally applying gradient ascent to a truncated series expansion of the log-likelihood. Additionally, Elbakri and Fessler have also presented a penalized weighted least squares (PWLS) update based on a Poisson model [21], which they extend to polyenergetic x-ray CT. There are many other examples of successful applications of statistical iterative reconstruction algorithm in the medical imaging literature. However, we focus on the algebraic methods in this work so that we can more thoroughly explore the algorithms. That said, the techniques presented in Chapter 4 directly extend to most algorithms, including the statistical iterative reconstruction algorithms.

3.1 A Simple Example

Throughout this chapter, we will use two simple linear systems to demonstrate the iterations produced by the various algorithms. The systems have the same coefficient matrix, but one is consistent (i.e., there exists an exact solution) and the other is not (i.e., there is no exact solution). The goal of including these systems is to establish some intuition for the functioning of the algorithms and to provide concrete examples for many of the observations made in the text and citations. The goal is not, however, to compare algorithmic properties such as speed of convergence. The linear systems used for these examples are

$$\begin{bmatrix} 1 & 10 \\ 2 & 3 \\ 15 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 29.5 \\ 16.5 \\ 70 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 10 \\ 2 & 3 \\ 15 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 30.5 \\ 15.5 \\ 71 \end{bmatrix}.$$

The first system is consistent with solution $x = [4.5 \;\; 2.5]^T$ while the second system is inconsistent. Note that the right hand side of the inconsistent system is simply a perturbed version of the consistent system. This perturbation is meant to mimic the variations that may occur during the measurement process in order to demonstrate how the algorithms may react to noisy data. Furthermore, since the columns of the matrix are linearly independent, the matrix is of full rank and there exists a unique least squares solution for the inconsistent system given by

$$x^* = (A^T A)^{-1} A^T b \approx \begin{bmatrix} 4.553084 \\ 2.557767 \end{bmatrix}. \tag{3.1}$$

Since the matrix for the system is 3×2, the system can be visualized as three lines in the Cartesian plane. In the consistent case, the three lines intersect at the exact solution. On the other hand, in the inconsistent case, the three lines do not all intersect at any single point, but do define a region in the plane where potential solutions are near all lines

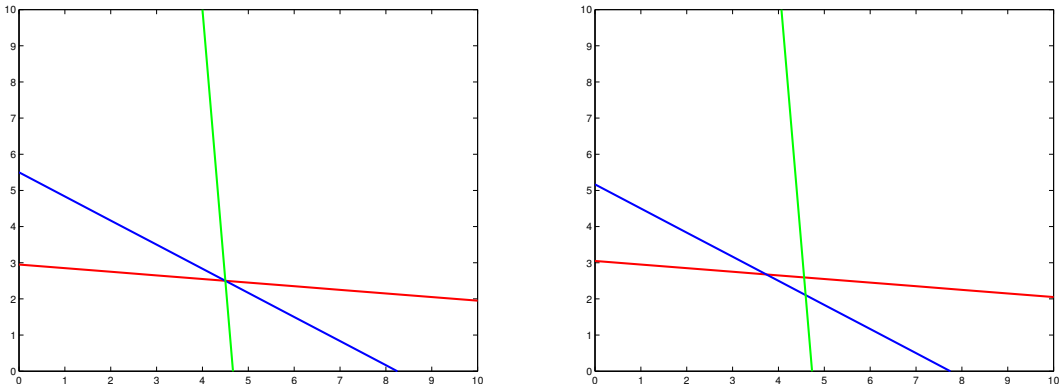

Figure 3.1: Graphs of the lines defined by the linear system examples. The left system has a unique solution where all three lines intersect while the right system has no unique solution.

(although some criterion would need to be defined to specify which point is "closest" to all three lines). Graphs of the two systems are shown in Fig. 3.1.

3.2 Algebraic Reconstruction Technique (ART)

In 1970, Gordon et al. proposed the algebraic reconstruction technique (ART) for reconstruction of objects in electron microscopy [18]. We will first present the more modern version of ART, using our notation, and then comment on its history and mathematical foundations. The ART algorithm, as presented by Kak and Slaney [22] and converted to our notation, is

$$x_j^{(k+1)} = x_j^{(k)} + a_{ij} \frac{b_i - \sum_{h=1}^{n} a_{ih} x_h^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} \tag{3.2}$$

where A is an $m \times n$ matrix, $i = k \mod (m+1)$, and $x^{(0)}$ is an initial image estimate. In vectorized form, ART is

$$x^{(k+1)} = x^{(k)} + a_{i*} \frac{b_i - a_{i*}^T x^{(k)}}{\|a_{i*}\|^2} \tag{3.3}$$

where a_{i*} is the ith row of A as a column vector. Since each update step only uses a single row of the system matrix, we will refer to a single update step as a subiteration, while an iteration denotes using each system matrix row once for a total of m subiterations. Some presentations of ART also include a relaxation parameter (e.g., [23]), but we do not include such a parameter in this work. Furthermore, if nonnegativity is required for the reconstructed image values, the values can be constrained to be nonnegative by simply assigning zero to an image element that would update to a negative value during a subiteration. If this requirement is imposed, we refer to the algorithm as constrained ART; otherwise, we refer to the algorithm as unconstrained ART. Since ART contains an additive update, it is possible for an image value to become positive in subsequent iterations after being constrained to zero.

The ART iteration presented by Gordon et al. [18] has a slightly different form because the authors used assumptions about the system model when developing the algorithm. In particular, the projection rays were modeled as strips of a given width extending through the image space and a binary system model was adopted: $a_{ij} = 1$ if voxel j is contained in projection ray i and $a_{ij} = 0$ otherwise. Thus, the leading weight given by a_{ij} in (3.2) disappears, as does the system matrix value given in the numerator, although the range of summation is then restricted to voxels intersected by the given projection ray. Furthermore, the weighting given by $\sum_{h=1}^{n} a_{ih}^2$ in the denominator reduces to the number of voxels contained in projection ray i. In order to maintain consistency in the presentation of the iterative algorithms, we will always separate the system model used to generate the system matrix A from the iterative algorithm used to solve the resulting system. Of course, certain knowledge about the system model may allow us to create a more efficient implementation of the algorithm, but the two can remain conceptually independent.

Herman et al. [24] generalized ART in 1973 to the modern version given above and placed the update on a firmer mathematical basis. In this paper, Herman et al. establish that both the constrained and unconstrained versions of ART converge to a solution, provided that such a solution exists. Furthermore, unconstrained ART converges to a minimum variance solution, although constrained ART does not. Finally, in the case of inconsistent data, and thus no solution, unconstrained ART cyclically converges. In other words, as the number of iterations approaches infinity, each subiteration converges to some solution vector, although the solution vectors for each subiteration may be different. Additional convergence work by Tanabe further characterized the convergence properties of the last element of the limit cycle, which is the value obtained after each full iteration of ART, as a generalized inverse applied to b [25]. Herman et al. and Tanabe further noted that ART had previously been proposed by Kaczmarz in 1937 for solving linear systems [26]. We will now present ART in the context of the method of projections as introduced by Kaczmarz.

3.2.1 ART as a method of projections

The method of projections offers an intuitive description for the functioning of ART. Although the 1937 paper by Kaczmarz is frequently cited, that paper is written in German and thus we have not read it. However, many others (e.g., [22, 23]) have summarized the method of projections; we give a presentation adapted from those sources in this section. The basic step of the method of projections, and thus of ART, is projecting the current solution estimate onto a hyperplane defined by a row of the system matrix. In the case of a consistent linear system, the hyperplanes all intersect in exactly one point, and by successively projecting the solution estimate orthogonally from one hyperplane to the next, the solution estimate continues to move closer to the true solution. In the inconsistent case, the solution estimate moves toward the region where the hyperplanes "nearly" intersect and oscillates in this region. We will first establish that a single step of ART does in fact orthogonally project the current solution estimate to the next hyperplane and then demonstrate the algorithm on our sample linear systems.

First consider the hyperplane defined by the ith row of the linear system:

$$H_i = \{z | a_{i*}^T z = b_i\} \tag{3.4}$$

where a_{i*}^T is the ith row vector of A. Assuming that a_{i*} has more than one nonzero element, then H_i contains an infinite number of points. Let z_1 and z_2 be two distinct points in H_i, then $a_{i*}^T(z_2 - z_1) = b_i - b_i = 0$, so a_{i*} is orthogonal to H_i. Furthermore, for any $z \in H_i$, the projection of z onto a_{i*} is given by

$$\text{proj}_{a_{i*}} z = \left(\frac{a_{i*}^T z}{\|a_{i*}\|^2} \right) a_{i*} = \left(\frac{b_i}{\|a_{i*}\|^2} \right) a_{i*}. \tag{3.5}$$

In particular, the orthogonal projection of our current solution estimate $x^{(k)}$ onto H_i, which we call $x^{(k+1)}$, is a point in H_i, so

$$\text{proj}_{a_{i*}} x^{(k+1)} = \frac{b_i}{\|a_{i*}\|^2} a_{i*}. \tag{3.6}$$

Now consider the projection of the current iterate, $x^{(k)}$, onto a_{i*}, or

$$\text{proj}_{a_{i*}} x^{(k)} = \left(\frac{a_{i*}^T x^{(k)}}{\|a_{i*}\|^2} \right) a_{i*}. \tag{3.7}$$

The vector $x^{(k+1)} - x^{(k)}$ is orthogonal to H_i by definition, and thus parallel to a_{i*}. Thus, the distance $\|\text{proj}_{a_{i*}} x^{(k+1)} - \text{proj}_{a_{i*}} x^{(k)}\|$ is equivalent to $\|x^{(k+1)} - x^{(k)}\|$. Furthermore, the vectors $\text{proj}_{a_{i*}} x^{(k+1)} - \text{proj}_{a_{i*}} x^{(k)}$ and $x^{(k+1)} - x^{(k)}$ both point in the same direction since they are parallel and the "heads" of the vectors both lie in H_i. In other words, since they have the same direction and magnitude, they are the same vector, so

$$x^{(k+1)} - x^{(k)} = \text{proj}_{a_{i*}} x^{(k+1)} - \text{proj}_{a_{i*}} x^{(k)}. \tag{3.8}$$

Therefore,

$$x^{(k+1)} = x^{(k)} + (x^{(k+1)} - x^{(k)}) = x^{(k)} + a_{i*} \frac{b_i - a_{i*}^T x^{(k)}}{\|a_{i*}\|^2} \tag{3.9}$$

which is equivalent to ART as presented in (3.3).

We will show several ART iterations on the sample linear systems in order to clarify the projection process. Assume that $x^{(0)}$ is the zero vector. Then one full iteration of ART, which consists of three subiterations, is shown in Fig. 3.2. The circles in the figure denote the computed solution estimates and the dotted lines denote the projections onto the hyperplanes (or lines in this case). In order to demonstrate the differing convergence behavior for consistent and inconsistent systems, we include the results for five iterations of ART in Fig. 3.3. In this figure, we see that the iterates for the consistent case move continually closer to the true solution, while the iterates for the inconsistent case begin to oscillate in a limit cycle as noted by Herman et al [24].

3.2.2 ART as Gauss-Seidel

ART can additionally be viewed as the widely known Gauss-Seidel method applied to alternative normal equations of the system $Ax = b$. This connection between ART and

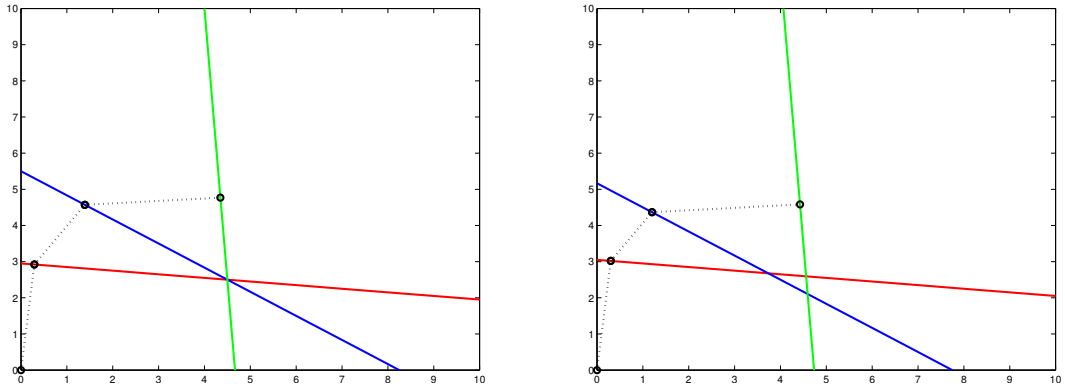

Figure 3.2: One full iteration of ART on the consistent (left) and inconsistent (right) sample systems.

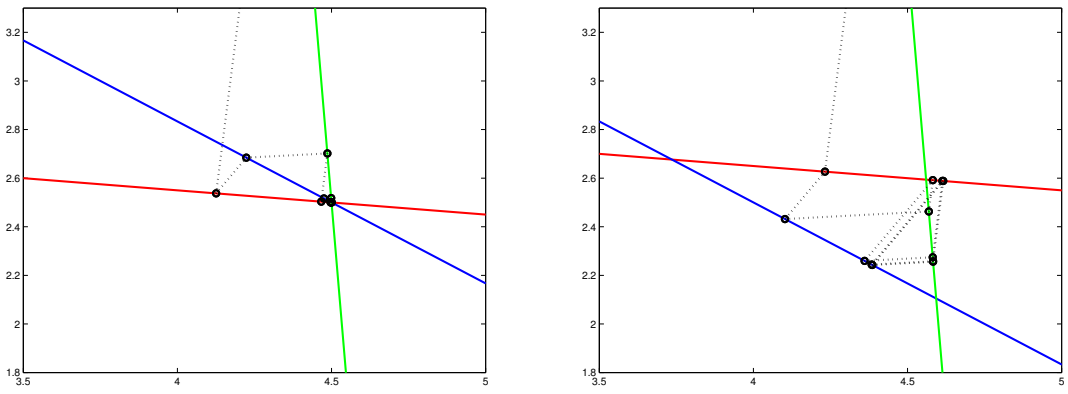

Figure 3.3: Five full iterations of ART on the consistent (left) and inconsistent (right) sample systems. The first several subiteration results are not visible as they are outside the axes boundaries.

Gauss-Seidel has been previously shown by Björck and Elfving [27] and observed by Barrett and Myers [28]. Our development is similar to that in [27], but includes more details.

Consider the least squares problem given by

$$\text{minimize } \|\mathrm{b} - \mathrm{Ax}\|_2. \tag{3.10}$$

Then the associated normal equations are given by [29]

$$A^T Ax = A^T b. \tag{3.11}$$

An alternative to the normal equations [30] solves the system

$$AA^T y = b \tag{3.12}$$

for y and computes the solution as $x = A^T y$. We will demonstrate that ART is equivalent to this alternative approach.

The Gauss-Seidel method for iteratively solving an $n \times n$ linear system $Ax = b$ is typically given as (see, e.g., [31])

$$x_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)}}{a_{ii}}, \quad i = 1, \ldots, n. \tag{3.13}$$

As written, Gauss-Seidel computes the ith component of the current iteration using elements $1, 2, \ldots, i-1$ of the current iteration and $i+1, i+2, \ldots, n$ of the previous iteration. Thus, given an initial solution estimate $x^{(0)}$, the estimate $x^{(1)}$ is computed after n component updates. However, we will instead view Gauss-Seidel as generating a complete solution estimate vector after each component update, but that new vector will differ from the previous vector only in the updated component. In other words, we will write the Gauss-Seidel update in (3.13) as

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)} \tag{3.14}$$

where

$$\Delta x^{(k)} = e_i (g_i^{(k)} - x_i^{(k)}). \tag{3.15}$$

In the latter equation, $i = k \mod n + 1$, e_i is the elementary unit column vector with a one in the ith position, $g_i^{(k)}$ is the Gauss-Seidel computed update for the ith component in the kth iteration, and $x_i^{(k)}$ is the ith element of the kth iteration of x. Since the previous iterate now contains the most recent values, the Gauss-Seidel update for the ith component of x is given by

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}. \tag{3.16}$$

Now consider applying Gauss-Seidel as given in (3.14) to solve for y in (3.12) where A is $m \times n$, y is $m \times 1$, b is $m \times 1$, and x is $n \times 1$. Then the (i, j) element of AA^T is $[AA^T]_{ij} = \sum_{h=1}^{n} a_{ih} a_{jh}$ and

$$g_i^{(k)} = \frac{b_i - \sum_{j \neq i} (\sum_{j=1}^{n} a_{ih} a_{jh}) y_j^{(k)}}{\sum_{h=1}^{n} a_{ih}^2}. \tag{3.17}$$

Substituting $g_i^{(k)}$ into (3.15), solving now for y instead of x, yields

$$\begin{aligned}
\Delta y^{(k)} &= e_i \left(\frac{b_i - \sum_{j \neq i} (\sum_{h=1}^{n} a_{ih} a_{jh}) y_j^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} - y_i^{(k)} \right) && (3.18) \\
&= e_i \left(\frac{b_i - \sum_{j \neq i} (\sum_{h=1}^{n} a_{ih} a_{jh}) y_j^{(k)} - \sum_{h=1}^{n} a_{ih}^2 y_i^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} \right) && (3.19) \\
&= e_i \left(\frac{b_i - \sum_{j=1}^{m} (\sum_{h=1}^{n} a_{ih} a_{jh} y_j^{(k)})}{\sum_{h=1}^{n} a_{ih}^2} \right). && (3.20)
\end{aligned}$$

Rearranging the summations and converting them to matrix-vector notation, we have

$$\begin{aligned}
\Delta y^{(k)} &= e_i \left(\frac{b_i - \sum_{h=1}^{n} a_{ih} \sum_{j \neq i} a_{jh} y_j^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} \right) && (3.21) \\
&= e_i \left(\frac{b_i - \sum_{h=1}^{n} a_{ih} [A^T y^{(k)}]_h}{\sum_{h=1}^{n} a_{ih}^2} \right) && (3.22) \\
&= e_i \left(\frac{b_i - a_{i*}^T A^T y^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} \right) && (3.23)
\end{aligned}$$

where $[A^T y^{(k)}]_h$ is the hth component of the matrix-vector multiplication and a_{i*}^T is the ith row of A. Since $x = A^T y$ by assumption, we have $x^{(k)} = A^T y^{(k)}$, so

$$\Delta y^{(k)} = e_i \left(\frac{b_i - a_{i*}^T x^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} \right). \tag{3.24}$$

Therefore,

$$x^{(k+1)} = A^T y^{(k+1)} = A^T (y^{(k)} + \Delta y^{(k)}) = x^{(k)} + A^T \Delta y^{(k)} \tag{3.25}$$

and

$$A^T \Delta y^{(k)} = A^T e_i \left(\frac{b_i - a_{i*}^T x^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} \right) = a_{i*} \left(\frac{b_i - a_{i*}^T x^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} \right). \tag{3.26}$$

Finally,

$$x^{(k+1)} = x^{(k)} + A^T \Delta y^{(k)} = x^{(k)} + a_{i*} \left(\frac{b_i - a_{i*}^T x^{(k)}}{\sum_{h=1}^{n} a_{ih}^2} \right) \tag{3.27}$$

which is equivalent to the vectorized form of ART given in (3.3).

Therefore, ART is equivalent to the Gauss-Seidel method applied to the alternative normal equation formulas given in (3.12) and is thus a reincarnation of a classical algebraic technique.

