UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Development of a New 3D Reconstruction Algorithm for Computed Tomography (CT)

Amadeo Iborra Carreres

Advisors

Dra. María José Rodríguez Álvarez

Dr. Antonio Soriano Asensi

Doctorado en Investigación Matemática
Instituto de Instrumentación para Imagen Molecular July, 2015

a Blanca

Fairy tales are more than true,
not because they tell us dragons exist,
but because they tell us dragons can be beaten.

G. K. Chesterton

Abstract

Model-based computed tomography (CT) image reconstruction is dominated by iterative algorithms. Although long reconstruction times remain as a barrier in practical applications, techniques to speed up its convergence are object of investigation, obtaining impressive results. In this thesis, a direct algorithm is proposed for model-based image reconstruction. The model-based approximation relies on the construction of a model matrix that poses a linear system which solution is the reconstructed image. The proposed algorithm consists in the QR decomposition of this matrix and the resolution of the system by a backward substitution process. The cost of this image reconstruction technique is a matrix vector multiplication and a backward substitution process, since the model construction and the QR decomposition are performed only once, because of each image reconstruction corresponds to the resolution of the same CT system for a different right hand side.

Several problems regarding the implementation of this algorithm arise, such as the exact calculation of a volume intersection, definition of fill-in reduction strategies optimized for CT model matrices, or CT symmetry exploit to reduce the size of the system. These problems have been detailed and solutions to overcome them have been proposed, and as a result, a proof of concept implementation has been obtained.

Reconstructed images have been analyzed and compared against the filtered backprojection (FBP) and maximum likelihood expectation maximization (MLEM) reconstruction algorithms, and results show several benefits of the proposed algorithm. Although high resolutions could not have been achieved yet, obtained results also demonstrate the prospective of this algorithm, as great performance and scalability improvements would be achieved with the success in the development of better fill-in strategies or additional symmetries in CT geometry.

Resumen

En la reconstrucción de imagen de tomografía axial computerizada (TAC), en su modalidad *model-based*, prevalecen los algoritmos iterativos. Aunque los altos tiempos de reconstrucción aún son una barrera para aplicaciones prácticas, diferentes técnicas para la aceleración de su convergencia están siendo objeto de investigación, obteniendo resultados impresionantes. En esta tesis, se propone un algoritmo directo para la reconstrucción de imagen *model-based*. La aproximación *model-based* se basa en la construcción de una matriz modelo que plantea un sistema lineal cuya solución es la imagen reconstruida. El algoritmo propuesto consiste en la descomposición QR de esta matriz y la resolución del sistema por un proceso de sustitución regresiva. El coste de esta técnica de reconstrucción de imagen es un producto matriz vector y una sustitución regresiva, ya que la construcción del modelo y la descomposición QR se realizan una sola vez, debido a que cada reconstrucción de imagen supone la resolución del mismo sistema TAC para un término independiente diferente.

Durante la implementación de este algoritmo aparecen varios problemas, tales como el cálculo exacto del volumen de intersección, la definición de estrategias de reducción del relleno optimizadas para matrices de modelo de TAC, o el aprovechamiento de simetrías del TAC que reduzcan el tamaño del sistema. Estos problemas han sido detallados y se han propuesto soluciones para superarlos, y como resultado, se ha obtenido una implementación de prueba de concepto.

Las imágenes reconstruidas han sido analizadas y comparadas frente a los algoritmos de reconstrucción filtered backprojection (FBP) y maximum likelihood expectation maximization (MLEM), y los resultados muestran varias ventajas del algoritmo propuesto. Aunque no se han podido obtener resoluciones altas aún, los resultados obtenidos también demuestran el futuro de este algoritmo, ya que se podrían obtener mejoras importantes en el rendimiento y la escalabilidad con el éxito en el desarrollo de mejores estrategias de reducción de relleno o simetrías en la geometría TAC.

Resum

En la reconstrucció de imatge tomografia axial computerizada (TAC) en la seua modalitat *model-based* prevaleixen els algorismes iteratius. Tot i que els alts temps de reconstrucció encara són un obstacle per a aplicacions pràctiques, diferents tècniques per a l'acceleració de la seua convergència estàn siguent objecte de investigació, obtenint resultats impressionants. En aquesta tesi, es proposa un algorisme direct per a la recconstrucció de image *model-based*. L'aproximació *model-based* es basa en la construcció d'una matriu model que planteja un sistema lineal quina sol·lució es la imatge reconstruida. L'algorisme propost consisteix en la descomposició QR d'aquesta matriu i la resolució del sistema per un procés de substitució regresiva. El cost d'aquesta tècnica de reconstrucció de imatge es un producte matriu vector i una substitució regresiva, ja que la construcció del model i la descomposició QR es realitzen una sola vegada, degut a que cada reconstrucció de imatge suposa la resolució del mateix sistema TAC per a un tèrme independent diferent.

Durant la implementació d'aquest algorisme sorgixen diferents problemes, tals com el càlcul exacte del volum de intersecció, la definició d'estratègies de reducció de farcit optimitzades per a matrius de model de TAC, o el aprofitament de simetries del TAC que redueixquen el tamany del sistema. Aquestos problemes han sigut detallats y s'han proposat solucions per a superar-los, i com a resultat, s'ha obtingut una implementació de prova de concepte.

Les imatges reconstruides han sigut analitzades i comparades front als algorismes de reconstrucció filtered backprojection (FBP) i maximum likelihood expectation maximization (MLEM), i els resultats mostren varies ventajes del algorisme propost. Encara que no s'han pogut obtindre resolucions altes ara per ara, els resultats obtinguts també demostren el futur d'aquest algorisme, ja que es prodrien obtindre millores importants en el rendiment i la escalabilitat amb l'éxit en el desemvolupament de millors estratègies de reducció de farcit o simetries en la geometria TAC.

Contents

Chapter 1

Introduction

1.1 Context of medical imaging

The main goal of medical imaging is to obtain an image representing relevant information of the interior of the body. This information is used to improve diagnoses, to locate internal structures or lesions, or to guide procedures such as surgery among other applications, becoming a key tool in the clinical practice. The medical imaging main advantage is that its images are obtained in a non-invasive way. It is not necessary to perform surgery or introduce any instrument inside the body to obtain the information.

Obtained information is related to a physical parameter of interest inside the body of the patient. Therefore, the diagnostic utility of a medical image relies in its technical quality. This quality is determined by the accuracy of the represented physical parameter and the accuracy in its spatial distribution through the body.

Different types of medical images can be made considering a different physical parameter of interest. The different modes of making images are referred to as *modalities* and each modality has its own applications in medicine.

1.1.1 Medical imaging modalities

Medical imaging requires some form of energy capable of penetrating tissues. The general idea is to detect energy that has passed through the body and experienced some type of interaction with the internal anatomy. Then, using that information, the internal anatomy is reconstructed. The existing medical imaging modalities differ from one another in the types of energies and the detection technology used.

Radiography

Radiography was the first medical imaging technique and it came through the discovery of x-rays in 1895 by the physicist Wilhelm Roentgen, who also made the first radiographic images of human anatomy. Radiography is based in the transmission of x-rays through the body. It is performed with an x-ray source and an x-ray detector. The x-ray source produces an x-ray beam that passes through the patient and reaches the detector. The radiographic image is a picture of the resulting x-ray distribution. The attenuation properties of tissues such as bone, soft tissue, and air inside the patient are very different. Therefore, the measured attenuation map in the detector contains anatomical information of the internal structures of the body (see Figure 1.1).

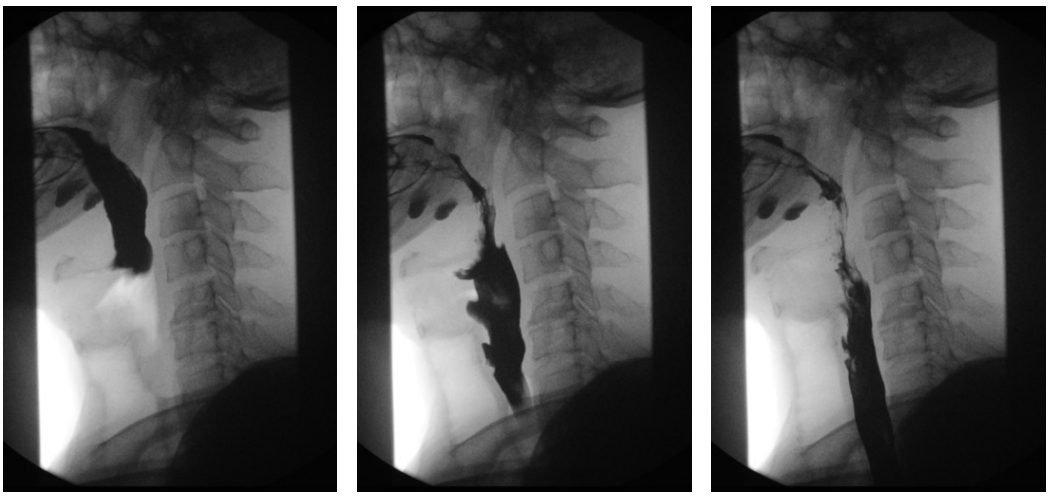
Radiographic images are 2D projections of the body along the x-ray beam. Internal structures at different depths along the x-ray beam appear superimposed in the resulting 2D image. Radiographic images are useful for the diagnosis of broken bones, lung cancer, cardiovascular disorders, among other medical indications.

Figure 1.1: Radiographic image made by Roentgen of his wife's hand. The bones of her hand as well as two rings on her finger are clearly visible.

Fluoroscopy

Fluoroscopy refers to the continuous acquisition of a sequence of radiographic images over time, using rapid x-ray detector systems (see Figure 1.2). Fluoroscopy is

often used for positioning catheters in arteries, visualizing contrast agents (liquids that are ingested or injected to the patient and have high attenuation coefficients), or for other medical applications where real-time image feedback is required.



(a) (b) (c)

Figure 1.2: Three different frames (a), (b) and (c), of a fluoroscopic image of a human neck during a barium sulfate swallow [1].

Magnetic resonance imaging

The magnetic resonance imaging (MRI) is based in the magnetic properties of the hydrogen nuclei, abundant in biological tissues. It is performed by introducing the patient inside a high electromagnetic field and variable radio-frequency fields. These magnetic fields align the protons of the hydrogen atoms. When the radio-frequency fields disappear, the aligned protons regain their original state, emitting radio signals. The MRI system uses the frequency and phase of the returning radio waves to determine the position of each signal from the patient. The resulting information is processed in a computer to form an image. Because different types of tissue have different local magnetic properties, images made using MRI contain anatomical information of the internal structures of the body (see Figure 1.3).

MRI produces a set of 2D images depicting slices through the patient, forming a 3D image. MRI is extensively used in neurological imaging (head and spine) or for musculoskeletal applications (such as imaging knee injuries) among other medical indications. However, due to the use of electromagnetic fields, MRI should not be performed on patients who have internal ferromagnetic implants, such as metal plates or cardiac pacemakers.

Figure 1.3: Sagittal projection of a brain MRI (Neuro-Ophthalmology unit of Vargas de Caracas Hospital, Venezuela).

Ultrasound imaging

Ultrasound refers to high-frequency sound below the human audible spectrum. Ultrasound imaging is based in the propagation of the mechanical energy of the sound through the tissue and in the reflection of the sound waves by internal structures inside the body. It is performed by the echo detection after an ultrasound pulse using piezoelectric materials for the ultrasound emission and echo detection. The Ultrasound imaging system uses the echo delay and amplitude to form an image (see Figure 1.4). Produced images can be 2D tomographic slices, 3D images, or motion studies. Because this modality is thought to be less harmful to a growing fetus than others (as x-ray ionizing radiation, for example), ultrasound imaging is extensively used in obstetrical patients.

Figure 1.4: Ecography using ultrasound imaging. Hospital Universitario La Fe, Valencia, España.

Nuclear medicine imaging

Nuclear medicine imaging is a form of functional imaging that provides information about a wide range of biological processes that take place inside the body. The basic principle of operation is to give to the patient orally, by injection or by inhalation a compound with a radioactive isotope (commonly gamma emitting or positron emitting isotope). The gamma rays emitted during radioactive decay of the isotope are detected outside of the patient body. This technique produces 3D images which contain information of the spatial concentration of the given compound inside the body, from which biological information is derived (see Figure 1.5). Nuclear medicine imaging has become a routine diagnostic tool for cancer staging and has applications in oncology, cardiology or neurology.

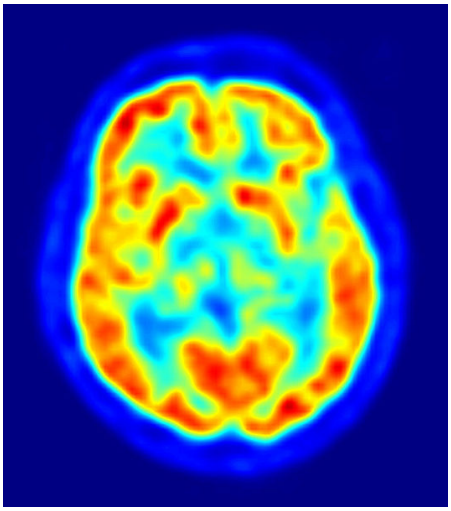


Figure 1.5: Transaxial slice of a brain [2] taken with positron emission tomography (PET). Red areas show more accumulated tracer substance and blue areas are regions where low to no tracer have been accumulated.

Computed tomography

A computed tomography (CT) scan is a medical imaging technique that combines a series of x-ray images taken from different angles and uses computer processing to create cross-sectional images of the bones, blood vessels and soft tissues inside the body of a patient (see Figure 1.6). Is based in the same principle as the radiography, that is, the anatomical structures inside the body will produce an attenuation map in the x-ray beam that reach the detector.

In 1971, G. N. Hounsfield presented the first CT scanner in the Atkinson Morley Hospital, London, England. In 1979, Hounsfield and A. M. Cormack were awarded with the Nobel Prize in Medicine for the development of the diagnostic technique of x-ray CT. Nowadays, the x-ray CT has become an important tool for diagnostic evaluation regarding internal injuries or bone fractures, the location of a tumor, infection or blood clot, the detection and monitoring of diseases and conditions such as cancer, heart disease, lung nodules or liver masses, and to guide procedures such as surgery, biopsy and radiation therapy.

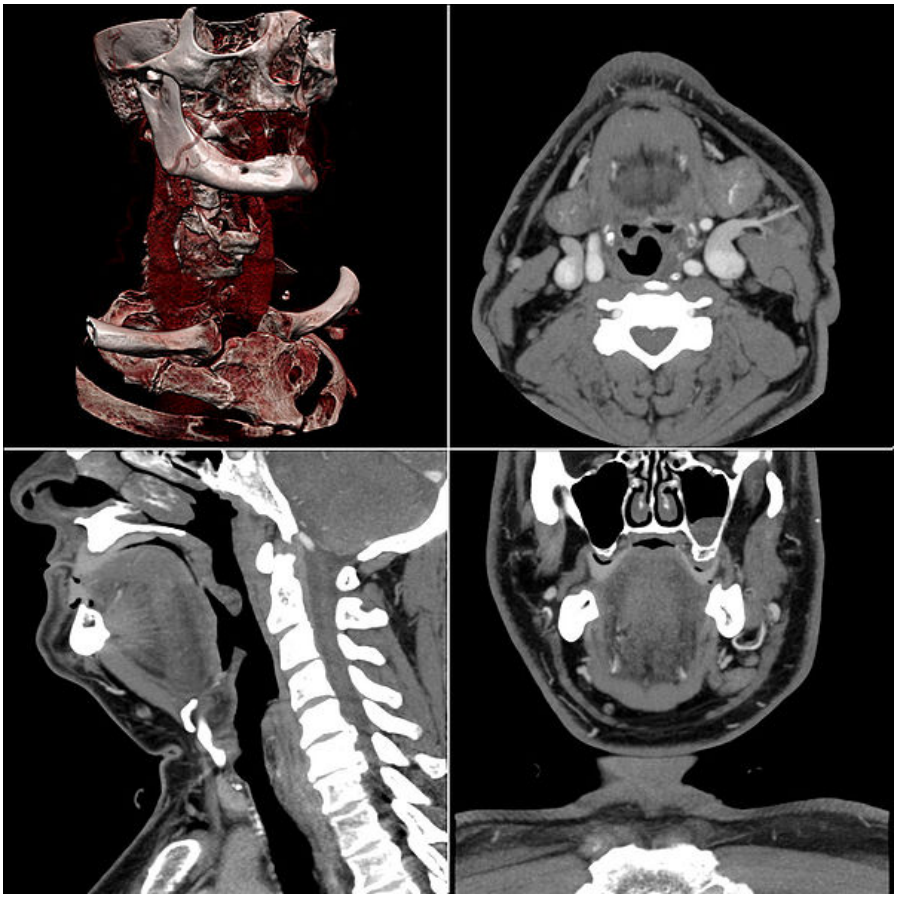


Figure 1.6: A CT study [3]. Clockwise from top-left: Volume rendering overview, axial slices, coronal slices and sagittal slices.

This thesis is devoted to the CT imaging modality and it will be further described in the following sections.

1.2 Computed tomography image reconstruction review

Since the first CT scanners, CT technology has evolved, driven by clinical demands. Advances in x-ray CT technologies have been accompanied by advances in image reconstruction algorithms. In the first CT systems, the algebraic reconstruction technique (ART) [4] was used. ART is based on the resolution of a linear system of equations. The large size of the linear systems encouraged the development of faster reconstruction algorithms. The algorithmic development has been gen-

erally classified into two major categories: analytical reconstruction and iterative reconstruction.

The analytical reconstruction approach, in general, try to formulate the solution in a closed-form equation. This problem, in its general form, was solved by J. Radon in 1917 [5]. The filtered backprojection (FBP) algorithm was developed originally by Feldkamp, Davis, and Kress in 1984 [6]. FBP is based in a rewrite of the Radon transform for two dimensions in the form of a convolution and a backprojection yielding an approximate 3D reconstruction algorithm from 2D projection data. A detailed introduction to this algorithm can be found in the third chapter of Hsieh's book *Computed Tomography* [7]. The image quality and efficiency of the FBP algorithm established it as the standard CT reconstruction algorithm. As the CT technology evolved, modifications to the FBP algorithm have been proposed, to further improve its efficiency [8] or to adapt it to various CT scanning patterns such the backprojection-filtration (BPF) [9], among many others [10, 11, 12, 13, 14, 15]. For further details in analytical reconstruction approaches see [16].

Iterative reconstruction tries to formulate the final result as the solution either to a set of equations or the solution of an optimization problem, which is solved in an iterative fashion. The simultaneous iterative reconstruction technique (SIRT) [17] proposed the update of the solution at the end of each iteration, once all equations have been considered. Then, the update of an element of the solution was the average of all changes computed for that element in the iteration. The simultaneous ART (SART) [18] was developed to combine the advantages of ART and SIRT. Among its main features, SART introduced a bilinear interpolation to reduce errors in the computation of the coefficients of the linear system regarding the CT geometry. Also, the update of its solution was simultaneously applied for an entire CT view.

Statistical methods have been contributed to the CT image reconstruction as well. The principle of statistical methods is to consider the photons involved in the reconstruction process Poisson distributed. The maximum likelihood expectation maximization (MLEM) algorithm consists in an expectation step, which computes the expectation of the log-likelihood using its current estimate, and a maximization step, which finds the next estimate through maximizing the expected log-likelihood. MLEM algorithm was first introduced to emission tomography [19] (becoming a prominent reconstruction algorithm in the field, especially in positron emission tomography [20]) and then, extended to CT [21].

Methods based on iterative coordinate descent (ICD) [22] are the contribution to the image reconstruction of optimization algorithms. These algorithms work by iteratively updating individual elements of the estimate image (or coordinates) to minimize a cost functional. Among its main features, ICD can be efficiently

applied to the log-likelihood expressions (from MLEM) and converges quickly when initialized with a preliminary reconstruction.

The use of ordered subsets (OS) reduced the reconstruction times further by accelerating the convergence of the iterative process. The OS method consists in the division of the projection data into groups called subsets. Then, the update of the solution is performed for each group instead of the complete set of projections. The OS method has been applied to SIRT (OS-SIRT) [23], MLEM (OSEM) [24], and ICD (OS-ICD) [25], increasing significantly its convergence speed.

The model-based designation emerged to classify the methods which consider a model of the problem to obtain the solution. Statistical methods can be classified inside model-based methods, as they model the reconstruction process as Poisson distributed. However, refined models can be proposed, such as finer geometric modeling or physical modeling. In geometric modeling, the device and the image are modeled as three dimensional objects. Within the CT system, the focal spot of the x-ray tube, the detector pixels comprising the detector panel, or the x-ray path through the volumetric elements of the scanned object can be modeled to a convenient degree of accuracy. In physical modeling, the interactions of photons in the measured object are addressed. Within these physical processes, the polychromatic nature of the x-ray beam, and the beam hardening or scatter effects, can be addressed to some level of detail. Regarding the most physical processes that can be modeled, their major challenge is that they require prior knowledge of the material composition of each volumetric element of the scanned object (or at least an approximation) which in turn, is the information that we wish to obtain in the first place. While model-based reconstruction algorithms obtain a better image quality by taking into account the above-mentioned considerations [16, 26], in general, they require to find a good balance between the computational complexity and the obtained improvements in the quality of the reconstructed image.

1.3 Motivation

Iterative reconstruction has exclusively accompanied model-based reconstruction to the point that the former is usually referred to model-based iterative reconstruction (MBIR). This is due to the high computational cost of the model-based algorithms, which was only tractable in an iterative fashion. In fact, there has been a large effort not only in the image quality improvement, but in the acceleration of the convergence speed.

The aim of this thesis is to provide an alternative to the iterative reconstruction of model based algorithms through the direct resolution of a linear system that models the CT device. While focusing to keep the superior image quality that MBIR obtains through a careful modeling of the CT system, this thesis provides a

proof of concept method and implementations for a MB direct image reconstruction via the well known QR decomposition procedure to obtain the least squares solution of an overdetermined system of equations.

Currently considered as an intractable problem, the direct solution of such linear system would provide an alternative to the current iterative search of the solution yielding to a faster image reconstruction. The contributions in this thesis are, therefore, oriented to the specializations and optimizations of the direct solution of the system for a matrix that models a CT, considering its definition, its sparse structure or the required operations to perform the QR-decomposition. Also, an image quality analysis of the reconstructed images and a comparison of its results against the prevailing reconstruction algorithm FBP and a state of the art model based iterative reconstruction algorithm MLEM, are included to confirm the image quality advantages of the MB reconstruction and, particularly, the advantages of the modeled processes that have been included.

1.4 Objectives

The main objective of this thesis is to provide an alternative to the iterative reconstruction of model based algorithms through the direct resolution of a linear system that models the CT device. This ambitious objective should be achieved through the development of the specific goals described in this section.

In order to define the linear system that will solve the image reconstruction problem, the computation of the volume intersection between the basic elements of the CT model must be addressed to compute the elements of the system matrix.

Considering the CT model, the size of the entire resulting matrix is intractable. This leads to the research in symmetries of this matrix to obtain a smaller system. Moreover, the obtained results must be compatible with the application of transformations to the matrix to reduce it to triangular form in order to exploit the advantages offered by the QR-decomposition.

The CT modeling process is subject to several parameters that will define the linear system for the image reconstruction. The effect of these parameters in the sensitivity of the system must be studied, in order to obtain knowledge of their impact on the reconstructed image quality.

Even achieving a reduction of the system matrix exploiting inherent symmetries in the CT model, the resulting matrix will have large dimensions and will be sparse. Moreover, it will be subject to massive numerical modifications during the QR-decomposition procedure. A sparse matrix scheme adapted to these requirements must be implemented; not only to assure the optimization of the required transformations for the QR-decomposition, but to gain full access to the structural information of the matrix (such as the number of non-zero elements in

a given row or the column index or the first and last non-zero elements) with an optimum performance.

Implementations of the QR-decomposition procedures must be up-to-date to the state of the art implementations of the most accepted scientific computation libraries, such as BLAS or LAPACK. However, they must make optimum use of the defined sparse matrix scheme previously mentioned.

In addition, due to the nature of the transformations performed by the QR-decomposition procedure in the sparse system matrix, fill-in will be produced. Taking advantage of the sparse matrix scheme implemented, an heuristic strategy to minimize the fill-in during the reduction to triangular form must be implemented. It must outperform the standard fill-in reduction procedures exploiting structural properties of the CT model matrix and provide insight in the fill-in problem, in order to allow further research in this subject.

A parallelization strategy must be provided for the most critical (time consuming) procedure of the new algorithm from the point of view of the user: the matrix vector product. Not necessary pursuing the optimum performance, but demonstrating a proof of concept implementation and providing insight in the parallelization of this procedure in order to be further investigated and improved.

The quality of the reconstructed images must be assessed against the industry standard, the FBP and a state of the art model based iterative reconstruction algorithm, the MLEM, in order to assure that image quality will not be lost.

A bound of the required time for the image reconstruction process as a function of the number of rotations needed to reduce the system matrix to triangular form is required to demonstrate the potential in reconstruction time reduction with further research in the development of better fill-in reduction strategies.

Finally, a proof of concept implementation of a CT image reconstruction algorithm must be obtained demonstrating correct image reconstruction and the potential of its advantages over the existing reconstruction methods.

Structure of the thesis

After this introductory reading about medical imaging, Chapter 2 covers the preliminaries regarding linear systems that will be needed along this thesis. The CT image reconstruction via a linear system and the computation of the CT model matrix are covered in Chapter 3. In Chapter 4, a sparse matrix allocation scheme is proposed and detailed. The different procedures of the QR decomposition are implemented in Chapter 5. Chapter 6 is devoted to the problem of fill-in during the QR decomposition and to the proposal of an alternative strategy for CT matrices. Some details on the parallelization of the QR decomposition are shown in Chapter 7. In Chapter 8, an image quality assessment of the proposed algorithms is performed and compared against the FBP and MLEM. Finally, the conclusions of the thesis are gathered in Chapter 9.

Chapter 2

Introduction to linear systems

The problem of solving a linear system is central to scientific computation, and it has been widely addressed in the literature. Among the best introductions to this problem are those of G. Golub and C. Van Loan [27] or D. S. Watkins [28]. This chapter summarizes both introductions, addressing only a few topics that will be needed along this thesis. A more detailed treatment of these topics can be found in [27, 28, 29, 30].

2.1 Systems of linear equations

The basic problem we will work with, is a system of m linear equations in n unknowns

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b_m, \end{aligned} \tag{2.1}$$

where the coefficients a_{ij} and b_i are given and we wish to find a $x_1, \ldots, x_n$ that satisfy the equations. For convenience and from now on, the System (2.1) will be written as a matrix equation

$$Ax = b, \tag{2.2}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}.$$

In our case, the coefficients (a_{ij} and b_i) are real numbers and we seek a real solution. Suppose $m = n$ (A is a square matrix) and A is nonsingular (A^{-1} exists). Then Equation (2.2) has a unique solution that can be obtained by

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}b \\ Ix &= A^{-1}b \\ x &= A^{-1}b, \end{aligned}$$

where I is the $n \times n$ identity matrix. However, this method implies the computation of A^{-1} which is computationally costly or unfeasible for most large problems.

2.1.1 Triangular systems

When A has a particular form, linear systems could be easy to solve. Let

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix},$$

where $a_{ij} = 0$ whenever $i > j$. That is, all elements of A below its main diagonal are zero. Then, provided that A is nonsingular ($a_{ii} \neq 0$ for $i = 1, \ldots, n$),

Equation (2.2) is particularly easy to solve by backward substitution as follows

$$\begin{aligned}
x_n &= \frac{b_n}{a_{nn}} \\
x_{n-1} &= \frac{b_{n-1} - a_{n-1n}x_n}{a_{n-1n-1}} \\
&\vdots \\
x_3 &= \frac{b_3 - a_{34}x_4 - a_{35}x_5 - \cdots - a_{3n}x_n}{a_{33}} \\
x_2 &= \frac{b_2 - a_{23}x_3 - a_{34}x_4 - \cdots - a_{2n}x_n}{a_{22}} \\
x_1 &= \frac{b_1 - a_{12}x_2 - a_{33}x_3 - \cdots - a_{1n}x_n}{a_{11}},
\end{aligned}$$

or equivalently

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}} \qquad \text{for } i = n, n-1, \ldots, 3, 2, 1. \tag{2.3}$$

2.1.2 Overdetermined systems

Suppose the previous assumption $m = n$ does not hold and instead, there are more equations than unknowns, *i.e.* $m > n$. Then, Equation (2.2) might not have an exact solution, especially if b comes from a collection of experimental measurements, which are subject to errors. In this case may be interesting to find an x such that the residual $r = Ax - b$ is small for some norm.

The strategy to find x in order to obtain a satisfactory r or the election of a particular norm are most problem dependent. In our case, we will wish to minimize r for the 2-norm, which is known as the least squares problem, considered in §2.2.

2.1.3 Sensitivity of linear systems

The sensitivity of a linear system is the effect that small perturbations in the coefficients have on the solution of a system. Suppose that we have a perturbed instance of the System (2.2)

$$A\hat{x} = b + \delta b \tag{2.4}$$

where δb is a small (relative to b) perturbation, $\hat{x}$ is the obtained solution for the perturbed system, and moreover, $\hat{x} = x + \delta x$ where δx is the perturbation in the solution.

We are interested in the definition of the condition number

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}, \tag{2.5}$$

where $\sigma_{max}(A)$ and $\sigma_{min}(A)$ are the largest and smallest singular values of A respectively, since we will make use of orthogonal transformations which preserve the 2-norm and obtain the condition of non-square matrices.

There is a useful bound for $\frac{\|\delta x\|}{\|x\|}$ in terms of $\frac{\|\delta b\|}{\|b\|}$ using the condition number $(\kappa(\cdot))$ of A

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq \kappa_2(A)\frac{\|\delta b\|_2}{\|b\|_2}. \tag{2.6}$$

Bound (2.6) indicates that the relative error in x can be $\kappa(A)$ times the relative error in b. Because it is desirable that if $\frac{\|\delta b\|}{\|b\|}$ is small, then $\frac{\|\delta x\|}{\|x\|}$ would be also small, it is desirable to have a system matrix with a low condition number (as nearest to 1 as possible).

2.2 The least squares problem

In presence of an overdetermined system (where $m > n$) Equation (2.2) might not have an exact solution. Let a_i be the i-th row of the system matrix A. Then, the residual r produced by a given solution x is

$$r_i = a_i x - b_i. \tag{2.7}$$

We will be interested in a solution that produces a small r, so we have to choose a norm to measure r. If we choose the 2-norm, then we have

$$\|r\|_2 = \sqrt{\sum_{i=1}^{m}(a_i x - b_i)^2}, \tag{2.8}$$

so we seek an x that minimizes $\|r\|_2$. The same x that minimizes Equation (2.8) will minimize

$$\|r\|_2^2 = \sum_{i=1}^{m}(a_i x - b_i)^2. \tag{2.9}$$

Thus, we seek an x that minimizes the sum of the squares of the residuals and for this reason the problem of minimizing $\|r\|_2$ is called *the least squares problem*. The solution to this problem can be obtained either with an iterative procedure (until some acceptable tolerance is achieved or minimizing a cost functional [22, 25]) or by the reduction of A to various canonical forms (including the triangular form).

Suppose $A \in \mathbb{R}^{m\times n}$ has full rank and we have a procedure to factor A into two matrices, say $R \in \mathbb{R}^{m\times n}$ and $Q \in \mathbb{R}^{m\times m}$. And moreover, suppose this procedure only involves orthogonal transformations (under which the 2-norm is preserved). Then we can pose a system equivalent to System (2.2)

$$QRx = b \tag{2.10}$$

and because only orthogonal transformations have been used, their residuals will be equivalent under the 2-norm

$$\|QRx - b\|_2 = \|Ax - b\|_2,$$

so we can seek the least squares solution of System (2.2) using the System (2.10). It would be most convenient if we can ask for Q to be an orthogonal matrix so that $Q^TQ = I$ were I is the identity matrix and for R to have a canonical form, say upper triangular. Provided that this factorization procedure also meets these requirements, we can pose an equivalent system to System (2.2)

$$Rx = Q^Tb \tag{2.11}$$

by multiplying both sides by Q^T and their residuals will still be equivalent under the 2-norm

$$\|Rx - Q^Tb\|_2 = \|Ax - b\|_2.$$

Moreover, as we asked for R to be an upper triangular matrix

$$R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix} \qquad \text{and if} \qquad Q^Tb = \begin{bmatrix} c \\ d \end{bmatrix} \begin{matrix} n \\ m-n \end{matrix}$$

we have

$$\|Ax - b\|_2^2 = \|Rx - Q^Tb\|_2^2 = \|R_1x - c\|_2^2 + \|d\|_2^2$$

and the least squares solution can be obtained solving the upper triangular system

$$R_1x = c \tag{2.12}$$

by a backward substitution process and its Residual (2.9) will be

$$\|r\|_2^2 = \|d\|_2^2.$$

Such a factorization of A into Q and R exists and is known as the QR decomposition. It is detailed in §5.1 along with its computer implementation.

2.2.1 Sensitivity of the least squares problem

The effect that small perturbations in the coefficients have on the solution is greater for the least squares problem than for linear systems.

Suppose δA and δb are small perturbations to the coefficients of System (2.2). Suppose that A and $A + \delta A$ are m-by-n matrices with $m \geq n$ and have full rank. Let

$$\begin{array}{lll} x & \text{minimize} & \|Ax - b\|_2, \qquad r = Ax - b, \\ \hat{x} & \text{minimize} & \|(A + \delta A)\hat{x} - (b + \delta b)\|_2, \qquad \hat{r} = (A + \delta A)\hat{x} - (b + \delta b), \text{ and} \end{array}$$

$\epsilon = \max(\frac{\|\delta A\|_2}{\|A\|_2}, \frac{\|\delta b\|_2}{\|b\|_2})$.

Then

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq \frac{\epsilon\kappa_2(A)}{1 - \epsilon\kappa_2(A)} \left(2 + (\kappa_2(A) + 1)\frac{\|r\|_2}{\|A\|_2\|x\|_2}\right) \tag{2.13}$$

and

$$\frac{\|\hat{r} - r\|_2}{\|r\|_2} \leq 1 + 2\epsilon\kappa_2(A) \tag{2.14}$$

are bounds for the perturbed solution and residual [31]. Bound (2.13) may be interpreted as follows. If $\|r\|_2$ is zero or very small then the effective condition number is $\approx 2\kappa_2(A)$, similar to Bound (2.6) for the perturbed solution of a linear system. However, as $\|r\|_2$ grows (but still far from $\|b\|_2$), the effective condition number becomes $\approx \kappa_2^2(A)$. Finally, if $\|r\|_2$ is near to $\|b\|_2$, then, changes in the solution due to perturbations become unbounded even if $\kappa_2(A)$ is small. When $\|r\|_2 = \|b\|_2$ (solution is zero), any small change that produces a slightly different $\hat{x}$ will yield an *infinitely* large relative change.

The accuracy of the least squares solution is governed by the residual. Most least squares problems will have an imposed A, but if A comes from a modeling process, this is a good reason to demand a model as compatible as possible with b (aside of well conditioned), in order to produce a *small residual* least squares problem.

The effective condition number of the least squares problem also depends on the solution strategy. Previously, System (2.2) has been transformed into the equivalent System (2.11) by the reduction of A to an upper triangular matrix R. It is shown that $\kappa_2(A) = \kappa_2(R)$ [32]. But there are other methods that produce larger condition numbers. For example, suppose that we seek the least squares solution through the equivalent system $(A^TA)x = A^Tb$. Then, the accuracy depends on $\kappa_2(A^TA) = \kappa_2^2(A)$. Therefore, even if $\|r\|_2$ is zero or very small, the effective condition number of the least squares problem will be $\approx \kappa_2^2(A)$.

Chapter 3

The CT image reconstruction as a linear system

After the introduction on some topics of linear systems, this chapter will be devoted to the definition of the CT image reconstruction problem. The description will be conducted through the principle of operation of the devices involved in a CT system. The definition of the CT system described, corresponds to the third generation scanners, that are the commonest nowadays.

Differing of most introductions, this chapter will focus in the computation of a system matrix instead of relating the CT measurement process to the Fourier slice theorem and deriving the filtered backprojection algorithm. A linear system of equations will be obtained by doing so, and some details on the particular system of the CT reconstruction will also be addressed, such as symmetries or the effect in $\kappa_2(A)$ produced by choices in the modeling of several CT components. The resolution of the linear system will be addressed later in §5. A more detailed treatment of the topics addressed in this chapter can be found in [7, 33].

The idea of using a linear system of equations for CT image reconstruction is the base of the ART [4] reconstruction method, used in the firstCT systems. However, the linear system derived in the following section adds several features (such as the *cone beam factor* [34]) which allow to classify it as a model-based reconstruction method. A tool to efficiently compute the volume intersection between two polyhedra is then needed. Although in the literature [34] more complicated and restrictive algorithms are available, in the following section an algorithm with a more general input (only the two polyhedra) will be addressed, and while it uses well known procedures in 3D object modeling, to the best of our knowledge, this is the first work that uses them to compute the volume intersections required to derive the linear system that solves the CT image reconstruction problem.

3.1 Modeling the CT

In this section, the linear system that solves the CT image reconstruction problem will be derived. In addition, the necessary tools to compute it will also be addressed, such as a generic volume intersection algorithm, in order to illustrate the computation of each a_{ij}, elements of matrix A, for a general third generation CT geometry.

3.1.1 X-ray source

An x-ray is an electromagnetic waveform. X-ray photons are produced by striking a target with high-speed electrons. The kinetic energy of electrons is transformed in electromagnetic radiation. X-ray energy is usually expressed in the unit of eV. 1 eV is the amount of kinetic energy with which an electron is accelerated across an electrical potential of 1 V.

X-ray energy depends on the amount of kinetic energy that is given off during interactions of accelerated electrons with the target. Since various interactions are possible, the produced x-ray photons of maximum energy will result of the accelerated electrons that suffered a total loss of their kinetic energy. But this is not always the case. Not all the kinetic energy of electrons is lost in all interactions. In fact, a continuous x-ray spectrum is produced (due to the bremsstrahlung process). For example, an x-ray source that accelerates electrons across an electrical potential of 120 kV, will produce x-ray photons with an energy of 120 keV or less.

The x-ray source is placed inside a housing with an aperture called port in order to prevent any x-ray photon to escape except for the port. After the port, a collimator is placed. The collimator adjusts the size and shape of the x-ray field that emerges from the housing port. Usually, the collimator is composed of two pairs of parallel-opposed blades that define a rectangular field.

3.1.2 X-ray detection

This section is devoted to illustrate the principle of operation of a flat panel x-ray detector, which is the commonest technology implemented in modern CT scanners. A flat panel detector is divided into individual detector elements arranged in a matrix. A detector element includes an electronics area and a sensitive area. The sensitive area will produce a charge proportional to the number of x-ray photons that arrives to the detector element (*i.e.* to the incident x-ray intensity). This charge will be stored into a capacitor in the electronics area. During an *exposure* time, the charge will be accumulated in the capacitor. When exposure is complete, a *readout* process starts; the capacitor will be discharged through a connection to outside of the active area of the panel. The voltage will be amplified and digitalized

to obtain the detector element measurement. Readout is usually organized in a sequential fashion (one row / column at a time), and when completed, a new exposure starts, and so on.

