3D Simultaneous Algebraic Reconstruction Technique for Cone-Beam Projections

Master of Science Thesis
by

Wojciech Chlewicki

Supervisor: Christian Badea
Promotor: Nicolas Pallikarakis

Patras 2001
Acknowledgements

I would like to thank my supervisor Christian Badea, PhD for introducing me to the field of computed tomography, for valuable help and discussions during the work, and his great attitude.

Professor Nikolas Pallikarakis for his guidance, advices and help in solving administrative tasks.

Professor Krzysztof Penkala who made my study in Greece possible.

My family for their "polydimensional" support.

This work was supported by a grant received from Greek State Foundation for Scholarships.
C three dimensional SART implementation
List of Figures

1.1 In case of circular trajectory, one cone beam projection provides Rodon values on the section of a sphere passing through the origin. If projections are acquired over 360°, the Rodon space will be filled inside a torus instead of a sphere. .... 10

3.1 Central slices of the Shepp-Logan phantom reconstructed with helical cone beam SART. .................. 29
3.2 Slices of the brain. Reconstruction of the SPECT data using SART. ................................. 29
3.3 The spine. Visualization of the reconstructed SPECT data using IDL. ................................. 30

4.1 Central slices of the mathematically simulated disk phantom. .. 32
4.2 Central slices of the mathematically simulated bar phantom. .. 32
4.3 Central slices of the mathematically simulated Shepp-Logan phantom. .......................... 33
4.4 Central slices of the Shepp-Logan phantom reconstructed with SART. .............................. 33
4.5 Central slices of the Shepp-Logan phantom reconstructed with FDK. ............................... 34
4.6 Central slices of the ellipsoid phantom reconstructed with SART - 6 iterations and 36 projections. .... 34
4.7 Central slices of the ellipsoid phantom reconstructed with FDK - 36 projections. .................. 35
4.8 SWRF plots for FDK using different number of projections. .. 36
4.9 SWRF plots for SART 6 iterations using different number of projections. ................................................................. 36
4.10 Illustrating root mean square reconstruction error versus number of projections for FDK and SART. ......................... 37
Chapter 1

Introduction

In 1972, Hounsfield has realized the first Computed Tomography (CT) scanner, which had revolutionized diagnostic medicine. His invention has showed that it was principally feasible, based on a very large number of measurements, to reconstruct a crossectional slice of a patient with fairly high accuracy.

The image quality of the slices, through the years improved significantly due to the development of computer power and reconstruction methods. Several algorithms for image reconstruction have been developed, and the techniques applied can be distinguished in two main categories [31]:

- analytical, that capitalize on the Fourier Slice Theorem,
- iterative, that seek to solve the reconstruction problem by solving a system of simultaneous linear equations.

Both approaches have advantages and disadvantages. Filtered Backprojection (FB) algorithm is a main representative of the first family. Reconstruction problem in this case is solved through an analytical way. In the second approach the solution (object function) is found iteratively. In the first CT unit iterative methods were used, however quality of the reconstructed images was poor and algorithms were time consuming. The discover of the Central Slice Theorem gave a base for developing of the less computationally expensive and less noisy analytical methods, making the FB method the most popular.
Due to changes of the rays geometry for more efficient data acquisition, image reconstruction methods started to be more and more sophisticated. Nowadays scanners with one dimensional detector array are in common use and two dimensional result (slice) is obtained. Usual method of acquiring a volumetric CT is scanning a patient slice by slice. In this circumstance, a linear detector array and a X-ray point source are mounted across from each other. By spinning the source/detector pair around the patient, a series of one dimensional projections are obtained, which are subsequently used for 2D slice reconstruction. A volumetric representation is obtained by advancing the table on which the patient rests after every spin, acquiring a stack of such slices. There are several problems in this case [31]:

- projection data covering patient regions between the thin slice sections is not available, which can be a problem if a tumor is located right between two slices,
- the stop-and-go table movement may cause patient displacement between subsequent slice-scans,
- a volumetric scan usually takes a long time, much longer than a single breath hold, due to the setup time prior to each slice acquisition (this introduces respiratory motion artifacts into the data).

The first two disadvantages have been eliminated with the recent introduction of the spiral (helical) CT scanning devices. Again, a one dimensional detector is used, but the patient table is translated continuously during the scan. Special reconstruction algorithms have been designed to produce a volumetric reconstruction based on the spiral projection data.

In spite of the fact that spiral CT is a tremendous progress for volumetric CT, some problems still exist [31]:

- although the volumetric scans are obtained 5 to 8 times faster than with slice-based CT, respiratory patient motion may still cause inconsistencies in the data set,
- the emitted X-rays that fan out in a cone shaped way are still underutilized, since there is only one (or in the more recent models, a few)
linear detectors array that receive them (those rays are not used for image formation and die out in the periphery contribute to the patient dose),

- 1D image data coming from the same gantry orientation, but from a different coil of the helix, are not temporally related (this makes it difficult to set up a protocol for gated imaging of moving physiological structures).

1.1 The cone beam approach

In term of data capture, using a one dimensional detector is a very inefficient procedure resulting in a large waste of X-ray photons. Basic conclusion arise that whole 3D data set should be acquired within one spin around the patient. It is possible to realize using 2D detectors. Additionally, the natural shape of an X-ray beam emitted from a point source is a cone. The task is therefore, to reconstruct a volume from cone beam projections and it proves to be of a great scientific interest. In this case many more of the photons are free to penetrate the object, and because all image data in 2D image now originate from the same time instance, it enables easy gated imaging, and also provides an opportunity to use image-based methods for 3D reconstruction.

In other words, if instead of a system with $N$ detectors as in the 1D detector array, it is used a system with $N \times N$ detectors in a 2D detector array, thus in principle, the data collection is $N$ times faster for the same signal-to-noise ratio.

1.2 The FDK algorithm as a representative of the analytical methods

The Feldkamp algorithm is most commonly used in cone beam tomography [2]. This algorithm belongs to the class of approximate reconstruction algorithms since, the cone beam projections are acquired on a circle and do not assure a complete coverage of the Fourier space. The success of the Feld-
Feldkamp algorithm stems from the fact that it is easy to implement, since it is a generalization of the reconstruction formula for fan beam and it only requires a circular source path.

The reconstruction formula used by Feldkamp can be written as follows:

$$ f(t, s, z) = \frac{1}{2} \int_0^{2\pi} \frac{D_{SO}}{(D_{SO} - s)^2} \int_{-\infty}^{+\infty} R_\beta(p, z) h(\frac{D_{SO}t}{D_{SO} - s} - p) \frac{D_{SO}}{\sqrt{D_{SO}^2 + z^2 + p^2}} dp d\beta $$  \hspace{1cm} (1.1)

$R_\beta(p, z)$ represents a cone beam projection taken at angle $\beta$ and expressed in a local coordinate system $(p, z)$. $D_{SO}$ corresponds to the source to origin distance. The projection data is weighted and then convolved with the ramp filter. The resulting pixel values are finally backprojected and weighted with $\frac{D_{SO}^2}{(D_{SO} - s)^2}$. A way to understand the shortcomings due to the circular source trajectory is to visualize the Radon data that can be derived from the projections (see figure 1.1).

One cone beam projection provides Radon values on the section of a sphere passing through the origin. This surface is also called an "umbrella", due to its particular shape. If projections are acquired over $360^\circ$, the Radon space will be filled inside a torus instead of a sphere as would be expected.
The missing data in the z direction causes a decreasing resolution in the z direction, when the image is reconstructed. The same reasoning can be applied to the complete trajectories in the previous paragraph to understand how they fill the Radon space.

In summary the recipe for the FDK reconstruction method is as follows [14]:

• weight the projection data,

• convolve the weighted projections with \( \omega e^{it} d\omega \) for all horizontal lines in the detector (the t-direction), which is the same as applying a ramp filter in the horizontal direction,

• backproject and weight each filtered projection over the 3D reconstruction volume.

1.3 The SART algorithm

This method has been invented by Anderson and Kak [21]. It connects advantages of other algebraic methods, which is described in chapter 2. This section presents properties which justify choosing the SART as method of interest.

