Comparison of different tomographic reconstruction techniques

Fredrik Dahlberg, José Jesús Fernández

January 22, 2002

Abstract

Computer Tomography enables the visualization of an objects interior without opening it up. It is a technique used in many fields from medicine to geological examinations. To get information about an object, exterior measurements are performed for example by the use of x-rays. Many different mathematical algorithms can be used to reconstruct an image of the objects interior using the available information from the measurements. This report explains and compares seven reconstruction algorithms. Some of them are used today and some only exist as ideas. It also examines how the choice of basis function (pixel or blob) influences the quality of the reconstructed image. The results showed that blobs were superior in terms of image quality to pixels as basis function and that two algorithms (ART and AVSP) were superior in terms of image quality to the other five (FBP, SIRT, SART, CAV and BICAV).

Contents

Chapter 1

Introduction

The need to see the interior of an object is a problem that arises in several contexts. A brain surgeon for example, needs to know where the tumor is situated before operating. Geologists want to localize precious metals without having to excavate whole land areas. The interior structure of a molecule can not be seen by breaking the molecule and looking at the parts. To solve these and other similar problems it is possible to use tomography. It is a technique that has been used in a wide range of areas, from the mapping of underground resources via crossborehole imaging [1], to the visualization of a body's inner temperature through measurement of the emitted heat (radiometry). With its help a brain tumor can be discovered or the molecular structure of a virus be determined.

Tomography refers to "...the cross-sectional imaging of an object from either transmission or reflection data collected by illuminating the objects from many different directions."[1] This means it enables the production of an image of the interior, by measuring something accessible from the exterior. The nature of the measured data changes from area to area. In medicine, x-rays are often used while in other cases it can be emitted heat or light bouncing of the object. Each fraction of the measured data provides a bit of information about the interior parts of the object, and with enough such bits of information it is possible to obtain an image of the interior. The quality of this image normally gets better with an increasing amount of provided data.

In the case of x-rays, pictures of an object are taken by letting rays penetrate the body from one side. On the other side of the body the rays will be collected as a trace, an x-ray image. This image gives information describing the matter that the rays have passed through. If you have several

of these images showing the same body from different angles, the interior can be reconstructed.

The mathematical grounds for Tomography were first studied by Radon in 1917 [2], but it was not until 1972 that the first CT (Computed Tomography) scanner was invented. For this achievement the inventors, the British and South African electronics engineers G.N. Hounsfield and Alan McCormack received the Nobel Prize in medicine 1979. Figure 1.1 shows a modern CT scanner used in hospitals today. The patient lies on the table, which can slide in and out of the gantry. The gantry is the frame housing the x-ray tube, collimators and detectors in a CT machine. It has a large opening into which the patient is inserted. When the patient is inside the gantry a number of x-ray images, about a thousand, are taken.

Figure 1.1: A modern CT scanner. (Taken from http://www.radiologyinfo.org).

What is happening inside the CT scanner is described by figure 1.2. It shows a very simple model of rays passing through an object. In order to relate it to figure 1.1, the cylinder is the person and the rays are emerging and being received by the gantry, the part with the big hole on the machine in figure 1.1.

In figure 1.2 the rays of each scan are parallel. It is also possible to send out rays in a fan shaped manner, but this master thesis will only deal with the parallel case. The attenuation of each ray is measured by a detector on the opposite side of the object from the emitter. The measurements are called projections. There is a number of different techniques used to reconstruct these projections into an actual image of the interior parts of the body. This master thesis addresses the comparison of different existing techniques in terms of correctness, speed and complexity.

So what are the demands on a good reconstruction? Most important is

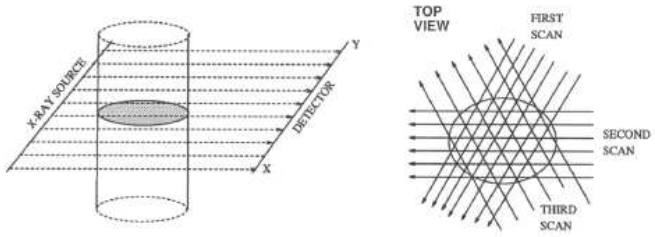


Figure 1.2: A simple scanning system. (Taken from[3])

that it gives a satisfactory image of the object in focus. What is satisfactory might change significantly depending on what one is looking for. Another important demand, especially in medicine, is that the number of rays, used to obtain the demanded image, should be minimized. This is important since x-rays are exposing the body to potentially dangerous radiation. Also when reconstructing very small objects (such as molecules) from pictures taken by an electron microscope, it is important to keep the number of projections to a minimum. Here the object is exposed to electrons fired from an electron canon. If too many projections are used, the object will deteriorate under the impact of the electrons during the process. This will cause bad reconstructions, since the projections will be from essentially different objects (the same object at different stages of deterioration). Of importance is also the complexity of the reconstruction algorithm used. With an algorithm using a fewer amount of mathematical operations, a less powerful computer might be used. This certainly increases the possibility of using it in many places around the world. If speed is of utmost importance, the possibility to parallelize the algorithm is important as well. The calculations can then be shared by a number of processors thus speeding up the process.

The rest of the report is organized in the following way. In chapter 2 the foundations of tomography and the different reconstruction algorithms are explained. Chapter 3 describes how the implementation of the algorithms was carried out. In chapter 4 the differences between the algorithms are discussed and the method used to compare them is explained. Chapter 5 contains the results obtained from the tests. Finally, the conclusions drawn from the tests are presented in chapter 6. The report also includes an appendix which contains: some additional results, an in depth explanation of the Fourier slice theorem and a short description of the program used to perform the tests.

Chapter 2

Reconstruction techniques

As mentioned in the introduction, several different techniques can be used to reconstruct an image from its projections. Taking into account the number of parameters possible to set for each technique, one ends up with an immense number of possibilities. This chapter will try to give an overview and explanation of the techniques studied in this project. It will give an idea of how difficult it is to know which way to take in the jungle of different options. There are two main types of reconstruction algorithms: analytical algorithms and algebraic algorithms. The analytical algorithms are based on a continuous description of the image and the data. They formulate a continuous solution which is discretized before being implemented by a computer program. These type of algorithms are represented here by the filtered backprojection algorithm. All the other algorithms mentioned in this report are algebraic. The algebraic algorithms start from a discretized version of the image formed by a limited set of components, for example pixels.

To start with some of the common concepts used in all of the algorithms, will be explained. After that follows a section on each algorithm studied in this master thesis.

2.1 Reconstruction concepts

2.1.1 The phantom

The idea of tomography is to see the interior of a body through non-invasive measurements — measurements that do not damage the body. In real, experimental situations, it is not possible to know exactly what the object looks like inside, i.e. the exact solution of the problem. Consequently, there is no way to objectively verify the correctness of our results when

working with real objects (e.g. a human body). That is why we construct an artificial object, a phantom, on which we can test the reconstruction methods. This object can exist both as a real object, made in plastic or some other material, or just as values in a computer. The important thing is that its characteristics are known exactly. Projections on the phantom are made or simulated in a computer, and the obtained measurements are used to reconstruct the object. With this procedure we can easily compare the original object to the reconstructed image, which tells us how good the used reconstruction method was. One phantom that is commonly used in the area of tomography is the Shepp-Logan head phantom (figure 2.1). The Shepp-Logan phantom contains ellipses with different absorption properties that resembles the outline of a head [1]. Since so many are using it, it has turned into something of a standard. This enables scientists to easier compare their results. The Shepp-Logan phantom was the only phantom used in this project.

Figure 2.1: The Shepp-Logan phantom

2.1.2 The projection

Put a heap of different objects on the surface of an overhead projector and turn it on. What you will see on the screen is a projection of the objects from one angle. Maybe you can discern some of the shapes but there is

no way to determine, for example, which element is on top of the other or how thick the elements are. However, it might be possible to tell something about the material of the objects. A thin piece of plastic or fabric might let some light pass through, while a stone will block all and show up as a completely black point on the screen. The x-rays going through a body give a similar image of the body's interior. What creates the projection is the fact that the intensity of the x-rays, just like the light, depends on the material they have passed through. The difference between x-rays and light is that the x-rays go through a lot of materials where ordinary light is completely blocked. Figure 2.2 shows the projection, $P_\theta(t)$ of an object. It can be seen as the x-ray shadow of the object.

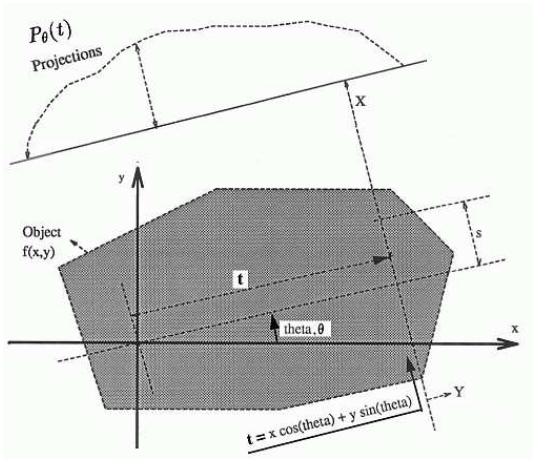


Figure 2.2: The projection of an object. (Taken from[3])

In the remainder of this report the projections will be the source of information for the reconstruction algorithms. Using this information and only this information, it is possible to make a reconstruction of the original object. An essential factor for the quality of the reconstruction is the number of projections available and the number of samples per projection. Normally projections are generated at equal intervals over a range of degrees. The projections used in this project for example are taken between -60^o and 60^o, with 2^o between each projection. The reason for using the limits of $\pm 60^o$ has to do with the mechanical limits of tomography used in microscopy. Therefore, reconstructions obtained from projections over this range of degrees are especially interesting.

The projections are collected in what is called a sinogram. The sinogram

is an image in which each line represents the projection at one angle. The lines are ordered according to the angles in a rising order, in this case from -60^o to 60^o. Figure 2.3 shows how the sinogram is connected to the original image through the projections. The name sinogram comes from the fact that each point in the original image is displayed in the sinogram as a sine curve. The amplitude of the sine curves describe the points' distance from the center and the phase it's angle. This should hopefully be obvious after having looked at figure 2.3. The sinogram can be described as a set of sine curves put on top of each other.

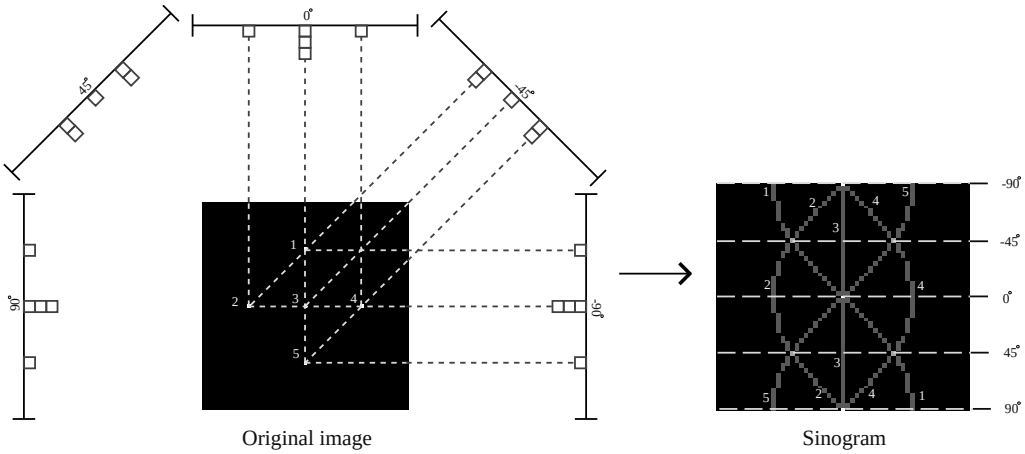


Figure 2.3: The figure shows how the object and projections are related to the sinogram. Each curve in the sinogram is related to one of the five points in the original image, and each row represents the values from one projection.

If one had been able to extract each sine curve from the sinogram, the reconstruction of the image would have been an easy task. Each curve could then have been used to exactly reconstruct a point in the image. Unfortunately this is not possible since the information about the individual curves is lost in the summation of all the curves. Figure 2.4 shows the sinogram obtained from the Shepp and Logan phantom which will be used throughout this report.

It is important to note that the method of collecting data about an object, explained above, is far from the only one. The source of the rays does not have to be on the outside of the body. In PET (Positron Emission Tomography) for example, a pill containing a positron emitting isotope is given to the patient before the examination. This isotope is emitting positrons

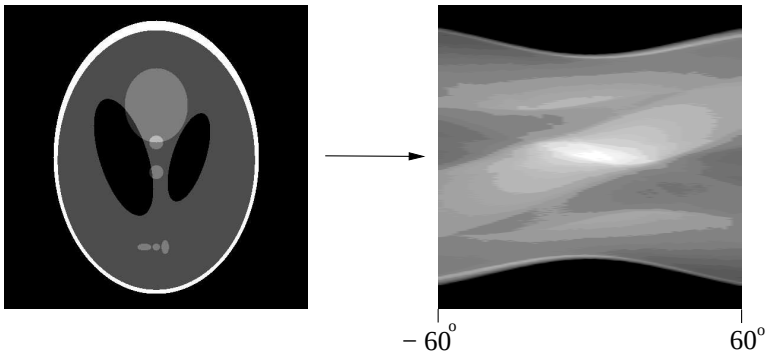


Figure 2.4: The sinogram consisting of projections between -60 and 60 degrees of the shepp and logan phantom

from inside the body which are collected by detectors on the outside. From information on the distribution of the radioactive tracer an image of the interior parts can be obtained.

2.1.3 Basis elements

In order to be processed by a computer, an image has to be discretized. This means that it will be described by a limited set of objects, in this report called *components*. These components will be put in a uniform cartesian grid (with node spacing Δ) consisting of a set of N points represented by $\{(x_j, y_j)\}_{j=1}^{N}$. The image will be represented by a function $f(x, y)$ which is discretized according to

$$f(x,y) \approx \hat{f}(x,y) \equiv \sum_{j=1}^{N} g_j \phi(x - x_j, y - y_j) \tag{2.1}$$

where $\hat{f}(x, y)$ represents the discretrized image, the g_j's are the coefficients of expansion (the weight of each basis function) and ϕ is the basis function. In other words, the image is constructed by a superposition of scaled and shifted copies of the basis function ϕ. Once the basis function is known, we only have to store the coefficients of expansion to retrieve the approximated image. With a denser positioning of the components (smaller node spacing Δ), a better resolution of the approximated image will be obtained. In this paper, the number of components will be denoted by N.

To sum up, the component is the building block of the discretized image, like a brick in a wall or a cell in the body. The shape of this building

block is determined by the basis function, which strongly influences the reconstructed image. The two shapes of "building blocks" considered here are the common *pixel* and the *blob*. Figure 2.5 shows these two basis elements.

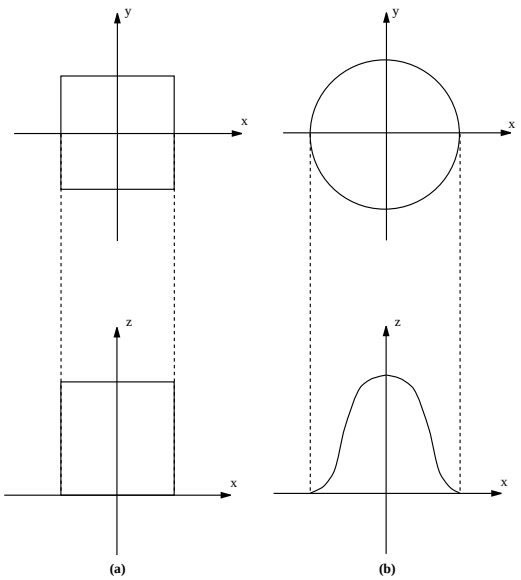


Figure 2.5: The two basis functions; (a) pixel and (b) blob.

The pixel basis is obtained by simply dividing the image into small subsquares. This is expressed by 2.2.

$$\phi(x,y) = \begin{cases} 1 & |x|,|y| \leq \Delta/2 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Reconstructions made using pixels often suffer from a so called "salt and pepper noise" [1], which makes the reconstructed image look as though sprinkled by salt and pepper. For this and other reasons that will become clearer later, the pixel might not be the best way of representing an image.