There are two main categories in the implementation of the sensitive area: direct and indirect detectors. Indirect detectors have a layer of a scintillator material to convert x-rays photons into light photons. A photosensitive area placed below the scintillator detects the generated light. Direct detectors have a layer of semiconductor material that produces electron-hole pairs which produces a charge in the detector in proportion to the incident x-ray intensity.

As a result of each pair of exposure and readout processes, a matrix of measured intensities is produced which is called a *view* or *projection*. A collection of views is gathered to feed the reconstruction algorithm.

3.1.3 X-ray attenuation

X-ray photons interact with matter. As a result of the x-ray photons interactions, some of the photons are absorbed or scattered when they pass through a material. The importance of these interactions is proportional to some material characteristics and dependent on the x-ray beam energy. Usually, this is condensed in a single parameter μ (as a function of the material and the x-ray beam energy) referred as the attenuation coefficient.

Suppose we have a monoenergetic, narrow collimated, and parallel x-ray beam that passes through a material with a uniform attenuation coefficient. The attenuation of the x-ray beam can be expressed by an exponential relationship known as the Beer–Lambert law:

$$I = I_0 e^{-\mu x}, \tag{3.1}$$

where I_0 and I are the original (entrance) and attenuated (exit) x-ray beam intensities respectively, μ is the attenuation coefficient of the material, and x is the length of the x-ray beam in the material.

Now suppose we have nonuniform object (with different attenuation coefficients). If we divide the object into n small regions so that the lengths of the beam in the regions are $x_1, x_2, \ldots, x_n$ and assume that if the largest x_i is sufficiently small, each region can be considered as a uniform object (with an attenuation coefficient μ_i), then, the total attenuation can be calculated with the repeated application of the Beer–Lambert law, yielding

$$I = I_0 e^{-\mu_1 x_1} e^{-\mu_2 x_2} e^{-\mu_3 x_3} \cdots e^{-\mu_n x_n} = I_0 e^{-\sum_{i=1}^{n} \mu_i x_i}. \tag{3.2}$$

An alternative form for Equation (3.2) can be obtained

$$p = -\ln\left(\frac{I}{I_0}\right) = \sum_{i=1}^{n} \mu_i x_i \tag{3.3}$$

by dividing both sides by I_0 and taking logarithms. p is known as a projection measurement and is (the logarithm of) the ratio between entrance and exit x-ray beam intensities. The Beer–Lambert law says that p represents the result of the integration of the attenuation coefficients (of the different previously defined regions) along the x-ray beam path. This is known as a *pencil beam*.

The Relationship (3.1) relies in the assumption that the x-ray beam is monoenergetic, which is, all of the x-ray photons have the same energy. However, in practice, x-ray beam is polyenergetic due to bremsstrahlung processes. Moreover, low-energy x-ray photons have a higher probability of being absorbed than high-energy x-ray photons (mainly due to the photoelectric process). As a result, the x-ray beam spectrum averages a higher energy as it passes through material. This phenomenon is known as *beam hardening*.

Considering that μ values range significantly with the beam energy and that for most materials showing the same characteristics, their μ values decrease as the beam energy increases (high-energy x-ray photons have a lower probability of being absorbed), Equation (3.3) will yield an overestimated value $\hat{p}$ for a polyenergetic x-ray beam. Two main approaches reduce the beam hardening phenomenon and restore the Relationship (3.1):

- Beam filtration [35, 36]: a thin layer of material placed between the beam source and the scanned object in order to absorb (*filter*) low-energy x-ray photons.
- Estimation process [37, 38, 39]: considering the total attenuation, $\hat{p}$, of a polyenergetic x-ray beam, a function f such that $f(\hat{p})$ is the total attenuation that a monoenergetic beam would have been suffered along the same path (p in Equation (3.3)).

The discrepancy between predicted monoenergetic beam attenuation and polyenergetic beam attenuation changes with the scanned object material, shape or x-ray source. The implementation of these strategies requires experimentation, calibration and testing. Therefore, an adjustment of parameters and / or filter must be made in case of the replacement of the x-ray source of the device, for example, and in a clinical environment, different protocols are defined for head or chest scans, for instance.

After the careful application of these techniques, polyenergetic x-ray beam attenuation measurements will be sufficiently close to monoenergetic beam attenuations to consider the Relationship (3.1), and therefore Equation (3.3), valid.

3.1.4 The CT reconstruction problem

Consider an object placed between an x-ray source and an x-ray detector, divide the space occupied by the object into n regions (a regular grid for instance), and

consider Equation (3.3). Then, a formulation of the CT reconstruction problem can be stated as follows: compute the attenuation coefficient of each of the n previously defined regions using the information provided by the detector as the attenuation of monoenergetic x-ray beams.

Lets start directly with a 3D model of the problem. Suppose we have a flat panel composed of d detector elements which have a square shape, are arranged as a matrix and located on the YZ plane ($x = 0$) such that the X axis crosses the plane at the center of the flat panel (the center of the flat panel is located at $\{0, 0, 0\}$). Let the collimator of the x-ray source has a rectangular area sufficiently small to be considered as a point (from now on, this point will be referred as the x-ray source) and located on the X axis at a distance D from the detector plane (at $\{D, 0, 0\}$). Then, we define a line of response (LoR) as a pyramid such as its base matches a detector and its apex matches the x-ray source (see Figure 3.1). This way, the x-ray field from the x-ray source to the flat panel detector is composed of d LoRs, each one with its base in one detector element. This shape of the x-ray field is known as *cone beam geometry*. Let $R_{det}(i)$ be the distance between the center of the detector element i and the x-ray source. Let the n regions in which the space occupied by the object is divided into, have a cubic shape and form a three dimensional regular grid. Then, we define a voxel as each one of these regions and (as before) μ is assumed to be constant over a voxel. Let $R_{vox}(j)$ be the distance between the center of the voxel j and the x-ray source.

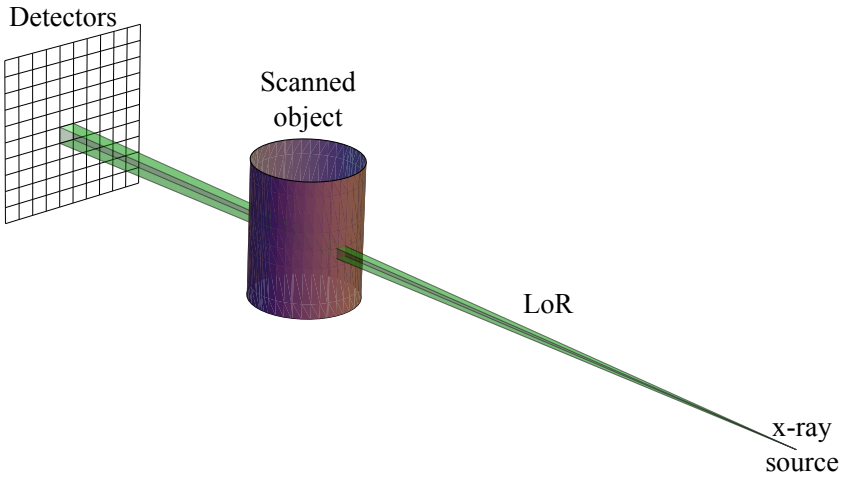


Figure 3.1: Illustration of a single LoR, from the x-ray source to one of the detector elements.

We need to extend the concept of pencil beam of Equation (3.3) to a volume integral. In order to do so, it is assumed that a LoR consists in infinitely many

beam paths, form the x-ray source to the detector that defines the LoR. Let

$$b_i = -\ln\left(\frac{I(i)}{I_0(i)}\right)$$

be (the logarithm of) the ratio between the intensity of the attenuated LoR that arrives to the detector element i ($I(i)$) and the intensity of the LoR that arrives to the detector element i when it is not attenuated ($I_0(i)$). Then

$$b_i = \sum_{j=1}^{n} \mu_j a_{ij} \tag{3.4}$$

represents the result of the integration of the attenuation coefficients of the voxels along the LoR volume following the Beer–Lambert law, where

$$a_{ij} = c(i,j)v_{ij},$$

and v_{ij} is the volume intersection between the i-th LoR and the j-th voxel ($v_{ij} = 0$ if the i-th LoR does not intersect the j-th voxel). In practice, a_{ij} is usually re-scaled by the voxel volume and the detector element area in order to control numerical error in the computation.

The term

$$c(i,j) = \frac{DR_{det}(i)}{\left(D - R_{vox}(j)\right)^2}$$

is referred as *the cone beam factor* [34]. The cone beam factor represents the divergence of the x-ray cone beam emanating from a point source. In fact, if we consider a parallel geometry where $D \to \infty$, then $R \to D$ and follows that in parallel beam geometry a_{ij} is simply proportional to v_{ij}.

From Equation (3.4) a system of linear equations can be derived in order to obtain the attenuation coefficients $\mu_1, \mu_2, \ldots, \mu_n$. d equations can be obtained from a single projection, and different projections can be obtained by rotating the x-ray source and detector along the scanned object. Let $b \in \mathbb{R}^{dP}$ be a collection of d detector element measurements consisting in P projections, such that b_i is a detector element measurement in one of the P projections. Let $A \in \mathbb{R}^{(dP)\times n}$ be a matrix such that each $a_{ij} \in A$ is the volume intersection of the i-th LoR (defined by a detector element in one projection) with the j-th voxel, corrected by the cone beam factor $c(i,j)$. Then, finding the solution to the linear system

$$A\mu = b \tag{3.5}$$

solves the CT reconstruction problem, provided that System (3.5) has a unique solution. §3.2 is devoted to the analysis of the matrix A.

3.1.5 Computation of the volume intersection

The computation of the volume intersection between a LoR (in an arbitrary projection) and a voxel is not straightforward. Usually, this procedure has been accomplished by a carefully decomposition of the voxel into volume-known regular polyhedra [34] or approximated computing the volume intersection with an angular invariant volume inscribed inside the voxel (as a cylinder [40], for example).

An algorithm will be defined below, consisting of well known procedures, for the computation of the exact volume intersection between a LoR and a voxel. The input of this algorithm are the voxel and LoR polyhedra, and it is a three step process: a *clipping*, a *capping*, and a *tessellation* stages.

First, the polyhedron intersection between the LoR and the voxel must be obtained. To do so, the four faces of the LoR (the pyramid base is not considered) are used to discard (clip) all voxel vertices outside its volume. Cases arise in which vertices have to be created in the intersection between the voxel edges and the plane, in order to complete the faces of the clipped polyhedron. Once the clipped polyhedron is contained inside the LoR, the possible holes created in it are covered (capped). The four faces of the LoR (and their normals) are used to detect the holes and to create new faces with their vertices in counterclockwise (CCW) order. This procedure is known as *clipping and capping* [41]. An implementation in C++ of this procedure is provided in Code 3.1 (construction of the original voxel and LoR polyhedra), Code 3.2 (edge wise voxel clipping), Code 3.3, (consideration for the edge formed by the last and first vertices and removal of invalid faces) and, Code 3.4 (capping stage). Also, an illustration of the cases when new vertices have to be created from intersections is shown in Figure 3.2 for clarification.

The last step is the actual volume computation. It is obtained from the tessellation of the previously computed intersection polyhedron into irregular tetrahedra, which volume, V, is

$$V = \frac{|\vec{a} \cdot (\vec{b} \times \vec{c})|}{6}, \tag{3.6}$$

where $\vec{a}$, $\vec{b}$, and $\vec{c}$ are three vectors from the same vertex to each one of the other three vertices respectively (see Figure 3.3).

The usual tessellation procedure of a polyhedron is illustrated in Figure 3.4. The process starts with the selection of one vertex called v_{apex}. Their adjacent vertices (those which share an edge with v_{apex}) are found. According to the definition of a polyhedron in Codes 3.1, 3.2, 3.3, and 3.4, the search for adjacent vertices should be performed in a face wise manner. For each face, if v_{apex} is present, its predecessor and successor in the CCW ordering are considered their adjacent. These two adjacent vertices are treated not as a pair of vertices but as a line segment. Once the adjacent vertices search is complete, a collection of line segments

```
Polyhedron Geometry3D::polyhedron_intersection(const Pixel3D& voxel, const LoR3D& LoR){

  // obtain the normals of the LoR planes
  Point3D normal_plane_LoR[4];
  normal_plane_LoR[0] = Geometry3D::normal_plane(LoR.v1, LoR.v2, LoR.v4);
  normal_plane_LoR[1] = Geometry3D::normal_plane(LoR.v3, LoR.v0, LoR.v4);
  normal_plane_LoR[2] = Geometry3D::normal_plane(LoR.v0, LoR.v1, LoR.v4);
  normal_plane_LoR[3] = Geometry3D::normal_plane(LoR.v2, LoR.v3, LoR.v4);

  // form a Polyhedron from the voxel vertices (assuming that is a cube)
  Polyhedron polyhedron_voxel, polyhedron_tmp;
  polyhedron_voxel.clear();
  // the vertex in each face are ordered conuter clock wise
  Polyhedron_voxel.add_face(voxel.v0, voxel.v1, voxel.v2, voxel.v3);
  Polyhedron_voxel.add_face(voxel.v1, voxel.v5, voxel.v6, voxel.v2);
  Polyhedron_voxel.add_face(voxel.v3, voxel.v2, voxel.v6, voxel.v7);
  Polyhedron_voxel.add_face(voxel.v0, voxel.v1, voxel.v5, voxel.v4);
  Polyhedron_voxel.add_face(voxel.v4, voxel.v0, voxel.v3, voxel.v7);
  Polyhedron_voxel.add_face(voxel.v7, voxel.v6, voxel.v5, voxel.v4);

  // new (clipped) faces will be computed to replace the originals
  Face3D face_tmp, face_new;
  // in each step, two vertices and its intersection in the plane will be computed
  Point3D vA, vB, intersection;
  // also, its distances to the plane will be of use
  double DvA, DvB;

  // [unfinished] code continues
```

Code 3.1: C++ definition of a method to compute the polyhedron that results of the intersection of a voxel (a cube) and a LoR (a pyramid). Source code is incomplete and continues in Code 3.2. A convention has been followed for the definition of `v1`, `v2`, ..., `v`n (voxel and LoR vertices), such that all the faces defined in this source code have their vertices in CCW order. The `Point3D` class contains three attributes to define a three dimensional point, and its `operator=` has been overloaded to copy the three attributes. The `Polyhedron` class consists of a list of faces, where a face consists of a list of vertices in CCW order. And the `normal_plane()` static method computes the normal of a plane defined by three points, provided that its parameters are supplied in CCW order.

```
// for each LoR plane normal
for(int p = 0; p < 4; ++p){
  // reset polyhedron that is going to be built
  polyhedron_tmp.clear();

  // for each face 'c' of the polyhedron
  for(int c = 0; c < polyhedron_voxel.get_size(); ++c){
    // retrieve a face for clipping
    face_tmp = polyhedron_voxel.get_face(static_cast<unsigned int>(c));
    face_new.clear();

    // check that current face has at least two vertices
    if(face_tmp.get_size() > 2){
      // for each vertex of the current face ...
      for(int v_idx = 0; v_idx < (face_tmp.get_size()-1); ++v_idx){

        // ... select two consecutive vertices ...
        // (vertices were ordered counter clock wise, so they form an edge of the
            face)
        vA = face_tmp.get_vertex(static_cast<unsigned int>(v_idx));
        vB = face_tmp.get_vertex(static_cast<unsigned int>(v_idx + 1));

        // ... compute its ''distance'' to the current clipping plane of the LoR ...
        DvA = Geometry3D::distance_point_plane(vA, LoR.v4, normal_plane_LoR[p]);
        //                                point ^ | ^ point on the plane | ^ normal of
             the plane
        DvB = Geometry3D::distance_point_plane(vB, LoR.v4, normal_plane_LoR[p]);
        // this is not exactly a distance, since it has a sign:
        // negative -> below the plane, positive -> above the plane

        // check if the plane intersects with the edge
        if((DvA > 0) && (DvB < 0)){ // if vA is above and vB is below
          intersection =
            Geometry3D::intersection_line_plane(vB, vA, LoR.v4, normal_plane_LoR[p]);
          face_new.add_vertex(intersection);
        }

        if((DvA <= 0) && (DvB <= 0)){ // if vA and vB are below
          face_new.add_vertex(vA);
        }

        if((DvA < 0) && (DvB > 0)){ // if vA is below and vB is above
          face_new.add_vertex(vA);
          intersection =
            Geometry3D::intersection_line_plane(vB, vA, LoR.v4, normal_plane_LoR[p]);
          face_new.add_vertex(intersection);
        }

        if((DvA == 0) && (DvB > 0)){ // if vA is the actual intersection and vB is
            above
          face_new.add_vertex(vA);
        }
      } // [for each vertex] all vertices except one have been processed

      // [unfinished] code continues
```

Code 3.2: C++ definition of a method to compute the polyhedron that results of the intersection of a voxel (a cube) and a LoR (a pyramid). Source code is the continuation of Code 3.1 and continues in Code 3.3. A new polyhedron is built using the faces of the voxel clipped by all the LoR planes. The `distance_point_plane()` static method computes the *distance* (with a sign) of a point, P, to a plane. The plane is defined by a point, P^{plane}, and a normal, N^{plane}. This method computes $N_x^{plane}(P_x - P_x^{plane}) + N_y^{plane}(P_y - P_y^{plane}) + N_z^{plane}(P_z - P_z^{plane})$, and the sign shows if P is above or below the plane. `if` cases are illustrated in Figure 3.2.

```
      // the edge vertex(n-1) -> vertex(0) has not been processed yet ...
      vA = face_tmp.get_vertex(static_cast<unsigned int>(face_tmp.get_size()-1));
      vB = face_tmp.get_vertex(0);

      // ... compute its ''distance'' to the current clipping plane of the LoR ...
      DvA = Geometry3D::distance_point_plane(vA, LoR.v4, normal_plane_LoR[p]);
      DvB = Geometry3D::distance_point_plane(vB, LoR.v4, normal_plane_LoR[p]);

      // check if the plane intersects with the edge
      if((DvA > 0) && (DvB < 0)){ // if vA is above and vB is below
        intersection =
          Geometry3D::intersection_line_plane(vB, vA, LoR.v4, normal_plane_LoR[p]);
        face_new.add_vertex(intersection);
      }

      if((DvA <= 0) && (DvB <= 0)){ // if vA and vB are below
        face_new.add_vertex(vA);
      }

      if((DvA < 0) && (DvB > 0)){ // if vA is below and vB is above
        face_new.add_vertex(vA);
        intersection =
          Geometry3D::intersection_line_plane(vB, vA, LoR.v4, normal_plane_LoR[p]);
        face_new.add_vertex(intersection);
      }

      if((DvA == 0) && (DvB > 0)){ // if vA is the actual intersection and vB is
          above
        face_new.add_vertex(vA);
      }
    } // [if(face_tmp.get_size() > 2)] the entire polyhedron face has been clipped

    // the clipped face has been stored in 'face_new'
    // if it has not been entirely clipped ...
    if(face_new.get_size() > 0){
      // append it to the clipped polyhedron
      polyhedron_tmp.add_face(face_new);
    }

  } // [for each face]

  // all faces of the original polyhedron have been clipped by the first LoR plane
  // and are stored in 'polyhedron_tmp'
  // so the original polyhedron is overwritten in order to continue clipping
  // the partially clipped polyhedron
  polyhedron_voxel = polyhedron_tmp;
} // [for each LoR plane]

// removes invalid faces:
// removes repeated vertices and faces that have only two (or less) vertices
polyhedron_voxel.remove_invalid_faces();

// [unfinished] code continues
```

Code 3.3: C++ definition of a method to compute the polyhedron that results of the intersection of a voxel (a cube) and a LoR (a pyramid). Source code is the continuation of Code 3.2 and continues in Code 3.4. This code processes the last edge of a face, formed by vertices $n-1$ and 0. The original polyhedron is overwritten before continuing with the LoR plane loop in order to continue clipping the partially clipped polyhedron. Finally, all faces that remained with two or less vertices are removed and the clipping procedure is finished.

```
  // clipping stage is finished: now capping starts
  bool face_needed_Q;
  face_new.clear();
  face_tmp.clear();

  // if a face is needed it will be on one LoR plane
  // for each LoR plane
  for(int p = 0; p < 4; ++p){

    // suppose that a new face is needed ...
    face_needed_Q = true;
    face_new.clear();

    // for each face 'c' of the polyhedron (and provided that a new face is still
        needed) ...
    for(int c = 0; (c < polyhedron_voxel.get_size()) && face_needed_Q; ++c){

      // retrieve the current face
      face_tmp = polyhedron_voxel.get_face(static_cast<unsigned int>(c));
      face_needed_Q = false;
      // if all vertices of the current face are on the LoR plane, capping is not
          needed
      // check that not all vertices of the current face are on the current LoR plane
      // for each vertex of the current face ...
      for(int i = 0; i < face_tmp.get_size(); ++i){

        // retrieve a vertex and compute its ''distance'' to the LoR plane
        vA = face_tmp.get_vertex(static_cast<unsigned int>(i));
        DvA = Geometry3D::distance_point_plane(vA, LoR.v4, normal_plane_LoR[p]);

        // only one vertex outside the LoR plane will be sufficiente to confirm
        // that a new face is still needed
        face_needed_Q = face_needed_Q || (DvA != 0);

        // in case that a new face is needed, their vertices will be those that
        // are in the LoR plane
        if(DvA == 0){
          face_new.add_vertex(vA);
        }
      } // an entire face has been checked
      // if a new face is not needed, code will continue for the next LoR plane
    } // all faces have been checked

    // if a new face is not needed, code will continue for the next LoR plane
    if(face_needed_Q && (polyhedron_voxel.get_size() > 0)){
      // if a face is needed, it is composed of all vertices in the LoR plane
      // remove repeated vertices and sort them in a counter clock wise order
      // (according to the LoR plane) ...
      face_new.fix_face(normal_plano_LoR[p]);
      // ... and if after the fix is useful, append it to the polyhedron
      if(face_new.get_size() > 0){
        polyhedron_voxel.add_face(face_new);
      }
    }
  } // [for each LoR plane]
  // return the intersection
  return polyhedron_voxel;
}
```

Code 3.4: C++ definition of a method to compute the polyhedron that results of the intersection of a voxel (a cube) and a LoR (a pyramid). Source code is the continuation of Code 3.3. This code implements the capping procedure and uses the LoR planes to build the necessary faces. The vertex of a new face will be in a LoR plane that previously clipped the polyhedron. A check to avoid duplicate faces is also implemented with the `face_needed_Q` variable. The final result is a polyhedron intersection between the voxel and LoR given.

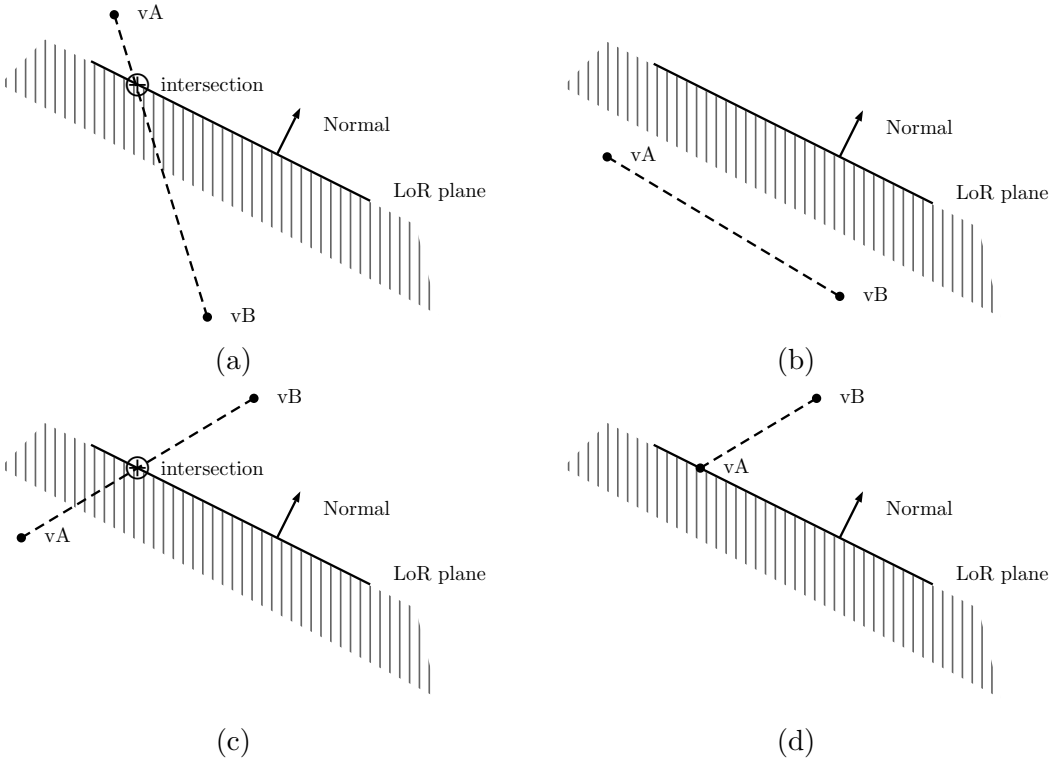


Figure 3.2: Illustration of the `if` cases that appear in Codes 3.2 and 3.3, where `vA` and `vB` are the initial and final vertices of an edge respectively, and the dashed area represents the region of space *inside* the LoR. `if` cases select `vA` and / or an intersection point to be vertices of the new face that is clipped by the LoR plane. When `vA` is above and `vB` is below the plane (a), only the edge intersection should be part of the new face. When `vA` and `vB` are below the plane (b), only `vA` should be considered for the new face. When `vA` is below and `vB` is above the plane (c), `vA` and the edge intersection should be part of the new face. When `vA` is the actual intersection (d), `vA` should be a vertex of the new face.

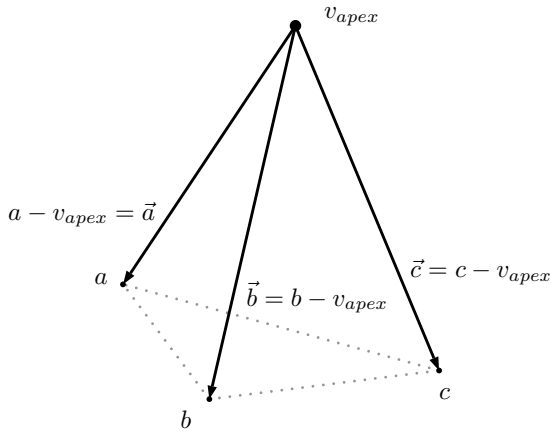


Figure 3.3: Illustration of the vertex naming used for the computation of the volume of a tetrahedron. v_{apex} is used to define the three vectors that will be used to obtain the volume, according to Equation (3.6).

is obtained. These line segments are then ordered such that the last point of a segment matches the first point of the next, until a closed polyline is obtained.

For now, suppose that only three adjacent vertices, a, b, and c, are found (a more general case will be addressed later). Then, the obtained polyline is converted into a new face (by a duplicate vertex removal and assurance of CCW ordering). v_{apex} is removed from the polyhedron and the new face is appended to it to cap the produced hole, and by doing so, a properly defined polyhedron is obtained after the v_{apex} removal. The polyhedron volume loss is equal to the tetrahedron defined by v_{apex}, a, b, and c, which volume is computed with Equation (3.6) (see Figure 3.3). This procedure is repeated until no vertices remain. The aggregation of the volumes of the obtained tetrahedra yields the volume of the original polyhedron.

The general case can be solved with little modifications to the previously detailed base case if some restrictions are imposed. First, the polyhedron that is going to be tessellated must be convex, *i.e.*, a line connecting any two (noncoplanar) points on its surface always lies in the interior of the polyhedron. And second, after a vertex removal (during the tessellation process) the resulting polyhedron must remain convex. It is useful to define a convex polyhedron as the set of solutions $x \in \mathbb{R}^3$ to a finite system of linear inequalities

$$Mx \leq b,$$

where $M \in \mathbb{R}^{k\times 3}$ and $b \in \mathbb{R}^k$. As for the CT problem, a LoR (a pyramid) and a voxel (a cube) are convex polyhedra, let M_{voxel}, b_{voxel}, M_{LoR}, and b_{LoR} define a voxel and a LoR respectively. Its polyhedron intersection is the set of solutions x

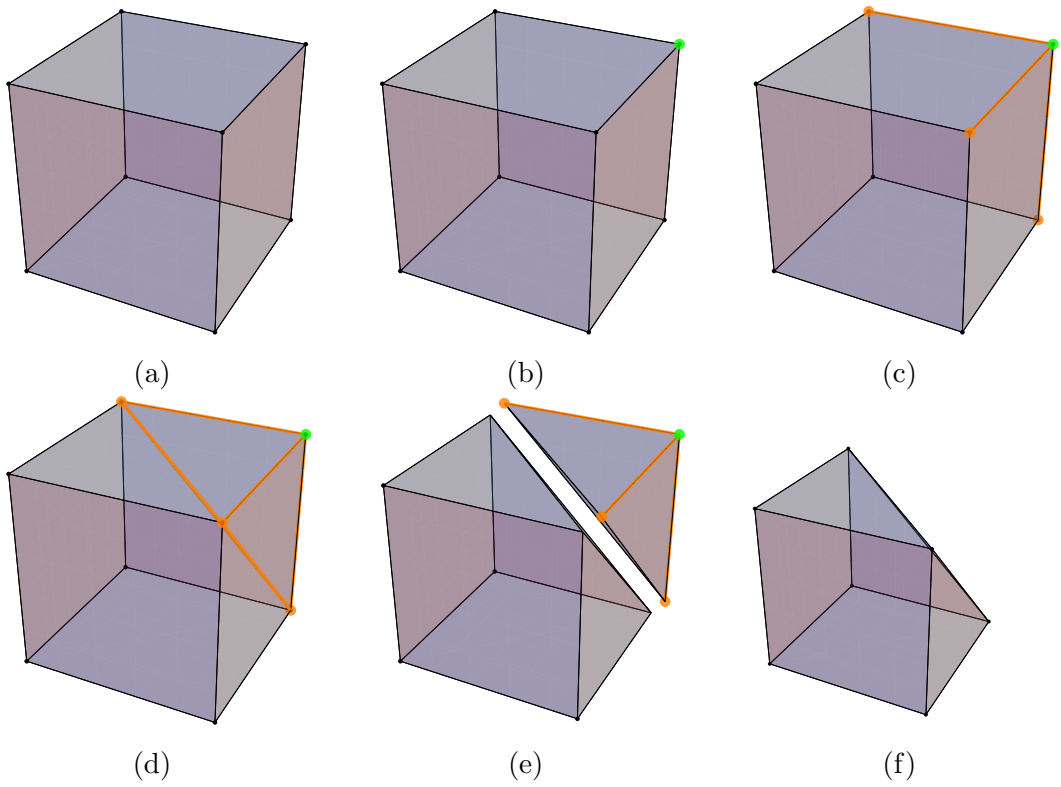


Figure 3.4: Illustration of the tessellation procedure. For simplicity, a cube has been used in this illustration (a). First, a vertex v_{apex} (green vertex in the illustration) is selected (b). Also, its adjacent vertices are found (c). Its adjacent vertices are then stored in CCW ordering and an edge is defined between two consecutive vertices (d). This leads to two separated polyhedra (e) a tetrahedron and the rest of the cube (a gap has been place between them for clarity). The volume of the tetrahedron is computed (as shown in Figure 3.3) and v_{apex} is deleted from the cube (f). The removal of the tetrahedron produces a hole in the remaining polyhedron, so an additional face consisting of the v_{apex} adjacent vertices is added to cap the hole. The process is repeated until no vertices remain.

to the system

$$\begin{bmatrix} M_{LoR} \\ M_{voxel} \end{bmatrix} x \leq \begin{bmatrix} b_{LoR} \\ b_{voxel} \end{bmatrix},$$

and, therefore, the first condition always holds.

The adjacent vertices of the selected v_{apex} during the tessellation must be coplanar, in order to guarantee the second condition. This can be seen as an imposition in the v_{apex} removal to reduce it to the case previously detailed, where the polyhedron was clipped by one of the LoR planes. In fact, if the polyhedron under tessellation is defined by M_{poly} and b_{poly}, and the clipping plane is defined by M_{clip} and b_{clip}, the clipped polyhedron will be the set of solutions x to the system

$$\begin{bmatrix} M_{poly} \\ M_{clip} \end{bmatrix} x \leq \begin{bmatrix} b_{poly} \\ b_{clip} \end{bmatrix},$$

and, therefore, provided that the adjacent vertices of the selected v_{apex} are coplanar, the second condition will hold.

The implementation of the criteria to choose the v_{apex} vertex is derived from the second condition. The vertex with the minimum adjacent vertices will be checked to be v_{apex}. Hopefully, it will have three adjacent vertices. If not, its adjacent vertices will be coplanar. If not, next vertex with less adjacent vertices is checked to be v_{apex}. If no vertex is a valid candidate, the code stops. We have been unable to find a case in which no vertex was eligible to be v_{apex}. However, we also have been unable to provide a general proof which assures the existence of an eligible vertex in each phase of the tessellation.

The last case remaining is when v_{apex} has more than three adjacent vertices that are coplanar. The capping stage needs no modifications since adjacent vertices are coplanar, the polyline construction and the face conversion are independent of the number of line segments involved. However, the polyhedron formed with the adjacent vertices is not a tetrahedron. This can be reduced to the base case due to the CCW ordering of the face formed by the vertices adjacent to v_{apex} splitting the clipped volume into various tetrahedra. Namely, let v_0, v_1, v_2, v_3, ..., v_{n-2}, v_{n-1} be the n adjacent vertices of an eligible v_{apex}. Then, the clipped volume of the polyhedron under tessellation can be computed with the repeated application of the Equation (3.6), naming $\{a, b, c\}$ consecutively the vertices $\{v_0, v_1, v_2\}$, $\{v_0, v_2, v_3\}$, and so on. Generally, if $V(i, j, k)$ equals Equation (3.6) being $a = v_i$, $b = v_j$, $c = v_k$, then the total volume is

$$\sum_{i=1}^{n-2} V(0, i, i+1).$$

In Figure 3.5 the bases of the obtained tetrahedra are shaded in different textures for clarification.

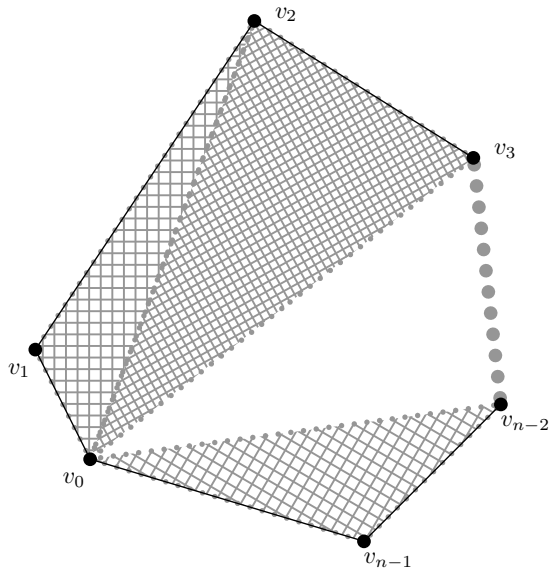


Figure 3.5: Bases of the different tetrahedra (in different textures) formed with v_{apex} (not present in the figure) and vertices $\{v_0, v_1, v_2\}$, $\{v_0, v_2, v_3\}$, ..., $\{v_0, v_{n-2}, v_{n-1}\}$. Face edges are represented with a thin black line joining the vertices.

It should be noted that the enumeration of these tetrahedra can be performed blindly, following the CCW order, because of the face formed by the adjacent vertices to an eligible v_{apex} is convex, *i.e.*, a line connecting any two points on its edges always lies in the interior of the face. This is due to the fact that this face is also defined by the clipping of the (convex) polyhedron under tessellation with a given plane. Volume overestimation could occur otherwise (see Figure 3.6). However, imposed restrictions (a convex polyhedron and coplanar adjacent vertices) guarantee the construction of a convex face.

This closes the definition of a procedure to compute the exact volume of the intersection of a LoR and a voxel that only requires the LoR and a voxel polyhedra with their faces stored in CCW order. Polyhedra which can be trivially defined when coming from a regular grid as the voxellation of the space occupied by the object under a CT scan, or the matrix of detector elements that form a CT flat panel. However, this algorithm allows more complicated voxellations that could be of some advantage, being the only problem for the exact volume computation, the definition of the voxel and LoR polyhedra with their faces stored in CCW order.

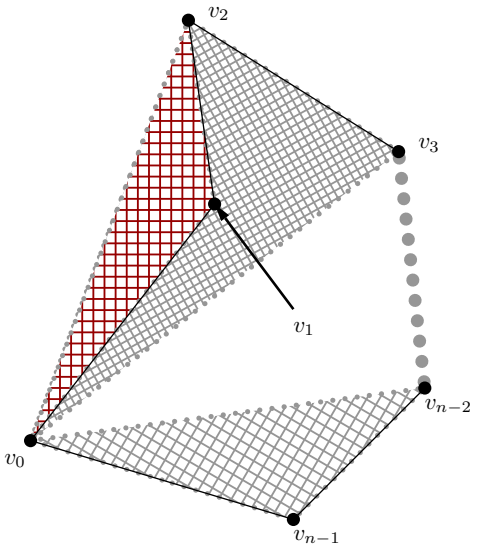


Figure 3.6: Illustration of volume overestimation if adjacent vertices form a non-convex face. Red texture represent the base of a tetrahedron formed with v_{apex} (not present in the figure), that will be accounted twice for the total volume while it lies outside of the face. Face edges are represented with a thin black line joining the vertices.

3.2 Properties of the CT linear system

In §3.1, all the necessary tools to obtain the System (3.5)

$$A\mu = b$$

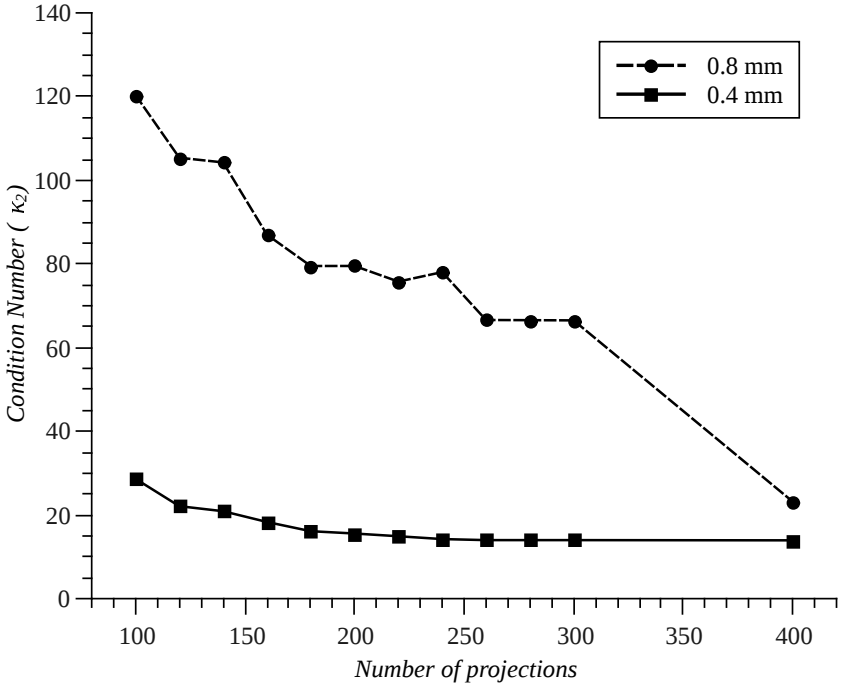
that solves the CT reconstruction problem have been detailed, where the space occupied by the object under scan is voxellated in n regions, d detector elements are considered in the detector flat panel, P views of the object are taken, $A \in \mathbb{R}^{(dP)\times n}$, $\mu \in \mathbb{R}^n$, and $b \in \mathbb{R}^{dP}$. It is useful to summarize several properties of A in order to specify the linear system to solve.