In spite of the computational cost, algebraic methods have several advantages like:

• different rays geometry is easy to implement,

• it is possible in an easy way to provide a priori knowledge about the reconstructed object,

• less projections than for the analytical methods are required which is proved mathematically [31] (see section 2.3),

• metal artifacts are reduced [46],

• it is possible to handle detectors of variable size inside projections, provided that detectors geometry remains unchanged from a projection to another [34].
Among several types of algebraic methods the SART algorithm has been chosen, which connects advantages of other algebraic techniques, and proves to be efficient in implementation and the most accurate [21], [31], [1].

Considering these advantages and the constant development of computer power, although computationally demanding, SART could be a method of considerations, if provides better performances than other methods.

1.4 The purpose of this work

Having in mind all properties, and drawbacks of the algebraic reconstruction techniques, the purpose of this work have been formulated as follows.

The aim of this work is to invent possible way of efficient implementation of the three dimensional SART and investigate the performance of this method comparing with other algorithms, first of all, in case of limited number of projections and different source/detector trajectories.
Chapter 2

Algebraic Reconstruction Techniques - ART

2.1 Basic concept

In the algebraic approach for tomographic imaging it is assumed that the reconstructed object (cross section in 2D CT or volume in 3D) consists of a matrix of unknowns, and then algebraic equations are solved for the unknowns in terms of measured projection data.

The matter of this work is three dimensional reconstruction, so it will be assumed that the reconstructed volume contains set of voxels

\[ f(x, y, z) \]

which are placed in cubic grid. In each cell the function \( f(x, y, z) \) is constant. Let \( f_j \) denote this constant value in the \( j \)th cell and let \( N \) be the total number of cells. A ray is a line running through the \((x, y, z)\)-volume. The projections will be represented as three dimensional matrix. Let \( p_{kl\Theta} \) be the ray sum placed in \( k \)-th row and \( l \)-th column of the 2D projection with angle \( \Theta \). The relationship between the \( f_j \) and \( p_{kl\Theta} \) may be represented as

\[ p_{kl\Theta} = \sum_{j=1}^{N} w_{jkl\Theta} f_j \]  \hspace{1cm} (2.1)

where \( w_{jkl\Theta} \) is a weighting factor that represents the contribution of the \( j \)th cell to the particular ray integral. The factor \( w_{jkl\Theta} f_j \) is equal to the fractional
area of the \( j \)th image cell intercepted by the ray-sum with index \( kl \Theta \). Most of the \( w_{jkl}f_j \)'s are zero since only a small number of cells contribute to any given ray-sum. This fact will be used to make algorithms faster.

If \( N \) and the number of ray-sums were small, it would be possible to use conventional matrix theory methods, to invert the system of equations in (2.1). In practice \( N \) may be a large number (in this case \( 64 \times 64 \times 64 = 262144 \)) and in most cases the number of ray-sums (called later as \( M \)) will also have the same magnitude. For these values of \( M \) and \( N \) the size of the matrix \([w_{ij}]\) in (2.1) is \( 262144 \times 262144 \) which precludes any possibility of direct matrix inversion.

When \( M \) and \( N \) have large values there exists iterative method for solving (2.1), based on the "method of projections" proposed by Kaczmarz [20], and later elucidated further by Tanabe [42].

To explain computational steps involved in these methods, we first write (2.1) in an expanded form [21]:

\[
\begin{align*}
    w_{11}f_1 + w_{12}f_2 &+ w_{13}f_3 + \cdots + w_{1N}f_N = p_1 \\
    w_{21}f_1 + w_{22}f_2 &+ w_{23}f_3 + \cdots + w_{2N}f_N = p_2 \\
    &\vdots \\
    w_{M1}f_1 + w_{M2}f_2 &+ w_{M3}f_3 + \cdots + w_{MN}f_N = p_M
\end{align*}
\]  

(2.2)

An image represented by \((f_1, f_2, \ldots, f_N)\), may be considered to be a single point in an \( N \)-dimensional space. In this space each of the above equations represents a hyperplane. When a unique solution to these equations exists, the intersection of all these hyperplanes is a single point giving that solution.

For the computer implementation of this method, initial guess of the solution is made. This guess, denoted by \( f_1^{(0)}, f_2^{(0)}, \ldots, f_N^{(0)} \) is represented vectorially by \( \vec{f}^{(0)} \) in the \( N \)-dimensional space. In this work, it has been assigned a value of zero to all the initial \( f_i \)'s. This initial guess is projected on the hyperplane represented by the first equation in (2.2) giving \( \vec{f}^{(1)} \). \( \vec{f}^{(1)} \) is projected on the hyperplane represented, by the \( i \)th equation to yield \( \vec{f}^{(2)} \) and so on. When \( \vec{f}^{(i-1)} \) is projected on the hyperplane represented, by the \( i \)th equation to yield \( \vec{f}^{(i)} \), the process can be mathematically described by
\[ f^{(i)} = f^{(i-1)} - \left( \frac{f^{(i-1)} \cdot \overrightarrow{w}_i - p_i}{\overrightarrow{w}_i \cdot \overrightarrow{w}_i} \right) \overrightarrow{w}_i \]  

(2.3)

where \( \overrightarrow{w}_i = (w_{i1}, w_{i2}, ..., w_{iN}) \) and \( \overrightarrow{w}_i \cdot \overrightarrow{w}_i \) is the dot product of \( \overrightarrow{w}_i \) with itself.

In applications requiring a large number of views and where large-sized reconstructions are made, the difficulty with using (2.2) can be in the calculation, storage, and fast retrieval of the weight coefficient [21].

To get around the implementation difficulties, caused by the weight coefficient, several algebraic approaches have been suggested, many of which are approximations to (2.3). To discuss some of the more implementable approximations, we first recast (2.3) in a different form:

\[ f_j^{(i)} = f_j^{(i-1)} + \frac{p_i - q_i}{\sum_{k=1}^{N} w_{ik}^2} w_{ij} \]  

(2.4)

where

\[ q_i = f^{(i-1)} \cdot \overrightarrow{w}_i = \sum_{k=1}^{N} f_k^{(i-1)} w_{ik} \]  

(2.5)

These equations say that when we project the \((i-1)\)th solution onto the \(i\)th hyperplane (\(i\)th equation in (2)) the gray level of the \(j\)th element whose current value is \(f_j^{(j-1)}\), is obtained by correcting its current value by \(\Delta f_j^{(i)}\), where

\[ \Delta f_j^{(i)} = f_j^{(i-1)} - \frac{p_i - q_i}{\sum_{k=1}^{N} w_{ik}^2} \]  

(2.6)

While \( p_i \) is the measured ray-sum along the \(i\)th ray, \( q_i \) may be considered to be the computed ray-sum for the same ray based on the \((i-1)\)th solution for the image gray levels. The correction \(\Delta f_j\) to the \(j\)th cell is obtained by first calculating the difference between the measured ray-sum and the computed ray-sum, normalizing this difference by \(\sum_{k=1}^{N} w_{ik}^2\), and then assigning this value to all the image cells in the \(i\)th ray, each assignment being weighted by the corresponding \(w_{ij}\).

### 2.2 Kinds of ART

Iterative methods consists necessarily of four major steps [41]:

- assumption of the test field,
calculation of correction,

• application of correction,

• test for convergence.

These algorithms differ in the manner in which corrections are applied and presented in brief below.

Simple ART

Mayinger [29] has suggested the simplest possible iterative reconstruction algorithm which in many ways resembles the algebraic reconstruction algorithm. Let \( p_{i\Theta} \) be the projection due to the \( i\Theta \)th ray with angle of irradiation \( \Theta \) and \( \hat{f}_j \) be the initial guess of the field value. We compute the approximation projection \( \hat{p}_{i\Theta} \) using the test field as

\[
\hat{p}_{i\Theta} = \sum_{j=1}^{N} w_{i\Theta,j} \hat{f}_j \quad i\Theta = 1, 2, \ldots, M_{\Theta}
\]  

(2.7)

where \( i\Theta \) denotes the \( i \)th ray of an irradiation with angle \( \Theta \), and \( 1 \leq i\Theta \leq M_{\Theta} \). The subsequent steps are as follows.