3.2.3 ART as Gauss-Seidel in matrix form

We can also write the ART iteration in full matrix form. The Gauss-Seidel iteration applied to a square linear system $Ax = b$ is

$$x^{(q+1)} = (D + L)^{-1}(b - Ux^{(q)}) = x^{(q)} + (D + L)^{-1}(b - Ax^{(q)}) \tag{3.28}$$

where D contains the diagonal elements of A, L contains the elements of A below the diagonal, and U contains the elements of A above the diagonal so that $A = D + L + U$. If we apply Gauss-Seidel to the alternative normal equations given by $AA^T y = b$ to solve for y, we have

$$y^{(q+1)} = y^{(q)} + (D + L)^{-1}(b - AA^T y^{(q)}) \tag{3.29}$$

where D and L are now the diagonal and lower components of AA^T, respectively. Since $x = A^T y$, we have

$$x^{(q+1)} = A^T y^{(q+1)} = x^{(q)} + A^T (D + L)^{-1}(b - Ax^{(q)}). \tag{3.30}$$

This matrix form of ART computes only the full iterations, not the subiterations as computed by the more typical presentations of ART. In other words, if the total subiteration index is k, such as in (3.3), then $q = km$. In Section 3.3.2, we will introduce the concept of matrix splitting as a general approach to solving linear systems that includes as special cases the Jacobi and Gauss-Seidel iterations.

3.2.4 Additional Notes

Other researchers have recently investigated efficient implementations of ART. For example, Mueller has presented fast implementations for the projector-backprojector pair [9] as well as ART and SART reconstructions driven by commodity graphics hardware [32]. In this work, we implement the reconstruction software on a cluster of commodity PCs. However, since ART only uses one system matrix row per image update, it does not easily benefit from a parallel implementation. Thus, rather than implementing ART in our reconstruction framework, we implement the simultaneous analogues of ART.

3.3 Simultaneous Iterative Reconstruction Algorithm (SIRT)

Introduced in 1972 by Gilbert [33], the simultaneous iterative reconstruction technique (SIRT) only updates the image after considering the entire matrix. In our notation, the matrix-vector version of SIRT is

$$x^{(k+1)} = x^{(k)} + CA^T R(b - Ax^{(k)}) \tag{3.31}$$

where C and R are diagonal matrices containing the inverses of the column and row sums of A, or $c_{jj} = 1/\sum_{h=1}^{m} a_{hj}$ and $r_{ii} = 1/\sum_{h=1}^{n} a_{ih}$. Although SIRT is typically presented as a natural extension of ART, there are some important differences. Most importantly, if the matrix A has full column rank and the system is inconsistent, then SIRT converges to a weighted least squares solution. As we will see shortly, the weighting in the SIRT algorithm is related to the row sums of A.

3.3.1 SIRT as the generalized Landweber iteration

Landweber proposed an iteration method in 1951 [34] to solve Fredholm integral equations of the first kind, which can in turn be used to solve linear systems. The Landweber iteration applied to the system $Ax = b$ is

$$x^{(k+1)} = x^{(k)} + A^T(b - Ax^{(k)}). \tag{3.32}$$

The iteration scheme was further investigated by Strand [35], who also proposed a generalized Landweber iteration defined as

$$x^{(k+1)} = x^{(k)} + DA^T(b - Ax^{(k)}) \tag{3.33}$$

where Strand refers to the matrix D as a "shaping" operator. We will now show that SIRT is an instance of the generalized Landweber iteration with $D = C$.

First consider the matrix R from SIRT, which is diagonal with positive diagonal elements, and thus symmetric positive definite. Since symmetric positive definite matrices have associated matrix square roots [29, p. 149], we can define $R^{1/2}$, which in this case is simply a diagonal matrix containing the square roots of the elements of R. Now consider the linear system

$$R^{1/2}Ax = R^{1/2}b. \tag{3.34}$$

The generalized Landweber iteration for this preconditioned linear system is

$$x^{(k+1)} = x^{(k)} + DA^TR^{1/2}(R^{1/2}b - R^{1/2}Ax^{(k)}) = x^{(k)} + DA^TR(b - Ax^{(k)}) \tag{3.35}$$

which is equivalent to SIRT with $D = C$.

3.3.2 SIRT as a matrix splitting

Matrix splitting is a standard approach for solving linear systems that generates a large family of algorithms, including the Jacobi and Gauss-Seidel iterations. We will show in the following that SIRT, as well as the Landweber and generalized Landweber algorithms, can be viewed as specific matrix splittings.

Assume we have a linear system $Ax = b$ with A an $n \times n$ nonsingular matrix. Consider $n \times n$ matrices M and N such that $A = M - N$. If M is nonsingular, then the iteration defined by

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b \tag{3.36}$$

converges to $x = A^{-1}b$ if $\rho(M^{-1}N) < 1$ where the spectral radius ρ is the largest eigenvalue magnitude of A, or $\rho(A) = \max\{|\lambda| : \lambda \in \Lambda(A)\}$ with $\Lambda(A)$ representing the set of eigenvalues of A [29, Theorem 10.1.1].

Now consider the matrix splitting applied to the normal equations for an overdetermined system $Ax = b$ given by $A^TA = I - (I - A^TA)$. This splitting yields the iteration

$$x^{(k+1)} = (I - A^TA)x^{(k)} + A^Tb = x^{(k)} + A^T(b - Ax^{(k)}) \tag{3.37}$$

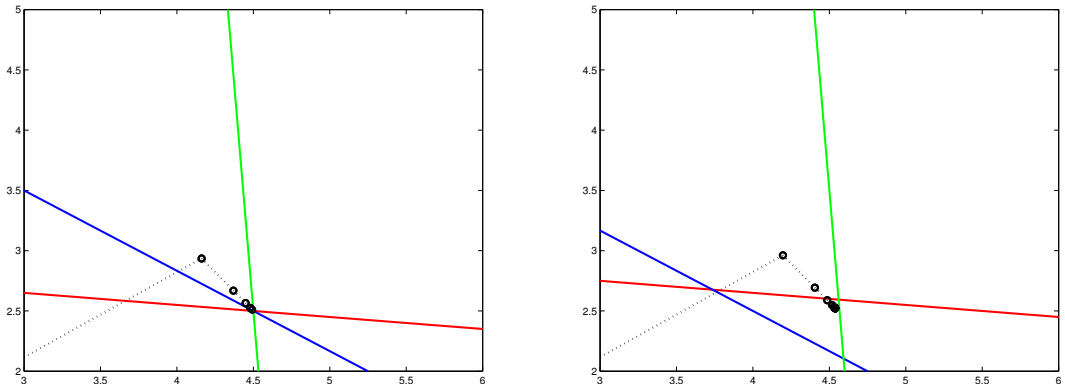

Figure 3.4: Five iterations of the SIRT algorithm on the consistent (left) and inconsistent (right) sample systems.

which is the Landweber iteration applied to the original linear system. Furthermore, if we instead consider a preconditioned version of the normal equations given by

$$DA^TAx = DA^Tb \tag{3.38}$$

for some nonsingular preconditioning matrix D, then the iteration associated with the splitting $DA^TA = I - (I - DA^TA)$ is

$$x^{(k+1)} = (I - DA^TA)x^{(k)} + DA^Tb = x^{(k)} + DA^T(b - Ax^{(k)}). \tag{3.39}$$

This latter equation is the generalized Landweber algorithm.

It should now be clear that SIRT is also an iteration resulting from a matrix splitting. In particular, consider the preconditioned normal equations associated with the weighted least squares problem:

$$CA^TRAx = CA^TRb. \tag{3.40}$$

Then the matrix splitting $CA^TRA = I - (I - CA^TRA)$ yields the SIRT algorithm:

$$x^{(k+1)} = (I - CA^TRA)x^{(k)} + CA^TRb = x^{(k)} + CA^TR(b - Ax^{(k)}). \tag{3.41}$$

We show the results of applying five iterations of SIRT to the consistent and inconsistent sample problem in Fig. 3.4. Notice that SIRT does not express the oscillatory behavior associated with ART on the inconsistent system. Furthermore, in the inconsistent case, SIRT converges to

$$x^* \approx \begin{bmatrix} 4.537222 \\ 2.523572 \end{bmatrix}. \tag{3.42}$$

which is $(A^TRA)^{-1}A^TRb$, so for the sample problem SIRT does in fact solve the weighted least squares problem.

As noted before, the Jacobi and Gauss-Seidel iterations are also specific matrix splittings. In the next section, we note the similarities and differences between SIRT and the Jacobi

iteration. We will also analyze the convergence of SIRT based on the matrix splitting formulation in Section 3.5 and modify the algorithm to create the parallel SIRT algorithm.

3.3.3 SIRT similarities to the Jacobi iteration

Since we previously established that ART is simply Gauss-Seidel applied to the alternative set of normal equations, and SIRT is considered the simultaneous version of ART, it seems reasonable to expect SIRT to be the Jacobi iteration applied to some system of equations. However, this is not the case. That said, we will show that SIRT is very similar to the Jacobi iteration applied to a weighted version of the normal equations.

The Jacobi iteration applied to a square linear system $Ax = b$ is

$$\begin{aligned} x^{(k+1)} &= (I - D^{-1}A)x^{(k)} + D^{-1}b \quad (3.43) \\ &= x^{(k)} + D^{-1}(b - Ax^{(k)}) \quad (3.44) \end{aligned}$$

where D is the diagonal of A. Define $L_R = \|b - Ax\|_R^2$ to be the weighted least squares functional with R defined as before. Then the associated normal equations are given by

$$A^T RAx = A^T Rb \quad (3.45)$$

and the Jacobi iteration applied to these normal equations is

$$x^{(k+1)} = x^{(k)} + D^{-1}A^T R(b - Ax^{(k)}) \quad (3.46)$$

where D is now the diagonal of $A^T RA$. The only difference between this Jacobi iteration and SIRT is the D^{-1} term of the Jacobi iteration in place of the C term of the SIRT iteration. Both of these matrices are $n \times n$ diagonal matrices and the diagonal elements are given by

$$\begin{aligned} d_{jj}^{-1} &= 1/\left(\sum_{h=1}^{m} \frac{a_{hj}^2}{\sum_{p=1}^{n} a_{hp}}\right) = 1/\left(\sum_{h=1}^{m} a_{hj} w_{hj}\right) \quad (3.47) \\ c_{jj} &= 1/\left(\sum_{h=1}^{m} a_{hj}\right) \quad (3.48) \end{aligned}$$

where $w_{hj} = a_{hj}/\sum_{p=1}^{n} a_{hp}$. Thus, D^{-1} contains weighted column sums, and the weights are all less than or equal to one since they are normalized contributions of a given matrix element to its row. Therefore, $w_{hj} \leq 1$ for all h, so $c_{jj} \leq d_{jj}^{-1}$ for all j. If we interpret the C and D^{-1} matrices as step sizes for the corrections to the current estimates, then we see that the Jacobi step sizes are always greater than or equal to the SIRT step sizes. We depict five iterations of the Jacobi iteration on the consistent and inconsistent sample systems in Fig. 3.5. Note that the update steps for the Jacobi algorithm are in fact larger than for the SIRT iterations shown in Fig. 3.4. We do not implement the Jacobi iteration because it would require access to the full system matrix elements and as described in Section 2.2.1, we can only easily access the partial system matrix elements when using trilinear interpolation as the system model.

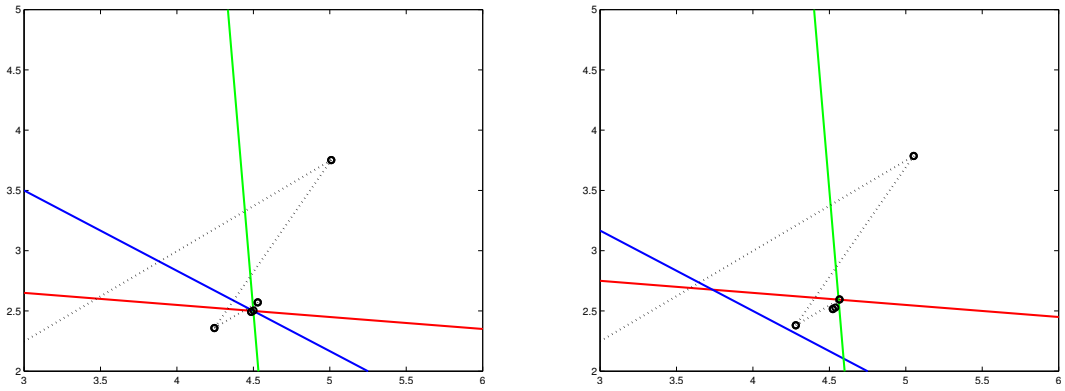

Figure 3.5: Five iterations of the Jacobi algorithm on the consistent (left) and inconsistent (right) sample systems.

3.3.4 SIRT as Richardson's iteration

Richardson's iteration, which is also used to solve square linear systems given by $Ax = b$, is given by [36]

$$x^{(k+1)} = x^{(k)} + \beta(b - Ax^{(k)}) \tag{3.49}$$

where β is a scalar relaxation parameter. If we let $\beta = 1$ and apply Richardson's iteration to the system $CA^TRAx = CA^TRb$, then we get the SIRT iteration. We can also view Richardson's iteration as applying a matrix splitting given by $\beta A = I - (I - \beta A)$ to the system $\beta Ax = \beta b$ where β is a scalar preconditioner.

3.4 Simultaneous Algebraic Reconstruction Technique (SART)

Andersen and Kak introduced the simultaneous algebraic reconstruction technique (SART) in 1984 [16] as an additional algebraic reconstruction technique. SART is basically a hybrid between ART and SIRT. With ART, only one matrix row is used per image update, while SIRT uses all matrix rows during each update. The SART algorithm, on the other hand, divides the system matrix rows in terms of projection view angles and performs updates based on the matrix rows corresponding to a single view angle using the SIRT algorithm. In other words, one update will be performed using the linear system that results from considering only the system matrix rows corresponding to a given projection view. It will be evident after we introduce ordered subsets in Section 4.2 that SART is an ordered subsets version of SIRT. Furthermore, note that some authors use the name SART for the algorithm that we refer to as SIRT.

3.5 Parallel SIRT (PSIRT)

In Section 3.3, we demonstrated that the SIRT iteration can be viewed as resulting from a matrix splitting of CA^TRA and stated a theorem related to matrix splitting. In this section, we employ that theorem and establish that SIRT converges to a unique solution given a set of

stated assumptions. We will then modify the SIRT algorithm in such a way that maintains convergence, but will ultimately be more efficient in a parallelized implementation.

Assume that A is an $m \times n$ nonnegative matrix with $m > n$ and $\text{rank}(A) = n$. Since all of the system models discussed in Chapter 2 generate nonnegative system matrix elements, A is nonnegative in our case. Furthermore, A is overdetermined, or $m > n$, if there is enough projection data, which is generally under our control. However, it is more difficult to determine if the system matrix A has full column rank, or in other words, if $\text{rank}(A) = n$. We proceed with the assumption that A does indeed have full column rank.

Consider the following linear system discussed in Section 3.3:

$$CA^TRAx = CA^TRb. \tag{3.50}$$

Since R and C are both diagonal matrices with positive entries corresponding to the inverse row and column sums of the system matrix A, respectively, both R and C are symmetric positive definite (SPD). Then since R is positive definite and A has rank n, A^TRA is a positive definite $n \times n$ matrix [29, Theorem 4.2.1]. Furthermore, $(A^TRA)^T = A^TRA$, so A^TRA is SPD. Finally, since both C and A^TRA are SPD, CA^TRA is an $n \times n$ nonsingular matrix, so the system $CA^TRAx = CA^TRb$ is subject to matrix splitting.

As demonstrated above, the matrix splitting given by $CA^TRA = I - (I - CA^TRA)$ yields a rewritten form of the SIRT iteration:

$$x^{(k+1)} = (I - CA^TRA)x^{(k)} + CA^TRb. \tag{3.51}$$

Thus, if $\rho(I - CA^TRA) < 1$, then the SIRT iteration converges to the unique solution

$$x^* = (CA^TRA)^{-1}CA^TRb = (A^TRA)^{-1}A^TRb. \tag{3.52}$$

Note that the matrix C does not affect the solution, but will impact convergence since it is part of the iteration matrix.

We will first use properties of induced matrix norms to bound the spectral radius of CA^TRA and then extend the result to the iteration matrix $I - CA^TRA$. The vector ∞-norm induces the matrix ∞-norm given by $\|A\|_\infty = \max_{1 \le i \le m} \|a_i^T\|_1$, which is the maximum row sum of A. Since R is a diagonal matrix containing the inverse row sums of A, or $r_{ii} = 1/\sum_{j=1}^n a_{ij}$, each row of RA sums to one. Similarly, since C contains the inverse column sums of A, and the rows of A^T are the columns of A, each row of CA^T sums to one. Furthermore, for any induced matrix norm, $\rho(A) \le \|A\|$ and $\|AB\| \le \|A\|\|B\|$. Thus,

$$\rho(CA^TRA) \le \|CA^TRA\| \le \|CA^T\|\|RA\| = 1. \tag{3.53}$$

In addition, since C is positive definite and A^TRA is symmetric, the matrix CA^TRA has the same number of positive, negative, and zero eigenvalues as A^TRA [37, Theorem 7.6.3]. Thus, since A^TRA is SPD, CA^TRA has all positive eigenvalues by the referenced theorem, and all of the eigenvalues are less than or equal to one by (3.53).

Assume that CA^TRA has eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and let the associated eigenvectors be $x_1, x_2, \ldots, x_n$. Then $Ax_i = \lambda_i x_i$ and $(I - A)x_i = x_i - Ax_i = (1 - \lambda_i)x_i$, so the eigenvalues associated with the iteration matrix $I - CA^TRA$ are given by $1 - \lambda_1, 1 - \lambda_2, \ldots, 1 - \lambda_n$.

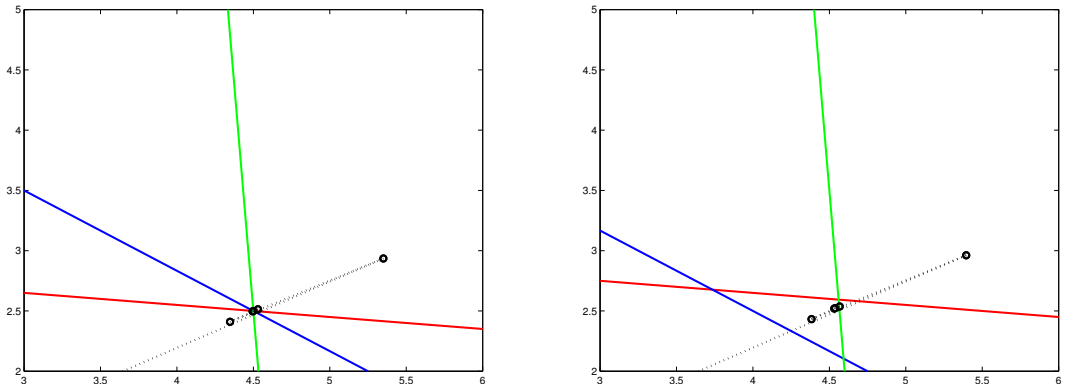

Figure 3.6: Five iterations of the PSIRT algorithm on the consistent (left) and inconsistent (right) sample systems.

Hence, the eigenvalues λ_i of $I - CA^TRA$ all satisfy $0 \leq \lambda_i < 1$, so

$$\rho(I - CA^TRA) < 1. \tag{3.54}$$

Therefore, the SIRT iteration converges to the solution described in (3.52).

