# Models for biomedical image reconstruction based on Newton-Cotes formulae for integral approximations

**Daniel Heilper**

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE MASTER'S DEGREE

University of Haifa
Faculty of Social Sciences
Department of Computer Science

November, 2012

# Models for biomedical image reconstruction based on Newton-Cotes formulae for integral approximations


## By: Daniel Heilper

## Supervised by: Prof. Dan Gordon


THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE MASTER OF SCIENCE DEGREE


University of Haifa

Faculty of Social Sciences

Department of Computer Science


November, 2012

Approved by: _____ Date: _____

(Supervisor)

Approved by: _____ Date: _____

(Chairman of the M.A. Committee)

# Contents

# Models for biomedical image reconstruction based on Newton-Cotes formulae for integral approximations

**By: Daniel Heilper**

## Abstract

Biomedical image reconstruction techniques acquire physical measurements from some data acquisition process, such as CT, MRI, PET, and so on. The purpose of these techniques is to reconstruct the underlying image. In order to do so, the reconstruction techniques require some assumed representation for the underlying image. In two dimensions, the most prevalent method is the use of pixels, with the assumption that the value of the image is constant throughout the pixel; see Herman [14]. Alternative basis functions have been proposed and implemented; see [12, 39, 36], and also the "blob" model [29]. Furthermore, the literature on various algorithms for image reconstruction is very extensive [14, 30]. Some algorithms are based on the FFT, some are iterative, such as ART [32, 24], and some seek some optimization criterion [15, 9, 2]. However, even the "best" algorithms cannot derive more information from the measurements if the underlying model of data representation is lacking. This work contributes to the field of data representation by exploring new models. Since the approximation of line integrals is fundamental to many image reconstruction problems, it is proposed to use basis functions that are derived from standard integral approximations. Three well known rules for integral approximation are the rectangle rule, the trapezoidal rule, and Simpson's rule. These rules are examples of the Newton-Cotes formulas for numerical integration, which are based on Lagrange basis polynomials. The general approach will be to use the Lagrange polynomials (of one variable) as blending functions in order to produce basis functions of higher dimension; this is a well-known technique from the field of spline approximations. The basic blending functions will have a finite support. Blending functions for higher dimensions are obtained by taking all possible products of the one-dimensional blending functions. For example, if $f(x)$ and $g(x)$ are the two basic functions in one dimension, then the 2D functions obtained by blending will be $f(x)f(y)$, $f(x)g(y)$, $g(x)f(y)$ and $g(x)g(y)$. This approach can be extended to higher dimensions. For biomedical image

reconstruction, the main interest will be two and three dimensions, and also four dimensions for temporal image reconstruction.

**Keywords.** *Basis functions, biomedical image reconstruction, integral approximation, iterative algorithms, Newton-Cotes formulas, Pixels, Blobs.*

# List of Tables

# List of Figures

# List of notations and assumptions

- $N$ - Size of one dimension of the grid
- $A$- the weights Matrix
- $x$ - A 1D vector representation of the image.
- The image is square
- Grid pixels are squares

# List of Abbreviations

**ART** Algebraic reconstruction Technique

**CG** Conjugate gradient

**CT** Computed tomography

**DDA** Digital differential analyzer

**DICOM** Digital Imaging and Communications in Medicine

**EM** Expectation maximization

**EMAP** Expecation maximization with maximum a posteriori probability

**LBF** Linear basis function

**MB** Mega bytes

**MAP** Maximum a posteriori probability

**OG** Original grid

**OS** Operating system

**PBF** Pixel basis function

**PC** Personal computer

**PET** Positron emission tomography

**PGM** Portable grey-map

**QO** Quadratic optimization

**RAM** Random access memory

**ROI** Region of interest

**RBF** Radial basis function

**SG** Shifted grid

**w.l.o.g** Without loss of generality

# Chapter 1

# Introduction

## 1.1 Image basis functions for image reconstruction

Biomedical image reconstruction techniques acquire physical measurements from some specific data acquisition process, such as CT, MRI, or some other modalities. The purpose of these techniques is to reconstruct the underlying image. In order to do so, the reconstruction techniques require some assumed representation for the underlying image. In two dimensions, the most prevalent method is the use of pixels, with the assumption that the value of the image is constant throughout the pixel; see Herman[13]. In recent years, the alternative "blob" representation, introduced by Lewitt in 1990, has gained more and more popularity; see Lewitt [28, 27] and Matej and Lewitt [29]. Blobs are radially symmetric modifications of the Kaiser-Bessel window functions. In the one-dimensional case, they are bell-shaped and have a finite support. The value of the unknown image function is approximated by a linear combination of suitable shifts of the basis function. Blobs have also been successfully used in the area of electron tomography (ET)—see [34, 22, 21, 33, 3, 38, 20]. In order to get good results from blobs, it is necessary to determine several parameters, such as the radius of the finite support. This is not always simple; see for example Garduno and Herman [11]. Other basis functions have also been considered. Shieh and Byrne [36]consider the problem of reconstructing an image from limited Fourier data. The approximation used in [36] is taken as a linear combination of suitable shifts of the sinc function $\sin(\sigma x)/(\sigma x)$, where

$\sigma > 0$ is a parameter. The method developed in [36] is also applicable to other image reconstruction problems. Hanson and Wecksung [12] employ cubic B-splines as basis functions for image reconstruction. The computational cost involved in this approach is reduced by using look-up tables for the line (or strip) integrals. Splines have also been used successfully for signal and image processing—see Unser's tutorial [39]. In two-dimensional transmission computerized tomography (CT), the measurements are approximated by line or strip integrals through the medium, and the problem is to reconstruct the value of the (unknown) density function $f(x, y)$ from the measurements. For purposes of exposition, this proposal will concentrate on line integrals. The general framework of the reconstruction problem is to find some approximating function $g(x, y)$ such that:

- $g(x, y)$ is represented in some simple form as a linear combination of certain basis functions;
- the line integrals through the domain are a "good" approximation to the measurements.

In positron-emission tomography (PET), line integrals also need to be approximated. Here, the line integrals are evaluated on the line-of-response (LOR), which is the line along which two opposing photons move when a positron collides with an electron; see Defrise and Kinahan [7]. Blobs have also been used with PET; see, for example, Hu et al.[41]. The literature on various iterative reconstruction algorithms suitable for image reconstruction is extensive; see, for example, Byrne [7]. These algorithms presuppose that the data is approximated by some given model, such as pixels or blobs, obtain a system of equations, and then use some iterative technique in order to obtain an image that satisfies some optimization criterion. However, even the "best" algorithms cannot derive more information from the measurements if the underlying model of data representation is lacking. The purpose of this research is to contribute to the field of data representation by exploring new models. Since the approximation of line integrals is fundamental to the image reconstruction problem, we propose to use basis functions that are usually used for integral approximation. As known from elementary calculus, the three most used forms for integral approximation are the rectangle rule, the trapezoidal rule, and Simpson's rule. These three basic rules, which are detailed in the next section, are examples of the Newton-Cotes formulas for numerical integration; see, for example, [37]. The Newton-Cotes formulas, which are based on Lagrange basis polynomials, are a continuing source for new numerical methods, such as [25]. When the integral is

evaluated over a finite interval $[a, b]$, the interval is subdivided into sub-intervals of uniform length $h$. The error of the above-mentioned approximation methods are, respectively, $O(h)$, $O(h^2)$, and $O(h^4)$, so it is expected that higher-order approximations will produce better images, particularly in the presence of noise. One potential advantage of these models over blobs is that there are no parameters which need fine-tuning. The general approach will be to use the Lagrange polynomials (of one variable) as blending functions in order to produce basis functions of higher dimension; this is a well-known technique from the field of spline approximations. The basic blending functions will have a finite support. Blending functions for higher dimensions, are obtained by taking all possible products of the one-dimensional blending functions. For example, if in one dimension $f(x)$ and $g(x)$ are the basic blending functions, then the two-dimensional blending functions will be $f(x)f(y)$, $f(x)g(y)$, $g(x)f(y)$ and $g(x)g(y)$. This approach can be extended to higher dimensions. For biomedical image reconstruction, the main interest will be two and three dimensions, and also four dimensions for temporal image reconstruction. The above approach differs significantly from other basis-function methods in that the latter use a single basis function in all dimensions, whereas our approach uses several basis functions. One advantage of our approach is that it enables exact calculations of the line integrals, whereas the approach outlined in [12]requires table look-up techniques.

## 1.2 The Reconstruction Problem

In transmission CT, X-rays are passed through a cross-section of an object and readings are taken by detectors at the opposite side. Different parts of the cross-section attenuate the X-rays differently, resulting in different readings at the detectors. The problem is to reconstruct the attenuation map – or image – of the cross-section from the detector readings. Mathematically, the unknown attenuation is a non-negative function of two variables defined on the cross-section. The most elementary method for reconstructing the attenuation is to overlay the cross-section with a Cartesian grid of pixels, and to consider the attenuation to be constant inside every pixel.

In a typical setup, we assume that a line of parallel X-ray sources are aimed at a line of detectors, as shown in Figure 1.2. X-rays are cast toward the detectors, which measure

the attenuation along the ray's paths. The orientation of the sources and detectors is then rotated by some small angle, and the process repeats until the entire cross-section has been "covered" by equiangular orientations. We denote by $m$ the total number of rays obtained in this manner and by $n$ the total number of pixels in a line.

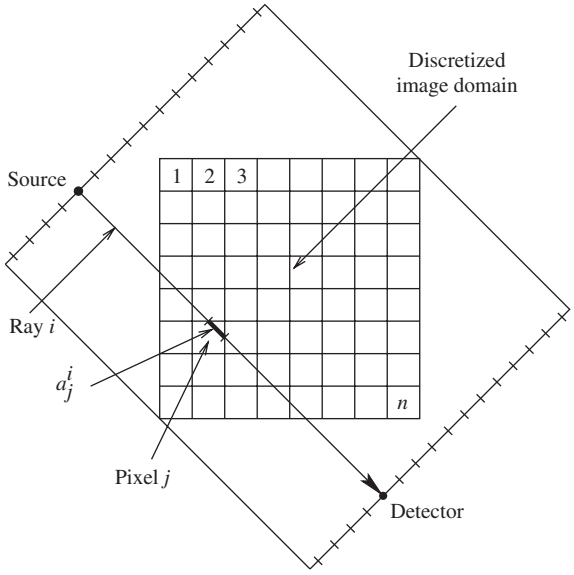


Figure 1.1: Tomographic image reconstruction in the discretized model.

We assume that the pixels and rays are numbered consecutively in some manner from 1 to $n$ and from 1 to $m$, respectively. Let $x_j$ denote the (assumed constant) value of the attenuation function in the $j$th pixel. It is assumed that the X-ray sources and detectors are points, and that the rays between them are lines (a more realistic assumption models the rays as strips). In the rectangle approximation of the line integral it is further assumed that the length of intersection of the $i$th ray with the $j$th pixel, denoted by $a_{ij}$, represents the contribution of the $j$th pixel to the total attenuation along the $i$th ray.

The physical measurement denoted by $b_i$, amounts to the total attenuation along the $i$th ray, is represented by the line integral of the unknown attenuation function along the path of the ray. In the discretized setup, each line integral is approximated by a finite sum, so the (discretized) attenuation function $x_i$ is the solution of a system of linear equations

$$\sum_{i=1}^{n} a_{ji} x_i = b_i \qquad 1 \le i \le m$$

or in matrix form

$$Ax = b \tag{1.2.1}$$

It clear that in the rectangle approximation of the line integral, the sum above involves only the pixels encountered during the ray traversal. An extension where the ray is considered to be a strip will involve the pixels covered by the strip. The approximation of line integral by higher order approximations to the line integral will involve not only the pixels traversed by the ray, but some additional pixels neighboring the ray in its traversal.

# Chapter 2

# Description of the model

## 2.1 One dimensional approximations and the corresponding models

We first explain the one-dimensional models. Suppose $f(x)$ is defined on an interval $[a, b]$. We divide $[a, b]$ into $n$ sub-intervals of equal length: $a = a_0 < a_1 < \ldots < a_n = b$, and denote $h = (b - a)/n$. The approximating function $g(x)$ will be a linear combination of basis or blending functions $g_i(x)$ specified for the interval $[a_i, a_{i+1}]$,

$$f(x) \approx g(x) = \sum_{i=1}^{n-1} \alpha_i g_i(x),$$

where the coefficients $\alpha_i$ are computed from the values of the function $f(x)$.

Without loss of generality we can specify the blending functions $g_i(x)$ by using functions defined on the interval $[0, 1]$ and then map them to the interval $[a_i, a_{i+1}]$ by using the variable transformation given by $x \to (x - a_i)/(a_{i+1} - a_i) = (x - a_i)/h_i$, where $h_i = a_{i+1} - a_i$.

The Newton-Cotes approximation approximates the integral $\int_a^b f(x)dx$ by the integral of the approximating function $\int_a^b g(x)dx$.

### The rectangle rule: middle point model.

The rectangle rule has as blending functions $g_i(x) = s_1(\frac{x - a_i}{h_i})$ , where $s_1(x)$ is the step function

$$s_1(x) = \begin{cases} 1 & \text{for } 0 \le x \le 1, \\ 0 & \text{otherwise} \end{cases}$$

The coefficients $\alpha_i$ are the value of the function in the middle of the segment $[a_i, a_{i+1}]$,

$$\alpha_i = f(a_i + \frac{h_i}{2}),$$

so the approximating function will be

$$g(x) = \sum_{i=1}^{n-1} f(a_i + \frac{h_i}{2})\, s_1(\frac{x - a_i}{h_i}).$$

Its integral is given by

$$\int_a^b g(x)dx = \sum_{i=1}^{n-1} f(a_i + \frac{h_i}{2}) \int_{a_i}^{a_{i+1}} s_1\left(\frac{x - a_i}{h_i}\right) dx = \sum_{i=1}^{n-1} f(a_i + \frac{h_i}{2}) h_i$$

## The trapezoidal rule: linear blending functions

The basis functions for this model are the Lagrange basic interpolation functions[1] $\{\ell_i(x)\}_{i\in\{0,1\}}$ for the interpolation points $\{x_i\}_{i\in\{0,1\}} = \{0, 1\}$, namely $\ell_i(x_j) = \delta_{ij}$ for $i, j = 0, 1$, where $\delta_{ij}$ is Kronecker's delta. Since we have only two points this results in the linear functions $\ell_0(x) = 1 - x$ and $\ell_1(x) = x$.

The approximation function will be

$$g(x) = \sum_{i=1}^{n-1} \left[ f(a_i)\ell_0\left(\frac{x - a_i}{h}\right) + f(a_{i+1})\ell_1\left(\frac{x - a_i}{h}\right) \right].$$

## Simpson's rule: the quadratic blending functions

Simpson's rule for integral approximation is based on the use of piecewise quadratic functions to approximate a function $f(x)$. The quadratic functions overlap. We assume here that we have three points $\{x_i\}_{i\in\{0,1,2\}} = \{0, 1, 2\}$ and we compute the Lagrange interpolation functions $\{q_i\}_{i\in\{0,1,2\}} = \{0, 1, 2\}$ which means that $\ell_i(x_j) = \delta_{ij}$ for $i, j = 0, 1, 2$.

[1]The basic Lagrange interpolation functions for $n$ points $x_1, \ldots, x_n$ are

$$\ell(x) = \frac{\prod_{j=1, j\neq i}^{n} (x - x_j)}{\prod_{j=1, j\neq i}^{n} (x_i - x_j)}$$

From this, we get for the following three basic quadratic polynomial blending functions, defined over the interval $0 \leq x \leq 2$:

$$q_0(x) = \frac{1}{2}x^2 - \frac{3}{2}x + 1, \quad q_1(x) = -x^2 + 2x, \quad q_2(x) = \frac{1}{2}x^2 - \frac{1}{2}x \tag{2.1.1}$$

.

## 2.2 Extension to 2D interpolation

We assume that the plane is divided into square pixels by axis-aligned lines. We can also assume w.l.o.g. that the length of the side of a pixel is 1 and that the entire region of interest is $D = [0, n] \times [0, n]$.

We assume that the data is related to a two dimensional integer valued grid and we want to approximate a two-variables function $f(x, y)$ by a two variable blending function $g(x, y)$.

The grid points are at points $(i, j)_{0 \leq i,j \leq n}$ , while the values $f_{ij}$ of the function $f(x, y)$ are assumed to be collected at the center of the pixel: $(x_i y_j) = (i + \frac{1}{2}, j + \frac{1}{2})$. In each computation we must be aware of this half pixel shift in order to prevent biasing of results.

Blending functions for higher dimensions are obtained by taking as basic functions all the possible products of the one-dimensional blending functions and combining them linearly by using again coefficients $\alpha_{ij}$ depending on the values of the function $f(x, y)$:

$$f(x, y) \approx g(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_{ij} g_{ij}(x, y) \tag{2.2.1}$$

### The pixel model

For the rectangle rule we have a single function $s_1(x)$ and therefore we have a single possible product $s_1(x)s_1(y)$. This will result in the blending function

$$g_{ij}(x, y) = f(i + \frac{1}{2}, j + \frac{1}{2}) s_1(x - i) s_1(y - j) \tag{2.2.2}$$

The value of the function is constant over the square and it is located in the middle of the pixel which corresponds to the definition of a pixel. Therefore we do not need to make correction for half pixel biasing.

### Two-dimensional linear model

For the linear model we will have 2×2 two-dimensional functions, namely $\ell_{ij}(x,y) = \ell_i(x)\ell_j(y)$ where $i,j \in \{0,1\}$. The blending function will be a *bilinear interpolation*

$$\begin{aligned} g_{ij}(x,y) &= f(i,j)\ell_{00}(x-i,y-j) + f(i+1,j)\ell_{10}(x-i,y-j) + \\ & \quad f(i,j+1)\ell_{01}(x-i,y-j) + f(i+1,j+1)\ell_{11}(x-i,y-j) \end{aligned}$$

For the special case when the pixel is the unit square we can see that the values of the blending function $g_{ij}(x,y)$ uses the four corners of a pixel $(0,0)$, $(1,0)$, $(0,1)$, $(1,1)$.

### Two-dimensional quadratic model

In 2D, we get $3 \times 3 = 9$ basis functions 2.1.1defined over $[0,2] \times [0,2]$: $q_{ij}(x,y) = q_i(x)q_j(y)$, for $i,j \in \{0,1,2\}$ and there will be an overlap between adjacent pixels.