• For each angle of radiation \( \Theta \)

  1. For each ray \( i\Theta \) calculate the correction \( \Delta p_{i\Theta} = p_{i\Theta} - \hat{p}_{i\Theta} \)

  2. Compute the total value of the weight function along each ray as \( W_{i\Theta} = \sum_{j=i}^{N} w_{i\Theta,j} \)

  3. Calculate the average value of correction

\[
\frac{\Delta p_{i\Theta}}{W_{i\Theta}} = \frac{\Delta p_{i\Theta}}{W_{i\Theta}}
\]  

(2.8)

4. Repeat Steps 1-3 for all rays.

5. Apply a correction for each cell \( j \) of the test field as

\[
f_{j}^{new} = f_{j}^{old} + \lambda \Delta p_{i\Theta}
\]  

(2.9)

where \( \lambda \) is a relaxation factor.
6. Repeat step 5 for all the rays of the irradiation with angle $\Theta$

- Update the approximate projection using equation (2.7).
- Repeat the above procedure for all angles of irradiation. This completes the $k$th global iteration.

**Gordon ART**

The ART algorithm originally proposed for CT applications by Gordon et al. [41] is considered. In this method corrections are applied to all the cells through which the $i$th ray passes, before calculating the correction for the next ray. Hence the number of rays per angle of irradiation is not important, the approximate projection data is computed as

$$\hat{p}_i = \sum_{j=1}^{N} w_{ij} f_j \quad i = 1, 2, ..., M$$  \hspace{1cm} (2.10)

where $M$ is the grand total number of rays and $N$ is the number of cells. The remaining part of the algorithm can be stated as follows.

- For each iteration $k$
  
  1. For each ray $i$ calculate the correction $\Delta p_i = p_i - \hat{p}_i$
  2. Compute the correction coefficient $\alpha_i = \sum_{j=1}^{N} w_{ij}^2$
  3. Apply a correction to each cell $j$ of the test field through which the present ray passes as

$$\hat{f}_{j}^{\text{new}} = \hat{f}_{j}^{\text{old}} + \lambda \frac{w_{ij} \Delta p_i}{\alpha_i}$$  \hspace{1cm} (2.11)

  4. Repeat steps 1-3 for all the rays. This completes the $k$th iteration.

- Calculate again the approximate projection using equation (2.10)
Gilbert ART

Gilbert has developed independently a form of an ART, called the simultaneous iterative reconstruction technique (SIRT). In SIRT, the elements of the field function are modified after all the correction values corresponding to individual rays have been calculated. The algorithm is similar to ART but the correction is applied as given below.

• For each iteration $k$

1. For each ray $i$ calculate the correction $\Delta p_i = p_i - \hat{p}_i$
2. Compute the correction coefficient $\alpha_i = \sum_{j=1}^{N} w_{ij}^2$
3. Repeat steps 1 and 2 to all the rays.

• Identify all the rays ($Nc_j$) passing through a given cell and the corresponding $w_{ij}$ and $\Delta p_i$.

• For each cell $j$ apply the algebraic sum of all possible correction terms as

$$\mathcal{F}_{j}^{\text{new}} = \mathcal{F}_{j}^{\text{old}} + \sum_{ic=1}^{Nc_j} \lambda w_{ij}^2 \frac{\Delta p_i}{\alpha_i}$$

This completes the $k$th iteration.

Simultaneous ART - SART

Anderson and Kak [21] have proposed a new algorithm, simultaneous ART (SART) which combines the ART and SIRT algorithms. It was found to be very efficient and superior in implementation. The method of applying a correction is similar to simple ART but the structure is similar to SIRT. The algorithm is as follows.

• For each angle of radiation $\Theta$

1. For each ray $i\Theta$ calculate the correction $\Delta p_{i\Theta} = p_{i\Theta} - \hat{p}_{i\Theta}$
2. Compute the correction coefficient $\alpha_{i\Theta} = \sum_{j=1}^{N} w_{i\Theta,j}^2$
3. Apply a correction to each cell $j$ of the test field as

$$ f_{new}^j = f_{old}^j + \lambda w_i \Theta \Delta p_i \Theta $$

(2.13)

Here $\lambda$ is the relaxation factor.

4. Repeat Step 3 to all rays of the irradiation with angle $\Theta$.

- Calculate the new value of the approximate projection using equation (2.7).

- Repeat the above procedure to all angles of irradiation. This complete the $k$th global iteration.

**Multiplicative ART**

The correction strategies presented above are called additive art (or simply ART). When the correction is multiplicative, the ART is called multiplicative ART (MART). The initial approximate projection is computed using equation (2.10). The MART algorithms considered in the present study are as follows.

- For each iteration $k$

1. For each ray $i$ calculate the approximate projection $\bar{p}_i$

2. Identify all the rays passing through a given cell (the total number of rays per cell being $N_c_j$) and corresponding $i$, $w_{ij}$, $p_i$, and $\bar{p}_i$

3. For each cell $j$ compute the product of all possible correction terms. This can be accomplished in three different ways as

$$ MART1 : \quad f_{new}^j = f_{old}^j \times \prod N_c_j (1 - \lambda (1 - \frac{p_i}{\bar{p}_i})) $$

(2.14)

$$ MART2 : \quad f_{new}^j = f_{old}^j \times \prod N_c_j (1 - \lambda w_{ij} (1 - \frac{p_i}{\bar{p}_i})) $$

(2.15)

$$ MART3 : \quad f_{new}^j = f_{old}^j \times \prod N_c_j (\frac{p_i}{\bar{p}_i})^{\lambda w_{ij}} $$

(2.16)

This completes the $k$th iteration.
Summary

In this section types of algebraic methods has been presented. All algorithms considered here show a systematic behaviour with respect to the number of projections, view angle and noise level [41].

The MART method has the advantage that, in contrast to ART, a voxel can never be corrected to values less then zero (which is not in the solution space). However, MART has never reached the popularity of ART, and the original authors found that while ART seems to minimize the overall variance of the reconstruction, MART maximize the entropy [31].

The research of Mueller [31] has indicated, that it was probably not the image-based SART correction procedure itself that has prevented the noise-like artifacts from occuring, rather, it was Andersen and Kak's use of a better interpolation kernel, i.e. the bilinear function, as a voxel basis. Still SART has many advantages, due to its image-based approach:

- It proves to be better method for cone beam reconstruction,
- It lends itself very well for a graphics hardware-accelerated approach.

However SART is slightly slower than ART in software, due to the voxel-based pooling of correctional updates [31].

2.3 Why ART needs fewer projections than FB

This question was answered by Guan and Gordon [17] for the 2D parallel beam case and it has been extended to 3D case by Mueller in [31]. It will be cited here to show possibility of using SART in case of limited number of projections.

Usually one reconstructs on a square voxel grid with a sidelength of $n$ voxels, thus the number of grid voxels $N = n^2$. Also, we generally assume a circular reconstruction region. Voxels outside this region may be ignored. In this case we have $(1/4) \pi n^2$ unknown voxels values and $n$ pixels per 1D
image. For equation system (2.2) to be determined the number of projection images $M_{\text{ART}}$ has to be:

$$M_{\text{ART}} = \frac{(1/4)\pi n^2}{n} = \frac{\pi n^2}{4} = 0.785 \cdot n$$

(2.17)

Let’s see how many projections are needed for Filtered Backprojection (FB). The sampling interval in Fourier space is at least $\Delta \omega = 1/nT_g$, and the maximum frequency is given by $\omega_{\text{max}} = 1/(2T_g)$. Due to polar sampling in frequency space, the density of samples decreases as we go outward in the polar grid. To ensure a sampling rate of at least $\Delta \omega$ everywhere in the polar grid, even at the boundary, the angular spacing between the projections (i.e., the Fourier slices) in frequency space needs to be:

$$\Delta \varphi_P = \frac{\Delta \omega}{\omega_{\text{max}}} = \frac{2T_g}{nT_g} = \frac{2}{n}$$

(2.18)

In order to provide adequate sampling in the periphery, one must oversample in the interior frequency regions. The number of projections $M_{FB}$ is then:

$$M_{FB} = \frac{\pi}{\Delta \varphi_P} = \frac{\pi n}{2} = 1.57 \cdot n$$

(2.19)

Thus, $M_{\text{ART}} = M_{FB}/2$.