The most general definition of A is that in each element $a_{ij} \in A$ is quantified the contribution of the voxel j to the attenuation measured in a detector element d_k during a projection P_v, such that $d_k P_v = i$. It is easily seen that not all of the n voxels will contribute to the measured attenuation. A is then a sparse matrix. We can roughly estimate the sparsity of A by picturing how many voxels contribute to a single LoR. Suppose the length of an edge of a voxel and an edge of a detector element are approximately the same. Then, if the voxellation is performed as a regular cubic grid ($n^{1/3} \cdot n^{1/3} \cdot n^{1/3} = n$), we can assume that each LoR will transverse $\mathcal{O}(n^{1/3})$ out of n voxels.

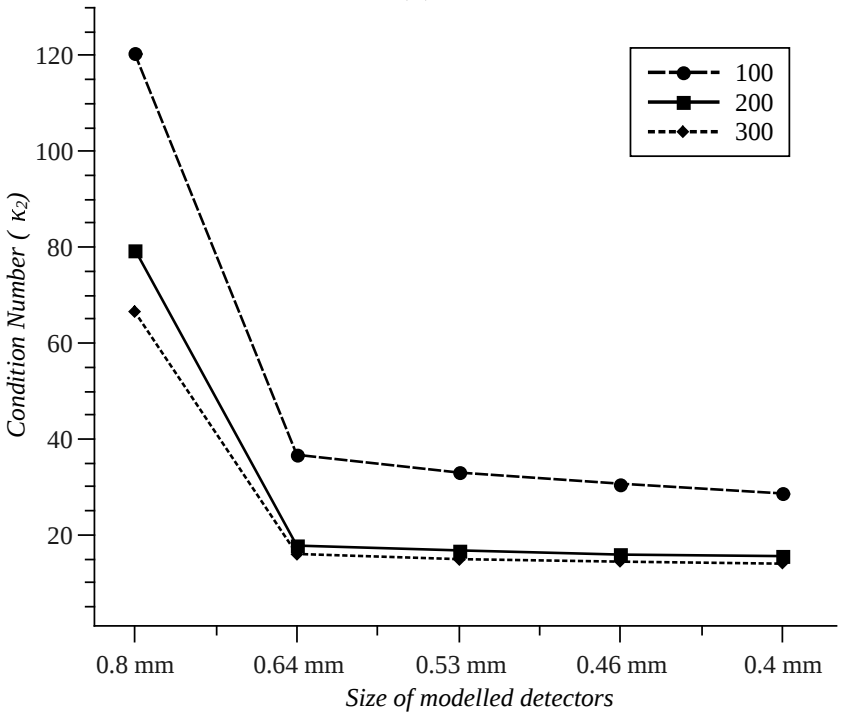
Intuitively, as long as sufficient projections are taken, there will be enough information to uniquely solve the μ distribution of the object. Often, sufficient projections imply $(dP) > n$ (see [7]) and System (3.5) becomes overdetermined. Moreover, we seek to to uniquely solve μ, so we want to define a set of projections that does not produce mainly linearly dependent rows in A. In other words, by enough information we are referring to A having full rank.

Considering the same voxellation of the scanned object, there are different choices of d and P that produce a sparse matrix A which has full rank. However, the election of d and P affects to $\kappa_2(A)$, and therefore, it has an effect on the solution of the overdetermined system. Suppose we have a flat panel consisting of d detector elements of a given size. We could model a flat panel with a *different* number of detector elements by performing a rebinning process in the measurement (in order to ease the computational cost, for example) and obtain a modeled flat panel with $d/4$ detector elements of an edge size twice the original. Similarly, we can ask the CT system for obtaining certain number of views.

We modeled the Albira μCT system [42] with different d and P for the same voxellation of the scanned object (1.28 mm cubic grid), and computed the condition number of the resulting A, in order to estimate how the election of these parameters influence the solution. Figure 3.7 shows $\kappa_2(A)$ for different configurations. In Figure 3.7 (a) is reflected that increasing the number of projections from 100 to 400 with detectors of 0.8 mm produces a condition number comparable to that obtained with the reduction of the detector size from 0.8 mm to 0.4 mm with 100 projections. Increasing the number of projections and decreasing the detector size produce a growth on the number of equations of the model. As the number of equations grows the condition number tends to stabilize. Although, the condition number is reduced by the number of projections and the detector size, it decreases faster to its limit as the detector size is reduced. Similar results were obtained with voxel edges from 2.13 mm to 0.91 mm and varying d and P proportionally. These results have been published in the journal *IEEE Transactions on Nuclear Science* [43].



(a)



(b)

Figure 3.7: Variation of the condition number of the system matrix as a function of the number of projections (a) and the modeled size of detector elements (b). System models have been configured with voxels of 1.28 mm, modeled detectors of 0.8 mm and 0.4 mm, and 100, 200 and 300 projections.

3.3 Exploiting symmetries

Finding symmetries in the contribution of the voxels to the measured attenuations could reduce the computational burden of the reconstruction process. It could lead to the computation of a *small* matrix, which properly rotated or reordered, would express the entire A, avoiding allocation space and repeated calculations.

However, the reduction of A to a triangular form will break some symmetries, because of the transformations applied. An advantageous symmetry would be one such that a transformation in one symmetry would be reflected into the rest, taking the entire A one step closer to the triangular form.

Suppose that only one half of the flat panel (along the axial coordinate) is considered. In Figure 3.8 is depicted the considered x-ray field of various projections. Intuitively, it is quickly seen that only half of the scanned object will be reconstructed. Nevertheless, (in a scatter free situation) it brings the attention to the fact that no voxel of the other half of the object has contributed to the measured attenuation in any projection. In other words, the same columns (half of the total) contain zeros in the half of the matrix rows, regardless of the projection. This can be expressed as

$$PA = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix},$$

were A_1 and A_2 are two matrices with half the rows and columns of A, and P is a permutation matrix. Also, this implies that System (3.5) is composed of two independent linear systems

$$A_1\mu_1 = b_1$$
$$A_2\mu_2 = b_2,$$

where

$$\mu' = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \qquad b' = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

are μ and b after applying some reordering.

In addition, A_2 is equal to A_1 under some permutation, due to the axial symmetry of the CT geometry. In Figure 3.9 is shown the exact row correspondence of the flat panel illustrated in Figure 3.8. Of course, the permutation depends on the flat panel logical ordering, but to clarify the symmetry, let be $i_0, i_2, \ldots, i_9$ a *detector row*. Then, $A_1\mu_1 = b_1$, where A_1 and b_1 contain only *white detector rows* and the *white* measurements of the detector elements in a proper order respectively, and μ_1 is a half of the object. Moreover, $A_1\mu_2 = b_2$, if b_2 contains only the *green* measurements of the detector elements in a reverse *detector row* order, and μ_2 is the other half of the object with its slices reversed.

The symmetry exploit allows to solve the entire system by the reduction to triangular form of a half rows, half columns matrix. The additional cost is the

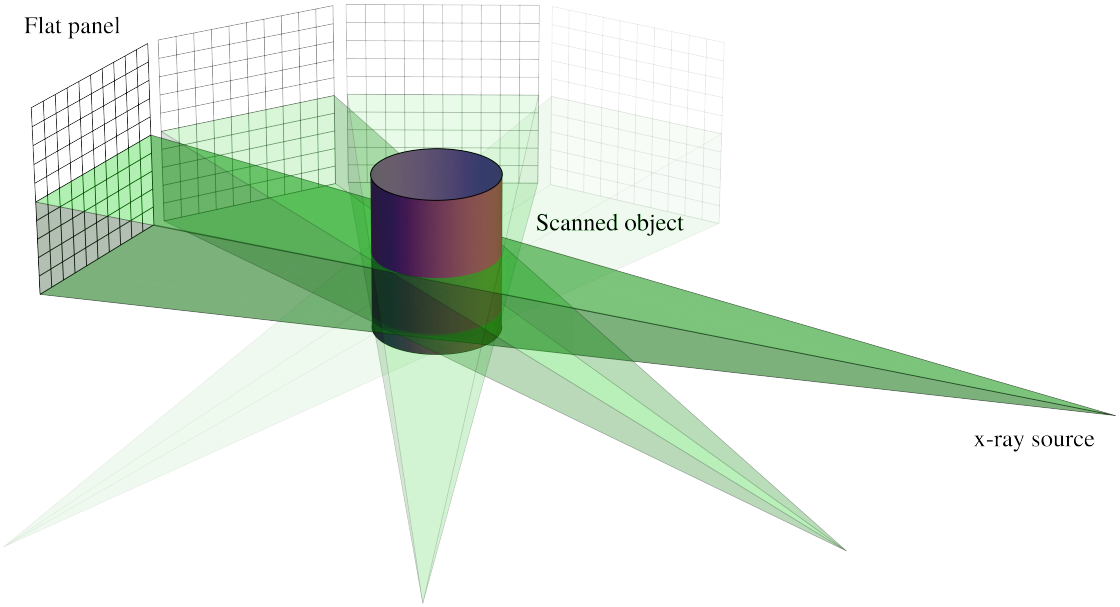


Figure 3.8: Illustration of one half of the x-ray field (in green) emerging form the source, going through the object, and arriving to the flat panel. Four superimposed projections are shown. This figure shows that there is no LoR arriving to the *lower* part of the flat panel that transverses the *upper* part of the object.

resolution of the same system for two right hand sides (applying a reordering to b_2), but it is much lower than the cost of the reduction to triangular form of a matrix four times larger.

Other matrix symmetries [44] are detailed in the literature, based on the rotation of a submatrix to compute the entire A. However, to the best of our knowledge, this is the first work that describes a symmetry that expresses A as a block matrix and therefore, allows methods like QR-decomposition to take advantage of it.

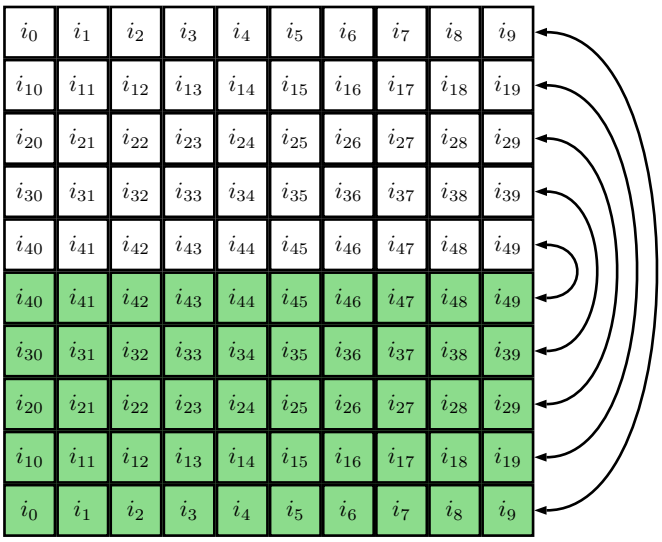


Figure 3.9: Row correspondence of the flat panel illustrated in Figure 3.8.

Chapter 4

Implementation of data structures

The amount of space needed to store a sparse matrix in full-length arrays is often prohibitive. Therefore, it is required a compact representation for sparse matrices. The aim of this chapter is to define a suitable data structure for holding sparse matrices. There is no one best data structure; the suitability of a data structure is highly related to the operations to be performed on the matrices.

Matrix operations needed in the QR decomposition are element wise (access, deletion and insertion), due to zeroing positions and fill-in during the matrix reduction to triangular form. The storage scheme must meet a space complexity of $\mathcal{O}(n_0)$, where n_0 is the number of non-zero values in the sparse matrix. It would be desirable to have at most $\mathcal{O}(\log n)$ time complexity for element access, insertion and deletion, where n is the average number of non-zero values in the sparse matrix rows.

4.1 Sparse matrix data structures

There are commonly used schemes for storing sparse matrices that meet the requirements for some of the operations and are simple to implement. The simplest scheme for storing a sparse matrix A is the coordinate list (COO) [29]. It consists in an unordered set of elements $\{a_{ij}, i, j\},\ \forall a_{i,j} \neq 0 \in A$, where a_{ij} is the value of A in row i and column j.

This is commonly done by maintaining three arrays, one for a_{ij} values (`VAL`), one for i values (`IRN`) and one for j values (`JCN`). Given a subscript e, $\texttt{VAL}_e$, $\texttt{IRN}_e$ and $\texttt{JCN}_e$ contain its respective information of a non-zero element of A (see figures 4.1a and 4.1b). The complexity analysis for our required operations is $\mathcal{O}(n_0)$ for access, $\mathcal{O}(1)$ for insertion, $\mathcal{O}(n_0)$ for deletion and $\mathcal{O}(n_0)$ of required space. As the easiest scheme to implement, it has the drawback of searching over all the non-zero elements of the matrix.

A refinement of the COO scheme is the compressed sparse row (CSR) [29]. This format is probably the most popular for storing general sparse matrices. It is equivalent to the COO but the row information is not stored explicitly; called compressed sparse column (CSC) [29] if the column information is not stored explicitly.

This is commonly done by maintaining three arrays, one for a_{ij} values (`VAL`), one for j values (`JCN`) and one for the subscript of `VAL` and `JCN` in which the i-th row starts (`IRS`). This requires that the information of non-zero elements in `VAL` and `JCN` were ordered by rows consecutively in ascending order (see figures 4.1a, 4.1c and 4.1d). The complexity analysis for our required operations is $\mathcal{O}(n)$ for access, $\mathcal{O}(n_0)$ for insertion, $\mathcal{O}(n_0)$ for deletion and $\mathcal{O}(n_0)$ of required space. As a refinement of COO, the ordering of the elements by rows (or columns), eases the access cost but it has the drawback of inserting and deleting elements.

There are other commonly used schemes for storing sparse matrices that can be found in [45, 30, 46] along with a more detailed explanation of COO, CSR and CSC. In the following section we present and define an easy to implement scheme that meet the requirements proposed: $\mathcal{O}(\log n)$ for element wise access, insertion, and deletion operations and a space complexity of $\mathcal{O}(n_0)$.

$$A = \begin{pmatrix} 0 & 3 & 0 & 0 \\ 4 & 1 & 0 & 0 \\ 0 & 5 & 9 & 2 \\ 6 & 0 & 0 & 5 \end{pmatrix}$$

(a) Example of a sparse matrix.

subscript	0	1	2	3	4	5	6	7
IRN	0	1	1	2	2	2	3	3
JCN	1	0	1	1	2	3	0	3
VAL	3	4	1	5	9	2	6	5

(b) Sparse matrix A expressed in COO scheme. Elements have been inserted in the array, from left to right, top to bottom, but there is no need of a particular order.

subscript	0	1	2	3	4	5	6	7
IRS	0	1	3	6				
JCN	1	0	1	1	2	3	0	3
VAL	3	4	1	5	9	2	6	5

(c) Sparse matrix A expressed in CSR scheme. Row elements have been inserted in the array, from left to right, but there is no need of a particular order inside a row.

subscript	0	1	2	3	4	5	6	7
JCS	0	2	5	6				
IRN	1	3	0	1	2	2	2	3
VAL	4	6	3	1	5	9	2	5

(d) Sparse matrix A expressed in CSC scheme. Column elements have been inserted in the array, from top to bottom, but there is no need of a particular order inside a column.

Figure 4.1: Representation of the sparse matrix A (a) in the COO (b), CSR (c) and CSC (d) schemes.

4.2 Binary search tree

A *binary tree* is a data structure recursively defined as either being empty or consisting of a node called the root, together with two binary trees called the left and right subtrees, respectively.

A binary search tree (BST) [47] labels each node in a binary tree with a single key such that keys in the left subtree are smaller than the key in the root node and keys in the right subtree are greater than the key in the root node. The left and right subtrees of a BST are also BSTs.

A BST *node* is a data structure consisting of a key, a value and two node references called left and right, respectively (see Code 4.1). A BST node may have none, one or two children, depending on none, one or both of its left and right references are pointing to another node. A node that has no children is called a *leaf* node.

```
struct node{
  unsigned int key;
  <T> value;
  node *left;
  node *right;
};
```

Code 4.1: C++ definition of a BST node, where <T> is the type of the value stored. Negative key values are not allowed, since node keys will map sparse array indexes.

An example of BST is illustrated in figure 4.2. The *level* of a node is calculated as the number of node references that have been followed to reach it from the root. Therefore, the root node always has level zero, its children have level one and so on.

The *height* of a BST is calculated as the maximum level of the tree nodes. A BST with n nodes may have height between a maximum of n and a minimum of $\log n$, depending on the BST structure. For example, if all nodes (except the leaf) have only one child, the BST will have height n and if all nodes have two children, the BST will have height $\log n$. In a *balanced* BST the levels of all leafs have a maximum difference of 1. Otherwise, the BST is called *unbalanced*. A balanced BST with n nodes has a height of $\log n$. This property is desirable since BST operations cost depends on tree height.

Three basic operations must be implemented for storing and retrieving data in a BST: search, insert and delete a BST node (see Code 4.2).

When searching for a specific key k_{search} in a BST, the root node (which has a key k_{root}) is visited. If $k_{search} < k_{root}$, then left node is visited next, if $k_{search} > k_{root}$, then right node is visited next, if $k_{search} = k_{root}$, then the search ends successfully

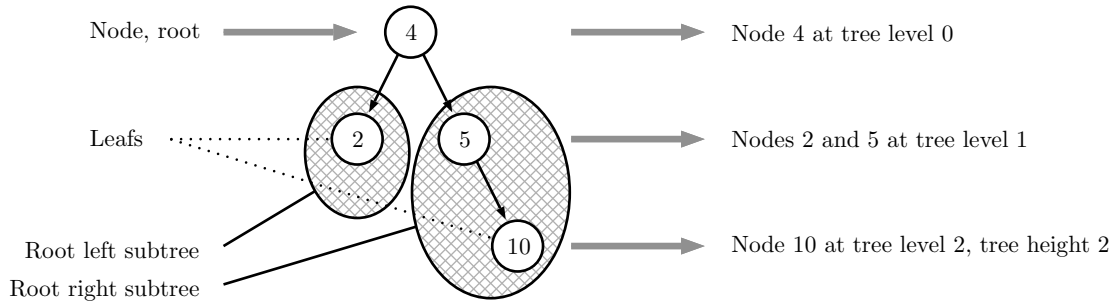


Figure 4.2: Example of a BST with four nodes, two leafs, two levels and therefore height two. The root node, its left and right subtrees and the two leaf nodes are indicated.

```
class BSTree{
  node *root;
  <T>  search_tree(unsigned int _key);
  void insert_node(unsigned int _key, <T> _value);
  void delete_node(unsigned int _key);
};
```

Code 4.2: C++ definition of a BST, where <T> is the type of the value stored. Three basic operations are needed to store data: search, insert and delete.

and the value of this node is returned. This process is repeated for visiting the next node until the search is successful or there is no remaining node. In the last case, the search is unsuccessful and there is no node with key k_{search} in the BST. Code 4.3 is an implementation of this procedure. As we will use BSTs to store non-zero values of sparse data, unsuccessful search return a zero value. Search algorithm runs in $\mathcal{O}(h)$ time, where h denotes the height of the tree.

For example, suppose v denotes a sparse vector and T denotes an empty BST. v will be stored in T if $\forall v_i \neq 0 \in v$ a node is inserted in T with key i and value v_i. A search for a node with key i is performed in T in order to retrieve the i-th element v_i. If the search is successful, the node value is returned. Zero is returned otherwise.

A search procedure for k_{insert} is performed to insert a node with key k_{insert} and value v_{insert} in a BST. A reference to the last visited node n_{last} during the search must be stored. If the search is successful, the node already exists and its value is updated to v_{insert} (duplicate keys are not allowed since a key represent a non-zero element index in sparse data). Otherwise, a new node is placed on n_{last} left or right, depending on $k_{insert} < k_{last}$ or $k_{insert} > k_{last}$, respectively. Where k_{last} is the key of n_{last}. Code 4.4 is an implementation of this procedure. Creating a new node and linking it into the tree is a constant-time operation after the search has

```
<T> BSTree::search_tree(unsigned int _key){
  node *target = root;
  while(target != NULL){

    if( _key < target->key){
      target = target->left;

    }else if( _key > target->_key){
      target = target->right;

    }else {
      return target->value;
    }
  }
  // key is not found,
  // the element must be zero.
  return 0;
}
```

Code 4.3: C++ definition of a node search in a BST, where <T> is the type of the value stored. This is an iterative version of a BST search. The search ends when the reference to the next node is NULL.

been performed in $\mathcal{O}(h)$ time.

Suppose v denotes a sparse vector, T denotes a BST and v is stored in T. When the value of v_i changes from v_i^{old} to $v_i^{new} \neq 0$, an insertion of a node with key i and value v_i^{new} in T is performed. If $v_i^{old} \neq 0$, a node already exists in the BST and its value is updated to v_i^{new}. Otherwise, a new node will be created at the end of the search path. An insertion of $v_i^{new} = 0$ is considered a deletion.

When deleting a node n with key k_{delete} from a BST, three cases arise: n has no children, one child or two children. As with the insertion, deletion starts with a search for n, and a reference to the last visited node n_{last} must be stored during the search. Once n is located, the first two cases are straightforward. If n has no children, the reference in n_{last} of n must be cleared, and n must be deleted. If n has one child, the reference in n_{last} of n must be updated to point to n's child, and n must be deleted. Otherwise, n has two children, then it has left and right subtrees. Let c denote the leftmost node in n right subtree and k_c denote its key. For the labeling properties of BSTs, $k_{delete} < k_c$ as c is in n right subtree and k_c is the smallest key in the n right subtree as c is its leftmost node. These two conditions allow to swap n and c key and value (not subtree references) without violating the properties of the BST. Therefore, in the case that n has two children, c is found, the key and value of n and c are swapped, and c is deleted. As c is the leftmost node in n right subtree, it is only possible that it has none or one (right) child, as otherwise its left child would be the leftmost node in the subtree. Code 4.5 and Figure 4.3 are an implementation and an illustration of this procedure, respectively. Deleting a node, updating its parent references or swapping the key and the value of two

```
void BSTree::insert_node(unsigned int _key, <T> _value){
  // search the tree
  node** p = &root;
  while( (*p) != NULL){

    if( _key < (*p)->key){
      p = &( (*p)->left);

    } else if( _key > (*p)->key){
      p = &( (*p)->right);

    } else {
      (*p)->value = _value;
      return;
    }
  }
  // node not found,
  // (*p) is a good place to store the new node
  (*p) = new_node();
  if( (*p) != NULL){
    (*p)->key = _key;
    (*p)->value = _value;
    return;
  }
}
```

Code 4.4: C++ definition of a method for a node insertion in a BST, where `new_node()` returns a reference to a new node and <T> is the type of the value stored. This is an iterative version of a BST insertion.

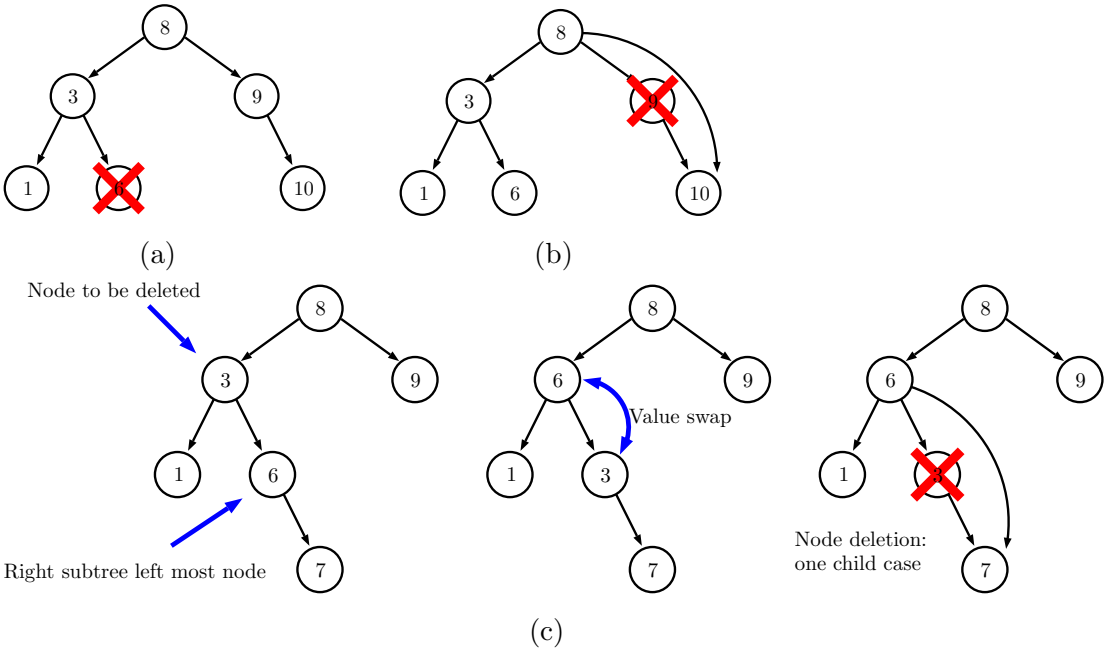


Figure 4.3: The three cases of BST node deletion: deletion of a BST node with no children (a), deletion of a BST node with one child (b), deletion of a BST node with two children (c).

nodes are constant-time operations after the search has been performed in $\mathcal{O}(h)$ time.

Suppose v denotes a sparse vector, T denotes a BST and v is stored in T. When the value of v_i changes from v_i^{old} to $v_i^{new} = 0$, a deletion of a node with key i in T is performed. If $v_i^{old} = 0$, the node does not exist in the BST and does not need to be deleted. Otherwise, a deletion of the node with key i is performed.

4.2.1 Balanced Binary search tree

The operations search, insert and delete a node, can end up with an unbalanced BST. These operations (as implemented in Codes 4.3, 4.4 and 4.5) only assure that the BST properties hold, regardless of the tree height. A more sophisticated implementations of self balanced tree data structures, such as AVL (named after Georgy Adelson-Velsky and E. M. Landis) or red-black trees are available. These implementations guarantee that the height of the tree always will be $\mathcal{O}(\log n)$ but they are more complex and the height of the tree is maintained incrementally during each operation. In the framework of sparse array data storing, we know the operations used by the algorithm that are going to be applied to the matrix (for example, a Givens rotation of two rows). Then, an estimation of *how much* these

```
void BSTree::delete_node(unsigned int _key){
  // search the tree
  node** p = &root;
  bool found = false;

  while( ( (*p) != NULL) && (!found) ){

    if( _key < (*p)->key){
      p = &( (*p)->left);

    } else if( _key > (*p)->key){
      p = &( (*p)->right);

    } else {
      found = true;
    }
  }

  if(found){
    node* n = (*p);

    // no children
    if( (n->left == NULL) && (n->right == NULL) ){
      (*p) = NULL;
      delete n;

      // left child only
    } else if( (n->left != NULL) && (n->right == NULL) ){
      (*p) = (*p)->left;
      delete n;

      // right child only
    } else if( (n->left == NULL) && (n->right != NULL) ){
      (*p) = (*p)->right;
      delete n;

      // both
    } else {
      // search for the left-most node of the right subtree
      // * all the right subtree keys are greater than n key
      // * all the right subtree keys are greater than
      //   the left-most key of the right subtree
      node** aux = &(n->right);
      while( (*aux)->left != 0){
        aux = &( (*aux)->left );
      }

      // copy left-most node
      n->key = (*aux)->key;
      n->value = (*aux)->value;

      // old left-most node has to be deleted
      n = (*aux);

      // replace (if any) left-most node by its child
      (*aux) = n->right;

      delete n;
    }
  }

  // not found, nothing to do
  return;
}
```

Code 4.5: C++ definition of an iterative version of a node deletion in a BST. The deletion procedure has three cases depending on the number of children of the node that has to be deleted.

operations unbalance the BST together with an algorithm to balance the BST would lead to an efficient and simple code. Moreover, the cost of the balance can be amortized over many operations. In 1986 was published [48] the Day–Stout–Warren (DSW) algorithm to balance a basic BST. This algorithm was designed by Q. Stout and B. Warren, based on work done by C. Day and is optimal, both in space (performed in-place with $\mathcal{O}(1)$ additional space) and time (runs in $\mathcal{O}(n)$). As the DSW algorithm is well known, and no modifications are needed to fit our implementation of BST, it will be not detailed here, but is explained in detail in [49].

The estimation of *how much* the BST is unbalanced, from the BST point of view, can be obtained with two additional attributes in the tree: `unsigned int tree_height` and `unsigned int tree_nodes`.

- `tree_nodes` is constructed with a value of zero, this value is incremented each time `new_node()` is called in Code 4.4 and is decremented each time `delete` is called in Code 4.5.

- `tree_height` is constructed with a value of zero and this value is updated in reference to a local variable `unsigned int search_depth` in each search procedure performed in the insertion and deletion. `search_depth` is initialized to zero at the beginning of a search and incremented when the search reaches the next level of the BST. If `search_depth` > `tree_height`, `tree_height` is updated to the value of `search_depth`. When a DSW balancing is applied, `tree_height` is reset to the value of $\log_2$ `tree_nodes`.

This is an approximation, as it is never assured that a particular search path would be the longest, but it stores the worst search performed on the BST between balances. While `tree_height` $\approx \log$ `tree_nodes`, it can be assumed that the BST is balanced. As `tree_height` tends to the value of `tree_nodes`, we get proof that the BST is unbalanced. The tolerances and limits of the above conditions have to be considered regarding each use case of the BST, but when `tree_height` = `tree_nodes` holds, next search procedure (present in the three basic operations) will run in $\mathcal{O}(n)$ and a DSW balance (which have the same cost) will speed up next basic operations.

4.3 Implementation of the BST sparse matrix

Using the definitions of the previous section, in which a BST and its operations are defined for effectively manage a sparse array, the implementation of a sparse matrix structure based on BST is now straightforward. As a matrix is an array of arrays of the same length, a sparse matrix structure can be implemented as a list of

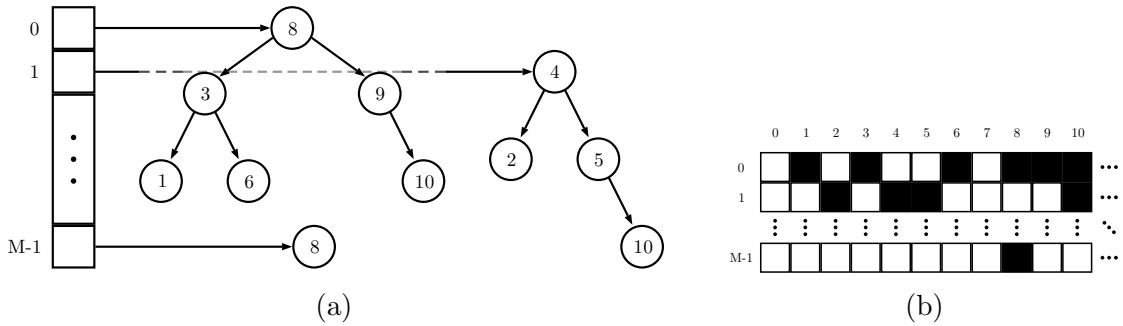


Figure 4.4: Example of storage of a sparse matrix (b) in a list of balanced BSTs (a). Each element of the array stores a reference to a tree that stores non-zero elements of a row. The elements are sorted in the tree by their column index.

(balanced) BSTs. Each element of the list represents a matrix row, and non-zero elements of a row are values of a balanced tree, sorted by its column index (see figure 4.4).

Using this storage format, matrix element access, modification, deletion and insertion run in $\mathcal{O}(h)$ time, where h denotes the height of the tree in which element is stored. If the tree is balanced, then $h = \log n$, where n is the number of non-zero elements in a row (a BST must remain balanced in order to maintain operations at this cost). The space required to store the whole matrix is $\mathcal{O}(n + M)$, where n is the number of non-zero elements and M is the number of rows of the matrix. Code 4.6 is an implementation of a list of BSTs to store a matrix.

An element a_{ij} can be accessed (through the `get()` method, see Code 4.7) by a search for j in the tree referenced by the i-th position of the array. If the search is successful, the node value is obtained. If j is not found, the element must be zero.

For the insertion (modification or deletion, through the `set()` method, see Code 4.8) of an element a_{ij} in this format, first, the position i of the array is accessed in order to obtain a tree reference. Then, an insertion with key j is performed in the obtained tree. If the key is found during the insertion, the value of the node is updated. If it is no found, a new node is created. We will suppose that an insertion of $a_{ij} = 0$ is actually a deletion. For the deletion of an element (i, j), a tree reference is obtained from the i-th position of the array. Then, a search for j is performed in the obtained tree. If the search is successful, the node is deleted. If j is not found, the deletion is not necessary.

```
class BSTMatrix{
  BSTree *row_bst;
  unsigned int rows;
  unsigned int columns;

  BSTMatrix(unsigned int _rows, unsigned int _columns);
  ~BSTMatrix();
  <T>  get(unsigned int i, unsigned int j);
  void set(unsigned int i, unsigned int j, <T> value);
};

BSTMatrix::BSTMatrix(unsigned int _rows, unsigned int _columns){
  rows    = _rows;
  columns = _columns;
  row_bst = new BSTree[rows];
  return;
}

BSTMatrix::~BSTMatrix(){
  rows    = 0;
  columns = 0;
  delete[] row_bst;
  return;
}
```

Code 4.6: C++ definition of a list of BSTs to store a matrix. Attributes `rows` and `columns` are considered for further use (as matrix bounds checking). `get()` and `set()` methods are needed to read and write in the matrix.

```
<T> BSTMatrix::get(unsigned int i, unsigned int j){
  return row_bst[i].search_tree(j);
}
```

Code 4.7: C++ definition of the `get()` method used to read elements from the matrix. The implementation of this method relies on the tree `search` method and assumes that zero will be returned if the element does not exist in the tree.

```
void BSTMatrix::set(unsigned int i, unsigned int j, <T> value){
  if( value != 0){
    // insertion
    row_bst[i].insert_node(j, value);

  } else {
    // deletion
    row_bst[i].delete_node(j);
  }
  return;
}
```

Code 4.8: C++ definition of the `set()` method used to write elements on the matrix. The implementation of this method relies on the tree `insert` and `delete` methods. A `value` $\neq 0$ is considered an insertion or modification and otherwise a deletion.

4.3.1 Performance improvements

The methods `get(i, *)` and `set(i, *, *)` run in $\mathcal{O}(\log n)$ time, where n is the number of non-zero elements of the i-th row, as the location of the i-th BST runs in constant time. In general, best performance will be achieved if, in average, there are less non-zero elements in a row than in a column and the rows of the matrix are stored in the BSTs (or *vice versa*).

The BST based sparse matrix can be modified to improve run time or space. As usual, improvement in run time will compromise space and improvement in space will compromise run time. The choice between space and run time improvement relies on the problem to solve, for example, the QR decomposition of a large sparse matrix requires a large number of rotations that will require an even larger number of matrix element access, although there may be no space available to store a dense list of references for each row.

Suppose that there are less non-zero elements in a row than in a column and for all rows, their non-zero elements can be divided in k sets of approximately the same size. Further run time improvement can be achieved if the columns of the matrix are divided in k sets. If $k = 2$, $A \in \mathbb{R}^{M\times N}$ is the matrix that is going to be stored and for all rows, the number of non-zero elements in the first $\frac{N}{k}$ columns are approximately the same as the number of non-zero elements in the last $\frac{N}{k}$ columns, then two arrays (s and t), of M BST references each, can be used and non-zero elements with column indexes $j < \frac{N}{k}$ are stored (and searched) in trees referenced by s and non-zero elements with column indexes $j \geq \frac{N}{k}$ are stored (and searched) in trees referenced by t. The corresponding tree for a matrix element is still found in constant time and its search runs, in average, in $\mathcal{O}(\log \frac{n}{k})$ time. But this approach uses kM additional space and the division in k sets of approximately the same number of non-zero elements for all rows is a condition that many sparse

matrices do not hold (for example, a band matrix).

Considering large sparse matrices in which not all rows contain non-zero values, using an array of M BST references may be a waste of space, as a sparse array will be stored in a dense structure. Applying the same principle as in §4.2, the array containing the BST references may be stored in a BST T_r. `get(i, *)` method must assume now that if there is no node with key i in T_r, returned value must be zero. `set(i, *, *)` method must assume that the insertion of the first element of the i-th row implies the creation of a node with key i in T_r and the deletion of the last element of the i-th row implies the deletion of the the node with key i in T_r. With this approach, the methods `get(i, *)` and `set(i, *, *)` run in $\mathcal{O}(\log m + \log n)$ time, where n is the number of non-zero elements of the i-th row and m is the number of rows that contain at least one non-zero element.

Finally, the interface of this data structure allows an extension in order to extract a whole row, which may be useful for some problems. Assuming enough space is provided to this new method to store all non-zero elements of a row, once located its corresponding BST, the well known in-order traversal algorithm (see [47]) will fill the provided space with the non-zero elements of the row (sorted by their column index) in $\mathcal{O}(n)$ time, where n is the number of non-zero elements of the row. The same does not apply in order to extract a whole column, in which case for each row, a certain column index must be searched in its corresponding BST, running in $\mathcal{O}(Mn)$ time, where M is the number of rows and n is the average number of non-zeros in a row.

Summarizing, a sparse matrix data structure based on BSTs has been defined, and will be used to store and work with sparse matrices during the rest of this thesis as a black box. This data structure is easy to implement and offers the methods `get()` and `set()` for data manipulation. In one hand, the space required to store the whole matrix is $\mathcal{O}(n_0 + M)$, where n_0 is the number of non-zero elements and M is the number of rows of the matrix, which is comparable to traditional sparse matrix data structures like CSC. On the other hand, the methods `get(i, *)` and `set(i, *, *)` run in $\mathcal{O}(\log n)$ time, where n is the number of non-zero elements of the i-th row, which is an order of magnitude lower than element access run time of traditional sparse matrix data structures like CSC ($\mathcal{O}(n)$). Furthermore, space and time complexities can be adjusted to meet the requirements of certain problems, for example, run time improvement of $\mathcal{O}(\log \frac{n}{k})$ with a space requirement of $\mathcal{O}(n_0 + kM)$ or space improvement of $\mathcal{O}(n_0)$ with a run time drawback of $\mathcal{O}(\log m + \log n)$ time, where m is the number of rows that contain at least one non-zero element.