## 2.3 Computation of line integrals

The computation of line integral involves the intersection of the geometrical line with square which represents the pixel (see figure 2.1).
Since for each pixel we have different intersections of the line with the grid we need to specify the geometric positions.

### Rectangle/pixel approximation

As the value of the approximation $g(x,y)$ is constant over the pixel it is easy to see that the line integral of $g(x,y)$ over the pixel is the product of this constant value with length of the segment resulting from the intersection of the line with the pixel. So the only non-trivial part in this case is the calculation of the length of the segment.

For the computation of the results of the line integral after bilinear interpolation we will use the notations in figure 2.2

If we denote the blending functions $\{\ell_{ij}(x,y)\}_{i,j=0,1}$ according to which corner of the unit square they belong. the integral over the above pixel will have the value:

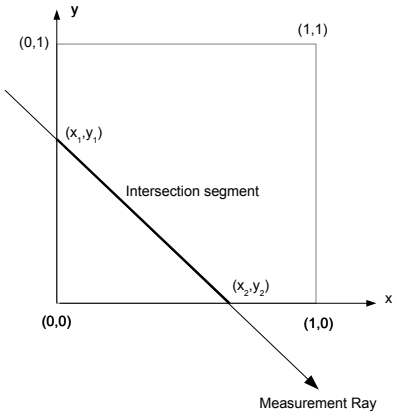


Figure 2.1: Intersection between a measurement ray and pixel

$$I = \int_{(x_1,y_1)}^{(x_2,y_2)} \ell(x,y)ds$$

with

$$\ell(x,y) = \nu_{00} \cdot \ell_{00}(x,y) + \nu_{01} \cdot \ell_{01}(x,y) + \nu_{10} \cdot \ell_{10}(x,y) + \nu_{11} \cdot \ell_{11}(x,y)$$

while the blending functions are

$\ell_{00}(x,y) = (1-x) \cdot (1-y)$, $\ell_{01}(x,y) = (1-x) \cdot y$, $\ell_{10}(x,y) = x \cdot (1-y)$, $\ell_{11}(x,y) = x \cdot y$.

To evaluate this integral we use the parametric representation of the segment: if $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ , then a point $(x, y)$ on the segment connecting $P_1$ to $P_2$ will be $P(t) = (1-t) \cdot P_1 + t \cdot P_2$ and the integral becomes

$$I = \int_0^1 L(x(t), y(t))dt$$

where $x(t) = (1-t) \cdot x_1 + t \cdot x_2$ and $y(t) = (1-t) \cdot x_1 + t \cdot y_2$.

The result will be

$$I = c_{00}\nu_{00} + c_{01}\nu_{01} + c_{10}\nu_{10} + c_{11}\nu_{11}$$

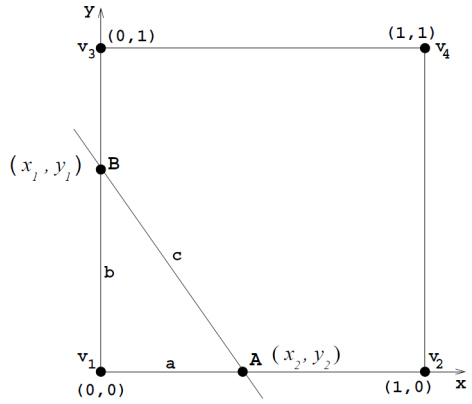


Figure 2.2: Description of the segment for bilinear interpolation

If we denote

$$dx = x_2 - x_1,\ dy = y_2 - y_1,$$

then the coefficients $c_{00}$, $c_{01}$, $c_{10}$, $c_{11}$ of the contributions to the line integral after bilinear interpolation will be [2]:

$$\begin{array}{lcllll}
c_{00} & = & +\frac{1}{3}dx \cdot dy & -\frac{1}{2} \cdot (1 - x_1) \cdot dy & -\frac{1}{2} \cdot dx \cdot (1 - y_1) & +(1 - x_1) \cdot (1 - y_1) \\
c_{10} & = & -\frac{1}{3} \cdot dx \cdot dy & -\frac{1}{2} \cdot x_1 \cdot dy & +\frac{1}{2} \cdot dx \cdot (1 - y_1) & +x_1 \cdot (1 - y_1) \\
c_{01} & = & -\frac{1}{3} \cdot dx \cdot dy & +\frac{1}{2} \cdot (1 - x_1) \cdot dy & -\frac{1}{2} \cdot dx \cdot y_1 & +(1 - x_1) \cdot y_1 \\
c_{11} & = & +\frac{1}{3} \cdot dx \cdot dy & +\frac{1}{2} \cdot x_1 \cdot dy & +\frac{1}{2} \cdot dx \cdot y_1 & +x_1 \cdot y_1
\end{array}$$

[2]For higher degree approximations or 3 dimensions, these computations become unwieldy. Therefore they should be performed using symbolic algebra. We show in the appendix how this can be done for the bilinear.

# Chapter 3

# Implementation of the linear basis method

## 3.1 Top level description of linear basis method

As we already explained, the reconstruction algorithms (in the forward process, backward process and so on) compute line integrals along a ray $L$ as a discretized sum

$$\int_L f(x,y)ds = \sum_{i,j} w_{ij} f_{ij}$$

where $f_{ij}$ are the measured values of some function $f(x,y)$ at the points of a given grid.

For reasons of simplicity we will assume that pixel size is 1 and the center of the world is in the center of the image.

Under these assumptions the weights computations requires as input the parameters of the ray in normal form $(\rho, \theta)$ and the size of the grid $N$. The output of the algorithm are the weights of each pixel $w_{ij}$. Since only the pixels on the ray and around it have non-zero weights we should use a sparse representation. The one used is a pair that contains only the non-zero weights and the index of these weights in a linear indexation of the image.

Each pixel model has its algorithm for weights computation with the interface described above. Each such algorithm is a pure computational geometry routine, which depends only on the grid and the ray intersecting the grid.

This chapter presents the basic algorithm for the computation of the linear model weights.

## 3.2 Half-Pixel Shifted Grid representation

The bilinear interpolation uses the values of 4 pixels that form a $2 \times 2$ square. The data collected assumes that the data is located in the center of the pixel. Would we use the original grid, we would get a half pixel bias, depending on the position where we place our 4 pixel square. This way we would get inaccurate results as we are using points for which we do not have valid data.

Therefore we need to overlay the original grid with this grid half-pixel shifted both in the $X$ and $Y$ directions.

Also, in order to cover the original grid, we now need to resort to a larger grid of size $(N + 1) \times (N + 1)$.

But this is not the single complexity which will be added.

Our first thought was to adapt existing DDA[1] algorithms to a shifted grid. But DDA algorithms are not easily adaptable to be used on a shifted grid.

Another major item was the adaptation of the ray representation to the shifted/translated grid. A workaround that we implemented was to shift the ray itself by half pixel in the opposite direction.

Smaller nuisances came from the fact due to reasons of symmetry, DDA algorithms assume an odd number for grid dimension. Incrementing the grid size by one would make the grid dimension even and non-symmetric in that one edge column and one edge line of the region of interest will be outside of the grid.

An immediate solution would be to apply the algorithm twice, once shifted half pixel in one direction, and secondly half pixel shifted in the diagonally opposite direction. Of course that this would be an expensive computation as in the second call we need only the pixel clipping information from a single border column and a single border line.

A second solution is to use a grid that was padded around by one pixel in the 4 direction, so we are working on a grid of dimension $N = 2 \times K + 3$. This approach implies that we make a careful handling of the boundary pixels and renumbering of the indexes of the intersected pixels in order to discard outlier pixels along the edges. Putting aside the careful programming, the performance is not affected as we need a single DDA run.

The finally adopted solution was to use a grid with size $N' = N + 1$, with a different

[1]DDA means Digital Differential Analyzer.

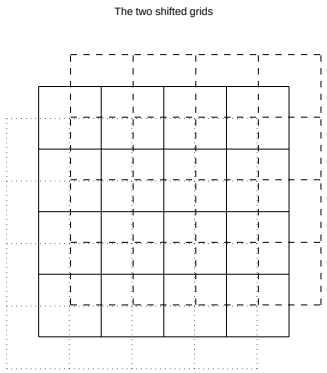


Figure 3.1: Two shifted grids solution for DDA

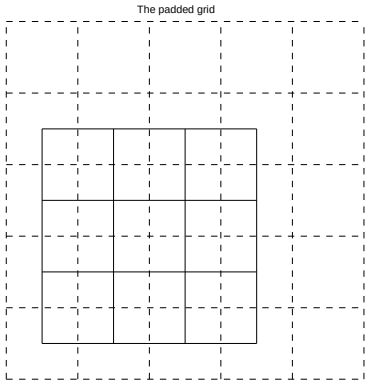


Figure 3.2: Padded single grid solution for DDA

algorithm, which is described in a later section. This algorithm, called grid traversal, which has many benefits over the DDA, will be discussed in another section.

**Rays parallel to the coordinate axes:**

These rays should have a different treatment. The reason is that except for the case of a ray-grid overlap, the computation is trivial (one dimensional interpolation for the first intersected pixel) and the weights and intersection list for that ray can be produced without using an incremental traversal algorithm.

**Ray overlapping the grid**

This is a special case of the previous case when the equation of the ray is identical to that of one of the grid lines. In this case, we are resorting to the original grid and we are using

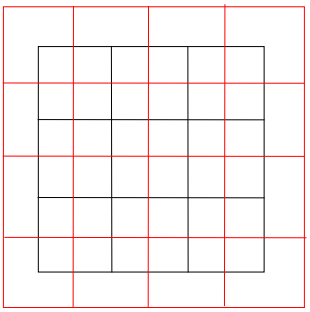

Figure 3.3: In red: the Overlayed grid of size $N+1$. In black: the original grid of size $N$

the pixel model (as the intersecting segment becomes a mid-line in the original pixel).

**Note:** In order to allow to handle these cases it is necessary to modify the algorithm so it will return not only the index of the intersected pixel and its weight for a single pixel, but for two pixels. This implies to make the necessary modifications both to the summation algorithm and to the DDA. A simple shift by an infinitesimal number is not a robust solution due to the fact that, as a result of accumulation of numerical rounding errors and cancellation errors, the DDA will advance in a zigzag jitter around the ray.

## 3.3 Shifting the ray representation by half pixel

DDA algorithms require a reference point. This reference point is chosen to be the intersection of the ray with the normal to the ray from the origin of coordinates.

Assume that we have a line written in normal form $x\cos\theta + y\sin\theta = \rho$ which in vector form is

$$\mathbf{r}\cdot\mathbf{u} = \rho$$

where $\mathbf{u} = (cos\theta, \sin\theta)$.

The intersection of the perpendicular from the origin with line is given by the point[2]

[2]This representation may cause problems when the angle $\theta$ is close to a multiple of $\pi/2$. For $\theta$ close to 0 we can retrieve $\rho$ as $\rho = \frac{a_x}{\cos\theta}$ and for $\theta$ close to $\pi/2$ we can use $\rho = \frac{a_x}{\sin\theta}$

$$\mathbf{a} = (\cos\theta, \sin\theta) = \rho\mathbf{u}.$$

Now assume that we shift the coordinate system $\mathbf{r} = (x, y)$ by the vector $\Delta\mathbf{r} = (\Delta x, \Delta y)$ to a new coordinate system $\mathbf{r'} = (x', y')$. Then we have $\mathbf{r} = \mathbf{r'} + \Delta\mathbf{r}$ and since the we are just shifting the coordinate system, the angle of the perpendicular to the line remains the same so the line's equation becomes

$$(\mathbf{r'} + \boldsymbol{\Delta}\mathbf{r}) \cdot \mathbf{u} = \rho$$

or

$$\mathbf{r'} \cdot \mathbf{u} = \rho'$$

where $\rho' = \rho - \boldsymbol{\Delta}\mathbf{r} \cdot \mathbf{u}$.

The intersection point in the new coordinate system will be

$$\mathbf{a'} = \rho'\mathbf{u} = (\rho - \Delta\mathbf{r} \cdot \mathbf{u})\mathbf{u}$$

so it changed by

$$\Delta a = \mathbf{a'} - \mathbf{a} = \rho'\mathbf{u} = (\rho - \Delta\mathbf{r} \cdot \mathbf{u})\mathbf{u} - \rho\mathbf{u} = -(\boldsymbol{\Delta}\mathbf{r} \cdot \mathbf{u})\mathbf{u}.$$

Finally we substitute $\Delta\mathbf{r} = (\Delta x, \Delta y)$ and $\mathbf{u} = (cos\theta, \sin\theta)$, and get

$$\Delta\mathbf{a} = -(\Delta x \cos\theta + \Delta y \sin\theta)\,(\cos\theta, \sin\theta)$$

For a half pixel shift $(\Delta x, \Delta y) = (\frac{1}{2}, \frac{1}{2})$ which gives

$$\Delta\mathbf{a} = -\frac{1}{2}(\cos\theta + \sin\theta)\,(\cos\theta, \sin\theta)$$

so

$$\begin{aligned} \Delta a_x &= -\frac{1}{2}(\cos\theta + \sin\theta)\,\cos\theta \\ \Delta a_y &= -\frac{1}{2}(\cos\theta + \sin\theta)\,\sin\theta \end{aligned}$$

## 3.4 Grid Traversal

The CT reconstruction programs [1] are usually using DDA (a variant of Bresenham algorithm) for computing the intersection points and all other geometric information. This process is really complex and requires a combinatorial analysis of all the possible configurations. But the main problem of DDA is the accumulation of rounding errors.

The method we propose is very simple and delegates all the combinatorial manipulation to the ray and *aabb* (*a*xis *a*ligned *b*ounding *b*ox) intersection algorithm proposed by [26]
In addition there are no rounding error accumulation and the algorithm allows vectorization.

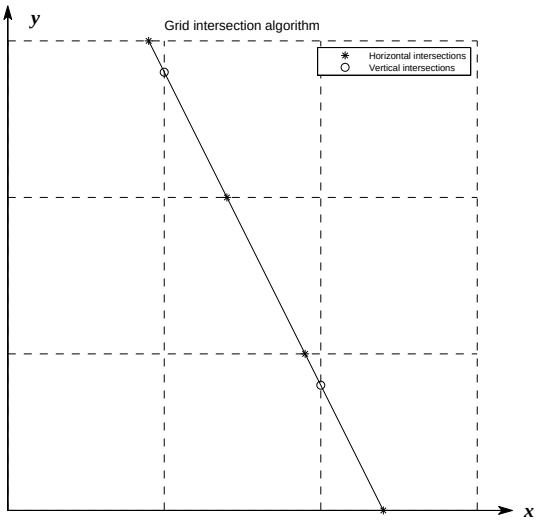


Figure 3.4: Grid traversal algorithm

**Algorithm description:**

First we determine bounds of the region of interest. Then we calculate all the parametric values of the intersection of the ray with the vertical grid lines, and again the same but with the horizontal lines. We keep 2 lists of values, a list for vertical intersection and a list for horizontal intersection. Then we merge the 2 lists into a single list, while preserving monotonicity. Finally we translate this list into a list of the ray-grid intersection points, and also obtain the list of clipped pixels, and the list of cases of pixel traversal induced by the ray crossing the grid.

**Algorithm details:**

The algorithm works as follows: Assume that the ray is defined by a point name $p_{orig}$ and a direction $dir = (dir_x, dir_y)$. The *aabb* is defined by two corners $P_{low}$ and $P_{high}$.
The K& K algorithm idea is to express the ray parametrically as $P_t = P_{orig} + t * dir$. The intersection points with the *aabb* are defined by the parametric values $P_{near}$ and $P_{far}$. The K& K algorithm can be modified so that it returns also the 4 values $t_{xlow}$, $t_{xhigh}$, $t_{ylow}$ and $t_{yhigh}$ of the intersection of *aabb* with the lines $x_{low}$, $x_{high}$, $y_{low}$ and $y_{high}$. The grid division is performed now, not on the ranges of values of $X$ and $Y$ but on the range $[t_{xlow}, t_{xhigh}]$ for $X$ grid lines and on the range $[t_{ylow}, t_{yhigh}]$ for $Y$ grid lines. This way we get the sets of parametric values $T_x$ for the $X$ range and $T_y$ for the $Y$ range. Finally we restrict these ranges to the range $[t_{near}, t_{far}]$ returned by the K & K algorithm. The restricted $T_x = T_x \cap ([t_{near}, t_{far}])$ and $T_y = T_y \cap ([t_{near}, t_{far}])$ are finally merged into a single set $T = T_x \cup T_y$. The coordinates of the grid intersection will be $X = P_{orig} + T * dir_x$ and $Y = P_{orig} + T * dir_y$

## 3.5 Bilinear weights summation

In the middle point model, each pixel intersected by a ray will contribute a weight for that ray projection. In the linear basis model, all the pixels neighboring the intersected pixels will have a non-zero contribution to the weights of the ray projection. This is a result of the fact that in bilinear interpolation 4 pixels are involved in the computation of the weights. Each pixel intersected in the shifted grid, contributes 4 corners to the weights vector.

A pixel with non-zero weight gain its weights in at least 2 intersections, and may gain weight from up to 4 intersections. In such case, if a pixel has already been weighted, the weight computed for the current intersection will add to existing value.

In the case of DDA sparse grid representation, summing a pixel weight require access to an existing memory item that stores the current weight. The multi-linear interpolation introduces a combinatorial difficulty in knowing the weights previous state. The lack of monotonicity adds complexity.

### Sparse representation of weights summation in linear model

The pixel weight update is built so that, based on the geometrical position of the intersecting ray, we know in advance which pixels will be involved. We should know if a pixel is already

in the list or not, so we can add the contribution to the existing pixels and set the weight contribution to a newly allocated pixel. The mechanism to implement this is based on 4 pointers into the pixel list - each pointer holds the entry in the weight list for one of the corners. When the algorithm steps to next intersection, a pointer is either substituted with one of the other pointers, or is expanding the weight list. A pointer is substituted if and only if the pixel weight was non-zero. Otherwise, its number is added to the index list.

In principal there are 8 traverse cases. Using symmetry as in [6], helps reducing the number of cases into 3:

1. MoveRight
2. MoveUpRight
3. MoveUp

In the case of MoveRight 2 pixels that represent the right corners of the shifted pixel, are expanded to the list of weights, and the 2 others are summed, and their pointers are substituted. In the case of MoveUpRight 3 new indexes are expanded to the list, and one (pixel that correspond to the down left shifted grid corner) is substituted. In the case of MoveUp 2 pixels that correspond to the bottom corners are expanded, and the other are summed and their pointers are substituted.