Therefore this paper will also consider the possibility of using a family of radially symmetric basis elements, called blobs, for which the basis function is represented as

$$\phi(x,y) = b(r) = b(\sqrt{x^2+y^2}) \tag{2.3}$$

where r is the radial distance from the blob center. Their smoothness properties can be controlled by a few parameters and their projections have a

particularly convenient analytical form[4]. A blob is pretty much the opposite of a pixel. It is round and it does not have a constant value. Instead its value is decreasing from the center towards the edges. It is radially symmetric. In opposite to pixels, blobs are arranged on a grid in an overlapping way, allowing a continuous smoothing of the image. The blob is characterized by three parameters: the radius a, a non-negative integer m and a real number α. The latter two variables, m and α, determine the shape of the blob.

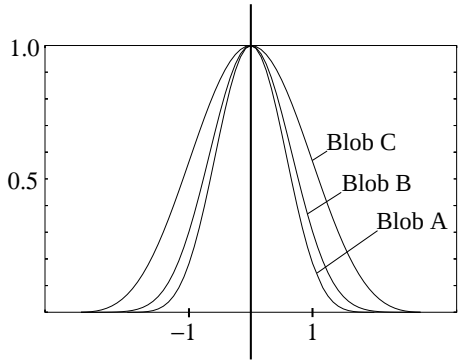
$$b_{m,a,\alpha}(r) = \begin{cases} \frac{\left[\sqrt{1-(r/a)^2}\right]^m I_m\left[\alpha\sqrt{1-(r/a)^2}\right]}{I_m[\alpha]} & \text{for } 0 \leq r \leq a \\ 0 & \text{for } r > a \end{cases} \tag{2.4}$$

Equation 2.4, which describes the blob, is actually a generalization of the Kaiser-Bessel window function [5] from digital signal processing. I_m is the modified Bessel function. The variable r represents the radial distance from the blob center. Once the characteristics of the blob are set, its value depends only on r. This will turn out to be a significant advantage later on. With the parameter m we can control the continuity conditions of the blob. For $m = 0$ the blob is not continuous at the radius a. For $m > 0$ the blob is a continuous function with $m - 1$ continuous derivatives at the boundary. It is desirable to use $m \geq 2$ so that the function and its first derivative are continuous at the boundary [6]. The values for a and α (the taper parameter) that will be used in this report come from a report on image reconstruction techniques for PET[7]. In this report a number of parameter settings were found to be the most appropriate ones, based on the quality of the reconstructions they are leading to. The quality in [7] was determined using both visual and mathematical analysis of the reconstructed images.

The projection of the original object can be seen as a sampling of it. The sampling interval is the distance between the components in the reconstruction, which in this case always is set to one. This means, according to the Nyquist sampling theorem, that the highest frequency in our reconstructed image will be

$$\frac{1}{2 \cdot \text{sample interval}}.$$

We want the fraction of the basis function consisting of frequencies outside this Nyquist range to be as small as possible, since all the information there will be lost. This is the most important criterion to consider when choosing the parameters for the blob function. Another criterion is that there should be a zero crossing in the blobs frequency band at the sampling frequency. If that is the case, the artifacts due to aliasing (the overlapping among the



	Blob A	Blob B	Blob C
radius	2.0	2.795	2.795
α	10.83	16.36	6.72
m	2	2	2

Figure 2.6: The profiles and the corresponding parameters of the blobs that will be used in this report.

replicated Fourier spectra) will be minimized. Considering all this, the three blobs shown in figure 2.6 were found to be appropriate to work with in this project (all of them taken from [7]). They all fulfil the above requirements but have different shapes and involve different resolution limits (a wider blob gives a reconstruction with less resolution).

2.2 Analytical Reconstruction Algorithms

To illustrate the analytical algorithms the filtered backprojection algorithm was chosen. It is based on the Fourier slice theorem which will be explained in the following.

By now the reader should be familiar with the concept of projections. Those can be interpreted as line integrals of the object, $f(x, y)$, each uniquely described by the projection angle and its distance from the center of the object. Since only projections obtained with parallel rays were examined, in this thesis all the line integrals in a projection will have the same angle. As can be seen in figure 2.2, the angle is represented by θ and the distance from the center by t. The line integral will hence be defined as:

$$P_\theta(t) = \int_{(\theta,t)line} f(x, y)\,\mathrm{d}s. \tag{2.5}$$

The function $P_\theta(t)$ is known as the Radon transform of the function $f(x, y)$.

2.2.1 The Fourier slice theorem

The base of the Fourier slice theorem is the fact that the one dimensional Fourier transform of a projection is equal to a slice of the two dimensional Fourier transform of the original object. This means that, given the projection data, it is possible to reconstruct the object using an inverse Fourier transform. Mathematically this can be expressed as:

$$S_\theta(w) = F(w\cos\theta, w\sin\theta) \tag{2.6}$$

where $F(x, y)$ is the Fourier transform of the image and $S_\theta(w)$ is the Fourier transform of the projection. The mathematical reasoning that leads to equation 2.6 can be found in appendix B.

2.2.2 The Filtered Backprojection

In the previous section we saw that, according to the Fourier slice theorem, every projection describes a line of the two dimensional Fourier transform of the image. Hence the information we now have about the original image can be depicted as in figure 2.7. We see that there is a great deal of area not covered by the lines. The further away from the center, the worse is the obtained coverage. In other words, we have less information about the high frequencies.

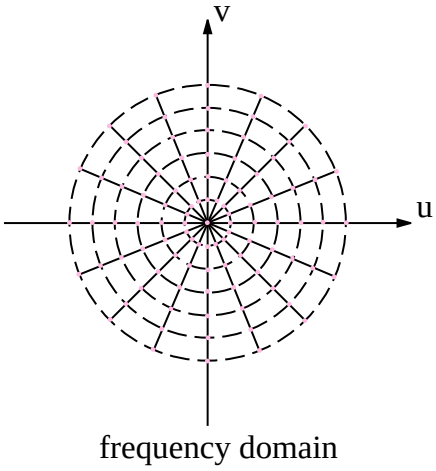


Figure 2.7: The points on one radial line in the figure, represent the information we get from one projection about the 2-D Fourier transform of the original object.

An ideal situation would be to obtain wedge shaped slices, such as the one in figure 2.8 (a), which could cover the 2-D frequency space completely. Since this is not the case, we need to compensate the lack of information by weighting the slices. This means that we change the shape of the slices from lines (figure 2.8(b)) to wedges (figure 2.8(c)). By doing this we will achieve an approximation of the actual 2-D transform of the original image.

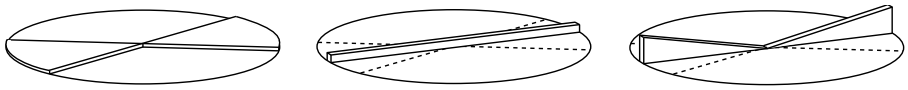


Figure 2.8: The figure shows the frequency domain available from one projection. (a) is the ideal situation and (b) is what we have. To produce an approximation of (a) we have to do a weighting of (b) which will lead to something like (c).

The Fourier transform of the projection, $P_\theta(t)$ is, as mentioned earlier, called $S_\theta(t)$ and is obtained from equation B.2. The simplest way to estimate the pie shaped wedge in the object's Fourier transform is probably to do a weighting of $S_\theta(t)$. The weighting multiplies $S_\theta(t)$ by the width of the desired wedge at each frequency. The width of the wedge is $2\pi|w|/K$, where w is the frequency and K is the number of projections per 180^o. The weighted version of $S_\theta(t)$ is called $Q_\theta(t)$.

In figure 2.8 the effect of the weighting can be seen. It is obvious that it results in an error, but this error can be made as small as desired by using a sufficient amount of projections.

To produce a reconstructed image we need to calculate the inverse Fourier transform of the approximated 2-D Fourier transform of the original image. This can be done by adding the weighted 1-D Fourier transforms of the projections ($Q_\theta(t)$) in the 2-D frequency plane, and then performing a 2-D inverse Fourier transform. A simpler way to carry it out, is to perform an inverse Fourier transform of $Q_\theta(t)$ for each θ and then to calculate the sum of them all. The summation is done by a process called backprojection in which the values are smeared back over the image along the rays that produced the projection. Mathematically, the difference can be described as

$$\mathcal{F}^{-1}\{\sum_{i=1}^{K} Q_{\theta_i}\} = \sum_{i=1}^{K}(\mathcal{F}^{-1}\{Q_{\theta_i}\}) \tag{2.7}$$

where K is the number of projections. The right hand side of 2.7 is easy to implement in a computer, since the 2-D Fourier transform approximation of the original image never has to be calculated or stored.

The summation on the right hand side of 2.7 is usually called Backprojection, since it can be seen as smearing back the values from the projections over the reconstructed image. The fact that these values have been weighted in the frequency plane, in other words filtered, explains the name of the algorithm. Hereby the complete filtered backprojection algorithm [1] can be written as:

For each of the K projection angles, θ:

- Measure the projection.
- Fourier transform it to find $S_\theta(w)$.
- Multiply it by the weighting function $2\pi|w|/K$.
- Sum over the image plane the inverse Fourier transform of the filtered projection (the backprojection process).

2.3 Algebraic Reconstruction Algorithms

The rest of the reconstruction techniques described in this report are all algebraic. To start with a few concepts that apply to all the algebraic algorithms will be explained.

2.3.1 The Equation System

In the algebraic techniques the problem of reconstruction is seen as a system of equations. It consists of M equations, one for each ray, with N unknowns, each representing a component. The system describes the known relations between the attenuation of the rays, the projections and the object. In this thesis the equation system will be represented as

$$\begin{array}{ccccccccc} a_{11}f_1 & + & a_{12}f_2 & + & \dots & + & a_{1N}f_N = & p_1 \\ a_{21}f_1 & + & a_{22}f_2 & + & \dots & + & a_{2N}f_N = & p_2 \\ \vdots & & & & & & & \vdots \\ a_{M1}f_1 & + & a_{M2}f_2 & + & \dots & + & a_{MN}f_N = & p_M \end{array} \tag{2.8}$$

or in matrix form

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_N \end{pmatrix} \tag{2.9}$$

where $f_i, i = 1...N$, is the weight of the i:th component in the reconstructed image, $p_i, i = 1...M$, is the attenuated value from ray i, and $a_{ij}, i = 1...M, j = 1...N$, is a factor that describes the relation between the i:th ray and the j:th component. Each row in the equation system describes the progression of one ray from the source to the detector, since it contains one element for each component, telling how ray i is passing in relation to that. Each part in the summation on the left hand side of 2.8, describes how much the ray is supposed to be attenuated by one component. Since the ray normally only passes through a small fraction of the object, most of these parts will be equal to zero. The sum of the attenuation enforced by each of the N components will be the ray-sum, or projection value, p_i. Therefore, each ray leads to one equation in the system.

The algebraic algorithms are not dependent on the orientation of the rays, since these independently supply the system with information. This shall be seen in contrast to the analytical algorithms, where a certain orientation of the rays is needed in order to perform a Fourier transform. There, the rays were "cooperating" in order to supply information to the reconstruction.

The unknown part in the equation system is of course $(f_1, f_2, \ldots, f_N)$, since this represents, as mentioned before, the reconstructed image. N is the number of components used to describe the image. The known variables are p and A. $p_1, p_2, \ldots p_M$ are the projection values actually obtained from some sort of measurement, for example by x-rays. A is an $M \times N$ matrix describing how each piece of the object — each component — is assumed to affect — attenuate — the ray. The content of A is strongly influenced by the choice of basis function. A wider basis function will for example be crossed by a larger amount of rays, which will lead to fewer non-zero elements in the A matrix. There are many different ways of translating the relation between components and rays into a number. A common way of doing it is to set the value a_{ij} in the A matrix to the ratio of the j:th component that is covered by the i:th ray (figure 2.9). Note that for the pixel, the covered part can be seen as an area, but for blobs it can only be obtained from integrating the blob function over the covered area. The values of A are very important to the resulting reconstructed image that eventually will be obtained. Other ways of describing the relationship between rays and basis functions will be discussed in section 3.1 on page 30.

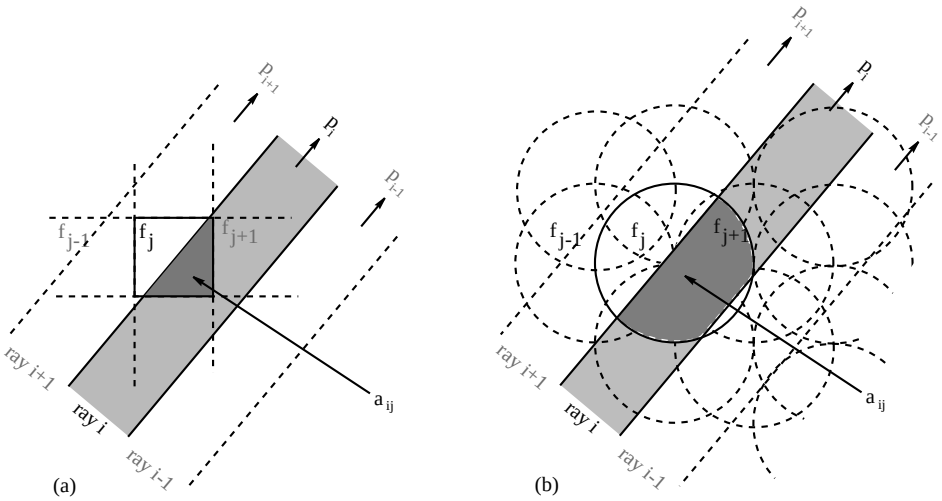


Figure 2.9: The values a_{ij} of the A matrix can be chosen to be the ratio of the pixel (a) or blob (b), f_j that is covered by the i:th ray. Note that in the pixel case the ratio is just the area, while in the blob case you have to integrate over the blob function to get the ratio.

2.3.2 Iterating Towards the Answer

Algebraic algorithms are solved by iteration. The size of the system and the lack of an exact answer makes analytical methods impossible to use. Iterating means that from an arbitrary initial guess, the solution is approached by using the equations one after the other. The order in which the equations are processed has a great impact on the speed and the correctness of the algorithm. It is here that the algebraic techniques mainly differ, which will be apparent in the following sections where the different algorithms are described

The representation of the reconstructed image, $(f_1, f_2, \ldots, f_N)$ can be seen as a single point in an N-dimensional space. Each one of the M equations in 2.8 can then be seen as a hyperplane. When a unique solution exists for the system, there is a single point where all these hyperplanes intersect. This point is the solution to the system. In two dimensions, this can be illustrated by figure 2.10, which shows an approach to the solution using the following equations:

$$\begin{aligned} a_{11}f_1 + a_{12}f_2 &= p_1 \\ a_{21}f_1 + a_{22}f_2 &= p_2 \end{aligned} \tag{2.10}$$

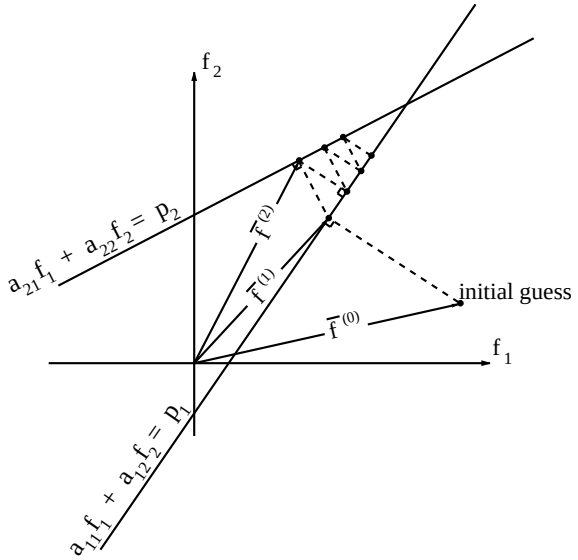


Figure 2.10: An illustration of the iterative process.

The initial guess $\overline{f}^{(0)}$, which can be any point in space, is first projected onto the line described by the first equation. The result, $\overline{f}^{(1)}$, is projected onto the second line, from where that result, $\overline{f}^{(2)}$, is projected back onto the first line and so on. Following this scheme one is approaching the answer to the equation system. If there is a unique solution this will always be reached.

The projection onto the hyperplanes is mathematically described by

$$\overline{f}^{(k)} = \overline{f}^{(k-1)} - \frac{(\overline{f}^{(k-1)} \cdot \overline{a}_i - p_i)}{\overline{a}_i \cdot \overline{a}_i} \overline{a}_i. \tag{2.11}$$

where $\overline{a}_i = (a_{i1}, a_{i2}, \ldots, a_{iN})$, and $\overline{a}_i \cdot \overline{a}_i$ is the scalar product of $\overline{a}_i$ with itself. For a thorough explanation of the origin of equation 2.11, see appendix C.