Chapter 5

Implementation of the QR algorithm

In §3.1, we proposed the problem of CT reconstruction, through an overdetermined linear system. Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{M\times N}$, $x \in \mathbb{R}^N$, $b \in \mathbb{R}^M$ and $M > N$. And suppose that

$$A = \begin{pmatrix} R_{11} \\ A_{21} \end{pmatrix}; \qquad \begin{pmatrix} R_{11} \\ A_{21} \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

where R_{11} is upper triangular and nonsingular, so we can obtain *an* x applying backward substitution to $R_{11}x = b_1$. However, an overdetermined system may have no solution. This x will only be solution of the overall system if $A_{21}(R_{11}^{-1}b_1) = b_2$, that is, x solves the rest of the system. Otherwise, we need to find an x that *fits* in the linear system and makes $Ax - b$ *small* enough to approve x as a solution. This problem is studied in detail in [27, 28, 29] among others, and during this chapter we are going to collect the required tools to solve it and propose an efficient implementation to compute our solution. Although the QR factorization has been widely used to solve least squares problems, to the best of our knowledge, this is the first work that uses the QR factorization for image reconstruction in CT.

5.1 The QR algorithm

The QR factorization of a matrix $A \in \mathbb{R}^{M\times N}$ is given by

$$A = QR \tag{5.1}$$

where $R \in \mathbb{R}^{M\times N}$ is upper triangular with positive main-diagonal entries and $Q \in \mathbb{R}^{M\times M}$ is *orthogonal*, which means that $QQ^T = I$, so Q has an inverse, and

$Q^{-1} = Q^T$ (proof for uniqueness of Q and R can be found in [28]). The general idea for solving a linear system $Ax = b$ is to perform the factorization 5.1, exploiting the easy calculation of $Q^{-1} = Q^T$ and applying the backward substitution algorithm (see §2.1.1) in the equivalent system 5.2d to obtain x.

$$Ax = b \tag{5.2a}$$
$$QRx = b \tag{5.2b}$$
$$Q^TQRx = Q^Tb \tag{5.2c}$$
$$Rx = Q^Tb \tag{5.2d}$$

In §3.1 we obtained an overdetermined linear system $Ax = b$, where $A \in \mathbb{R}^{M\times N}$, $b \in \mathbb{R}^M$, $x \in \mathbb{R}^N$ and $M > N$, which solution x will be the reconstructed image from a CT measurement. As usual, this overdetermined system has no exact solution, so we need to minimize $||Ax - b||_p$ for some p. The solution of the least squares (LS) problem (for $p = 2$)

$$\min_{x\in\mathbb{R}^N} \|Ax - b\|_2 \tag{5.3}$$

can be obtained via the equivalent problem as in 5.2

$$\min_{x\in\mathbb{R}^N} \|Rx - (Q^Tb)\|_2 \tag{5.4}$$

because the 2-norm is preserved under orthogonal transformations. Also, there is an unique LS solution for 5.4 (proof for uniqueness can be found in [27]) and it is the solution we are looking for.

5.2 Rotations and reflections

Only orthogonal transformations are allowed to perform the QR decomposition. Since those transformations will be applied to a large sparse matrix, the chosen transformation must *preserve* the sparsity of the matrix as much as possible and lead to intermediate results during the QR decomposition that can be held in memory. In this section we are going to introduce the *Householder* and *Givens transformations* considering these factors.

5.2.1 Householder reflections

A *reflection*, *Householder transformation* or *Householder reflection* [50] $H = I - \gamma uu^T$ is a linear transformation that reflects a vector through a hyperplane $\mathcal{P}$, where I is the identity matrix, u is a non-zero vector orthogonal to $\mathcal{P}$ and $\gamma = \frac{2}{\|u\|_2^2}$

(see Davis [27], §5). It can be used to create zeros in the first column of a matrix $A \in \mathbb{R}^{M\times N}$, $M > N$.

$$H \begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{pmatrix} = \begin{pmatrix} -\tau \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$H_1 A = \left(\begin{array}{c|ccc} -\tau_1 & \hat{a}_{12} & \dots & \hat{a}_{1n} \\ \hline 0 & & & \\ \vdots & & \hat{A}_1 & \\ 0 & & & \end{array}\right)$$

Leaving the first row and the first column unchanged, a second reflection can be computed to create zeros in the second column of A,

$$H_2 = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & I - \gamma_2 u^{(2)} u^{(2)T} \end{array}\right)$$

$$H_2 \hat{A}_1 = \left(\begin{array}{c|ccc} -\tau_2 & \hat{a}_{23} & \dots & \hat{a}_{2n} \\ \hline 0 & & & \\ \vdots & & \hat{A}_2 & \\ 0 & & & \end{array}\right)$$

and in general in the j-th column of A with

$$H_j = \left(\begin{array}{c|c} I_{j-1} & 0 \\ \hline 0 & I - \gamma_j u^{(j)} u^{(j)T} \end{array}\right)$$

where I_{j-1} is the identity matrix of dimension $j-1$. After N steps, $H_N H_{N-1} \dots H_1 = Q^T$ and $A = QR$, where $R \in \mathbb{R}^{M\times N}$ is upper triangular. The QR decomposition through Householder transformations runs in $\mathcal{O}(MN^2)$ time. Note that there is no need to calculate the matrix Q explicitly (see Watkins [28], §3.2, algorithm 3.2.40), only γ_j and u_j have to be stored.

In the sparse case, the QR decomposition through Householder transformations may not be convenient. This depends on the non-zero pattern of A. The non-zero pattern of each row modified by a Householder transformation will be the set union of non-zero patterns of all rows modified by the Householder transformation (see Davis [29], §5). For some sparse matrices (as the matrix A described in §3.1, for example) $\hat{A}_1$ will be less sparse than A, $\hat{A}_2$ will be less sparse than $\hat{A}_1$ and in general, at the early stages of the QR decomposition, $\hat{A}_j$ will become a full matrix. In our case, there is simply not enough space to hold and work with a full version of A, so we need to make use of a more selectively transformation to zero elements.

5.2.2 Givens rotations

A *rotation*, *Givens transformation* or *Givens rotation* [51] is a linear transformation that rotates a vector through an angle θ. It can be represented by a matrix

$$G_{ji} = \begin{array}{c} \\ \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccccc} & & & & i & & & & j & & \end{array} \\ \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & \cos\theta & & & & -\sin\theta & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & \sin\theta & & & & \cos\theta & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix} \end{array} \tag{5.5}$$

where blank entries are zeros, and used to zero the element a_{ji} of A.

For example, consider $x \in \mathbb{R}^2$ and suppose we want to find G_{21} so that $G_{21}^T x$ has a zero in its second component, where

$$G_{21} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \qquad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad G_{21}^T x = \begin{pmatrix} y \\ 0 \end{pmatrix}$$

$$G_{21}^T x = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} (\cos\theta)x_1 + (\sin\theta)x_2 \\ -(\sin\theta)x_1 + (\cos\theta)x_2 \end{pmatrix} \quad \text{and}$$

$$(\cos\theta)x_2 - (\sin\theta)x_1 = 0 \tag{5.6}$$

There is no need to calculate θ to compute a rotation in order to zero x_2. We only need $\cos\theta$ and $\sin\theta$ to determine G. If we take

$$\cos\theta = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} \quad \text{and} \quad \sin\theta = \frac{x_2}{\sqrt{x_1^2 + x_2^2}} \tag{5.7}$$

equation 5.6 will be satisfied and there is a unique $\theta \in [0, 2\pi)$ for which choice 5.7 holds.

Consider now $A \in \mathbb{R}^{M \times N}$, $M > N$ and we want to find Q so that $Q^T A$ is a triangular matrix. As in the above example, we must compute rotations G_{M1}^T, ..., G_{31}^T, G_{21}^T in order to zero the first column of A except a_{11}.

$$G_{21}^T G_{31}^T \dots G_{M1}^T A = \left(\begin{array}{c|ccc} \hat{a}_{11} & \hat{a}_{12} & \dots & \hat{a}_{1N} \\ \hline 0 & & & \\ \vdots & & \hat{A}_1 & \\ 0 & & & \end{array}\right)$$

Then, as for the first column, we must compute rotations G_{M2}^T, ..., G_{42}^T, G_{32}^T in order to zero the second column except a_{12} and a_{22}.

$$G_{32}^T G_{42}^T \dots G_{M2}^T \hat{A}_1 = \left(\begin{array}{c|ccc} \hat{a}_{22} & \hat{a}_{23} & \dots & \hat{a}_{2N} \\ \hline 0 & & & \\ \vdots & & \hat{A}_2 & \\ 0 & & & \end{array}\right)$$

After N steps, $G_{N+1,N}^T \dots G_{32}^T G_{42}^T \dots G_{M2}^T G_{21}^T G_{31}^T \dots G_{M1}^T = Q^T$ and $A = QR$, where $R \in \mathbb{R}^{M \times N}$ is upper triangular. Note that (as in Householder reflections) there is no need to calculate the matrix Q explicitly, it is sufficient to store the values of i, j, $\cos\theta$ and $\sin\theta$ to define a rotation and store rotations in the same order that they were applied to A.

For simplicity, we generated the rotations column-oriented, bottom to top, but the QR decomposition with Givens rotations can be implemented row-oriented (left to right), or in any order while the two rows involved in the rotation have their first non-zero element in the same column. The QR decomposition through Givens transformations also runs in $\mathcal{O}(MN^2)$ time, but choosing carefully the rows involved in the rotations (see §6) we can maintain $\hat{A}_1, \hat{A}_2, \dots, \hat{A}_N$ *relatively* sparse. Consider that $G^T A$ changes two rows, i and j, to zero element a_{ji} of A. This is convenient because brings the possibility of choosing rows i and j to have similar sparsity pattern in order to reduce fill-in and maintain the sparsity of A. This will be treated in §6, and, for now, lets suppose that the sparsity of A will be *damaged* but not lost during the QR decomposition, so Givens rotations will be our transformation of choice.

Numerical properties of rotations and reflections are excellent. Along with a detailed error analysis of rotations and reflections, Wilkinson [52] has shown that the computation of $Q^T A$ (where Q is either a rotation or a reflection) is normwise backward stable. That is, the computed result of $Q^T A$ is the exact result of applying Q^T to a slightly different matrix $A + E$, where $\frac{\|E\|_2}{\|A\|_2}$ is small. This stability is preserved under repeated applications of rotations or reflections, so the QR decomposition is also normwise backward stable.

5.3 Implementation

The QR decomposition using Givens rotations is the appropriate algorithm to solve our problem. General implementations of this algorithm may not cover all required features such as using a custom sparse matrix scheme or control over the sequence of rotations to reduce the matrix to triangular form, so we implemented in C++ the whole algorithm from scratch. Our implementation uses the BST sparse

matrix scheme and is organized in three blocks: the computation of a Givens rotation (§5.3.1), the reduction to triangular form of a matrix (§5.3.2) and the storage of the rotations applied to the matrix along with a especialized procedure to compute $Q^T b$ (§5.3.3).

5.3.1 Givens rotations

The implementation of a Givens rotation

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$$

where $c = \cos\theta$, $s = \sin\theta$ and x_1, x_2, $r \in \mathbb{R}$, is available in standard software packages of linear algebra such as BLAS [53] or LAPACK [54]. In this implementations, given $\{x_1, x_2\}$, $\{c, s, r\}$ are computed. The numerical stability of this procedure has two key points: x_1 and x_2 must be scaled to avoid underflow or overflow in the computation of formulas 5.7 and (as Anderson [55] observed) there are implementations of this procedure that have discontinuity in the value of r. This section is a review of the Anderson's algorithm (see Code 5.1) to implement a Givens rotation avoiding underflow or overflow and assuring continuity in the computation of r.

inputs: `x1, x2`
outputs: `c, s, r`

```
if (|x1| > |x2|) then

  t = x2 / x1
  u = sgn(x1) * sqrt(1 + (t * t))
  c = 1 / u
  s = t * c
  r = x1 * u

else

  t = x1 / x2
  u = sgn(x2) * sqrt(1 + (t * t))
  s = 1 / u
  c = t * s
  r = x2 * u

end if
```

Code 5.1: Pseudocode of Anderson's algorithm to implement a Givens rotation, where **sgn** (x) returns -1 if $x < 0$ and 1 otherwise and **sqrt** (x) returns $\sqrt{x}$.

Overflow or underflow may occur in the computation of the formulas 5.7. When $|x| < \sqrt{R_{min}}$, where R_{min} is the smallest positive floating-point number representable on the machine, x^2 will underflow and when $|x| > \sqrt{R_{max}}$, where R_{max}

is the largest positive floating-point number representable on the machine, x^2 will overflow. In Code 5.1, x_1 and x_2 are scaled to 1 and $t \in [0,1]$, avoiding overflow or underflow on the computation of x_1^2 and x_2^2. Suppose that $|x_1| > |x_2|$ then $t = \frac{x_2}{x_1}$,

$$c = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} = \frac{1}{\frac{1}{x_1}\sqrt{x_1^2 + x_2^2}} = \frac{1}{\sqrt{\frac{1}{x_1^2}(x_1^2 + x_2^2)}} = \frac{1}{\sqrt{1+t^2}}$$

$$s = \frac{x_2}{\sqrt{x_1^2 + x_2^2}} = \frac{x_2}{x_1}\frac{1}{\frac{1}{x_1}\sqrt{x_1^2 + x_2^2}} = tc$$

and similarly when $|x_1| \leq |x_2|$.

There are implementations of the Givens rotation that may have discontinuity in the value of r, produced by changes in its sign when, for example, $x_2 = -x_1$. Note that

$$r = cx_1 + sx_2 = \frac{x_1^2}{\sqrt{x_1^2 + x_2^2}} + \frac{x_2^2}{\sqrt{x_1^2 + x_2^2}} = \frac{x_1^2 + x_2^2}{\sqrt{x_1^2 + x_2^2}} = \sqrt{x_1^2 + x_2^2} \tag{5.8}$$

is always positive. In Code 5.1,

$$\text{sgn}(r) = \begin{cases} \text{sgn}(x_1)\text{sgn}(x_1) & \text{if } |x_1| > |x_2| \\ \text{sgn}(x_2)\text{sgn}(x_2) & \text{if } |x_1| \leq |x_2| \end{cases}$$

is always positive, since $\sqrt{1+t^2}$ is always positive.

Finally, there are three especial cases that have to be considered. When $x_1 \geq 0$ and $x_2 = 0$ there is no need of a rotation. The computation of this rotation will produce $c = 1$, $s = 0$ (the identity matrix) and $r = x_1$. When the sparse matrix to be reduced to triangular form is sufficiently large, as our case, performing unneeded rotations will increase considerably the required storage space for Q, the required computation time of the reduction to triangular form and the required computation time of the operation $Q^T b$ (see 5.2). For these reasons, in Code 5.1 there is no consideration for the case $x_1 = 0$ and $x_2 = 0$ (that will end in a division by zero) and the users of this algorithm must avoid calling it in this case.

When $x_1 < 0$ and $x_2 = 0$, Code 5.1 will produce $c = -1$, $s = 0$ and $r = |x_1|$. This is to ensure that all computed values of r will be positive (see 5.8). It may be considered to detect this especial case and avoid calling this algorithm, storing separately a *especial operation* instead. This operation consists of at least two elements: the row index in which the change of sign will be performed and its relative order with the rest of rotations, which is half the space needed for a rotation. But this leads to a drawback of maintaining a second structure of operations which leads to a more complicated programming than consider it a rotation and a continuous check on the number of rotations already applied in the computation of $Q^T b$. Therefore, we will store a rotation for this case.

In the last especial case, when $x_1 = 0$ and $x_2 \neq 0$, Code 5.1 will produce $c = 0$, $s = +/-1$ and $r = |x_2|$. This, in fact, will perform a swap of the rows in the matrix that contain x_1 and x_2, automatically taking care of the sign of r. The same consideration as the previous especial case can be made. But in this case, the *especial operation* will be formed, at least, by the two row indexes to swap, the sign to assure r will be positive and its relative order with the rest of rotations, which is the same space needed for a rotation. Therefore, there is no advantage considering this case separately.

5.3.2 Givens QR decomposition

Using Code 5.1 of previous section it is possible to introduce zeros in a matrix $A \in \mathbb{R}^{M \times N}$ to reduce it to triangular form. For simplicity on following codes, we define the function *givens* as Code 5.1 which computes c and s, the function *rotate* as

$$\texttt{rotate}(c, s, A, j, i) : A(\{j, i\}, *) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} A(\{j, i\}, *)$$

where $A(\{j, i\}, *)$ is the submatrix of A formed by the rows i and j (with all its columns) and the function *store* which saves $\{c, s, j, i\}$ for later use (such as performing $Q^T b$ operations).

There are different strategies to introduce zeros in a matrix. Perhaps, the most intuitive strategy is select the first N rows, one by one from top to bottom, and then zero the column under its left-most element, from bottom to top. This strategy is depicted in Figure 5.1.

In the sparse case, may have sense to establish a sequence to introduce the zeros in a column, based on the sparsity pattern of each row, in order to minimize the fill-in (see §6). In cases that an effective ordering can not be found, like our case, this strategy will use $\mathcal{O}(MN)$ intermediate space to store elements produced by the fill-in during the reduction to triangular form. Even if some of the elements in a column are zero, the fill-in will spread for the most of the matrix, requiring more rotations to zero the rest of the columns, which in turn will produce more fill-in.

If $M \gg N$, this may be a critical problem. Consider $M \gg N$, such that there is enough space to store a $\mathbb{R}^{N \times N}$ matrix but not a $\mathbb{R}^{M \times N}$ matrix. Row-wise zeroing strategy is suitable for this case. This strategy is depicted in Figure 5.2, and introduces zeros, row by row, until the matrix main-diagonal is reached. Once the first N rows are reduced to triangular form (first three steps of Figure 5.2), each remaining row will be zeroed completely against the first N rows (last three steps of Figure 5.2). This constraints the fill-in in the matrix and use $\mathcal{O}(N^2)$

intermediate space to store elements produced by the fill-in during the reduction to triangular form, so we commit to the Figure 5.2 strategy.

The especial cases viewed in §5.3.1 are of interest in the sparse case. Both strategies in Figure 5.1 and Figure 5.2 have a guard against zeroing an element $A(i, j)$ that is already zero. This leaves two especial cases:

- Element $A(j, j) = 0$ and $A(i, j) \neq 0$, in which case Code 5.1 will perform a swap of rows automatically taking care of the sign of the swapped $A(j, j)$ thus, leaving $A(i, j) = 0$.
- Element $A(j, j) < 0$ and $A(i, j) = 0$, in which case no rotation is applied and it is expected that, further in the reduction process, a rotation involving $A(j, j)$ and $A(i, j)$ will be issued such that $A(i, j) \neq 0$, in order to take care of the sign of $A(j, j)$.

As this *might* not happen, at the end of the reduction, a search for elements $A(j, j) < 0$ must be performed and a rotation $\{c, s\} = \texttt{givens}(A(j, j), A(j+1, j))$ must be applied to A if any. Note that this rotation will leave $A(j + 1, j) = 0$ and will take care of the sign of $A(j, j)$. In practice, this last especial case is rare, as it implies there is no element to rotate with $A(j, j)$ and $A(j, j)$ will not produce fill-in. That is, $A(j, j)$ is the only non-zero element in its column and its left non-zero elements of its row (if any) are the only non-zero elements in their respective columns.

The next step in the implementation of the QR decomposition with Givens rotations is the implementation of $\texttt{rotate}(c, s, A, j, i)$. The time complexity for the QR decomposition using Givens rotations given in §5.2 assumes that the access of a element of the matrix $A \in \mathbb{R}^{M \times N}$ is a constant time operation, which means that A is fully stored. For large sparse matrices, this may be not convenient nor possible. The time complexity considering the use of a sparse matrix scheme is $\mathcal{O}(MN)$ times the cost of $\texttt{rotate}(c, s, A, j, i)$ (see Code 5.2).

Due to fill-in, note that some of the `A.set()` calls will end up in a insertion in A rather than a modification. Considering the time complexity of the CSR operations (§4.1), $\texttt{rotate}(c, s, A, j, i)$ runs in $\mathcal{O}(Nn_0)$, where n_0 is the number of non-zeros in A. Modifications can be made to Code 5.2 to operate with the entire two rows of A instead of each element separately, leading to a cost of $\mathcal{O}(n_0)$, but also requiring to dynamically manage the space in the CSR structure to store the rotated rows, which likely will have more elements due to fill-in. In our case, the determination of a tight upper bound for the space required by the upper triangular matrix R is not trivial (see §6).

Considering the time complexity of the `BSTMatrix` methods of §4.3, $\texttt{rotate}(c, s, A, j, i)$ runs in $\mathcal{O}(N \log n)$, where n is the average number of non-zeros in each row of A. Modifications can be made to Code 5.2 to operate with

```
for j = 1 to n
  for i = m to j+1 in steps of -1
    if(A(i, j) != 0) // avoid needless rotations in the sparse case
      {c, s} = givens(A(j, j), A(i, j))
      rotate(c, s, A, j, i)
      store(c, s, j, i)
    end if
  end for
end for
```

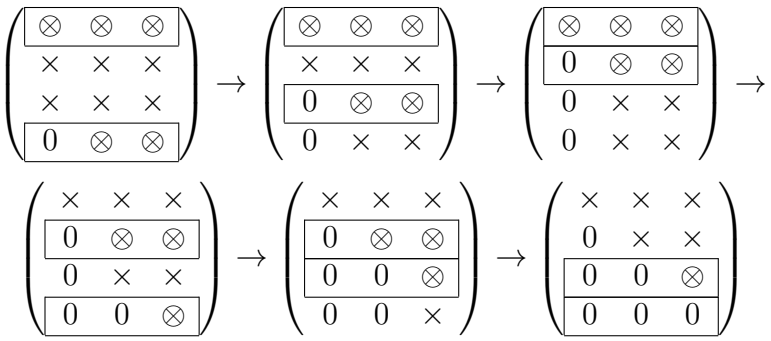


Figure 5.1: Reduction to triangular form using column-wise strategy. Rotations Q_{41}^T, Q_{31}^T, Q_{21}^T, Q_{42}^T, Q_{32}^T, Q_{43}^T are applied. In each stage of the reduction, framed rows are involved in its corresponding rotation and circled elements are susceptible of fill-in.

the entire two rows of A instead of each element separately, leading to a cost of $\mathcal{O}(kn)$. This can be achieved by making use of the in-order traversal of a BST to compute an array of pairs <column index, element value> and a modification of the `set_union` algorithm (provided in the C++ stl library, for example) to iterate over the indexes and perform rotations of two elements (if found in both sets) or one element with a zero (if found only in one set). As the explanation suggests, the efficient implementation of this modification is not simple and will determine the value of k, which is not negligible. Furthermore, this will imply the destruction, reconstruction and balance of the two BST involved in the rotation. The overhead of the memory allocations and destructions will likely overwhelm the improvement on the time complexity.

Suppose that $n_0 \approx Mn$, where n_0 is the number of non-zero elements of $A \in \mathbb{R}^{M \times N}$ and n is the average number of non-zero elements per row in A. Balancing the costs of access and insertion/deletion in the `BSTMatrix` methods we obtained a cost of $\mathcal{O}(N \log n)$ in the rotation procedure, which is an entire order of magnitude less that the implementation using CSR scheme, even with a row-wise implementation of the rotation, which time complexity is $\mathcal{O}(n_0) \approx \mathcal{O}(Mn)$. Moreover, if $M \gg N$, that is our case, the time cost benefit is even larger.

```
for i = 2 to m
  for j = 1 to min(i-1, n)
    if(A(i, j) != 0) // avoid needless rotations in the sparse case
      {c, s} = givens(A(j, j), A(i, j))
      rotate(c, s, A, j, i)
      store(c, s, j, i)
    end if
  end for
end for
```

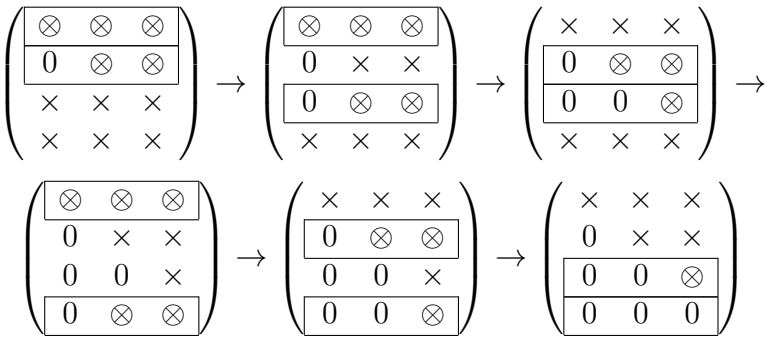


Figure 5.2: Reduction to triangular form using row-wise strategy. Rotations Q_{21}^T, Q_{31}^T, Q_{32}^T, Q_{41}^T, Q_{42}^T, Q_{43}^T are applied. In each stage of the reduction, framed rows are involved in its corresponding rotation and circled elements are susceptible of fill-in.

```
void QRDecomposition::givens_rotation(float c, float s, BSTMatrix& A, int j, int i){

  int N = A.columns;

  // through all N columns (element before column j will be zero)
  //   for(int col = 0; col < N; ++col){
  // starting from column j (first non-zero elements of the rows)
  for(int col = j; col < N; ++col){

    // retrieve values from matrix
    ajj = A.get(j, col);
    aij = A.get(i, col); // element to be zeroed

    // rotate values
    rjj =  (c * ajj) + (s * aij);
    rij = -(s * ajj) + (c * aij); // if col = j then rij = 0

    // save values
    A.set(j, col, rjj);
    A.set(i, col, rij);
  }

  return;
}
```

Code 5.2: Element-wise implementation in C++ (as a method of the class *QRDecomposition*) of the operation `rotate`(c, s, A, j, i) that appears in Figure 5.1 and Figure 5.2.

In conclusion, the total cost of the QR decomposition process, making use of the `BSTMatrix` in a element-wise access is $\mathcal{O}(MN^2 \log n)$ which is near the optimum cost of $\mathcal{O}(MN^2)$ using full matrices.

5.3.3 Q storage

The orthogonal matrix Q^T is composed of the Givens rotations used to reduce A to triangular form. Later, Q^T will be used in the operation $Q^T b$ to solve the linear system. In the previous implementation of the QR decomposition algorithm, we stored Q^T implicitly with the procedure `store`(c, s, j, i). We intentionally left the implementation of this procedure open because its strong dependence of the dimensions of the problem to solve with the QR decomposition. Suppose that this procedure stores each rotation in the data structure of Code 5.3. For example, it is a matter of how many rotations are performed to determine if rotations are held in memory and written to disk at the end of the process, or there is a need for some kind of buffered writing to disk during the decomposition. Moreover, if it is planned to solve a single linear system, and enough memory is available, is no even necessary to write them to disk.

In our case, we use the data structure of Code 5.3 to store a rotation. The

```
struct GivensRotation{
  int e;   // from "element": row index of the element to be zeroed
  int p;   // from "pivot": row index of the element not to be zeroed
  float c; // from "cosine": cosine of the rotation
  float s; // from "sine": sine of the rotation
};
```

Code 5.3: Data structure to hold a Givens rotation (not in compact form). `e` and `p` are chosen to be `int` because it is expected that system matrices have at most $\approx 2^{30}$ rows. `c` and `s` are chosen to be `float` because the expected precision of the solution is `float`. This structure has a size of 4×32 bits $= 16$ Bytes.

problem of CT reconstruction requires to solve different systems with the same matrix (the CT) and different right hand sides (CT measurements), so we need to store the rotations to disk. The system matrix will be sufficiently large to expect that there will be not enough memory available for all the rotations, so we need a buffered writing to disk during the decomposition. We especialized the `givens_rotation` method for the computation of the operation $Q^T b$, which has b as input, uses a buffered reading and overwrites b with $Q^T b$ (see Code 5.4). The time complexity of this method is $\mathcal{O}(g)$, where g is the number of Givens rotations that have to be applied. It is difficult to express g in terms of matrix dimensions M and N. Considering $n_0 \approx Mn$, where n_0 is the total number of non-zero elements of the matrix and n is the average number of non-zero elements of a matrix row, $Mn \leq g \leq MN$ and g will depend on the fill-in minimization success.

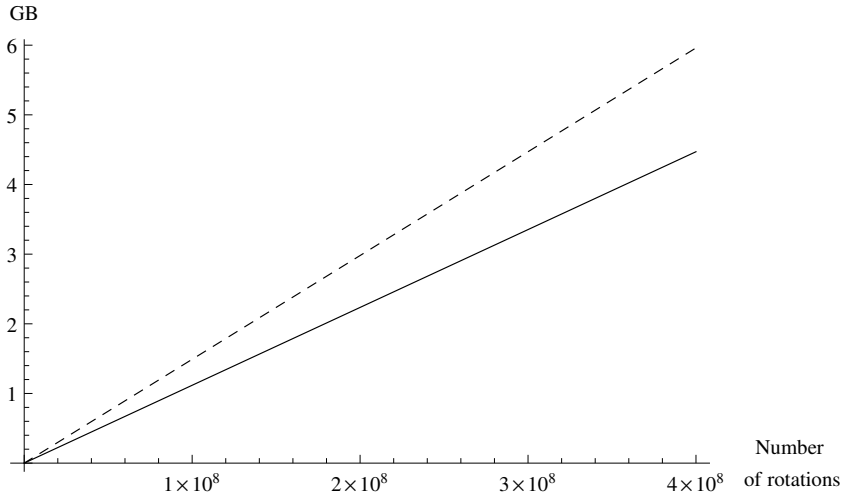
The details of the file writing and the buffer size determination are left to the user to optimize them for their platform. Instead, we are going to discuss the use of a compact representation of a rotation. The values of c and s can be represented in a single value ρ [56]. This means that a rotation can be stored in three rather than four elements using a floating point division to encode and a floating point square root to decode them (see Golub [27], §5.1.11, algorithms 5.1.9 and 5.1.10). The only advantage of the compact representation is the space saving (see Figure 5.3). It has to be considered if the space saving compensates the impact of the additional operations. In relative terms, compact representation supposes a 25% less of space required and one additional operation will increase the computational cost of a rotation by a $\approx 15\%$, in the case of the operation $Q^T b$ (since each non-compact rotation needs four products and two sums for this operation). In absolute terms, for example, in order to store 10^8 rotations, 1.12GB and 1.5GB will be required for compact and non-compact representations respectively. It is very likely that if 1.12GB are available, then 1.5GB will be also available and as a drawback, 10^8 floating point square roots will be additionally performed. In many applications we want to wait as little as possible to obtain the solution of a system, especially if it is repeatedly solved against different right hand sides, so it is desirable to

```
void QRDecomposition::compute_QtB(float* QtB){

  float temp_p, result_p, temp_e, result_e;
  GivensRotation g; // next rotation to apply

  // cond flag is false if EOF is reached, true otherwise
  bool cond = get_G_element( &g); // extraction of the next rotation

  while( cond ){ // while not EOF

    // rotate
    temp_e = QtB[g.e];
    temp_p = QtB[g.p];

    // new values are calculated
    result_p =  (g.c * temp_p) + (g.s * temp_e);
    result_e = -(g.s * temp_p) + (g.c * temp_e);

    // store the result
    QtB[g.p] = result_p;
    QtB[g.e] = result_e;

    // extraction of the next rotation
    cond = get_G_element( &g);

  }
  return;
}
```

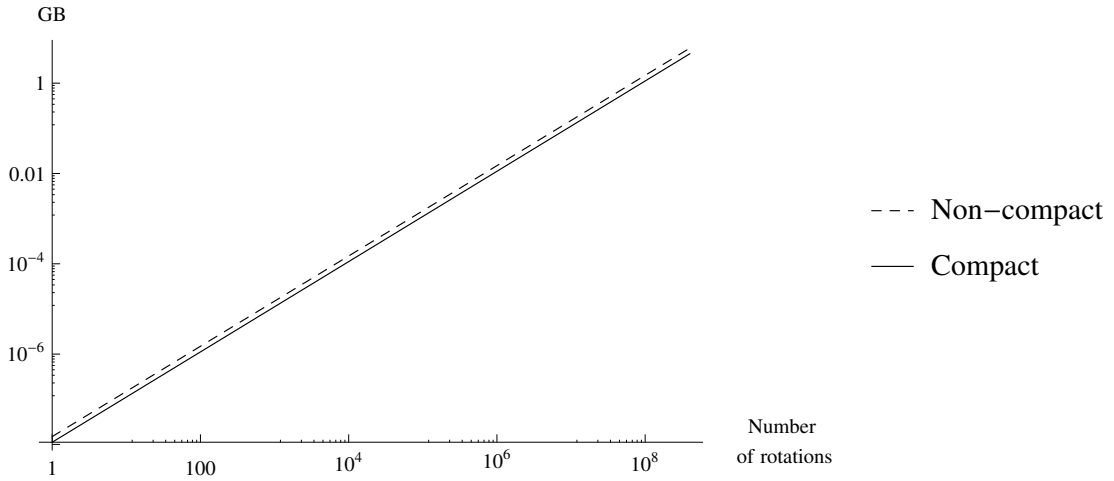
Code 5.4: Implementation in C++ of the computation of $Q^T b$. `float* QtB` must be a reference to the vector b that is used as input and is overwritten with $Q^T b$ and `get_G_element()` performs a buffered reading from disk. This method assumes that the class *QRDecomposition* already has a file pointer to the Q^T matrix file.

avoid as much computation as possible in the operation $Q^T b$. For these reasons we choose to store the givens rotations in non-compact representation.

In conclusion, we complemented our implementation of the QR decomposition with the implementation of the computation of a Givens rotation that assures continuity in the computation of r, the selection of a zeroing strategy that reduces the required space of intermediate results, the implementation of the rotation of two rows that introduces a zero in a matrix, a data structure for the representation of the Givens rotations and the implementation of the rotation of two elements of a vector that computes $Q^T b$. Also, extra computation in the $Q^T b$ process is avoided selecting the non-compact representation of the Givens rotations.



(a) Linear scale.



(b) Logarithmic scale.

Figure 5.3: Space needed to store Givens rotations in compact and non-compact representations. Both representations need to store the two row indexes. Non-compact representation uses two `float` for c and s (16 Bytes) and compact representation uses one `float` for ρ (12 Bytes).

Chapter 6

Fill-in in the QR decomposition

The *fill-in* accounts, in general, for the creation of new non-zeros in a sparse matrix during a transformation. In §5, we used Givens rotations to gradually transform our system matrix A in an upper triangular matrix R, producing fill-in. As a sparse matrix has a considerable amount of zero elements, it is expected that computational costs and space requirements will be eased. However, the fill-in can equate these costs to those with a full matrix. In this chapter, we are going to discuss several strategies to reduce the impact of fill-in during the QR decomposition of our system matrix, using Givens rotations. We are going to apply four of the most commonly used strategies to reduce the fill-in and compare them with an *ad hoc* strategy for our system matrix. To the best of our knowledge, this is the first work that presents a specialized fill-in reduction strategy for sparse matrices derived from the CT image reconstruction problem.

6.1 Standard fill-in reduction

The problem of fill-in reduction is to find a row and column ordering such that the fill-in produced during the factorization is minimized. Minimizing the fill-in is NP-complete [57], so, in practice, different strategies are used to reduce it.

The success on the fill-in reduction depends on the propagation of the fill-in caused by the transformation and the structure of the sparse matrix. A Givens rotation produces fill-in as follows: the sparsity pattern of each row involved in the rotation will be the union of both rows sparsity pattern, except for the zeroed element (see Figure 6.1). The structure of the sparse matrix may promote the occurrence of fill-in with sparsity patterns that are heavily filled by the transformation (see Figure 6.2).

Regarding the QR decomposition, in which an upper triangular matrix R is computed, the term *fill-in* is usually used to refer to new non-zero elements that

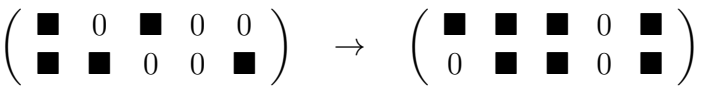
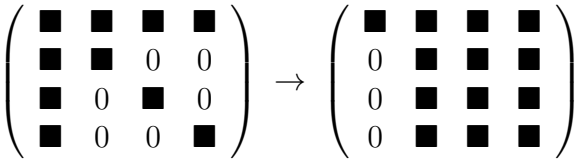
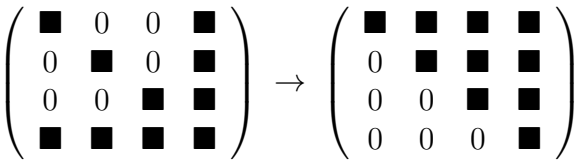


Figure 6.1: Example of fill-in produced in two rows by a Givens rotation. Left and right represent before and after the rotation respectively. For both rows, the resulting sparsity pattern is the union of both sparsity patterns.



(a)



(b)

Figure 6.2: Example of fill-in produced in the arrow sparse matrix. There are two different sparse matrix structures illustrated: up arrow structure (a) and down arrow structure (b). In each illustration, left and right represent before and after the application of three rotations respectively.

occur in the upper triangular part of A and *intermediate fill-in* refers to new non-zero elements that occur below the main diagonal of A and will become zero again later, during the decomposition. Hereinafter, we will use this distinction because these two types of new non-zeros have different effects. Essentially, fill-in increases the required space for intermediate and final results (see Figure 6.2b), while, intermediate fill-in increases the required space for intermediate results and the number of rotations needed to transform A in R (see Figure 6.2a). Due to intermediate fill-in, matrix of Figure 6.2a is at 50% of the QR decomposition, and has already became a full matrix. Instead, matrix of Figure 6.2b has already completed the QR decomposition and only suffered fill-in above the main diagonal.

6.1.1 Standard fill-in reduction strategies

There are three basic approaches to minimize fill-in: nested dissection, minimum degree, and band reduction (see [29] §7 for details). Nested dissection separates the sparse matrix in blocks, then, recursively, tries to separate each block again, so the fill-in will be contained in small submatrices. Minimum degree focuses in the structural properties of rows (which amount of fill-in will they generate, for example), so the firstly eliminated rows do not generate much fill-in. The band reduction strategy tries to organize non-zero elements in a band, so fill-in is avoided below and above the band.

The structure of our sparse matrix is not appropriated for strategies related with nested dissection. We tried several nested dissection orderings, such as clique and node separators (see [58] for details), without being able to reduce the fill-in. However, the non-zero elements of our matrix can be rearranged in a *band*. This band is drawn between the upper left and the lower right corners of the matrix (not necessary along the main diagonal) and its bandwidth w is measured as the maximum value of $j_{lst} - j_{fst} + 1$ of all rows, where $a_{ij_{fst}}$ and $a_{ij_{lst}}$ are the first and last non-zero elements respectively. The idea behind the band ordering is to obtain a band as narrow as possible, automatically skip the rotations below the band and try to keep the upper right corner without fill-in.