There is a special treatment for pixels that lies on the edge. Since we use for the linear interpolation a shifted grid with dimension size $N + 1$ , weights of outliers are ignored.

It is easy to see that all the weights are summed in a correct manner. The processed can be generalized easily to higher-order blending functions, e.g quadratic model. In quadratic model, the difference would be in the number of pointers to maintain, and of course the quadratic blending weights would have other integrands, but the processes would still be linear.

**Summation with shifted grid boundaries** When the ray encounters a pixel which lies in the shifted grid boundaries, a special care in the form of sub-pixel calculation is needed in order to sum correct weights. Essentially, there are 2 cases: a grid corner pixel, and a grid edge pixel. In both cases, the pixel is divided into 4 sub-pixels which are squares. In the case of corner, the calculation is performed only on the sub-pixel that lies in the original grid. In

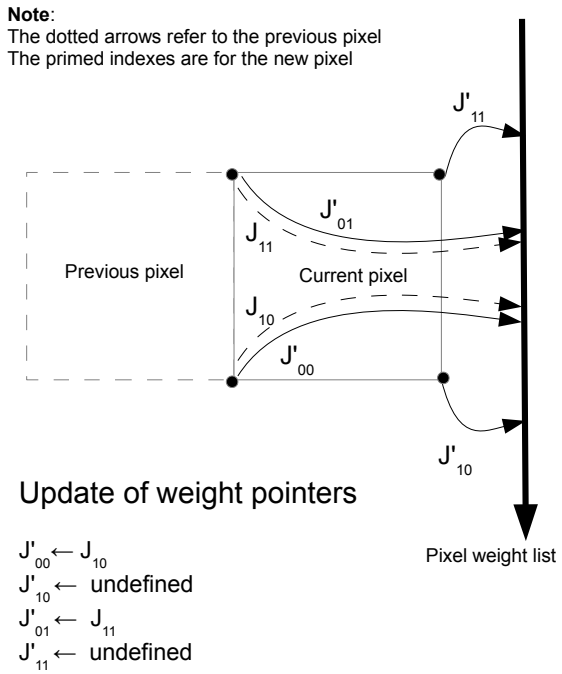


Figure 3.5: Tracking indexes in the summation process. Case of moving right

addition, only the weight that correspond to the corner pixel in the original grid is summed. In case of edge pixel, the calculation is performed on 2 sub-pixels that lies in the original grid, and 2 pixels of the original grid are summed.

This delicate handling in the boundaries does not seem to be critical, as applying a more coarse approach such as padding or ignoring outliers in the pixel weights computation can be sufficient with minor added error.

**Complexity Analysis:** The time complexity of the above described algorithm is linear to the number of intersected pixels. Space complexity is constant since we need to use only 4 pointers in the process. Thus, the algorithm is both optimal in time and optimal in space, since computing weights requires a grid traversal algorithm which has $O(N)$ time complexity.

### Alternative weights summation algorithms

**Random access update.** We define a vector $S$ of size $|x|$. $S_i$ is the coefficient for the $i$th pixel. Each time we obtain coefficient of a pixel, it is added to the current value in $S$ . In addition we maintain a vector that stores the pixel indexes of the non-zero weights. After the last update had been done we "collect" the weights that are non zero using the indexes vector. The time complexity of this algorithm linear to the number of intersected pixels of the ray ($O(\sqrt{|x|})$). However , the space complexity is $\sqrt{|x|}$ for each ray. This may seems feasible as we already need $O(|x|)$ space to store $x$ . However, if we want to parallelize weights computation, space complexity is multiplied by the number of parallel processes.

**Search tree data structure.** Using a search tree with pixel index as the key. A tree node stores both a pixel index and its linear model weight. Each update requires tree traverse to the relevant node. The last step of the algorithm the tree is traversed in in-order manner and the weights are obtained. Time complexity is $O(\sqrt{|x|}\log\sqrt{|x|})$. Space complexity linear to the size of non-zero weights - $O(\sqrt{|x|})$ .

## 3.6 Singular cases

Singular cases occur when a ray and a grid are collinear, i.e, the ray does not intersect any pixel of the grid but intersects the grid it self. With the pixel basis functions, this causes a case of singularity, since the ray equation is undefined. Of course this can be handled simply by spreading the weights equally between the 2 columns or 2 rows that the ray cross between. On the other hand, with linear basis functions such a case is well defined. Since we use a shifted grid, the ray intersects pixels, and the bilinear interpolation is performed for each pixel. In cases where a ray and shifted grid are collinear, than we can see that the ray intersects pixels in the original grid, and so we reduce the weight computation back in the original grid.

## 3.7 Symmetry considerations

As similar to DDA [5][6], the ray-grid intersection consists of strong symmetry properties, which can be utilized for a computation optimization of the linear equation system used by algebraic reconstruction algorithms family. The optimization is both on memory needed, and on the computation time needed for calculation of the rays. In addition it is used to simplify linear weights summation described earlier.

### Symmetry usage in the linear model weights computation

When we need to calculate the ray weights, as in other ray-grid processes, it is most convenient to have the ray angle on the $1^{st}$ octant - i.e $\{r = (\theta, \rho) : 0 \leq \theta \leq \pi/4\}$. So the first step of the algorithm the ray is transformed to a symmetrical equivalent ray. At the last step, after weights had been calculated for the transformed ray, a simple transformation of indexes is done, and the algorithm returns the weights as should have been for the input ray. This process is well suited for any model of blending functions. In addition, a great benefit can be gained for calculating 8 symmetric rays at the same time, as calculating single ray weights is sufficient for all the other symmetric equivalent rays.

### Symmetry usage for storage optimization

Besides the computation benefit of using a single weights computation of one ray for other symmetric congruent rays, applying symmtery properties can reduce the amount of space needed to store the weights, when avoiding re-calculation in forward and backward projection in the reconstruction algorithms steps. It will be explained later in 3.8.

.

## 3.8 Optimizations

### Using Symmetry

The use of symmetry congruence of rays has a great computational impact: only about 1/8 of the number of rays are needed to calculate their coefficients. Except a small number of

special cases, each ray has potentially symmetrical equivalent rays. Hence, we reduce the processing time by 7/8.

**Index mapping tables.**

Straight forward computation of symmetrical index requires this process:

matrix index→matrix subscript→Cartesian points→

symmetrical Cartesian points→matrix subscript→matrix index.

This process can accelerated by preparing index symmetry permutations as look-up tables. This index symmetry permutations can be calculated by linear transformations of matrix of indexes such as orthogonal rotations and reflection. In addition the permutation look-up tables can be used for a fast way of getting weights indexes from stored weights storage using symmetry considerations.

We considered 2 efficient ways for generating the mapping tables:

1. Using geometrical symmetry of Cartesian points
2. Linear transformation of the pixels indices

**Geometrical symmetry of Cartesian points**

Recalling Bresenham's [5] circle mid-point algorithm, at the final stage we mark 8 points simultaneously: $(c_x + x_i, c_y + y_i)$, where $(c_x, c_y)$ is the circle's center and $(x_i, y_i), 1 \leq i \leq 8$ are all the combinations of $+/-$ of the $x$, $y$ and switching between them. If we assume that center of matrix is $(0, 0)$, then the equivalent points from the 1st octant to the 8th octant are

Map = $[(x, y), (y, x), (y, -x), (-x, y), (-x, -y), (-y, -x), (-y, x), (x, -y)]$

We need to map every index to these 8 options for mapping corners.

Assume that we have a grid of size $N \times N$ and $1 \leq i \leq N^2$ is an index running on this grid and the Cartesian origin is located at the position $(O_x, O_y) = (N/2, -N/2)$. This index can be mapped to Cartesian coordinates $(x, y)$ and back with$(x, y) = (i \mod N, \lfloor i/N \rfloor) - (O_x, O_y)$ and $i = (x + O_x) + (y + O_y) \cdot N$.

With these preparations, the map preparation goes as follows:

**for** $i$ =1 **to** $N^2$

Get the Cartesian coordinates $(x, y)$ corresponding to index $i$

**for** *octant* = 2 **to** 8

Using Map put $(x, y)$ in the *octant* then convert it to index.

**end**

**end**

**Linear transformation of the pixels indices**

Another way we found for mapping the indices for symmetrical rays, were by applying operations such as combination of orthogonal rotation ($\theta = k * \pi/2, k \in \mathbb{Z}$ ) and reflection.

## Avoiding the re-computation of the weights matrix

Recall that the form of the reconstruction problem is $Ax = b$, where $A$ represents the weights matrix. Here we need to call only the weight calculation in the first iteration, and store the weights in the memory for the following iterations. However, the storage is expensive, even for sparse representations of $A$ standard image resolutions. For example, if the number of rays is close to $|x|$ (almost determined system), then storing linear model weights would require close to $4 \cdot |x|^{3/2}$real numbers for storing the weights with the same number of integers for storing the indexes. Of course that using symmetry as described in 3.8 can reduce that number to about $0.5 \cdot |x|^{3/2}$ A typical space size can be $7|x|^{3/2}$ bytes when using double precision representation of float numbers. For example, handling an image of size $256 \cdot 256$ which is a standard DICOM [4] size of A single CT image, requires a space of 112 MB at most.

**Storage utilization** There are several strategies to store the weights matrix. For real applications a constant scanning and visualization configuration may be widely used, i.e, number of detectors, number of projections, and pixel size are constants, the weights matrix can be calculated offline, stored on hard-drive, and loaded to shared memory when needed. For simulation purposes, for images that are not too big (up to $256 \times 256$), calculating the weights on the first iteration may be more addressable, since configurations may change frequently, and the weights matrix can be fully stored in typical modern RAMs. When using RAM for the storage, the whole weights matrix can be per-allocated once, to save OS calls for allocations. For per-allocating a tight bounded size of the required memory, we use statistics about the the number of non-zero weights for the blending function model, from previous

experiments. Another allocation approach is to allocate each ray's weight vector separately, after the weights calculation has been performed for that ray. This would be most efficient in memory. However, it would slow down performance to some extent compared to the former approach.

### Small optimizations

**Outliers detection look-up table** As mentioned in the weights summation description, the borders of the shifted grid are outliers that must be ignored and not include in the weights vector output. Checking each pixel in the shifted grid if it contains outliers is time consuming. Thus we can build a table that contains relevant data for each index.

## 3.9 Complexity analysis

### Space complexity

Experimentally, Given $r$ = number of rays, $p$ = number of projections, $n_X$, $n_Y$ = picture dimensions (for simplicity assume $n_X = n_y = n$). $\sqrt{2}n$ is the maximum number of pixels intersected by a single ray, which is occurred for $\theta = \pi/4 + k \cdot \pi/2, \rho = 0$. The upper bound of the number of pixel model weights is $\sqrt{2} \cdot n \cdot r \cdot p$. As for the bilinear model, the upper bound is $3 \cdot \sqrt{2} \cdot n \cdot r \cdot p$, which occur in the same ray as in pixel model. In the linear case, every newly traversed pixel expand the list by 3 new weights. However statistical experiments has shown that the ratio in the number of non-zero weights between pixel model and linear model is about 1.5,as in pixel it is $2.5 \cdot n \cdot r \cdot p$, and in .bilinear it is $3.8 \cdot n \cdot r \cdot p$.

### Time complexity*:*

The described process of the linear model weights calculation has time complexity which is linear to the number of intersected pixels, which is bounded by $O(\sqrt{x})$ However, it is slower than pixel model weights calculation, since it requires additional process of bilinear interpolation of each pixel. The number of additions, subtractions, and multiplications for each ray-pixel intersection weights is about 50, and divisions can be entirely avoided (divisions are needed for computing the intersection length with $\frac{a}{\sin\phi}$where $a$ is described in 2.2.As to

the ray projections, due to the increased number of non-zero weights, the projection in linear model will be slower by the ratio of the number of non-zeros in the pixel model. We can improve running time in the pixel and linear methods by using symmetry consideration.

# Chapter 4

# Experimental results and discussion

## 4.1 Parallel ray-detector CT simulation

The linear model was implemented in 2D within the framework of the SNARK09 image reconstruction software package [31]. which can handle both CT and PET reconstructions. This package contains the pixel, linear and the blob models for image reconstruction. As for the blob basis functions, default parameters were used, which are calculated by the Snark09 program, as stated in [14] the Snark09 program has the ability to choose the best parameters for the blob basis function (for detailed explanation of Blob see 6.1. Those 3 basis functions were tested and compared in 5 different phantoms under different configuration, such as noise, detector resolution, number of projections, and phantom contrast. The quality of the images were analyzed both qualitative (subjective visual judgment), and quantitative - measuring the reconstructed images in terms of relative error, distance, and in one example also point-wise accuracy, and structural accuracy. First, we present 2 simple phantoms. The first is a checker board, with only 3 different intensities - black, grey, and white. The second phantom is A second phantom is an image that like the first phantom, has only 3 different intensities, but each different is different from its neighbors. This phantom was chosen to emphasize a phenomenon that occurs when using the blob basis functions. The third example is of a synthetic brain phantom with small contrast which was originally created to identify brain activity in PET scans. The fourth example is of Shepp-Logan head phantom [35], which is a well-known test-bed example, and is used extensively for measuring quality of image reconstruction. The last phantom is a realistic data, of a sagital slice image of brain,

acquired by MRI scan and reconstruction [1]. This last phantom should be considered merely as a sampled data, unlike all the other phantoms which are mathematically described.

## 4.2 Usage of Reconstruction algorithms

### 4.2.1 ART

Reconstruction was done with ART (algebraic reconstruction technique), which is actually the Kaczmarz algorithm [24], but independently rediscovered in the context of image reconstruction from projections[32]. A relaxation parameter of 0.05 was mostly used. It is well known that ART with a small relaxation parameter provides good images [14], and the result is very close to the least squares solution [8]. However, this parameter was tuned in special cases, such as under-determined system and noise. The initialization vector - $x^0$ was set such that all the components were given the value of the average density of the phantom. A good introductory explanation on ART is given at ...

#### Use of Selective smoothing

In cases where the system $Ax = b$ was ill-posed, particularly in noisy and under-determined environments, use of ART is problematic since there can be many solutions that reach minimum least squares. As such, it was invited to induce some "regularization like" operation. In each reconstruction step, we performed selective smoothing after each iteration. As mentioned in [14], using selective smoothing is a good approach to tackle noisy data, and in some sense it shares similar characteristics to the behavior of the Blob. The method of selective smoothing was proposed for the cases when the image is composed from regions within which the densities fall in a small number of ranges, but these ranges are well separated from each other. A good example for this are the head phantoms, where we have three clearly distinct regions: the background, the skull and the contents of the skull.

Assume that we have a reconstructed pixel $a$ and $b_1, \ldots, b_4$, $c_1, \ldots, c_4$ are its neighbors as below

$$\begin{matrix} c_1 & b_2 & c_2 \\ b_1 & a & b_3 \\ c_4 & b_4 & c_3 \end{matrix}$$

Then we construct a new image where the value of $a$ is replaced by the weighted value

$$\widetilde{a} = \frac{Aa + B\sum_{i=1}^{4} f_i b_i + C\sum_{i=1}^{4} f_i c_i}{A + B\sum_{i=1}^{4} f_i + C\sum_{i=1}^{4} f_i}$$

where $A$,$B$,$C$ are the smoothing factors and $f_i$ are the selection factors. The selection factors are computed as follows: if the difference between the value of the center pixel $a$ and the of the value of a neighboring pixel $v$ (either of type $a$ or type $b$) is less than a threshold $t$, then this pixel will be selected; otherwise ignored.

$$f = \begin{cases} 1 & \text{if} \quad |a - v| < t \\ 0 & \text{otherwise} \end{cases}$$

We found that a smoothing where $C = B/\sqrt{2}$, which can be easily justified on geometric grounds, gives a significant improvement in case of noise and low contrast.

### 4.2.2 EMAP

For ill-posed problems as mentioned in 1.2.1, a suitable reconstruction algorithm would use a regularization element in its optimization criterion. One very popular such algorithm is the expectation maximization with maximum a posteriori probability (EMAP) [16]. This algorithm is widely used in other imaging filed than CT, such as PET, since it performs well under noise. The optimization criterion is maximization of the log-posterior functional:

$$\ln p(x|y) = \sum_{i=1}^{I} \left[ y_i \ln \left( \sum_{j=1}^{J} l_{ij} x_j \right) - \sum_{j=1}^{J} l_{ij} x_j \right] - \frac{\gamma}{2} x^t S x$$

where $x_j$ is the jth basis function of the image vector $x(1 \leq i \leq I)$, $y_i$ is the ith component of the measurement vector $y(1 \leq j \leq J)$ and $l_{ij}$ is the length of intersection of the ith measurement line with the jth basis function

The regularization part is a straight forward smoothing, as minimizing the difference with the average of the neighboring pixels:

$$x^t S x = \sum\nolimits_{r\in N}(x_r - \tfrac{1}{8}\textstyle\sum_{j\in N_r} x_j)^2$$

Note that this type of regularization was given by SNARK09 and apparently not "state of the art", for instance since it does not preserve edges. however in recent years there have been a great progress in the study of regularization types, one notably used is the TV, total variation minimization [40].

## 4.3 Measures

Two measures were used to define how much a reconstructed image diverges from the phantom, the *distance* and the *relative error*. These are calculated by [31], and defined as follows. Let $\widehat{x}$ denote the phantom and $x^k$ denote the reconstructed image after $k$ iterations. Let $S$ denote the set of indices $j$ of pixels which are in the region of interest and let $\alpha$ be the number of elements in $S$. The *average* value of $\widehat{x}$ is defined by

$$\widehat{\rho} = \frac{1}{\alpha}\sum_{j\in S}\widehat{x_j},$$

and the standard deviation of $\widehat{x}$ is defined as

$$\widehat{\sigma} = \left(\frac{1}{\alpha}\sum_{j\in S}(\widehat{x_j} - \widehat{\rho})^2\right)^{1/2}$$

The *distance* between $x^k$ and $\widehat{x}$ is defined as

$$\delta_k = \frac{1}{\widehat{\sigma}}\left(\frac{1}{\alpha}\sum_{j\in S}(x_j^k - \widehat{x_j})^2\right)^{1/2}$$

Setting $\widehat{\tau} = \sum_{j \in S} |\widehat{x}_j|$, the *relative error* of $x^k$ is defined by

$$\epsilon_k = \frac{1}{\widehat{\tau}} \sum_{j \in S} |x_j^k - \widehat{x}_j|.$$

**Statistics and other Figures of Merit (FOM) of Phantom reconstruction** The quality of reconstruction was estimated using the following figures of metric proposed by [17]

Assume that we have a phantom $P$, which contains $N$ elements (or synonymously structures) $A_i$, where $1 \leq i \leq N$. We denote by $\mu(A)$ the average of the pixels in the element $A$ and by $P^r$ the reconstructed phantom.