In 3D reconstruction, one usually reconstructs on a cubic voxel grid, again with a sidelength of $n$ voxels. Thus the number of grid voxels $N = n^3$. Also, for a 3D single-orbit reconstruction we generally assume a spherical reconstruction region. In this case we have $(1/6)\pi n^3$ unknown voxel values and $(1/4)\pi n^2$ relevant pixels per image. For the equation system (2.2) to be determined, the number of projection images $M_{\text{ART3D}}$ has to be:

$$M_{\text{ART3D}} = \frac{(1/6)\pi n^3}{(1/4)\pi n^2} = 0.67n$$

(2.20)

This means that for $n = 128$, a total of 86 projections is required.

The widely accepted rule of thumb is that FB requires at least $n$ projection images for good image quality. The previous calculation showed that ART requires about half of that. As a matter of fact, this is true for all algebraic methods, including SART.
Chapter 3

3D SART method

Three dimensional iterative reconstruction requires relatively big number of computations, so from practical reasons, on the first stage of work two dimensional SART reconstruction for parallel projections has been studied based on [21] and [1].

Algorithm has been implemented under IDL environment. Furthermore to speed up the reconstruction process SART has been coded in C and run under Linux. In further steps the geometry of the rays was changed from parallel beam to fan beam and finally to cone beam geometry.

3.1 2D reconstruction for parallel beam geometry

As mentioned in chapter 2, ART reconstruction is made through solving the system of linear equations. In equation (2.1) projection data were modeled by

\[ \displaystyle p_{k\ell \Theta} = \sum_{j=1}^{N} w_{j k \ell \Theta} f_j \]  

(3.1)

The object function is obtained for set of ray-sums and coefficients which describe the rays. This projection model gives good results, however it is very important that \( w_{j k \ell \Theta} \) coefficients should be computed with high accuracy, which in practice is hard to realize.
To seek alternative methods for modeling the projection process, the relationship between a continuous image and the discrete projection data can be expressed by the following general form

\[ p_i = R_i f(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) \delta(r_i(x, y)) \, dx \, dy \]  

(3.2)

where

\[ r_i(x, y) = 0 \]

is the equation of the \( i \)th ray and \( R_i \) is the projection operator along that ray. The integral on the right-hand side serves as the definition of the projection operator.

Now suppose we assume that in an expansion for the image \( f(x, y) \), we use basis functions \( b_j(x, y) \) and that a good approximation to \( f(x, y) \) is obtained by using \( N \) of them. This assumption can be represented mathematically by

\[ f(x, y) \approx \hat{f}(x, y) = \sum_{j=1}^{N} g_j b_j(x, y) \]  

(3.3)

where \( g_j \)'s are the coefficients of expansion; they form a finite set of numbers which describe the image \( f(x, y) \).

Substituting (3.3) in (3.2), we can write the forward process

\[ p_i = R_i f(x, y) \approx R_i \hat{f}(x, y) = \sum_{j=1}^{N} g_j R_i b_j(x, y) = \sum_{j=1}^{N} g_j a_{ij} \]  

(3.4)

where \( a_{ij} \) represents the line integral of \( b_j(x, y) \) along the \( i \)th ray. This equation has the same basic form as equation (2.1) yet it is more general in the sense that \( g_j \)'s aren’t constrained to be image gray level values over an array of points. This means that, the basis function can be a voxel, but using other basis function could be for example bilinear (in case of 3D trilinear) elements [21], [31] or blobs [28].

In this case the form here reduces to equation (2.1) because the following voxel basis function is used that is obtained by dividing the image frame into \( N \) identical sub-cubes; these are referred to as voxels and identified by the index:

\[ b_j(x, y) = \begin{cases} 
1 & \text{inside the } j \text{th voxel} \\
0 & \text{everywhere else.} 
\end{cases} \]  

(3.5)
In keeping with the nature of $f_j$'s in (2.1), $g_j$'s with these basis functions represent the average of $f(x,y)$ over the $j$th voxel and $R_ib_j(x,y)$ represents the length of the intersection of the $i$th ray with the $j$th voxel. Although equation (3.2) implies rays of zero width, if it is going to be considered a finite width with each ray, the elements of the projection matrix will represent the areas of intersection of these ray strips with the pixels.

In SART [21], superior reconstructions are obtained using more accurate than described briefly above voxel basis function model of the forward projection process. This is done by using bilinear elements which are the simplest higher order basis functions. In this work two dimensional interpolation has been expanded to 3D case and instead of bilinear, trilinear interpolation has been used. It can be shown that the $g_j$'s appearing in (3.3) for the case of trilinear elements are the sample values of the image function $f(x,y)$ on a cubic lattice. It can further be shown that whereas the voxel basis leads to discontinuous image representation, the trilinear elements allow a continuous form of $f(x,y)$ to be regenerated for computation. However, finding the exact ray integrals across such trilinear elements (as called for by $R_ib_j(x,y)$ in equation (3.4) for a large number of rays is a time-consuming task and approximation will be used.

Rather then try to find separately the individual coefficients $a_{ij}$ for a particular ray, we approximate the overall ray integral $R_if(sim)$ by a finite sum involving a set of $M_i$ equidistant points $\hat{f}(sim)$, for $1 \leq m \leq M_i$.

$$p_i = \sum_{m=1}^{M_i} \hat{f}(sim) \Delta s. \quad (3.6)$$

The value $\hat{f}(sim)$ is determined from the values $g_j$ of $f(x,y)$ on the eight neighboring points of the sampling lattice, i.e., by trilinear interpolation. It can be written

$$\hat{f}(sim) = \sum_{j=1}^{N} d_{ijm}g_j \quad \text{for } 1, 2, \ldots, M_i. \quad (3.7)$$

The coefficient $d_{ijm}$ is therefore the contribution that is made by the $j$th image sample to the $m$th point on the $i$th ray. Combining equations (3.6) and (3.7), we obtain an approximation to the ray integral $p_i$ as a linear
function of the image samples \( g_j \):

\[
p_i = \sum_{m=1}^{M_i} \sum_{j=1}^{N} d_{ijm} g_j \Delta s
\]

\[= \sum_{j=1}^{N} \sum_{m=1}^{M_i} d_{ijm} g_j \Delta s \quad \text{for } 1 \leq i \leq J \tag{3.9}\]

\[
= \sum_{j}^{N} a_{ij} g_j \tag{3.10}
\]

where the \( g_{ij} \) coefficients represent net effect of the linear transformations. They are determined as the sum of the contributions from different points along the ray:

\[
a_{ij} = \sum_{m=1}^{M_i} d_{ijm} \Delta s \tag{3.11}
\]

Therefore, \( a_{ij} \) is proportional to the sum of contributions made by the \( j \)th image sample to all the points on the \( i \)th ray. It is important to the overall accuracy of the model that for \( m = 1 \) and for \( m = M_i \), i.e., for the first and last points of the ray within the reconstruction region, the weights are adjusted so that \( \sum_{j=1}^{N} a_{ij} \) equals the actual physical length \( L_i \).

For such computed coefficients SART reconstruction can be carried out by using formula (2.13) mentioned in chapter 2.

### 3.2 Cone beam geometry

In case of cone beam we should take into account that rays are not parallel anymore. In this work ray integral is computed considering one ray integral going from the source to the center of the particular detector cell. The virtual detector has been assumed (similarly like for the projection simulator [33]) meaning that it lies in the center of the trajectory.