For reasons that will become apparent in Chapter 4, it would be beneficial to replace the inverse column sum matrix C by a scalar, primarily due to interprocessor communications required in a parallelized implementation. Recall that C has no effect on the solution computed in (3.52), so replacing C is reasonable if convergence is maintained. We will replace C by a scalar α to form the parallel SIRT iteration (PSIRT):

$$x^{(k+1)} = x^{(k)} + \alpha A^TR(b - Ax^{(k)}). \tag{3.55}$$

Define α as the inverse of the maximum column sum of A, or

$$\alpha = \frac{1}{\max_j \sum_{i=1}^{n} a_{ij}}. \tag{3.56}$$

The iteration matrix now becomes $I - \alpha A^TRA$ and we have $\|\alpha A^T\|_\infty = 1$ since all of the rows of αA^T sum to less than or equal to one with at least one row summing to exactly one. Thus, (3.53) still holds with C replaced by α, and the rest of the analysis remains the same, although αA^TRA is now SPD so the analysis is simplified. Thus, the parallel SIRT algorithm converges to

$$x^* = (\alpha A^TRA)^{-1}\alpha A^TRb = (A^TRA)^{-1}A^TRb, \tag{3.57}$$

which is the same solution given for SIRT in (3.52). Therefore, SIRT and PSIRT converge to the same solution. In Fig. 3.6, we show the first five iterations of the PSIRT iteration on the consistent and inconsistent sample systems. We will demonstrate the advantages of PSIRT in Chapter 5.

Chapter 4

Implementation Issues

Due to the size of the linear system, special care must be taken when implementing an iterative image reconstruction algorithm. We employ many methods of reducing the computation and storage burdens associated with the reconstruction algorithms. The first method employed is cluster computing, which significantly reduces the computational runtime and somewhat reduces the memory requirements, but also introduces interprocessor communication. We then apply ordered subsets, which also significantly reduces the computation time, but increases the relative cost of the communication. The combination of parallel computing and ordered subsets makes the computation manageable, but does not significantly address memory concerns and also introduces communication costs. In order to address the latter issues, we restrict the reconstruction to its image support region. This significantly reduces memory consumption and communication costs, and also somewhat reduces computation costs. In addition, we employ a heuristic data-driven technique called focus of attention that reduces the amount of considered data in the image and projection spaces, which in turn reduces memory consumption, communication, and computation costs. Finally, we implement parallel SIRT (PSIRT) to further reduce communication.

4.1 Parallel Computing

We have developed an image reconstruction framework that runs on clusters of commodity PCs. The primary cluster used consists of sixty-four nodes. Each node has dual-processors and a Myrinet interconnection; this cluster is more thoroughly described in Appendix B.1. We employ POSIX threads for parallelization on each node and an adaptation of the MPICH implementation of MPI [38,39] for communication between nodes. While we could avoid the added complexity of developing the reconstruction software to use multiple threads by instead running several processes per node (e.g., one process per processor), this would require a great deal of duplication in memory usage. As we will demonstrate later, memory constraints are a major concern for iterative image reconstruction, and running two processes on a single node would approximately double the amount of memory per node required to perform the reconstructions. Thus, it is important to use multiple threads per node rather than multiple processes in order to conserve memory. We will first discuss the method of distributing the reconstruction to multiple nodes and then explore the effectiveness of the load balancing strategy.

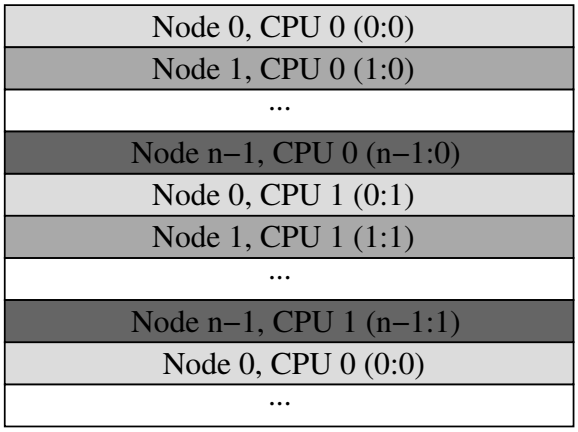

Figure 4.1: Depiction of the workload assignment to each of two processors on each of n nodes. In this figure $(i : j)$ refers to node i and processor j.

4.1.1 Problem Distribution

Recall that the ultimate goal of the iterative reconstruction algorithms is to find an approximate solution to the linear system $Ax = b$. Due to the size of the system matrix A, the matrix elements cannot reasonably be stored and thus must be computed on demand. There are two obvious approaches to computing the elements: by row or by column. These two approaches correspond to ray-driven and voxel-driven methods, which we introduced in Chapter 2. As discussed there, we focus on ray-driven, and thus row based, approaches in this work.

When partitioning the problem in terms of rows, we must communicate the results of backprojections, which are image sized arrays. On the other hand, when partitioning the problem in terms of columns, and thus using a voxel-driven system model, we must communicate the results of forward projections, which are projection data sized. Since we typically expect to have an overdetermined linear system, and thus more rows than columns, the ray-driven approach involves communicating less data than the voxel-driven approach.

We divide the rows among the cluster nodes by treating a detector column, which we will henceforth refer to as a detector group to avoid confusion with system matrix columns, as an atomic computational unit. These detector groups in turn correspond to some number of system matrix rows, one per detector element. We then distribute these groups to the N nodes in a round robin fashion: node 0 manages columns 0, N, $2N$, ..., node 1 manages groups 1, $N+1$, $2N+1$, ..., etc. The groups are then further divided among each available processor on each node, again in a round robin fashion.

We give a visual depiction of the workload distribution in Fig. 4.1. Each block in the figure corresponds to a detector group, which is assigned to some processor on some node. We assume that there are only two processors per node, as is the case on our computing resources, but this approach generalizes easily to more processors. With n nodes, node i $(0 \leq i \leq n-1)$ manages detector groups $nk + i$ where $0 \leq k < K$ and K is the total number of detector groups times the number of projections.

There are two major computational operations involving the system matrix that must be implemented: forward projections and backprojections. These operations correspond

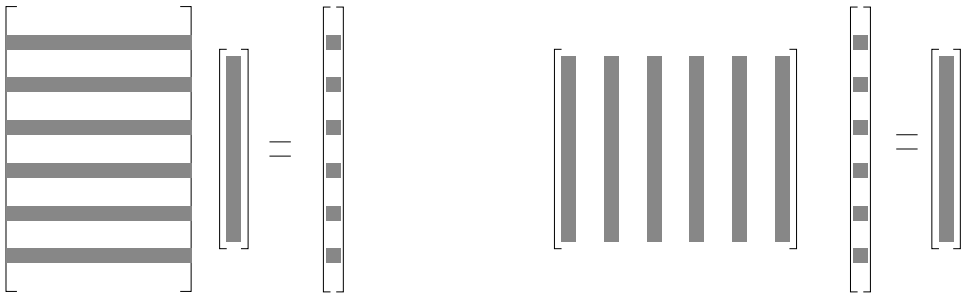

Figure 4.2: Visual depiction of the forward projection (left) and backprojection (right) processes for a single node. The node computes specified system matrix rows which then correspond to certain columns during the backprojection.

to the matrix operations Ax and $A^T b$, respectively. For most algorithms, each of these operations is used at least once per iteration. For example, recall the SIRT algorithm

$$x^{(k+1)} = x^{(k)} + CA^T R(b - Ax^{(k)}), \tag{4.1}$$

which superficially includes one forward projection and one backprojection. We demonstrate later that the R and C matrices actually constitute an additional forward and backprojection. Thus, if we compute the system matrix elements separately for the forward and backprojections, we will compute each element at least twice during each iteration. Fortunately, once a node calculates a system matrix row, we can use the calculated elements for all projection operations, so we only generate each system matrix element once per iteration. We will now describe how this is accomplished.

First consider the elements produced from a forward and backprojection. The ith component of Ax and the jth component of $A^T b$ are given by

$$[Ax]_i = \sum_{j=1}^{n} a_{ij} x_j \tag{4.2}$$

$$[A^T b]_j = \sum_{i=1}^{m} a_{ij} b_i. \tag{4.3}$$

Thus, since each node stores the full image x, the ith component of Ax can be fully computed on the node to which the ith system matrix row is assigned. Furthermore, this value will not be needed on any other node. However, the jth component of $A^T b$ cannot be fully computed without input from every system matrix row. Thus, each node only computes partial sums for the backprojection values and MPI communication is required to calculate the full back projection values. Figure 4.2 depicts the forward and backprojection processes for the data stored on a given node.

More precisely, assume that the system matrix row indices are partitioned into the N sets $B_0, B_1, \ldots, B_{N-1}$, which we then assign to the N nodes in the natural order. Then the

partial sum of the jth component of $A^T b$ computed on node 0 is

$$[A^T b]_j^{(0)} = \sum_{i \in B_0} a_{ij} b_i \tag{4.4}$$

where the superscript on $[A^T b]_j$ denotes the node. Thus, the value $[A^T b]_j$ must be computed by summing all of the partial values, i.e.

$$[A^T b]_j = \sum_{h=0}^{N-1} [A^T b]_j^{(h)}. \tag{4.5}$$

We borrow the terminology used by MPI and refer to these summations across nodes as global reductions. As we will see later, these global reductions can consume a significant portion of the total reconstruction time.

Furthermore, the column sums computed in the SIRT algorithm are the result of an implicit backprojection operation given by $A^T 1$ where 1 denotes the $m \times 1$ column vector containing all ones. Of course, in the implementation we simply sum the values rather than multiplying each element by one and then summing the results, but it is sometimes useful to conceptually classify the column sum operation as a backprojection. Thus, communication is also required to aggregate the partial column sums. The row sums are also the result of an implicit forward projection given by $A1$ where 1 is now the $n \times 1$ column vector of ones. However, as with the other forward projection, no communication is required because the necessary row sums are fully computed on each node.

As mentioned above, the system matrix elements are only computed once per iteration, although they are used for several operations. Although Li et al. [40] took this approach, and it is likely used by others as well, it also seems popular to calculate the system matrix elements separately for each projection operation. For example, Zeng and Gullberg discuss using an "unmatched" projector and backprojector pair [41]. The motivation is that a simpler system model can be used for the backprojection than for the forward projection, thus saving time spent in the projectors. However, if the calculated forward projection coefficients are reused in the backprojector, then there is no need to use a simpler backprojector model since using the same values requires no additional computation. Note that an additional reason given for the unmatched projector pairs is to remove reconstruction artifacts that are present in a matched projector pair reconstruction, which may justify the approach despite the computational disadvantages. As presented above, the forward projection operations are simple to compute given a system matrix row because that row and the image that is stored on each node are all that are needed to compute the corresponding forward projection element. However, the backprojection operation is less direct since it requires a column of the system matrix, but only rows are available. Thus, when a system matrix row is available, we iterate through the row adding contributions to an image sized array as shown in Algorithm 4.1. This is a simple concept, but must be implemented carefully in a multi-threaded application due to the possibility of a race condition.

Algorithm 4.1 Pseudocode for the backprojection operation.
The backprojection values accumulate in the image sized vector v.

1: **for all** nonzero columns j in row i **do**
2: $\quad v_j = v_j + a_{ij} b_i$
3: **end for**

4.1.2 Backprojection Race Condition

While the approach in Algorithm 4.1 works straightforwardly when using one thread, the inclusion of multiple threads introduces the possibility for a race condition. In particular, if two (or more) threads are executing the loop in Algorithm 4.1 simultaneously using different system matrix rows, then they may both update a voxel v_j at the same time. If this occurs, then one of the two updates may be lost and the computed sum will not be correct. Since the voxel space generally contains tens or hundreds of millions of voxels, and the projection rays in a circular orbit cone-beam geometry diverge from one another, this update problem seems unlikely to be a major concern. Thus, we will first quantify the issue to determine its severity and then consider possible solutions.

First we need to identify, on average, how often updates are actually lost for real reconstructions. On the surface, this seems to be a simple procedure: compute a backprojection first with a single thread, which cannot exhibit the behavior, and then with multiple threads and compare the results. However, if we use the trilinear interpolation model, then we will be comparing computed floating point numbers for equality, which is generally problematic. For example, just changing the order in which finite precision floating point numbers are added can change their sum even if no updates are lost. Instead, we first use a modified discrete volumetric approximation system model that returns the integral number of subvoxel centers contained in a given projection ray to quantify the effect of the race condition. We then reconsider the effect of the race condition on the trilinear interpolation based computations. By using an integral system model, we can avoid the complications arising from the limitations of finite precision floating point representations. We first compute the column sums of the corresponding system matrix, which as noted is a backprojection operation. By then summing these integral column sums and comparing the known correct results with the results obtained with multiple threads, we can get a sense of how often the race condition manifests itself. Note that summing the column sums is mathematically equivalent to summing all of the system matrix elements. The race condition, however, only manifests itself during the column sum computation since that is a backprojection and not during the final summation of the column sums, which is performed by a single thread.

We use twenty-seven subvoxels per voxel for the discrete volumetric approximation and the geometry corresponding to the Shepp-Logan phantom data set described in Appendix A.1. If only one point, or one subvoxel, is used per voxel, then the race condition is very unlikely to exhibit itself because that subvoxel can only be contained within one projection ray per projection and thus v_j can only be simultaneously updated from projections corresponding to different view angles. Furthermore, the threads are unlikely to be computing at such different rates that they are processing projection rays from different projection view angles for any substantial amount of time. The results of the column sum summations

Table 4.1: Total sum of the column sums computed using the modified discrete volumetric approximation system model. The correct value is 62,891,309,074, so the only reconstructions not computing the correct value were single node reconstructions.

Nodes	Recon 1	Recon 2	Recon 3
1	62,891,309,019	62,891,308,988	62,891,309,005
2	62,891,309,074	62,891,309,074	62,891,309,074
4	62,891,309,074	62,891,309,074	62,891,309,074
8	62,891,309,074	62,891,309,074	62,891,309,074
16	62,891,309,074	62,891,309,074	62,891,309,074
32	62,891,309,074	62,891,309,074	62,891,309,074

for varying numbers of nodes and multiple reconstructions per node configuration are given in Table 4.1. Each of these reconstructions used two threads per dual-processor node.

The correct column sum value for this system matrix, as computed by using a single thread, is 62,891,309,074. Thus, only the single node configuration exhibits the race condition for any of the three reconstructions. This is a result of the way in which the work is distributed. With N nodes, the detector groups managed by a given node are all separated by $N-1$ detector groups that are managed by other nodes. Furthermore, the detector groups managed by a given node are assigned in a round robin fashion to each thread, so in order for both threads to attempt updating a given voxel simultaneously, two projection rays separated by $N-1$ detector columns must interact with that voxel. This becomes increasingly unlikely as N increases, and we can see from Table 4.1 that it only seems to be a problem for single node reconstructions. With single node reconstructions, the same node manages all of the projection rays, so assigning the columns in a round robin fashion to the threads introduces the possibility of two threads computing system matrix values for projection rays corresponding to neighboring detector elements. When this occurs, the race condition is much more likely to manifest itself.

This could be easily prevented for single node reconstructions by assigning blocks of detector groups to each thread rather than using a round robin assignment, but since we do not perform large reconstructions on single nodes, this is not a concern. Also, even with the worst result, the relative error is less than .00000014%, which would likely not have a noticeable impact on the reconstruction. Thus, it appears that, at least with the volumetric approximation system model, the race condition is of no practical concern.

However, since we typically use trilinear interpolation for our reconstructions, we should also investigate the impact of the race condition on that system model. Trilinear interpolation assigns values to the neighbors of the voxels intersected by the projection ray, so it may tend to exhibit the race condition more often. Unlike the discrete volumetric case, the trilinear interpolation coefficients are floating point numbers, so comparing the error is a bit more difficult. In Table 4.2 we show the relative errors of the sum of the system matrix elements in parts per billion (ppb) when using the trilinear interpolation system model. For example, the relative percentage error for one node on the first run was 0.0000029% or 29 ppb. We compared the first fifteen decimal places of the results computed using a single thread and two threads to calculate these errors. IEEE single precision floating point

Table 4.2: Relative errors of the sum of the column sums in parts per billion when using the trilinear interpolation system model. Three reconstructions were performed with each node configuration.

Nodes	Recon 1	Recon 2	Recon 3
1	29 ppb	36 ppb	35 ppb
2	6.9 ppb	3.7 ppb	7.3 ppb
4	< 0.1 ppb	0 ppb	< 0.1 ppb
8	0 ppb	0 ppb	0 ppb
16	0 ppb	0 ppb	0 ppb
32	0 ppb	0 ppb	0 ppb

numbers only have an accuracy of approximately 1.19×10^{-7} [42], so the errors reported may not even be computationally significant. From this, we see that errors are introduced when using four or fewer nodes, as opposed to the discrete volumetric case in which errors only existed in the case of a single node reconstruction. Furthermore, the maximum relative error is higher in the trilinear interpolation case, but still exceptionally small.

While it is debatable if this problem even needs to be addressed due to the incredibly small errors generated, we will explore possible solutions and present their advantages and disadvantages. The most obvious solution is to avoid the situation altogether by only using a single thread, but this is not a reasonable solution as it fails to utilize both processors, and we already addressed the importance of using multiple threads as opposed to multiple processes in Section 4.1. Another approach would be to allocate one image sized array per thread, perform the backprojections into the respective arrays, and sum the results afterward. However, this would consume too much memory since the images can become quite large. An additional solution is the use of some locking mechanism to ensure that at most one thread can update the backprojection values at a time.

The locking approach, using for example the POSIX thread library, can be implemented at several levels of granularity. It is important to remember, however, that the lock and unlock operations add overhead and should thus be positioned carefully. One option is it create a lock for every voxel and require the backprojection threads to acquire the lock before updating the associated voxel and to release the lock afterward. This approach is far too fine grained because the locks themselves would consume considerable memory, and the run-time would be severely impacted due to the constant locking and unlocking. Instead, we use only a single lock and surround the backprojection loop with this lock. Since each thread also generates the system matrix rows, performs the forward projection operations, and any other operations dictated by the algorithm, the backprojection process is only a small part of that thread's computation. Thus, locking only that operation should not generate too much overhead. In our tests, these locks seem to increase the computation time associated with the SIRT algorithm by about five percent. Thus, if the errors created by not locking the backprojection operation are deemed to be a concern, then this approach provides a reasonable solution. We include the locking mechanism for all of the timing results presented in Chapter 5 in order to guarantee accurate results.

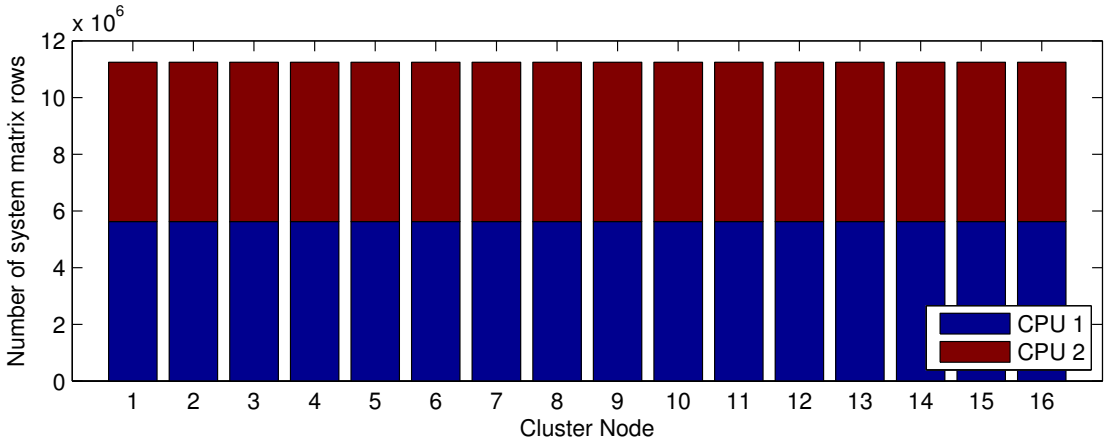

Figure 4.3: Bar chart depicting the number of system matrix rows assigned to each node for a sixteen node reconstruction. The number of rows assigned to each node is further divided among each of the two processors.

4.1.3 Load Balancing Analysis

One major concern when developing a parallelized algorithm implementation is the effectiveness of the load balancing strategy. If one processor is assigned significantly more work than the other processors, or for some reason takes longer to complete, then all processors will be forced to wait for the overloaded processor before completing the next communication. Since the iterative reconstruction algorithms require at least one communication during each iteration, uneven load balancing would significantly impact the final reconstruction time.