We selected the three better-performing (in our case) band ordering strategies and the *Modified Minimum Degree* to test with our system matrix, as these four approaches gave the best results for the standard fill-in reduction. Fill-in reduction strategies reviewed here rely on graph theory to compute different row and column orderings and to do so, they process matrices as graphs. A graph is a pair $G = (V, E)$, where V is a set of nodes and E is a set of pairs $(v_i, v_j) \mid v_i, v_j \in V$ that represents edges between nodes. Let A be a sparse matrix, the graph nodes represent the rows and columns of A and there will be an edge connecting nodes v_i and v_j of G for each non-zero element $a_{ij} \in A$. Provided that A is symmetric, G is built from A directly. For non-symmetric matrices, such our case, G is built from

$$B = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$$

There are some terms from graph theory that are needed for the fill-in reduction strategies. Two nodes of G are said to be *adjacent* if they are connected by an edge. Nodes v_i and v_j are adjacent if $a_{ij} \neq 0$. The *degree* of a node v_i is the number of edges that contain v_i. As a node can represent a row or a column, its degree is the number of non-zero elements in the corresponding row or column of A.

The following four algorithms have been tested with our system matrix. We introduce them with a brief summary:

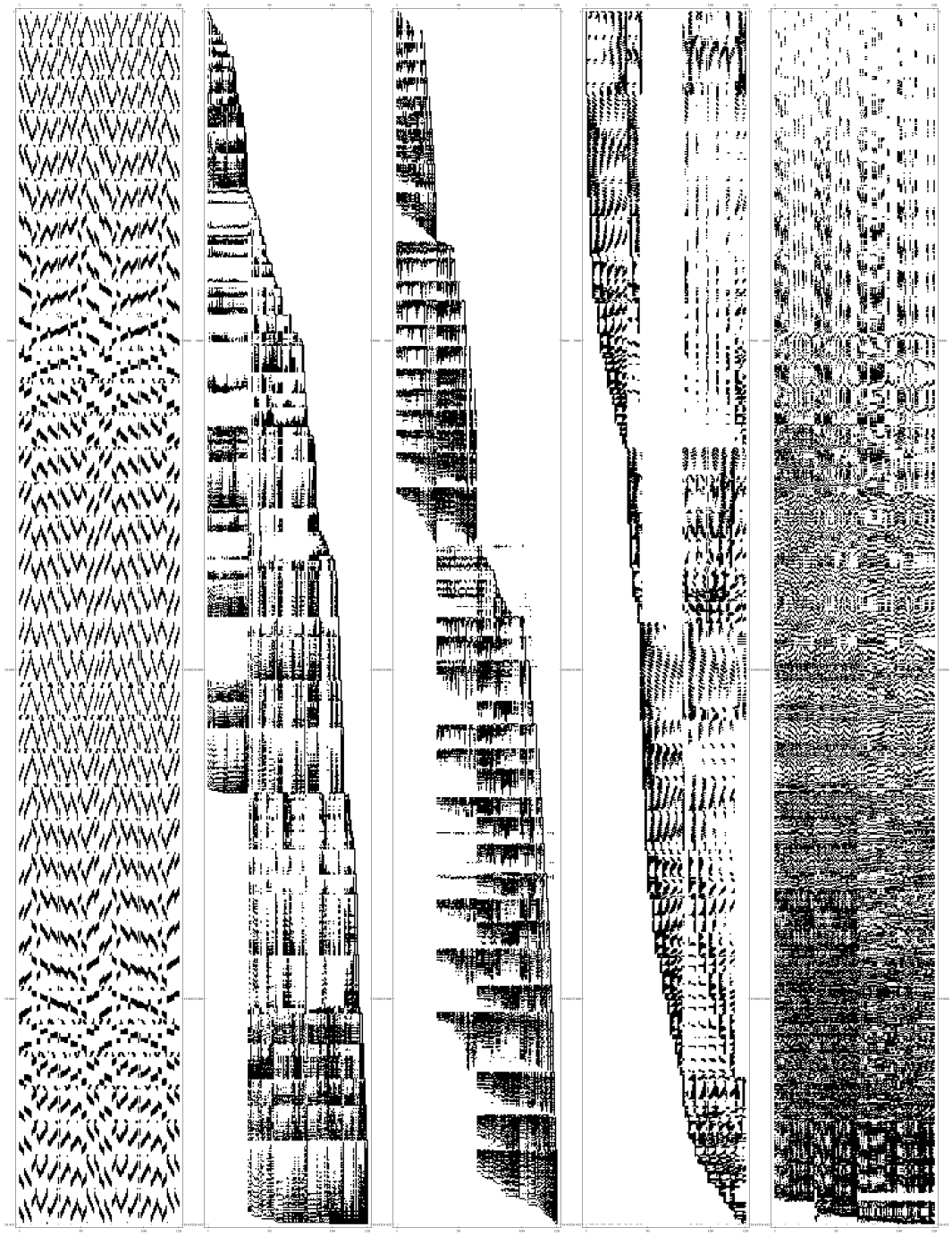
- The *Cuthill–McKee* ordering [59] selects a starting node which might be a node of minimum degree. Then, the nodes adjacent to this node are selected in the order of their increasing degree. The procedure is repeated starting with the nodes adjacent to ones selected, in sequence, until all nodes have been selected. The *Reverse Cuthill–McKee* ordering reverses the previous computed ordering because often this improves the fill-in reduction [60].
- The *Minimum Bandwidth* ordering is a variant of the Reverse Cuthill–McKee algorithm which breaks the tie between nodes that have the same degree, based on ordered sequence of already selected adjacent nodes.
- The *King's* ordering [61] is a variant of the Reverse Cuthill–McKee algorithm which instead of selecting the nodes based on their total degree, nodes are selected in the order of how many already selected adjacent nodes they have.
- The *Minimum Degree* ordering selects a node of minimum degree, eliminates it and updates the degrees of the uneliminated nodes. The procedure is repeated until all nodes are eliminated. The *Modified Minimum Degree* ordering [62] improves the overall performance of the previous algorithm by eliminating a subset of nodes, all at the same time.

An efficient implementation of the Minimum Bandwidth ordering is available in [63] and simplified pseudocode of the rest is available in [64]. The result of the application of these ordering strategies is represented in Figure 6.3.

6.1.2 Standard fill-in reduction performance

We consider three parameters in order to assess the effectiveness of these strategies with the system matrix: the fill-in, the intermediate fill-in and the number of rotations needed to perform the QR decomposition. A low resolution system matrix has been used for this comparison, but similar results have been obtained with higher resolutions. The test matrix models a system with a flat panel consisting of 32×32 detectors, 40 views and $8 \times 8 \times 8$ voxels. Sizes of the resulting matrix are 18432 rows, 128 columns, and 151592 non-zero elements.

First, we will focus on the fill-in. This parameter can be easily evaluated as the number of non-zero elements in R once the QR decomposition is finished. Results for our test matrix are illustrated in Figure 6.4. Results are, at best, as those obtained without any ordering. Specifically, Reverse Cuthill–McKee and King's ordering generate more fill-in than the system matrix with no ordering, but with matrices as narrow as ours, the differences between shown strategies ($5000 \sim 7000$) are almost negligible as the process started with a system matrix of ~ 150000 non-zero elements. Therefore, this parameter does not have a significant impact in final and intermediate results size.



(a) (b) (c) (d) (e)

Figure 6.3: The system matrix as it was generated, with no ordering applied (a) and ordered with Reverse Cuthill–McKee (b), Minimum Bandwidth (c), King's (d), and Modified Minimum Degree (e).

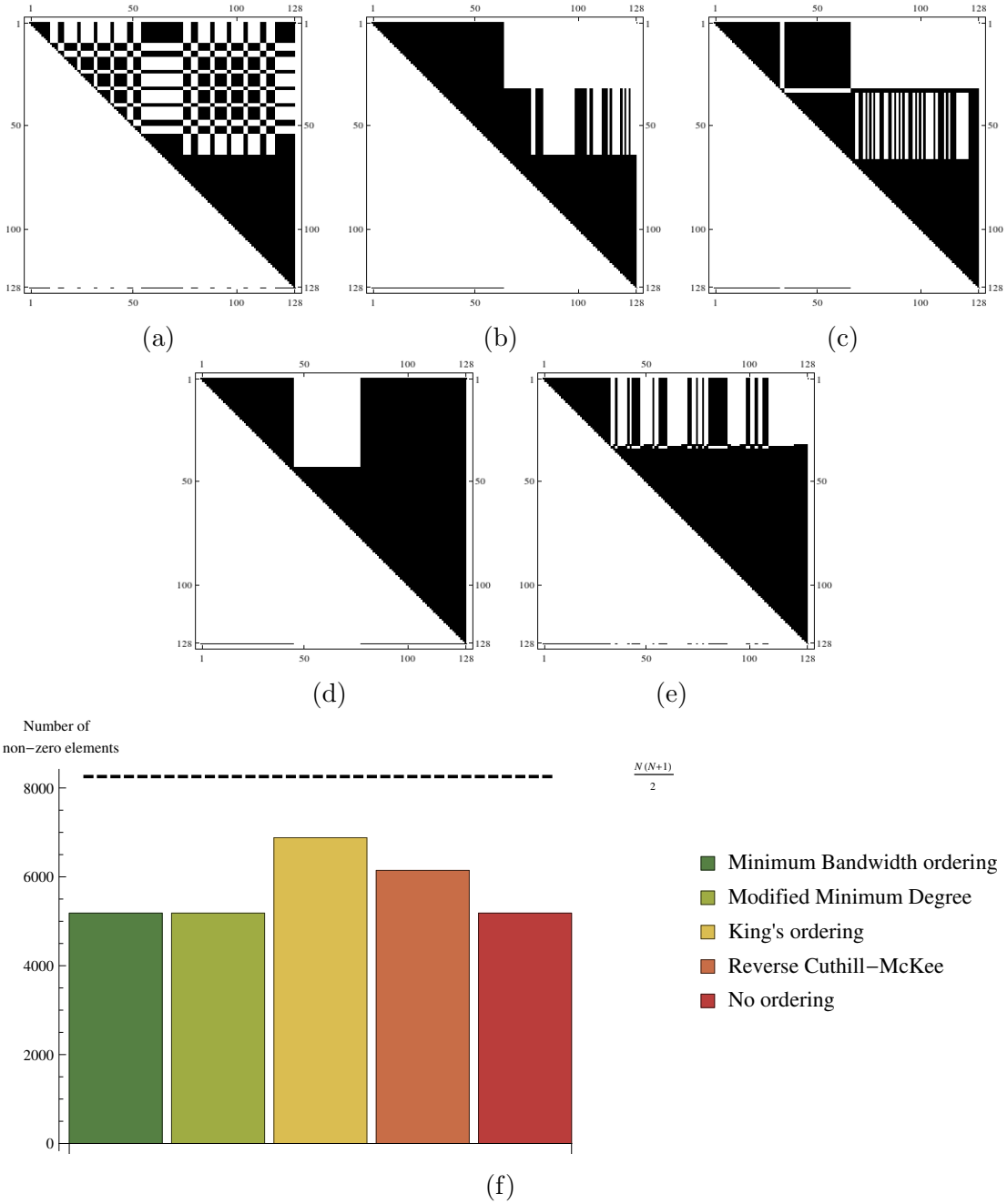


Figure 6.4: The R factor of the QR decomposition of a system matrix with no ordering applied (a) and ordered with Minimum Bandwidth (b), Modified Minimum Degree (c), King's (d), and Reverse Cuthill–McKee (e) algorithms. For better comparison between matrices, the number of non-zero elements in each R is presented (f). Dashed line indicates the total number of non-zero elements of an upper triangular full matrix.

The intermediate fill-in is a more limiting parameter, as it will be the most space consuming during the decomposition. This is due to the narrowness of our sparse matrix: fill-in produced below the main diagonal might be of three orders of magnitude greater than the fill-in produce above. Even if intermediate fill-in is eliminated after, during the reduction, it had to be stored in the sparse matrix. In other words, with our matrix dimensions, it is possible to work with a full triangular matrix, but it is not possible to work with all elements below the main diagonal.

We evaluated the intermediate fill-in by monitoring the number of non-zero elements in the matrix in each step of the process. These measurements consist of both the fill-in and the intermediate fill-in, but as said before, intermediate fill-in is almost entirely the overall fill-in. Results for this parameter are illustrated in Figure 6.5. In both Figures 6.5a and 6.5b, the number of non-zero elements generated in the matrix without ordering has been plotted for reference. Strictly from the point of view of the required space, the peak value for each ordering is its required space for the QR decomposition. Also, the entire curve is interesting in order to assess possible overhead creating and destroying matrix elements, or additional costs in the rotations, as the element access time depends on the total number of non-zero elements in the accessed rows. For example, suppose that, with the same peak value, two different curves are presented: one that quickly grows to an almost constant number of non-zeros and other that stays low the majority of the process and shows a narrow *pulse*. With the same peak value, the second curve describes a better performance than the first, because it shows reduced access costs during the majority of the process. The measured curves have shapes that are not as obvious as the example presented. In this sense, the area below a curve (presented in Figure 6.5c) is a good quantifier for comparison between them.

Regarding to the number of rotations needed for the entire QR decomposition, the results for each fill-in reducing strategy are compiled in Figure 6.6. As the Q factor is saved implicitly (consisting of a list of four values representing a rotation), the evaluation of this parameter is a matter of counting the rotations contained in the file. This parameter reflects the computation time needed for the QR decomposition and, also, represents the space needed to store the Q matrix, which in high resolution systems is in the order of Giga Bytes. Not only all strategies exceed the number of rotations needed to complete the process without any ordering, but in all cases, needed rotations are closer to their upper bound than to their lower bound.

All approaches considered have similar or higher results in the evaluated parameters than applying no orderings. These algorithms have proven its effectiveness with general sparse matrices, but in our case do not produce any effective fill-in,

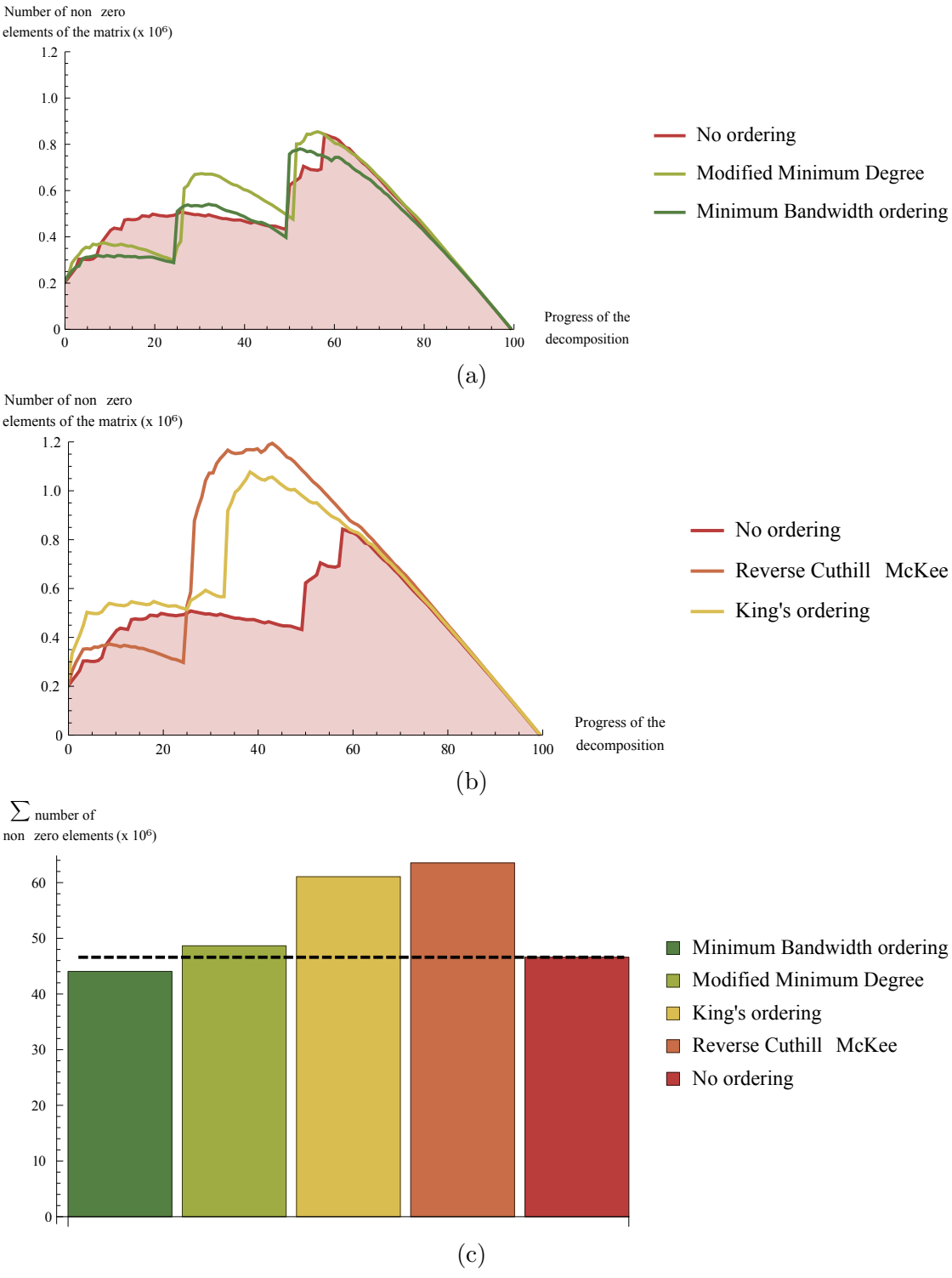


Figure 6.5: Number of non-zero elements during the QR decomposition of our system matrix with Modified Minimum Degree (a), Minimum Bandwidth (a), Reverse Cuthill–McKee (b), and King's orderings (b). For better comparison between curves, their totalized number of non-zero elements is presented (c). Dashed line indicates the totalized number of non-zero elements of the QR decomposition of our system matrix with no ordering.

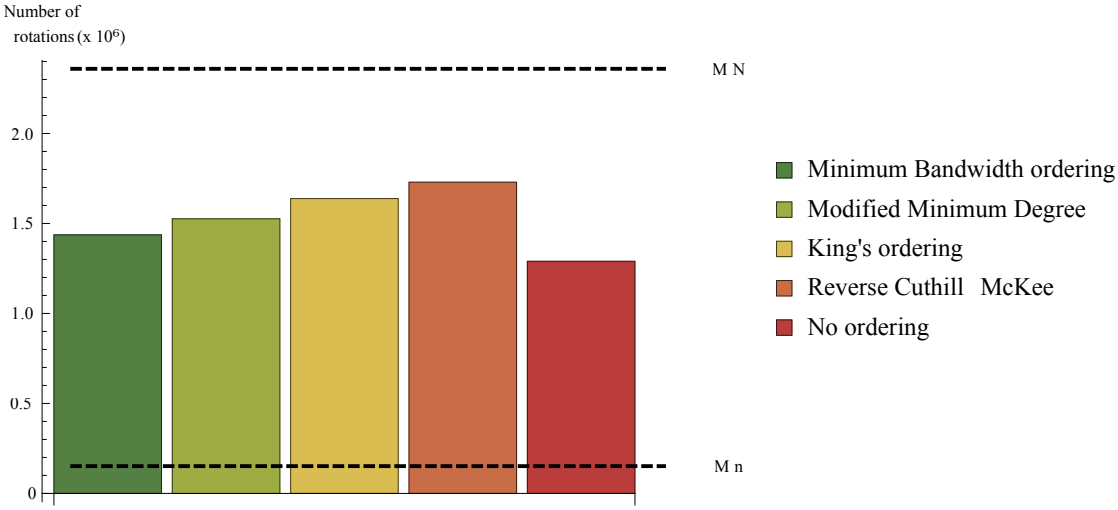


Figure 6.6: Number of Givens rotations needed for the entire QR decomposition depending on which ordering has been applied. Dashed lines represent theoretical upper and lower bounds of the number of rotations needed, where M is the number of rows, N is the number of columns, and n is the average number of non-zero elements in each row.

intermediate fill-in or number of rotations reduction. This is due to the structure and narrowness of our system matrix combined with band orderings with high bandwidth. When the QR decomposition reaches certain rows, the intermediate fill-in propagates throughout most of the matrix (it is appreciated as sudden growths in the number of non-zeros in Figures 6.5a and 6.5b).

The performance of classical approaches is not affordable in practice. The intermediate fill-in is not properly reduced in the QR decomposition of our sparse matrix and along with its extra space required to store the intermediate results, this extra intermediate fill-in increases the total number of rotations of the factorization. Therefore, this motivates the development of an *ad hoc* heuristic to reduce intermediate fill-in.

6.2 Fill-in reduction alternative

The standard fill-in reduction approaches do not reduce intermediate fill-in in our matrix structure. Provided that the sparse matrix is ordered as a band matrix, during the decomposition, the first non-zero element of a row will be eliminated as the rest of the elements will probably generate intermediate fill-in. An heuristic has been developed to limit the propagation of the intermediate fill-in to approximately the bandwidth.

6.2.1 Heuristic strategy

Suppose that our system matrix $A \in \mathbb{R}^{M \times N}$ is ordered as a band matrix with a bandwidth w in the sense of §6.1, during the decomposition, intermediate fill-in can be bounded if the last non-zero elements of a row involved in a rotation do not increase w. Suppose we have a group of m rows such that $m > w$ and for each of these rows, its last non-zero element is in the same column. Let $G \in \mathbb{R}^{m \times N}$ be a submatrix composed of these m rows. Then, G also has a bandwidth $w_g \leq w$ and has an interesting structural property: Givens rotations required to reduce G to upper triangular form will not increase w. We call G a *group*. Intermediate fill-in will still occur in G, but in this case with an upper bound of

$$mw \tag{6.1}$$

elements, due to the criteria applied to column indexes of last non-zero elements of its rows. Moreover, provided that we started the reduction to triangular form of G from its first column that has any non-zero element (ignoring the columns that are entirely zero), after the reduction, at most,

$$\frac{w(w+1)}{2} \tag{6.2}$$

non-zero elements will remain in G and $m - w$ rows will have been eliminated. These rows will never be part of a rotation or produce any fill-in.

Suppose now that we have two groups G_1 and G_2 of m_1 and m_2 rows respectively, with w_1 and w_2 bandwidths respectively, j_1 and j_2 are its rightmost columns that contain any non-zero element respectively, $j_1 \leq j_2$ and both have been reduced to triangular form. If we join them in

$$G_{12} = \begin{pmatrix} G_1 \\ G_2 \end{pmatrix}$$

based on Equation 6.2, in the worst case G_{12} will have $\frac{w_1(w_1+1)}{2} + \frac{w_2(w_2+1)}{2}$ non-zero elements and reducing G_{12} to triangular form with Givens rotations will produce no intermediate fill-in and a fill-in of

$$w_1(j_2 - j_1) \tag{6.3}$$

elements in the reduced G_{12}. Provided that we performed the reduction to triangular form of G_{12} ignoring the columns that are entirely zero, we have completely eliminated all non-zero rows $(w_1 + w_2)$ but the non-zero rows of G_1 (w_1) plus a number of rows equal to the difference between the rightmost non-zero columns of both groups $(j_2 - j_1)$

$$(w_1 + w_2) - (w_1 + j_2 - j_1) = w_2 + j_1 - j_2$$

which is almost the entire G_2 if j_1 and j_2 are close.

Consider that the reduction to triangular form of A will be composed of the reductions of G_i, then, the fill-in produced for the different G_i account as intermediate fill-in produced in the reduction of A. Therefore, the entire reduction of these two groups into one, in triangular form, has produced a number of elements of intermediate fill-in with an upper bound of

$$m_1 w_1 + m_2 w_2 + w_1(j_2 - j_1)$$

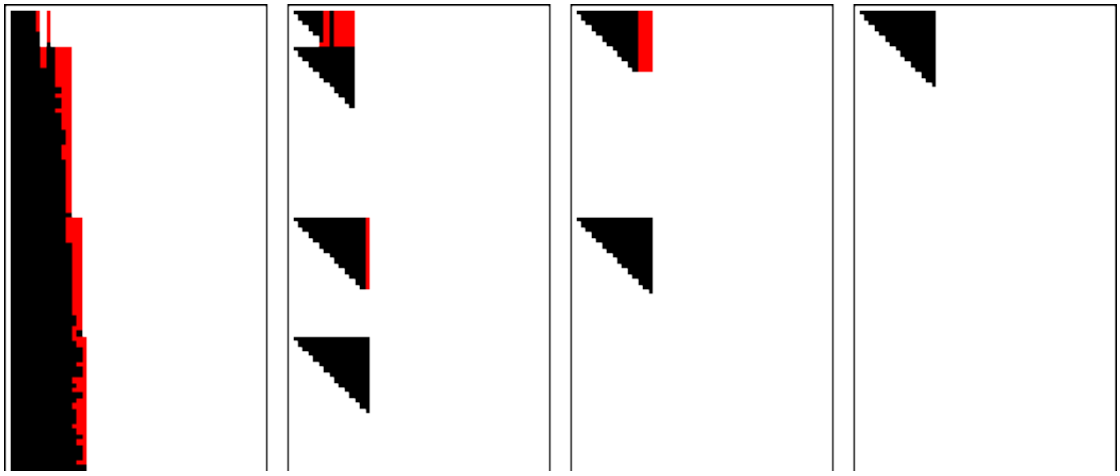
based on its both bandwidths (see Equation 6.1) and the fill-in produced between its rightmost non-zero columns (see Equation 6.3).

An optimal group would have m as large as possible and w as small as possible, so almost the m rows are completely eliminated producing little intermediate fill-in. Once reduced, an optimal pair of groups would have close j_1 and j_2 for the same reason. However, it is too optimistic to ask for the last non-zero element in all rows of a group to be in the same column. Instead, j will be considered the highest column index of the elements of the group and the rest of the rows will be asked to have its last non-zero element *close* to j (see Figure 6.7a for an illustration of the groups, joined groups reduction, and its associated intermediate fill-in).

As an example of our heuristic workflow, we present in Figure 6.7 the reduction to triangular form of the top part of a low resolution sparse matrix. In Figure 6.7a is shown the process driven by the group making and in Figure 6.7b is shown the process without it. The intermediate fill-in produced in the next step of each case is highlighted in red, while in Figure 6.7c the fill-in avoided by the row grouping is highlighted in green. The effect of the intermediate fill-in reduction is mitigated, as we are only considering here the top part of the matrix, but it is sufficiently large to be appreciated. Approximately a 25% of the intermediate fill-in is avoided if groups are used. In Figure 6.7c can be observed in the two green zones that intermediate fill-in has not been propagated outside of the groups in the first step and this propagation could be larger if a greater section of the matrix would be considered. In addition, and thanks to the band ordering, the intermediate fill-in produced by the paired groups reduction is scarce.

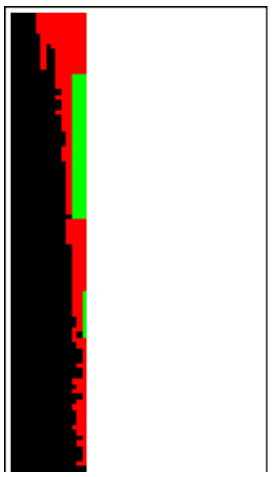
6.2.2 Heuristic implementation

The first step of the implementation of our heuristic is the band ordering. The aim is to compute an ordering that organizes the non-zero elements of A in a band. This ordering locally holds that last non-zero element of rows are *approximately* in the same column. This band ordering is implemented in Code 6.1 and makes use of the non-zero structure of the sparse matrix, contained in a class `MatrixStructure`.



(a)

(b)



(c)

Figure 6.7: Fill-in produced during the reduction to triangular form of the first rows of our sparse matrix ordered with our heuristic. Four groups (a) and only one group (b) have been considered. In (a) and (b), each frame represents a stage of the process and it has been highlighted in red the fill-in that will be produced in the next step. In (c), a single frame is presented with the original sparse matrix (black), all the fill-in produced in (a) (red) and all the fill-in avoided in (a) regarding (b) (green). In this case, (a) produces a $\sim 25\%$ less of fill-in than (b).

This class contains a list of all rows. Each element of this list is composed of its index, its number of non-zero elements and a link to another list with its column indexes.

```
void Orderings::band_ordering(MatrixStructure& matrix){

  matrix.first_nzidx_ordering(); // initial row reordering

  matrix.transpose();
  matrix.first_nzidx_ordering(); // initial column reordering
  matrix.transpose();

  for(int i = 0; i < (50); ++i){ // iterative process

    matrix.first_nzidx_ordering(); // row reordering

    matrix.transpose();
    matrix.node_centroid_ordering(); // column reordering
    matrix.transpose();
  }

  matrix.first_nzidx_ordering(); // final row reordering

  return;
}
```

Code 6.1: Implementation of our band ordering heuristic in C++. Class `MatrixStructure` received as input contains the row and column indexes of non-zero elements. Its method `transpose()` swaps the row and column indexes of these elements. `first_nzidx_ordering()` method, reorder the matrix rows in their first non-zero column index ascending order. `node_centroid_ordering()` method is a modification of the standard node centroid algorithm, but without considering the current row index for the new row ordering. We experimentally determined that beyond 50 iterations there is no further improvement in bandwidth reduction.

We used the `MatrixStructure` class for two reasons: fast row swapping and to ease the `transpose()` method. Regarding the row swapping, this method swaps the number of non-zero elements and the link to the list with its column indexes. Regarding the `transpose()` method, all of number of non-zero elements of transposed rows are computed iterating once over the *old* rows list with their column indexes. The required space for the new structure (to hold the transposed matrix) is then allocated and the list of all transposed rows with their column indexes are computed. In one iteration over the *old* rows list with their column indexes, each *old* row index is stored as a *new* column index in its corresponding list regarding the *old* column index. This process has both time and space complexities of $\mathcal{O}(Mn)$, where M is the number of rows and n is the average number of non-zero elements in rows.

The band ordering makes use of two different orderings. The `first_nzidx` ordering algorithm (from *first non-zero index*) that sorts each row by the column index of its first non-zero element, in ascending order and a modified `node_centroid` ordering algorithm [65] that sorts each row by the average of their column indexes and the current row index itself, in ascending order. Our modification does not consider the current row index for the new row ordering (see Code 6.2).

The band ordering procedure has two parts: an initialization ordering of rows and columns and an iterative process. The initialization sorts the non-zero elements in a *wide* band and the iterative part reduces it as much as possible. The advantage of this ordering is not a high bandwidth reduction but the local alignment of the last non-zero element of rows around a column (see Figure 6.8). This allows the determination of j locally (see §6.2.1) as the highest column index of the nearby rows, which will have its last non-zero element *close* to j.

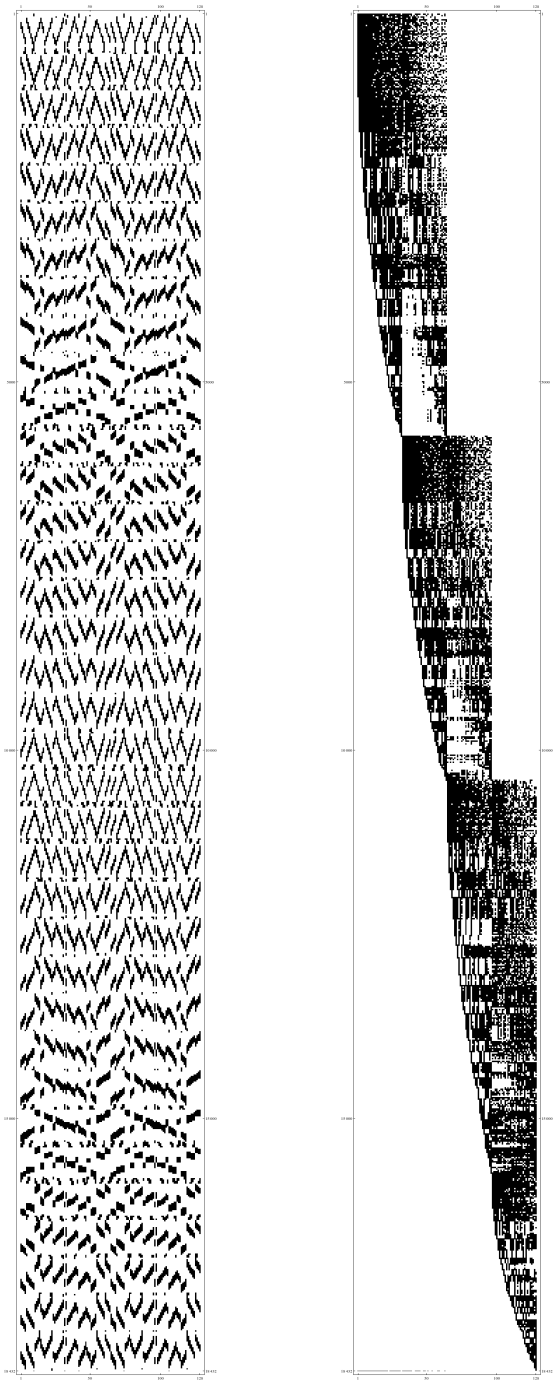
Once we have the band ordering, the next step of the implementation of our heuristic is the group making. We developed a simple greedy algorithm to determine j and select a group of nearby rows as large as possible, without rearranging the row order. For better performance, this algorithm has a parameter that allows some of last non-zero element of nearby rows to be larger than j. This increases considerably the number of rows selected per group while the user stills in control of the degradation of the bandwidth, assessing the intermediate fill-in that this bandwidth degradation would cause. It was experimentally determined that a 1% of the total columns (for our matrix) is a good compromise between the additional intermediate fill-in produced and the additional rows added per group. The group making algorithm works as follows:

1. Determines the next group free row, based on last row of previous group.
2. Selects a minimum number of rows, based on the width of the first row, where the width of a row is $j_{end} - j_{ini} + 1$, being j_{ini} and j_{end} the column index of the first and last non-zero elements of the row respectively.
3. Determines j for this preliminary group of rows.
4. Extends the preliminary group of rows as follows: "add a new row while the column index of its last non-zero element is less or equal to j plus the bandwidth degradation parameter".
5. End when there are no more free rows.

The result of the algorithm is the number of groups created and an array such that the row index stored in its i-th position is the last row of the i-th group (see Code 6.3). This algorithm uses as input an array `row_info` that contains in its i-th position the first and last non-zero column index of the i-th row, in order to

```
// pair {row index, row weight}
struct w_t{
  int row; // row index
  float w; // row weight
};

// comparison functor for w_t type
bool order_for_w_t(const w_t& a, const w_t& b){
  return (a.w < b.w);
}

void Orderings::node_centroid_ordering(MatrixStructure& matrix){

  int number_of_rows = matrix.rows; // get the number of rows
  w_t* w_list = new w_t[number_of_rows]; // create a list for row scores
  int* new_row_ordering = new int[number_of_rows]; // new row index ordering

  for(int i = 0; i < number_of_rows; ++i){

    // initialization of the row score
    w_list[i].row = i;
    w_list[i].w = 0;

    // if there is any non-zero element ...
    if(matrix.number_nonzeros_in_each_row[i] != 0){

      // sum its column indexes ...
      for(int j = 0; j < matrix.number_nonzeros_in_each_row[i]; ++j){
        w_list[i].w += matrix.column_indices_per_row[i][j];
      }

      // and compute their average value as its score
      w_list[i].w = w_list[i].w / static_cast<float>(matrix.number_nonzeros_in_each_row
          [i]);
    }
  }

  // reorder rows by their score
  std::sort(w_list+0, w_list+number_of_rows, order_for_w_t);

  // extract row ordering
  for(int i = 0; i < number_of_rows; ++i){
    new_row_ordering[i] = w_list[i].row;
  }

  // apply row ordering
  matrix.set_row_ordering(new_row_ordering);

  delete[] w_list;
  delete[] new_row_ordering;
  return;
}
```

Code 6.2: Implementation in C++ of our modification of the node centroid algorithm. Our modification excludes the current row index for the row new ordering. Namely, in our version `w_list[i].w` is not updated with the value of `i` before its average value is computed. The final row order is stored in the `new_row_ordering` array and then applied to the input matrix. Also, the pair {*row index*, *row score*} with its comparison functor are included. The functor is provided to the `std::sort()` algorithm in order to obtain the new ordering.



(a) (b)

Figure 6.8: The system matrix as it is generated, with no ordering applied (a) and ordered with our heuristic (b).

avoid matrix lookups (its computation is excluded form Code 6.3 for simplicity). This group making algorithm has both time and space complexities of $\mathcal{O}(M)$, where M is the number of rows.

The final step in this implementation is the definition of a group pair making policy. Once all groups have been reduced they must be joined in pairs, so the reduction to triangular form can be continued. Consider three groups, G_a with j_a, G_b with j_b and G_c with j_c, such that they are contiguous. The best pair for G_b will be one of its two contiguous neighbors (the band ordering make them have the smallest distance between its j) and it will be a matter of choose G_a or G_c based on which $|j_a - j_b|$ or $|j_c - j_b|$ is lower.

6.2.3 Heuristic performance

We consider the same three parameters as in §6.1.2 to assess the performance of our heuristic and compare it with the standard fill-in reduction strategies: the fill-in, the intermediate fill-in and the number of rotations needed to perform the QR decomposition. The same low resolution system matrix has been used for this comparison and also, similar results have been obtained with higher resolutions.

In the evaluation of the performance parameters, two versions of our heuristic have been included: one, with our heuristic making groups as large as possible (five groups will be generated for this test matrix) and other, with our heuristic forced to make no groups (the triangular reduction will be of all the matrix at once). It is interesting to highlight the importance of the group making and discard that the ordering itself reduces any kind of fill-in. Moreover, the corresponding results of the standard fill-in reduction strategies have also been included in figures below for comparison.

The fill-in results of our test matrix are illustrated in Figure 6.9. Both heuristic versions produced the same non-zero pattern for the R factor. Even if this R seems sparser than those obtained with previous strategies, it has the same number of non-zeros present in the better performing standard strategies.