First, it is determined for which ray-sum computations will be done. Particular projection is stored in two dimensional matrix, so it is needed to know values of two indexes. For those values - \( \alpha \) and \( \beta \) angles are computed, which describe the direction of the ray (line which goes from source to the
center of the detector cell) according to the line connecting source and the center of the trajectory (in this case also center of the detector).

\[
\alpha = \arctan\left(\frac{y}{h_{SD}}\right) \tag{3.12}
\]

\[
\beta = \arctan\left(\frac{z}{h_{SD}}\right) \tag{3.13}
\]

where \(h_{SD}\) is the source-detector distance and \(y\) and \(z\) represents position of the ray-sum in the projection matrix. Next, \(\Delta x\) step for the points is computed, which depends on \(\alpha\) and \(\beta\).

\[
\Delta x = \Delta s \cdot \cos(\alpha) \cdot \cos(\beta) \tag{3.14}
\]

where \(\Delta s\) is the distance between two equidistant points. For known value of \(\Delta x\) position of the equidistant point is computed as follows

\[
x = \text{point} \cdot \Delta x - \text{detsiz}/\cos(\alpha) \cdot \cos(\beta) \tag{3.15}
\]

where \(\text{point}\) is equidistant point index, and \(\text{detsiz}\) is a detector size in voxels. Additionally to reduce ”ring like” artifacts angular shifting, positions of the equidistant points is made:

\[
x = (\text{point} + 0.5 \cdot \frac{\text{proj}}{\text{npr}}) \cdot \Delta x - \text{detsiz}/\cos(\alpha) \cdot \cos(\beta) \tag{3.16}
\]

where \(\text{proj}\) is a projection index and \(\text{npr}\) is a number of projections. Knowing \(x\) it is founded position \(z\) and \(y\) for all equidistant points according to the formula:

\[
y_{\text{cone}} = y + x \tan(\alpha) \tag{3.17}
\]

\[
z_{\text{cone}} = z + x \tan(\beta) \tag{3.18}
\]

Finally, rotation with angle \(\theta\) (projection angle) is made:

\[
y' = y_{\text{cone}} \cos(\theta) - x \sin(\theta) \tag{3.19}
\]

\[
x' = y_{\text{cone}} \sin(\theta) + x \cos(\theta) \tag{3.20}
\]

In this way positions for all equidistant points inside the reconstruction region are computed, and through trilinear interpolation \(w_{ij}\) coefficients for eight neighboring voxels are computed.
3.3 Weight coefficient matrix compression

Direct implementation of the SART method described in [21], [1] requires computational power which in 3D case makes SART impractical. For example, considering set of 120 2D projections containing $64 \times 64$ ray sums and reconstructed volume $64 \times 64 \times 64$ w coefficient matrix should be in size equal to:

$$\text{sizeof}(w_{ij}) = 64 \times 64 \times 120 \times 64 \times 64 \times 64 \quad (3.21)$$

There are however some properties, which could be taken into account to speed up the reconstruction process, without any loss of accuracy.

Considering (2.2) for each ray-sum $p_i$, we need to compute contribution of each voxel in reconstruction error. In reality however, only a little part of all voxels make significant contribution to the value of particular $p_i$. Most of the $w_{ij}$ coefficients have value 0 and does not change final results, making however computation time longer.

Main conclusion appears that computations of particular $p_i$ value should be made taking into account only nonzero values. In order to make it possible, it is desired to know positions of the none-zero $w_{ij}$ coefficients. Finally it can be written that for efficient implementation of SART, following information is required:

- values of nonzero coefficients
- positions of nonzero coefficients

To proceed approximation of integration along the ray, set of equidistant points is taken into account, and for for each point though trilinear interpolation contribution of eight neighboring voxels to the attenuation coefficient in each equidistant point is computed. The maximal number of non zero values is:

$$mn = ns \times 8 \quad (3.22)$$

where $mn$ is the maximal number of nonzero $w_{ij}$ coefficients for particular $p_i$ and $ns$ is the number of equidistant points along the ray related with $p_i$ inside the reconstruction region.
Due to overlapping (particular voxels usually contribute to more then one point) the number of nonzero $w_{ij}$ coefficients is much less then maximal value, because the positions of nonzero coefficients are stored two times. In the second step positions, which occur two times in the matrix due to overlapping are detected and stored once. Finally two vectors are obtained, which contain desired, mentioned above information. For better understanding of this procedure appendix of this work includes source codes of IDL and C implementations.

3.4 The SART method with helical trajectory

As mentioned at the beginning of this thesis basic advantage of ART type algorithms is the fact that any type of the rays geometry can be implemented in easy way. In this section it is shown, how described in previous sections algorithm can be easily adapted for helical cone beam CT.

Additional translation in $z$ direction is made for every turn of the source/detector system. It is described as follows:

$$z = z_o + \frac{transl}{nofpr} \cdot proj \tag{3.23}$$

where $z_o$ is the initial position in direction of $z$ axis, $transl$ is a translation value per turn and $proj$ is a projection index among $nofpr$ projections.

In this experiment 120 projections has been used along the 360 degree arc. The trajectory has been helical with translation equal 5 voxels per turn. The object has been a Shepp-Logan head phantom described in detail in [21].

The results of the reconstruction are shown in the figure 3.1.

The quality of the reconstruction is improved comparing with circular trajectory. However, on the boundaries of the object, artifacts occur due to the long object problem. They are the result of the fact that in case of translation there are places which are not irradiated around full arc.
3.5 Reconstruction of the SPECT data

Algebraic reconstruction techniques are widely used for reconstruction of Single Photon Emission Tomography (SPECT) [44], [25].

To test properties of the described method, two sets of data have been chosen. They have been obtained in the Department of Nuclear Medicine of the University Hospital in Patras. For both attenuation correction has been made through adding of the opposite oriented projection from two gamma cameras. The results of the reconstruction are presented in the figure 3.2 and 3.3. The first one is a reconstruction of the brain activity. The second one is a reconstruction of the spine and its surface is presented in 3D form. Both reconstructed volumes are visualized using IDL.
Figure 3.3: The spine. Visualization of the reconstructed SPECT data using IDL.
Chapter 4

Comparison of 3D SART and FDK

4.1 Generation of projections

In order to analyze any reconstruction method a set of projections as an input data is desired. In this study the source of projections has been not a real tomographic unit, but a software [33] [26], which has simulated the process of acquiring images.

Projection images for three different phantoms has been generated considering an isocentric rotation over 360° at an angular step of (3°, 4°, 6°, 8°, and 10°). The cone angle was limited to 10°, which has been shown to provide acceptable errors in the case of radiotherapy applications [10]. The reconstruction artifacts has been evaluated using a phantom containing 7 high contrast ellipsoids [14] (figure 4.1). Furthermore, the Shepp-Logan (figure 4.3) and a bar pattern phantom with line pairs varying in size from 9.14 to 32 (lp/detector size) has been used (figure 4.2). The bars were considered to have a thickness of 2 voxels and to be inclined 45° with respect to the source detector plane [3].
4.2 Objective image characteristics

As mentioned in previous section several kinds of phantoms has been generated in order to make evaluation of particular properties.

The following objective image characteristics has been studied for both methods [9]:

**Spatial Resolution** was assessed using the bar pattern phantom and by means of Square Wave Response Function (SWRF), determined over the complete range of spatial frequencies. Reconstruction using different angular steps were considered. The stopping criterion was determined on the bases of the following condition:

\[
\frac{1}{N} \sum_{j=1}^{N} \left| f_j^{(p)} - f_j^{(p+1)} \right| \leq \frac{1}{N} \sum_{j=1}^{N} |f_j^{(p)}| \leq 0.1 \tag{4.1}
\]

where \( f_j \) is the reconstructed volume function with number of voxels equal \( N \). The value 0.1 has been arbitrary chosen as the best trade off
between quality and speed. The number of 6 iterations proved to fulfill this criterion.

**Root Mean Square (RMS) error** has been computed for the 3D reconstructions of the Shepp-Logan phantom using different angular steps ($3^\circ$, $6^\circ$, $10^\circ$), using the following formula:

$$RMS = \frac{1}{N} \sqrt{\frac{1}{N} \sum_{j=1}^{N} (f_j - \bar{f}_j)^2} \quad (4.2)$$

where the symbols are explained above.
4.3 Results

Figures 4.8 and 4.9 present the SWRF plots for FDK and SART (6 iterations) reconstruction when 3 different degree steps (3°, 6°, 10°) were used. Plots of RMS errors for both for the FDK and SART reconstructions are shown in figure 4.10. A visual insight about the low contrast capabilities for both techniques is given by figures 4.4 and 4.5 showing the central slices reconstructed with a degree step of 6 degrees through the Shepp and Logan phantom. Windowing from 0.98 to 1.05 was applied for better inspection. Images affected by artifacts due limited number of projections (degree step 10) are shown in figures 4.6 and 4.7 for the ellipsoids phantom. For this purpose, the windowing was in range -0.5 to 1.5.