Thus, we quantify the effects of our current load balancing algorithm, which we described in Section 4.1.1. We perform the quantification in several ways: by calculating the number of system matrix rows per processor, the number of nonzero system matrix elements per processor, and the time spent awaiting synchronization for MPI communications. In this section, we will show results for such an analysis using a full reconstruction with all of the projection and image data considered. In subsequent sections, we will present techniques of identifying and focusing on parts of the data, which modifies the amount of data used and thus affects the load distribution. Afterward, we present an additional analysis including those modifications.

The number of system matrix rows per processor corresponding to a sixteen node reconstruction is shown in bar chart format in Fig. 4.3 and in tabular format in Table 4.3. Figure 4.3 suggests that the workload distribution, in terms of system matrix rows, is very even for each processor. In fact, it is difficult to see any disparity between the processor workloads in the bar chart. The number of system matrix rows per processor presented in Table 4.3 supports this notion. The maximum number of system matrix rows for any processor is $5,625,534$ on node ten, cpu one, and the minimum number is $5,624,275$ on node nine, cpu two. Thus, the processor with the highest workload only has 0.02% more rows than the processor with the smallest workload. Therefore, using this metric, the workload distribution approach is very effective.

Table 4.3: Number of system matrix rows assigned to each processor of each node for a sixteen node reconstruction. This data is the source for the bar chart in Fig. 4.3.

Node	CPU 1	CPU 2
1	5,624,683	5,624,642
2	5,625,227	5,625,088
3	5,624,865	5,624,460
4	5,625,406	5,624,909
5	5,624,967	5,624,358
6	5,625,485	5,624,830
7	5,625,027	5,624,298
8	5,625,528	5,624,787

Node	CPU 1	CPU 2
9	5,625,050	5,624,275
10	5,625,534	5,624,781
11	5,625,044	5,624,281
12	5,625,511	5,624,804
13	5,625,001	5,624,324
14	5,625,451	5,624,864
15	5,624,821	5,624,504
16	5,625,270	5,625,045

While the number of system matrix rows per processor is a good initial indicator of the amount of work required per processor, the number of nonzero system matrix elements calculated per processor may be a better indicator. In particular, different system matrix rows can have differing numbers of nonzero elements since the corresponding rays have different paths through the image space. As noted in Section 2.2, our implementation of the trilinear interpolation system model stores partial system matrix elements to avoid the overhead of coalescing the elements, so the number of partial elements produced is actually higher than the number of nonzero system matrix elements. We report the number of partial elements computed per node per processor in Table 4.4. This is an important measure because it indicates the number of elements inserted into the row data structures and subsequently used in the projection operations. This metric also seems to indicate an effective load balancing scheme. In addition, this table emphasizes the scale of these reconstructions: there are over half a trillion partial system matrix elements computed during a single iteration. If we instead consider the true number of nonzero system matrix elements rather than the partials, then there are approximately 12.3 billion per processor for a total of about 390 billion nonzero system matrix elements.

An additional metric that quantifies the effectiveness of a load distribution approach is the amount of time spent by each node waiting for one or more nodes to reach the same point in the computation. We calculate this time by having each node call `MPI_Barrier()` before attempting to communicate with other nodes and recording the amount of time spent in this function. `MPI_Barrier()` does not return until all nodes have made the function call, indicating that they are at the same point in the computation. This final metric used to quantify the workload distribution is also generally the most important.

Since the MPI communication cannot commence (or at least cannot complete) until all nodes have reached the same point in the computation, it will often be the case that some nodes will idly wait on other nodes to "catch up" in the computation. While this metric is ultimately the most important because it quantifies the effect of the workload distribution on the final reconstruction run-time, it is also the most difficult to accurately measure. The source of this difficulty is that the synchronization time, as measured by timing `MPI_Barrier()` calls as described previously, includes everything affecting the run-time, not just the workload distribution. For example, other processes running on a given node

Table 4.4: Number of partial system matrix elements computed per node per cpu for a sixteen node reconstruction.

Node	CPU 1	CPU 2
1	21,534,968,008	21,534,901,144
2	21,536,299,952	21,535,912,632
3	21,535,442,504	21,534,418,096
4	21,536,753,696	21,535,444,968
5	21,535,712,104	21,534,128,160
6	21,536,956,936	21,535,232,672
7	21,535,875,808	21,533,958,056
8	21,537,076,088	21,535,119,112

Node	CPU 1	CPU 2
9	21,535,957,120	21,533,882,464
10	21,537,126,264	21,535,086,720
11	21,535,975,344	21,533,885,048
12	21,537,081,040	21,535,142,960
13	21,535,856,208	21,534,021,144
14	21,536,907,064	21,535,336,464
15	21,535,390,056	21,534,497,720
16	21,536,411,912	21,535,816,184

may slow the computation on that node, which will manifest itself in higher synchronization times for other nodes and a slower reconstruction. Thus, successive reconstructions may result in different reported synchronization times.

Therefore, since it is difficult to accurately and consistently measure the synchronization times, it is important to instead search for trends in the data. We performed two reconstructions of three iterations each and measured the time spent per node per iteration awaiting synchronization. We then calculated the percent of the total time per iteration spent on synchronization and plotted these percents for the reconstructions in Figs. 4.4 and 4.5. The first iteration includes the computation and communication of the column sums, which are then stored for subsequent iterations, which may explain the disparity between the first iteration and the latter ones in both samples.

In the first reconstruction, nodes three, eight, and fifteen consistently have lower synchronization times, although on the first iteration node three has a synchronization time more in line with the other nodes. However, for the second reconstruction, nodes four and eleven have lower synchronization times than the other nodes. These two reconstructions were run in the same day on the same nodes (i.e., node n corresponds to the same physical node in each case). Therefore, it does not appear that the same nodes are slower from one reconstruction to the next, and thus it seems unlikely that the disparity is due to the workload assignment.

Since the synchronization times are consistently below five percent of the total iteration times, and often around two percent, we do not further pursue the potential causes of the differences from one reconstruction to the next. Furthermore, we will present several mechanisms in this chapter that we use to effectively decrease the amount of work required, so the workload distribution will also be affected. We will revisit the issue of load balancing in Section 4.6 after introducing these other techniques.

4.2 Ordered Subsets

A typical method for reducing the amount of computation required in order to achieve an acceptable reconstruction is the ordered subsets (OS) method, which was popularized by Hudson and Larkin [43]. The basic premise of the ordered subsets method is to compute

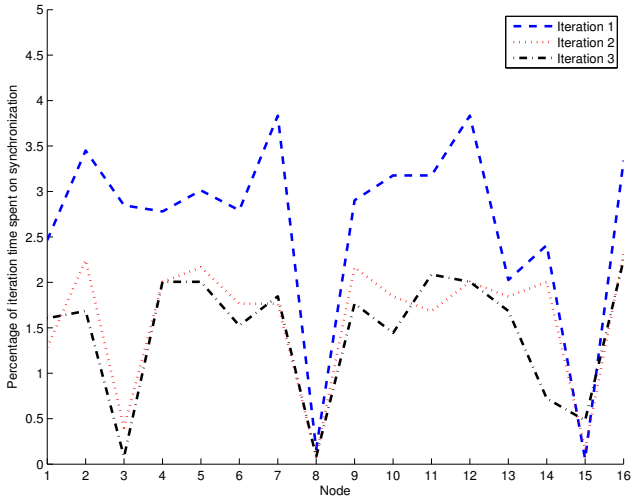

Figure 4.4: The percent of the total iteration time for each node spent on synchronization for the first reconstruction. Nodes three, eight, and fifteen consistently have low synchronization times.

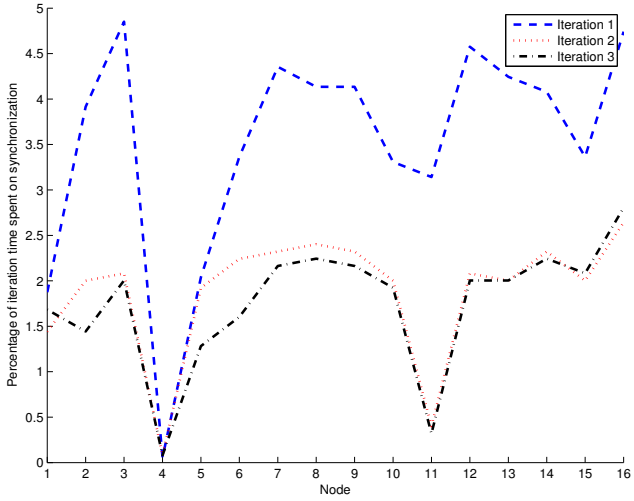

Figure 4.5: The percent of the total iteration time for each node spent on synchronization for the second reconstruction. Nodes four and eleven consistently have low synchronization times.

subiterations of the reconstruction algorithm using only a subset of the rows of the system matrix during each subiteration. Although not strictly required, we only consider the case where the system matrix rows are partitioned into some number of subsets such that each row is in exactly one subset. The algorithm then proceeds by computing one subiteration per subset of data and updating the image after each subiteration. After each subset has been used exactly once, a full iteration is complete. For example, ART uses only one system matrix row per subiteration and thus resembles an ordered subsets technique with each row forming a subset. Similarly, as we observed in Section 3.4, the SART iteration is an ordered subset version of SIRT with the system matrix rows for each projection view angle forming a subset.

The general rule for ordered subsets is that n subsets reduces the number of iterations required by a factor of n up to some point, after which increasing the number of subsets generates less significant gains. In previous work [44], we demonstrated that this linear gain diminishes after about 32 subsets in terms of reducing the error norm for some of our sample data. For example, when using 64 subsets the acceleration factor was approximately 47 rather than 64. We typically use 48 subsets for our reconstructions as it seems to be a reasonable compromise.

Since an OS implementation still uses each system matrix row once per iteration, the computation cost per iteration is approximately the same as a non OS implementation, but significantly fewer iterations are required. Thus, the total amount of computation is also significantly reduced. However, since the image is updated after each subiteration, an MPI communication is required after each subiteration. Furthermore, although the column sum vectors corresponding to a specific subset of rows are identical from iteration to iteration, their storage requires one image-sized vector per subset, which is too costly for high-resolution reconstructions. Thus, at least with SIRT, two MPI communications are required per subiteration for an OS implementation.

We denote an OS implementation of an algorithm by prefixing the algorithm name with OS and suffixing the name by the number of subsets. For example, OS-SIRT-48 is an OS version of SIRT with 48 subsets. Due to the more frequent communication and the reduced total computation, the OS implementations have a higher relative communication cost than a typical implementation. For example, let T_c be the total computation time per SIRT iteration per node and T_m be the total communication time required per MPI reduction of an image sized vector. Then the approximate time, T, for a SIRT reconstruction with k iterations is

$$T = kT_c + (k+1)T_m = k(T_c + T_m) + T_m \tag{4.6}$$

where the extra communication is required to compute the column sums, which can then be stored for subsequent iterations. On the other hand, an OS-SIRT implementation with N subsets and k/N iterations yields a total run-time of approximately

$$T = \frac{kT_c}{N} + 2kT_m = k(\frac{T_c}{N} + 2T_m). \tag{4.7}$$

The number of iterations, k/N, will never in practice drop below one, so this equation is not accurate when using too many subsets, although we already noted that additional subsets reach the point of diminishing returns after some point. However, it is clear from

the equations that as N increases the communication time begins to dominate and thus the relative cost of communication increases with increasing numbers of subsets.

We employ a projection based method for determining the subsets of system matrix rows to use. For example, if we use two subsets, then the even projections will form one subset and the odd projections will form the other subset. In general, if we have P projections and specify K subsets, then subset B_i, $i \in \{0, 1, \ldots, K-1\}$ contains projections $pK + i$ where $p \in \{0, i, \ldots, \lfloor \frac{P-i}{K} \rfloor\}$ and $\lfloor \cdot \rfloor$ denotes the floor operator.

4.3 Image Compression

We store the image as a contiguous array of single-precision floating point values. Thus, since the voxels have three coordinates, one corresponding to each dimension, there must be some mapping from these three coordinates to an index in this contiguous array. In the simplest case, we store values for all of the voxels, so one possible mapping is defined as

$$\text{index} = \text{i} + \text{jN}_\text{x} + \text{kN}_\text{x}\text{N}_\text{y}$$

where i, j, and k are the integral coordinates of the voxel in question and N_x and N_y are the number of voxels in the x and y directions. However, as we will present in the following sections, in many cases we do not actually need to store values for all of the voxels as some may be outside of our region of interest. In this case, since we still want a contiguous array for the image and do not want to waste memory on the voxels that do not need to be stored, such an indexing scheme will not work.

The indexing scheme is, however, very important for performance reasons because an index must be calculated for every nonzero system matrix element. As we demonstrated in Section 4.1.3, there may be tens of billions of system matrix elements calculated per processor per iteration, so if the indexing scheme is slow, then that will have a considerably negative impact on performance. We use the phrase compressed voxel index to denote the index in the contiguous memory array corresponding to a voxel with some indices i, j, and k.

An exhaustive mapping that simply translates a triplet of voxel indices into the corresponding compressed voxel index would require too much memory. Instead, we store the minimum and maximum voxel indices in the x direction for all fixed y and z voxel indices. Thus, we store values for all voxels between the minimum and maximum voxel indices. While this constrains the shape of our region of interest masks that will be presented shortly, it is not a major concern in practice due to the convexity of the support region defined by circular orbit cone-beam scanners as well as the nature of most of the scanned objects that we consider.

Denote the minimum and maximum voxel indices in the x direction as xMin[z][y] and xMax[z][y], respectively, for fixed y and z. We also store the compressed voxel index corresponding to each xMin[z][y] as xIndex[z][y]. Then, given some voxel indices i, j, and k, we calculate the compressed voxel index on-the-fly as

$$\text{compressed voxel index} = \text{xIndex[k][j]} + \text{i} - \text{xMin[k][j]}.$$

Thus, for each computed system matrix element, we compute its corresponding compressed voxel index for use in the reconstruction algorithm. When writing the image to disk, we must retranslate the compressed voxel indices into i, j, and k indices. We can accomplish this retranslation using the same data structures, although in this case performance is less important since the operation is typically performed only once per reconstruction.

4.4 Support Regions

In Section 4.2, we demonstrated that the relative cost of communication increases with an increasing number of subsets. Thus, it is important to reduce the time required for these communications. One seemingly simple method of reducing the communication time is to identify data that does not need to be communicated and avoid communicating it. Of course, identifying the unnecessary data can be difficult. In this section, we define the concept of support regions to identify such unnecessary data.

Consider a circular orbit cone-beam scanner. We define a point p to be within the fully supported region (FSR) prescribed by this scanner if for each projection view the ray connecting the x-ray source (which we assume to be a point source) and the point p is within the cone-beam and intersects the scanner detector. Therefore, the geometry of a scanner determines its FSR. We further define a voxel to be fully supported if it lies within the FSR; otherwise, it is not fully supported. When performing a reconstruction, we will only consider fully supported voxels. Thus, we will not store values or communicate values for non-fully supported voxels, and we will not calculate elements of the system matrix pertaining to non-fully supported voxels. However, we must first determine the shape and dimensions of the FSR, which will also be the determining factor for how we define our voxel space.

The FSR region is determined by three sets of boundaries: the x-ray cone-beam boundary, the vertical boundaries of the detector, and the horizontal boundaries of the detector. The cone-beam defines a spherical support region as it rotates through its circular orbit while the vertical detector boundaries define a cylindrical support region. On the other hand, the horizontal detector boundaries define a dual-cone shape with the bases of the cones given by the x-ray source orbit and the apexes of the cones lying in opposite directions along the axis of rotation. The intersection of these three shapes describes the FSR region. If the detector is fully illuminated, or in other words if every line segment joining a point on the detector to the x-ray source lies within the x-ray cone-beam, then the FSR is given by the intersection of the latter two shapes as that intersection will lie fully within the spherical region. In Fig. 4.6 we show an example of the cylindrical and dual-cone support regions and the resulting FSR, which resembles a pencil sharpened at both ends. Support regions for circular orbit cone-beam CT systems resulting after projection data rebinning were also discussed by Grass et al. [45, 46].

Now that we have established the nature of the FSR pertaining to our detector geometry, we must determine how to position the voxel space. The dimensions of the voxel space will also determine the size of the voxels for a fixed number of voxels in the x, y, and z directions. Our goal is to include the entire FSR within the voxel space while including as little of the non-FSR as possible. To do this, we specify the x and y dimensions of the voxel space such that they circumscribe the cylindrical portion of the FSR and the z dimension of the voxel

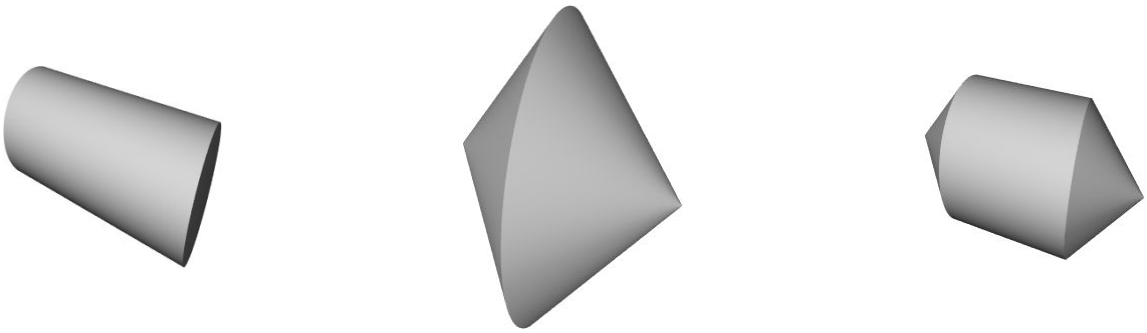

Figure 4.6: These shapes illustrate the support regions defined by the vertical detector boundaries (left), horizontal detector boundaries (center), and the intersection of the two (right).

space such that it extends to the apexes of the conic portions of the FSR. We depict these chosen dimensions in Fig. 4.7 where o is the origin, Y and Z are the detector dimensions in the y and z directions, respectively, r is the radius of the cylinder, θ is half of the cone-angle, d is the distance from the x-ray source to the center of rotation, and D is the distance from the x-ray source to the detector. Thus, we specify the x and y dimensions of the voxel space to be $2r$ to circumscribe the cylinder, or $x = y = 2r = 2d \sin\theta$. Furthermore, z is given as $z = (d/D)Z$.

By restricting the computing resources to the voxels contained within the support region, we reduce the amount of computation as well as the memory requirements and communication time. In Chapter 5 we will present the computational impact of restricting the reconstruction to the support region.

4.5 3D Focus of Attention

With the 3D focus of attention (FOA) algorithm, we further restrict the reconstruction region of interest by identifying voxels which may have significant attenuation coefficients, presumably due to containing some object, and ignoring the rest. Focus of attention is a heuristic data-driven approach to reducing the amount of data under consideration without appreciably affecting the quality of the reconstruction. While the scanner geometry determines the FSR region, the 3D FOA region is obtained by processing the projection data in order to identify the location of the object being scanned. This is similar to an approach taken by Li et al. [40] for cone-beam SPECT in which these voxels were referred to as "active" voxels. In that work, the FOA regions in projection and image space were rectangular and cubic, respectively. In this work, the FOA regions are much less constrained and thus in practice much smaller.

The focus of attention algorithm proceeds in two major steps. The first step is to threshold and filter the projection data to obtain a projection space mask. The second step includes backprojecting the projection space masks in order to obtain a support profile in the image space. This support profile is thresholded to obtain the final 3D FOA region. The algorithm is presented in Algorithm 4.2. We will now address these steps in detail using output from the mouse data set described in Appendix A.3.