The *structural accuracy* is the negative of the average of the inaccuracy over all elements in the phantom

$$-\frac{1}{N} \sum_{i=1}^{N} |\mu(P^r \cap A_i) - \mu(A_i)|$$

The *pointwise accuracy* is defined as the negative of the normalized root mean square distance, namely

$$-\frac{||P - P^r||_2}{||P - \mu(P)||_2}$$

This measure would be 1 if the reconstruction would be a uniformly dense picture with the average density of the input. A large difference in few places would will cause this FOM to be large.

The *hit-ratio* is calculated only for phantoms which are generated according to the G. T. Herman's *paired structures* method. In this method, the phantom contains $N$ pairs $(A_+, A_-)$ which are positioned symmetrically with respect to the $y$-axis with unequal densities. We have a hit if

$$\text{sign}\left(\mu(A_+) - \mu(A_-)\right) = \text{sign}\left(\mu(A_+^r) - \mu(A_-^r)\right)$$

In other words, a pair of symmetric structures provides us with a hit, if the structure in the pair with the higher average density in the phantom is also the structure in the pair with the higher average density in the reconstruction.

The hit-ratio is the number of hits divided by the total number of pairs $N$ in the phantom.

For additional reference on the above described FOM, see [18] [10]

## 4.4 Noiseless and almost determined scan environment

Experiments on this section were done using a scan configuration that can produce close to optimal results, for each system. The Detector and Projection angle resolutions were set such that number of projection and number of rays are almost equal, and each is equally spaced and covers the whole ROI. In addition the scanning is noiseless. The configuration for each example is shown at 4.4

| Case Example | Variables | Equations | Projections | Rays |
|---|---|---|---|---|
| CheckerBoard | 17161 | 17423 | 131 | 133 |
| "High Frequency" CheckerBoard | 121 | 143 | 11 | 13 |
| Shepp-Logan | 9025 | 9215 | 95 | 97 |
| Herman-Head | 13225 | 13455 | 115 | 117 |
| Low contrast brain | 9025 | 9215 | 95 | 97 |
| Brain MRI | 66049 | 66563 | 257 | 259 |

Table 4.1: Phantoms used for the experiments in almost determined system

### 4.4.1 CheckerBoard

This simple example illustrates a mathematical function that is composed of square tiles, and the result is a checker board image. The phantom and the reconstructed images are illustrated in 4.1. The function in most of the places is constant, thus it is well suited for using Blob. However, it can be seen that although using LBF cause some noise, the image is sharper than in Blob, and compared to PBF the noise is at much lower level. We can see in 4.2that LBF is competitive to Blob. The explanation of the quantitative measures may not be straight forward. The strength of the Blob compared to LBF as seen in the images is the image smoothness and reduced noise. On the other hand, LBF images are slightly sharper, and the edges are stronger.

### 4.4.2 High frequency spatial aliasing.

In this example we took a checker board and resized it to the smallest resolution possible without reducing information. Here an aliasing effect is taking place. This illustrates the problem of using Blob in highly detailed images. In this example, we have 3 intensities: 0 (black), 0.7 (grey), 1 (white). Each pixel differs from its neighbors.The images were magnified 16 times larger, in order to see each pixel of the images. This example was chosen to emphasize that smoothing like behavior of the basis functions can vanish small details. As can seen in 4.3, in this example the PBF was the best fit, and was able to produce almost perfect image. LBF was second best, and Blob was the worst. The first test with Blob was performed using default parameters calculated by the SNARK09, which usually give close to optimal results [14]. We have tried to seek for optimal parameters of Blob for this example, so the obvious adaptations were tuning $a$ ,the support size, which is most affecting for this example, and $\alpha$, the shape of the blob. Support size was decreased to minimize smoothing effect, and shape was much widened. The tuning had greatly improved the Blob result, although checker players would still fill very uncomfortable trying to play on such checkerboard. It may be explained due to the fact that the Blob uses an hexagonal grid, or other causes. However, it is beyond the scope of this work to get a depth of the Blob intrinsics.

### 4.4.3 Shepp-Logan phantom

In this experiments we tried to achieve the best results for all of the 3 basis functions. It was found that the most influential parameter for this goal was the relaxation parameter of the ART algorithm. However, a more thorough examination of the best fit relaxation parameter for each basis function is beyond the scope of this work. However, it has been observed that LBF can be used with higher relaxation values than Blob and PBF. First examination is with relaxation parameter of 0.05, and the second examination is with varying values as defined in 4.4.3.

| Method<br>Examination number | PBF | LBF | Blob |
|---|---|---|---|
| 1 | 0.05 | 0.05 | 0.05 |
| 2 | 0.4 | 0.6 | 0.4 |

Table 4.2: Shepp-Logan with different relaxation parameters.

### 4.4.4 MRI scan of human brain

| Measure<br>iteration 20 | Pixel | Linear | Blob |
|---|---|---|---|
| Distance | 0.1502 | 0.1005 | 0.0952 |
| Relative Error | 0.1454 | 0.0895 | 0.0729 |

Table 4.3: Brain-MRI minimal distance and relative error

As it can be seen on figure 4.4.4 It is obvious that LBF is superior to PBF. As Compared to Blob, the Blob image has less noise than LBF image. However, highly detailed regions get too blurred by the Blob, particularly where exists narrow objects, which is assimilated in the surrounding background. In those cases the LBF manage to perform better.

By looking at 4.4.4, ,

## 4.5 Low contrast

In this section we test LBF in situation where objects in the phantoms have low contrast, i.e x-ray absorption intensities (defined as CT numbers in CT applications) of the phantom are very close to each other. In 4.9 the phantom was picked from set of examples in Snark09 framework[31]. The objects inside the big ellipse has intensities of $a = 1.95$ and $b = 2.0$, and the background ellipse has intensity value of 1.0. In order to illustrate the differences we've done histogram equalization on the phantom and reconstructed images.

We can see that that after 1'st iteration, in the reconstructed images it is impossible to tell the differences between the objects with intensity a and b. In the 10th iteration, the PBF and LBF fails to map intensities correctly, while Blob can give some partial indication. As for the 20th iteration, The differences between objects of different intensities is clear in Blob

image for almost all of the objects. As for LBF, the contrast is worse than Blob, but better than PBF, and compared to the last, more objects can be correctly mapped by a reviewer.

## 4.6 Under determined systems

When using reconstruction methods from the series expansion methods, the problem of equation system size "explosion", occurs in almost any realistic configuration of inverse problem, notably in computed tomography (see 3.8 for computational analysis). As such, it is a common setting for the equation system in our scope to be under-determined (and moreover strongly under-determined). In addition, under-determined equation system is a type of an ill-posed problem, in which the solution is not unique, and the number of solution may be numerous (depends on the degree of freedom of $A$) . In such case, using least squares as optimization criterion is not the best approach [14]. A more robust method for such cases is one that uses some sort of regularization in the minimization criterion [19]. One widely used regularization type is one that takes into account how similar is the pixel's value compared to his surrounding neighborhood.

### 4.6.1 Under-determined system due to low projection resolution

In this type of under-determined system, we set the number of projections to be half of the image grid resolution. (for $N \times N$ image, number of projection is $\frac{N}{2}$). As can be seen in 4.10 , The LBF is showing a good competition to Blob , and is superior to to PBF.

### 4.6.2 Under-determined system due to low detector resolution

In this type of under-determined system, the detector resolution is reduced by such that for image with size $N \times N$ the number of detectors is $\frac{N}{2}$ , and the number of projection is $N$. Low detector resolution produce more acute artifacts than in low projection resolution, since there amount of data for the pixels is unbalanced (for some pixels none of the ray will pass through them). When using EMAP as reconstruction algorithm, the LBF is slightly better than Blob. In the case of ART, it is obvious that LBF images are inferior to Blob, but still significantly better than PBF.

### 4.6.3 Strongly under-determined system due to low detector and projection resolution

In this examination the system has number of ray equations which are about 1/4 of the number of the pixels of the reconstructed image, and can be seen as strongly under-determined. Here both the number of projections and the number of ray-detector pairs are reduced by half. As seen also in 4.6.2the choice of the optimization criterion is crucial for LBF. As for EMAP, we used value of $\gamma = 5$ for LBF. In 4.12 Using EMAP and LBF with low smoothing weight ($\gamma = 2$), gives almost identical image to using EMAP and Blob. This interesting observation strengthen the claim that Blob is a another use of regularization. We can see that using EMAP and LBF with optimal $\gamma$ gives the best image (although it does take a lot of iterations).

### 4.6.4 Summary

It is clear that in the scope of under-determined system, the cases of detector resolution and projection resolution are essentially different. As for low projection resolution, the LBF reconstruction were competitive to Blob. On the other hand In low detector resolution, it was clear that the Blob is superior. The main reason was that optimal Blob parameters are selected with consideration to scanning resolution. In future work, the LBF can also take into consideration the scanning configuration, and define a coarser grid for the bilinear interpolation.

We present two test cases on which the differences between the three methods are worth noting. Both cases are at a resolution of $255 \times 255$. Case 1 was the Herman head phantom [**?**, Section 4.3], which is specified by a set of ellipses, with a specific attenuation value (called *density*) attached to each elliptical region. The pixel size was set identical to the inter-ray distance; this setup ensures the least amount of artifacts in the reconstructed images (as can be readily verified by experimentation). The reconstruction used 180 equidistant angles, resulting in a system of 65,025 variables and 64,980 rays – an almost exactly determined system.

Case 2 is an example of quite an adverse situation for any reconstruction method: under-determined and of low contrast. The phantom is based on example 7 of SNARK09, modified

to have very low contrast, i.e., the differences between the densities are small. The number of equations is only about 25% of the number of variables; this was obtained from the geometry of Case 1 by doubling both the inter-ray distances and the angle between successive projection angles. The resulting system of equations is thus strongly under-determined (from a mathematical point of view). Such a setup can be regarded as mimicking a situation of low dose radiation, as may be mandated by clinical requirements. The different test cases are summarized in Table.

## 4.7 Noise

Experiment on noisy projection is has big importance for simulating realistic scenarios of medical imaging. There are many types of noise that are described in the medical imaging literature, for example noise caused by the x-ray source (quantum noise), noise caused during the ray's path (scatter noise for example ), noise caused by the detectors (after glow), and other types as well. However interesting they are, examination of all types noise is beyond the scope of this work. 2 types of noise are examined here: multiplicative noise, and scatter noise.

### 4.7.1 Multiplicative Noise

Multiplication Noise is a by-product of the probabilistic nature of signal amplification technologies. The noise is simulated such that projection data is altered in as $P_{new} = P_{old} gauss(\widehat{\sigma}, \widehat{\rho})$, were Gauss returns a random value with mean $\widehat{\sigma}$ and standard deviation$\widehat{\rho}$. In the experiments we made for multiplicative noise, the parameters for the noise are $\widehat{\sigma} = 1.00$, and $\widehat{\rho} = 0.02$. In addition a selective smoothing was performed with parameters $t = 0.1$, $A = 0.25$ $B = 0.073223305$ $C = 0.051776695$ (see 4.2.1)Here in 4.13 the images are reconstructed under multiplicative noise. We can see the effect of selective smoothing, as smoothed LBF looks most similar to Blob without smoothing, and Blob with the same smoothing parameters as LBF is too aggressive.

### 4.7.2 Scatter Noise

Scatter noise is the most important type of noise caused by interaction between x-ray photons and the material which they pass through. This mechanism is illustrated in 4.14, and is best known as Compton affect [23]. This type of problem is more acute in scanner with fan-beam geometry (and multislice as well) than in parallel-beam geometry, since in wide angles the scatter increases. However, we focused in our experiments only on parallel-beam geometry, and even withing that framework, the resulting images shows how destructive is that phenomenon. The Compton scatter cause some of the x-ray photons to reach the wrong detectors, thus cause the projection data to be affected by the scatter. The most apparent seen artifacts are streaks, intensity (CT numbers) shifts, and reduced contrast.

We have used Shepp-Logan phantom for the scatter noise experiment. The simulated scatter noise was simulated as a convolution of the projection data with triangle filter function, which has 2 parameters - peak value and scatter width. The values picked for our experiment were Scatter peak = 0.01 and Scatter width = 4* (#detector-width), the total amount of scatter is about 2.17% (the convolution function definition is given at [31]. Here, we used ART as the reconstruction algorithm, and EMAP with the smoothing regularization that was used in previous experiments was found unfit for all basis functions, since it blurs the reconstruction of an already blurred measurements (use of an alternative regularization operator such as Total Variation wasn't supported in the simulation system). As such optimal $\gamma$ will be 0, and we won't see any benefit from using EMAP, and it will just be . and As can be seen in 4.15 Each basis function corresponds differently to the scatter noise. All of the images were batch processed using Matlab with adaptive histogram equalization (because of the darkening affect of the scatter), and 'unsharp' filter which is the negative of the Laplacian filter with parameter $\alpha = 0.9$. The worse handled is PBF, which is most vulnerable to streaks artifact. The LBF images also suffer from streaks, but in several magnitude less than PBF. As for the Blob images, it does not suffer from streak artifacts. However, there are 2 different artifacts that are visible only in Blob images, one is ringing, and the second are stripes.

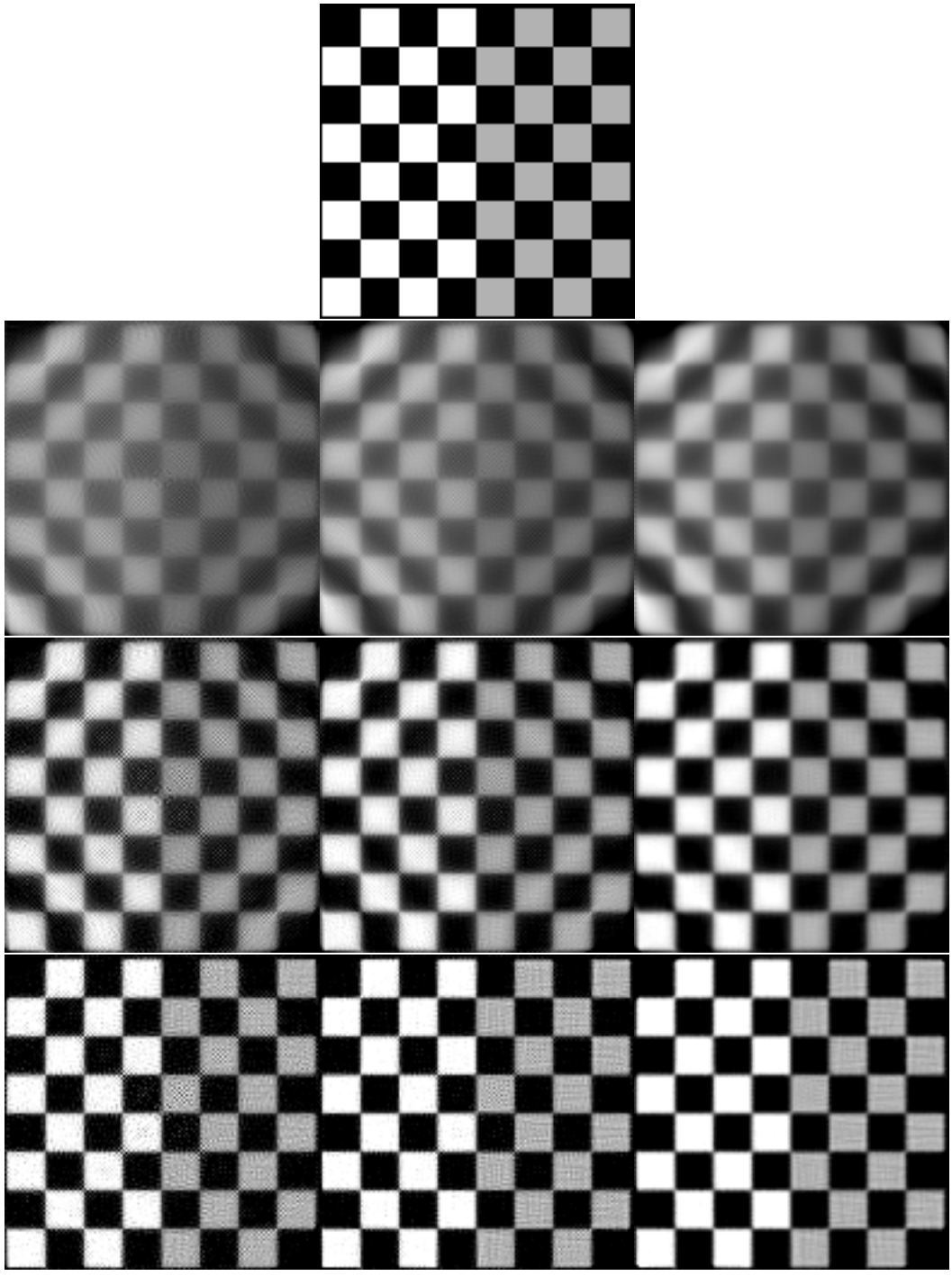

Figure 4.1: checker board phantom (row 1) with square size 1 and reconstructed images after 1 (row 2), 5 (row 3), and 20 (row 4) iterations. Reconstructed images for PBF, LBF and Blob are in left, middle, and right in that order.