Nevertheless, the intermediate fill-in is considerably reduced by our heuristic (see Figure 6.10). The version of our heuristic without group making presents poor results comparable to those obtained by the standard approaches. However, the group making effectively contains the spread of intermediate fill-in over the matrix. The number of non-zero elements of the matrix has a distinctive evolution when the group making is used: until approximately the 60% of the decomposition, the bandwidths of the different groups are progressively filled as they are reduced to triangular form, then, the number of non-zero elements drops and remain low while the reduced groups are joined in pairs and reduced again. The fill-in produced in this final part of the process is remarkably low. Moreover, the first part of the evolution of this curve is smoother than the other approaches. This smoothness

```
void Orderings::make_groups(const MatrixStructure& matrix){

  int number_of_rows = matrix.rows; // get the number of rows
  int number_of_columns = matrix.columns; // get the number of columns
  double inflation = 0.01; // allowed bandwidth degradation percentage
  int mark_width = static_cast<int>( floor( static_cast<double>(number_of_columns) *
      inflation));
  int width, j, mark;
  bool sentinel;
  // group limits inicialization
  number_of_groups = 0;
  for(int i = 0; i < number_of_rows; ++i){
    last_row_of_group[i] = -1;
  }

  // overall starting point
  int starting_row;
  int ending_row = -1;
  while( ending_row < (number_of_rows - 1) ){ // beginning of the "main loop"
    // current group starting point
    starting_row = ending_row + 1;
    // the width of the first row will be the minimum number of rows of the group ...
    width = row_info[starting_row].last_nozero - row_info[starting_row].first_nozero +
        1;
    // but taking care of not exceeding matrix dimensions
    ending_row = std::min(starting_row + width, number_of_rows - 1);
    // compute the leftmost non-zero column index of the group "j"
    j = row_info[starting_row].last_nozero;
    for(int i = starting_row; (i < number_of_rows) && (i <= ending_row); ++i){
      if(row_info[i].last_nozero > j){
        j = row_info[i].last_nozero;
      }
    }
    // this mark defines the criteria to enter in the current group
    mark = j + mark_width;
    // now try to increase the number of rows in the current group
    // with rows that meet previous criteria
    sentinel = true;
    while(sentinel){
      // if next row is inside the matrix dimensions and meets previous criteria ...
      if( (ending_row < (number_of_rows - 1)) &&
          (row_info[ending_row].last_nozero <= mark) ){
        // ... it can join current group
        sentinel = true;
        ++ending_row;
      // search for new rows is over
      } else {
        sentinel = false;
      }
    }
    // current group is completed, so we save its last row
    last_row_of_group[number_of_groups] = ending_row;
    // in next "main loop" we will define the next group
    // and its first row will be current ending_row + 1
    ++number_of_groups;
  }
  return;
}
```

Code 6.3: Implementation in C++ of the algorithm that assigns rows to groups, with a cost $\mathcal{O}(M)$, where M is the number of rows. When finished, this algorithm has generated `number_of_groups` groups (from G_0 to $G_{\texttt{number\_of\_groups}-1}$). Group G_0 starts in row 0 and ends in row `last_row_of_group[0]` and Group G_i starts in row `last_row_of_group[`i`-1]` and ends in row `last_row_of_group[`i`]`.

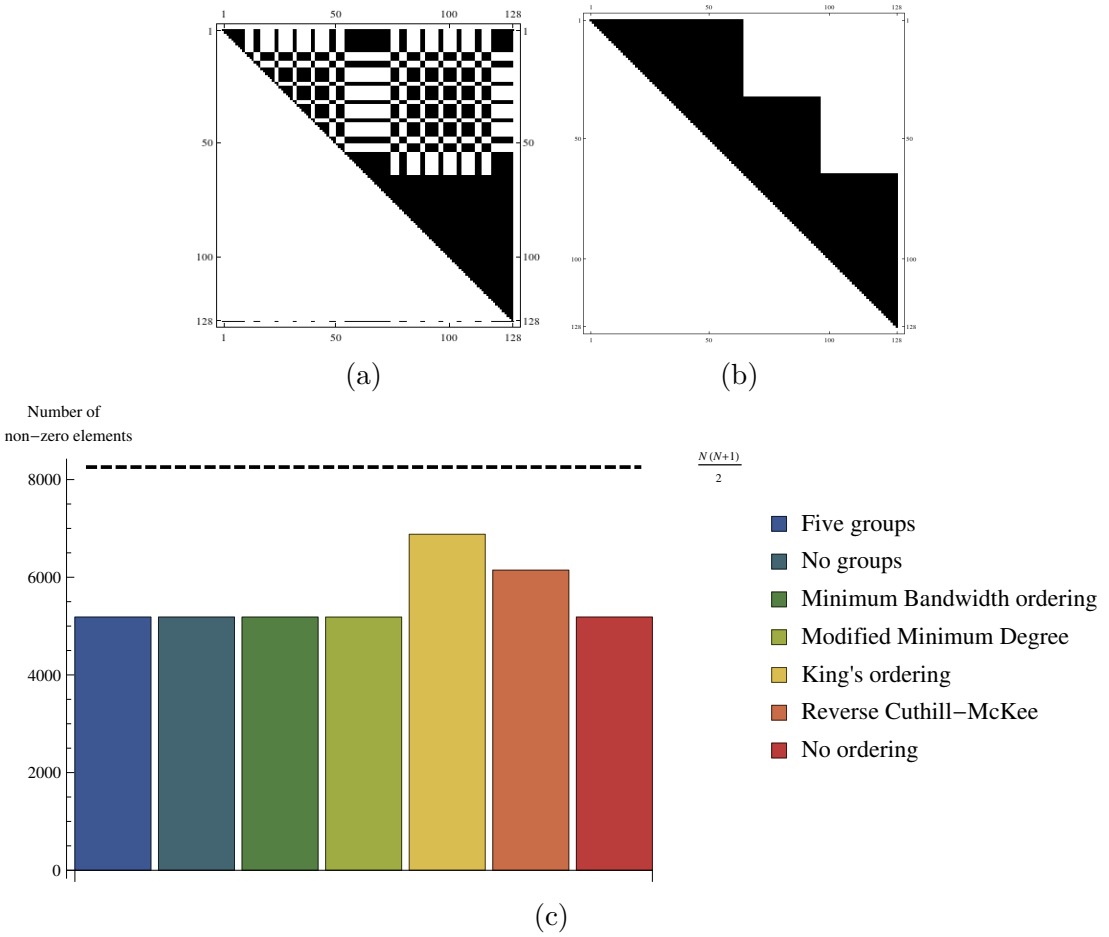


Figure 6.9: The R factor of the QR decomposition of a system matrix with no ordering applied (a) and ordered with our heuristic (b). For better comparison between matrices, the number of non-zero elements in each R is presented (f). *Five groups* represents the results of our heuristic making groups as large as possible. *No groups* represents the results of our heuristic forced to make no groups, and corresponds to the triangular reduction of the entire matrix at once. Moreover, the corresponding results of the standard fill-in reduction strategies have been also included for comparison. Dashed line indicates the number of non-zero elements of an upper triangular full matrix.

will benefit the QR decomposition algorithm, as it will not face massive memory allocations or intense filled rows. Finally, the maximum value reached by this curve reveals the minimum space requirements of all studied strategies.

The results of the number of rotations needed for the entire QR decomposition are shown in Figure 6.11. These results confirm those obtained with the number of non-zero elements: the ordering of our heuristic itself does not reduce the number of rotations needed, but the use of group making reduces it compared to the rest of strategies and brings it closer to the lower bound than to the upper bound. This reduction will not only affect the time cost of the QR decomposition, but, also, the space needed for the Q storage and the time cost of the $Q^T b$ product.

Unfortunately, the intermediate fill-in reduction obtained with our heuristic is not sufficient for the practical QR decomposition of systems of high resolution. Even with this reduction, the time needed for the decomposition, the space required or the size of the generated Q file are excessive for its use in commercially available computers. Nevertheless, our heuristic brings the possibility of performing the QR decomposition in bigger low resolution systems compared with the standard fill-in reduction strategies or no ordering at all.

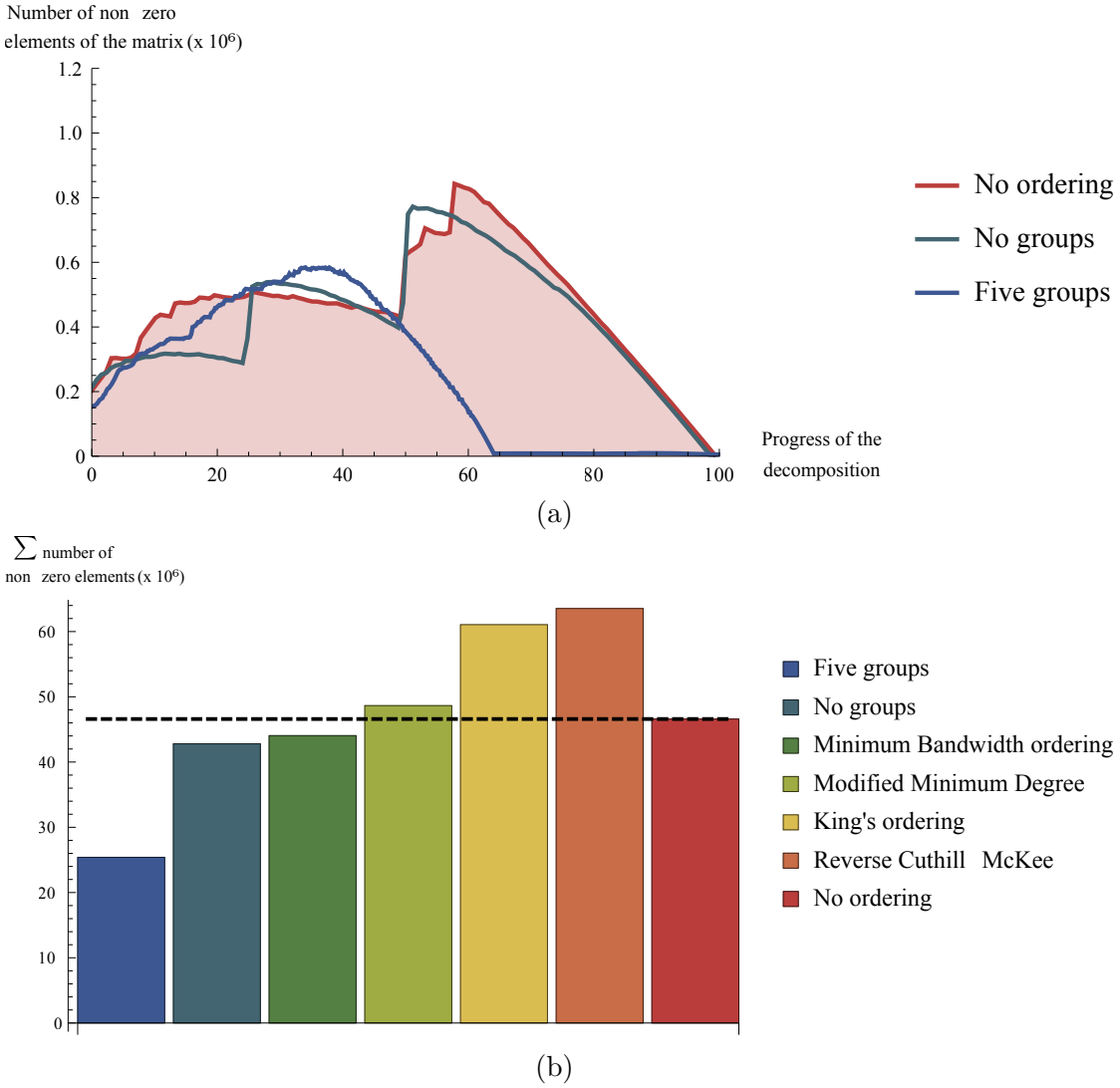


Figure 6.10: Number of non-zero elements during the QR decomposition of our system matrix with our heuristic (a). For better comparison between curves, their totalized number of non-zero elements are presented (b). Moreover, the corresponding results of the standard fill-in reduction strategies have been also included for comparison. Dashed line indicates the totalized number of non-zero elements of the QR decomposition of our system matrix with no ordering.

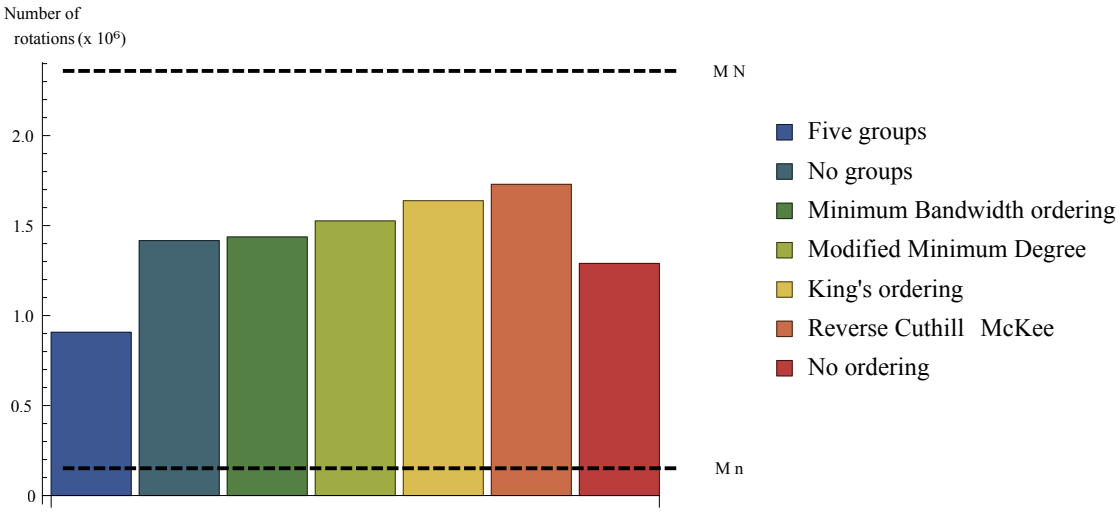


Figure 6.11: Number of Givens rotations needed for the entire QR decomposition depending on which ordering has been applied. Dashed lines represent theoretical upper and lower bounds of the number of rotations needed, where M is the number of rows, N is the number of columns, and n is the average number of non-zero elements in each row.

Chapter 7

Parallelization of the QR algorithm

Modern computers have multiple processing cores that enable parallel execution of different tasks. Nowadays, commercially available processors implement between four and eight cores. It makes sense to consider the parallelization of procedures to ease their computation time. The *speed-up* S of a procedure produced by its parallelization is given by

$$S = \frac{T_{seq}}{T_{par}}$$

where T_{seq} is its execution time with sequential computations and T_{par} is its execution time with parallel computations. There are two main strategies to parallelize a procedure. A *fine-grained* approach consisting of a large amount of small subtasks with little requirements and a *coarse-grained* approach composed of a small amount of large subtasks that may require significant computation resources.

7.1 Fine-grained subtasks

Fine-grained strategies consist in the decomposition of the procedure in small subtasks that have no dependence on each other. Each subtask will demand little computation resources and will produce small intermediate results, which are easy to store or send to other subtasks. Usually, the subtask creation and communication have an associated overhead that becomes non-negligible with this strategy as it creates a large amount of subtasks.

The reduction to triangular form of a matrix $A \in \mathbb{R}^{M \times N}$ is performed by means of a series of rotations. A rotation G_{ij} only affects rows i and j of A and is independent of another rotation G_{kl} if and only if $i \neq j \neq k \neq l$. So a fine-grained

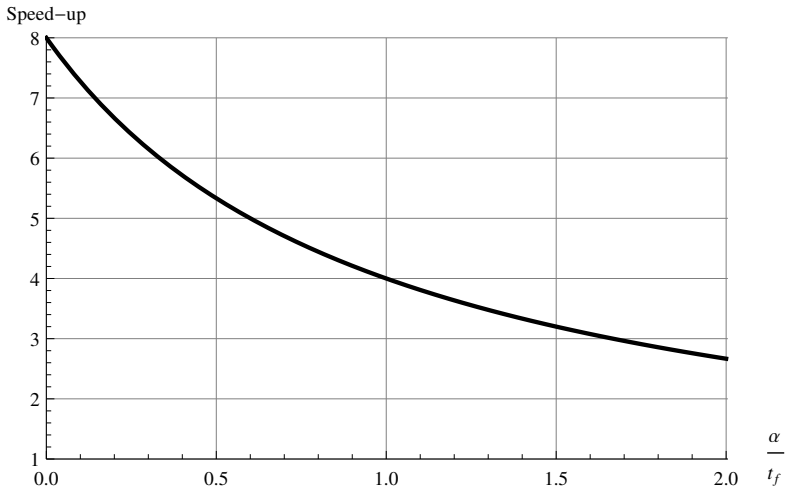


Figure 7.1: Speed-up of a procedure with eight subtasks executed in parallel, as a function of the ratio between the parallel subtask overhead (α) and its amount of computation (t_f). In this case, $\frac{\alpha}{t_f}$ varies between zero (theoretical optimal case) and two (non-negligible performance loss).

subtask consists in a rotation of two rows of A and two subtasks can be executed in parallel if the above condition holds.

Suppose that the rows of A have, in average, n non-zero elements, then, the time complexity of this subtask is $\mathcal{O}(N \log n)$. The entire reduction to triangular form will be composed of g subtasks, where $Mn < g < MN$. Let t_f be the average time to compute a fine-grained subtask. If all subtasks are executed sequentially, this process will be completed in

$$T_{seq} = gt_f$$

Consider that there are $c = 2, 4, 8, \ldots$ cores available, so we are going to execute c subtasks in parallel. Now, the reduction to triangular form will be completed in

$$T_{par} = \frac{g(\alpha + t_f)}{c}$$

where α is the overhead associated to the execution of a subtask in parallel, which mostly depends of the hardware and the operative system. Therefore, the parallelization of this procedure achieves a speed-up of

$$S = \frac{ct_f}{\alpha + t_f} \tag{7.1}$$

Theoretically, if we could achieve an $\alpha = 0$, we would obtain a speed-up equal to the number of cores. But in practice α will be non-negligible. In fact, fine-grained

subtasks must be defined carefully, so that α will not overwhelm its amount of computation (see Figure 7.1). In our implementation, when fine-grained strategy is used with matrices of $64 \times 64 \times 64$, experimental results show that $\frac{\alpha}{t_f} \approx 0.4$. These experimental results where obtained with an Intel Core i7-2600 processor and suggest that fine-grained strategy is not suitable for smaller matrices.

In terms of space, the QR decomposition using the row-wise strategy requires $\mathcal{O}(N^2)$. Each fine-grained subtask will require two rows as input and will produce two rows as output, that is $\mathcal{O}(N)$ space, so, all c tasks in parallel will require $\mathcal{O}(cN)$ additional space, which is negligible considering the requirements overall reduction to triangular form.

The Givens rotations generated by the QR decomposition must be stored and applied in the same order that they were generated and applied to the reduced matrix. Executing fine-grained subtasks in parallel will produce several rotations at the same time. Suppose there are two rotations G_{ij} and G_{kl}, where $i \neq j \neq k \neq l$ and they are computed and applied in parallel. In this case, G_{ij} and G_{kl} can be stored and applied in any order. In other words, the second rotation will no *see* the changes performed by the first. Suppose now there are c independent rotations computed and applied in parallel. As before, they can be stored and applied in any order, but they need to be serialized in order to be written in file. After c rotations are executed in parallel, we need to store the rotations in some kind of shared buffer and synchronize the executions, in order to assure that all c rotations have been saved, so, the next c rotations can share rows with the previous. The synchronization between the different subtasks leads to a bigger overhead, but simplifies the row selection for new subtasks. Moreover, if the save operation in the shared buffer is fast, the additional overhead will be lowered. An array of rotations with a shared mutex combined with the small size of a rotation structure will lead to an easy and fast enough implementation of this shared buffer.

Aside of the independence of the subtasks by assuring they affect to different rows, mutual exclusion must be achieved. In §4.3 we implemented a sparse matrix scheme based on BSTs. Each row is stored in an independent BST, so, subtasks can read/write freely. But, given a row, to obtain its BST, all subtasks must seek in the list of BSTs by the row index. This list can be implemented as an array (if there is enough memory) or as another BST (see §4.3.1). Therefore, a mutex must be placed on the list of rows to prevent race conditions while different subtasks access this list at the same time. Again, the mutex will lead to a bigger, but tolerable, overhead, as the time complexity to obtain the appropriate BST reference is constant (in the array implementation) or logarithmic (in the BST implementation).

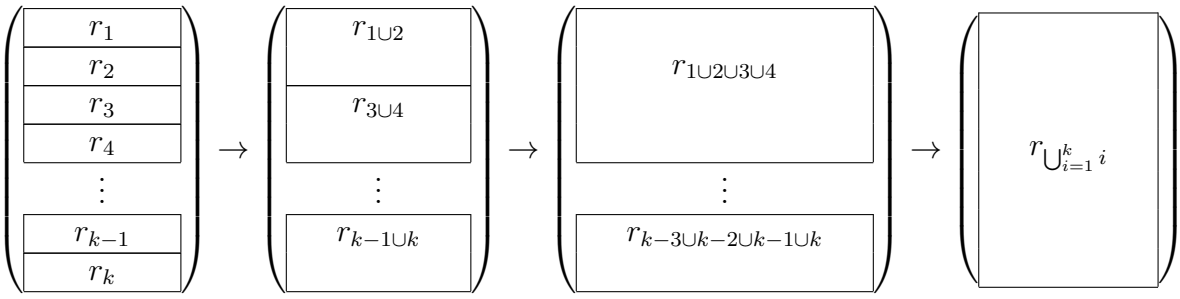


Figure 7.2: Subtasks in the different stages of the QR decomposition using coarse-grained strategy. Subtask $r_{i\cup j}$ will not be computed until subtasks r_i and r_j finish.

7.2 Coarse-grained subtasks

Coarse-grained strategy consists in the decomposition of the procedure in big subtasks that have no dependence on each other. As few subtasks are created, their associated overhead becomes negligible. But big subtasks will demand significant computation resources and produce large intermediate results, which are difficult to store or send.

A coarse-grained subtask consists in the reduction to triangular form of a set r of rows of $A \in \mathbb{R}^{M\times N}$. The reduction to triangular form of the rows in r_1 will be independent of the reduction of the rows in r_2 if and only if r_1 and r_2 are disjoint. Therefore the reduction to triangular form of the rows in r_1 and r_2 can be executed in parallel if the above condition holds. Suppose now that all M rows are distributed in k disjoint sets r_1, r_2, ..., r_k of approximately the same number of rows, so k subtasks can be executed in parallel. Following the previous example, after completing the subtasks related with r_1 and r_2, the next subtask to execute will be the reduction to triangular form of the rows in the set $r_{1\cup 2} = r_1 \bigcup r_2$ (see Figure 7.2). Note that $r_{1\cup 2}$ and the rest of the sets r_3, r_4, ..., r_k are disjoint, so they can be executed in parallel. In fact, the only dependence would be between $r_{1\cup 2}$ and r_1 or r_2, but as $r_{1\cup 2}$ will be computed as a result of r_1 and r_2, these subtasks will never coexist. This way, the entire reduction to triangular form will be completed in g subtasks, where

$$g = k + \frac{k}{2} + \frac{k}{4} + \ldots + 1 = \sum_{i=0}^{\log_2 k} \frac{k}{2^i} = 2k - 1$$

Suppose that the rows of A have, in average, n non-zero elements, then, the time complexity of a coarse-grained subtask is $\mathcal{O}(|r|N^2 \log n)$, where $|r|$ is the number of rows in r. For now, we assume $|r| \approx N$. If we compare the time

complexity of coarse and fine-grained subtasks

$$\frac{\mathcal{O}(|r|N^2 \log n)}{\mathcal{O}(N \log n)} = \frac{\mathcal{O}(N^3 \log n)}{\mathcal{O}(N \log n)} = \mathcal{O}(N^2)$$

we obtain that coarse-grained subtasks have a cost two orders of magnitude greater than fine-grained subtasks. Let t_c be the average time to compute a coarse-grained subtask. We can derive from above cost that $t_c \approx 100t_f$. Consider that there are $c = 2, 4, 8, \ldots$ cores available, so we are going to execute c subtasks in parallel. The speed-up achieved with the parallelization of this procedure will be the same of 7.1,

$$S = \frac{ct_c}{\alpha + t_c}$$

but the cost of a subtask computation t_c makes α negligible. In Figure 7.3 is represented the speed-up of a procedure when ranging the subtask overhead as we did for fine-grained subtasks, but considering now t_c.

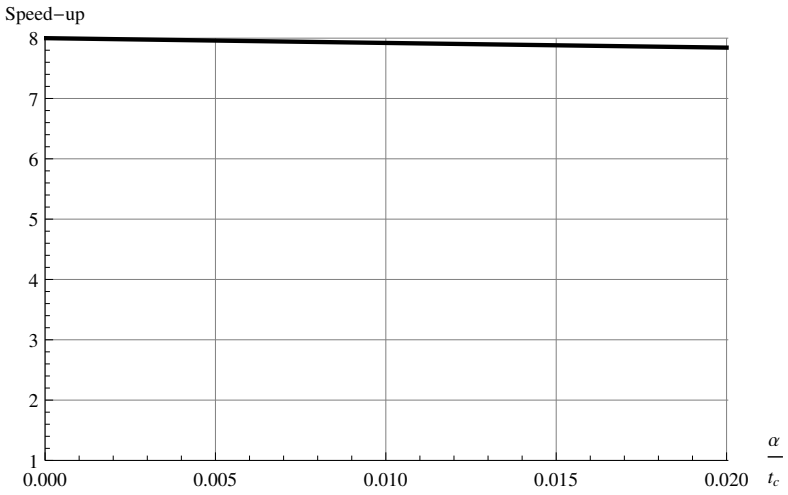


Figure 7.3: Speed-up of a procedure with eight subtasks executed in parallel, as a function of the ratio between the parallel subtask overhead (α) and its amount of computation (t_c). In this case, the same subtask overhead than fine-grained subtasks has been considered, but with $t_c \approx 100t_f$.

Unlike the fine-grained strategy, coarse-grained has one additional parameter to define: the number of rows of each set. In §6.2 we defined a strategy to reduce the fill-in based on performing the QR decomposition separately in different sets of rows, instead of in the whole matrix. We use these sets as the r_i in coarse-grained strategy. There will be different $|r_i|$ established by the fill-in reduction criterion, but experimental data show that, in average, $|r_i| \approx N$ which is compatible with our time complexity calculation.

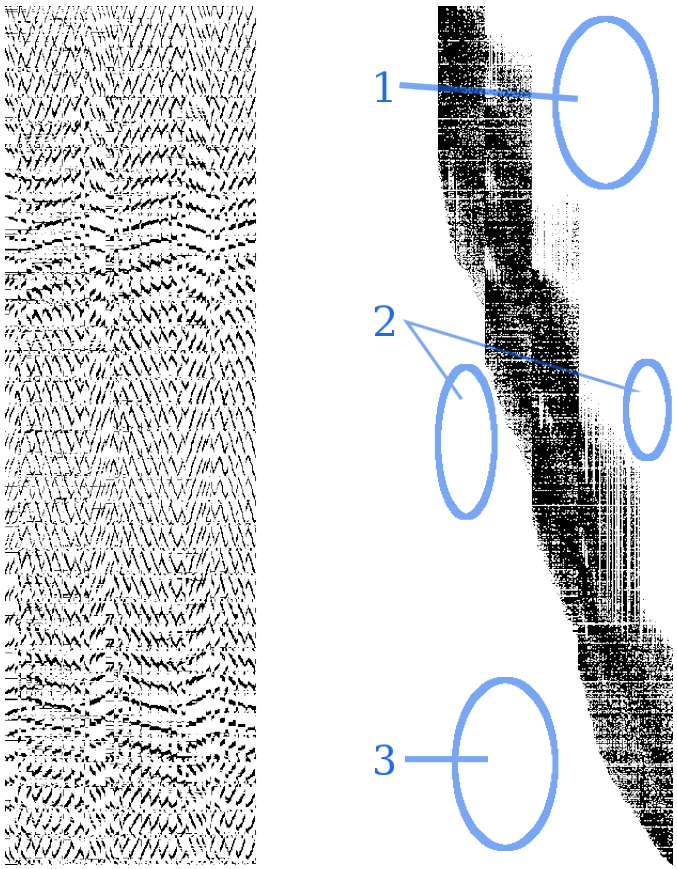


Figure 7.4: Non-zero elements of the CT model matrix, as they are generated (left) and reordered with the band ordering discussed in §6.2 (right). Highlighted zones mark columns of zeros that are useful to reduce the size of intermediate results of coarse-grained subtasks.

In terms of space, each subtask will require a set of rows (submatrix) as input and will produce a set of rows (upper triangular submatrix) as output. This will require $\mathcal{O}(N^2)$ space at most. Less space will be needed if, for example, we demand the last i columns be zero for all rows of the subtask, needing $\mathcal{O}((N-i)^2)$. If i is sufficiently large, the space requirement will be considerably eased. The band ordering discussed in §6.2 can be used to make this row selection (see Figure 7.4). After the ordering is applied to A, first rows will have their last columns zero (Figure 7.4: 1), last rows will have their first columns zero (Figure 7.4: 3) and beginning and ending columns will be zero for the rows located in between (Figure 7.4: 2).

In the previous section, we synchronized fine-grained subtasks in order to save the simultaneously generated Givens rotations. In the case of coarse-grained subtasks, there is no need of synchronization, as we defined the sets of rows in a way that all subtasks always will be independent of each other. In other words, all rotations applied by each subtask will not *see* the changes made by any rotation of coexistent subtasks. Instead, inside each subtask, the next rotation will not be generated until the current rotation is saved.

No further considerations than made for fine-grained strategy are needed to achieve mutual exclusion. The same shared buffer can be used in order to save the rotations of all subtasks and a mutex in the list of BSTs will allow all parallel subtasks to search for rows in the sparse matrix scheme.

7.3 Parallelization of the backward substitution process

The backward substitution process is the final stage of the image reconstruction. As the QR decomposition $A \in \mathbb{R}^{M\times N}$ and $A = QR$ is computed *a priori*, the backward substitution of

$$Rx = Q^T b \tag{7.2}$$

will be the process that defines the reconstruction time form the user's point of view. Therefore, the optimization of this task is of the utmost importance. There are five subtasks to complete the image reconstruction that are inherently sequential between them:

1. The CT measurement loading, which will initialize b ($\mathcal{O}(M)$).

2. The reordering of b according to any ordering applied to the rows of A before the QR decomposition (see §6) ($\mathcal{O}(M)$).

3. The computation of the right hand side $Q^T b$ (see Equation 7.2). The right hand side is obtained by applying to b the Givens rotations used to reduce A to R (see §5.3.3) ($\mathcal{O}(g)$, where g is the total number of Givens rotations).

4. The computation of the solution x by backward substitution ($\mathcal{O}(N^2)$).

5. The reordering of x according to any ordering applied to the columns of A before the QR decomposition (see §6) in order to form an image ($\mathcal{O}(N)$).

Our strategy consists in the implementation of a circular buffer as the reading buffer that is used to load the Q^T file. Two subtasks will manage this buffer, one that will load from disk to the available space in buffer (producing rotations), and other that will apply the rotations stored in the buffer to an array (consuming rotations). Several contiguous sections must be defined within the buffer to assure mutual exclusion between the two subtasks, while the consuming task is applying the rotations stored in a section, the producing task can be loading the next rotations freely in the other sections.

Each section must have a different mutex, so if a subtask depletes or completely fills the buffer, it will be paused in behalf of the complementary subtask. In our case, we experimentally determined that the better performance was achieved with a circular buffer size of 16 MB divided in four equally spaced sections.

Regarding the forth subtask, as a standard backward substitution process, its parallelization is already implemented in the Intel MKL PARDISO [66], for example. The third and forth subtasks are the most time consuming and little or nothing can be done for the rest. The main advantage of this approach is that it is simple and fast to implement.

Chapter 8

The QR solution of the linear system

Reconstructed images are the solution to the CT model matrix considering a particular CT measurement. It is important to perform an analysis from the image quality point of view to assess the suitability of QR as a CT reconstruction algorithm. In this Chapter, two measurements will be analyzed: a phantom with homogeneous materials and a mouse. Different features will be studied from both cases such as correct translation of the x-ray attenuation coefficients, image contrast, noise, sharpness, or fine feature reproducibility, yielding a complete assessment of the QR reconstruction algorithm.

8.1 Image quality

Measurements for the assessment of the image quality were obtained with an Albira μCT [42], which is a trimodal PET/SPECT/CT scanner. The CT subsystem is a cone beam CT that uses a microfocus x-ray source with a focal spot size of 35 μm and a CMOS flat-panel with an active area of 120×120 mm^2 that consists of a 2400×2400 pixelated array sensors. Fixed distances from the x-ray tube to the isocenter (290 mm) and x-ray detector (425 mm) lead to a magnification factor of 1.46. This geometry allows a transaxial field of view (FoV) of about 80 mm in diameter and an axial FoV of 65 mm. As discussed in Section 3.1, the detectors can be projected to the isocenter, in order to avoid considering the magnification factor. Thus, from now on, this projection will be considered and sensor pixels will be dimensioned accordingly. Following this correction, the 2400×2400 pixelated array sensor will be considered with an active area of 82.2×82.2 mm^2 and each sensor pixel will be considered with an edge size of 34.2 μm.

Unfortunately, a model based approach considering a sensor of a 2400×2400

pixels is, now a days, not viable computationally for QR reconstruction. Therefore, two measure configurations have been defined, depending on the binning of the device sensor pixels. On one hand, the device sensor has been rebinned to a 256×256 pixelated array. This renders sensor pixels with an edge size of 0.3 mm. On the other hand, the device sensor has been rebinned to a 392×392 pixelated array. This renders sensor pixels with an edge size of 0.2 mm. These configurations allow to produce 3D images with voxel sizes up to 0.3 mm in the first case and 0.2 mm in the second case for FBP and MLEM reconstruction methods. Regarding the QR, These configurations allow to produce 3D images with voxel sizes of 1.28 mm in the first case and 0.8 mm in the second case. As discussed in Section 3.2, smaller voxel sizes will lead to near-rank-deficient system matrix (because of linearly dependent rows generated by this CT modeling), and therefore, a break down in the back-substitution phase can be expected. Hence, in this chapter, QR generated 3D images of 1.28 mm voxel size and 0.3 mm detector size will be compared against FBP and MLEM generated 3D images of 0.3, 0.8, and 1.2 mm voxel size and 0.3 mm detector size. In the same way, QR generated 3D images of 0.8 mm voxel size and 0.2 mm detector size will be compared against FBP and MLEM generated 3D images of 0.2, 0.5, and 0.8 mm voxel size and 0.2 mm detector size.

Before the image comparison, it is necessary to consider the configuration of the FBP and MLEM algorithms. The reconstruction filter function used in the FBP implementation is a band-limited ramp filter, using the Gaussian apodization function $e^{-x^2/(2\sigma^2)}$ where σ is chosen so that the Gaussian apodization function has a full width at half maximum of 0.8. The MLEM is configured to perform 40 iterations and no regularization is applied.

A phantom with homogeneous materials was used, in order to produce 3D images with homogeneous volume of interest (VoI). Mean and standard deviation of the voxel values of these VoIs will be useful to obtain image quality metrics related to signal and noise. Mean and the standard deviation are computed according to equations (8.1) and (8.2)

$$\mu_v = \frac{1}{N_v} \sum_{i=1}^{N_v} x_i \tag{8.1}$$

$$\sigma_v = \sqrt{\frac{1}{N_v - 1} \sum_{i=1}^{N_v} (x_i - \mu_v)^2} \tag{8.2}$$

where N_v is the number of pixels in the VoI and x_i is the value of each voxel.

The phantom used for these measurements (see Figure 8.1) consists on a PMMA cylinder of 50 mm height and 55 mm in diameter and contains cylinders of different materials. The center of each cylinder is 16 mm off the axis and are 8 mm in diameter. Aside of the container cylinder of PMMA, the rest of cylin-

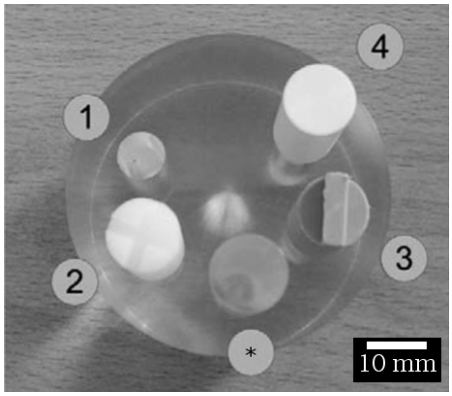


Figure 8.1: Phantom of PMMA with inserts of PMMA (*) for alignment purposes, air (1), PTFE (or Teflon) (2), PE (3), and POM (4), which model regions filled with air inside the body, soft bone, adipose tissue, and organs tissue, respectively.

ders are made of air, PTFE (or Teflon), PE and POM. These materials model air regions, PMMA for soft tissue, PTFE for soft bone, PE for adipose tissue and POM for organs tissue, inside the human body.

The correct translation of the attenuation coefficients of all the considered materials has to be evaluated before the signal and noise analysis. A plot of the calculated attenuation coefficients *versus* the nominal attenuation coefficients of the phantom materials is used to verify the translation of the coefficients of the reconstruction algorithm. This is often called linearity check [67] and ideally, the graph of the calculated *versus* the phantom attenuation coefficients should be related with the same line. The nominal attenuation coefficients of the phantom materials have been obtained from [68, 69].

Measurements of phantom shown in Figure 8.1 allow the calculation of attenuation coefficients of materials with well known nominal attenuation coefficients. In Figures 8.2, 8.3, and 8.4, a linearity check of the QR algorithm is presented (Figure 8.2) alongside of a linearity check of the FBP and MLEM algorithms (Figures 8.3 and 8.4) which are important references in the field of CT reconstruction. In Figures 8.2, 8.3, and 8.4 can be observed that the air values are above the ideal line in all cases. This is due to the fact that the air VoI is inside the phantom body of PMMA. The air VoI is positioned inside the phantom because of the difficulties experienced by some reconstruction methods emptying image areas corresponding to areas of air inside a body. Unless the atypically high value of air, the rest of the materials are translated accordingly to their nominal attenuation coefficients.

As a general comparison, in Figure 8.5, the central slices of the 3D reconstructions of the phantom showed in Figure 8.1 are presented. Each row corresponds to a different reconstruction method, while each column corresponds to a different

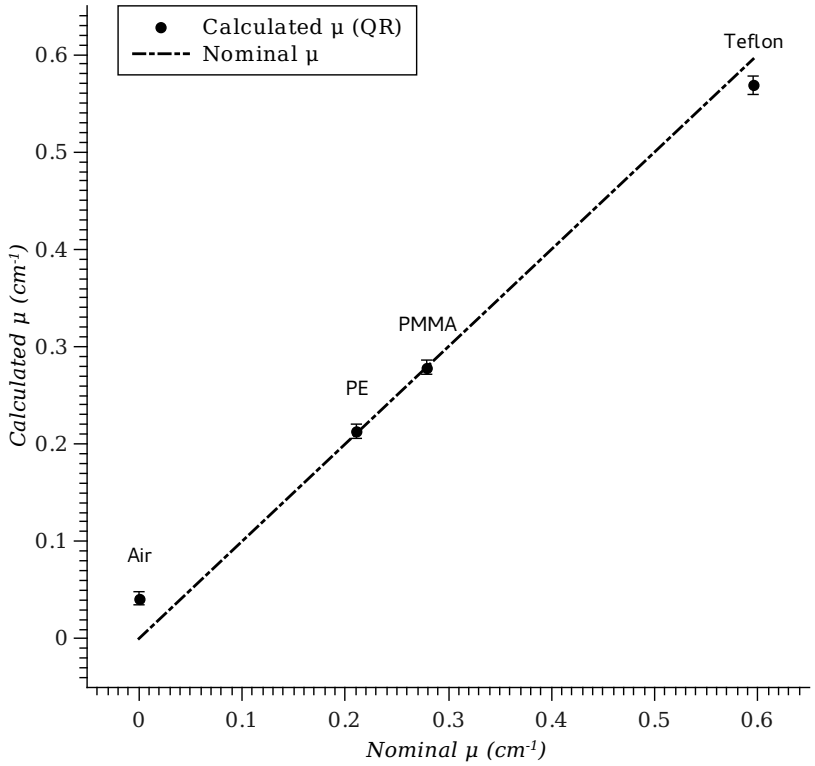


Figure 8.2: Relationship between the nominal and the reconstructed attenuation coefficients (at 40 kV) of the phantom inserts (see Figure 8.1) air, PE, PMMA, and Teflon. Reconstructed attenuation coefficients were obtained using QR.