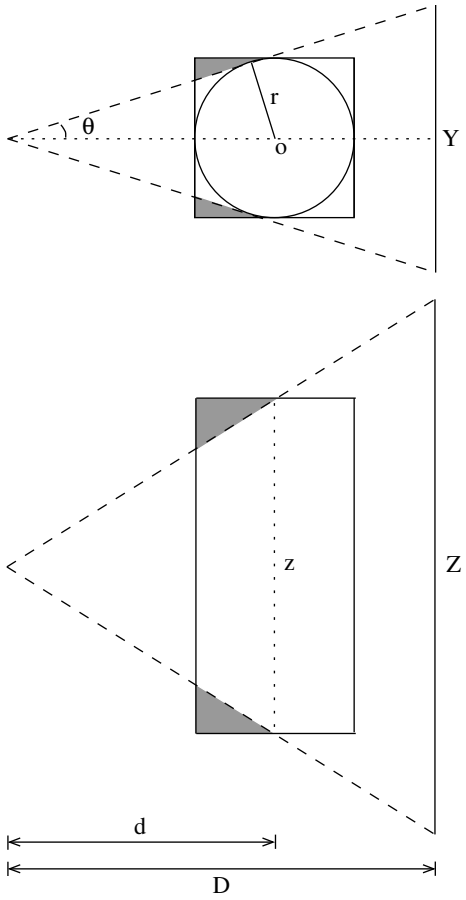

Figure 4.7: Calculating the dimensions of the voxel space. The shaded regions denote areas not supported by this single projection view. The axis of rotation for the top diagram is orthogonal to the page and passes through the center of the circular support region denoted by o, while the axis of rotation for the bottom diagram is the dotted line denoted by z.

Algorithm 4.2 Calculation of a 3D FOA region.

```
1: for all projections p do
2:    segment p into object and background
3:    apply median filter to p
4:    apply dilation filter to p
5:    for all voxels v do
6:       project v to detector plane
7:       if v projects to object detector element then
8:          increment counter for v
9:       end if
10:   end for
11: end for
12: for all voxels v do
13:   if counter for v exceeds threshold t then
14:      add v to object
15:   else
16:      add v to background
17:   end if
18: end for
19: extract FOA region containing object voxels
```

In step 2, we segment the projection data into object and background using the blank scan data, which consists of a projection with no object present. For each detector element, we mark the element as an object element if the value is less than 98% of the corresponding blank scan value. Figures 4.8 and 4.9 demonstrate a sample blank scan as well as projection data corresponding to a single view angle of a mouse. The segmentation resulting from thresholding the projection as described above is shown in Fig. 4.10. However, this segmentation contains some salt and pepper noise. In other words, some elements have been marked as object when they should be background or vice versa. In order to remove this noise, we apply a median filter, which is a standard image processing technique used to reduce such artifacts [47], to the segmented image. Furthermore, we do not want the segmentation to be too tight around the object as that will generate a tight segmentation in image space as well, which may cause reconstruction artifacts. Thus, after the median filter we apply a dilation filter, a well-known image processing technique for growing regions and filling holes, to generate a looser focus of attention mask in projection space, which will ultimately generate a looser image space FOA mask. The filtered version of the segmented image in Fig. 4.10 is shown in Fig. 4.11.

Rather than storing a value for each detector element to indicate whether or not it is an object element, we only store the indices for the minimum and maximum object element in each detector column and assume that all elements between the minimum and maximum are object elements as well. Since only two integer values must be stored per detector column, this approach uses significantly less memory than approaches that would store some value for each detector element. Furthermore, when calculating the support profile in image space, we will operate on ranges as well in order to reduce computation, so storing the object masks as ranges seems reasonable. Note that this may treat more detector elements as object elements than is strictly necessary, although this is not likely to occur with objects such as mice.

After calculating the filtered projection segmentation, we generate the corresponding support profile in image space as shown in steps 5-10 of the algorithm. However, since we only store the minimum and maximum detector object element indices in a detector column and assume that the contained elements are also object elements, we can avoid performing a projection for every voxel as is indicated in step 5 of the algorithm. Instead, for each voxel column we identify the minimum and maximum indices that project to detector object elements, and increment the counters for all voxels contained between the minimum and maximum voxel indices in that column. This can be done since all of the contained voxels will also project to the same detector column as the minimum and maximum object voxels, and therefore must correspond to object detector elements because there are no vertical holes in the segmented projections. Results of these steps for a single transaxial slice are shown in Fig. 4.12.

Once all of the projections have been processed, the final support profile contains a count for the number of projections that have identified a particular voxel as potentially contributing a non-negligible amount of attenuation. We then threshold this support profile to yield the 3D FOA region. In this work, we require that a voxel has support from 95% of the projections to be in the 3D FOA region.

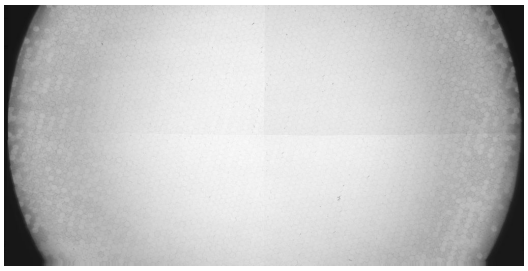

Figure 4.8: Blank scan projection data.

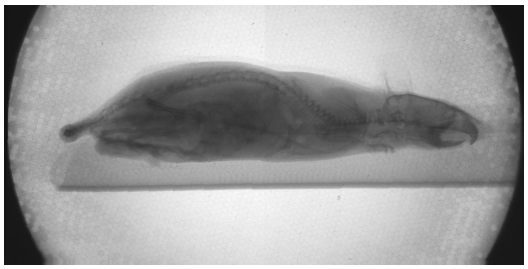

Figure 4.9: Raw projection data corresponding to a mouse at view angle zero.

Figure 4.10: Initial segmentation of the projection from Fig. 4.9 using a threshold value of 98%. This segmentation contains some salt and pepper noise.

Figure 4.11: Result of applying median and dilation filters to the segmentation in Fig. 4.10.

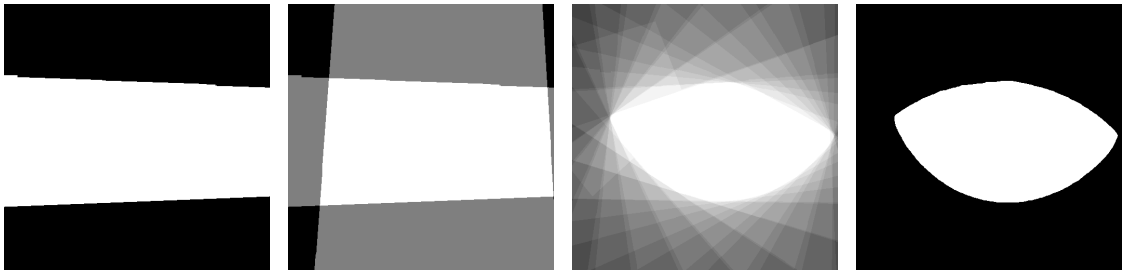

Figure 4.12: Steps of the FOA algorithm for a transaxial slice of the 3D FOA region. The first two images are the result of backprojecting one and two projections, respectively, while the third is the result of backprojecting all 360 projections. The rightmost image is the final thresholded mask.

4.6 Load Balancing Revisited

In Section 4.1.3, we evaluated the effectiveness of our load balancing approach for solving the standard linear system. However, by employing ordered subsets and 3D FOA, we substantially impact several aspects of the workload. The OS algorithm implementations increase the relative cost of the MPI communications, but they also increase the number of synchronization points. In particular, each subset requires communication, so there will be one synchronization point per subiteration rather than one per iteration. Furthermore, since 3D FOA eliminates some data from consideration in both the projection and image spaces, the total amount of work per iteration, and thus the amount of work per processor, is affected. For example, detector elements that are not within the projection space FOA region are ignored, along with their associated system matrix rows, so the detector groups that are distributed to the processors no longer have the same number of corresponding system matrix rows. Furthermore, each system matrix row will only contain entries for voxels within the 3D FOA region, so the number of nonzero system matrix rows will be affected as well. Thus, it is important to re-evaluate our load balancing approach to determine if it remains effective. Since the 3D FOA algorithm depends upon the object being scanned, these results are specific to our reconstruction of the mouse data set given in Appendix A.3. We will use the same metrics in this section as we used in Section 4.1.3 to evaluate the load balancing approach. These metrics include the number of considered system matrix rows per processor, the number of nonzero system matrix elements, and time spent on synchronization.

The first metric, the number of considered system matrix rows, is shown in Table 4.5. We see that the workload distribution remains quite even, with approximately 3.5 million system matrix rows per processor. Recall that without the 3D FOA algorithm, each processor managed approximately 5.6 million system matrix row per processor, so 3D FOA reduced the number of system matrix rows by about 37%.

While the system matrix rows generally have different numbers of elements due to the varying intersection lengths of the associated projection rays with the voxel space, the use of 3D FOA may exacerbate this problem because some voxels will be ignored. Thus, we show the number of partial nonzero system matrix elements per processor in Table 4.6. Again, the numbers appear quite even across the processors. Furthermore, there are approximately

Table 4.5: Number of system matrix rows assigned to each processor of each node for a sixteen node reconstruction when using the 3D FOA algorithm.

Node	CPU 1	CPU 2
1	3,568,091	3,571,513
2	3,569,189	3,571,911
3	3,567,571	3,570,783
4	3,570,318	3,571,136
5	3,569,419	3,569,485
6	3,570,246	3,570,495
7	3,569,403	3,568,877
8	3,570,336	3,569,128

Node	CPU 1	CPU 2
9	3,567,890	3,568,135
10	3,570,458	3,569,587
11	3,569,991	3,569,164
12	3,569,885	3,571,535
13	3,569,496	3,570,151
14	3,571,548	3,570,969
15	3,572,031	3,570,069
16	3,573,071	3,570,214

Table 4.6: Number of partial system matrix elements computed per node per cpu for a sixteen node reconstruction when using the 3D FOA algorithm.

Node	CPU 1	CPU 2
1	5,995,682,062	5,995,817,258
2	5,995,726,153	5,995,797,589
3	5,995,811,583	5,995,817,473
4	5,995,889,843	5,995,647,777
5	5,995,970,791	5,995,277,932
6	5,996,017,345	5,995,086,984
7	5,995,933,664	5,995,218,467
8	5,996,106,816	5,995,372,374

Node	CPU 1	CPU 2
9	5,995,964,261	5,995,529,934
10	5,995,875,064	5,995,427,989
11	5,995,890,335	5,995,471,701
12	5,996,029,716	5,995,519,513
13	5,996,125,679	5,995,598,450
14	5,996,064,375	5,995,682,366
15	5,996,205,251	5,995,646,523
16	5,995,955,727	5,995,637,056

6 billion nonzero partial system matrix elements when using 3D FOA, as compared to about 21.5 billion when not using any region of interest restriction. Thus, the 3D FOA algorithm reduces the number of nonzero partial system matrix elements by about 72% in this case. Additionally, there are around 3.4 billion true nonzero system matrix elements, so the real reduction in the number of nonzeros is also approximately 72%. We will more thoroughly consider the impact of the support region and 3D FOA techniques on the reconstruction performance in Chapter 5. In this section, we only consider the effectiveness of the load balancing strategy when using these approaches.

As before, the time spent on synchronization is probably the best measure of the effectiveness of the load balancing algorithm. However, we now have one synchronization point per subiteration rather than one per iteration. In Figs. 4.13 and 4.14, we display the percentage of the total iteration time spent on synchronization costs for two reconstructions of the mouse data set described in Appendix A.3 using OS-SIRT-48 with 3D FOA. These figures demonstrate that this implementation is particularly well-balanced. In fact, the synchronization costs consume less than one percent of the total iteration time and do not exhibit the behavior demonstrated in Section 4.1.3 where some nodes would slow the others down, seemingly at random. Note, however, that since there are now forty-eight synchronization points per iteration, variability due to noise rather than uneven workload distribution will tend to average out so that no single node stands out as problematic. Regardless, the current workload distribution approach is clearly effective for this reconstruction.

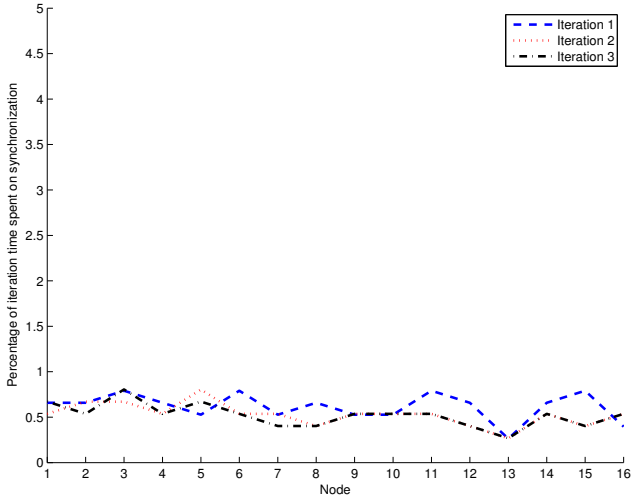

Figure 4.13: The percent of the total iteration time for each node spent on synchronization for the first reconstruction using OS-SIRT-48 with 3D FOA.

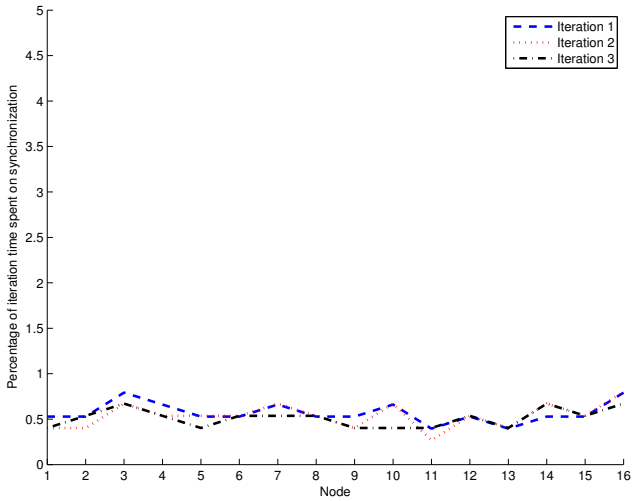

Figure 4.14: The percent of the total iteration time for each node spent on synchronization for the second reconstruction using OS-SIRT-48 with 3D FOA.

Chapter 5

Results

In this chapter, we will quantify the benefits of the techniques discussed in Chapter 4. All of the results in this chapter use the trilinear interpolation system model and either the SIRT or PSIRT algorithm. Additionally, all of the timings correspond to reconstructions performed on the grig cluster, except for some of the results in Section 5.5 that correspond to reconstructions on the frodo cluster. The grig and frodo clusters are described in Appendices B.1 and B.2, respectively.

We will focus on a single technique at a time and demonstrate its advantages rather than attempt to compare all possible combinations of techniques and algorithms. In Section 5.1, we will establish the run-time benefits of distributing the computation of the SIRT algorithm to multiple nodes. We then focus on the benefits of implementing ordered subsets in terms of reduction of the weighted error norm from the associated least squares problem as well as decreasing the final run-time in Section 5.2. We then address the memory and run-time advantages of restricting the reconstruction to the support region or focus of attention region in Section 5.3. In Section 5.4 we experimentally compare the error norms computed for the SIRT and PSIRT algorithms to demonstrate the effectiveness of PSIRT and also compare the run-times of the algorithms. We then revisit the parallel computing issue in Section 5.5 and compare run-times for reconstructions on the frodo and grig clusters using varying numbers of nodes. Finally, in Section 5.6 we present several slices from reconstructions of the mouse data set described in Appendix A.3,

5.1 Benefits of Parallelization

Our first goal is to demonstrate the advantages of parallelizing the reconstruction software to compute on multiple cluster nodes. We presented details of our workload distribution in Section 4.1 and established that our approach effectively distributes the required processing to several cluster nodes, but we did not present run-time comparisons in that section. We will do so in this section.

Parallelization does not affect the total amount of computation performed, except for a negligible amount of overhead required for determining the computational responsibilities of each node, but does decrease the workload for any given node and thus decreases the overall reconstruction time. We present a chart of the time required for the computation of the first SIRT iteration using varying numbers of nodes in Fig. 5.1. In this figure, we see

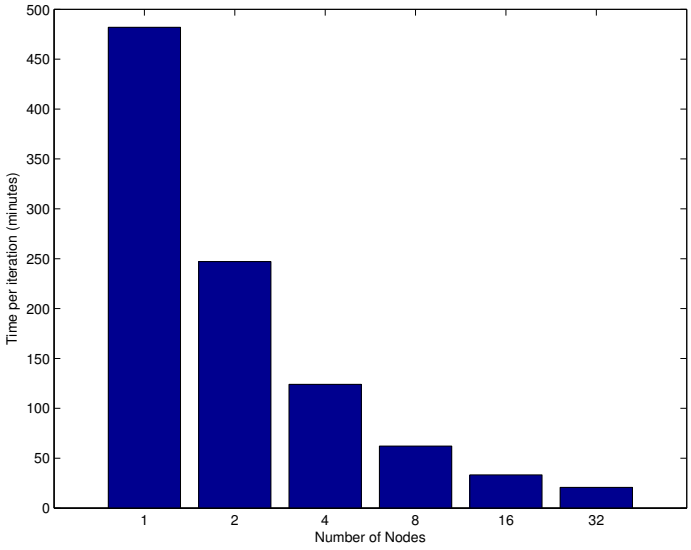

Figure 5.1: Run-times for the first iteration of SIRT using varying numbers of nodes.

that doubling the number of nodes approximately halves the reconstruction time, although the effect diminishes with an increasing number of nodes. One cause of the diminishing returns is the increasing cost of the MPI communications required when using more nodes.

Unfortunately, even with thirty-two nodes, the first iteration consumes over twenty minutes. The subsequent iterations compute faster because the column sums calculated and communicated during the first iteration are stored for future use. Thus, the subsequent iterations require approximately 13.5 minutes with thirty-two nodes. While this reduction is substantial, many iterations will be required to yield acceptable reconstructions, so the total run-time of a reconstruction would still be quite high. Computing ten SIRT iterations using thirty-two nodes, for example, would require over two hours. As we will demonstrate later, many more iterations are needed for acceptable reconstructions. Thus, while parallelization effectively distributes the workload, we need to reduce the amount of required computation to make the reconstructions more feasible. After presenting the impact of the workload reduction techniques, we will revisit the issue of parallelization and more thoroughly analyze the cost breakdown of the reconstructions in Section 5.5.

5.2 Benefits of Ordered Subsets

Ordered subsets, as described in Section 4.2, is a method of decreasing the number of iterations required to obtain acceptable reconstructions by computing subiterations using subsets of the data. In this section, we will analyze the impact of implementing ordered subsets in terms of reducing the weighted error norm and the total reconstruction run-time. Recall that SIRT is basically a weighted least squares solver. Thus, we can compute this weighted error norm after each iteration in order to quantify the effectiveness of SIRT in reducing the error norm. Furthermore, we can compute the same norm after each subiteration of OS-SIRT and compare the resulting norms. We present the weighted error norms

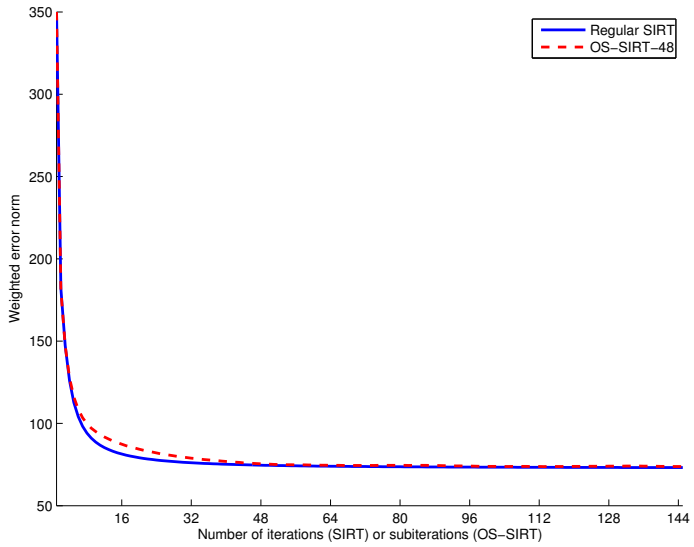

Figure 5.2: Computed weighted error norms for the SIRT and OS-SIRT algorithms.

for the first 144 SIRT iterations and each subiteration of the first three iterations of OS-SIRT-48 in Fig. 5.2. From this figure, we see that computing a subiteration of OS-SIRT-48 is approximately equivalent in terms of error norm reduction to computing a full iteration of regular SIRT, especially after the first full OS-SIRT-48 iteration. Thus, computing three OS-SIRT-48 iterations generates a reconstruction with an error norm similar to using 144 regular SIRT iterations. While it is difficult to discern in Fig. 5.2, regular SIRT generates a monotonically decreasing norm sequence to the extent run whereas OS-SIRT-48 does not, although when the OS-SIRT-48 generated norm increases it does so only slightly.