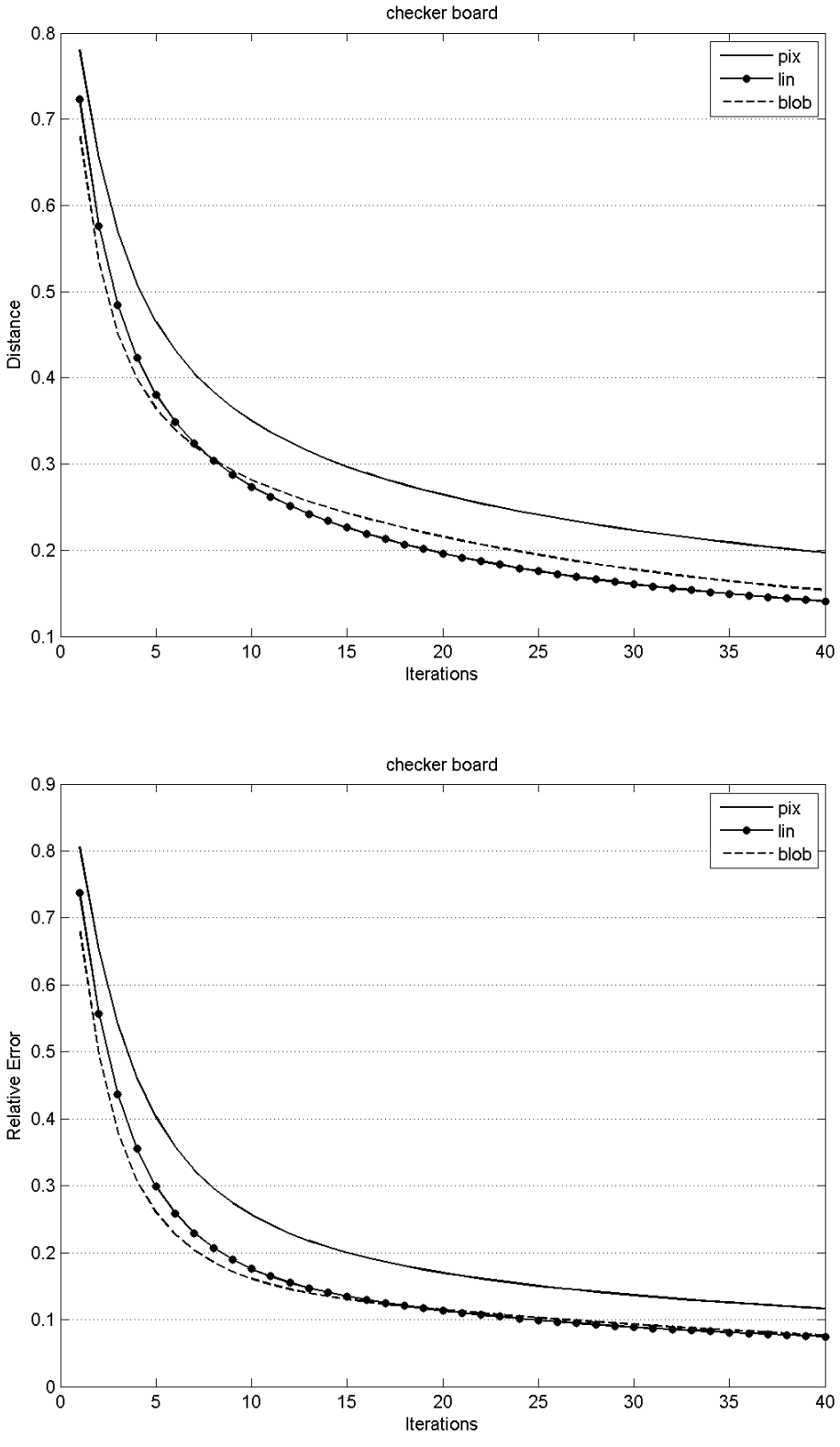


Figure 4.2: Distance and Relative Error for checker board with square size 16.

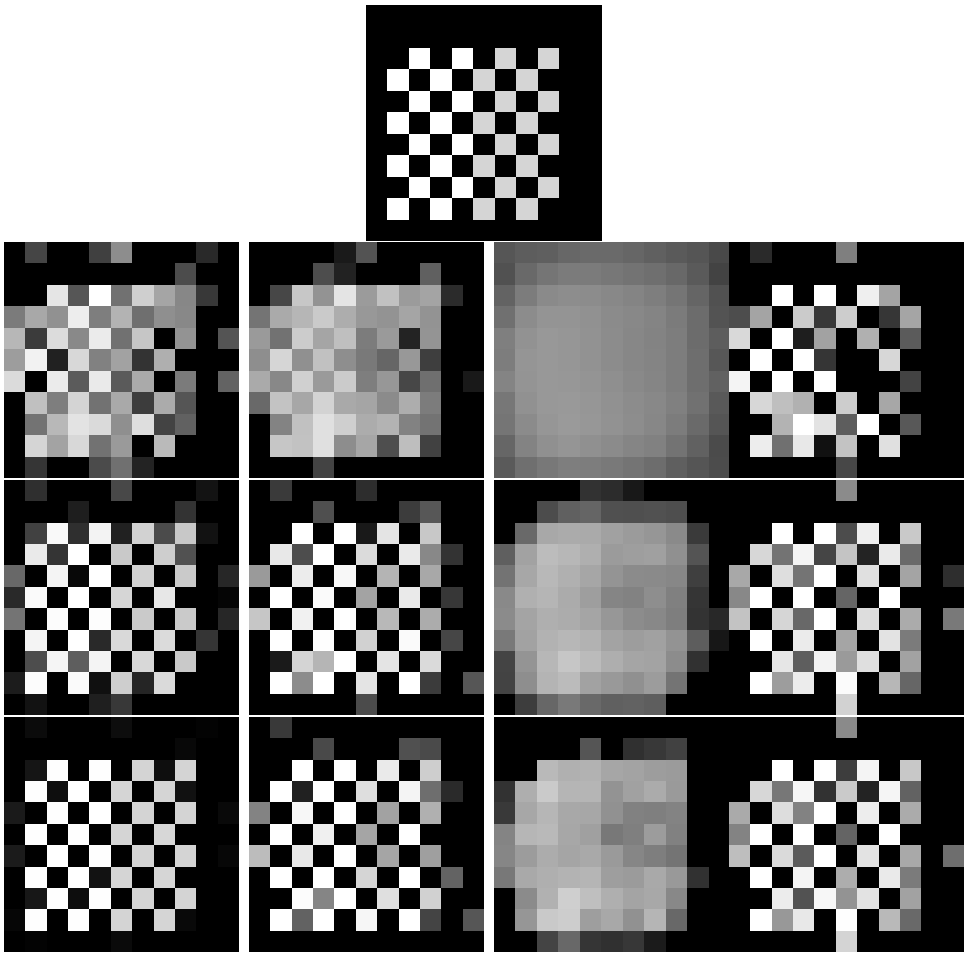

Figure 4.3: High frequency checker board and reconstructed images after 1 (row 2), 10 (row 3), and 20 (row 4) iterations. Reconstructed images for PBF, LBF are in the 2 leftmost columns in that order, followed by Blob images with default parameter and with optimal parameters.

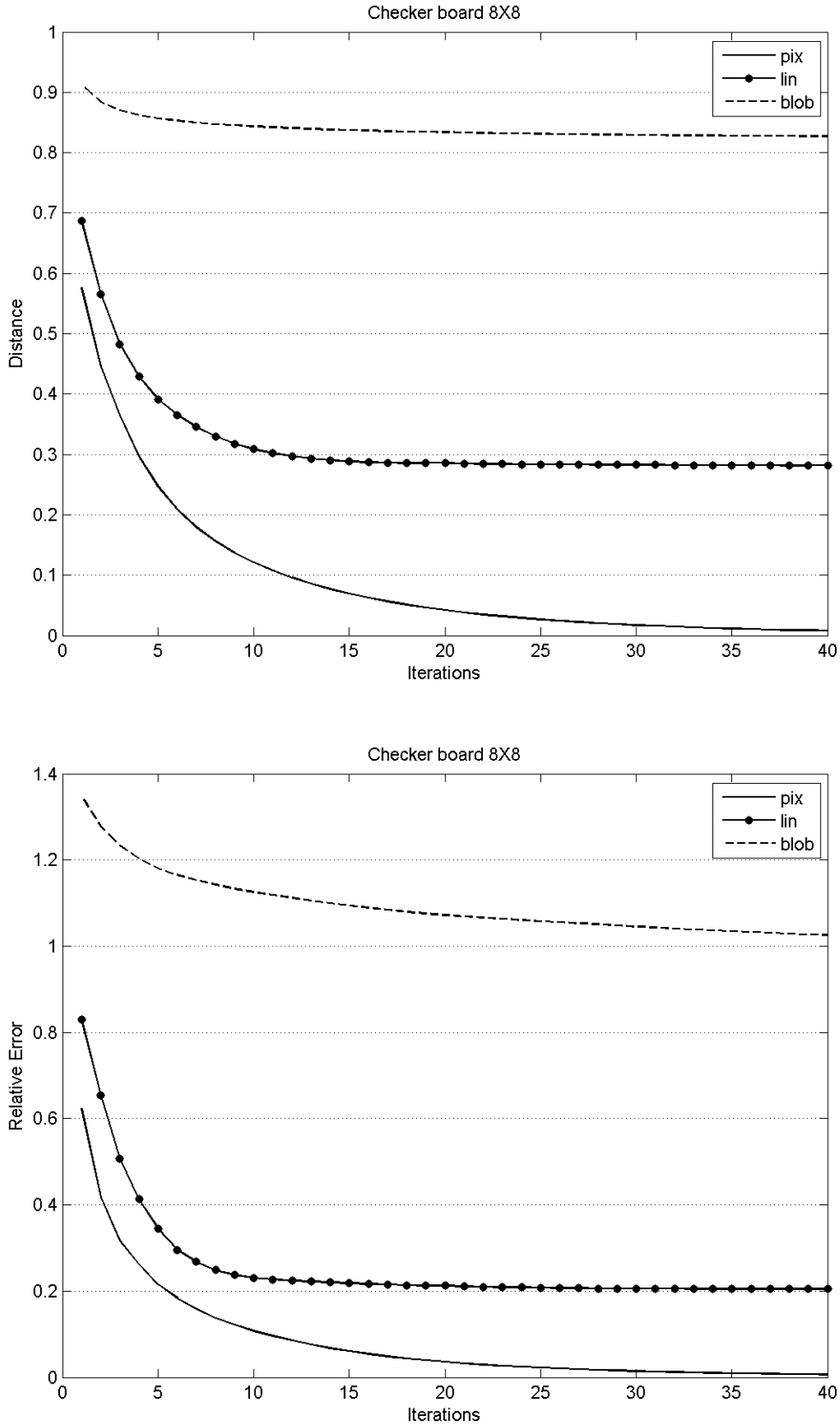


Figure 4.4: Distance and Relative Error for "High frequency" checker board. The catastrophic behavior of the blob is well illustrated here.

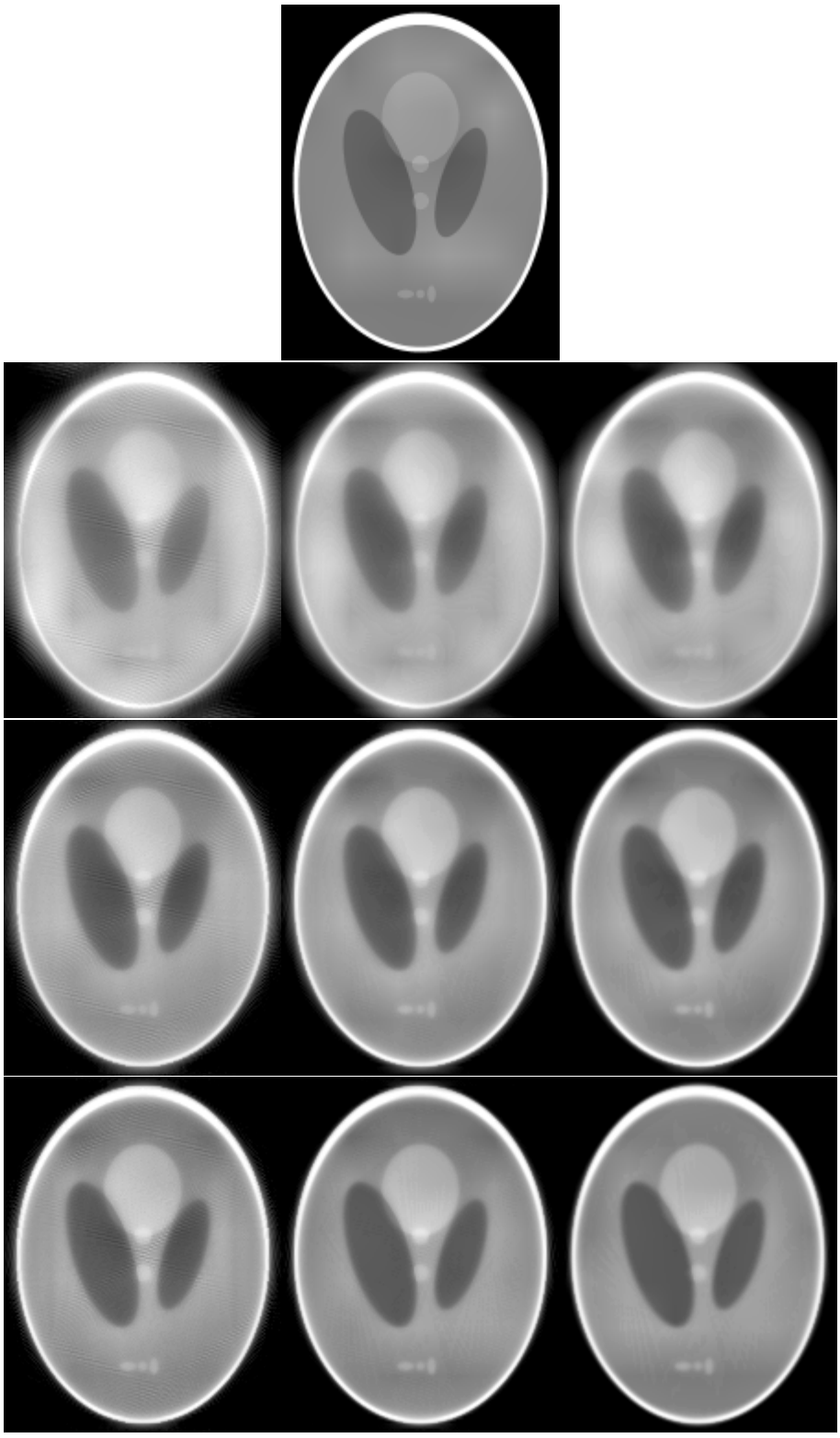

Figure 4.5: Shepp-Logan phantom and reconstructed images after 1 (row 2), 3 (row 3), and 5 (row 4) iterations. Reconstructed images for PBF, LBF and Blob are in left, middle, and right in that order.

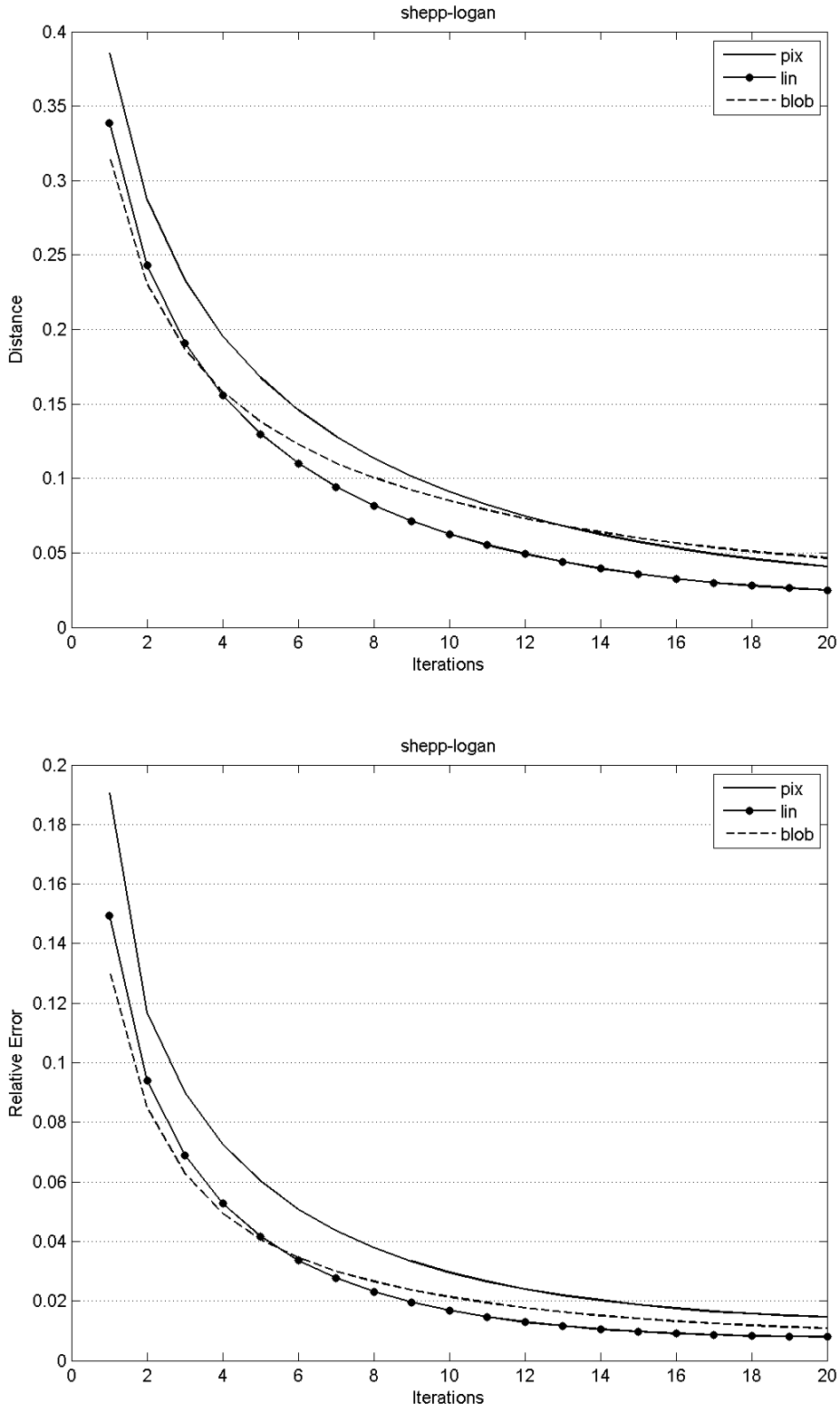


Figure 4.6: Shepp-Logan phantom and reconstructed images with minimum relative error