SART technique proved to provide better results than FDK in terms of spatial resolution (figures 4.8 and 4.9), RMS (figure 4.10) and less artifacts.
(figures 4.6 and 4.7) when a limited number of projections were used. However, a visual inspection of tomograms reconstructed through the Shepp and Logan phantom and containing low contrast structures, shows better results for the FDK, due to noise occurred in SART reconstructions and caused by discretization errors. Globally (according to RMS) SART produce more accurate reconstructions in comparison with FDK even after three iterations for all degree steps. A higher attenuation of low frequencies occurs for FDK than for SART when less and less projections are used as noticed in figure 4.8 and 4.9. On the other hand, the SART method involves repeated projection and backprojection operations and requires more reconstruction time compared with FDK.
Figure 4.8: SWRF plots for FDK using different number of projections.

Figure 4.9: SWRF plots for SART 6 iterations using different number of projections.
Figure 4.10: Illustrating root mean square reconstruction error versus number of projections for FDK and SART.
Chapter 5

Conclusions and further work

In the near future cone-beam equipment will be widely involved in 3D computed tomography, due to the described advantages. This type of beam geometry requires special extension for this type of ray tracing. In this work such extension has been presented with additional description of the fast computer implementation. Several experiments which has been made using simulated and real data and it can be concluded as follows.

Although slower than FDK algorithm, SART method proved to give better results and to be the technique of choice for 3D CBCT reconstruction when a limited number of projections were used. Such cases are of practical interest considering the dose saving as well as the faster acquisition in acquiring less projections. Although more computationally demanding, given the constant development of computer power and possibility of parallelisation, SART could be a method of interest for reconstruction using a limited number of projections.

It has been proven that comparing with analytical methods implementation of any type of beam geometry (particularly helical) for algebraic methods is a very easy task.

The method presented however has several drawbacks, which are the matter of further work. Using trilinear interpolation in case of reconstruction from cone beam projections is not enough for low contrast objects and using of other types of basic functions should be investigated.

According to [1] there is a compromise between noise and speed of con-
vergence. In this work to obtain as good quality as possible small angular distance between subsequent projections has been chosen. However according to [31] it is possible using another ordering schemes to speed up convergence without loss of accuracy.

Cone beam is a symmetric structure and it could be possible to use this property to speed up the reconstruction process.

Finally, the matter of further work is combining the SART method with tomosynthesis.
Bibliography


3D Simultaneous Algebraic Reconstruction...


Appendix - Source files

IDL three dimensional SART implementation

;This is the 3D SART implementation with cone beam ray geometry
; Wojciech Chlewicki
; Patra 2000

pro cone_3d
n_of_pr=65.0 ; Define number of projections
detsiz=64.0 ; Detector size [pixel]
detlen=2.0 ; Size of the original detector
deltas=1.0 ; Define delta s
detsizpow3=detsiz^3
radius=detsiz/2-0.5 ;Radius of the recon circle
print, 'radius', radius
radiussq=radius^2 ; Radius power 3 of the recon circle
print, 'radiussq', radiussq
recangle=(8.78*!dpi)/180; Reconstruction angle in radians
print, 'recangle [rad]', recangle
h=radius/tan(recangle/2) ; Distance between source and det [pixel]
print, 'h', h
astep=(3.0*!dpi)/180 ; Angle step
print,'angle step', astep
f3d=fltarr(detsiz^3)

for it= 0, 0 do begin ;iteration
for m=0, n_of_pr-1 do begin ;Index of projection
theta=m*astep
count=long(0)
ftemp=fltarr(detsiz^3)
denom=fltarr(detsiz^3)

; Read the projection
filn='C:\My Documents\wchlew\cone_beam\projections\s13d64_360_120'
filn2='.dat'
filn1=filn+strcompress(string(m))+filn2
restore, filename = filn1
help, image
image=(image*detsiz/detlen)/deltas;(ratio: recregsize/detsize)
image=rebin(image, 64, 64)

;Computation the way of rays
for o=0, detsiz-1 do begin ;Index of the ray position in z direction
  print, 'proj', m,' row o', o
for n=0, detsiz-1 do begin ;Index of the ray position in y direction
  w=fltarr(detsiz+1,detsiz+1,detsiz+1) ;wij coeff for one ray
  y=n-radius ; position of the ray on y axis
  z=o-radius ; position of the ray on z axis
  alpha=atan(y/h)
  beta=atan(z/h)
  deltax = delta*x*cos(alpha)*cos(beta)
  for point=0, detsiz/deltax do begin; Number of points along the ray
    x=point*deltax-(detsiz/(cos(alpha)*cos(beta)))/2
    yfan=y*x*tan(alpha)
    zfan=z*x*tan(beta)
    yprim=yfan*cos(theta)-x*sin(theta)
    xprim=yfan*sin(theta)+x*cos(theta)
    xyzsq=x^2+yfan^2
    cond=(xyzsq LE radiussq)*(abs(zfan) le radius)
    xprim=(xprim+0.5*detsiz/(cos(alpha)*cos(betha)))*cond
    yprim=(yprim+radius)*cond
    zprim=(zfan+radius)*cond
  end
end

:Interpolation
w(fix(xprim),fix(yprim),fix(zprim))=w(fix(xprim),fix(yprim),fix(zprim))+$
  (1-(xprim-fix(xprim)))*(1-(yprim-fix(yprim)))*(1-(zprim-fix(zprim)))*cond
w(fix(xprim)+1,fix(yprim),fix(zprim))=w(fix(xprim)+1,fix(yprim),fix(zprim))+$
  (xprim-fix(xprim))*(1-(yprim-fix(yprim)))*(1-(zprim-fix(zprim)))*cond
w(fix(xprim),fix(yprim)+1,fix(zprim))=w(fix(xprim),fix(yprim)+1,fix(zprim))+$
  (1-(xprim-fix(xprim)))*(yprim-fix(yprim))*(1-(zprim-fix(zprim)))*cond
w(fix(xprim)+1,fix(yprim)+1,fix(zprim))=w(fix(xprim)+1,fix(yprim)+1,fix(zprim))+$
  (xprim-fix(xprim))*(yprim-fix(yprim))*(1-(zprim-fix(zprim)))*cond
w(fix(xprim),fix(yprim),fix(zprim)+1)=w(fix(xprim),fix(yprim),fix(zprim)+1)+$
  (1-(xprim-fix(xprim)))*(1-(yprim-fix(yprim)))*(zprim-fix(zprim))*cond
w(fix(xprim)+1,fix(yprim),fix(zprim)+1)=w(fix(xprim)+1,fix(yprim),fix(zprim)+1)+$
  (xprim-fix(xprim))*(1-(yprim-fix(yprim)))*(zprim-fix(zprim))*cond
w(fix(xprim),fix(yprim)+1,fix(zprim)+1)=w(fix(xprim),fix(yprim)+1,fix(zprim)+1)+$
  (1-(xprim-fix(xprim)))*(yprim-fix(yprim))*(zprim-fix(zprim))*cond
w(fix(xprim)+1,fix(yprim)+1,fix(zprim)+1)=w(fix(xprim)+1,fix(yprim)+1,fix(zprim)+1)+$
  (xprim-fix(xprim))*(yprim-fix(yprim))*(zprim-fix(zprim))*cond
w(fix(xprim)+1, fix(yprim)+1, fix(zprim)+1)=w(fix(xprim)+1, fix(yprim)+1, fix(zprim)+1) +
(xprim-fix(xprim))*(yprim-fix(yprim))*(zprim-fix(zprim))*cond
endfor; for point

; W coeff compression
wcomp=fltarr(1000)
indexw=lonarr(1000)

; Find non zero val in
w=w[0:detsiz-1,0:detsiz-1,0:detsiz-1]
nonzerind=where(w, howmany)

if howmany gt 0 then begin
  sum2=0
  sum1=0
  for count_1= 0, howmany-1 do begin
    wcomp[count_1]=w[nonzerind[count_1]]
    indexw[count_1]=nonzerind[count_1]
    sum1=sum1+wcomp(count_1)*f3d(indexw(count_1))
    sum2=sum2+wcomp(count_1)
  end ; for count
end ; if howmany

for count_1= 0, howmany-1 do begin
  ftemp(indexw(count_1))=ftemp(indexw(count_1)) + wcomp(count_1)*((image(n,o)-sum1)/(sum2+(sum2 eq 0)))
  denom(indexw(count_1))=denom(indexw(count_1)) + wcomp(count_1)
end ; for count
endfor; for n
endfor; o

wth=0.2+(it eq 0)*(0.2*(0.6-0.4*cos(pi*m/(n_of_pr-1)))-0.2)
i=lindgen(detsiz^3)
f3d(i)=f3d(i)+wth*ftemp(i)/(denom(i)+(denom(i) eq 0))
endfor ; for m
endfor ; for it

f3d=reform(f3d, detsiz,detsiz,detsiz)
save, filename = 'C:\My Documents\chlewi\cone_beam\sl3d_65proj_1it.dat', f3d
window, 0, xsize=256, ysize=256, title='Reconstruction'
zoomed=congrid(f3d[*,*,31], 256, 256)
WSET, 0 & TVSCL, zoomed >0.99 <1.05
R = PROFILE(zoomed) ; Extract a profile from the image.
;Mark two points on the image with the mouse.
WINDOW, /FREE ;Create a new plotting window.
PLOT, R
end
C three dimensional SART implementation