The projection formula is implemented by looking at the difference between a calculated and a measured projection value. The measured value in

$$\overline{f}^{(k)} = \overline{f}^{(k-1)} - \frac{(\overline{f}^{(k-1)} \cdot \overline{a}_i - p_i)}{\overline{a}_i \cdot \overline{a}_i} \overline{a}_i.$$

is p_i, and the calculated projection value is $(\overline{f}^{(i-1)} \cdot \overline{a}_i)$. The calculated and measured projections are collected in the so called calculated and the

measured sinograms. These play an important role in the implementation of the algorithms, which will be discussed in the next chapter.

All the algebraic algorithms are built on the idea that the calculated and measured sinograms should be identical. In other words, the calculated projection of the reconstructed image should be the same as the measured projection from the real object. The projection formula determines the difference between these two projections and updates the reconstructed image so that the difference is minimized. This updating, which is the right side of 2.11, does not have to be done immediately. As will be seen later on, in some algorithms the average of several different corrections is calculated, which then is used to update the reconstructed image, $\overline{f}$. This will slightly change the appearance of 2.11. In equation 2.11 the correction is the part on the right hand side to the right of the minus sign.

The equation 2.11 is the basis of all the algebraic algorithms. They all use this projection method in order to move along towards the answer in their iteration method. As said before, this will always lead to the correct answer if there is only one intersection point. This does not have to be the case in an overdetermined system, which means that $M > N$. It is easy to imagine, in the two dimensional case, how for example three lines could lead to three intersection points like in figure 2.11. This is actually what is bound to happen in a real situation, since the process of measuring and treating the data is not exact. Instead, one has to try to get as close as possible to the place where the unique intersection would have been with an exact process.

The better the resolution of the image that we endeavor, the more rays we need to send through the object. A reconstruction using a number of rays (M) larger than the number of components (N) in the reconstructed image, leads to an overdetermined system. It is preferable to have an overdetermined system, or at least a system where $M = N$. If $M < N$ we will get an infinite number of solutions. For our 2-D system, showed by figure 2.10, an underdetermined system would have only one or none line. This would mean that any point on the line, or any point at all respectively, would be a solution. Therefore the best situation is when we have more rays (equations) than components (unknowns) in our system.

Another thing worth mentioning is how the order of projection on the hyperplanes affects the speed at which the algorithm advances towards the center. If we in the example of figure 2.10 have two perpendicular lines, we only need two projections to get to the intersection point. It therefore seems favorable to arrange the equations in such a way that consecutive hyperplanes are perpendicular. Supposedly this would lead to a quicker

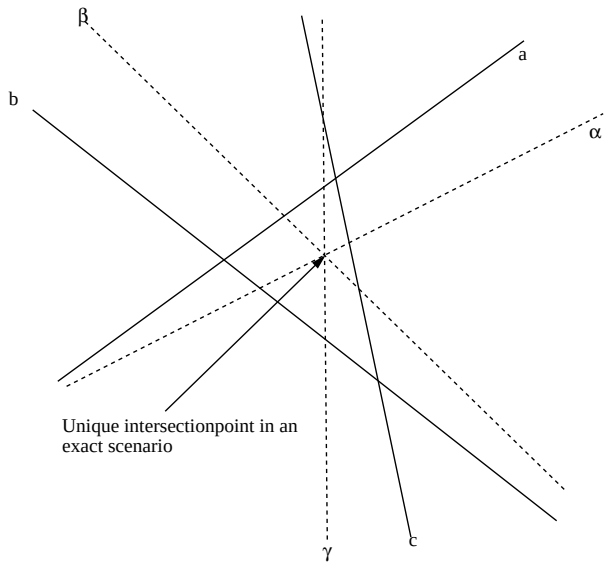


Figure 2.11: The lines a, b and c represents three equations containing errors. The lines α, β and γ symbolize the same equations for a system without any errors.

solution. The problem is that it also enhances the effect of the ever present noise in the real case because of an increased instability.

Finally, there is a factor that is added to all the algebraic algorithms. A factor, not mathematically necessary, but nevertheless very valuable in a real reconstruction. It is the so called relaxation factor, λ, which is a variable included in all the algebraic algorithms. When it is set to one it does not affect the reconstruction, but when it is set to a value less or greater than one it slows down or speeds up the process. The relaxation factor has to be chosen with respect to the current situation. A λ set too high can lead to an unstable algorithm, while a λ set too low might lead to a very slow reconstruction. In the illustrations of the algorithms in the following sections on the different algebraic algorithms λ is set to one. λ is always influencing the reconstruction when the reconstructed image is updated.

Iterating towards a solution can be seen as walking towards a goal. The length and the direction of the strides taken are given by the corrections. If you want to reach the goal faster, you take longer steps (increasing λ). By doing this, your speed certainly increases, but you are also running the risk of stepping over the goal (unstable algorithm). You can be careful and take small steps or you can walk fast trusting the correctness of your direction.

Some algorithms are a little bit too "self-confident" and have to be slowed down by using a $\lambda < 1$. Others are too careful and have to be sped up using a relaxation factor greater than one.

2.3.3 Algebraic Reconstruction Technique

The Algebraic Reconstruction Technique (ART) is the simplest of the algebraic algorithms. It projects the current solution onto the hyperplanes (equations) one after the other like figure 2.12 shows. A projection onto H_i can be described by

$$\overline{f}^{(k)} = \overline{f}^{(k-1)} - \lambda \frac{(\overline{f}^{(k-1)} \cdot \overline{a}_i - p_i)}{\overline{a}_i \cdot \overline{a}_i} \overline{a}_i. \tag{2.12}$$

When all the equations have been traversed, one says that the algorithm has performed one iteration. Normally several iterations are used in order to reach an acceptable solution.

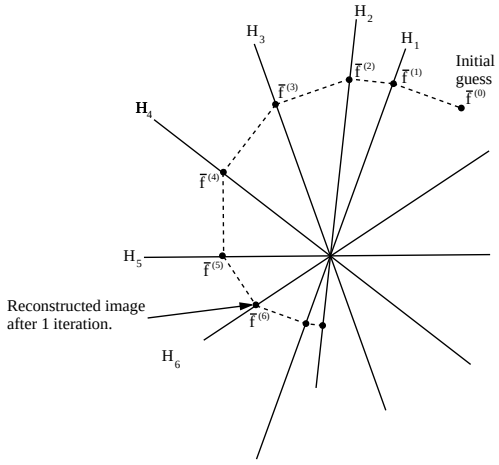


Figure 2.12: The progress of the ART algorithm through the equation system visualized in two dimensions. The black lines marked H_1 to H_6 symbolize equations/hyperplanes and every black dot is a new updated solution in the iteration process.

2.3.4 Simultaneous Iterative Reconstruction Technique

The Simultaneous Iterative Reconstruction Technique (SIRT) is an algorithm that takes a totally different approach to the iteration compared to

ART. Instead of updating the reconstructed image after each projection, it projects the current $\overline{f}^{(k)}$ onto all hyperplanes before updating. This is visualized in figure 2.13, in which one can see how the initial guess, $\overline{f}^{(0)}$ is first projected onto every hyperplane. After that the average of all the projected points is calculated. This calculated point will be the next updated reconstructed image, $\overline{f}^{(1)}$, in the iteration process. The projections on all of the hyperplanes and the following averaging is considered to be one iteration for SIRT. When the average is calculated the sum of the corrections is divided by M, the number of rays, as the following equation shows.

$$\overline{f}^{(k)} = \overline{f}^{(k-1)} - \frac{\lambda}{M} \sum_{i=1}^{M} \frac{(\overline{f}^{(k-1)} \cdot \overline{a}_i - p_i)}{\overline{a}_i \cdot \overline{a}_i} \overline{a}_i.$$

SIRT will need many more iterations than ART to reach a good reconstruction, since averaging makes the algorithm advance very slowly against the intersection point. On the other hand, thanks to this, it is a very stable algorithm.

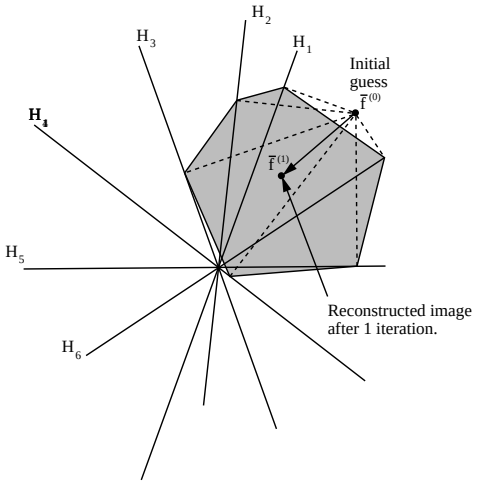


Figure 2.13: The progress of the SIRT algorithm through the equation system visualized in two dimensions. The black lines marked H_1 to H_6 symbolize equations/hyperplanes and every black dot is a new updated solution in the iteration process.

2.3.5 Simultaneous Algebraic Reconstruction Technique

Somewhere in between the ART and SIRT methodologies is the Simultaneous Algebraic Reconstruction Technique (SART). It tries to combine the

advantages of ART and SIRT. Both the speed of ART and the stability of SIRT. It does this by dividing the hyperplanes/equations into groups within which it uses the SIRT methodology. In between the groups, SART is making use of the ART methodology. It starts, as figure 2.14 shows, by projecting the initial guess onto all the members of the first group. The average of these projections is calculated, as in SIRT, and the answer is a new updated image, $\overline{f}^{(1)}$. After that, $\overline{f}^{(1)}$ is projected onto all the members of the second group, the average is calculated and a new updated image, $\overline{f}^{(2)}$, is obtained. When all the equations have been traversed, one iteration has been done. Mathematically SART can be expressed as

$$\overline{f}^{(k)} = \overline{f}^{(k-1)} - \frac{\lambda}{\sum_{i \in G_l}(1)} \sum_{i \in G_l} \frac{(\overline{f}^{(k-1)} \cdot \overline{a}_i - p_i)}{\overline{a}_i \cdot \overline{a}_i} \overline{a}_i.$$

where G_l is the set that contains all the members of group l. The sum of the corrections is divided by the number of elements in the group.

As one can understand this algorithm will depend on how one partitions the equations into groups. If all are put in one group it is equivalent to the SIRT algorithm, while, on the other hand, if every equation gets a group of its own, SART will be equal to ART. In other words, the behavior of SART using small groups will resemble that of ART and with larger groups that of SIRT. The trick is to find a size of group that is right for the purpose it shall be used for.

2.3.6 Component Averaging

In SIRT, which was explained earlier, the average of the corrections for each hyperplane is calculated. This is done by dividing the sum by the number of corrections, M, which obviously is the correct way to calculate an average. If one instead considers it from the viewpoint of a component, a possible weakness in this approach becomes visible. Each calculated correction can be said to consist of a set $C_i, i = 1 \ldots M$ containing N values, one for each component. Each set contains the correction information supplied by one ray. Since each ray is only passing through a small part of the image, most of the correction values are zero. The other non-zero corrections, let us call them "useful" corrections, are very few. In SIRT the sum of corrections is divided by the number of rays, M, which is a lot greater than the number of "useful" corrections for some component.

The idea of the Component Averaging(CAV) [8] is to ignore the correction values equal to zero, and when calculating the average, to divide by the

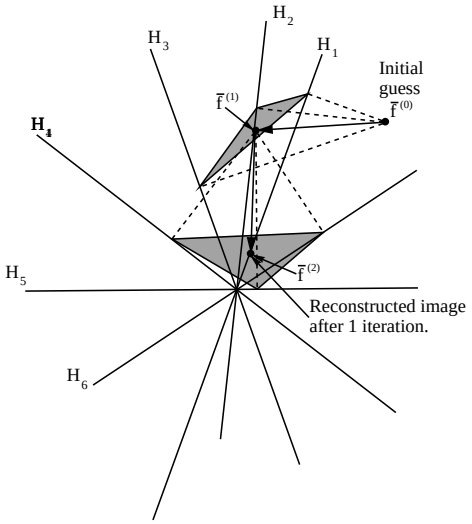


Figure 2.14: The progress of the SART algorithm through the equation system, visualized in two dimensions. The black lines marked H_1 to H_6 symbolize equations/hyperplanes and every black dot is a new updated solution in the iteration process. The hyperplanes are divided into the two groups $\{H_1, H_2, H_3\}$ and $\{H_4, H_5, H_6\}$.

number of "useful" corrections. This number is the number of rays actually passing through the component. In figure 2.16 the sum of correction values for f_j, would be divided by two instead of nine since only two of the nine rays pass through f_j. To show this mathematically the iteration algorithm has to be written for each component instead of for the whole reconstructed image like earlier. We begin by rewriting the formula for SIRT as:

$$f_j^{(k)} = f_j^{(k-1)} - \frac{\lambda}{M} \sum_{i=1}^{M} \frac{(\overline{f}^{(i-1)} \cdot \overline{a_i} - p_i)}{\overline{a}_i \overline{a}_i} a_{ij}. \tag{2.13}$$

In this version, the equation can be used to update each component function individually, which also gives us the opportunity to individually adjust it to each component. In the CAV case, this adjustment consists in dividing the sum of correction values only by the number of rays passing through the component. This number is the number of non zero elements in a column in the A matrix (see figure 2.15). The number of non zero elements in column

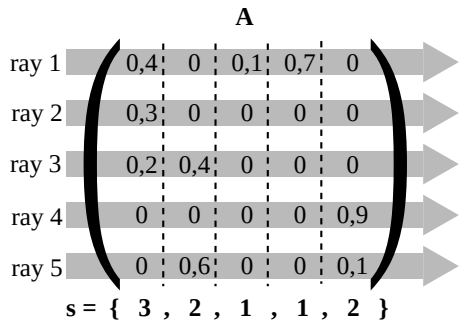


Figure 2.15: Relation between s and the A matrix in CAV.

j will be denoted by s_j. Hence, the equation will look like this:

$$f_j^{(k)} = f_j^{(k-1)} - \frac{\lambda}{s_j} \sum_{i=1}^{M} \frac{(\overline{f}^{(i-1)} \cdot \overline{a_i} - p_i)}{\overline{a}_i \overline{a}_i} a_{ij}. \tag{2.14}$$

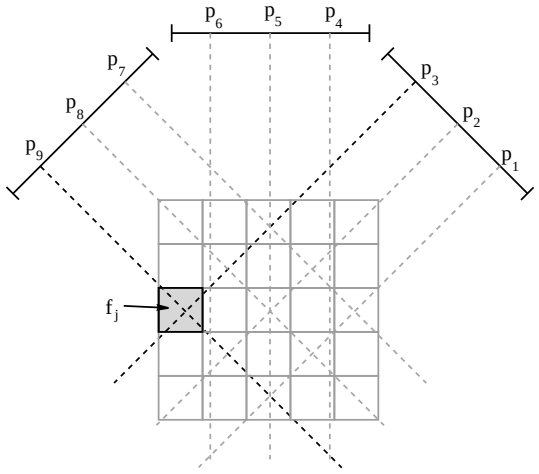


Figure 2.16: Only two of the nine rays in this projection are passing through basis function f_j. For simplicity pixels have been chosen as basis function.

2.3.7 Block Iterative Component AVeraging

Block Iterative Component Averaging (BICAV) is an algorithm that takes advantage of the ideas behind both SART and CAV. It partitions the equa-

tions into blocks. Within these blocks it performs the CAV algorithm (instead of SIRT like SART does), and in between the blocks it performs the ART algorithm (just like SART).

2.3.8 AVeraging of Sequential Projections

AVeraging of Sequential Projections (AVSP) (as it is called in this report) is a very new algorithm [9]. It takes on an approach that is the opposite of SART, regarding the order of iteration. The equations are partitioned into sets. Within the sets it works like ART, and in between the sets as SIRT (see figure 2.17).

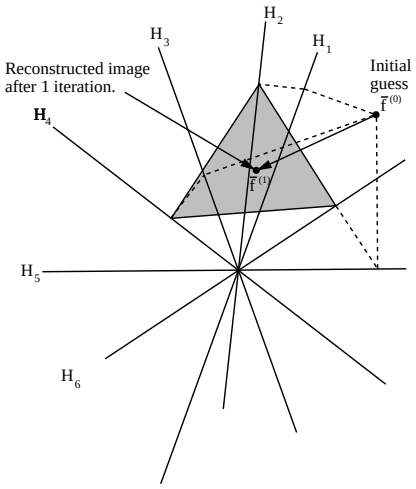


Figure 2.17: The progress of the AVSP algorithm through the equation system visualized in two dimensions. The black lines marked $H_1 to H_6$ symbolize equations/hyperplanes and every black dot is a new updated solution in the iteration process. The sets used in the figure are $\{H_1, H_2\}, \{H_3, H_4\}$ and $\{H_5, H_6\}$.