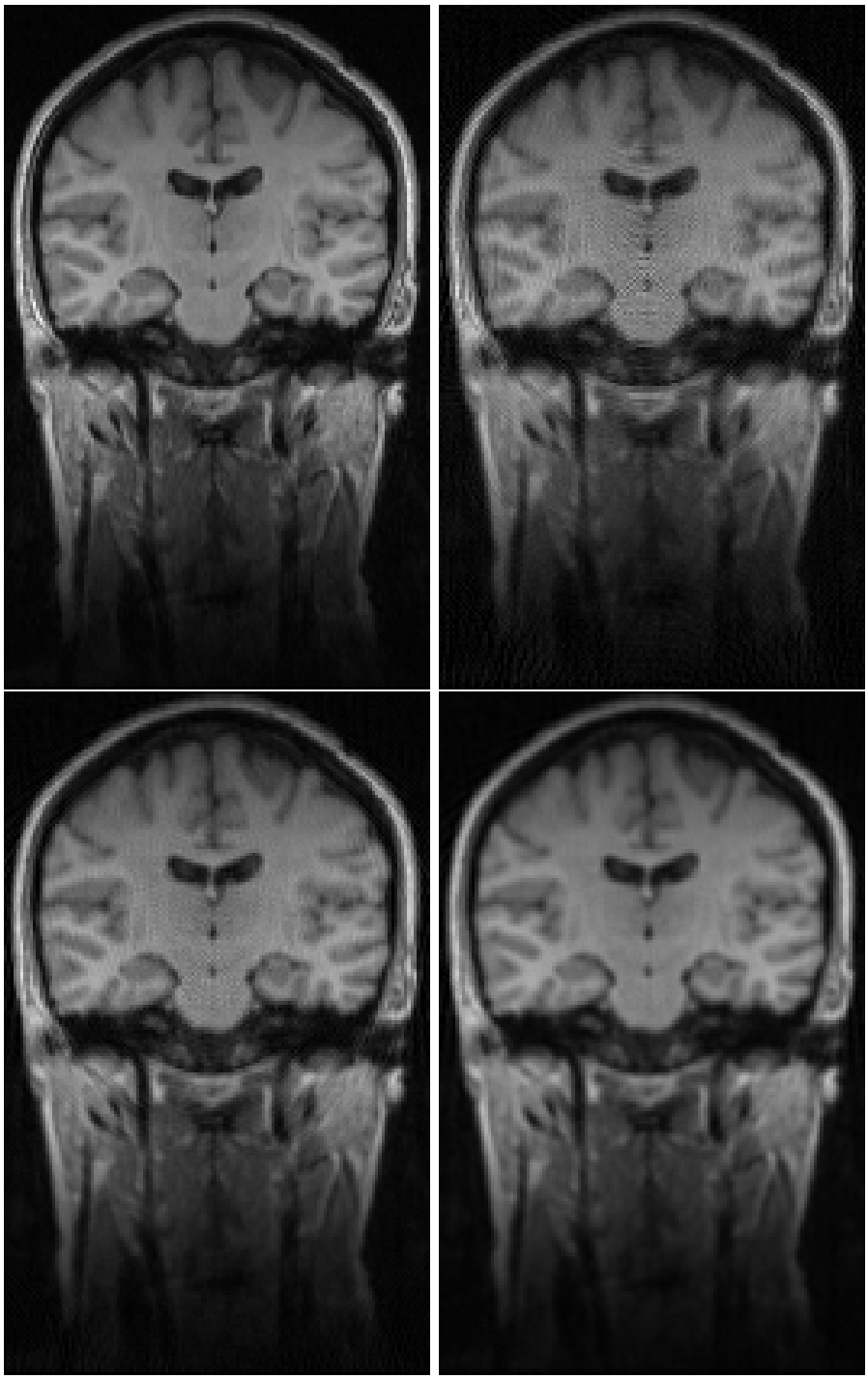

Figure 4.7: Brain-MRI scan and Reconstructed images after 20 iterations. Top: Left - Brain-MRI Phantom taken from [1]. Right - Reconstruction with PBF . Down : Left - Reconstruction with LBF. Right - Reconstruction with Blob

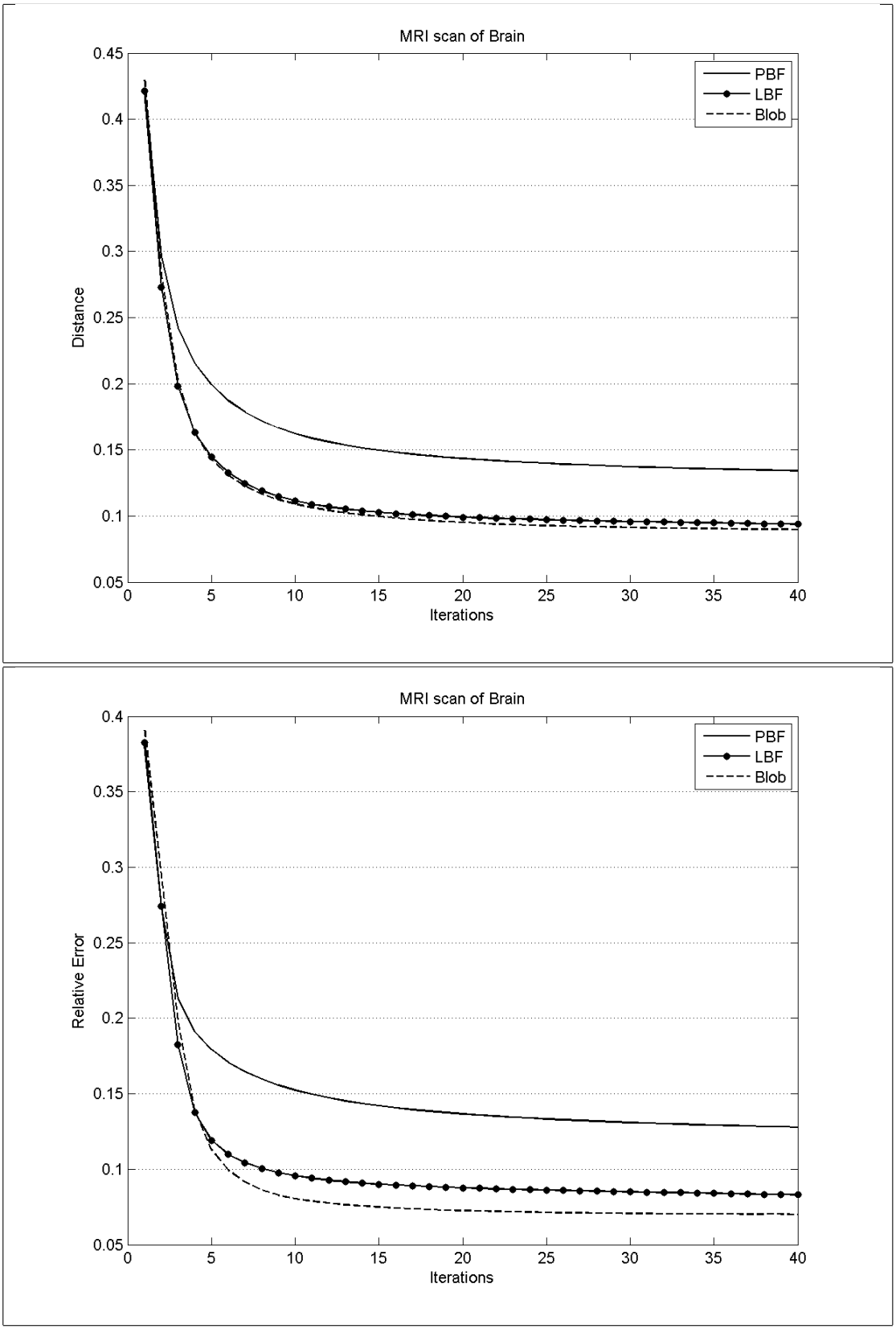


Figure 4.8: Distance and relative error of Brain-MRI phantom

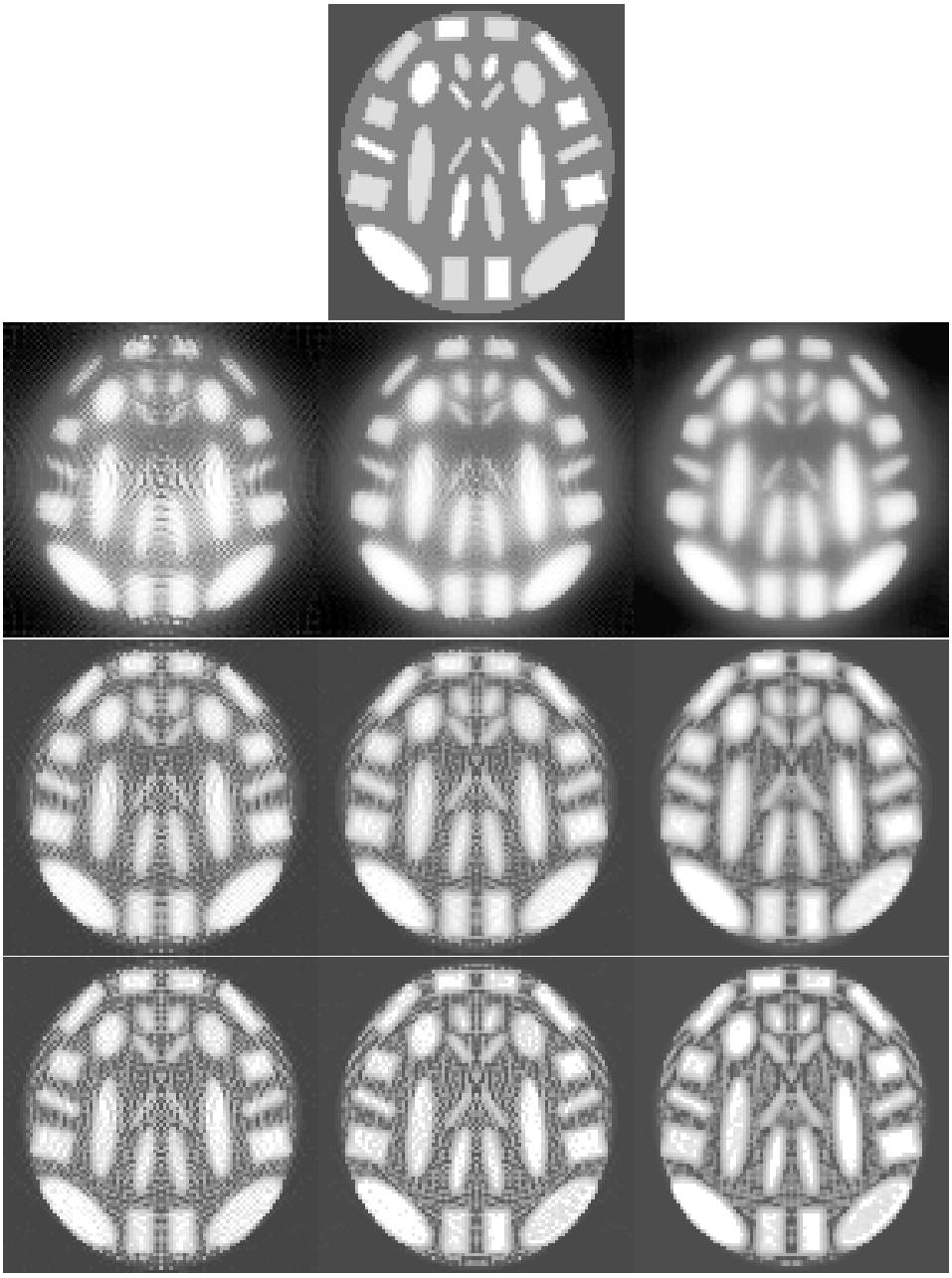

Figure 4.9: Low contrast brain phantom and reconstructed images after 1 (row 2), 10 (row 3), and 20 (row 4) iterations. Reconstructed images for PBF, LBF and Blob are in left, middle, and right in that order.

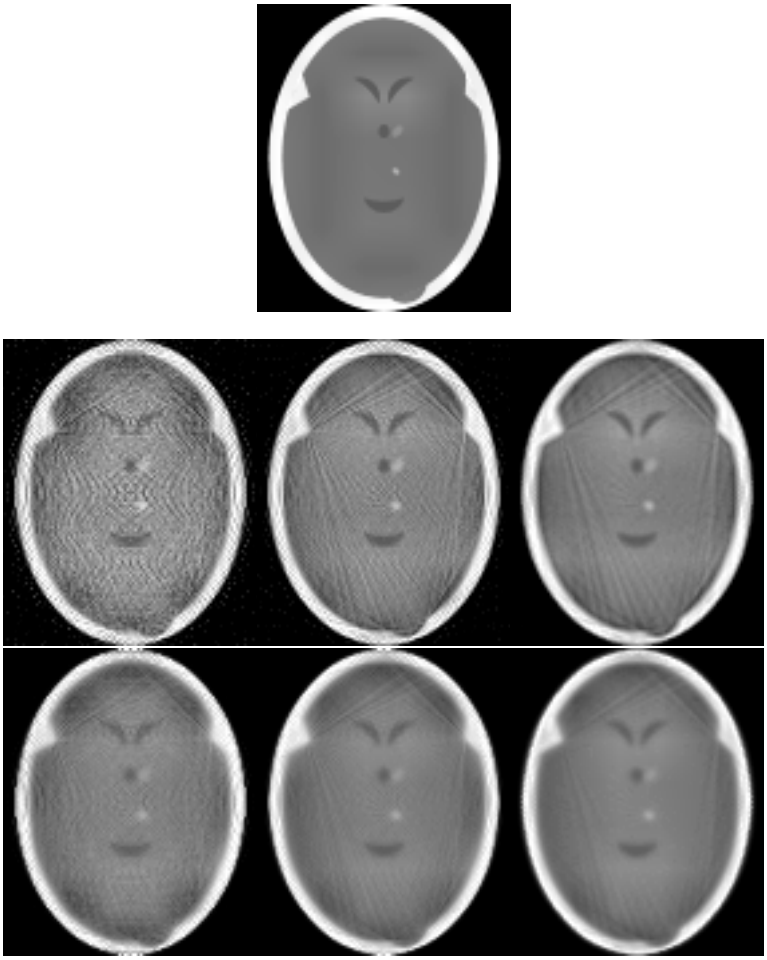

Figure 4.10: Reconstruction in low projection resolution. Row 2 from left to right: images after 20 iterations with ART+PBF, ART+LBF, and ART+Blob. Row 3: images after 40 iterations with EMAP+PBF, EMAP+LBF, EMAP+Blob

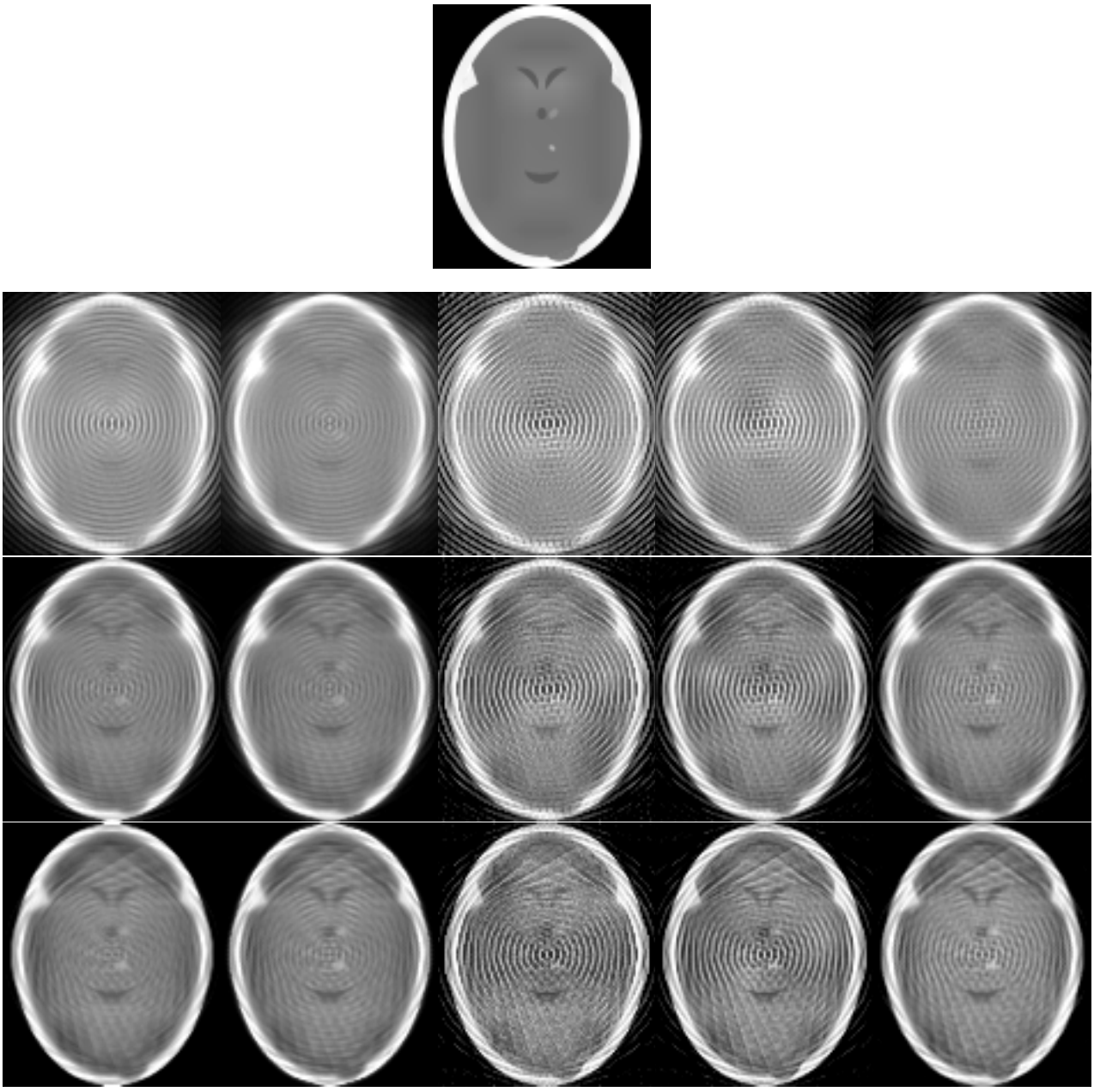

Figure 4.11: Herman head phantom and reconstructed images in low detector resolution. column 1,2 respectively - EMAP+LBF with $\gamma = 7$, EMAP+blob, after 5, 20, and 50 iterations from uppermost to lowermost row. Column 3,4,5 respectively - ART+PBF, ART+LBF and ART+Blob after 1, 5, 20 iterations from uppermost to lowermost row