Therefore, ordered subsets is a very effective method of decreasing the number of iterations required to yield an acceptable reconstruction using our data set. However, there is one disadvantage of implementing ordered subsets in a distributed environment: the relative cost of MPI communication per iteration increases with the number of subsets. This increase occurs because an MPI communication is required after each OS-SIRT subiteration as opposed to after each full SIRT iteration. Furthermore, since the column sums cannot be stored from one iteration to the next in the case of OS-SIRT due to memory constraints, the column sums must also be communicated after each subiteration. Despite these increased costs, the overall impact of implementing ordered subsets is significant. For example, using thirty-two nodes, one OS-SIRT-48 iteration requires approximately 46 minutes to compute, while one SIRT iteration (after the first) requires approximately 13.5 minutes. However, one OS-SIRT-48 iteration reduces the error norm approximately the same as 48 regular SIRT iterations, which would take almost 11 hours. Thus, in this case, using ordered subsets reduces the per iteration run-time by about 93%. It is important to note that the cost breakdown of a regular SIRT and an OS-SIRT-48 iteration are quite different. The regular SIRT iterations are nearly all computation, while the OS-SIRT-48 iterations have substantial MPI communication costs. We show these costs when using thirty-two nodes in pie chart format in Fig. 5.3. The computation times refer to the longest amount of time spent by any node computing since the nodes must be synchronized before completing the

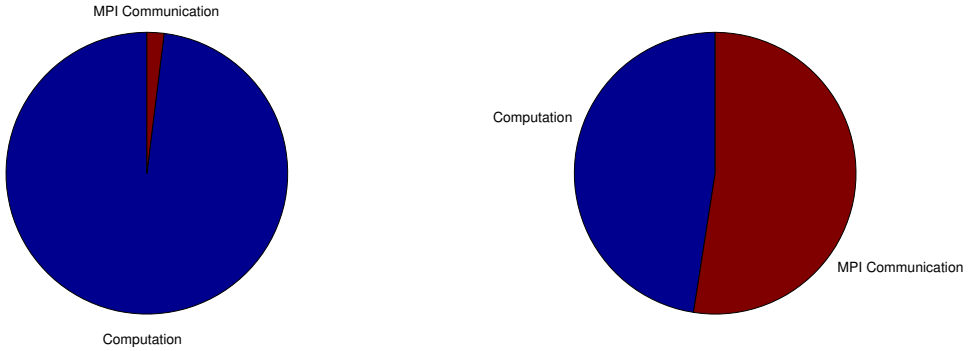

Figure 5.3: Per iteration relative-cost breakdown of SIRT (left) and OS-SIRT-48 (right).

MPI communication. These results are dependent on the number of nodes as well as the number of subsets in the case of OS-SIRT. We will more thoroughly consider how the MPI communication times change with a varying number of nodes in Section 5.5. However, it is clear that ordered subsets is a very effective mechanism for reducing the final run-time of a reconstruction.

5.3 Benefits of Region Restriction

The timings presented thus far in this chapter used the full image and projection spaces. We will now consider the implications of restricting the reconstructions to the support region and focus of attention region. The fully supported region (FSR), introduced in Section 4.4, consists of the set of voxels supported by the scanner geometry. By restricting the reconstruction to the FSR, we will reduce computation, memory consumption, and MPI communication. The 3D focus of attention (FOA) technique, discussed in Section 4.5, is a heuristic data-driven approach that further restricts the image space to voxels supported by the projection data and also restricts the projection space based on inferred object attenuation. In this section, we will quantify the effects of these approaches when using OS-SIRT-48.

Table 5.1 summarizes the main impacts of restricting the reconstruction to the FSR and FOA regions for an OS-SIRT-48 reconstruction of the mouse data set. By reducing the number of projection rays and voxels, which correspond to the rows and columns of the system matrix, respectively, we decrease the size of the problem. This eliminates a significant amount of computation because fewer system matrix elements must be computed. Furthermore, the reduction in the number of considered voxels lowers the memory burden imposed by storing the three image-sized data structures required for OS-SIRT. The true memory requirements are slightly higher than those shown in the table because memory for storing the projection data and other data structures is not included. Thus, if only 2 GB of memory is available per node, which is a common configuration, then a 3D FOA based reconstruction would be the only option. In addition to reducing the computation and memory burdens, decreasing the number of considered voxels also reduces the amount of

Table 5.1: Several metrics measuring the effectiveness of the region restriction techniques. The numbers are rounded and the memory calculation only considers the image sized vectors, which dominate the memory requirements. All numbers correspond to OS-SIRT-48 on thirty-two cluster nodes.

	No restriction	FSR	FOA
Number of projection rays	188 million	188 million	114 million
Number of voxels	268 million	178 million	67 million
Approximate memory (images)	3 GB	2 GB	0.75 GB
Per iteration run-time	46 min	29 min	12 min

MPI communication required because only values corresponding to considered voxels need to be communicated.

Thus, there are clearly many advantages to employing the region restriction approaches. In particular, by restricting the reconstruction to the 3D FOA region, we reduce the cost of a single OS-SIRT-48 iteration on thirty-two nodes from 46 minutes to 12 minutes for a 74% reduction. Of course, it is also important that the reconstructed images from each of the approaches be of similar quality. We have neither noticed nor measured any image quality degradation associated with restricting reconstructions to the FSR or 3D FOA regions. We include reconstructed slices using the different approaches in Section 5.6.

5.4 Benefits of Parallel SIRT

Using the techniques discussed in previous sections, a single iteration of OS-SIRT-48 distributed to thirty-two cluster nodes and using the 3D FOA based region restriction requires approximately 12 minutes of run-time. About half of this time is computation and the other half is communication. By employing the parallel SIRT algorithm (PSIRT), we will further reduce the computation and MPI communication in iterations after the first. However, before addressing the run-time of PSIRT, we will consider its ability to reduce the weighted error norm.

In Section 3.5 we demonstrated that SIRT and PSIRT converge based on a few assumptions. However, we did not address their rates of convergence. While the true rate of convergence depends upon the spectral radius of the iteration matrix, which we cannot easily compute, we will compare the convergence experimentally using the computed weighted error norms. Since SIRT and PSIRT are solving the same weighted least squares problem, we can directly compare the weighted error norm sequences generated by the two algorithms. Figures 5.4 and 5.5 show weighted error norms for the first 144 iterations of SIRT and PSIRT and the first three iterations of OS-SIRT-48 and OS-PSIRT-48. The SIRT and PSIRT algorithms both generate monotonically decreasing error norms to the extent run. The error norms generated by SIRT are generally smaller than those generated by PSIRT, but only marginally so. Thus, in terms of error norm reduction, PSIRT is competitive with SIRT for this data set. Since we will use ordered subsets in practice, we also consider the error norms generated by the first three iterations of OS-SIRT-48 and OS-PSIRT-48. These norms are very similar to those generated by SIRT and PSIRT, although the ordered subset

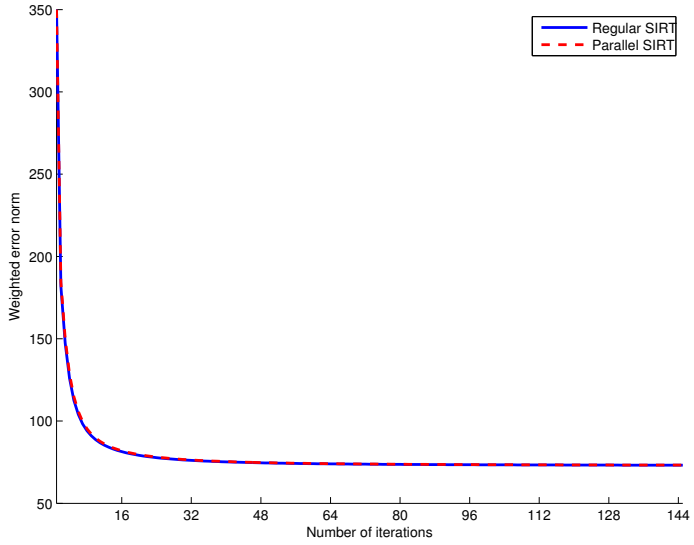

Figure 5.4: Computed weighted error norms for the first 144 iterations of SIRT and PSIRT.

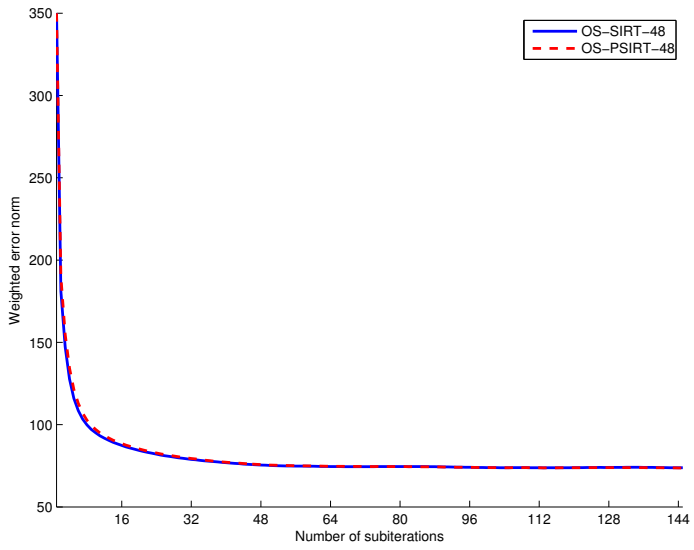

Figure 5.5: Computed weighted error norms for the first three iterations of OS-SIRT-48 vs OS-PSIRT-48.

versions do not generate monotonically decreasing error norm sequences. However, based on reducing the weighted error norm, OS-PSIRT-48 is very competitive with OS-SIRT-48. Thus, if OS-PSIRT has a significantly lower run-time, then it is reasonable to use it instead of OS-SIRT.

The first iterations of OS-SIRT-48 and OS-PSIRT-48 are essentially the same and thus have approximately the same run-time. In fact, since the full column sums are calculated during the first iteration of OS-PSIRT-48, we could use those full column sums during the first iteration and store the maximum column sum for each subset for use in subsequent iterations. If we do this, then the first iterations of OS-SIRT-48 and OS-PSIRT-48 generate identical images. However, we use the maximum column sums during the first iteration of OS-PSIRT-48 for consistency of presentation, so the image computed after the first iteration is slightly different than that computed by OS-SIRT-48. But, during the second and subsequent iterations, we do not have to compute or communicate the column sums when using OS-PSIRT-48. The first iteration of both OS-SIRT-48 and OS-PSIRT-48 requires about 10.2 minutes with nearly 5.9 minutes corresponding to MPI communication. However, the second and subsequent iterations of OS-PSIRT-48 require only 6.8 minutes each with 2.9 minutes of MPI communication, while the second and subsequent iterations of OS-SIRT-48 have the same cost breakdown as the first iteration. Thus, for the second and subsequent iterations, the computation and MPI communication times for OS-PSIRT-48 are reduced by approximately 9% and 50%, respectively, and the total run-time is reduced by 33%.

5.5 Parallel Computing Revisited

Now that we have established that using OS-PSIRT-48 and restricting the reconstruction to the 3D FOA region is substantially faster than regular SIRT, we will reconsider the reconstruction costs of OS-PSIRT-48 in terms of the number of cluster nodes. In particular, as the number of nodes increases, the computation time per node decreases, but the MPI communication time increases. Thus, with different node configurations, the mix of computation and communication times differ. At some point, adding cluster nodes will actually increase the total reconstruction time due to increasing communication costs, although that does not occur with our available number of nodes. In this section, we display results from both the grig cluster described in Appendix B.1 and the frodo cluster described in Appendix B.2. Note that both of these clusters have sixty-four nodes, but neither had all sixty-four nodes available as of this writing due to hardware issues, so the maximum number of nodes used is thirty-two.

Tables 5.2 and 5.3 show the cost breakdown for a single iteration (after the first) of OS-PSIRT-48 on the grig and frodo clusters, respectively, with varying numbers of nodes. The percentages spent on computation and communication on each cluster are very similar. The communication times increase from zero to 41% of the iteration run-time on frodo and 43% of the iteration run-time on grig. On the other hand, the computation times decrease by nearly half for both clusters as the number of nodes doubles, although the effect diminishes with an increasing number of nodes. We see from this data that increasing from sixteen nodes to thirty-two nodes is approximately 30% faster on grig and 28% faster on frodo. While the ideal increase would be 50%, the results are still quite good considering the

Table 5.2: Reconstruction timings of OS-PSIRT-48 with varying numbers of grig nodes. Timings are reported in minutes for the second and subsequent iterations and are separated into computation and communication.

Nodes	Per Iteration	Computation	Comp. (%)	Communication	Comm. (%)
1	101.1	101.1	100%	–	–
2	52.2	50.7	97%	1.5	3%
4	27.9	25.8	92%	2.1	8%
8	15.7	13.3	85%	2.4	15%
16	9.7	7.1	73%	2.6	27%
32	6.8	3.9	57%	2.9	43%

Table 5.3: Reconstruction timings of OS-PSIRT-48 with varying numbers of frodo nodes. Timings are reported in minutes for the second and subsequent iterations and are separated into computation and communication.

Nodes	Per Iteration	Computation	Comp. (%)	Communication	Comm. (%)
1	124.9	124.9	100%	–	–
2	64.3	62.1	97%	2.2	3%
4	35.1	32.3	92%	2.8	8%
8	19.4	16.2	84%	3.2	16%
16	12.0	8.6	72%	3.4	28%
32	8.6	5.1	59%	3.5	41%

amount of communication and synchronization that must occur during an iteration. Thus, this framework seems to scale quite well to many cluster nodes.

Furthermore, the computation times on grig are only about 20% lower than the computation times on frodo, despite the fact that the clock rates on grig's processors are more than double those on frodo's processors. This may be due in part to architectural differences, although we suspect that the primary bottleneck for a reconstruction is moving data in which case increasing the processor frequencies has less of an impact than may be expected.

5.6 Reconstructions

While the methods presented in Chapter 4 and quantified in this chapter are very effective at reducing the total reconstruction run-time, we must also consider the quality of the reconstructed images. By considering the reduction of the weighted error norm in Sections 5.2 and 5.4, we quantified the performance of ordered subsets and parallel SIRT in terms of solving the associated least squares problem, but that does not necessarily translate into acceptable image quality. Unfortunately, image quality is ultimately subjective and is therefore difficult to quantify. Thus, in order to demonstrate that the image quality is maintained after implementing the discussed techniques, we will first present several slices from four reconstructions using different reconstruction options. For each reconstruction, we present coronal slice 270, sagittal slice 225, and transaxial slices 425 and 575. We will then display line plots through the coronal and sagittal slices for each reconstruction to demonstrate that the reconstructed attenuation coefficients are close to each other for each method.

Figure 5.6 includes the four slices for a reconstruction using 144 iterations of SIRT and restricting the reconstruction to the support region. Since the voxels outside of the support region are not, by definition, supported by the scanner geometry, they cannot be accurately reconstructed. Thus, this reconstruction acts as a baseline with no special techniques used to reduce the run-time other than the support region restriction and parallelization. This reconstruction took almost 16.5 hours on thirty-two nodes.

In Fig. 5.7 we show the slices for a reconstruction using three iterations of OS-SIRT-48 and a support region restriction. Thus, these results demonstrate the image quality achieved after implementing ordered subsets for this data set. This reconstruction took about one hour and twenty minutes on thirty-two nodes for a run-time reduction of about 92% over 144 SIRT iterations. We already established that the results are similar in terms of reducing the weighted error norm, but we now see that they are also similar in terms of image quality. Thus, ordered subsets is clearly an effective mechanism for reducing the final reconstruction time.

We now consider the results of restricting the reconstruction to the 3D FOA region. Figure 5.8 displays the slices resulting from a three iteration OS-SIRT-48 reconstruction restricted to the FOA region. As we see, the results are very similar to those in Fig. 5.7. This seems reasonable since by restricting the reconstruction to the 3D FOA region, we simply do not expend computational resources on the areas of the reconstructed image that do not include the mouse. However, the FOA based reconstruction completed in approximately 34 minutes and is thus about 58% faster than the support region based reconstruction. Furthermore, the memory requirements are significantly reduced, which may allow us to run the reconstruction on clusters that otherwise would not have enough memory.

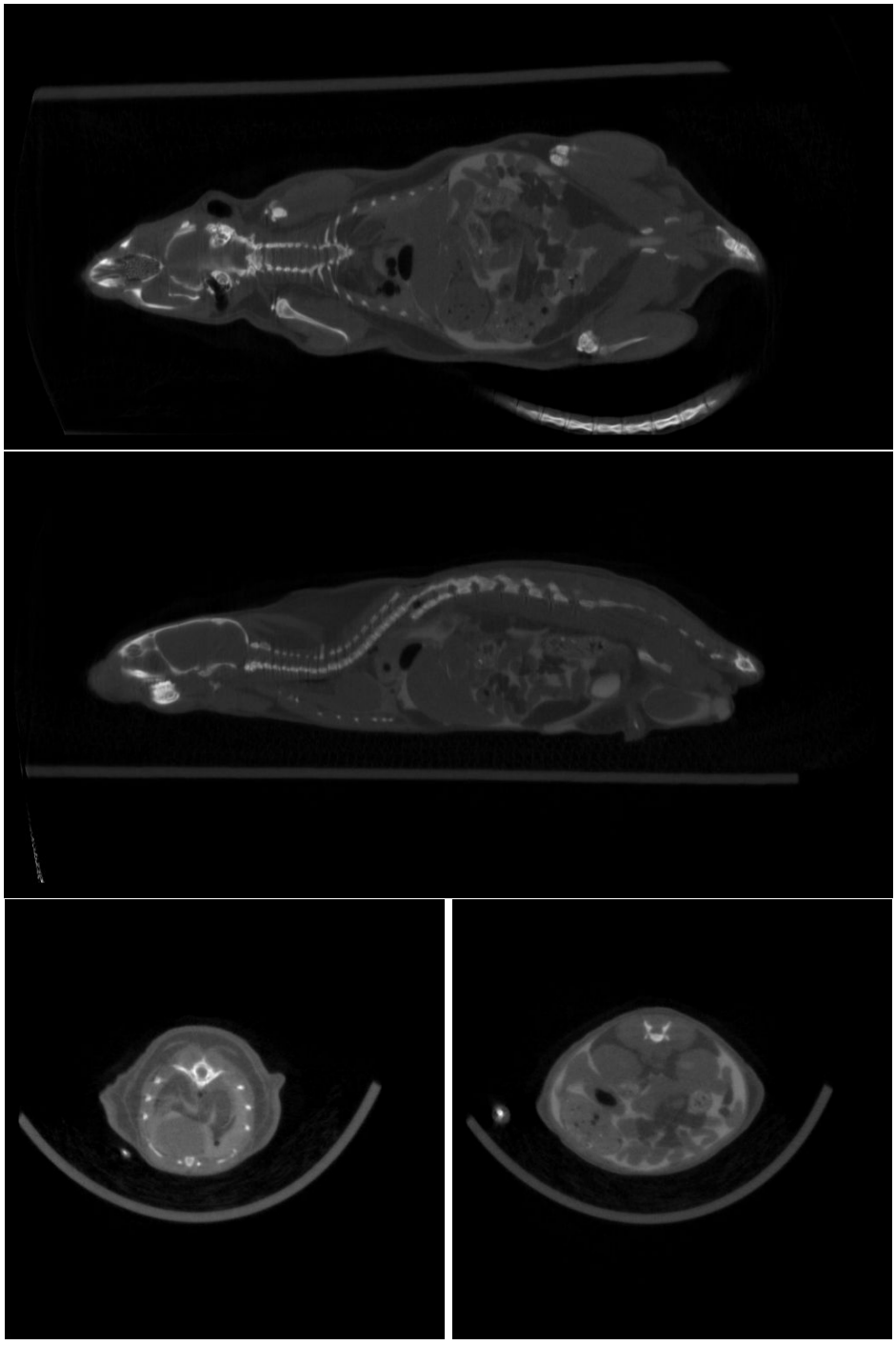

Figure 5.6: Slices from a 144 iteration SIRT based mouse reconstruction restricted to the support region. Slices are as follows: coronal slice 270 (top), sagittal slice 225 (middle), transaxial slice 425 (bottom left), and transaxial slice 575 (bottom right).