As in SART, we denote the sets by $G_l, l = 1 \dots T$. On each of these sets we perform one iteration of the ART algorithm which is denoted by $ART(G_l)$. This leads to T endpoints. The average of these T endpoints is calculated and used as the next updated solution. This can mathematically be described by

$$\overline{f}^{(k)} = \overline{f}^{(k-1)} - \frac{1}{T} \sum_{l=1}^{T} ART(G_l). \tag{2.15}$$

AVSP also uses a relaxation factor, even though it can not be seen in equation 2.15. It is embedded in the ART part of the equation.

Chapter 3

The Implementation

Not only the algorithm used is of importance for the results, but also the way it is implemented. Several stages in the reconstruction process are possible to implement in different ways. This chapter describes how important parts of the algorithms, explained in the previous chapter, have been implemented. The implementation was done in the form of a java program, which is briefly described in appendix D.

3.1 Ways to implement the A matrix

The A matrix is, as mentioned before, a description of how each ray passes through each component. Considering that a normal image might have a size of 100×100 components, and at least the same number of rays are used to reconstruct it, we end up with up with an A matrix containing about 10^8 elements. Hence, normally the A matrix is too big to be stored. This means that the A matrix can not be precalculated. Instead, the a_{ij} values will have to be calculated during the reconstruction process. As a consequence, it is important that they can be calculated quickly. This will be further explored in the following.

The a_{ij} values in the A matrix describe as was just mentioned, the connection between a ray and a component. There are different ways of describing this connection. The most straight forward and maybe also the most correct way of doing it is to look at the ratio of the component covered by the ray. For a pixel, a_{ij} would be calculated as figure 3.1 shows.

An other way of calculating a_{ij} is to set $a_{ij} = 1$ if the center of the j:th component is within the i:th ray, and zero otherwise (using this method on the situation in figure 3.1 would give $a_{ij} = 0$). Since the method in

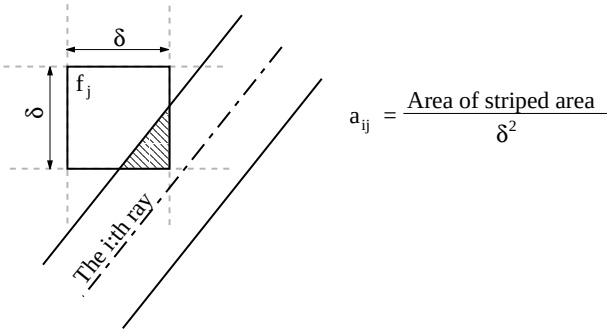


Figure 3.1: One way of calculating a_{ij}.

figure 3.1 gave very complex calculations for pixels, this was the method used to implement algorithms using pixels as their basis function. For blob algorithms the ratio of the blob covered by the ray is used to describe the a_{ij} values. This will be more precisely described in the following section about the implementation of the blob.

3.1.1 Implementation of the blob

Despite complicated at a first glance, the blob actually made the implementation easier. This is because it is a radial function and hence its value only depends on the distance from the center. The values in the A matrix will therefore only depend on how close the ray is passing the blob, whereas in the pixel case the angle of the ray has to be taken into account as well.

To simplify the calculation of the a-values, a look-up-table is constructed for the blob. For a number of distances from the center, this table will contain the integral of the area covered by the ray. Figure 3.2 shows how the look-up table for the blob is calculated. First line integrals (projections) of the blob, represented by a 3-D graph in 3.2(a), are calculated. This provides us with the so called footprint of the blob (figure 3.2(b)). In the program, the values of the actual blob never exist. Instead, the footprint is calculated straight away, using a formula describing the line integral through the Kaiser-Bessel radial function (the blob function, equation 2.4) at a distance s from the center. One could say that the projection of the blob is sampled, where each calculated line integral is one sample. In the program a sample rate of 50 samples per unit distance is normally used. For a blob of radius two, the footprint will then consist of: $2 \cdot 2 \cdot 50 + 1$ samples. The footprint is "normalized" so that the sum of all line integrals is equal to one. Now what

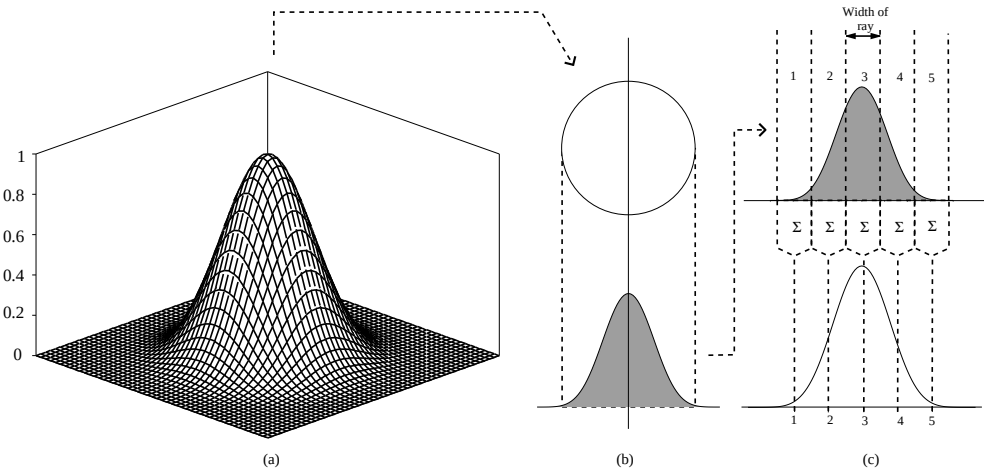


Figure 3.2: The process of obtaining the blob look up table. The blob (a), shown from above in (b), is projected in (b) to get the footprint, which is then sampled and integrated over each possible ray position (c).

we want is the sum of all the values of the footprint through which the ray is passing. For example: if the center of the ray is 1 unit distance away from the center of the blob, we would like to sum all the footprint values between 0.5 and 1.5. This sum for all possible distances, center of ray to center of blob, will be contained in the final look up table. Figure 3.2(c) shows how the sums for five ray positions are calculated and put into the look up table. Thanks to this method we can very fast access the ratio of the blob covered by the ray during the calculations, which is the a-value.

3.1.2 Implementation of the projections

In the world of tomography there are two types of projections: forward, and backprojection. The backprojection is used by all the algorithms implemented in this project while forwardprojection is only used by the algebraic ones. How they are used in the algorithms was discussed in section 2.2 and 2.3. Here their implementation in the program will be explained.

Forwardprojection, besides being a part of the algebraic algorithms, is used to create a sinogram from a phantom. It tries to simulate the way the rays are affected when they are passing through a real object, and caught by the detectors. To do this, the way the ray progresses through the object has to be known. Knowing this is the same as knowing the relation between the

ray and each component. Two different approaches to acquire this knowledge are taken depending on whether pixels or blobs are used as basis elements.

In the pixel case a ray tracing algorithm (Bresenham's line drawing algorithm) is used to calculate the path a ray is taking. The values of the pixels, found to be crossed by the ray according to the ray tracing algorithm, are summed up. The sum is the projected value. No attention is paid to how much the pixels are covered by the ray since a_{ij} is set to 1 when the center of the pixel is within the ray. This is considered to occur when the pixel is included by the ray tracing algorithm. Otherwise a_{ij} is set to 0.

The procedure of implementing the forwardprojection in the blob case is very different from the pixel case. In the blob case a more sophisticated method of calculating a_{ij} is used. As described earlier, we have a look-up table for the blob. If we supply this table with the distance from the center of the blob to the center of the ray, it provides us with the ratio of the blob covered by the ray. Then all we need to know to obtain a_{ij}, is the orthogonal distance from the center of blob j to the center of ray i. This distance for all N components gives us a value of the relation between ray i and each component. The projection caused by ray i is then given by

$$p_i = a_{i1}f_1 + a_{i2}f_2 + \ldots + a_{iN}f_N. \tag{3.1}$$

Backprojection is used by all the algorithms, although in different ways. The difference concerns *what* is backprojected. In the filtered backprojection algorithm it is the projection, weighted in the frequency plane, that is backprojected, while in the algebraic algorithms it is the correction values. These correction values are used to update the reconstructed images mentioned in section 2.3. Every time that the algorithm demands an update, it means that a backprojection has to be performed. The backprojection can be seen as smearing a value back along the ray that originally generated it. Essentially it is implemented the same way as the forwardprojection. The only difference is that, when the forwardprojection is collecting weighted values from the components covered by the ray, the backprojection is adding a value to all the those components. In this addition the value will be weighted by the corresponding a_{ij} value.

3.1.3 Correction values in the algebraic algorithms

As mentioned earlier, the values that are backprojected in the algebraic algorithms are correction values. Correction values, someone might ask. What are they correcting? The answer is that they are correcting, or trying to correct, the difference between the measured and the calculated sinogram

(see section 2.3.2). It is this difference, modified according to the reconstruction algorithms, that is backprojected. In the SIRT algorithm for example, each correction value will be divided by the number of rays. When the algorithm, in its forwardprojection has produced a calculated sinogram equal to the measured one, there is nothing more for this algorithm to correct. It has finished since it can not possibly obtain any better reconstruction.

There is one exception in the implementation from the formulas described in section 2.3. This concerns the pixel case. It was said earlier in this report that 0:s and 1:s were to be used as A-values when using pixels. This would lead to a correction value looking as follows for ART:

$$\Delta f_j^{(i)} = \frac{p_i - q_i}{N_i}. \tag{3.2}$$

$\Delta f_j^{(i)}$ is here the correction for the j:th component, from the i:th equation of 2.9 and N_i is the number of components passed by ray i. p_i is the measured and q_i is the calculated projection value. This, despite easy to implement, is according to [1] a method prone to give artifacts (errors). In [1] A.C Kak and M. Slaney instead recommend to use

$$\Delta f_j^{(i)} = \frac{p_i}{L_i} - \frac{q_i}{N_i} \tag{3.3}$$

where L_i is the length of ray i that is passing through the object. This recommendation has been followed and the method was used to calculate the correction value for algorithms using pixels as their basis function.

Chapter 4

Comparison of the reconstruction techniques

This chapter explains how the different algorithms have been compared. It tries to highlight the most important differences between the different reconstruction methods. It is meant to serve as a background to the next chapter, where the results of the tests are presented.

4.1 Comparison of basis functions

At a first glance, the obvious way to store an image seems to be using pixels as building blocks. Pixels are all we can see when we display the image on a computer screen anyway. It is easy to store the image since all we need is a value for each pixel. But when it comes to see the image as a representation of a real object and to perform operations used in reconstructions on it, pixels appear to have disadvantages.

The first thing one should realize when reconstructing, is that we are trying to obtain an image, which shall respond to projections in the same way as the real object. This is what all the algebraic algorithms are aiming for. Many different types of basis functions fulfil this criterion, without necessarily creating an image that looks like the object. Therefore, the actual question will be: which basis function is best describing a component, so that it resembles the building blocks of the object as closely as possible. Hence it is valuable to consider what the building blocks of the object actually look like. It stands clear that the human body for example, does not consist of small cubes put together like bricks in a wall. The components of a human body are better described as spheres with an uneven distribution. The blob

is an attempt to create a basis function that better resembles the building block of an organism.

The immediate disadvantage of blobs seems to be the increased complexity of its basis function (compare equation 2.2 and 2.4). This is not such an important disadvantage if a look-up table, as described in section 3.1.1, is used. It is the spherical shape of the blob that makes the use of a look-up table easy to implement.

Unlike pixels, blobs are normally overlapping each other. This makes the transition from one component to another softer, which gives a smoother image. With blobs, the so called "salt and pepper" noise, which is common using pixels, is mitigated. This may not lead to a better reconstruction mathematically, but visually the difference is significant. You might loose the distinct edges but trends are seen a lot clearer in an image reconstructed using blobs as basis function.

When comparing the reconstructions using different basis functions the same sinogram has to be used for all of them, since this would be the case in a real world scenario. The question is how to create the sinogram. The sinogram, as explained earlier, is created by forwardprojecting the phantom from different angles. The result of this process will depend on which basis function that is assumed to form the components of the phantom. In this report a standard blob will be used as basis function in the creation of the sinogram. The reason underlying this is that blobs are "closer" to the reality than pixels. Therefore a sinogram that better resembles a real one, will be obtained using blobs in the forwardprojection of the phantom.

In order to compare the different basis functions two versions of the ART algorithm were implemented. One version using pixels and the other using blobs as its basis function. The three different blobs used were the ones described in section 2.1.3, figure 2.6. Out of these four basis function one of the blobs was chosen to be used in the following comparison of the algorithms. The pixel was discarded from the beginning because so much research has already been done on that. Here it will just be used as a reference, showing that the blob truly is an improvement to the classical pixel.

4.2 Analytical or algebraic algorithms?

As described in chapter 2, there is a significant difference between algebraic and analytical algorithms. The advantage of the analytical algorithms is that they are fast, and do not require so much computer power. In the

tests performed in this project the analytical algorithm was superior to the algebraic ones in terms of speed. This is the reason why they are the most commonly used algorithms in hospitals today. Algebraic algorithms require a greater amount of calculations, which has made them unsuitable for practical use. But the times are changing, and the continual increase of computer capacity is now paving the way for these more demanding algorithms, thus making the arguments against them weaker. A significant advantage of the algebraic algorithms is their generality. All they demand are some equations describing the desired object. These equations are often formed by rays, which can be sent through the object from any angle. They do not, as in the case of filtered backprojection, require that the rays create a projection that can be Fourier transformed.

4.3 Comparison of the algorithms

What has been described so far in this and earlier chapters, touches most of the ways in which the different reconstruction algorithms differ. This section describes the method used to determine of what importance these differences are. As a measure of quality of the reconstructions, a few so called "Figures of Merit" are used. These will be described in the following and will hopefully reveal how much the differences mean, which ones are of importance and which ones are not so important.

First the preferred basis function was determined as described in section 4.1. Once having decided that, this basis function was used in all of the different algorithms.

Before comparing the different algorithms they were individually *trained*. *Train* in this context means to optimize their parameter settings in order to give the best correlation to the phantom possible. The parameters are mainly the relaxation factor (λ) but also in some algorithms(SART, BICAV and AVSP) the number of blocks. The parameters have been optimized for reconstruction of the Shepp-Logan Phantom. This means that these parameter settings are not necessarily the best ones for another type of phantom. The parameter settings for the individual algorithms used as a starting point are the ones shown in table 4.3. Starting from these values, commonly accepted as guidelines, the settings were optimized for this purpose. For the SART, BICAV and AVSP algorithms the recommended value of the relaxation factor depends on the number of blocks used. Naturally it varies between 0.2 and 2.0 since at their extremes they are all equal to SIRT/CAV at one side and ART at the other.

Algorithm	Relaxation factor
Filtered Backprojection	N/A
ART	< 0.2
SIRT	< 2.0
SART	$0.2 < \lambda < 2.0$
CAV	< 2.0
BICAV	$< 0.2 < \lambda < 2.0$
AVSP	$< 0.2 < \lambda < 2.0$

Table 4.1:

After the appropriate parameters had been selected for the different algorithms, all of the algorithms were compared. Since it is known that some algorithms are better suited for some task and some for some others, there was no attempt made to find an algorithm that was the best. Instead, for each of the four different numbers, 1, 20, 50 and 100, of iterations, one "winner" was selected. Through this procedure the most suitable algorithm, depending on the demands for speed and the computer power supplied, was chosen.

It is of course true that one iteration might take different amount of processor power for different algorithms. Still this method will give good estimation of the algorithms suitability, if at all suitable, in different situations.

4.4 Figures Of Merit (FOM)

To evaluate the quality of the reconstructions three different Figures Of Merit (FOM) are used. These figures describe how well the reconstructed image resembles the phantom. The FOM:s chosen to be used in this report are *the correlation*, *the distance* and *the relative error*. What these stand for will be explained in this section.

To start with, some variables used in the calculations will be defined. Since both the phantom and the reconstructed image are discretized images we will be talking about a limited set of points. The values at these points will be notated by $f_j^{(k)}$ for the reconstructed image after k iterations and $\widetilde{f}_j$ for the phantom. As usual $j = 1 \ldots N$ where N is the number of components.