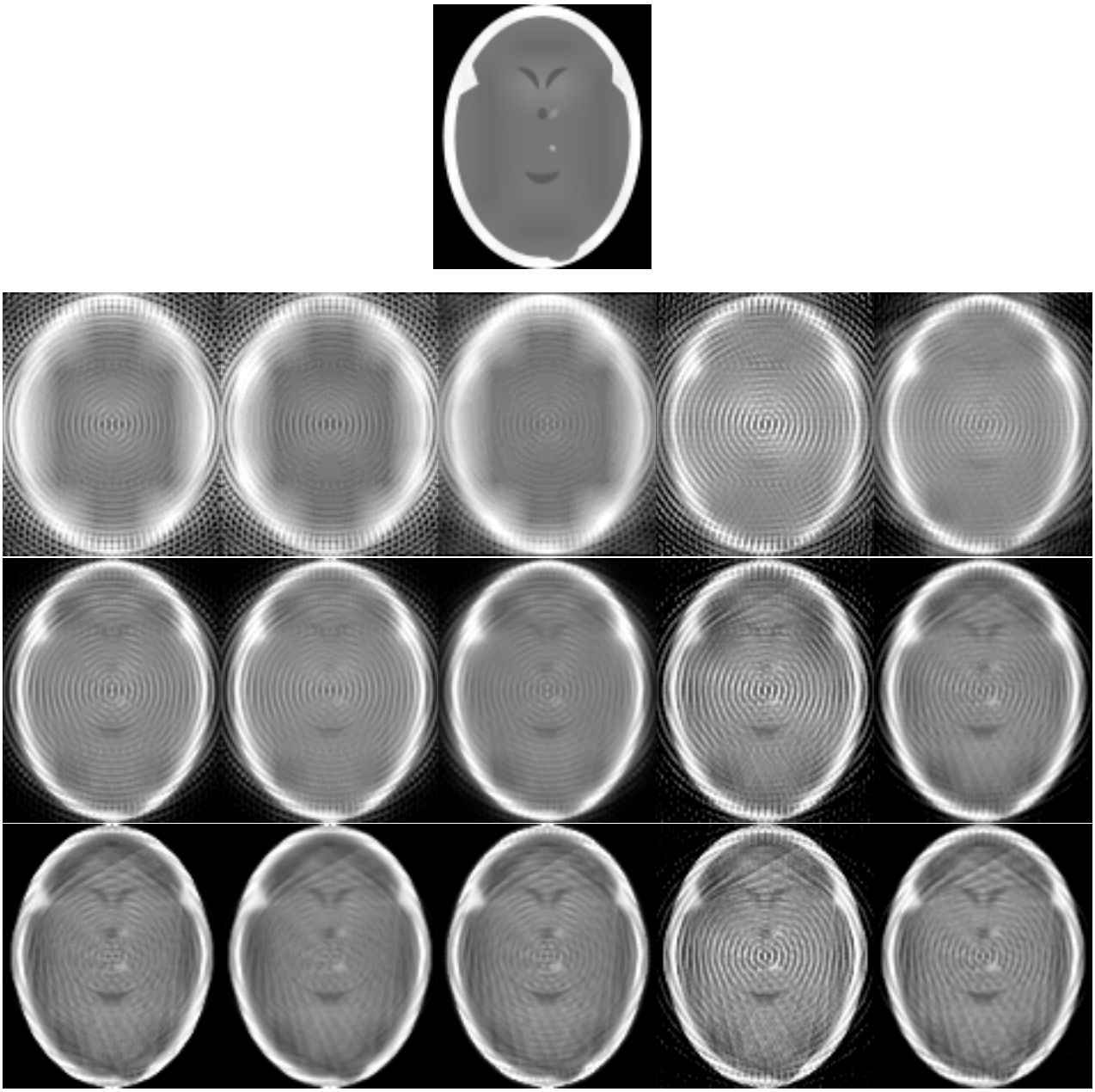

Figure 4.12: Reconstructed images in strongly under-determined system. column 1,2 and 3 respectively - EMAP+LBF with $\gamma = 2$, EMAP+LBF with $\gamma = 5$ and EMAP+blob, after 1, 10, and 50 iterations from uppermost to lowermost row. Column 4,5 - ART+LBF and ART+Blob after 1, 5, 20 iterations from uppermost to lowermost row

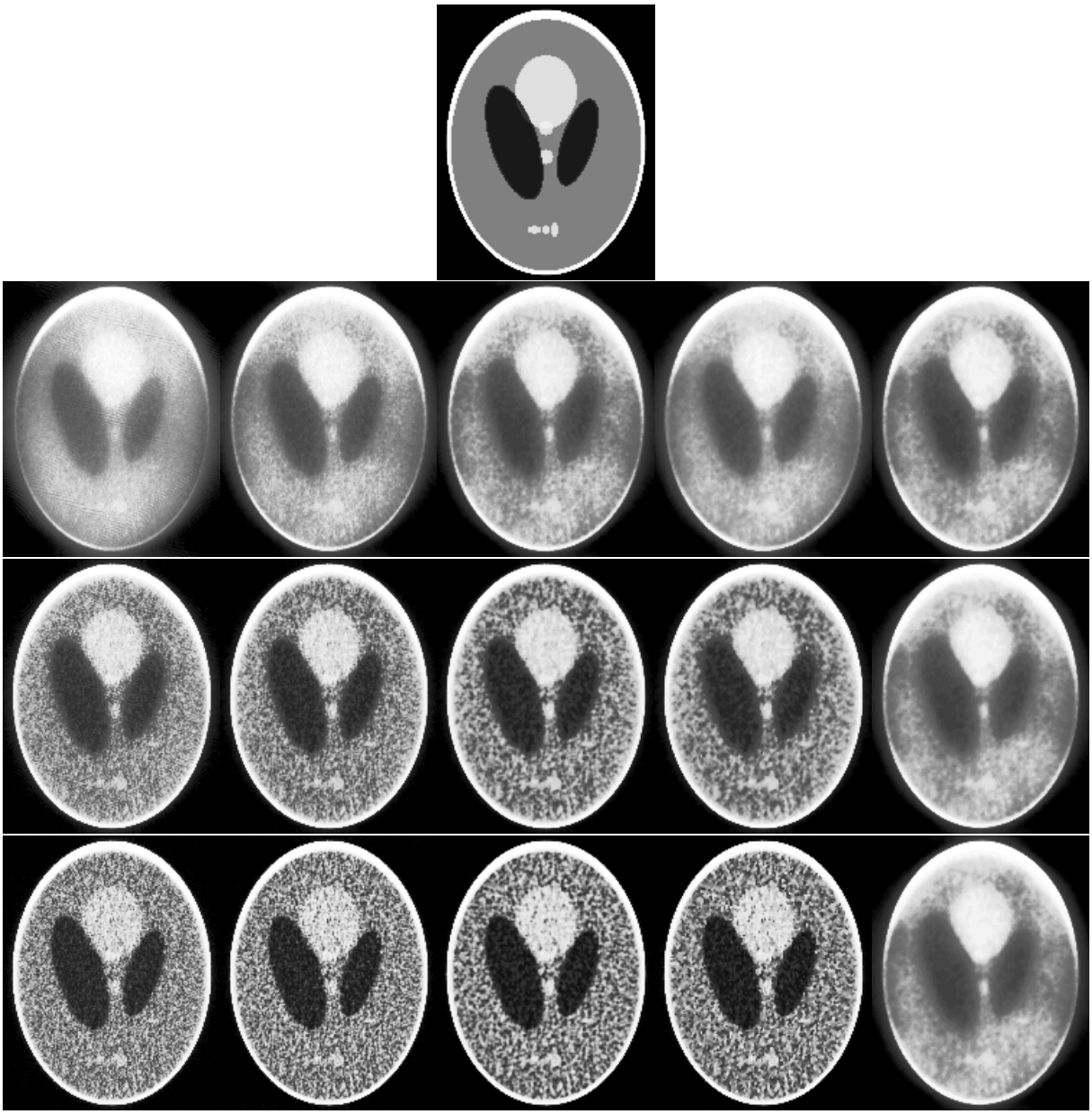

Figure 4.13: Shepp-Logan with noise and reconstructed images using ART after 1 (row 2), 3 (row 3), and 7 (row 4) iterations. Reconstructed images for PBF, LBF and Blob are in columns 1,2,3 in that order. LBF and Blob with selective smoothing are in Columns 4,5 respectively

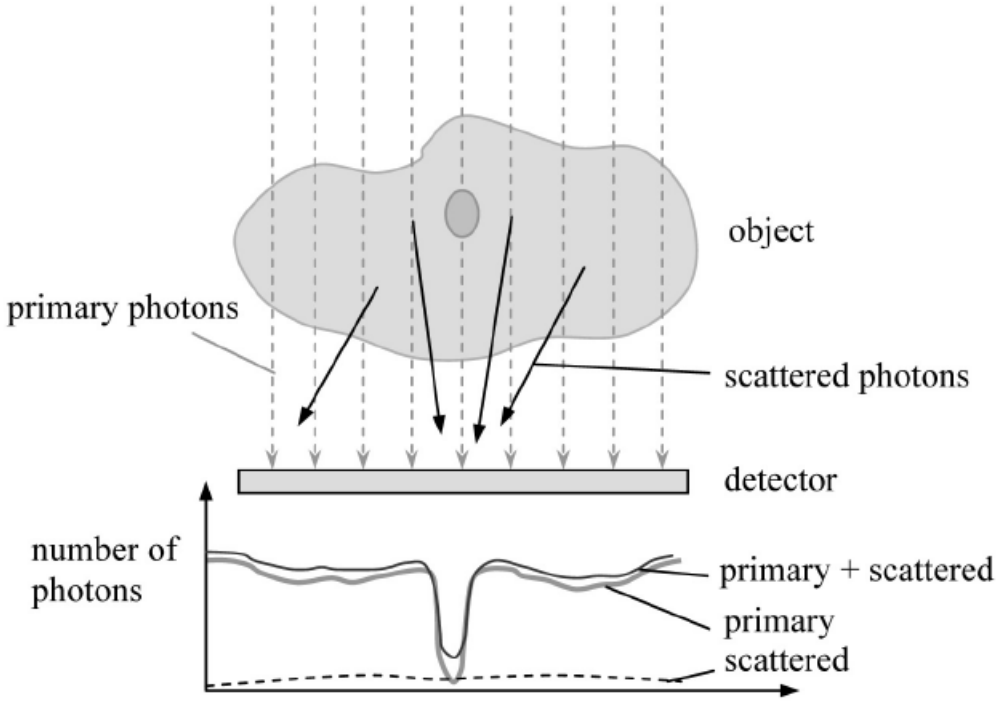


Figure 4.14: Schematic representation of the effect of Compton scattering, taken from [19]

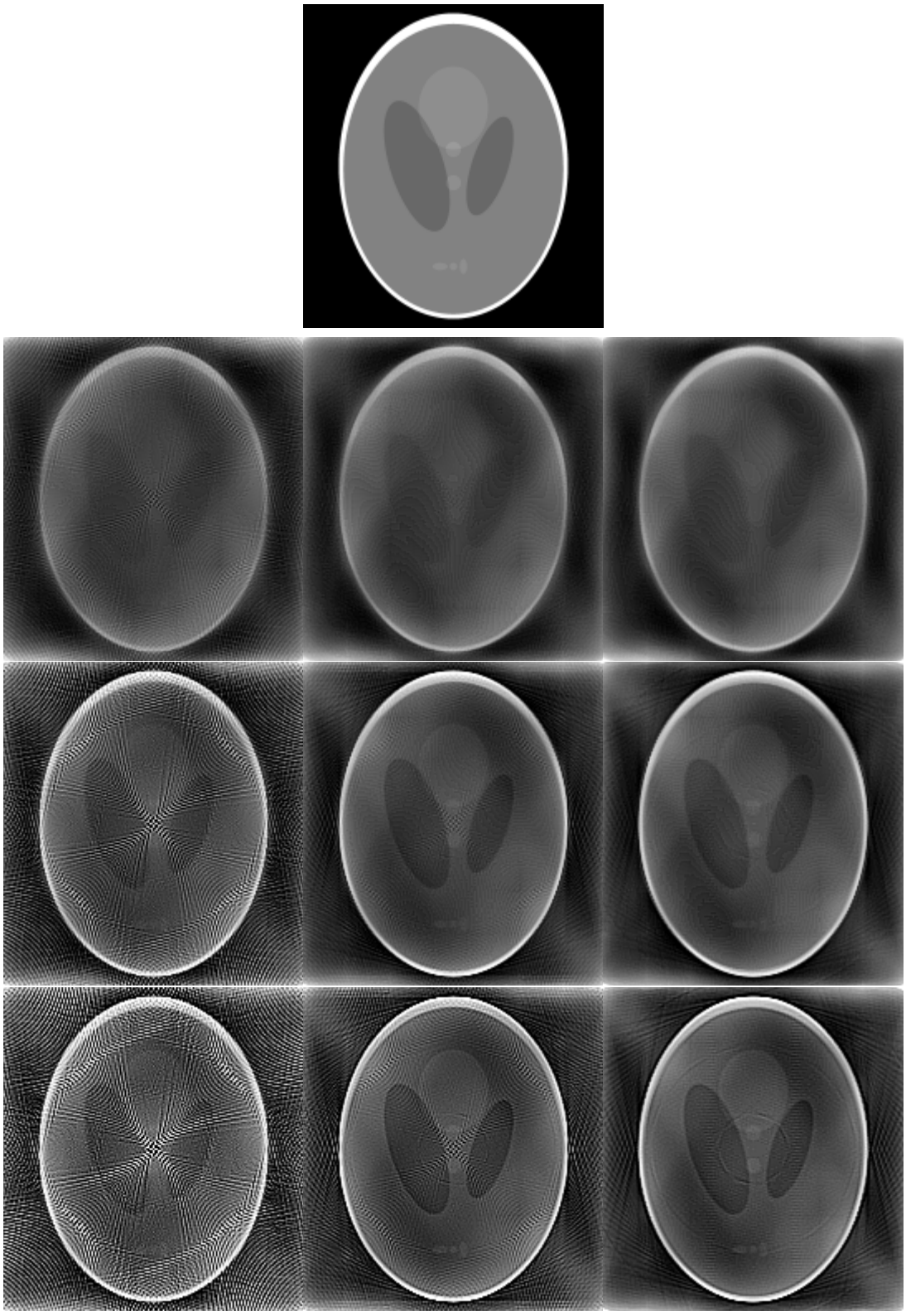

Figure 4.15: Shepp-Logan phantom and the affect of scatter noise and reconstructed images using ART after 1 (row 2), 5 (row 3), and 20 (row 4) iterations. Reconstructed images for PBF, LBF and Blob are in left, middle, and right in that order.

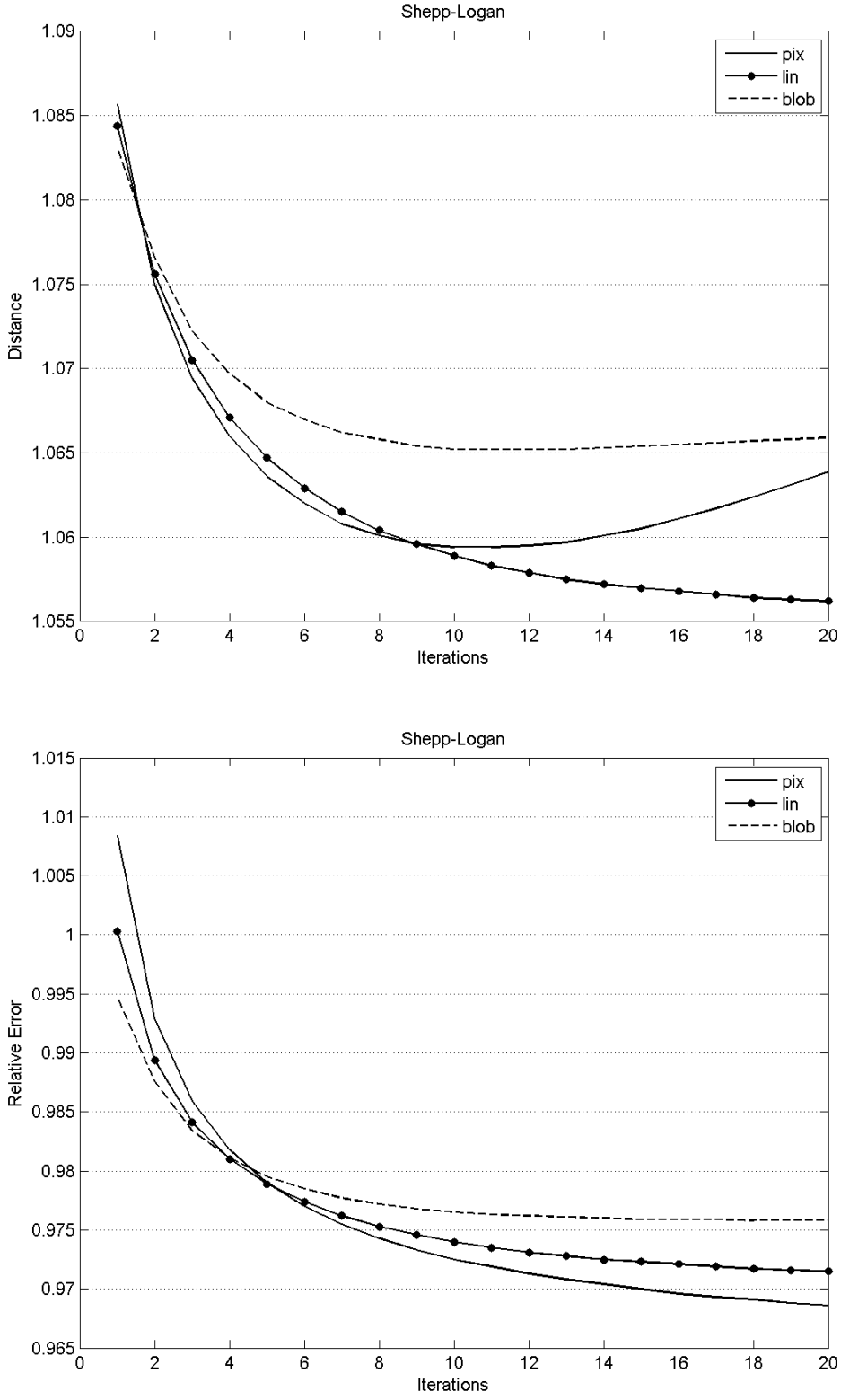


Figure 4.16: Distance and relative error measures for Shepp-Logan with scatter noise

# Chapter 5

# Conclusions and future research

In general speaking we can conclude easily that the linear basis function produce superior images over pixel basis function, under almost any configuration. It is reasonable to assume that higher-order Newton-Cotes basis function will produce even better results. However, it may be that a significant improvement in the reconstructed images will be received only up to some small order. Another issue is the computational cost, which its 2 main issues are the computation of the coefficients, and the number of non-zero coefficients for the computation of the line integral.