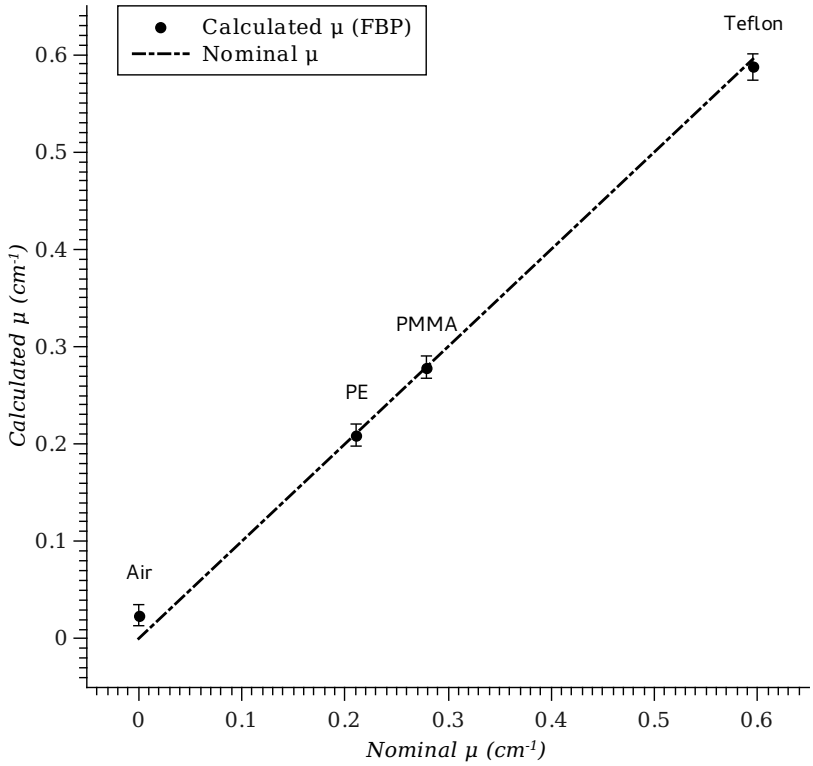


Figure 8.3: Relationship between the nominal and the reconstructed attenuation coefficients (at 40 kV) of the phantom inserts (see Figure 8.1) air, PE, PMMA, and Teflon. Reconstructed attenuation coefficients were obtained using FBP.

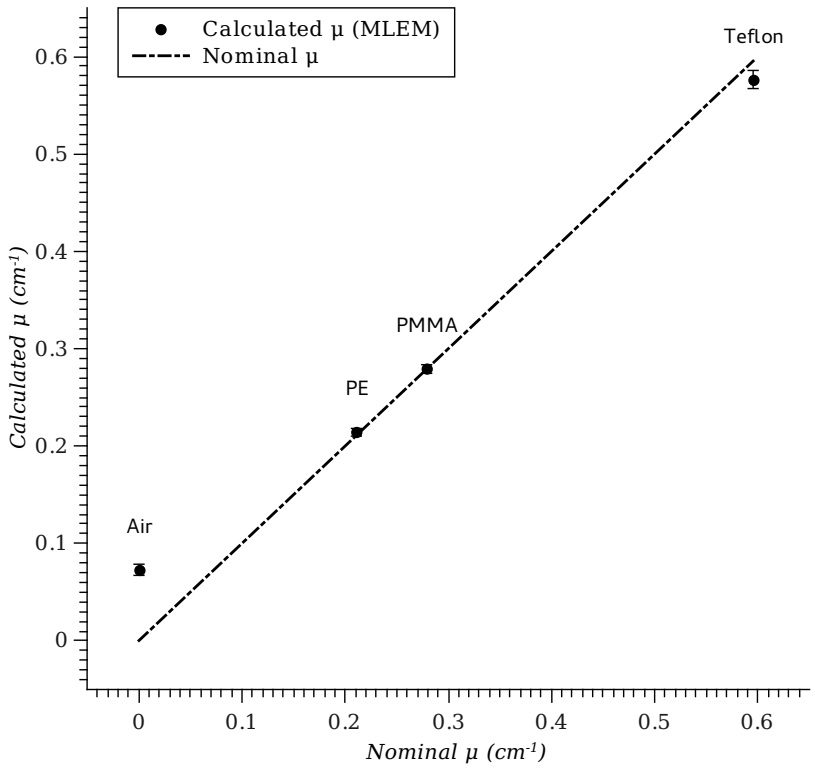


Figure 8.4: Relationship between the nominal and the reconstructed attenuation coefficients (at 40 kV) of the phantom inserts (see Figure 8.1) air, PE, PMMA, and Teflon. Reconstructed attenuation coefficients were obtained using MLEM.

voxel size. Specifically, top to bottom, are the QR, MLEM, and FBP methods, and left to right, are 0.3, 0.8, and 1.2 mm. In the case of the QR, only 0.8 and 1.28 mm are available, due to the computational cost of the 0.3 mm voxel size reconstruction.

In terms of visual (subjective) quality, the QR reconstruction with 1.28 mm can be placed in between the 0.8 and 1.2 mm reconstructions of FBP and MLEM. Likewise, the QR reconstruction with 0.8 mm can be placed in between the 0.3 and 0.8 mm reconstructions of FBP and MLEM. This is significant in the sense that *large* voxel size has an important role in the degradation of the image quality. In this case, a *large* voxel size ($\sim$ 1.2 mm) refers to the relation between the voxel edge size and each insert diameter (8 mm) which leaves $\sim$ 6 voxels in diameter for each insert. With so few voxels, the reconstruction algorithms should perform a fine estimate of the attenuation coefficients, in order to correctly translate curved edges (between materials) and avoid blurring. This effect is mostly appreciated in the FBP reconstruction with 1.2 mm voxels and is less evident in the QR and MLEM reconstructions with 1.28 and 1.2 mm. The better performing of QR and MLEM is mainly due to the fact that they are model based methods.

The first step for the numerical analysis of the reconstructed images is the definition of meaningful VoIs. All VoIs will be cylindrical along the axial axis with a height of approximately 38 mm (in order to maximize the number of considered slices in all image configurations). Moreover, all VoIs will have a diameter of 4 mm (inside each insert which is 8 mm in diameter), so that the exterior 2 mm of each insert is excluded in order to avoid considering voxels that are part of transition between materials. In this case, selected VoIs are the following:

- Air, centered in its insert, will be a measure of high contrast low valued region. Also, this VoI will demonstrate the ability for each algorithm to reproduce empty regions inside the body.

- Bone, centered in the Teflon insert, will be a measure of high contrast high valued region.

- Adipose tissue, centered in the PE insert, will be a measure of low contrast low valued region.

- Organs, centered in the POM insert, will be a measure of low contrast high valued region.

- Soft tissue, PMMA, displaced 16 mm off the axis (likewise the rest of inserts), will be a measure representing *background* to compute contrast measurements.

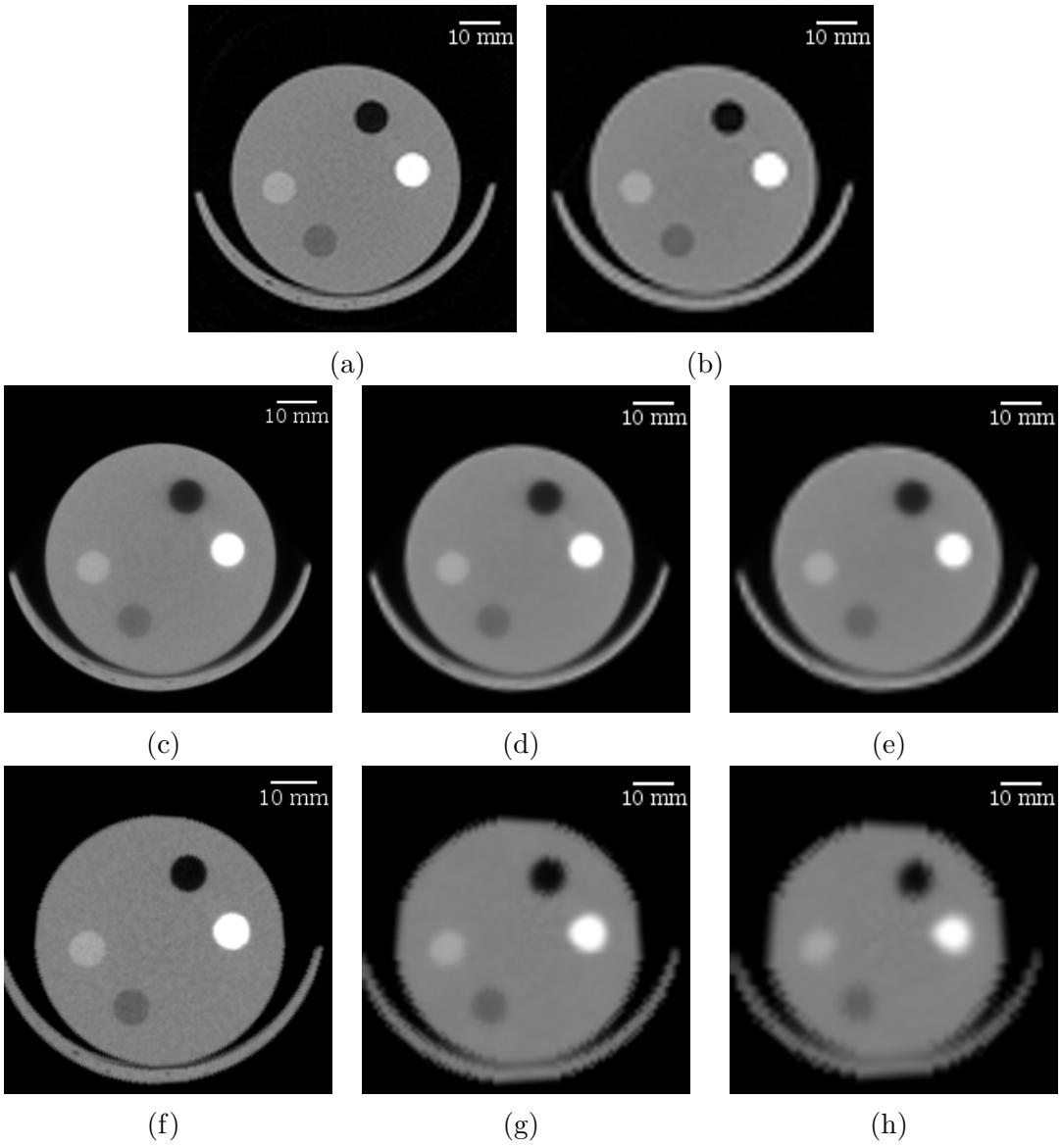


Figure 8.5: Central slice of a 3D reconstructed phantom (see Figure 8.1) with different reconstruction methods and voxel sizes: QR 0.8 mm (a), QR 1.28 mm (b), MLEM 0.3 mm (c), MLEM 0.8 mm (d), MLEM 1.2 mm (e), FBP 0.3 mm (f), FBP 0.8 mm (g), FBP 1.2 mm (h).

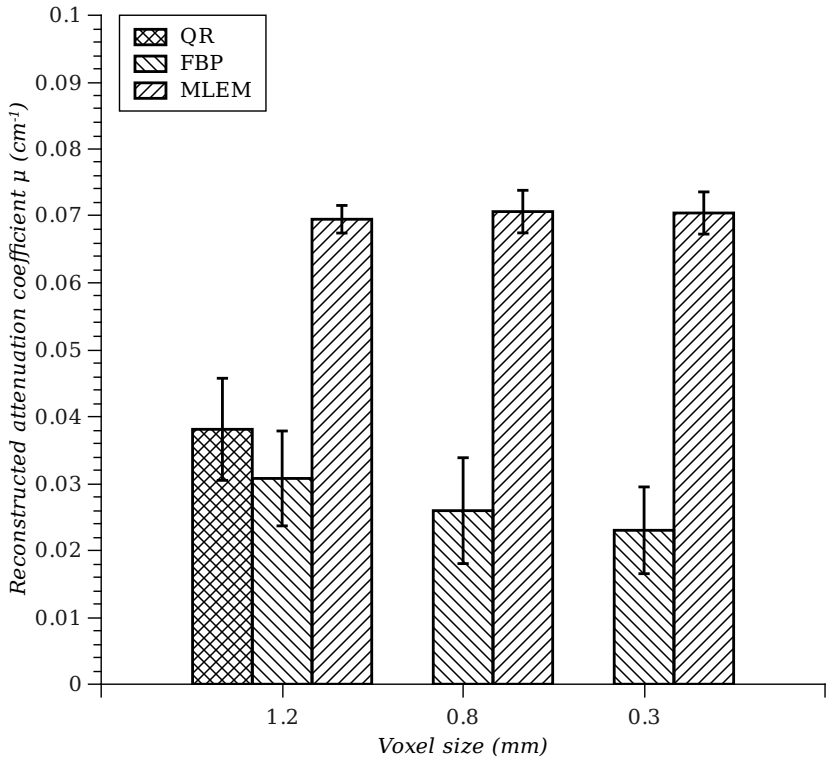
Regarding the mean values obtained from VoIs, in the case of air, reconstruction algorithms must obtain a value close to zero. Specifically, the attenuation coefficient for air at 40 kV is approximately 3×10^{-4} cm^{-1}. In practice, the reconstruction algorithms produce greater values. A figure of merit is how close to zero the reconstructed values of an air region inside the object are. In Figure 8.6, mean and standard deviation for the air VoI are shown. MLEM algorithm presents the highest air mean values. Taking MLEM as reference, in the case of 0.3 mm detectors, QR algorithm offers a reduction of 45.9% while FBP algorithm obtains a reduction up to 67.3% (in the best case scenario) of MLEM air mean value (which remains constant for all voxel sizes). In the case of 0.2 mm detectors, almost the same result is obtained (although, with slightly higher mean air values in all cases). The QR algorithm offers a reduction of 42.9% while FBP algorithm obtains a reduction up to 66.7% (in the best case scenario) of MLEM air mean value (which remains almost constant for all voxel sizes).

The complementary high contrast VoI is made of Teflon, modeling soft bone tissue. The attenuation coefficient for Teflon at 40 kV is approximately 0.6 cm^{-1}. In this case, reconstruction algorithms tend to obtain values lower than expected. In Figure 8.7, mean and standard deviation of the reconstructed attenuation coefficients for the Teflon VoI are shown. FBP algorithm presents the highest Teflon mean values. Taking Teflon nominal attenuation coefficient as reference, in the case of 0.3 mm detectors, QR algorithm reaches a 96% of the expected value while MLEM obtains a 97% and FBP obtains a 98.8% both with 0.8 and 0.3 mm voxels. In the case of 0.2 mm detectors, almost the same result is obtained (although, with slightly lower mean Teflon values in all cases). QR reaches a 95.5% of the expected value while MLEM obtains a 96.8% and FBP obtains a 98.6% both remaining almost constant for all voxel sizes.

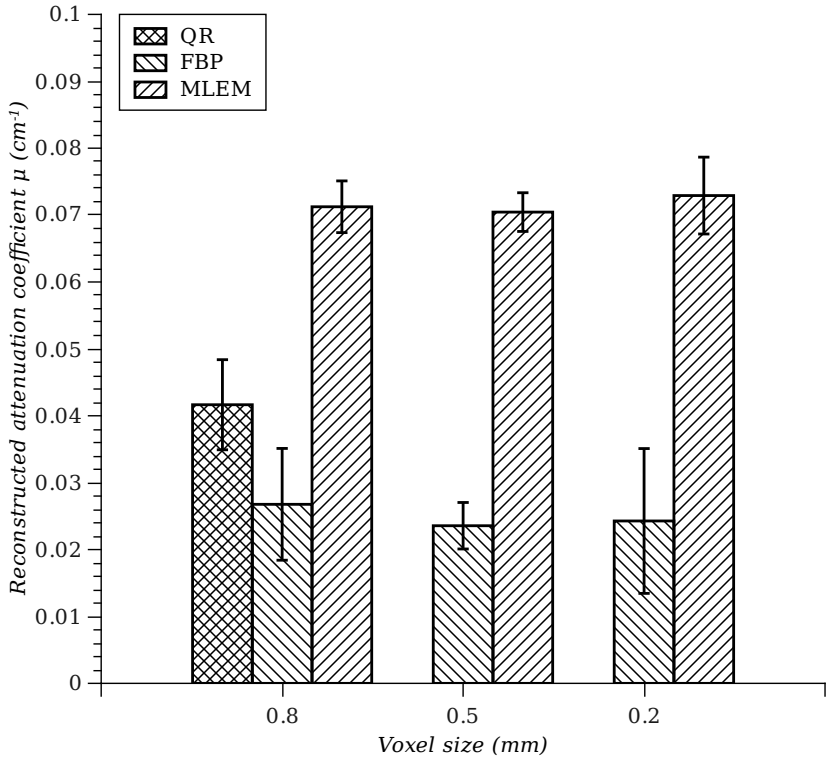
The results of the low contrast VoIs are shown in Figures 8.8 and 8.9. The expected attenuation coefficient for PE is displayed as a horizontal dashed line (POM nominal value was not available in [68]). In all cases, mean similar and close to the expected nominal attenuation coefficient.

As a result summary, the success in the translation of the attenuation coefficients of the different materials are collected in Table 8.1.

A notable increase of the standard deviation (see Section 8.2) of all VoI values appears in images with 0.2 mm detector and voxel sizes. This increase is even greater in the case of FBP. The detector rebinning performed to ease computational cost has a cancellation effect of the noise in measured data. As smaller detectors are considered, this cancellation effect tends to disappear. A similar effect is present regarding the voxel size. Larger voxels integrate the attenuation coefficients of near regions of the image space and also, noise cancellation occurs. This cancellation effect also tends to disappear as smaller voxels are considered.

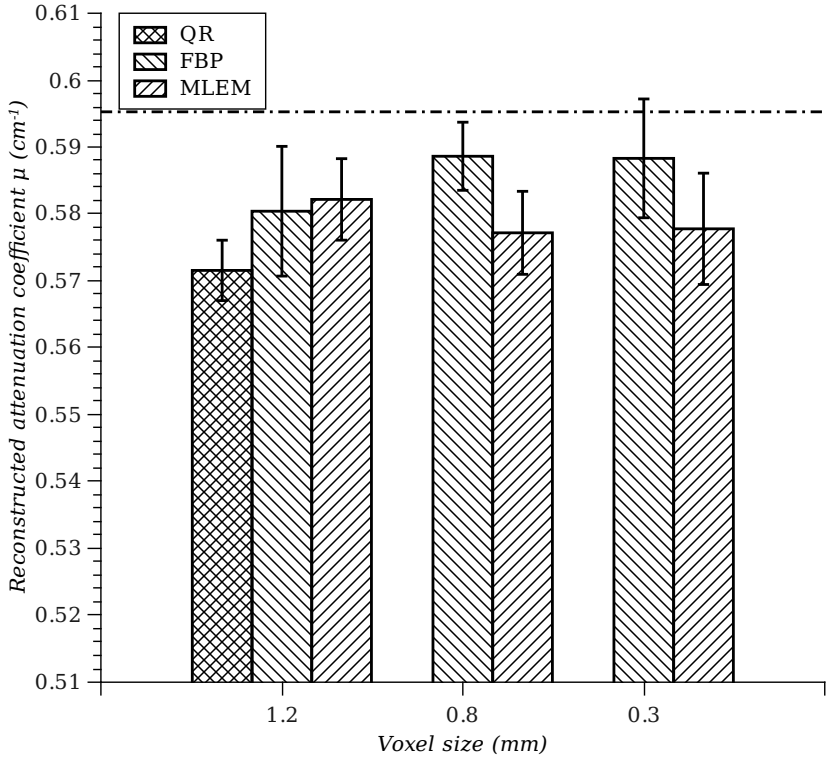


(a)

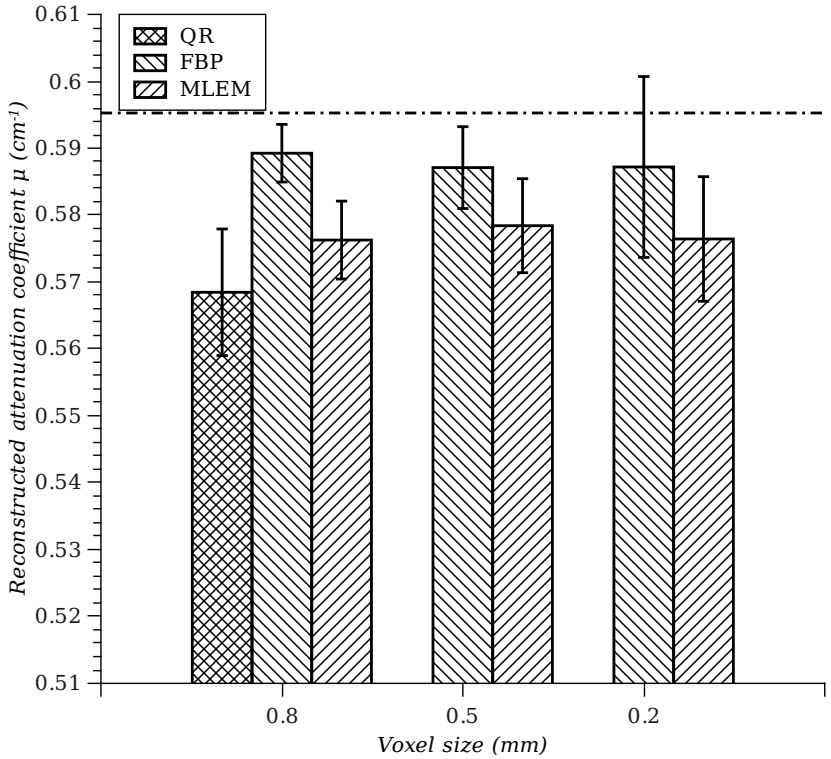


(b)

Figure 8.6: Reconstructed attenuation coefficients for the air insert, for detector sizes of 0.3 mm (a) and 0.2 mm (b). Expected attenuation coefficient is close to zero (approximately 3×10^{-4} cm^{-1}).

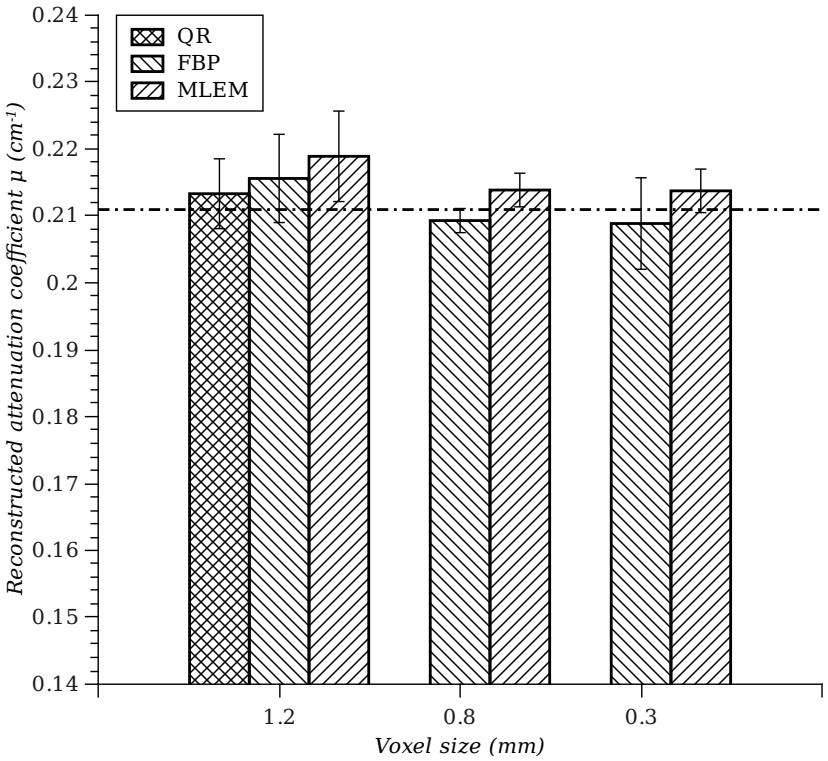


(a)

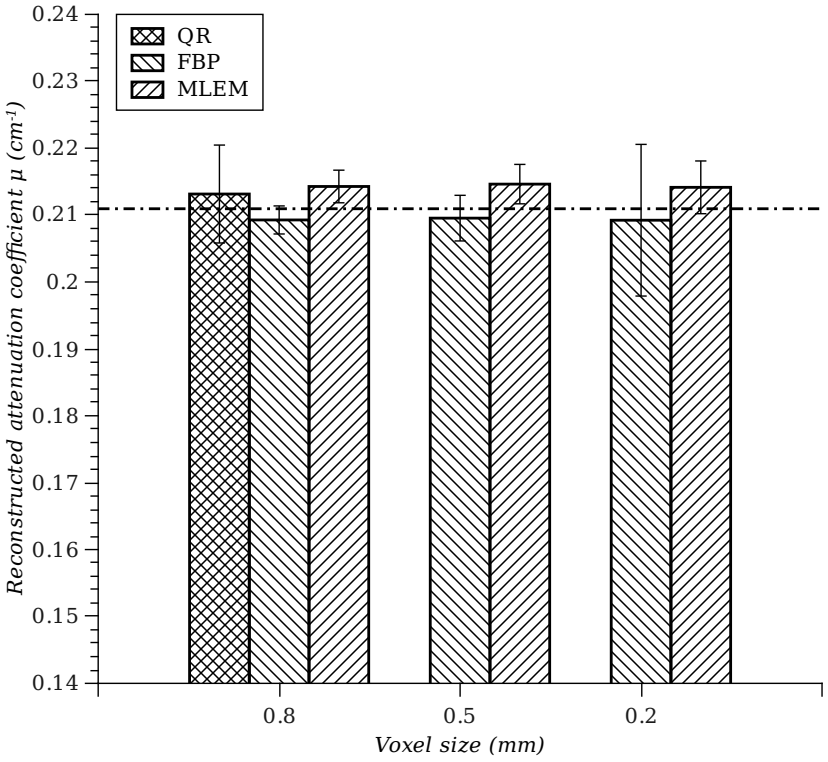


(b)

Figure 8.7: Reconstructed attenuation coefficients for the Teflon insert, for detector sizes of 0.3 mm (a) and 0.2 mm (b). Expected attenuation coefficient is represented by a horizontal dashed line.

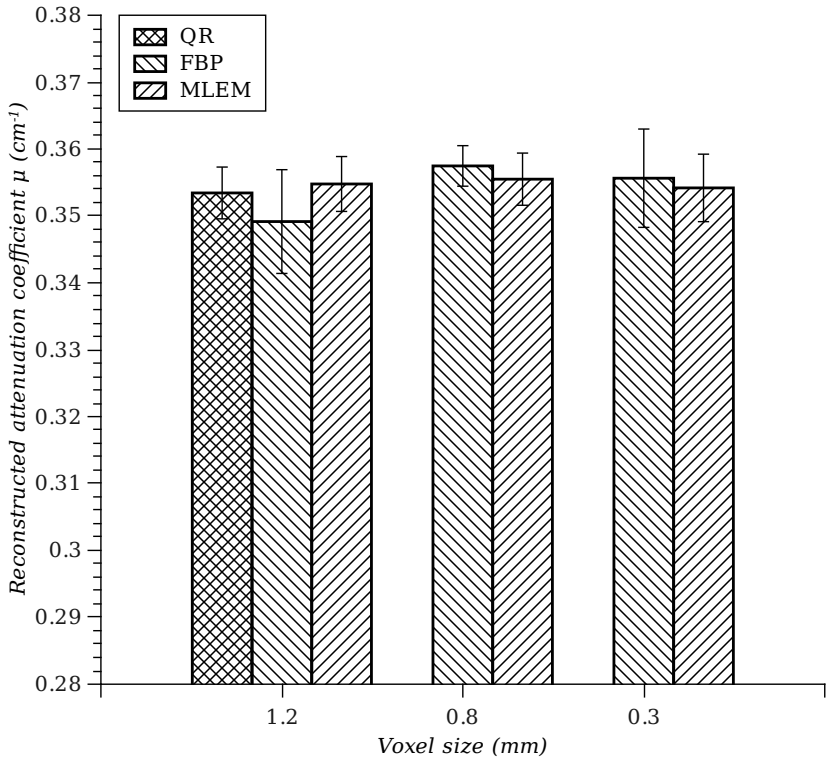


(a)

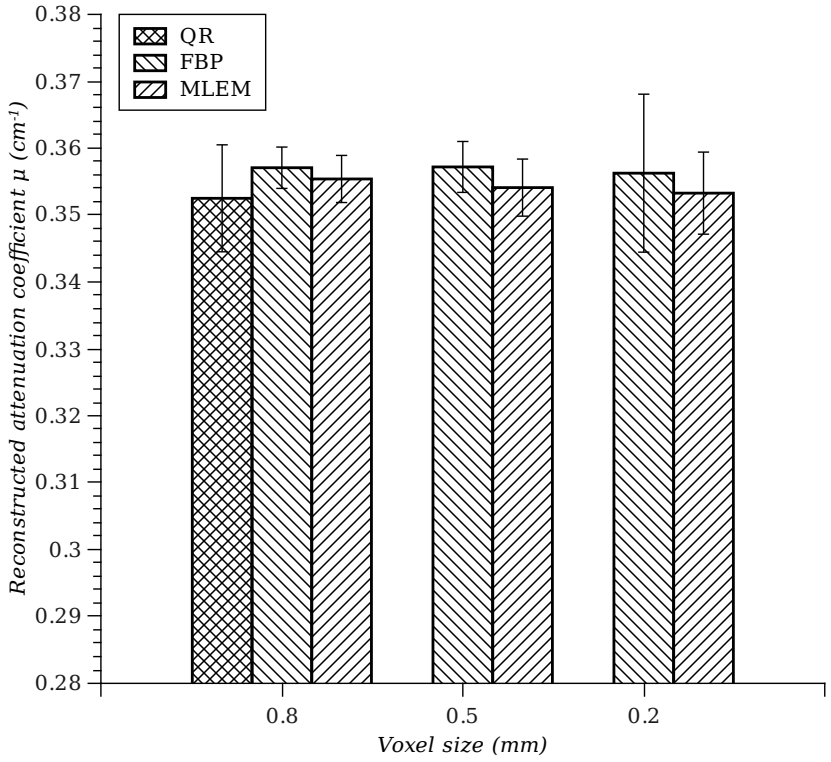


(b)

Figure 8.8: Reconstructed attenuation coefficients for the PE insert, for detector sizes of 0.3 mm (a) and 0.2 mm (b). Expected attenuation coefficient is represented by a horizontal dashed line.



(a)



(b)

Figure 8.9: Reconstructed attenuation coefficients for the POM insert, for detector sizes of 0.3 mm (a) and 0.2 mm (b). Expected attenuation coefficient was not available in [68].

Table 8.1: Mean attenuation coefficients translation summary. Results in air column are the percentage of reduction over the MLEM obtained mean value. Teflon and PE columns are the percentage of the mean value obtained over the nominal attenuation coefficient.

Algorithm	0.3 mm detectors			0.2 mm detectors		
	Air	Teflon	PE	Air	Teflon	PE
QR	45.9%	96%	101%	42.9%	95.5%	101%
FBP	67.3%	98.8%	99%	66.7%	98.6%	99.2%
MLEM	N/A	97%	101%	N/A	96.8%	101.5%

The 0.2 mm detector, 0.2 mm voxel configuration minimizes cancellation in both sides and therefore, standard deviation increase is particularly present in this configuration. In the next Section, a detailed analysis of the standard deviation will be performed.

8.2 Standard deviation noise analysis

Noise is a key characteristic in image analysis. In the field of CT images, two main directions have been consolidated to address this issue: standard deviation related analysis and spectral analysis. A standard deviation related analysis will follow now and in Section 8.4 a noise spectral analysis will be performed. In this section, two of figures of merit related with noise have been considered:

- The coefficient of variation (CV): is a measure of the variability between voxel values in a VoI. CV is divided by μ_v in order to obtain a dimensionless measure.
$$CV = \frac{\sigma_v}{\mu_v}\ 100 \tag{8.3}$$

- The contrast to noise ratio (CNR) [70]: is a measure of the relation between contrast (difference between means) and noise (standard deviation) of a VoI and the PMMA background.
$$CNR = \frac{2\ |\mu_v - \mu_b|}{\sigma_v + \sigma_b} \tag{8.4}$$
where μ_b and σ_b represent the CT number and the standard deviation of PMMA background respectively.

These two measures complement each other. On one hand, the CNR gives a value of *how much* a value difference can be seen through the noise that is present.

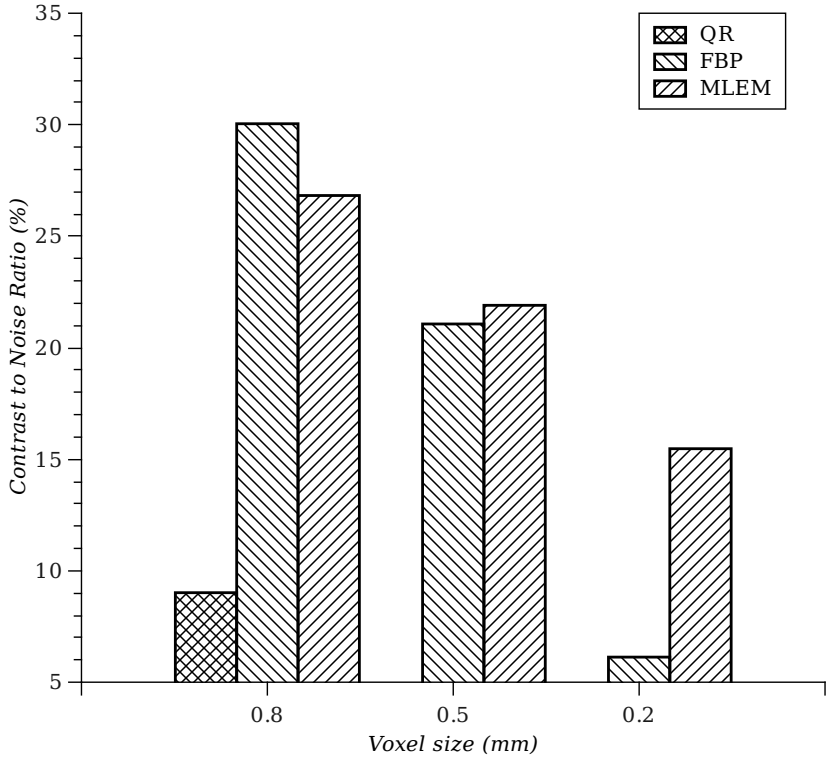
In other words, this measure answers the following question: will the current noise level let *see* image features with a certain contrast? The result is a percentage representing the visibility of the considered VoI. The larger the CNR, the better the visibility. A result close to 0% means that the considered VoI is invisible in presence of the current noise. Of course, this measure also increases with contrast and the following results must be divided in two types (comparable among them): high contrast (air and Teflon) and low contrast (PE and POM).

On the other hand, the CV gives a voxel variability independent of its mean value, in order to get a noise measure comparable between VoIs. As our VoIs are defined over homogeneous regions this accounts directly for the noise present (a result close to 0% will mean that the considered VoI has almost constant voxel values). When analyzing CNR, this measure will answer the following question: *how much* of the CNR variation is due to the noise increase/decrease and not to a possible change of contrast? The combination of these two measures illustrates the amount of noise present and its influence in the interpretation in the reconstructed image.

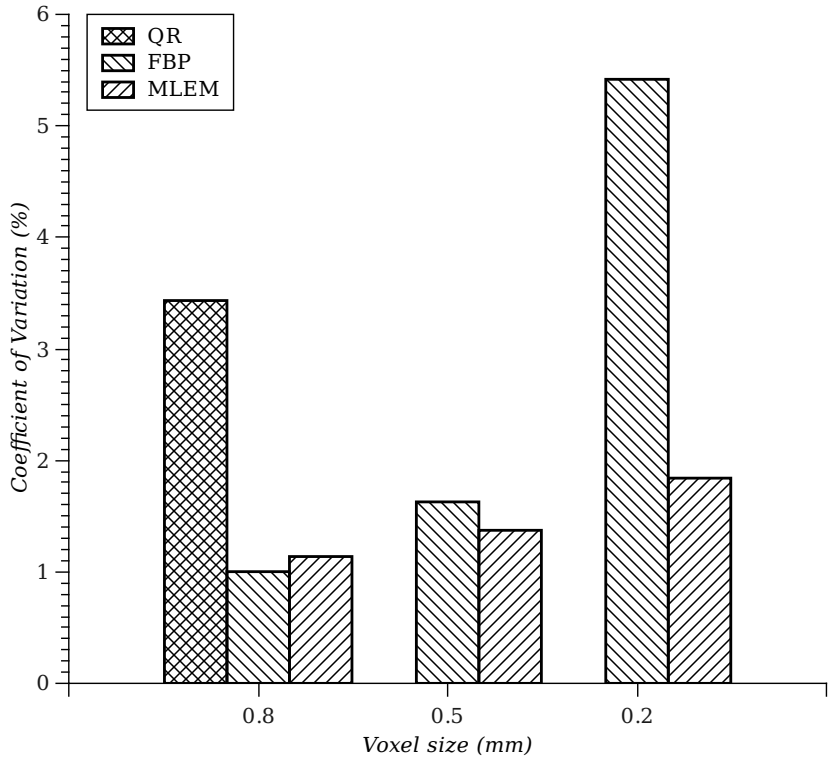
In Figures 8.10 and 8.11, results measured in the low contrast VoIs are presented (PE and POM, respectively). In all cases, degradation in CNR results is due to increase in CV. The comparison between QR and FBP or MLEM shows QR having poor CNR results using 0.8 mm voxels (same voxel size) or even with 0.5 mm voxels. CNR and CV results obtained with QR (0.8 mm voxels) are only comparable to those obtained by FBP or MLEM using 0.2 mm voxels.

In Figure 8.12, results measured in the high contrast Teflon VoI are presented. The low noise present in the Teflon measurements (due to its high attenuation coefficient) should be taken into account. In this case, CV result obtained with QR (0.8 mm voxels) can be comparable to those obtained with FBP or MLEM with 0.5 and 0.2 mm voxels. The discrepancy between CNR and CV is mainly due to the fact that attenuation coefficients obtained with QR in the Teflon VoI do not reach those obtained by FBP or MLEM (see Figure 8.7). Regarding the CV, QR obtains approximately two times the CV results obtained by FBP or MLEM using 0.8 mm (while in the low contrast cases were more than two or three times) and the difference between methods decreases with the voxel size, as well. Even the increase in CV suffered by the FBP using 0.2 mm voxels is mitigated in the case of Teflon.