The *average* value of the reconstructed image, $f^{(k)}$, is denoted by

$$\rho_k = \frac{1}{N}\sum_{j=1}^{N} f_j^{(k)} \tag{4.1}$$

and the *variance* by

$$v_k = \frac{1}{N}\sum_{j=1}^{N}(f_j^{(k)} - \rho_k)^2. \tag{4.2}$$

The *standard deviation* is then given by

$$\sigma_k = \sqrt{v_k}. \tag{4.3}$$

For the phantom the average value, $\widetilde{\rho}$, the variance, $\widetilde{v}$, and the standard deviation, $\widetilde{\sigma}$, will be defined similarly using the phantom values $\widetilde{f}$.

The correlation between $f^{(k)}$ and the phantom, $\widetilde{f}_j$ is

$$r(f_j^{(k)}, \widetilde{f}_j) = \frac{\frac{1}{N}\sum_{j=1}^{N}((f_j^{(k)} - \rho_k)(\widetilde{f}_j - \widetilde{\rho}))}{\sigma_k \cdot \widetilde{\sigma}} \tag{4.4}$$

which is the first FOM. The second FOM, the distance between $f^{(k)}$ and $\widetilde{f}$ is defined as

$$\delta_k = \begin{cases} (1/\widetilde{\sigma})\sqrt{(1/N)\sum_{j=1}^{N}(f_j^{(k)} - \widetilde{f}_j)^2} & \text{if } \widetilde{\sigma} > 0, \\ \sqrt{\sum_{j=1}^{N}(f_j^{(k)} - \widetilde{f}_j)^2} & \text{if } \widetilde{\sigma} \leq 0. \end{cases} \tag{4.5}$$

With $\tau = \sum_{j=1}^{N}|\widetilde{f}_j|$ the relative error of $f^{(k)}$ is defined by

$$\epsilon_k = \begin{cases} (1/\tau)\sum_{j=1}^{N}|f_j^{(k)} - \widetilde{f}_j| & \text{if } \tau > 0, \\ \sum_{j=1}^{N}|f_j^{(k)} - \widetilde{f}_j| & \text{if } \tau \leq 0. \end{cases} \tag{4.6}$$

As can be seen in formula 4.5 the distance is weighted by $\frac{\sqrt{1/N}}{\widetilde{\sigma}}$ (at least when the standard deviation is greater than zero). This is done to make *distance* a measure relative to the standard deviation. A higher value of the standard deviation means that the image has a wider distribution of values. An image with a wider distribution does not have to be reconstructed as exact as one with a narrower distribution. If for example, a component in

the phantom with a value $\widetilde{\rho}+100$ is reconstructed to $\widetilde{\rho}+98$ it is significantly better than a value $\widetilde{\rho}+5$ reconstructed to $\widetilde{\rho}+3$, although the distance is the same. With the weighting of the distance, this FOM can be used to compare reconstructions from different phantoms.

The same type of reasoning is the idea behind the term $\frac{1}{\tau}$ in the calculation of the relative error. It is an error that is relative to the sum of the component values.

The reason to use three different measures to estimate the quality of the reconstruction is that there is no unique measure of quality. What quality is, depends on what the image is meant to be used for. In addition to the three FOM:s, the reconstructions have also been assessed visually. Focus has been put on the amount of details possible to discern and their clearness.

Chapter 5

Results

This chapter contains all the results from the tests of the different algorithms. It tries to explain why the results look like they do and to point out the interesting ones.

5.1 Comparison of basis functions

The results presented here were obtained using the method described in section 4.1 with the ART algorithm was set to perform 20 iterations with each basis function. The results were then compared both visually and mathematically. For all of the reconstructions the relaxation factor was set to 0.20. The choice of such a small value depends on the instability of the ART algorithm.

Figure 5.1 shows the reconstructed images produced after 20 iterations for each of the four basis functions (see figure 2.6 for data about the different blobs), except the one reconstructed using pixels. That one shows the image after only 8 iterations, since the reconstruction deteriorates with further iterations. This probably depends on inconsistencies introduced in the system with the use of the approximated a-values, but also on the unstableness of the ART algorithm.

It is obvious, looking at figure 5.1, that details can be discerned much easier when looking at the "blob reconstructions", than at the image reconstructed by pixels. The smoothness of 5.1 (b), (c) and (d), due to the blob, stand in bright contrast to the noisiness of reconstruction 5.1 (a). Visually, the blob seems to be superior to the pixel. To also mathematically compare the impact of the basis function, the correlations between the reconstructions and the phantom were calculated (as described in section 4.4). Figure

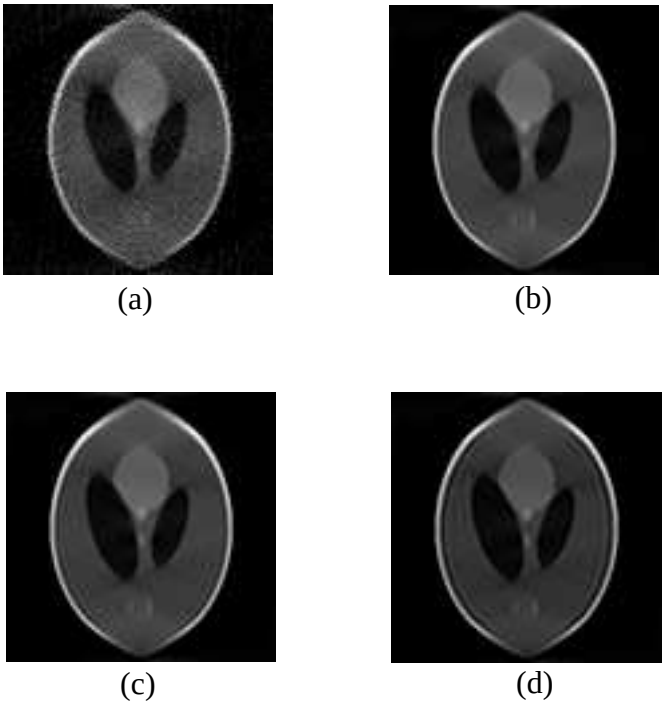


Figure 5.1: Four ART reconstructions of the Shepp-Logan phantom using pixels (a), blob A (b), blob B (c) and blob C (d) as their basis function.

5.2 shows how this correlation evolves for the ART algorithm using the four different basis functions. It is also here clear how the pixel distinguishes itself by giving results inferior to the ones obtained using blobs. Not only does it give a correlation significantly smaller than the one for blobs, but it also makes the reconstruction unstable. After 8 iterations the correlation starts to deteriorate without any hope for recovery in sight. To see if maybe a change would come later on, 200 iterations of the ART algorithm were performed, but no turning point was ever reached.

All this was convincing enough to chose a blob as basis function for the further comparisons of algorithms. Now comes the question of which one of the three blobs to choose. Visually it is hard to chose one before the other since they are all very similar and show the same amount of details. Judging from the correlation to the phantom shown in figure 5.2 it is a close call between blob A and blob B. To get some more information in order to chose one of them, the distance and the relative error to the phantom for the three blobs were calculated (see figure 5.3). It is the same scenario

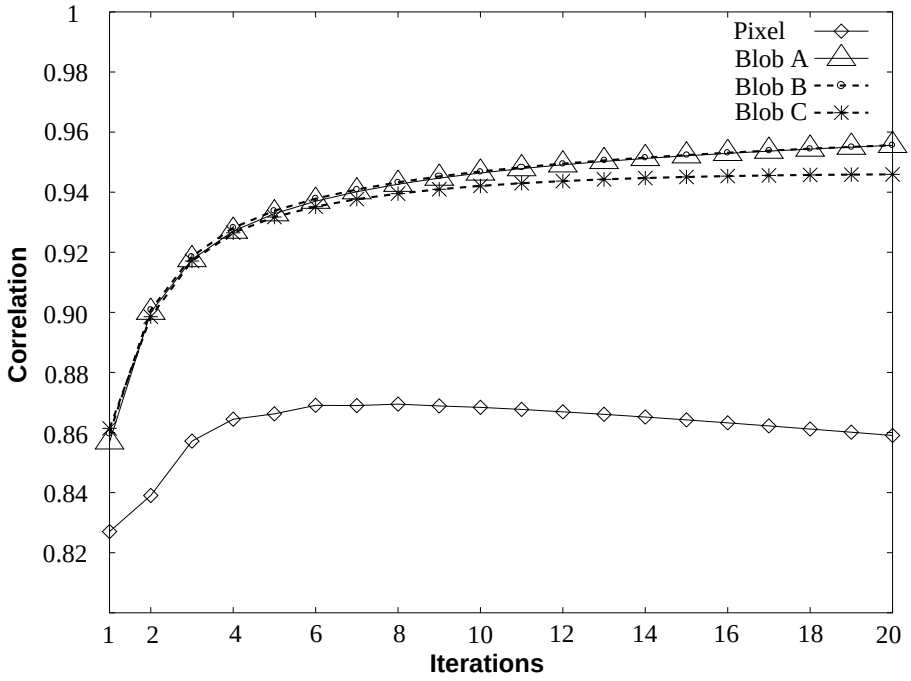


Figure 5.2: Correlation between the phantom and the reconstructed image obtained using ART, with four different basis functions.

for these FOM:s as for the correlation. In other words an extremely small difference between Blob A and B. Still, though the difference is small, it exists and on basis of this blob B was chosen to be used as basis function in the continuing. It proved to give the best correlation, and the lowest relative error and distance.

5.2 Settings of the algorithms

In this part the values of the relaxation factor and the number of blocks that were found to be the best for the different algorithms will be presented.

5.2.1 ART

The only parameter that has to be set for ART is the relaxation factor, λ. To decide the most approprite value of λ for the different numbers of iterations the algorithm was run with several different settings. The results are presented in table 5.1

Figure 5.3: How the relative error and distance to the phantom diminishes with the number of iterations for the ART algorithm, using three different blobs as basis functions.

	Relaxation factor (λ)				
# of Iterations	0.2	0.4	0.6	0.8	1.0
1	0.8597	0.8810	0.8880	**0.8897**	0.8887
20	0.9556	0.9591	0.9599	**0.9601**	0.9599
50	0.9642	0.9659	**0.9662**	0.9661	0.9659
100	0.9688	**0.9698**	0.9698	0.9696	0.9693

Table 5.1: Correlation between phantom and reconstructed image using the ART algorithm.

Table 5.1 shows clearly how the best suitable value of λ, decreases with a higher number of iterations. Based on these results the setting will be: $\lambda = 0.8$ for 1 and 20 iterations, $\lambda = 0.6$ for 50 iterations, and $\lambda = 0.4$ for 100 iterations.

5.2.2 SIRT

In table 5.2 some correlation values from the SIRT algorithm can be viewed. The most striking thing about it might be the very high relaxation factor needed to optimize the algorithm. It has to do with the fact that SIRT is dividing the sum of corrections by the number of rays when calculating the average of the forwardprojections (see section 2.3.3). Still each component is passed by far from all of the rays (the fact that CAV is built on). That is why the correction value added to each component will be very low and a high relaxation factor is needed to compensate. The optimal relaxation factor, in

this project found to be 90, is strongly related to the number of rays in each projection, in this case 101. This is why, in many implementations of the SIRT algorithm, the number of projections, instead of the number of rays (see equation 2.13), is used to calculate the average forwardprojection.

	Relaxation factor (λ)					
# of Iterations	1	10	50	80	90	100
1	0.6882	0.6882	0.6882	0.6882	0.6882	0.6882
20	0.6995	0.7663	0.8598	0.8811	**0.8867**	0.4717
50	0.7145	0.8238	0.8903	0.8981	**0.8999**	0.4658
100	0.7341	0.8591	0.8997	0.9021	**0.9024**	0.4517

Table 5.2: Correlation between phantom and reconstructed image using the SIRT algorithm.

It shall also be noted that, when the SIRT algorithm has been run only one iteration, the relaxation factor has no impact on the correlation with the phantom. This is because the reconstructed image is only updated once per iteration, and it is only when this happens that the relaxation factor is used. The only thing that affects the correlation is the pattern, not its magnitude. However, the distance and the relative error are affected.

For all numbers of iterations $\lambda = 90$ will be the setting of the relaxation factor.

5.2.3 CAV

As described in section 2.3.6, the CAV algorithm is the same as SIRT but with an additional weighting, added when backprojecting the correction values. Instead of dividing by the number of rays, the correction value is divided by the number of rays that actually did pass through the component (the CAV technique). As can be observed in table 5.3 this leads to a best correlation to the phantom, at a much lower relaxation factor. Also in CAV the correlation is independent of the relaxation factor after one iteration. The reason for this is the same as for the SIRT algorithm. For any number of iterations, $\lambda = 5$ turned out to be the best value of the relaxation factor. This value will be used for the CAV algorithm in the following comparisons with the other algorithms.

5.2.4 SART

Table 5.4 shows the correlation between the phantom and the reconstructed image after 100 iterations using the SART algorithm. It was run with dif-

	Relaxation factor (λ)						
# of Iterations	1	2	3	4	5	6	7
1	0.6911	0.6911	0.6911	0.6911	0.6911	0.6911	0.6911
20	0.7965	0.8379	0.8580	0.8705	**0.8810**	0.8053	0
50	0.8488	0.8779	0.8894	0.8950	**0.8981**	0.8545	0
100	0.8777	0.8948	0.8994	0.9012	**0.9021**	0.8744	0

Table 5.3: Correlation between phantom and reconstructed image using the CAV algorithm.

ferent values of the relaxation factor, λ, and the number of blocks.

	Relaxation factor (λ)								
# of Blocks	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10	20
9	-	-	0.8992	0.9006	0.9015	0.9021	0.9025	0.9028	**0.9035**
11	-	-	0.9030	0.9040	0.9046	0.9050	0.9053	***0.9055***	0.9054
13	0.8999	0.9024	0.9037	0.9044	0.9049	0.9051	0.9053	**0.9054**	0.9049
16	0.9016	0.9034	0.9042	0.9047	0.9049	0.9050	**0.9051**	0.9051	0.9040
21	0.9029	0.9039	0.9044	0.9046	**0.9047**	0.9046	0.9045	0.9044	0.9027
31	0.9035	0.9039	**0.9040**	0.9039	0.9037	0.9035	0.9033	0.9030	0.9006
61	**0.9025**	0.9021	0.9017	0.9013	0.9008	0.9004	0.8999	0.8995	0.8965

Table 5.4: Correlation to the phantom after 100 iterations using SART. The emphasized *values are the best ones in each row.*

It is easy to observe how the relaxation factor that gives the best reconstruction (see the emphasized figures in table 5.4), changes with the number of blocks used. With 61 blocks (the version closest to ART) the best correlation value is reached at $\lambda = 3$ while with less blocks it is obtained using a higher value of λ. For the SART algorithm to be equal to the ART, the number of blocks would be 6161 (one block for each equation) in this case. What table 5.4 first and foremost shows is that a lower value of λ is preferable when the SART algorithm is approaching ART, and a higher value is preferable when it is approaching SIRT (one block). It clearly shows how the algorithm grows less and less sensitive to a higher relaxation factor with a smaller number of blocks.

When it comes to the relation between relaxation factor and the number of iterations it turned out that, the fewer iterations one is using, the better it is to use a higher relaxation factor. Using 16 blocks for example, the best correlation after 20 iteration is achieved with $\lambda = 30$ (0.9030), while after 100 iterations $\lambda = 9$ gives the best correlation (0.9051). Contemplating all this, it is not possible to say that one combination of relaxation factor and number of blocks is the best one. It depends on the situation, especially on

the number of iterations performed.

For the four different numbers of iterations, four different combinations of relaxation factor and number of blocks will be used. These values have been chosen on basis of the correlation between the phantom and the reconstructed image. Almost all the results of the tests can be found in appendix A.1. In accordance with these results, the settings that will be used are: $\lambda = 80$ and number of blocks = 31 for 1 iteration, $\lambda = 50$ and number of blocks = 11 for 20 iterations, $\lambda = 50$ and number of blocks = 11 for 50 iterations and $\lambda = 10$ and number of blocks = 11 for 100 iterations.

5.2.5 BiCAV

The same relation as between SIRT and CAV exists between SART and BiCAV. In other words, BiCAV is the SART using the CAV technique. The results for the BiCAV can be viewed in appendix A.2. They showed that BiCAV behaved very much like SART except that a lower value of relaxation factor was needed to optimize the algorithm.