## 5.1 Image quality

The reconstructed images produced by the linear basis functions are superior to the pixel basis functions, both in qualitative and quantitative aspects. This was expected prior to the results of this research, by mathematical justifications . However, compared with Blobs, the picture is more complicated. For noiseless and almost determined system - In a qualitative judgment, compared to pixel basis function, the images in linear basis function have reduced amplitude of artifacts such as rings, salt & pepper, but still higher than those induced in Blobs. However, the linear basis function can give better contrast in images, especially small objects, such as tumors, which can assimilated in the surrounding background when using Blobs or pixel basis function. In the quantitative aspect, the linear basis function showed much better measures then pixel basis functions, and was as good as Blob, and, as was seen at 4.4.2, can manage highly detailed example, which pixel basis badly handle, and Blob is

catastrophic.

### Under-determined system and noise

In any case LBF images were much better than those of PBF, both in terms of visual subjective analysis, and quantitative measures. In the context of comparison between Blob and LBF, there were cases where the LBF was competitive to Blob (Scatter noise, and under-determined system due to projection resolution), and in other cases Blob had shown superior results. In this work, we used mainly ART as the reconstruction algorithms, with least squares as minimization criterion. However, in cases of under-determined system and noisy measurements, the use of regularization in the reconstruction steps in those environments is much preferred [19], and common algorithms with this element are the quadratic optimization and EMAP. With the later, it was shown that linear can be competitive and arguably better than Blob. The example with EMAP raise some thoughts, whether Blob isn't an alternative for regularization for reconstruction algorithms which use least squares as their minimization criterion. One sought after direction is investigating various types of regularization to determine which is best fit for LBF.

## 5.2 Computational cost

The computational cost of the linear basis functions are heavier then the pixel basis functions in the computation of the weights, but much cheaper than blob basis functions for any good configuration of the blob grid. The computation of the linear coefficients for each ray-pixel intersection requires significant extra computation time over the pixel basis function, since it additionally compute the bilinear blending functions . In some profiling tests, it was shown that it cause the computation of the linear basis function to be between 3 to 4 times slower than pixel basis function weights computation. In addition weights summation with minimum space also significantly slows down the computation, but it can be replaced by faster method, see 3.5. Compared to Blob, we have seen that LBF computation is about 3 times faster than However, those computation can be better optimized in the future. Further more, using storage for pre-calculation of weights, and symmetry considerations, also reduce the computation cost gaps up to the differences in line integrals computations. Statically,

we found that the linear basis function method produces between 1.5-2 times more non zero weights than the pixel basis function method, which is effectively the differences of computational cost of applying line integration, and 1.5-2 times less than the Blob basis function, using its default parameters. Additional note is that LBF and PBF are purely geometric, and can be calculated once for a constant configuration, regardless of the scanned phantom, while Blob is often fine tuned according to the measurement data.

## 5.3 Future Work

Clearly, more work is required to fully evaluate the new class of data representation. The following are some additional topics for further research:

- Comparisons with other data models, such as blobs.
- Tests with various types of data, including clinical sinograms, noisy and/or low contrast data, and limited measurements (resulting in under-determined linear systems).
- Experiments with other reconstruction algorithms.
- Examination of the quadratic model (based on Simpson's rule), which can be expected to provide even better results.
- Fully 3D reconstructions.
- Applications to PET and electron tomography.
- Experiments with other integral approximation methods, such as Gauss-Kronrod and Clenshaw-Curtis
- Expand the model to other types of ray integrals such strip rays
- Optimization of linear basis functions computation to improve computation time.
- Adapt the model according to the scanning resolution (under-determined and overdetermined systems)

# Chapter 6

# Software package contribution

## 6.1 Snark09 software

The linear model was implemented in the Snark09 and Snark05 Packages, and can be supplied as a special versions. It also include the Grid Traversal algorithm that can replace DDA for any matter.

Main contributions:

- Implementation of the linear basis functions.
- Another option for BASIS in input file. User can choose between 3 basis: Pixel, Linear, and Blob
- Implementation of Grid Traversal algorithm, may be an alternative to DDA anywhere.
- Support for weights storage, for pixel basis and for linear basis. There is an option to use the memory for storing the weights and pixel list calculated after a first iteration, and reconstruction algorithms can reuse this storage anytime for computing ray projections.
- Support for symmetry consideration in storage

```
BASIS PIXEL
*
STOP ITERATION 40
*
```

```
EXECUTE AVERAGE ART
ART WITH PIXEL BASIS
ART3 RELAXATION CONSTANT = 0.05
CONSTRAINT ART2
*
BASIS LINEAR
*
STOP ITERATION 40
*
EXECUTE AVERAGE ART
ART WITH BILINEAR BASIS
ART3 RELAXATION CONSTANT = 0.05
CONSTRAINT ART2
*
BASIS BLOB 2.062941955043030769426443141541 5 11.2828631556
1.154700538379251529018297561003 9
*
STOP ITERATION 40
*
EXECUTE AVERAGE ART
ART WITH BLOB BASIS
ART3 RELAXATION CONSTANT = 0.05
CONSTRAINT ART2
```

## New files:

**grid_traverse.cpp, grid_traverse.h** The files containing the implementation of the new suggested algorithm for grid traversal

**wray_bl.cpp, wray_bl.h** The files containing the implementation for the bilinear weighting algorithm

**blending.h** Contains the constants and definitions used by the linear basis

**wray_DDA.cpp, h** - the old **wray** with 3 more parameters returned: *cases*, *cc*

## Updated files:

**pseudo.cpp, pseudo.h** Support for coefficient and symmetry cases storage

**art.cpp** Added the option choose between weights memory used pseudo, or non-memory(Snark's pseudo) pseudo ,

**dist.cpp** Added the option to choose between weights memory used pseudo, or non-memory (Snark's pseudo) pseudo

**bldlst.cpp** Alternate different allocation sizes for pixel base and linear base

**wray.cpp** It is now a "proxy" between pseudo and real integral method, (the original **wray** has moved to **wray_DDA.cpp,h**)

**basis.cpp** Added an option to use linear basis from input file

# Bibliography


[1]

[2] N. M. Laird A. P. Dempster and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. of the Royal Statistical Society, Series B (Methodological)*, 39(1):1–38, 1977.

[3] M. Gruska C. Best R. Hegerl W. Baumeister A. P. Leis, M. Beck and J. W. Leis. Cryo-electron tomography of biological specimens. *IEEE Signal Process. Mag.*, 23(3):95–103, 2006.

[4] National Electrical Manufacturers Association. *Digital Imaging and Communications in Medicine (DICOM).* The Association, 1993.

[5] J. Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2):100–106, 1977.

[6] J.E. BRESENHAM. Algorithm for computer control of digital plotter. *IBM Syst. J.*, 4:25–30, 1965.

[7] C. L. Byrne. A unified treatment of some iterative algorithms in signal processing and image reconstruction. *Inverse Problems*, 20:103–120, 2004.

[8] Yair Censor, Paul P. B. Eggermont, and Dan Gordon. Strong underrelaxation in kaczmarz's method for inconsistent systems. *Numerische Mathematik*, 41(1):83–92, April 1983.

[9] T. Elfving E. Artzy and G. T. Herman. Quadratic optimization for image reconstruction, ii. *Computer Graphics & Image Processing,*, 11:242–261, 1979.

[10] SS Furuie, GT Herman, TK Narayan, PE Kinahan, JS Karp, RM Lewitt, and S. Matej. A methodology for testing for statistically significant differences between fully 3d pet reconstruction algorithms. *Physics in medicine and biology*, 39(3):341, 1999.

[11] E. Garduno and G. T. Herman. Optimization of basis functions for both reconstruction and visualization. *Discrete Applied Mathematics*, 139:95–111, 2004.

[12] K. M. Hanson and G. W. Wecksung. Local basis-function approach to computed tomography. *Applied Optics,*, 24(23):4028–4039, 1985.

[13] G. T. Herman. *Image Reconstruction From Projections: The Fundamentals of Computerized Tomography.* Academic Press, New York, 1980.

[14] G. T. Herman. *Fundamentals of Computerized Tomography: Image Reconstruction From Projections, 2nd edition.* Springer„ 2009.

[15] G. T. Herman and A. Lent. Quadratic optimization for image reconstruction, i. *Computer Graphics& Image Processing*, 5:319–332, 1976.

[16] G.T. Herman, A.R. De Pierro, and N. Gai. On methods for maximum a posteriori image reconstruction with a normal prior. *Journal of Visual Communication and Image Representation*, 3(4):316–324, 1992.

[17] G.T. Herman and D. Odhner. Performance evaluation of an iterative image reconstruction algorithm for positron emission tomography. *Medical Imaging, IEEE Transactions on*, 10(3):336–346, 1991.

[18] G.T. Herman and KT Yeung. Evaluators of image reconstruction algorithms. *International Journal of Imaging Systems and Technology*, 1(2):187–195, 1989.

[19] J. Hsieh. *Computed tomography: principles, design, artifacts, and recent advances*, volume 114. Society of Photo Optical, 2003.

[20] D. Gordon J.-J. Fernandez and R. Gordon. Efficient parallel implementation of iterative reconstruction algorithms for electron tomography. *J. of Parallel and Distributed Computing*, 68((5))::626–640, May 2008.

[21] J. M. Carazo J. J. Fernandez and I. Garcia. Three-dimensional reconstruction of cellular structures by electron microscope tomography and parallel computing. *J. Paral. Distrib. Computing*, 64:285–300, 2004.

[22] J. Roca I. Garcia M. H. Ellisman J. J. Fernandez, A. F. Lawrence and J. M. Carazo. High performance electron tomography of complex biological specimens. *J. Struct. Biol*, 138:6–20, 2002.

[23] H.E. Johns et al. The physics of radiology. 1983.

[24] S. Kaczmarz. Angenaherte auflosung von systemen linearer gleichungen. *Bulletin de lAcademie Polonaise des Sciences et Lettres*, A35:355–357, 1937.

[25] Z. Kalogiratou and T. E. Simos. Newton-cotes formulae for long-time integration. *J. of Computational & Applied Mathematics*, 158:75–82, 2003.

[26] T. L. Kay and J. T. Kayjia. Ray tracing complex scenes. In *SIGGRAPH Proceedings*, volume 20, 1986.

[27] R. M. Lewitt. Multidimensional digital image representations using generalized kaiser-bessel window functions. *Journal of the Optical Society of America A,*, 7((10)):1834–1846, 1990.

[28] R. M. Lewitt. Alternatives to voxels for image representation in iterative reconstruction algorithms. *Physics in Medicine and Biology,*, 37:705–716, 1992.

[29] S. Matej and R. M. Lewitt. Practical considerations for 3-d image reconstruction using spherically symmetric volume elements. *IEEE Transactions on Medical Imaging*, 15:68–78, 1996.

[30] F. Natterer and F. Wubbeling. *Mathematical methods in image reconstruction.* SIAM, Philadelphia, PA, 2001.

[31] G.T. Herman R. Davidi and J. Klukowska. *SNARK09: A Programming System for the Reconstruction of 2D Images from 1D Projections,*. http://www.snark09.com.

[32] R. Bender R. Gordon and G. T. Herman. Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and x-ray photography. *J. of Theoretical Biology,*, 29(3):471–481, Dec 1970.

[33] T. F. Fliervoet W. Voorhout R. H. M. Schoenmakers, R. A. Perquin and H. Schirmacher. New software for high resolution, high throughput electron tomography. *Microscopy and Analysis,*, 108:5–6, 2005.

[34] G. T. Herman R. Marabini and J. M. Carazo. 3d reconstruction in electron microscopy using art with smooth spherically symmetric volume elements (blobs). *Ultramicroscopy*, 72::53–65, 1998.

[35] L.A. Shepp and B.F. Logan. The fourier reconstruction of a head section. *IEEE Trans. Nucl. Sci*, 21(3):21–43, 1974.

[36] H. M. Shieh and C. L. Byrne. Image reconstruction from limited fourier data. *Journal of the Optical Society of America A*, 23:2732–2736, 2006.

[37] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, New York„ 1980.

[38] J. J. Fernandez O. Terasaki R. Ryoo T. J. V. Yates, J. M. Thomas and P. A. Midgley. Three-dimensional real-space crystallography of mcm-48 mesoporous silica revealed by scanning transmission electron tomography. *Chemical Physics Letters,*, 418:540–543, 2006.

[39] M. Unser. Splines: a perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, 16((6)), Nov 1999.

[40] WQ Yang, DM Spink, TA York, and H. McCann. An image-reconstruction algorithm based on landweber's iteration method for electrical-capacitance tomography. *Measurement Science and Technology*, 10(11):1065, 1999.

[41] E. E. Gualtieri Y.-L. Hsieh J. S. Karp S. Matej M. J. Parma C. H. Tung E. S. Walsh M. Werner Z. Hu, W. Wang and D. Gagnon. An lor-based fully-3d pet image reconstruction using a blob-basis function. In *Nuclear Science Symp. Conf. Record*, volume 6, pages 4415–4418, Los Alamitos, CA, USA„ 2007. IEEE.

# Appendix

## Symbolic computation of coefficients

To validate the correctness of the coefficients we used the following simple symbolic code

```
clear all
syms
syms x y x1 x2 y1 y2 dx dy t
syms N00 N01 N10 N11
P1 = [x1 y1]; P2 = [x2 y2];
P = (1-t)*P1 +t*P2;
x = P(1); y = P(2);
L00=(1-x)*(1-y); L01=(1-x)*y; L10=x*(1-y); L11=x*y;
L = N00*L00+N01*L01+N10*L10+N11*L11;
I = int(L,t,0,1);
% The second substitution necessary due to weakness of Matlab
I2 = subs(I, {x2-x1,y2-y1,x1-x2,y1-y2}, {dx,dy,-dx,-dy});
% Get the coefficients
c00 = simplify(subs(I2, {N00, N01, N10, N11}, {N00, 0, 0, 0})/N00);
c01 = simplify(subs(I2, {N00, N01, N10, N11}, { 0, N01, 0, 0})/N01);
c10 = simplify(subs(I2, {N00, N01, N10, N11}, { 0, 0, N10, 0})/N10);
c11 = simplify(subs(I2, {N00, N01, N10, N11}, { 0, 0, 0, N11})/N11);
fprintf(' c00 = %s\n c01 = %s\n c10 = %s\n c11 = %s\n', ...
          char(c00), char(c01), char(c10), char(c11));
```

# Mathematical background

## Blobs and other models for discretized images

Assume first that we have an $n$-element grid which is a division of the square $[0\ n] \times [0\ n\}$ into $n^2$ squares which are called pixels. Each square has its center at the points $(x_i,\ y_j) = (i + \frac{1}{2},\ j + \frac{1}{2})$, where $0 \leq i, j < n$ and the pixel $(i, j)$ is represented by the square $[i\ i+1] \times [j\ j+1]$.

A discretized image $\hat{f}(x, y)$ is an approximate representation of a two-dimensional function $f(x, y)$

$$\hat{f}(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} b_{ij}(x, y)$$

where $b_{ij}$ are the *basis functions*. These basis functions are supposed to fade as we move away from the pixel center.

Most of the models assume that the functions $b_{ij}(x, y)$ are shifts of a single function $b(x, y)$

$$\hat{f}(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} b(x - x_i,\ y - y_j)$$

Other models assume a hexagonal division of the plane, where the grid is given by the grid points $\left(\frac{\delta}{2} i, \frac{\sqrt{3}\delta}{2} j\right)$ where $i$ and $j$ are integers such that $i + j$ is even. The value of $\delta$ is the hexagonal grid's *sampling distance*.

### Pixel Model

This model assumes that the $b(x, y)$ is a product of step functions $s_1(x) \cdot s_1(y)$.

$$b(x, y) = \begin{cases} 1 & \text{if } 0 \leq x, y \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

.

**Blob model**

For the description of this model we will express the basis function in polar coordinates $b(r, \phi)$. Usually they are radial basis function meaning that they are not dependent on the angle $\phi$. The blob model that we are referring to is the one used in SNARK based on the *Kaiser-Bessel* window\footnote{In terms of signal-processing, the Kaiser-Bessel window is an approximation to a restricted time duration function with minimum energy outside some specified band (see Paul Bourke HTTP://paulbourke.net/miscellaneous/windows/)}.

The SNARK model is a hexagonal pixel model. If $I_k$ denotes the modified Bessel function of the first kind of order $k$, then the SNARK implementation is expressed by the following definition

$$b_{a,\alpha,\delta}(r,\phi) = \begin{cases} C_{a,\alpha,\delta} \cdot \frac{I_2\left(\alpha\sqrt{1-\left(\frac{r}{a}\right)^2}\right)}{I_2(\alpha)} \left(1-\left(\frac{r}{a}\right)^2\right) & \text{if } 0 \le r \le a \\ 0 & \text{otherwise} \end{cases}$$

where the constant $C_{a,\alpha,\delta} = \frac{\sqrt{3}\delta^2\alpha}{4\pi a^2 I_3(\alpha)}$ depends on

$a$ the radius of the blob

$\alpha$ a non negative real number that controls the blob's shape

$\delta$ the *sampling distance* of the hexagonal grid.

This large number of parameters require a careful tuning and they seem quite sensitive to small changes (see an example in pp. 121 where a change from $a = 0.1551$, $\alpha = 11.2829$ $\delta = 0.0868$ to $a = 0.16$, $\alpha = 11.28$ $\delta = 0.09$ will make sensible differences in the tomographic reconstruction)

# Idealized images for computer reconstruction evaluation

Shepp Logan proposed a handy method for constructing idealized images that can simulate a real phantom. These images are composed of ellipses of constant density. Each ellipse has its own density, position and inclination. A constant intensity ellipse seems to be a suitable object due to the fact that its projection can be analytically computed. Assuming that a ray

passes at an angle $\theta$ through the center of an inclined ellipse with semi-axes and $a$ and $b$, and $s$ denotes the distance of the ray from the origin then we get a projection

$$p_\theta(s) = \begin{cases} 2ab\frac{\sqrt{s_m^2 - s^2}}{s_m^2} & \text{if } |s| \leq s_m \\ 0 & \text{otherwise} \end{cases}$$

where $s_m = \sqrt{a^2 cos^2\theta + b^2 sin^2\theta}$ is the half-width of the projection in the direction $\theta$.

The advantage of this approach is that we compare with data uncorrupted by the physical noise of the data acquisition device. Using about 11 ellipses Shepp-Logan's model modeled quite realistically a head with skull, ventricles and tumors.