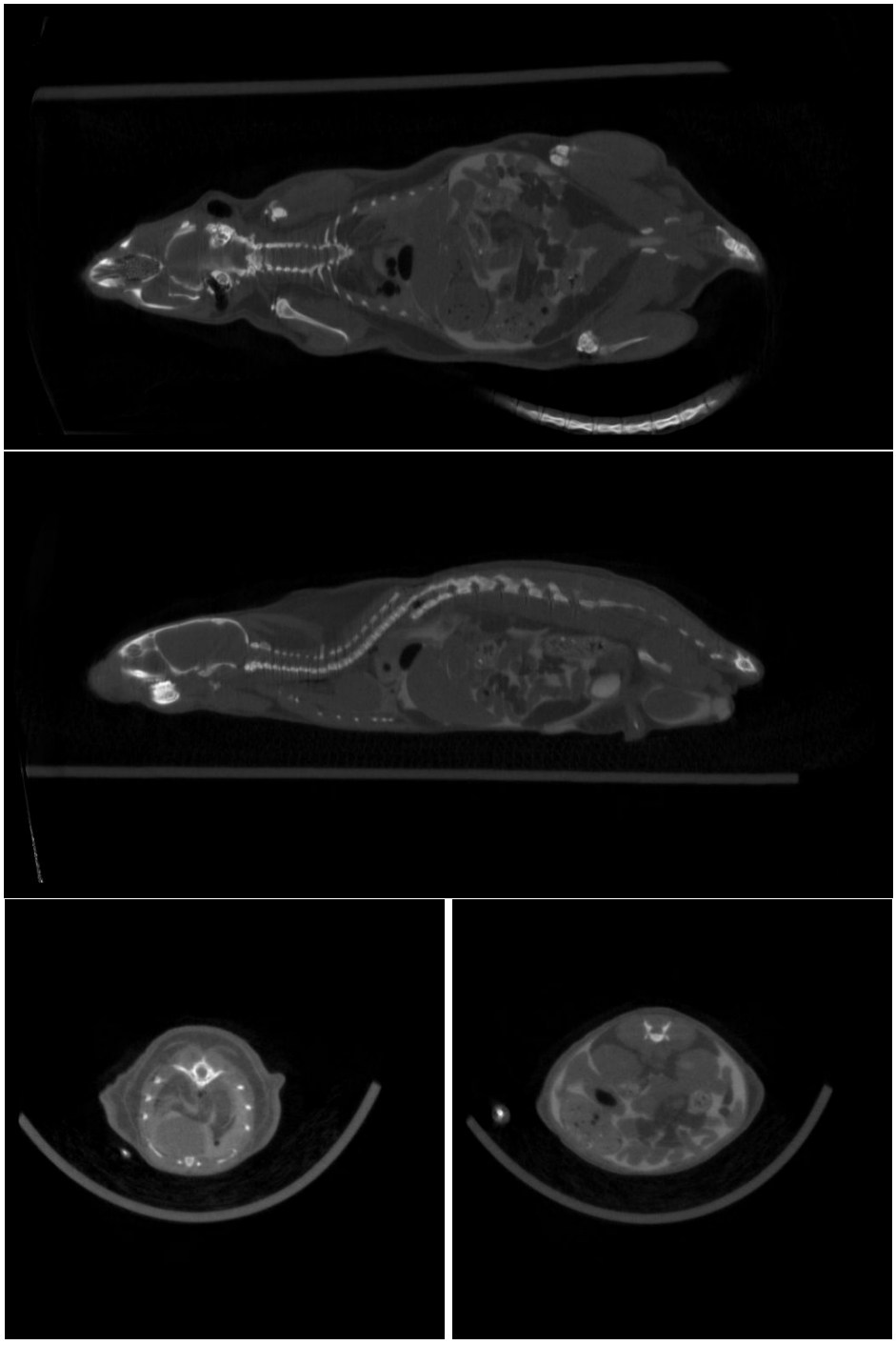

Figure 5.7: Slices from a three iteration OS-SIRT-48 based mouse reconstruction restricted to the support region. Slices are as follows: coronal slice 270 (top), sagittal slice 225 (middle), transaxial slice 425 (bottom left), and transaxial slice 575 (bottom right).

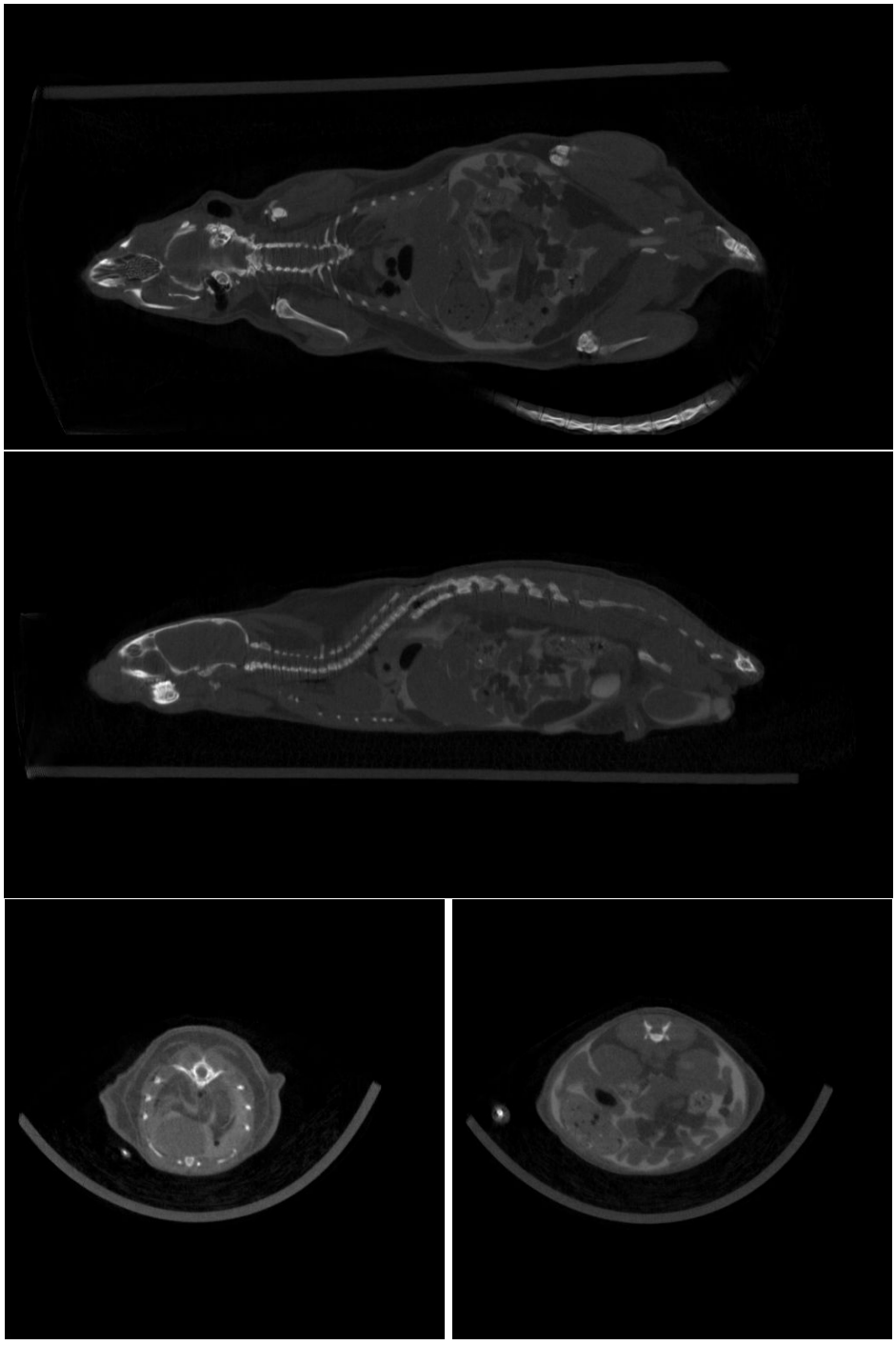

Figure 5.8: Slices from a three iteration OS-SIRT-48 based mouse reconstruction restricted to the focus of attention region. Slices are as follows: coronal slice 270 (top), sagittal slice 225 (middle), transaxial slice 425 (bottom left), and transaxial slice 575 (bottom right).

Finally, we present slices for a three iteration OS-PSIRT-48 reconstruction restricted to the 3D FOA region in Fig. 5.9. We demonstrated in Section 5.4 that PSIRT and OS-PSIRT yield very similar results to SIRT and OS-SIRT in terms of the weighted error norm. We now see in Fig. 5.9 that the results are very similar in terms of image quality as well. The OS-PSIRT-48 reconstruction completed in 28 minutes, or about 18% faster than OS-SIRT-48. The overall improvement from a 144 iteration SIRT based reconstruction to an OS-PSIRT-48 based reconstruction is over 97%.

We now consider line plots for each of the reconstructions through the coronal and sagittal slices. Figure 5.10 shows line plots for each of the coronal slices through $x = 240$ and Fig. 5.11 includes the same line plots with a tighter axis. We see in these figures that the reconstructed attenuation coefficients are very similar for each of the reconstructions. We see similar results in line plots for the sagittal slices through $y = 240$ in Figs. 5.12 and 5.13. While we do not have a baseline of correct values to compare against the reconstructed values, the similarity of the reconstructed values shows that three iterations of OS-PSIRT-48 with a 3D FOA restriction generates similar results to 144 iterations of SIRT with an FSR restriction. Therefore, the techniques introduced in Chapter 4 and quantified in this chapter are very effective at reducing the overall reconstruction run-time without significantly affecting the reconstruction quality.

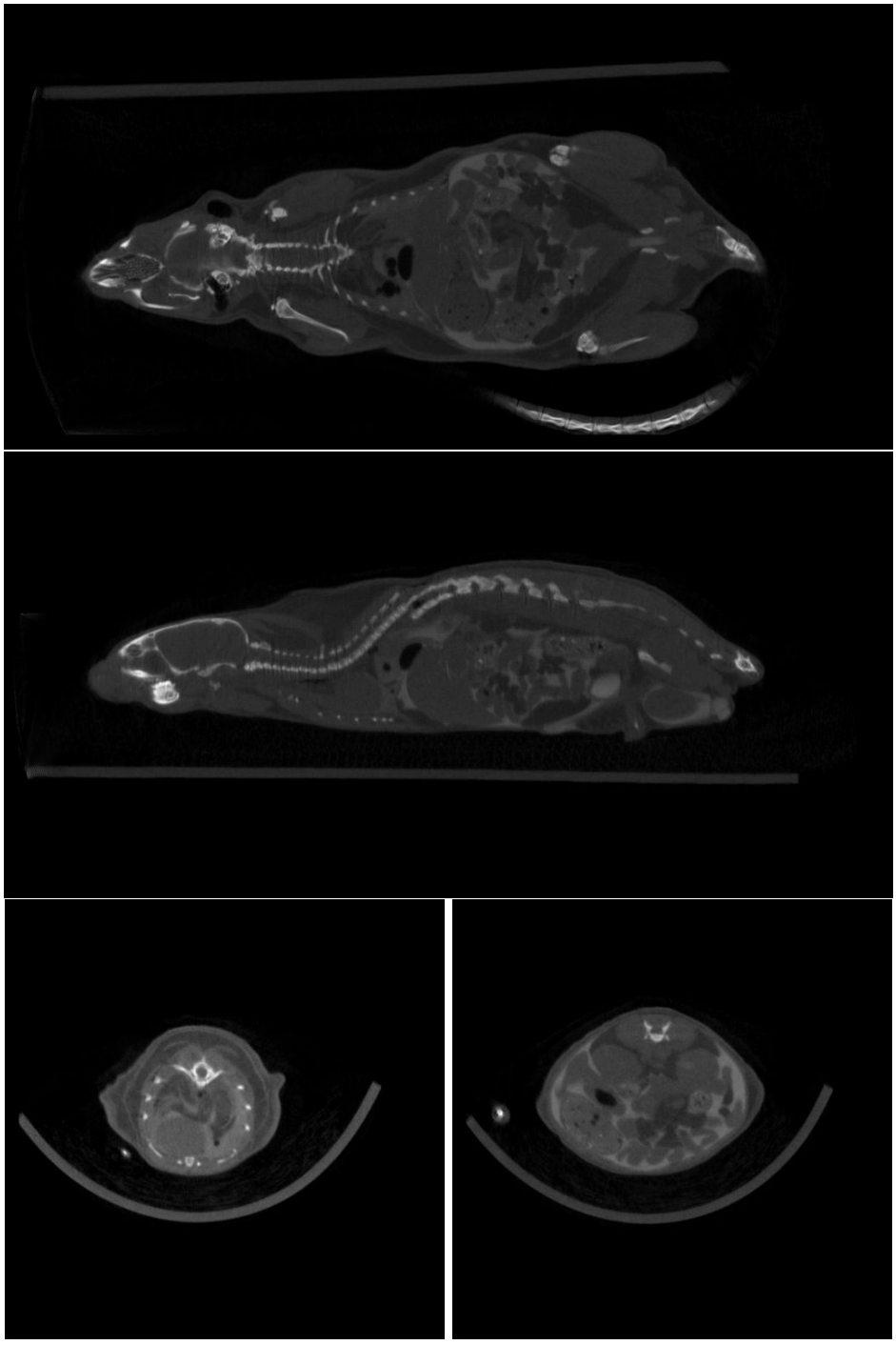

Figure 5.9: Slices from a three iteration OS-PSIRT-48 based mouse reconstruction restricted to the focus of attention region. Slices are as follows: coronal slice 270 (top), sagittal slice 225 (middle), transaxial slice 425 (bottom left), and transaxial slice 575 (bottom right).

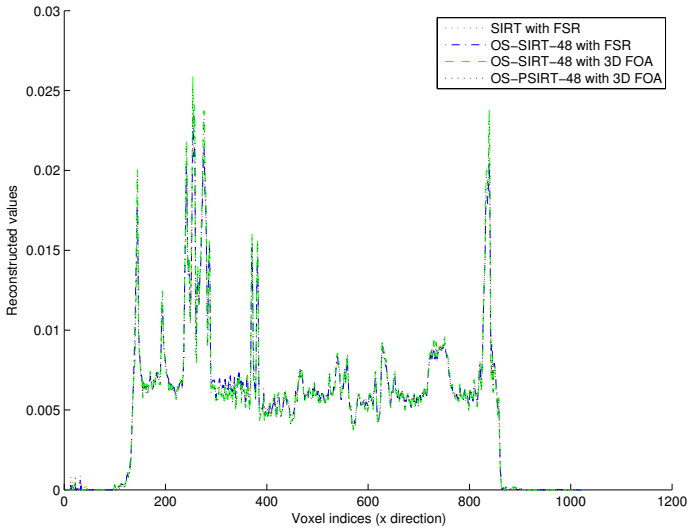

Figure 5.10: Line plot for the coronal slices through $x = 240$.

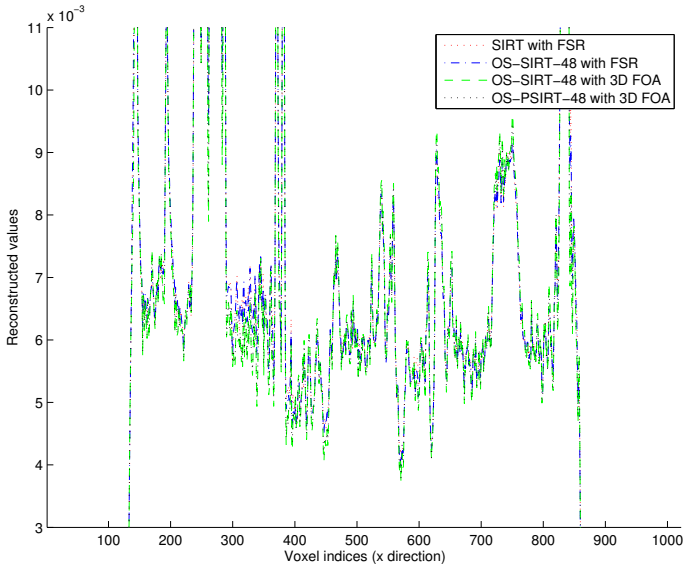

Figure 5.11: Zoomed line plot for the coronal slices through $x = 240$.

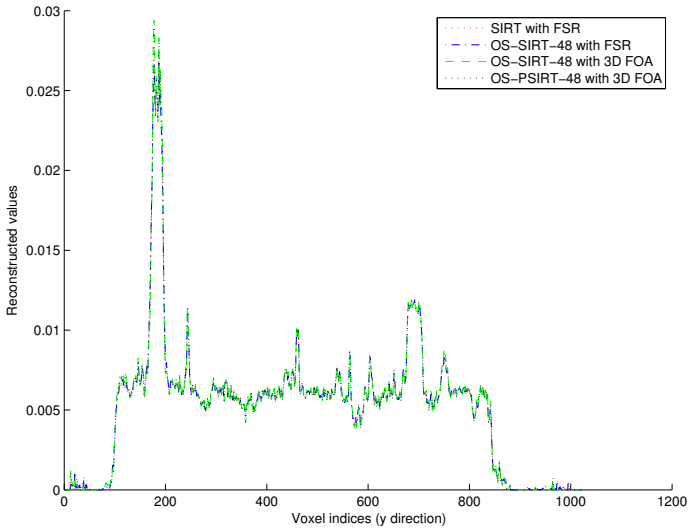

Figure 5.12: Line plot for the sagittal slices through $y = 240$.