In accordance with the results in appendix A.2, the settings that will be used are: $\lambda = 5$ and number of blocks = 31 for 1 iteration, $\lambda = 3$ and number of blocks = 11 for 20 iterations, $\lambda = 2$ and number of blocks = 11 for 50 iterations and $\lambda = 1$ and number of blocks = 11 for 100 iterations.

5.2.6 AVSP

The third algorithm using blocks is the AVSP. Since the order it calculates the equations is different compared to SART and BiCAV, its behavior will not be assumed to be the same as theirs. Several approaches can be taken to partition the equations into the sets/blocks. One way of doing it is to distribute the equations *randomly* to the different blocks. Another way is to put equations, representing rays from two projections as far away as possible from each other, in the same block. The latter is done with the intention to increase the speed of the iteration, assuming that rays with very different angles can be represented by perpendicular lines (see section 2.3.2). The two techniques were compared for some different values of λ and for the random-technique also the number of blocks. The results of these comparisons, after 100 iterations, are presented in table 5.5.

AVSP has at its extremes ART and SIRT, just like SART. The difference is that they are positioned in the opposite way. Using one block containing all the equations, AVSP is equal to ART and with one block for each equation it is equal to SIRT. This characteristic shows in table 5.5 through the higher

Partitioning method	# of Blocks	Relaxation factor (λ) 0.2	0.6	1.0	2.0	3.0	4.0
Greatest distance	31	0.9305	0.9494	0.9540	0.9494	0	-
Random	2	0.9663	0.9704	***0.9709***	0.9663	0	-
	4	0.9616	0.9679	**0.9695**	0.9687	0	-
	6	0.9576	0.9663	0.9679	**0.9685**	0.4510	-
	8	0.9545	0.9645	0.9666	**0.9675**	0.9561	-
	10	0.9516	0.9630	0.9659	**0.9678**	0.6346	-
	31	0.9312	0.9509	0.9568	0.9631	**0.9637**	0.9605
	50	0.9171	0.9437	0.9514	0.9589	**0.9621**	0.9609
	100	0.8874	0.9293	0.9404	0.9504	0.9558	**0.9567**
	200	0.8485	0.9062	0.9241	0.9402	0.9471	**0.9503**

Table 5.5: Correlation for 100 iterations with the AVSP algorithm.

susceptibility for a higher relaxation factor with more blocks. Choosing the "random-method" of partitioning, 2 blocks and $\lambda = 3$ for example, a total breakdown of the algorithm will occur. Using more blocks, the algorithm is stable for this relaxation factor. The same pattern can be seen for SART and BiCAV. In one aspect AVSP seems to differ significantly from SART and BiCAV; the best relaxation factor appears to change very little, no matter how many iterations were performed (see appendix A.3.

Table 5.5 also shows how the method of partitioning influences the result, both the correlation and the stability. When forming the blocks in the "greatest distance" method, all the equations belonging to two different projections are put into one block. The sum of the distances (in degrees) in between all pairs of projections is maximized. Since the projection is done over 120°, the distance between the projections in each pair is 60°. Putting equations belonging to rays positioned next to each other in the same block will increase the speed of the algorithm. This has to do with the influence that the result of the previous equation has when projecting on the current equation. This influence is greater when the equations belong to two consecutive rays in the same projection. On the same token the higher influence between equations consecutively calculated also makes the algorithm less stable, which is reflected in its higher sensitivity to a higher value of λ.

For all numbers of iterations the random partitioning will be used. λ will be set to 0.6 for 1 iteration and to 1 for 20, 50 and 100 iterations. The number of blocks will be 2 for all numbers of iterations. These settings turned out to give the best correlation. All of the results on which these choices are based, can be seen in appendix A.3. It is interesting to see that the correlation is

constantly improving with a smaller number of blocks. In other words, a higher correlation is achieved for AVSP when its structure is approaching that of the ART algorithm. The random version of the AVSP algorithm was also run with only one block, which is equal to the ART algorithm. This actually gave the best correlation (0.9717), but then it is no longer the AVSP algorithm, but an ART algorithm projecting on the equations in a random order. This might be of interest for further investigations.

5.3 Comparison of the algorithms

Now when the candidates from the different algorithms are chosen we have reached the final comparison. The parameter settings for the trained algorithms for 1, 20, 50 and 100 iterations are presented in table 5.6.

	1 Iteration		20 Iterations		50 Iterations		100 Iterations	
Algorithm	λ	Blocks	λ	Blocks	λ	Blocks	λ	Blocks
Filtered Backproj.	-	-	-	-	-	-	-	-
ART	0.8	-	0.8	-	0.6	-	0.4	-
SIRT	-	-	90	-	90	-	90	-
CAV	-	-	5	-	5	-	5	-
SART	80	31	50	11	50	11	10	11
BiCAV	5	31	3	11	2	11	1	11
AVSP	0.6	2	1	2	1	2	1	2

Table 5.6: Settings of the algorithms that will be compared, for 1, 20, 50 and 100 iterations.

Table 5.7 displays the correlation to the phantom, achieved by the different trained algorithms. Two algorithms, ART and AVSP, significantly distinguishes themselves after 100 iterations. They are both based on the ART technique and are significantly better than the other five algorithms. After 1 and 20 iterations ART is slightly better than AVSP, but the roles are reversed after 50 and 100 iterations.

The performance of the algorithms compared to one another can be observed in figure 5.4. The graph clearly shows how the algorithms can be divided into three groups, based on their achieved correlation. The ART-based algorithms in the top group, the SIRT-based algorithms in the middle group and the analytical Filtered backprojection algorithm in the bottom group. The differences within the two upper groups is very small. It could be noted that SART and BiCAV reach a good correlation significantly faster than SIRT and CAV in the middle group. This surely has to do with their

	# of Iterations			
	1	20	50	100
Filtered Backproj.	0.8624	-	-	-
ART	**0.8897**	**0.9601**	0.9662	0.9698
SIRT	0.6882	0.8867	0.8999	0.9024
CAV	0.6911	0.8810	0.8981	0.9021
SART	0.8534	0.9037	0.9051	0.9055
BiCAV	0.8531	0.9039	0.9052	0.9059
AVSP	0.8654	0.9598	**0.9670**	**0.9709**

Table 5.7: Correlation to the phantom using the best versions of the different algorithms.

closer resemblance to the ART algorithm, which is faster than SIRT. All the results just presented are optimized values for each number of iterations.

Now the development through the iterations for algorithms optimized for 100 iterations will be studied. The graphs in figure 5.5 show the development of the correlation, the distance and the relative error to the phantom. As can be seen in the graphs, all the FOM:s more or less speak the same language; ART and AVSP are in a class of their own. One thing that is interesting to note is the volatile progress of the SIRT and CAV algorithms. This is probably connected to the high relaxation factor (90 for SIRT and 5 for CAV) used for these reconstructions.

The relaxation factor's impact on the final result is the last issue dealt with in this section. Is it possible to see a difference between reconstructions achieving very similar values of the correlation, where different values of λ were used? In order to see this, four images reconstructed with 100 iterations by the AVSP algorithm were chosen. They all gave approximately the same correlation (0.9609-0.9631), but they were reconstructed using λ =0.2, 2, 3 and 4 respectively. The reason for doing this is to see how well the correlation actually describes the quality of the reconstruction. Figure 5.6 shows the four images obtained. The difference between the images is remarkable. The higher the value of the relaxation factor, the more course-grained the image seems to be. Though apparently not lowering the correlation, it definitely makes it harder to visually discern small patterns. The middle of the three ellipses in the lower part of the phantom for example, is more likely to be spotted in figure 5.6 (a) than in (d). Since 5.6 (d) is so patchy, the risk of missing important details is increased.

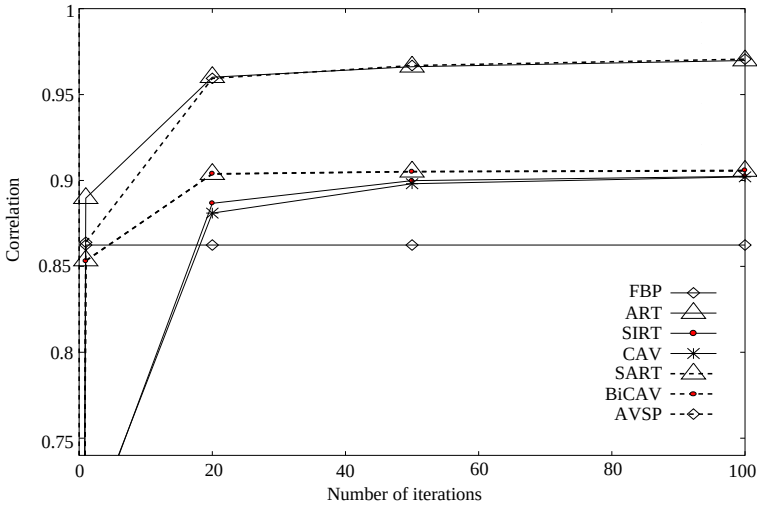

1
0.95
0.9
0.85
0.8
0.75
Correlation
0
20
40
60
80
100
Number of iterations
FBP
ART
SIRT
CAV
SART
BiCAV
AVSP

Figure 5.4:

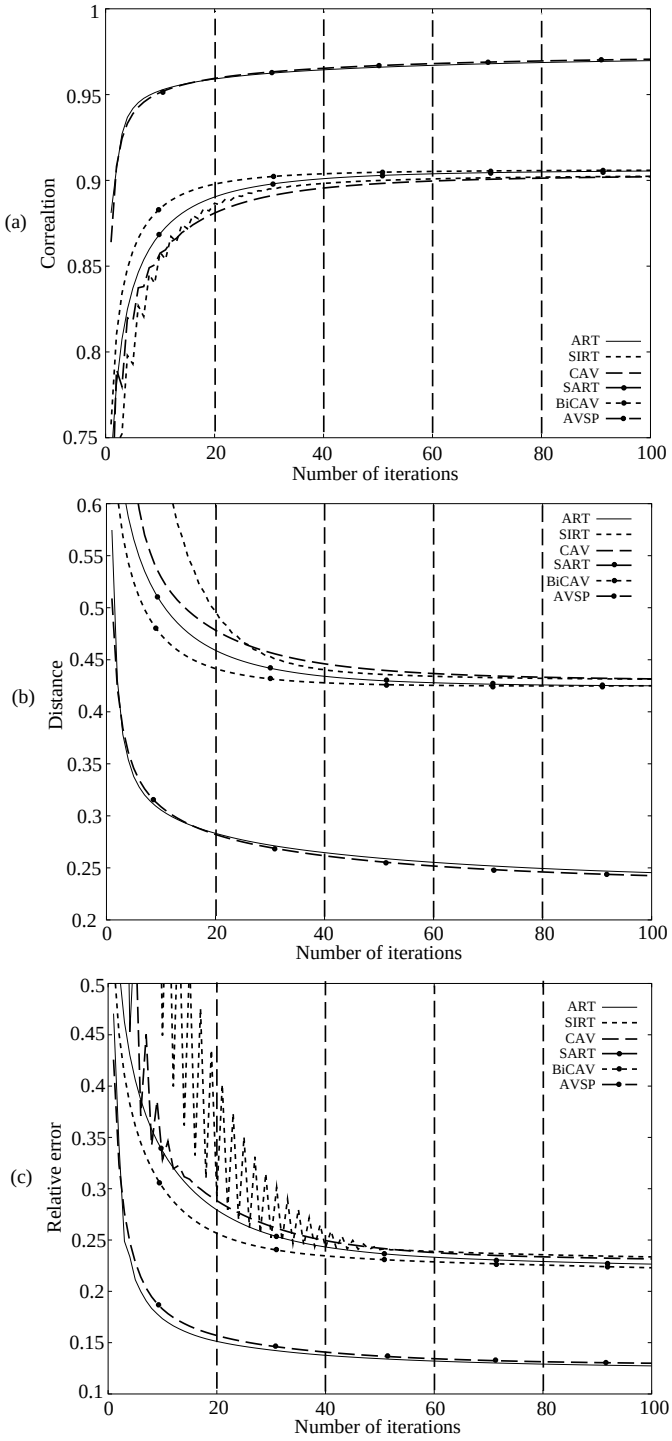


Figure 5.5: Correlation (a), distance (b) and relative error (c) to the phantom.

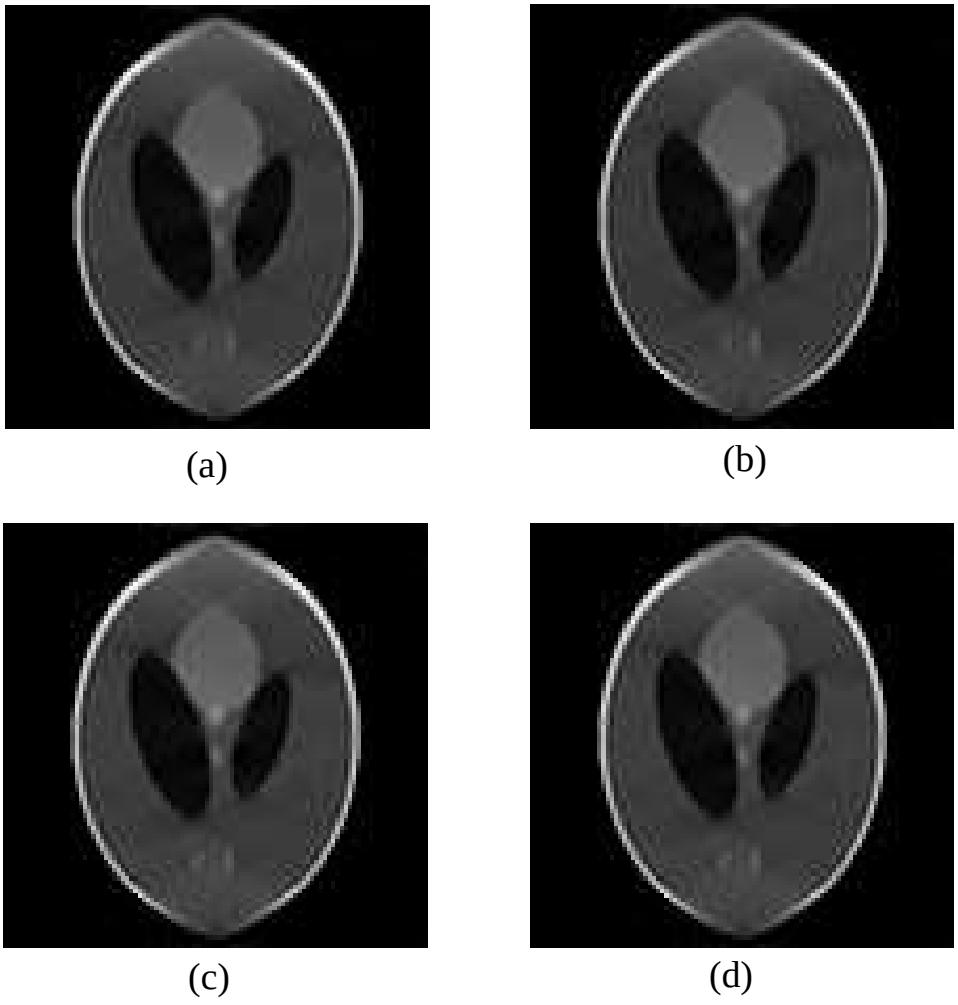


Figure 5.6: Images with same value of the correlation to the phantom. Reconstructed using λ =0.2 (a), 2 (b), 3 (c) and 4 (d).

Chapter 6

Conclusions

This final chapter will present the conclusions drawn from the constructing and testing of different tomographic reconstruction algorithms. All the testing has been done using only one phantom, the Shepp-Logan phantom, and only one way of simulating a projection of it (described in section 2.1.2). Therefore there is no guarantee that these conclusions are valid in other contexts than the one in this report. Still they will hopefully give a hint of what direction to take in the future.

6.1 Blobs are better than pixels.

The first thing shown by the tests was that the use of blobs as basis element renders superior reconstructions compared to pixels, especially visually. The blobs render a very smooth image, which can be interpreted much easier than the often very noisy image obtained by the use of pixels. The smoothness, at the cost of sharpness, can be increased by using wider blobs (blobs with a greater radius). During the work with the thesis it was established that a blob with the radius of 2.795 gave the best result. The other data of this blob is: $m = 2$ and $\alpha = 16.36$. The radial shape of the blob facilitates the calculation of correct and exact weighting values representing the relation between rays and components. Doing this using pixels, leads to complicated calculations because of their unnatural square shape.