/* Three dimensional SART implementation */
/* Wojciech Chlewicki */
/* Patra 2000 */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

main()
{
    /* Define variables */
    char *inf = "sl45_64_10.prj";
    char *outf1 = "sl45_10_1it.vox";
    char *outf2 = "sl45_10_2it.vox";
    char *outf3 = "sl45_10_3it.vox";
    char *outf4 = "sl45_10_4it.vox";
    char *outf5 = "sl45_10_5it.vox";
    char *outf6 = "sl45_10_6it.vox";
    char *outf7 = "sl45_10_7it.vox";
    char *outf8 = "sl45_10_8it.vox";
    char *outf9 = "sl45_10_9it.vox";
    char *outf10 = "sl45_10_10it.vox";

    float xprim, yprim, zprim, sum1, sum2, wth=0.07;
    float y, z, deltax, x, yfan, zfan;
    int max_x, max_y, max_z, point, howmany, loop;
    float alpha, betha, theta, xysq;
    float wcomp[1000];
    long indexw[1000];
    long indnonzer[2000];
    long count;
    int n_of_pr = 45;
    int detsiz = 64;
    float detsiz_sq=((float)detsiz*detsiz);
    float detlen = 2.1;
    float deltas = 1.0;
    long detsiz_pow_3=detsiz*detsiz*detsiz;
    long size_of_w=(detsiz+1)*(detsiz+1)*(detsiz+1);
    long actind, nzcount;
    long index1, index2, index3, index4, index5, index6, index7, index8;
    float radius=detsiz*0.5-0.5;
    float radius_sq = radius*radius;
    float rec_angle = 10.0*M_PI/180.0;
    float h=radius/tan(rec_angle*0.5); /* Distance between source and detector */
    float astep=8.0*M_PI/180.0; /* Angle step*/
    int order[121];
    int c, xbis, ybis, zbis;
    int proj, n, it, i, j, k, o, n;
    float sp1;
double h1;
FILE *input_file;
FILE *output_file;

float *ftemp;
float *denom;
float *w;
float *f3d;
float ***image;

/* Print data*/
printf("Cone-beam 3D reconstruction\n");
printf("detsiz_pow_3: %d\n",detsiz_pow_3);
printf("Distance between source and detector: ");
printf("%f\n",h);

/* Main program*/
/* Allocate memory */
/* for f3d */
if ( ( f3d=( float * )calloc(detsiz_pow_3,sizeof(float)) )==NULL)
   printf("Memory allocation for w fails\n");
/* for ftemp */
if ( ( ftemp=( float * )calloc(detsiz_pow_3,sizeof(float)) )==NULL)
   printf("Memory allocation for w fails\n");
/* for denom */
if ( ( denom=( float * )calloc(detsiz_pow_3,sizeof(float)) )==NULL)
   printf("Memory allocation for w fails\n");
/* for w */
/* Now w will be one-index matrix */
/* if ( ( w=calloc(detsiz+1,sizeof(int **)) )==NULL)
   printf("Memory allocation for w fails\n");
for (i=0;i<detsiz+1;i++)
{
   if ( (w[i]=(float ** )calloc(detsiz+1,sizeof(float*)))==NULL)
      printf("Memory allocation for w fails\n");
   for (j=0;j<detsiz+1;j++)
   {
      if ((w[i][j]=(float * )calloc(detsiz+1,sizeof(float)))==NULL)
         printf("Memory allocation for w fails\n");
   }
} */
/* for w */
if ( ( w=( float * )calloc(size_of_w,sizeof(float)) )==NULL)
   printf("Memory allocation for w fails\n");
/* Read projection file */
if ((input_file = fopen(inf, "r")) == NULL)
{
    fprintf(stderr, "Cannot open input file \n ");
}

/* Read three dimensions */
fread ((int*) &max_x, sizeof(int), 1,input_file);
fread ((int*) &max_y, sizeof(int), 1,input_file);
fread ((int*) &max_z, sizeof(int), 1,input_file);

/* Allocate memory for image */
if ( ( image=calloc(max_x,sizeof(int **)) )==NULL)
    printf("Memory allocation for w fails\n");
for (i=0;i<detsiz;i++)
{
    if ((image[i]=calloc(max_y,sizeof(float*)))==NULL)
        printf("Memory allocation for In[%u] fails\n",i);
    for (j=0;j<detsiz;j++)
    {
        if ((image[i][j]=calloc(max_z,sizeof(float)))==NULL)
            printf("Memory allocation for In[%u][%u] fails\n",i);
    }
}

/* Read float values */
for(k=0; k < max_z; k++)
{
    for(j=0; j < max_y; j++)
    {
        for(i=0; i < max_x; i++)
        {
            fread ((float*) &sp1, sizeof(float), 1,input_file);
            image[i][j][k]=(float)(sp1);
        }
    }
}
fclose(input_file);

printf("X size: ");
printf("%d\n",max_x);
printf("Y size: ");
printf("%d\n",max_y);
printf("Z size: ");
printf("%d\n",max_z);
printf("X\n",image[0][0][0]);
printf("X\n",image[31][3][0]);

/* Multiply image by (detsiz/detlen)*/
for(proj=0; proj < max_z; proj++)
{for(o=0; o < max_y; o+=1)
    {for(n=0; n < max_x; n+=1)
        {image[n][o][proj]=image[n][o][proj]*((float)(detsiz)/detlen)*4.0;
        }
    }
}

/* Iteration loop */
for(it=0; it < 10; it+=1)
{
    if (it > 0)
    {
        /* Read function file*/
        switch (it) {
            case 1:
                if ((input_file = fopen(outf1, "r")) == NULL)
                {
                    fprintf(stderr, "Cannot open input file \n" );
                }
                break;
            case 2:
                if ((input_file = fopen(outf2, "r")) == NULL)
                {
                    fprintf(stderr, "Cannot open input file \n" );
                }
                break;
            case 3:
                if ((input_file = fopen(outf3, "r")) == NULL)
                {
                    fprintf(stderr, "Cannot open input file \n" );
                }
                break;
            case 4:
                if ((input_file = fopen(outf4, "r")) == NULL)
                {
                    fprintf(stderr, "Cannot open input file \n" );
                }
                break;
            case 5:
                if ((input_file = fopen(outf5, "r")) == NULL)
                {
                    fprintf(stderr, "Cannot open input file \n" );
                }
                break;
        }
    }
}
case 6:
if ((input_file = fopen(outf6, "r")) == NULL)
{  
    fprintf(stderr, "Cannot open input file \n" );
}
break;
case 7:
if ((input_file = fopen(outf7, "r")) == NULL)
{  
    fprintf(stderr, "Cannot open input file \n" );
}
break;
case 8:
if ((input_file = fopen(outf8, "r")) == NULL)
{  
    fprintf(stderr, "Cannot open input file \n" );
}
break;
case 9:
if ((input_file = fopen(outf9, "r")) == NULL)
{  
    fprintf(stderr, "Cannot open input file \n" );
}
break;
} /* for switch */

/* if ((input_file = fopen(outf, "r")) == NULL) */
{  
    fprintf(stderr, "Cannot open input file \n" );
} /* */

/* Read three dimensions */
fread ((int*) &max_x, sizeof(int), 1,input_file);
fread ((int*) &max_y, sizeof(int), 1,input_file);
fread ((int*) &max_z, sizeof(int), 1,input_file);

/* Read float values */
for(k=0; k < max_z; k+=1)
{  
    for(j=0; j < max_y; j+=1)
    {  
        for(i=0; i < max_x; i+=1)
        {  
            fread ((float*) &sp1, sizeof(float), 1,input_file);
            f3d[(long)i+j*detsiz+k*detsiz*detsiz]=(float)(sp1);
        }
    }
}
fclose(input_file);
3D Simultaneous Algebraic Reconstruction...