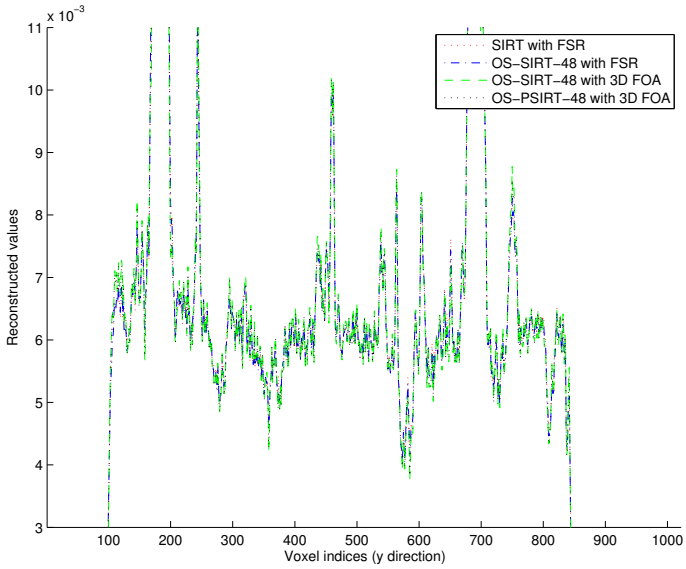

Figure 5.13: Zoomed line plot for the sagittal slices through $y = 240$.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The goal of this dissertation was to develop and implement techniques that facilitate iterative reconstructions of micro-CT data in a reasonable amount of time. We accomplished this goal in several stages. First, we employed parallel computing to distribute the workload to multiple processors on a single node via threads and to multiple nodes within a cluster via message passing (MPI). Parallelization significantly lowers the run-time by reducing the amount of work performed by any given processor, but it does not lower the total amount of computation needed and also introduces MPI communication costs.

We then implemented ordered subsets to reduce the amount of required computation. With an ordered subsets implementation, fewer full iterations are needed and thus the full system matrix is also computed fewer times. Although ordered subsets reduces the amount of computation, it increases the per iteration cost of the MPI communications. The increase in MPI costs is due to more frequent and larger communications. The former increase is a result of communication after each subiteration and the latter is due to always communicating the column sums because memory concerns prevent their storage from one iteration to the next. Despite these increased costs, ordered subsets very effectively reduces the overall run-time of a reconstruction. As demonstrated in Section 5.6, OS-SIRT produces an acceptable reconstruction 92% faster than regular SIRT for our data.

While ordered subsets reduces the amount of computation, it does so by considering only subsets of the data at a time as opposed to removing any of the data from consideration altogether. The region restriction techniques, on the other hand, take the latter approach. In the case of support region restriction, we restrict the reconstructed image volume to the voxels fully supported by the scanner geometry. This effectively reduces the number of columns in the system matrix and thus reduces the computational burden because system matrix elements are not computed for the ignored columns. Furthermore, it reduces the amount of memory required, which may allow us to run reconstructions that we otherwise could not, and reduces the amount of data that must be communicated because we do not transmit values for voxels outside of the support region. With the focus of attention technique, we utilize the projection data to identify background regions and restrict the image space as well as restricting the projection space. This further reduces the number of considered voxels and thus the number of system matrix columns and also reduces the

number of considered projection rays and their associated system matrix rows. Thus, focus of attention additionally decreases the computation, communication, and memory burdens.

The final approach to reducing the run-time is the incorporation of the parallel SIRT (PSIRT) algorithm. As mentioned above, due to implementing ordered subsets, we can no longer reasonably store the column sums from one iteration to the next because of memory consumption. The PSIRT algorithm addresses this shortcoming by replacing the full set of column sums for each subiteration by the inverse of that subiteration's maximum column sum in all iterations after the first. Thus, since the column sums no longer need to be computed or communicated for iterations after the first, we reduce both the computation and communication times.

By combining all of these approaches, we can perform an iterative reconstruction of a micro-CT data set in a reasonable amount of time. For example, a high-resolution reconstruction of a $512 \times 512 \times 1022$ image with $130\mu m$ voxels computes in less than thirty minutes. As a means of comparison, consider an implementation that uses the SIRT algorithm on both processors of a single node with no region restriction and no other acceleration techniques. The first iteration takes eight hours and each subsequent iteration takes about five and a half hours, so 144 iterations would require about 33 days. By restricting the reconstruction to the support region, the reconstruction would require about 22 days, which is still far too long. Of course, on a single node, it is more reasonable to implement either ART or SART, which are both similar to ordered subset implementations, to substantially reduce the run-time. However, these reconstructions would likely still take about a day, so our implementation remains much faster. Ultimately, while half an hour is still a substantial amount of time, it is a much less than would be required for a more direct implementation.

6.2 Future Work

While the techniques employed in this work were very effective at decreasing the run-time of an iterative reconstruction, more work can always be done. There are two methods in particular that we have considered during this work to potentially further decrease the reconstruction time. The first is an implementation detail while the second is an algorithmic change. There is also additional theoretical work that can be performed in regards to the algebraic properties of the system matrix.

As currently implemented, there is no reconstruction related computation performed during an MPI communication. This is because we cannot begin to compute the forward projection operation for the next iteration until the image update is complete. The image update, in turn, cannot complete before the MPI communication finishes. However, it is possible to calculate rows of the system matrix during the MPI communication, store those rows in memory, and then use the rows once the communication is complete and the image is updated. This would reduce the amount of computation associated with calculating system matrix rows after the communication completes and would thus potentially decrease the run-time. The primary difficulty with this approach would be the implementation. In particular, properly synchronizing the threads to perform such an overlapping approach would be difficult. However, if the time saved by overlapping the system matrix calculations exceeds the incurred overhead, then the overall run-time would be reduced without affecting the computed results. We are pursuing this option as an additional run-time reduction

technique and preliminary results indicate a run-time reduction of between ten and fifteen percent.

Another area for potential future work involves the reconstruction algorithm selection. In Chapter 3 we established connections between the popular algebraic reconstruction algorithms and traditional algebraic approaches to solving linear systems. We then modified the SIRT algorithm in such a way that the obtained results are similar but the computation and communication overhead are reduced. However, it would also be possible to implement a different traditional algorithm and use it for image reconstruction since we are ultimately just solving a linear system. In particular, we could apply the conjugate gradient method to the associated normal equations rather than pursue a matrix splitting approach. The conjugate gradient method has proved very useful in solving large sparse linear systems and thus may have substantial benefits in the field of reconstruction from projections as well. Thus, incorporating the conjugate gradient method into our framework also provides fertile ground for future work.

Finally, it would be useful to determine the rank of the system matrix resulting from the different system models. If it can be confirmed that the system matrix does in fact have full column rank, then the convergence argument given for SIRT in Chapter 3 provides a succinct convergence proof for both SIRT and PSIRT. Furthermore, if the spectral properties of the various iterations matrices could be established, then SIRT, Jacobi, and PSIRT could be compared in terms of convergence rates as well.

Bibliography

Bibliography

[1] L. A. Feldkamp, L. C. Davis, and J. W. Kress. Practical cone-beam algorithm. *Journal of the Optical Society of America*, 1:612–619, 1984.

[2] L. A. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *Trans. Med. Imag.*, MI-1(2):113–122, October 1982.

[3] K. Lange and R. Carson. EM reconstruction algorithms for emission and transmission tomography. *Journal of Computer Assisted Tomography*, 8(2):306–316, April 1984.

[4] G. Wang, D. L. Snyder, J. A. O'Sullivan, and M. W. Vannier. Iterative deblurring for CT metal artifact reduction. *Trans. Med. Imag.*, 15(5):657–663, October 1996.

[5] K. Mueller, R. Yagel, and J. J. Wheller. Anti-aliased three-dimensional cone-beam reconstruction of low-contrast objects with algebraic methods. *Trans. Med. Imag.*, 18:519–537, 1999.

[6] W. Chlewicki, C. Badea, and N. Pallikarakis. Cone based 3D reconstruction: A FDK-SART comparison for limited number of projections. In *Proceedings of the International Federation for Medical & Biological Engineering: Part I*, pages 495–497, June 2001. Proceedings are from the IX Mediterranean Conference on Medical and Biological Engineering and Computing.

[7] G. N. Hounsfield. Computerized transverse axial scanning (tomography): Part I. description of system. *British Journal of Radiology*, 46:1016–1022, December 1973.

[8] J. L. Prince and J. M. Links. *Medical Imaging: Signals and Systems*. Prentice Hall, 2006.

[9] K. Mueller, R. Yagel, and J. J. Wheller. Fast implementations of algebraic methods for three-dimensional reconstruction from cone-beam data. *Trans. Med. Imag.*, 18:538–548, June 1999.

[10] B. De Man and S. Basu. 3D distance-driven projection and backprojection. In *Proceedings of the VIIth International Conference on Fully 3D Reconstruction in Radiology and Nuclear Medicine*, July 2003.

[11] B. De Man and S. Basu. Distance-driven projection and backprojection in three dimensions. *Phys. Med. Biol.*, 49(11):2463–2475, June 2004.

[12] R. L. Siddon. Fast calculation of the exact radiological path for a three-dimensional CT array. *Medical Physics*, 12(2):252–255, 1985.

[13] F. Jacobs, E. Sundermann, B. De Sutter, M. Christiaens, and I. Lemahieu. A fast algorithm to calculate the exact radiological path through a pixel or voxel space. *Journal of Computing and Information Technology*, 6(1):89–94, 1998.

[14] G. L. Zeng and G. T. Gullberg. A study of reconstruction artifacts in cone beam tomography using filtered backprojection and iterative algorithms. *IEEE Trans. on Nuc. Sci.*, 37(2):759–767, 1990.

[15] G. Wang, M. W. Vannier, and P. Cheng. Iterative X-ray cone-beam tomography for metal artifact reduction and local region reconstruction. *Microscopy and Microanalysis*, 5:58–65, 1999.

[16] A. H. Andersen and A. C. Kak. Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm. *Ultrasonic Imaging*, 6:81–94, 1984.

[17] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, second edition, 1998.

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

[19] Y. Censor. Row-action methods for huge and sparse systems and their applications. 23(4):444–466, October 1981.

[20] J. Nuyts, B. De Man, P. Dupont, M. Defrise, P. Suetens, and L. Mortelmans. Iterative reconstruction for helical CT: a simulation study. *Phys. Med. Biol.*, 43:729–737, 1998.

[21] I. A. Elbakri and J. A. Fessler. Statistical image reconstruction for polyenergetic x-ray computed tomography. *Trans. Med. Imag.*, 21(2):89–99, February 2002.

[22] A. C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*. Society of Industrial and Applied Mathematics, 2001.

[23] G. T. Herman and A. Lent. Iterative reconstruction algorithms. *Computers in Biology and Medicine*, 6:273–294, October 1976.

[24] G. T. Herman, A. Lent, and S. W. Rowland. ART: Mathematics and applications : A report on the mathematical foundations and on the applicability to real data of the algebraic reconstruction techniques. *J. Theor. Biol.*, 42:1–32, November 1973.

[25] K. Tanabe. Projection method for solving a singular system of linear equations and its applications. *Numerische Mathematik*, 17(3):203–214, June 1971.

[26] S. Kaczmarz. Angenaherte auflosung von systemen linearer gleichungen. *Bull. Acad. Polon. Sci. Lett. A*, 6-8A:355–357, 1937.

[27] Å. Björck and T. Elfving. Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations. *BIT Numerical Mathematics*, 19(2):145–163, 1979.

[28] H. H. Barrett and K. J. Myers. *Foundations of Image Science*. John Wiley & Sons, Inc., 2004.

[29] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

[30] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

[31] M. T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, second edition, 2002.

[32] K. Mueller and R. Yagel. Rapid 3-D cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2-D texture mapping hardware. *Trans. Med. Imag.*, 19(12):1227–1237, December 2000.

[33] P. Gilbert. Iterative methods for the three-dimensional reconstruction of an object from projections. *J. Theor. Biol.*, 36:105–117, 1972.

[34] L. Landweber. An iteration formula for Fredholm integral equations of the first kind. *American Journal of Mathematics*, 73(3):615–624, July 1951.

[35] O. N. Strand. Theory and methods related to the singular-function expansion and Landweber's iteration for integral equations of the first kind. *SIAM J. Numer. Anal.*, 11(4):798–825, September 1974.

[36] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[37] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[38] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.

[39] W. D. Gropp and E. Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[40] J. Li, R. J. Jaszczak, K. L. Greer, and R. E. Coleman. Implementation of an accelerated iterative algorithm for cone-beam SPECT. *Phys. Med. Biol.*, 39(3):643–653, March 1994.

[41] G. L. Zeng and G. T. Gullberg. Unmatched projector/backprojector pairs in an iterative reconstruction algorithm. *Trans. Med. Imag.*, 19(5):548–555, 2000.

[42] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992.

[43] H. M. Hudson and R. S. Larkin. Accelerated image reconstruction using ordered subsets of projection data. *Trans. Med. Imag.*, 13(4):601–609, December 1994.

[44] T. M. Benson and J. Gregor. Distributed iterative image reconstruction for micro-CT with ordered-subsets and focus of attention problem reduction. *J. of X-ray Sci. and Tech.*, 12(4):231–240, December 2004.

[45] M. Grass, Th. Köhler, and R. Proksa. 3D cone-beam CT reconstruction for circular trajectories. *Phys. Med. Biol.*, 45(2):329–347, February 2000.

[46] M. Grass, Th. Köhler, and R. Proksa. Angular weighted hybrid cone-beam CT reconstruction for circular trajectories. *Phys. Med. Biol.*, 46(6):1595–1610, June 2001.

[47] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 2002.

Appendix

Appendix A

Data Sets

There are several data sets used for the results presented in this dissertation. These sections give more details about the data sets.

A.1 Shepp-Logan Phantom Data Set

The Shepp-Logan phantom is a widely used phantom described in detail in [22]. Our Shepp-Logan phantom data set contains 180 projections with 256×256 samples per projection. When using this phantom, we reconstruct a $256 \times 256 \times 256$ image.

A.2 Tiny Hole Phantom Data Set

The tiny hole phantom is a cubic inch of acrylic with groups of varying sized drill holes. There are nine rows of five drilled holes with sizes varying from row to row and remaining constant in each row. The center spacing of the holes on each row is twice the diameter of the holes on that row and the holes range in size from 0.05 mm to 1.5 mm. An identical set of holes is drilled into two sides of the phantom. Figure A.1 depicts the dimensions of the drilled holes and Fig. A.2 shows the phantom itself.

The phantom data set contains 720 projection view angles spaced evenly throughout the 360 degree scanning circle. The detector has physical dimensions of $33.4848\,\text{mm} \times 66.9696\,\text{mm}$ with 2048×1024 detector elements, which we downsample to 512×256 yielding a resampled detector pitch of approximately 131 μm. The distance from the x-ray source to the center of rotation is 256.2 mm and the distance from the x-ray source to the detector array is 309.6 mm. The detector was vertically offset by approximately 2.8 rebinned detector elements and the cone angle is approximately 12 degrees. Typically, we reconstruct a $512 \times 512 \times 256$ image from this data set.

A.3 Mouse Data Set

The mouse data set was acquired on a MicroCAT II scanner (Siemens / CTI-Concorde Microsystems). A projection was acquired for each degree of the 360 degree scanning circle for a total of 360 projection view angles. The detector has physical dimensions of

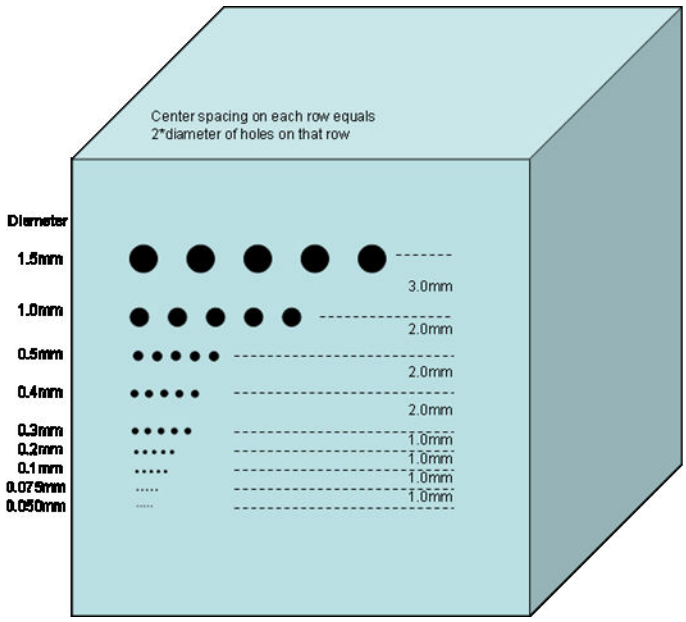

Figure A.1: The design of the tiny hole phantom.

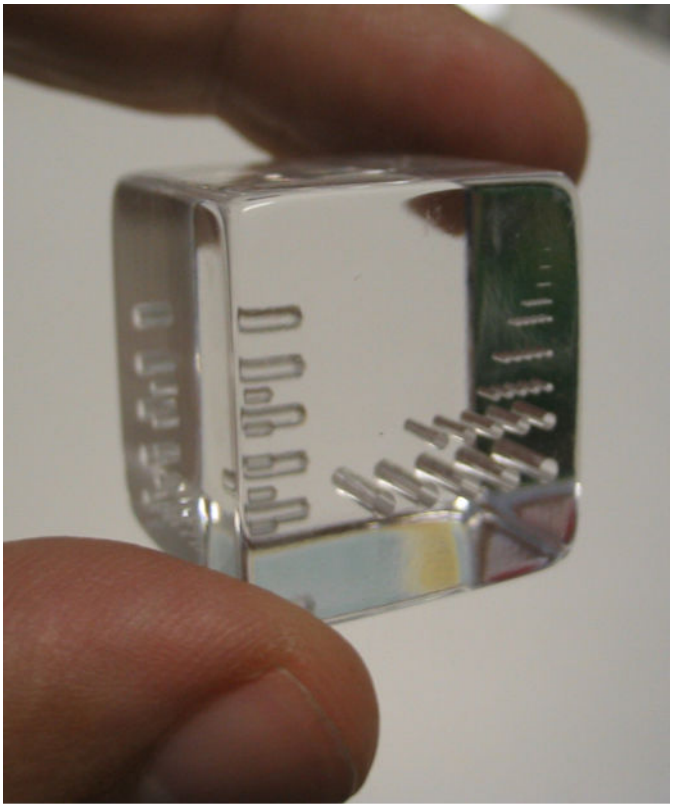

Figure A.2: A picture of the cubic inch tiny hole phantom.

82.5344 mm × 164.746 mm with 4096 × 4088 detector elements, which we downsample to 512 × 1022 yielding a resampled detector pitch of approximately 161 μm. The distance from the x-ray source to the center of rotation is 416.259 mm and the distance from the x-ray source to the detector array is 512.613 mm. The detector was vertically offset by approximately 14.5 rebinned detector elements and the cone angle is approximately 18 degrees. Typically, we reconstruct a $512 \times 512 \times 1022$ image from this data set. The mouse received an intraperitoneal injection of a water-soluble iodinated contrast agent prior to being scanned.

Appendix B

Computing Resources

We utilize two clusters of computers for computations in this work: grig and frodo. We performed most of the timings and reconstructions on grig. The computer equipment was acquired as part of SInRG, a University of Tennessee grid infrastructure supported by the NSF under grant number EIA–9972889.

B.1 Grig Cluster

The grig cluster contains sixty-four nodes with identical hardware. Each node has two Intel Xeon EM64T 3.2GHz processors, four gigabytes of RAM, and a Myrinet interconnection (with product ID M3F-PCIXD-2). The cluster runs Debian Linux with GCC version 3.4.3 and we compile the software using the optimization flag -O2.

B.2 Frodo Cluster

The frodo cluster also contains sixty-four nodes with identical hardware. Each node has two AMD Opteron 240 1.4 GHz processors, two gigabytes of RAM, and a Myrinet interconnection (with product ID M3F-PCIXD-2). The cluster runs Debian Linux with GCC version 3.4.3 and we compile the software identically to grig, using optimization flag -O2.

Vita

Thomas Matthew Benson was born in Greeneville, TN. He attended grade school and high school there and received a high school diploma from Greeneville High School in 1997. He subsequently attended the University of Tennessee, Knoxville, where he completed Bachelor of Science degrees with highest honors in Computer Science and Mathematics in May 2001. He then pursued graduate studies in Computer Science at the University of Tennessee, Knoxville, where he completed a Master of Science degree in December 2003 and the Doctor of Philosophy degree in May 2006.