6.2 The best algorithm

The results showed that two algorithms were significantly better than the other five algorithms that were tested. They are both algebraic algorithms

and are called ART (Arithmetic Reconstruction Technique) and AVSP (AVeraging of Sequential Projections, as it was chosen to be called in this report). AVSP is based on the algorithm described in [9]. With the AVSP algorithm different ways of partitioning the equations into blocks were tried out. The conclusion from this, was that a randomized partitioning of the equations gives the best results.

According to the results, the CAV technique, used by two algorithms (CAV and BiCAV) in this report, did not give more than a very small improvement of the correlation to the phantom. However, it did allow the algorithm to use a much lower relaxation factor. This showed, according to visual comparisons, to be an advantage. An image produced using a smaller relaxation factor is smoother and clearer, compared to one produced using a higher relaxation factor. This is true even when they have the same value of the correlation to the phantom. Therefore the CAV technique is valuable.

In spite of the fact that the analytical algorithm called Filtered Backprojection certainly provides the fastest reconstruction, it is not the best. It is the algorithm most widely used today, but have high demands when it comes to the arrangement of the projections. The algebraic algorithms are significantly slower, but in the end, give better results. Furthermore they are not dependent on the arrangement of the projections. This makes them superior when only data from scattered projections are available.

Another thing that should be taken into account is the possibility to parallelize the algorithms. All the algebraic algorithms except ART are, to different levels, possible to parallelize. This helps to convince me that AVSP is a better choice than ART. The work for calculating AVSP, could be divided on as many processors, as blocks that are being used. The same is valid for SIRT, SART, CAV and BiCAV. For SIRT, SART and CAV it holds that the less blocks the equations are being partitioned into, the higher is the level of parallelization that can be reached. For AVSP it is the opposite.

If an intent to rank the algorithms on basis of the results obtained in this project would be done, the list would look like this.

1. AVSP
2. ART
3. BICAV
4. SART
5. CAV

6. SIRT

7. Filtered backprojection

The differences between positions 3 to 6 is very small and it is not with any great certainty to claim that their positions, with respect to each other, should be like it is.

6.3 Further testing

In order to draw more general conclusions it is probably necessary to do the testings presented in this report also on other objects than the Shepp-Logan phantom. Since all the results are based on the reconstructions of this phantom, qualities and drawbacks of some algorithms might have been missed. The same thing is true for the projections. It would have been preferable to do more tests, when only data from sparse and scattered reconstructions are available. In that context it is probable that the algebraic algorithms would really show their true potential.

Another area for further testing is the algorithms stability using distorted projections (noisy data). Since no real projections are free from distortions, algorithms which are able to function well in spite of these would be valuable. Tests based on noisy data would lead to conclusions drawn on results closer to reality. It is probable that the blobs would have shown even more superior to pixels, had noisy data been used. Maybe the SIRT and SIRT-like algorithms would position themselves better in this context. Is their worse exactness perhaps a price that has to be paid for a higher stability?

6.4 Further development of the algorithm

One area where the algorithms probably could be developed, especially to give a quicker reconstruction, is the relaxation factor. In all the tests in this paper the relaxation factor has been fixed throughout the reconstruction. In general it has showed, that after a small number of iterations were performed, a higher relaxation gave better results. With more iterations a lower relaxation factor was preferable. It seems likely that a dynamic relaxation factor, with a value that is sinking with the number of iterations, might be a way well worth of trying. A problem is how the lowering of the relaxation factor should be done. Should it be connected to some logarithmic function perhaps, or maybe stand in relation to some variable describing the state of

the reconstruction progress (such as the correlation between measured and calculated sinogram)?

Another thought is to develop an algorithm that is constantly trying out different ways of improvement. Perhaps like AVSP, but instead of taking an average of the different ways it would choose the best one and then continue from there. This is of course a well known technique within algorithm theory but have not been applied to this area.

Chapter 7

Acknowledgements

The project was performed at the University of Almería from April till September 2001. It was conducted in the department of Computer Architecture and Electronics under the guidance of Assistant Professor José Jesús Fernández. The graduating department is the Department of Computer Science at Lund Institute of Technology in Sweden.

During my time at the University of Almería I have enjoyed a lot of support and help from the staff at "Departamento de Arquitectura de Computadores y Electrónica", which I would like to acknowledge. First and foremost I would like to thank my supervisor José Jesús Fernández Rodríguez, who was always there for me willing to answer my questions, and also helped me finding myself at home in Almería. Secondly I would like to thank all the people in the department, especially Vicente González Ruíz and Leocadio González Casado, who were very nice and always willing to help me with all sorts of problems.

Bibliography

[1] A. C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, New York, 1988.

[2] J. Radon. Uber due bestimmung von funktionen durch ihre intergralwerte langsgewisser mannigfaltigkeiten (on the determination of functions from their integrals along certain manifolds). *Berichte Saechsische Akademie der Wissenschaften*, 1917.

[3] Ronald D. Kriz et al Raman Rao. *Parallel Implementation of the Filtered Back Projection Algorithm for Tomographic Imaging*. PhD thesis, Virginia Polytechnic Institute and State University, 1995.

[4] R. M. Lewitt. Alternatives to voxels for image representation in iterative reconstruction algorithms. *Phys. Med. Biol.*, 37(3):705, 1992.

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

[6] Samuel Matej and Robert M. Lewitt. Practical considertions for 3-d image reconstruction using sperically symmetric volume elements. *IEEE Transactions on Medical Imaging*, 15(1):68–78, February 1996.

[7] F. Jacobs, S.Matej, and R. Lewitt. Image reconstruction techniques for pet. Technical report, Department of Radiology, University of Pennsylvania, USA and Department of Electronics and Information Systems, University of Ghent, Belgium, August 1998.

[8] Yair Censor, Dan Gordon, and Rachel Gordon. Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems. *Parallel Computing*, 27(6):777–808, May 2001.

[9] Y. Censor, T. Elfving, and G. T. Herman. Averaging strings of sequential iterations for convex feasibility problems. In D. Butnariu, Y. Censor, and S. Reich, editors, *Inherently Parallel Algorithms in Feasibility and Optimization and their Applications*, pages 101–114. Elsevier Science Publishers, Amsterdam, The Netherlands, 2001.

[10] A. C. Kak. *Tomographic imaging with diffracting and non-diffracting sources*. Prentice-Hall, 1985.

Appendix A

Results , Correlation

The tables in this appendix show the correlation to the phantom, for a reconstructed image. Some of the results and conclusions in this thesis are built on the results here. They have been put in the appendix since they are not necessary for the understanding of the previous chapters and also because they are relatively large. In the tables the **bold** values are the best ones in each row. The ***emphasized bold*** value is the best one from the whole table. This is the value that gave the choice of relaxation factor and number of blocks.

A.1 SART

	Relaxation factor								
# of Blocks	10	20	30	40	50	60	70	80	90
9	0.7233	0.7546	0.7822	0.8007	0.8131	0.8216	0.8277	**0.8324**	0.7937
11	0.7368	0.7740	0.7996	0.8148	0.8241	0.8300	**0.8335**	0.8280	0.6012
13	0.7425	0.7834	0.8082	0.8224	0.8308	0.8361	**0.8394**	0.8365	0.5402
16	0.7504	0.7945	0.8177	0.8303	0.8377	0.8423	0.8453	**0.8481**	0.7864
21	0.7634	0.8074	0.8277	0.8382	0.8444	0.8481	**0.8503**	0.8496	0.6200
31	0.7834	0.8224	0.8380	0.8458	0.8500	0.8522	0.8532	***0.8534***	0.3486
61	0.8134	0.8386	0.8469	0.8500	**0.8508**	0.8506	0.8498	-	-

Table A.1: 1 iteration

	Relaxation factor								
# of Blocks	7	8	9	10	20	30	40	50	60
9	0.8704	0.8752	0.8792	0.8823	0.8962	0.8998	0.9012	0.9017	**0.9018**
11	0.8810	0.8849	0.8879	0.8903	0.9005	0.9029	0.9036	***0.9037***	0.9036
13	0.8855	0.8889	0.8914	0.8934	0.9015	0.9031	**0.9035**	0.9034	0.9032
16	0.8902	0.8928	0.8949	0.8964	0.9022	**0.9030**	0.9030	0.9027	-
21	0.8947	0.8966	0.8980	0.8990	0.9023	**0.9025**	0.9020	0.9014	-
31	0.8985	0.8995	0.9002	0.9007	**0.9016**	0.9009	0.8999	0.8989	-
61	0.8997	**0.8999**	0.8999	0.8999	0.8982	0.8965	0.8950	0.8936	-

Table A.2: 20 iterations

	Relaxation factor								
# of Blocks	7	8	9	10	20	30	40	50	60
9	-	0.8885	0.8923	0.8949	0.8967	0.8981	0.8991	0.9026	**0.9030**
11	-	0.8951	0.8979	0.8998	0.9011	0.9021	0.9028	***0.9051***	0.9051
13	-	0.8975	0.8998	0.9012	0.9023	0.9030	0.9035	**0.9049**	0.9046
16	-	0.8998	0.9014	0.9024	0.9031	0.9036	0.9039	**0.9045**	0.9038
21	-	0.9016	0.9026	0.9032	0.9036	0.9038	**0.9039**	0.9035	-
31	0.9019	0.9027	0.9031	0.9032	0.9033	**0.9033**	0.9033	0.9019	-
61	**0.9018**	0.9017	0.9015	0.9012	0.9010	0.9007	0.9004	0.8978	-

Table A.3: 50 iterations

	Relaxation factor								
# of Blocks	3	4	5	6	7	8	9	10	20
9	-	-	0.8992	0.9006	0.9015	0.9021	0.9025	0.9028	**0.9035**
11	-	-	0.9030	0.9040	0.9046	0.9050	0.9053	***0.9055***	0.9054
13	0.8999	0.9024	0.9037	0.9044	0.9049	0.9051	0.9053	**0.9054**	0.9049
16	0.9016	0.9034	0.9042	0.9047	0.9049	0.9050	**0.9051**	0.9051	0.9040
21	0.9029	0.9039	0.9044	0.9046	**0.9047**	0.9046	0.9045	0.9044	0.9027
31	0.9035	0.9039	**0.9040**	0.9039	0.9037	0.9035	0.9033	0.9030	0.9006
61	**0.9025**	0.9021	0.9017	0.9013	0.9008	0.9004	0.8999	0.8995	0.8965

Table A.4: 100 iterations

A.2 BiCAV

	Relaxation factor								
# of Blocks	0.4	0.6	0.8	1	2	3	4	5	6
9	0.7136	0.7205	0.7296	0.7397	0.7844	0.8091	0.8228	**0.8298**	0.7414
11	0.7225	0.7331	0.7453	0.7578	0.8013	0.8210	0.8306	**0.8327**	0.2411
13	0.7258	0.7388	0.7531	0.7668	0.8098	0.8280	0.8367	**0.8394**	0.0648
16	0.7301	0.7466	0.7635	0.7780	0.8190	0.8353	0.8429	**0.8472**	0.6191
21	0.7376	0.7591	0.7776	0.7919	0.8289	0.8425	0.8486	**0.8507**	0
31	0.7535	0.7789	0.7967	0.8094	0.8389	0.8488	0.8525	***0.8531***	0
61	0.7879	0.8097	0.8226	0.8309	0.8474	**0.8508**	0.8507	0	0

Table A.5: 1 iteration

	Relaxation factor								
# of Blocks	0.2	0.4	0.6	0.8	1	2	3	4	5
9	0.8325	0.8656	0.8802	0.8880	0.8925	0.9002	0.9017	**0.9019**	0.9018
11	0.8474	0.8770	0.8887	0.8945	0.8978	0.9031	***0.9039***	0.9037	0.8869
13	0.8552	0.8821	0.8921	0.8969	0.8995	0.9033	**0.9036**	0.9033	0.8977
16	0.8640	0.8874	0.8954	0.8990	0.9009	**0.9032**	0.9030	0.9023	0.9008
21	0.8738	0.8927	0.8984	0.9008	0.9018	**0.9026**	0.9018	0.9006	0.8993
31	0.8846	0.8974	0.9005	0.9015	**0.9018**	0.9010	0.8994	0.8979	0.8958
61	0.8950	0.8996	**0.9000**	0.8998	0.8993	0.8965	0.8942	0.8922	0

Table A.6: 20 iterations

	Relaxation factor								
# of Blocks	0.2	0.4	0.6	0.8	1	2	3	4	5
9	0.8743	0.8930	0.8985	0.9008	0.9019	**0.9032**	0.9029	0.9024	0.9017
11	0.8843	0.8983	0.9024	0.9040	0.9047	***0.9052***	0.9045	0.9037	0.6021
13	0.8885	0.9001	0.9032	0.9043	**0.9048**	0.9047	0.9038	0.9028	0.6887
16	0.8928	0.9017	0.9038	0.9044	**0.9046**	0.9039	0.9028	0.9015	0.8869
21	0.8970	0.9028	0.9039	**0.9042**	0.9041	0.9026	0.9011	0.8996	0.8982
31	0.9004	0.9032	**0.9034**	0.9031	0.9027	0.9004	0.8984	0.8968	0.8952
61	0.9016	**0.9016**	0.9007	0.8999	0.8991	0.8958	0.8935	0.8918	0

Table A.7: 50 iterations

	Relaxation factor								
# of Blocks	0.1	0.2	0.4	0.6	0.8	1	2	3	4
9	-	0.8931	0.9009	0.9027	0.9033	**0.9036**	0.9032	0.9022	0.9012
11	-	0.8985	0.9042	0.9054	0.9058	***0.9059***	0.9046	0.9030	0.9018
13	-	0.9003	0.9046	0.9054	**0.9056**	0.9054	0.9038	0.9021	0.9008
16	-	0.9019	0.9048	**0.9052**	0.9051	0.9048	0.9026	0.9009	0.8996
21	-	0.9031	**0.9047**	0.9046	0.9042	0.9036	0.9011	0.8993	0.8978
31	-	0.9036	**0.9040**	0.9033	0.9025	0.9018	0.8988	0.8967	0.8951
61	0.9021	0.9026	0.9013	0.9000	0.8988	0.8979	0.8945	0.8922	0.8905

Table A.8: 100 iterations

A.3 AVSP

Partitioning method	# of Blocks	Relaxation factor 0.2	0.6	1	2	3	4
Greatest distance	31	0.6992	0.6963	**0.7047**	0.6821	0	-
Random	2	0.8302	***0.8654***	0.8602	0.6971	0.0267	-
	4	0.7929	0.8429	**0.8498**	0.7601	0.2371	-
	6	0.7694	0.8271	**0.8428**	0.8060	0.4621	-
	8	0.7540	0.8164	**0.8337**	0.8078	0.5834	-
	10	0.7442	0.8035	**0.8253**	0.8135	0.6659	-
	31	0.7140	0.7410	0.7634	**0.7985**	0.7859	0.7185
	50	0.7065	0.7269	0.7400	0.7730	**0.7840**	0.7643
	100	0.7000	0.7133	0.7210	0.7414	0.7584	**0.7592**
	200	0.6956	0.7031	0.7078	0.7206	0.7271	**0.7420**

Table A.9: 1 iteration

Partitioning method	# of Blocks	Relaxation factor 0.2	0.6	1	2	3	4
Greatest distance	31	0.8600	0.9083	0.9207	**0.9214**	0	-
Random	2	0.9489	0.9578	***0.9598***	0.9522	0	-
	4	0.9386	0.9529	**0.9559**	0.9544	0	-
	6	0.9302	0.9485	0.9528	**0.9547**	0.5916	-
	8	0.9225	0.9452	0.9502	**0.9531**	0.9180	-
	10	0.9156	0.9416	0.9476	**0.9524**	0.9292	-
	31	0.8628	0.9140	0.9282	0.9419	**0.9438**	0.9336
	50	0.8342	0.8941	0.9153	0.9335	0.9388	**0.9395**
	100	0.7877	0.8588	0.8858	0.9136	0.9266	**0.9294**
	200	0.7478	0.8155	0.8477	0.8852	0.9033	**0.9136**

Table A.10: 20 iterations

Partitioning method	# of Blocks	Relaxation factor 0.2	0.6	1	2	3	4
Greatest distance	31	0.9071	0.9372	0.9438	0.9407	0	-
Random	2	0.9602	0.9660	***0.9670***	0.9619	0	-
	4	0.9536	0.9628	**0.9647**	0.9640	0	-
	6	0.9485	0.9601	0.9627	**0.9639**	0.5338	-
	8	0.9442	0.9577	0.9609	**0.9627**	0.9449	-
	10	0.9403	0.9555	0.9594	**0.9625**	0.9354	-
	31	0.9086	0.9394	0.9474	0.9557	**0.9567**	0.9515
	50	0.8870	0.9286	0.9401	0.9504	**0.9541**	0.9538
	100	0.8486	0.9056	0.9236	0.9387	0.9460	**0.9478**
	200	0.8027	0.8722	0.8975	0.9233	0.9338	**0.9387**

Table A.11: 50 iterations

Partitioning method	# of Blocks	Relaxation factor 0.2	0.6	1	2	3	4
Greatest distance	31	0.9305	0.9494	0.9540	0.9494	0	-
Random	2	0.9663	0.9704	***0.9709***	0.9663	0	-
	4	0.9616	0.9679	**0.9695**	0.9687	0	-
	6	0.9576	0.9663	0.9679	**0.9685**	0.4510	-
	8	0.9545	0.9645	0.9666	**0.9675**	0.9561	-
	10	0.9516	0.9630	0.9659	0.9678	0.6346	-
	31	0.9312	0.9509	0.9568	0.9631	**0.9637**	0.9605
	50	0.9171	0.9437	0.9514	0.9589	**0.9621**	0.9609
	100	0.8874	0.9293	0.9404	0.9504	0.9558	**0.9567**
	200	0.8485	0.9062	0.9241	0.9402	0.9471	**0.9503**

Table A.12: 100 iterations

Appendix B

The Fourier slice theorem

This appendix contains the mathematical explanation of the Fourier slice theorem. It is based on the description in [1].