} /* for if it >0 */

/* Projection loop */
for(proj=0; proj< n_of_pr; proj+=1)
/* for(proj=10; proj< 21; proj+=1) */
{
  m=proj;
  theta=(float)(m)*astep;
  count=0;
  for(i=0; i< detsiz_pow_3; i+=1)
    {ftemp[i]=0.0;
     denom[i]=0.0;
     }/*for i*/
  printf("Projection: %d\n",m);
  /* Computations the way of rays*/
  for(o=0; o < detsiz; o+=1)
    { /* printf("Proj: %d Row:%d \n ",m,o); */
     for(n=0; n < detsiz; n+=1)
       {
        y=(radius+0.25-0.5*proj/n_of_pr); /* Position of the ray on y axis */
        z=(radius+0.25-0.5*proj/n_of_pr); /* Position of the ray on z axis*/
  
        alpha =atan (y/h);
        beta =atan (z/h);
        deltaz=deltax*cos(alpha)*cos(beta);
  
  /* Travel along the ray */
  nzcount=0;
  for(point=0; point <(int)(detsiz/deltax); point+=1)
    { for (loop=0; loop < 4; loop+=1)
      {
        x=(point+0.25*loop+0.25*proj/n_of_pr)*deltax-(detsiz/(cos(alpha)*cos(beta)))/2;
        yfan=y+x*tan(alpha);
        zfan=z+x*tan(beta);
        yprim=yfan*cos(theta)-x*sin(theta);
        xprim=yfan*sin(theta)+x*cos(theta);
        xysq=x*x+yfan*yfan;
        if (xysq <= radius_sq){
          if (abs((int)zfan) <= radius){
            xprim=xprim+0.5*detsiz/(cos(alpha)*cos(beta));
            yprim=(yprim+(radius+0.25-0.5*proj/n_of_pr));
            zprim=(zfan+(radius+0.25-0.5*proj/n_of_pr));
            xbis=((int)xprim);
            ybis=((int)yprim);
            zbis=((int)zprim);
            if (xbis < 64){
/* Interpolation */
index1=xbis+ybis*detsiz+zbis*detsiz_sq;
index2=(xbis+1)+ybis*detsiz+zbis*detsiz_sq;
index3=xbis+(ybis+1)*detsiz+zbis*detsiz_sq;
index4=(xbis+1)+(ybis+1)*detsiz+zbis*detsiz_sq;
index5=xbis+ybis*detsiz+(zbis+1)*detsiz_sq;
index6=(xbis+1)+ybis*detsiz+(zbis+1)*detsiz_sq;
index7=xbis+(ybis+1)*detsiz+(zbis+1)*detsiz_sq;
index8=(xbis+1)+(ybis+1)*detsiz+(zbis+1)*detsiz_sq;

w[index1]=w[index1]+(1-(xprim-xbis))*(1-(yprim-ybis))*(1-(zprim-zbis));
w[index2]=w[index2]+(xprim-xbis)*(1-(yprim-ybis))*(1-(zprim-zbis));
w[index3]=w[index3]+(1-(xprim-xbis))*(yprim-ybis)*(1-(zprim-zbis));
w[index4]=w[index4]+(xprim-xbis)*(yprim-ybis)*(1-(zprim-zbis));
w[index5]=w[index5]+(1-(xprim-xbis))*(1-(yprim-ybis))*(zprim-zbis);
w[index6]=w[index6]+(xprim-xbis)*(1-(yprim-ybis))*(zprim-zbis);
w[index7]=w[index7]+(1-(xprim-xbis))*(yprim-ybis)*(zprim-zbis);
w[index8]=w[index8]+(xprim-xbis)*(yprim-ybis)*(zprim-zbis);

indnonzer[nzcount]=index1;
indnonzer[nzcount+1]=index2;
indnonzer[nzcount+2]=index3;
indnonzer[nzcount+3]=index4;
indnonzer[nzcount+4]=index5;
indnonzer[nzcount+5]=index6;
indnonzer[nzcount+6]=index7;
indnonzer[nzcount+7]=index8;
nzcount=nzcount+8;

} /* for if xbis */
} /* for if z*/
} /* for if xyzsq*/
} /* for loop */
}/* for point*/

/* W coeff compression */

/* Find nonzero values in w */
for(i=0; i<howmany; i=i+1)
{wcomp[i]=0;
 indexw[i]=0;
}

howmany=0;
for(i=0; i < nzcount; i=i+1)
{
   if (w[indnonzer[i]] > 0)
   {
      wcomp[howmany]=w[indnonzer[i]];
      indexw[howmany]=indnonzer[i];
      howmany=howmany+1;
      w[indnonzer[i]] = 0;
   }
} /*for if*/
/* Reconstruction */
if(howmany>0)
{sum2=0;
 sum1=0;
 for(count=0;count<howmany;count+=1)
 {sum1=sum1+wcomp[count]*f3d[indexw[count]];
  sum2=sum2+wcomp[count];
 } /*for count*/

 for(count=0;count<howmany;count+=1)
 {actind=indexw[count];
  /*printf("actpos: %d\n",actind);*/
  ftemp[actind]=ftemp[actind]+wcomp[count]*((image[n][o][m]-sum1)/sum2);
  denom[actind]=denom[actind]+wcomp[count];
 } /*for count*/
}

 for (count=0;count<detsiz_pow_3;count+=1)
 {if (denom[count]>0)
  f3d[count]=f3d[count]+wth*ftemp[count]/denom[count];
 } /*for projection*/

} /*for n*/
} /*for o*/

/* Write to file */
switch (it) {
 case 0:
  if ((output_file = fopen(outf1, "w")) == NULL)
  {
   fprintf(stderr, "Cannot open output file \n ");
  }
  break;

 case 1:
  if ((output_file = fopen(outf2, "w")) == NULL)
  {
   fprintf(stderr, "Cannot open output file \n ");
  }
  break;

 case 2:
  if ((output_file = fopen(outf3, "w")) == NULL)
  {
   fprintf(stderr, "Cannot open output file \n ");
  }
}
break;

case 3:
if ((output_file = fopen(outf4, "w")) == NULL)
{
    fprintf(stderr, "Cannot open output file \n" );
}
break;

case 4:
if ((output_file = fopen(outf5, "w")) == NULL)
{
    fprintf(stderr, "Cannot open output file \n" );
}
break;
case 5:
if ((output_file = fopen(outf6, "w")) == NULL)
{
    fprintf(stderr, "Cannot open output file \n" );
}
break;
case 6:
if ((output_file = fopen(outf7, "w")) == NULL)
{
    fprintf(stderr, "Cannot open output file \n" );
}
break;
case 7:
if ((output_file = fopen(outf8, "w")) == NULL)
{
    fprintf(stderr, "Cannot open output file \n" );
}
break;
case 8:
if ((output_file = fopen(outf9, "w")) == NULL)
{
    fprintf(stderr, "Cannot open output file \n" );
}
break;
case 9:
if ((output_file = fopen(outf10, "w")) == NULL)
{
    fprintf(stderr, "Cannot open output file \n" );
}
break;

} /* for switch */

/* if ((output_file = fopen(outf, "w")) == NULL)
{  fprintf(stderr, "Cannot open input file \n ");  
}*/

/* Read three dimensions */
fwrite ((int*) &detsiz, sizeof(int), 1, output_file);
fwrite ((int*) &detsiz, sizeof(int), 1, output_file);
fwrite ((int*) &detsiz, sizeof(int), 1, output_file);

/* Write float values */
for(k=0; k < detsiz; k+=1)
{for(j=0; j < detsiz; j+=1)
  {for(i=0; i < detsiz; i+=1)
    {sp1=f3d[(long)i+j*detsiz+k*detsiz*detsiz];
     fwrite ((float*) &sp1, sizeof(float), 1, output_file);
    }
  }
}
fclose(output_file);

}/*for iteration*/

/* Free memory */
free(image);
free(ftemp);
free(denom);
free(w);

/* Free memory*/
free(f3d);
return 0;
}