We start by defining the two dimensional Fourier transform of the object as

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}\,\mathrm{d}x\,\mathrm{d}y. \tag{B.1}$$

Then we define the one dimensional Fourier transform of a projection at an angle θ, $P_\theta(t)$, as

$$S_\theta(w) = \int_{-\infty}^{\infty} P_\theta(t)e^{-j2\pi wt}\mathrm{d}t. \tag{B.2}$$

In order to see the relation between B.1 and B.2 we will start by looking at an example of the Fourier slice theorem. The simplest one is given by taking a projection at $\theta = 0^o$. We first take a look at the two dimensional projection of the object along the line in the frequency domain given by $v = 0$. This is given by

$$F(u,0) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi ux}\mathrm{d}x\mathrm{d}y. \tag{B.3}$$

Since the phase factor is no longer dependent on v we can split the equation into two parts.

$$F(u,0) = \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(x,y)\mathrm{d}y\right] e^{-j2\pi ux}\mathrm{d}x \tag{B.4}$$

We recognize the term within the brackets as the line integral (2.5) for a line with constant x value or, expressed mathematically,

$$P_{\theta=0}(x) = \int_{-\infty}^{\infty} f(x,y)dy. \tag{B.5}$$

Substituting the term in brackets from B.4 by $P_{\theta=0}(x)$ we get

$$F(u,0) = \int_{-\infty}^{\infty} P_{\theta=0}(x) e^{-j2\pi ux} \mathrm{d}x \tag{B.6}$$

The right hand side of this equation is the one dimensional Fourier transform of the projection $P_{\theta=0}$. We now have the following relationship between the 1-D transform of the vertical projection and the 2-D transform of the object function:

$$F(u,0) = S_{\theta=0}(u) \tag{B.7}$$

This result describes the Fourier slice theorem for $\theta = 0^o$, but the theorem is valid for any angle. This is easy to observe if one takes into account the fact that the result is independent of the orientation between the coordinate system and the object. For example if B.5 is used with the (t,s) coordinate system from figure B.1 we would get the two dimensional Fourier transform along a line rotated by θ. This shows that the relationship holds for any angle, which leads us to the definition of the Fourier slice theorem [10] which is stated as:

> The Fourier transform of a parallel projection of an image $f(x, y)$ taken at an angle θ gives a slice of the two-dimensional transform, $F(u, v)$, subtending an angle θ with the u-axis. In other words, the Fourier transform of $P_\theta(t)$ gives the values of $F(u, v)$ along line BB in figure B.1.

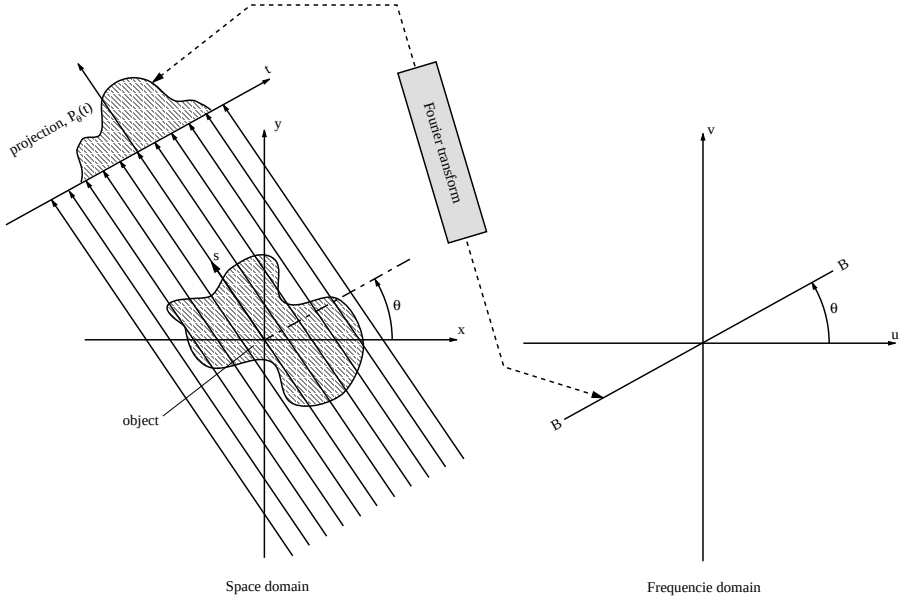


Figure B.1: The relation between the Fourier transform of the projection and the Fourier transform of the original image that is showed by the Fourier slice theorem.

Appendix C

The projection formula

The orthogonal projection on a hyperplane from a point in the n-dimensional space is described by:

$$\overline{f}^{(k)} = \overline{f}^{(k-1)} - \frac{(\overline{f}^{(k-1)} \cdot \overline{a}_i - p_i)}{\overline{a}_i \cdot \overline{a}_i} \overline{a}_i. \tag{C.1}$$

where $\overline{a}_i = (a_{i1}, a_{i2}, \ldots, a_{iN})$, and $\overline{a}_i \cdot \overline{a}_i$ is the scalar product of $\overline{a}_i$ with itself. To deduce the origin of equation C.1, we will start by writing the first equation of 2.8 as

$$\overline{a}_1 \cdot \overline{f} = p_1. \tag{C.2}$$

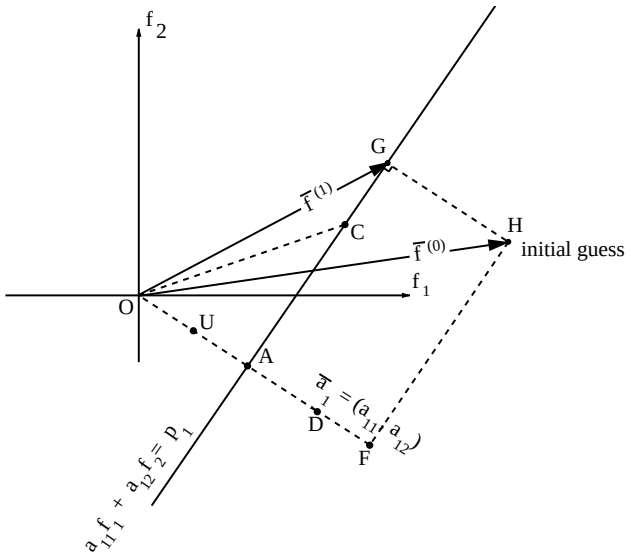


Figure C.1: The figure displays the points and vectors used in the explanation of the formula for perpendicular projection onto the hyperplanes, equation 2.11.

The vector $\overline{a}_1$ in this equation describes a line perpendicular to the one described by the equation itself. This is illustrated in figure C.1, where the

equation is described by the line crossing A and G, and the vector $\overline{a}_1$ is visualized by the line OD. The following equation gives us the unity vector along $\overline{a}_1$:

$$\overrightarrow{OU} = \frac{\overline{a}_1}{\sqrt{\overline{a}_1 \cdot \overline{a}_1}} \tag{C.3}$$

Since the projection of $\overrightarrow{OC}$ (for any point C on the hyperplane described by equation one) on $\overline{a}_1$, will be equal to $\overrightarrow{OA}$ in figure C.1, we have an expression for the perpendicular distance to the hyperplane from the origin:

$$\begin{aligned} |\overrightarrow{OA}| &= \overrightarrow{OU} \cdot \overrightarrow{OC} = \frac{1}{\sqrt{\overline{a}_1 \cdot \overline{a}_1}} (\overline{a}_1 \cdot \overrightarrow{OC}) = \\ &= \frac{1}{\sqrt{\overline{a}_1 \cdot \overline{a}_1}} (\overline{a}_1 \cdot \overline{f}^{(1)}) = \frac{p_1}{\sqrt{\overline{a}_1 \cdot \overline{a}_1}} \end{aligned} \tag{C.4}$$

We can use the distance from the hyperplane to the initial guess, $\overrightarrow{GH}$, to describe a connection between $\overline{f}^{\,(1)}$ and $\overline{f}^{\,(0)}$, which is what we are ultimately looking for. This connection is described by:

$$\overline{f}^{\,(1)} = \overline{f}^{\,(0)} - \overrightarrow{GH} \tag{C.5}$$

where the length of the vector $\overrightarrow{GH}$ is given by:

$$|\overrightarrow{GH}| = |\overrightarrow{OF}| - |\overrightarrow{OA}| = \overline{f}^{\,(0)} \cdot \overrightarrow{OU} - |\overrightarrow{OA}| \tag{C.6}$$

Substituting C.3 and C.4 into C.6 we get:

$$|\overrightarrow{GH}| = \frac{\overline{f}^{\,(0)} \cdot \overline{a}_1 - p_1}{\sqrt{\overline{a}_1 \cdot \overline{a}_1}} \tag{C.7}$$

Since the direction of $\overrightarrow{GH}$ is the same as for the unity vector we can write:

$$\overrightarrow{GH} = |\overrightarrow{GH}|\,\overrightarrow{OU} = \frac{\overline{f}^{\,(0)} \cdot \overline{a}_1 - p_1}{\sqrt{\overline{a}_1 \cdot \overline{a}_1}} \cdot \frac{\overline{a}_1}{\sqrt{\overline{a}_1 \cdot \overline{a}_1}} = \frac{\overline{f}^{\,(0)} \cdot \overline{a}_1 - p_1}{\overline{a}_1 \cdot \overline{a}_1} \overline{a}_1 \tag{C.8}$$

Substituting C.8 in C.5 we get C.1.

Appendix D

The program

The program clarifies the process of reconstruction through a graphical interface. It shows how the reconstructed image develops through the different stages of the reconstruction both visually and in terms of for example correlation to the phantom. All the results from the reconstructions are easily accessible by the use of an "info-frame". This is displayed when ever the left mouse button is pressed over the image of interest and contains all the interesting data about an object. For the reconstructed images this data includes for example: the correlation, distance and relative error between the reconstructed image and the phantom, for each number of iterations up to the last one. This lets the user follow the progress of the algorithm throughout the reconstruction. To perform a large number of reconstructions, the program can read files containing all the settings for the different reconstructions listed in the file. Thanks to this, it is possible to perform lots of different very time consuming reconstructions automatically.

The program is structured around a main frame (a window), which contains a dynamic menu system from where all the functions can be accessed. The menu is dynamic in the sense that its contents change depending on the state of the program. New objects created or loaded are for example added to the show menu.

The menu has four main parts; the *file*, *show*, *tool* and *frames* menu, as can be seen in figure D.1. In the *file* menu all the ordinary commands connected to file handling, like *open* and *save* can be found. The possibility of saving the object as a GIF image is also given. The *show* menu allows you to know which objects are available and lets you decide which one of them to show. It also lets you see the pixel values of the current image, which is useful when evaluating algorithms. The third menu is the *tool* menu. Here you find all the tools you can use to create or convert an object. These include the most important one: *reconstruct*. The last menu is the *frames* menu. This one so far only includes the option to show the image in a new frame, which is very useful when one wants to compare visually two or more images.

The following objects constitute the main part of the program.

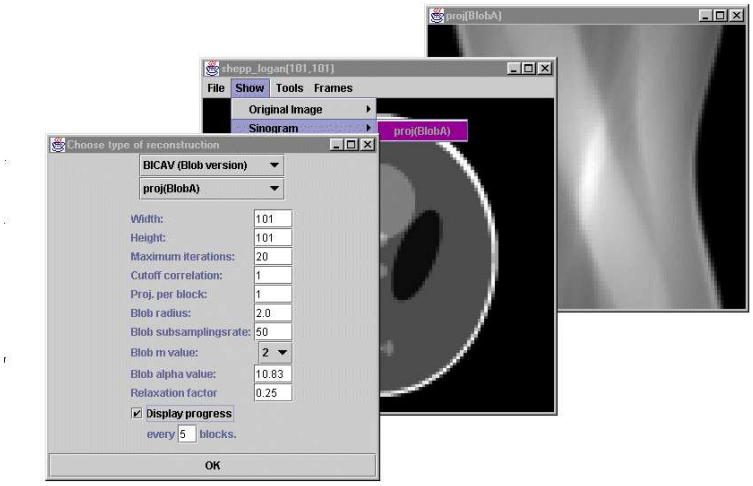


Figure D.1: A screen shot of the program .

- **OrigImage**
- **Sinogram**
- **ReconstructedImage**
- **The reconstruction algorithms**.
- **The main window**.
- **The option frames**

The three first objects represent different stages of the reconstruction process. *OrigImage* contains the phantom (see section 2.1.1). *Sinogram* contains the projections of one *OrigImage*, but also information on how the projection was done, number of projection angles etc., and on which *OrigImage* the projection was done. The object *ReconstructedImage* of course contains the reconstructed image. It also holds all the facts on how the reconstruction was performed plus information about the *Sinogram* that was used. These three objects are actually all connected to each other so that the chain of the reconstruction can be followed. This is implemented by a link in *ReconstructedImage* to the *Sinogram* and a link in the *Sinogram* to the *OrigImage* that was projected.

In the main window, which extends JFrame, pointers to all the above mentioned objects are kept in linked lists. This makes it easy to access, add and remove them. Through the menus, new windows are opened when for example a reconstruction will be done. In these windows, all the settings for the operation is set. In figure D.1 a BICAV reconstruction is about to be performed.

Every reconstruction technique is implemented as a thread. This makes it possible to stop it at any time, which is good when you graphically want to represent the stage at which the algorithm is at the moment. This would not be possible without a thread since the algorithm takes up all of the processor power. Therefore the image will not be updated until the algorithm has finished. With the algorithm as a thread it is possible to stop the reconstruction while the image is being drawn, leaving all the processor power to the drawing. This of course, slows down the process and therefore it is optional to display the progress of the reconstruction. Figure D.2 shows how the program is displaying the progress of the Filtered Backprojection algorithm. Having the algorithm as a thread also enables the performance of other tasks while the reconstruction is running. Two or more reconstructions can even be run concurrently.

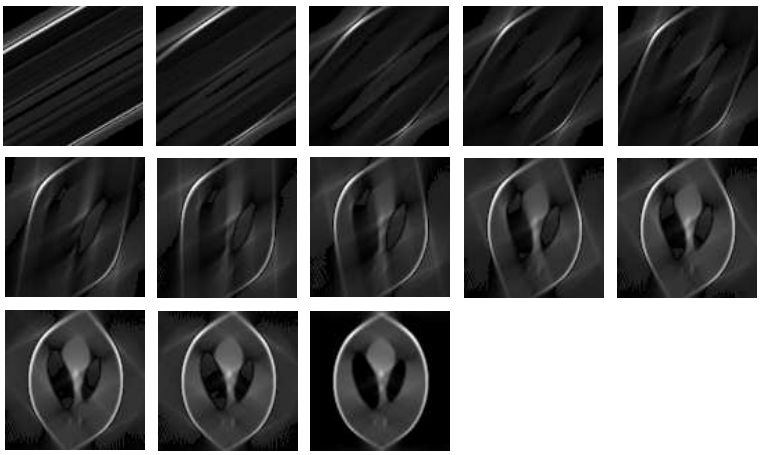


Figure D.2: The progress of the Filtered Backprojection algorithm through the reconstruction, as visualized by the program.