The air VoI presents a different case (see Figure 8.13). First of all, the great influence of noise in the air VoI (due to its extremely low attenuation coefficient) should be taken into account. In previous cases, the CV obtained with FBP and MLEM increase as the voxel size decreases. In this case, the minimum CV is obtained with 0.5 mm voxels. Regarding the FBP results with 0.8 mm voxels, although the VoIs have been defined with a 2 mm margin in order to avoid voxels

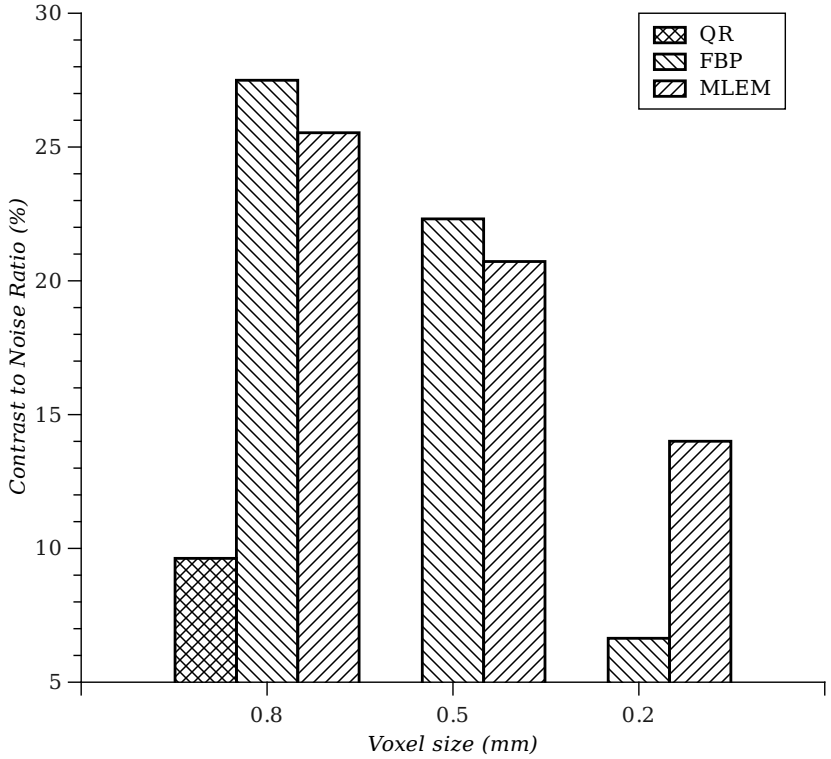


(a)

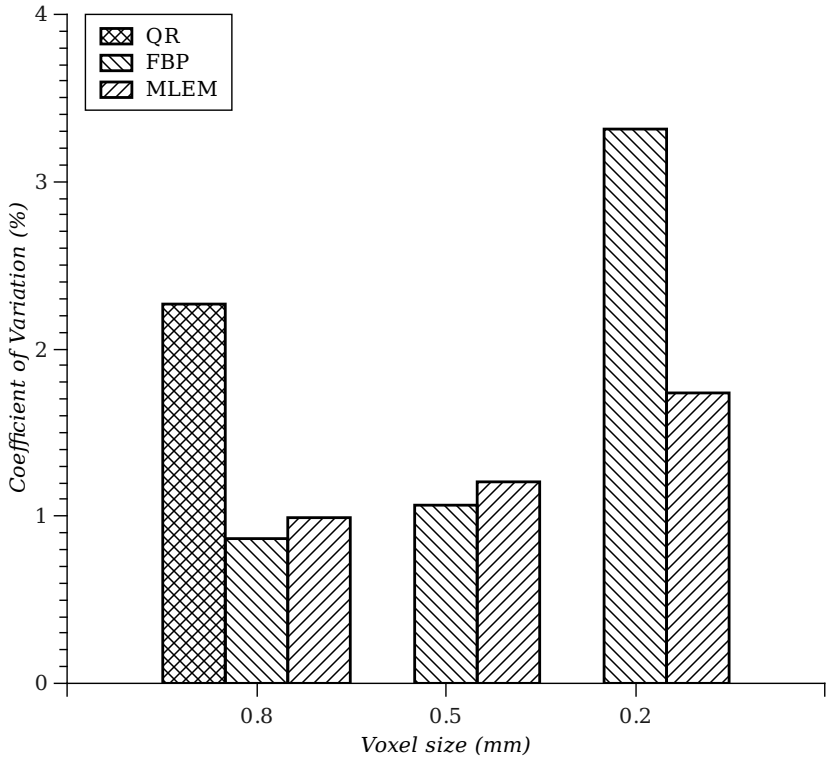


(b)

Figure 8.10: CNR (a) and CV (b) of the PE VoI for each algorithm and voxel size (using 0.2 mm detector size). CV is presented alongside CNR, in order to highlight noise contribution to CNR reduction.

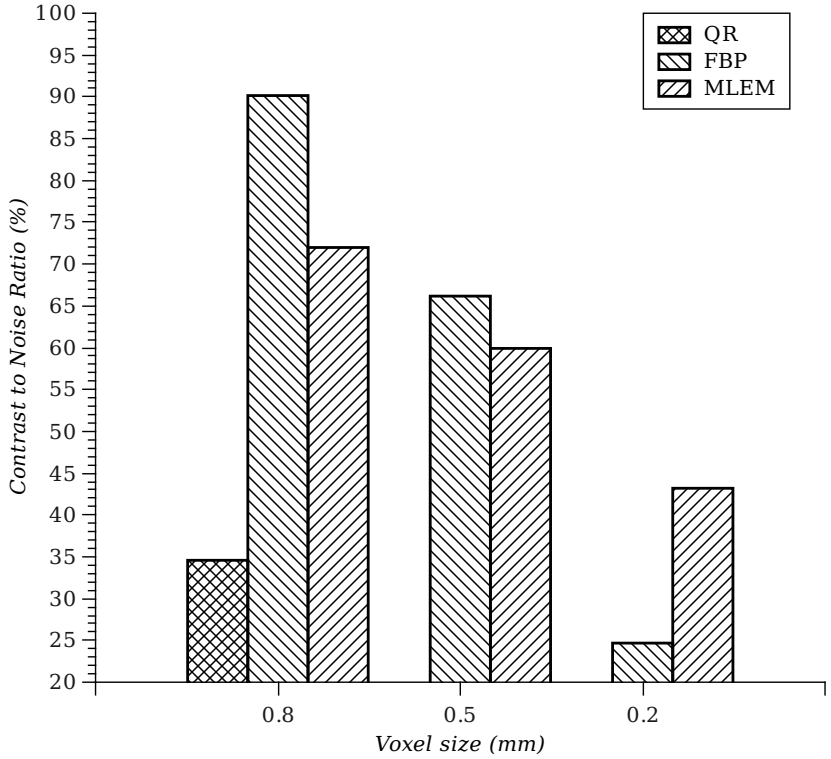


(a)

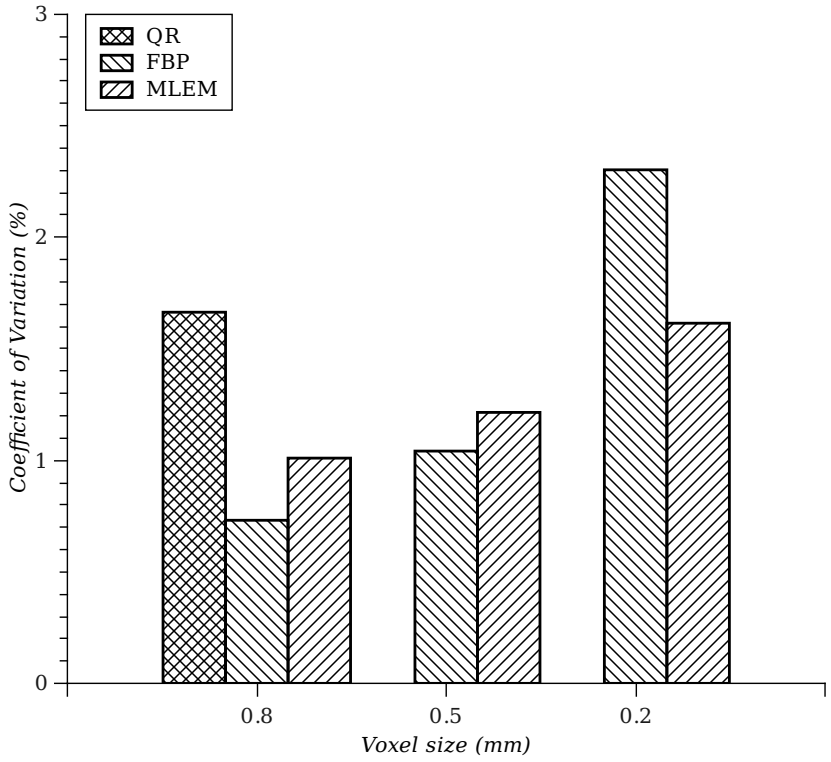


(b)

Figure 8.11: CNR (a) and CV (b) of the POM VoI for each algorithm and voxel size (using 0.2 mm detector size). CV is presented alongside CNR, in order to highlight noise contribution to CNR reduction.



(a)

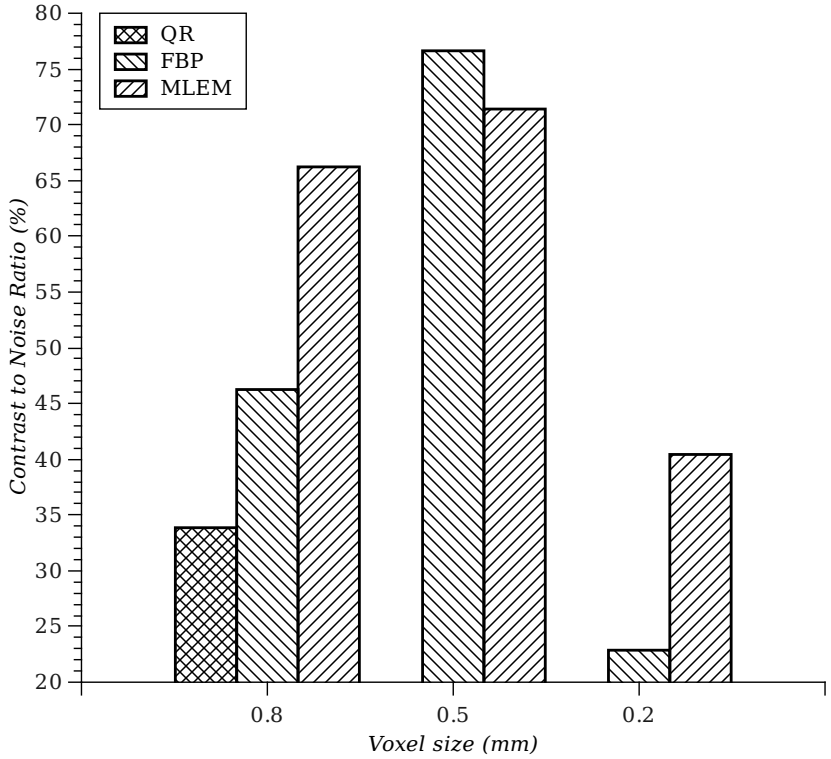


(b)

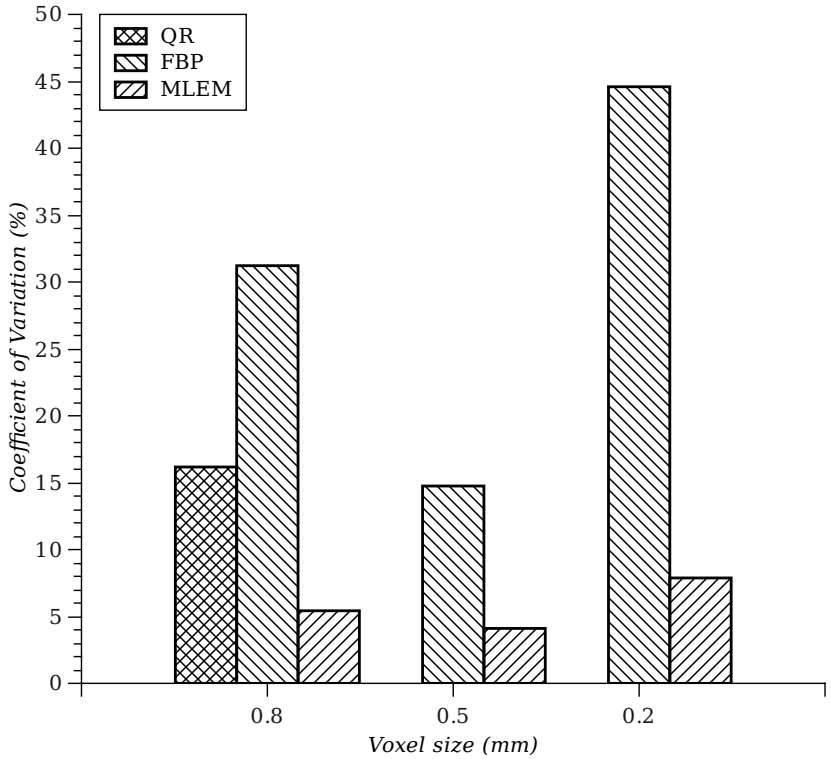
Figure 8.12: CNR (a) and CV (b) of the Teflon VoI for each algorithm and voxel size (using 0.2 mm detector size). CV is presented alongside CNR, in order to highlight noise contribution to CNR reduction.

that are part of transition between materials, some voxels inside the air VoI are influenced by surrounding PMMA. Regarding the MLEM results with 0.8 mm voxels, the CNR obtained is mainly due to the fact that attenuation coefficients obtained in the air VoI do not reach the low values obtained by FBP. Otherwise, QR results are comparable (as in low contrast VoIs) to those obtained by FBP or MLEM using 0.2 mm voxels.

As a summary, noise present in the reconstructed images with QR using 0.8 mm voxels is comparable to noise present in reconstructed images with FBP or MLEM using 0.2 mm voxels.



(a)



(b)

Figure 8.13: CNR (a) and CV (b) of the air VoI for each algorithm and voxel size (using 0.2 mm detector size). CV is presented alongside CNR, in order to highlight noise contribution to CNR reduction.

8.3 Sharpness analysis

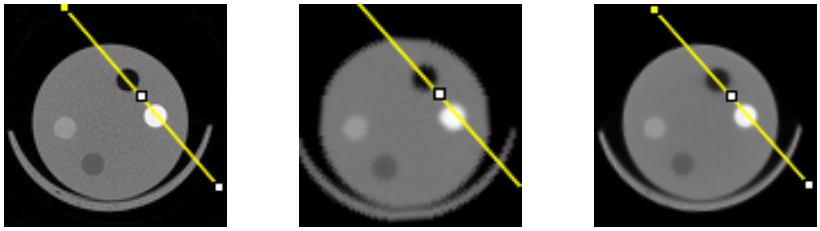
Several metrics have been proposed in the literature for the quantification of blur in an image. For example, variance [71] or kurtosis [72, 73] based metrics, among many others [74], have been proposed. However, these metrics are influenced by the amount of noise in the reconstructed image and in our case, are more sensitive to changes in voxel size or reconstruction algorithm than blur.

One of the effects of blur is the spread of the edges in an image. QR, FBP and MLEM show different *definition* in their reconstructed images. In Figures 8.14 (a), (b), (c), and 8.15 (a), (b), (c) are shown the central slices of the homogeneous regions phantom reconstructed with QR, FBP, and MLEM using a voxel size of 0.8 mm. In this case, differences in the edge definition can be perceived even with a naked eye. The idea of an analysis of the spread of the edges [75] has been adopted and adapted to our particular case, in order to quantify the definition of the reconstruction algorithms.

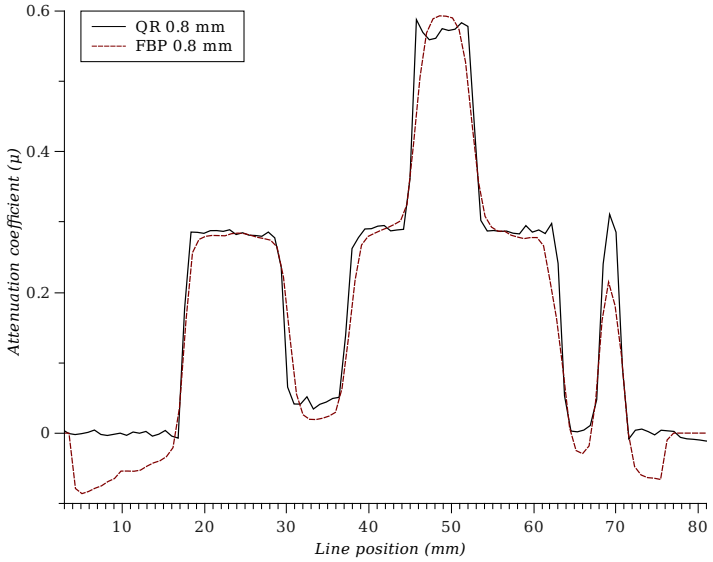
In Figures 8.14 and 8.15 is shown a line profile through the high contrast and low contrast inserts respectively. Also, the line profiles of FBP and MLEM are compared to the line profile of QR. Differences in smoothness of the transition between materials can be noticed. Regarding the blur, the main feature of the transition between materials is how *quickly* it is made. Ideally, transitions should be quickly and therefore, abrupt. However, smooth transitions can be observed, for example, in the case of Teflon insert reconstructed with FBP or air insert reconstructed with MLEM.

The Sobel operator [76] provides a gradient estimate at an image point by the vector summation of the 4 possible gradient estimates obtainable in a 3×3 point neighborhood. This operator has been widely used in image processing and nowadays, is available in almost every image software. Applying the Sobel operator to the central slice image of the homogeneous region phantom (see Figure 8.16 (a)) will lead to an image (see Figure 8.16 (b)) in which the transitions between materials can be easily characterized.

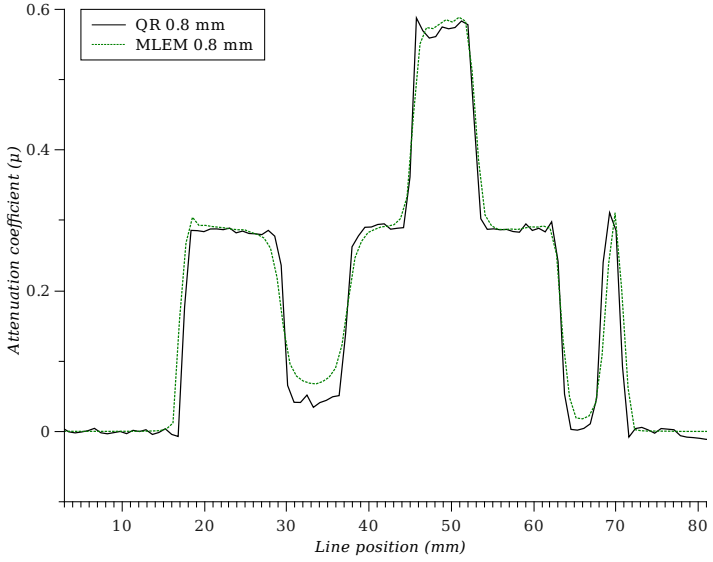
In Figure 8.16, a summary of the analysis of the spread of the edges is illustrated. The Sobel filter is applied to the original image, obtaining the gradient estimate image. Then, the same line profiles as applied in Figures 8.14 and 8.15 are applied to the gradient estimate image. These line profiles, now, show peaks in transitions between materials. The four central peaks (transitions into and out of the inserts) are selected (by cutting off the leftmost and rightmost peaks) and then, four Gaussian distributions are fitted to them (as seen in Figure 8.16 (c)). The full width at half maximum (FWHM) of these distributions represents the distance needed for the reconstruction algorithms to perform the transition between materials. Finally, the two FWHMs of the two transitions involving an insert are averaged, in order to obtain a single value for each material.



(a) (b) (c)

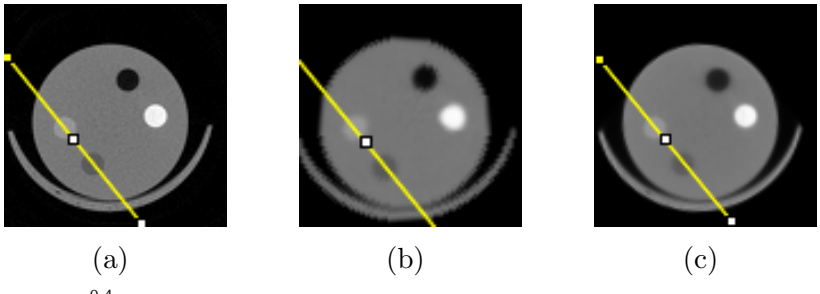


(d)

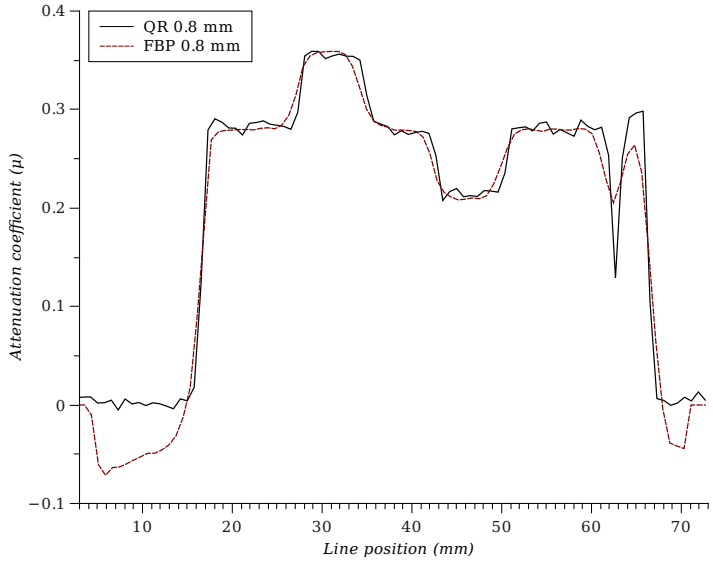


(e)

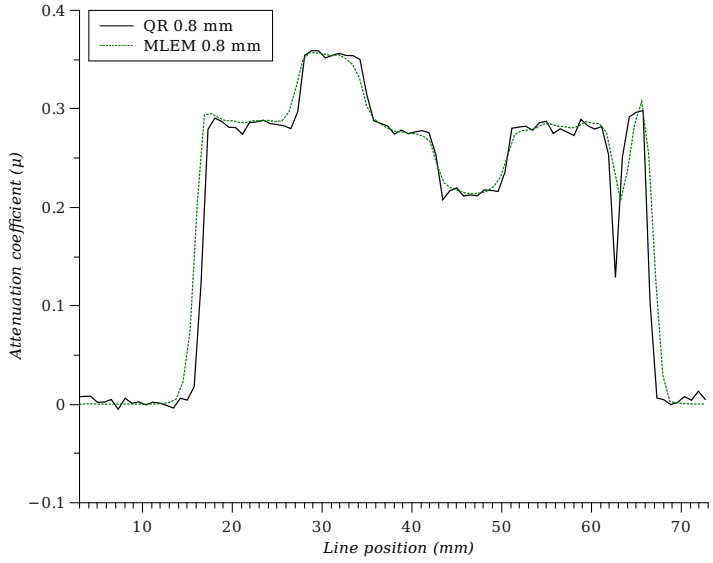
Figure 8.14: Line profiles through high contrast inserts at central slice of images reconstructed with QR (a), FBP (b), and MLEM (c). Line profiles have been separated into QR *vs.* FBP (d) and QR *vs.* MLEM (e), in order to simplify the plots. In the first valley (around 35 mm) are located the line values through the air insert and in the first peak (around 50 mm) are located the line values through the Teflon insert. Smooth transitions can be observed in the FBP profile of Teflon or the MLEM profile of air.



(a) (b) (c)

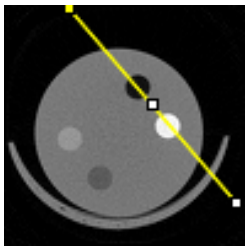


(d)

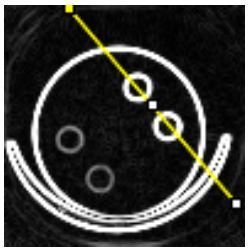


(e)

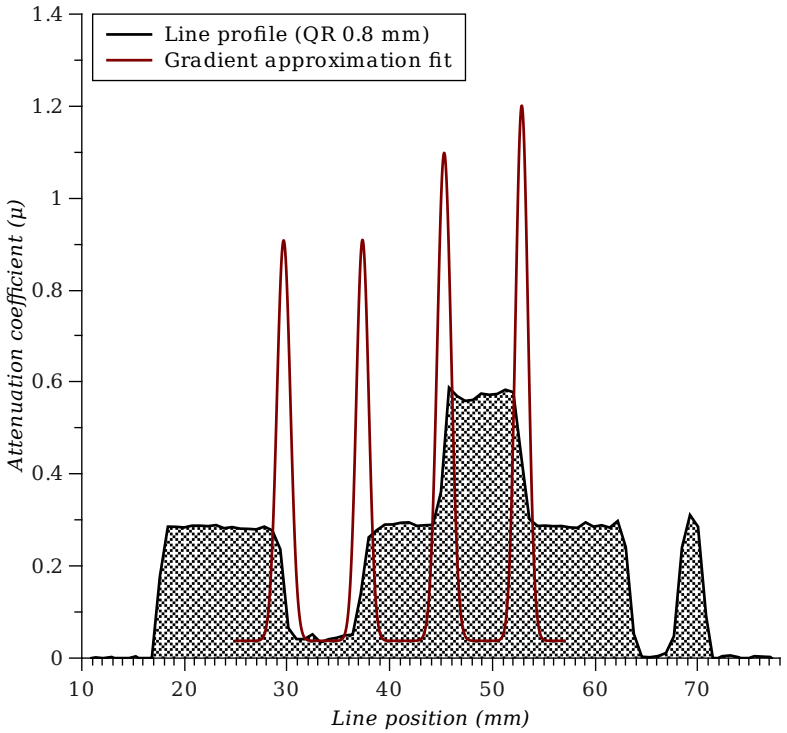
Figure 8.15: Line profiles through low contrast inserts at central slice of images reconstructed with QR (a), FBP (b), and MLEM (c). Line profiles have been separated into QR *vs.* FBP (d) and QR *vs.* MLEM (e), in order to simplify the plots. In the first peak (around 30 mm) are located the line values through the POM insert and in the first valley (around 45 mm) are located the line values through the PE insert. Smooth transitions can be observed in the FBP and MLEM profiles of both inserts.



(a)



(b)



(c)

Figure 8.16: Illustration of the characterization of transition between materials. In this example, image has been reconstructed with QR using 0.8 mm voxels. A line profile through the high contrast inserts at central slice (a) is selected, and shown as a filled curve in (c). In the first valley (around 35 mm) are located the line values through the air insert and in the first peak (around 50 mm) are located the line values through the Teflon insert. A Sobel operator is applied to the central slice to obtain a gradient estimate image (b) and the previous line profile is reevaluated to yield gradient estimates. Only the line segment between 25 and 57 mm (where the transitions involving inserts are located) is considered. Four Gaussian distributions are fitted to this line segment and shown in (c). The FWHM of the Gaussian distributions will be considered as the transition width.

The Gaussian distribution fit reasonably well to the gradient estimates, both to the high contrast gradients (see Figure 8.17) and to the low contrast gradients (see Figure 8.18). In our case, the most important feature of the gradient peaks is its width. Thin peaks represent short transitions, and therefore, sharp edges. The FWHM is a commonly used parameter that describes the width of a curve. Once fitted, the FWHM is a good choice to represent the width of the gradient estimate peaks. In Figures 8.17 and 8.18, the fit of gradient estimates from images with 0.8 mm voxels are shown (high contrast and low contrast respectively). The different peaks represent, from left to right, transitions inside and outside the first insert and inside and outside the second insert (air and Teflon in the high contrast case and POM and PE in the low contrast case).

For a better comparison between algorithms, fits to all gradients (high and low contrast) from images using 0.8 mm voxels are shown in Figures 8.19 (a) and 8.20 (a) respectively. In all cases, QR algorithm produces thinner gradients. And gradients produced by QR using 0.8 mm voxels are comparable to those produced by FBP or MLEM using 0.5 mm voxels (see Figures 8.19 (b) and 8.20 (b)). Considering this last case, it has to be noted that in some cases, FBP and MLEM produce thinner gradients than QR, but the increase in image size (from 0.8 to 0.5 mm) is considerable. In other words, with fewer voxels, QR is able to produce transitions as narrow as FBP or MLEM with almost half sized voxels.

Until now, a pair of gradient estimates have been shown for each material. As mentioned before this is due to the fact that an insert produces two transitions: one from the background to the insert and the other from the insert to the background again. Both transitions related to the same insert have similar FWHMs. This is best seen, for example, in the case of high contrast inserts (air and Teflon) reconstructed with MLEM (see Figure 8.17 (f)), due to the difference between the smoothness of the transitions involving air and Teflon. Therefore, the two FWHMs related to the same insert have been averaged, in order to produce an estimate of each material.

The final considerations of this analysis can be made from the transition width of each material, for each algorithm, as a function of the voxel size. These measurements are compiled in Figure 8.21. QR results correspond to images using 0.8 mm voxels (the only voxel size available for QR) and kept constant for the rest of voxel sizes for a better visual comparison (gray dashed line).

FBP results appear as continuous lines. Its results for low contrast materials using 0.2 mm voxel size are not included due to amount of noise present in the reconstructed image. The gradient width can not be estimated with this technique due to the similarity and proximity between gradients produced by noise fluctuations and gradients produced by transitions between low contrast materials. Nevertheless, narrow transitions are achieved (less than 1.5 mm FWHM).

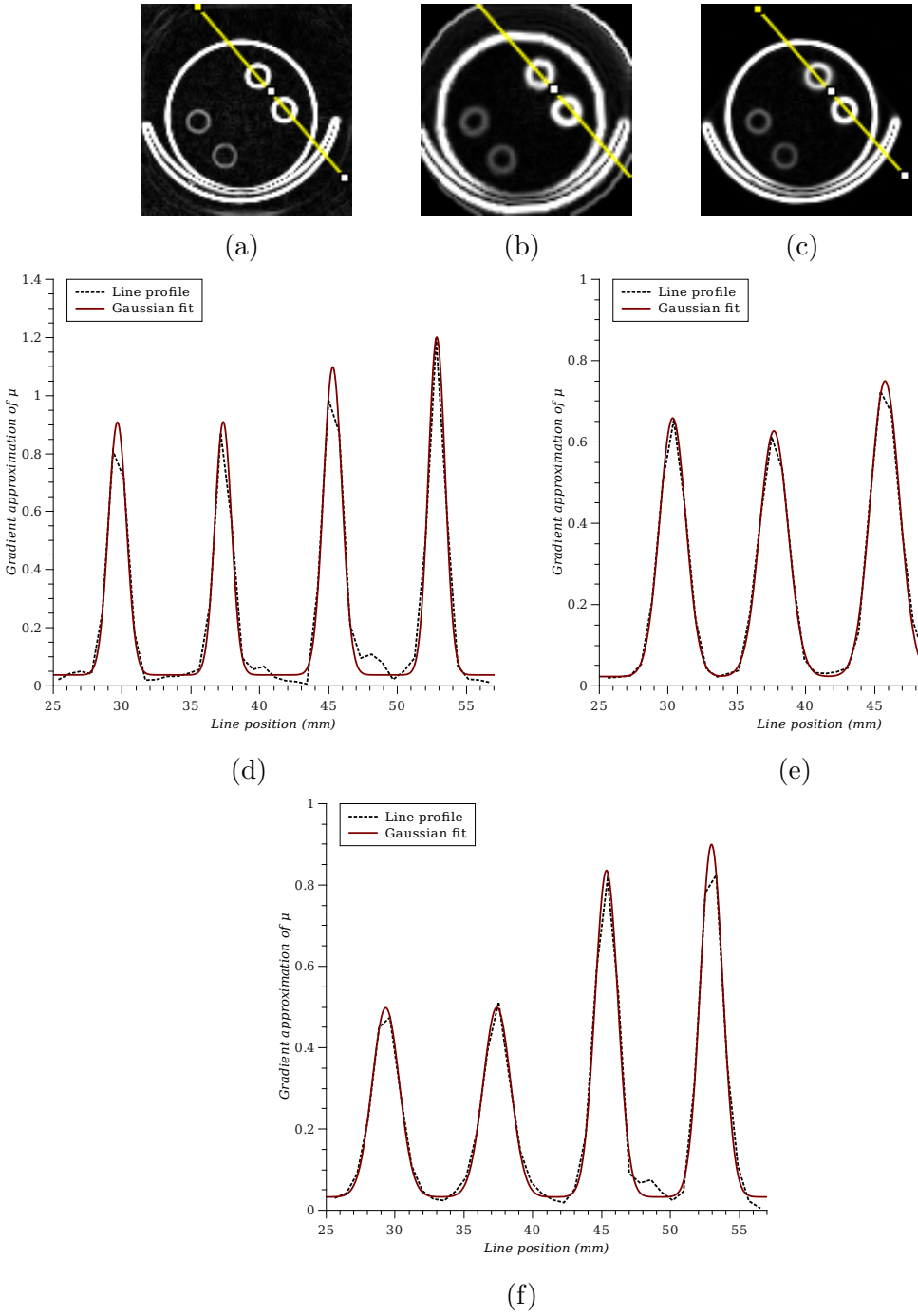


Figure 8.17: Gradient estimate fitting of line profiles through high contrast inserts. Gradient estimates of 0.8 mm voxel images reconstructed with QR (a), FBP (b), and MLEM (c) are shown. Gradient estimates of transitions involving inserts (air and Teflon) and obtained from QR (d), FBP (e), and MLEM (f) are plotted in dashed lines. Gaussian fits to each peak are plotted in continuous lines. The different peaks represent, from left to right, transitions inside and outside the air insert and inside and outside the Teflon insert, respectively.

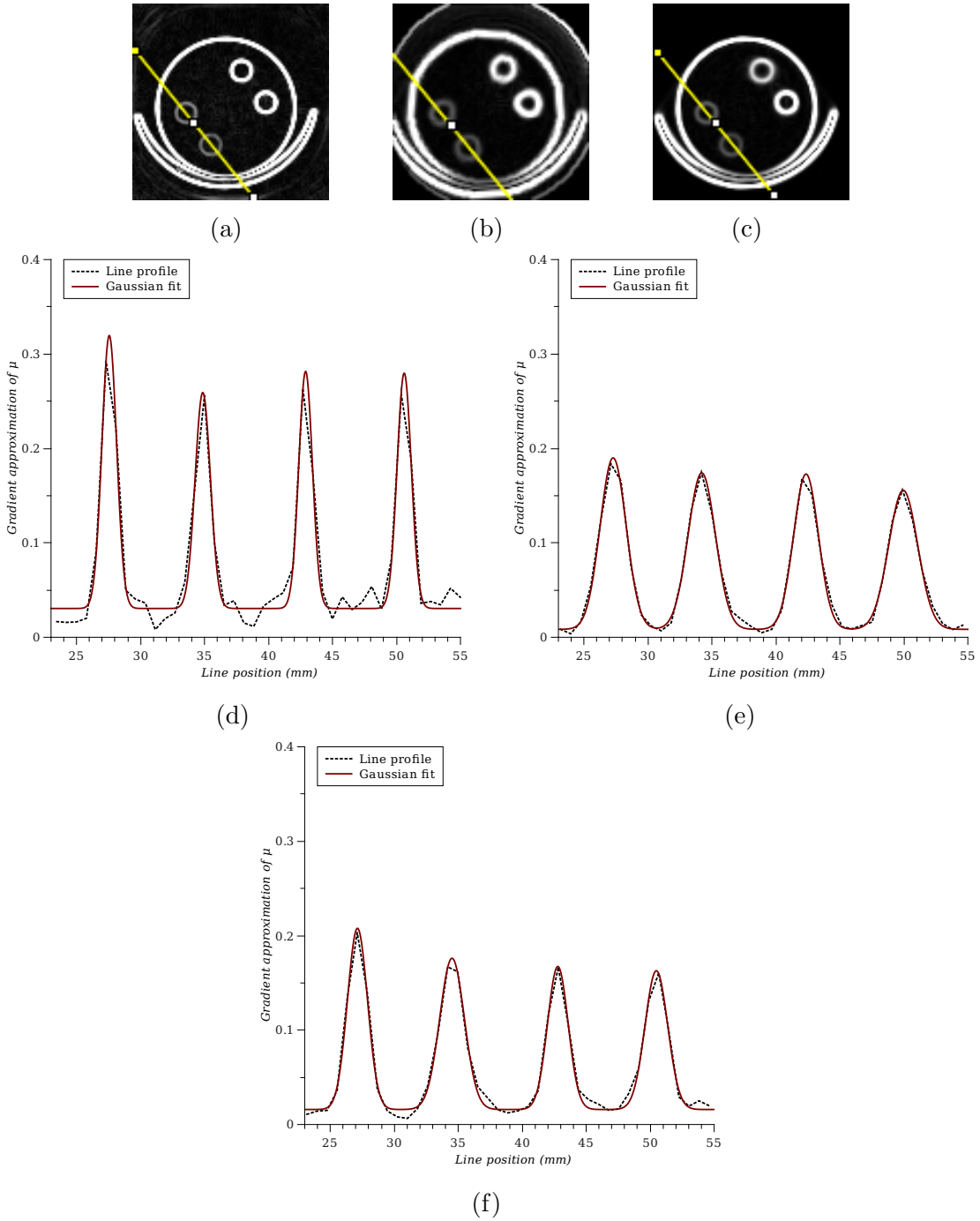
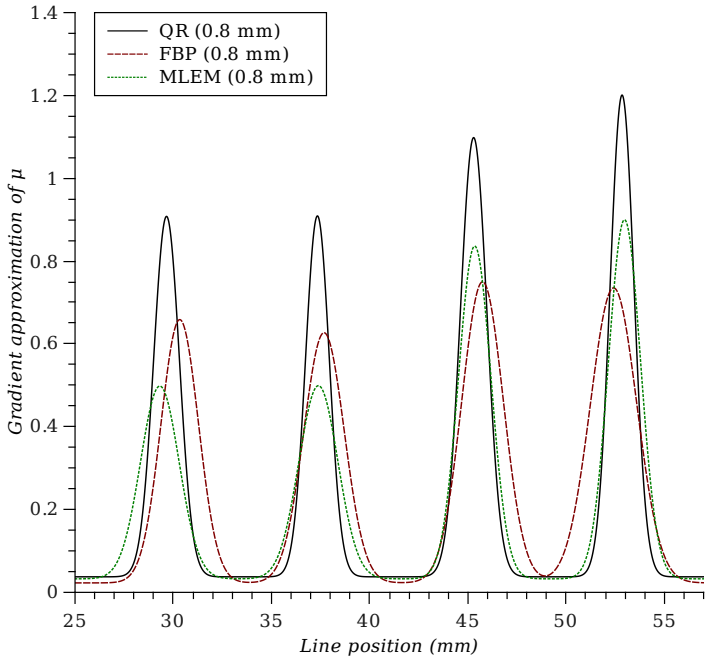
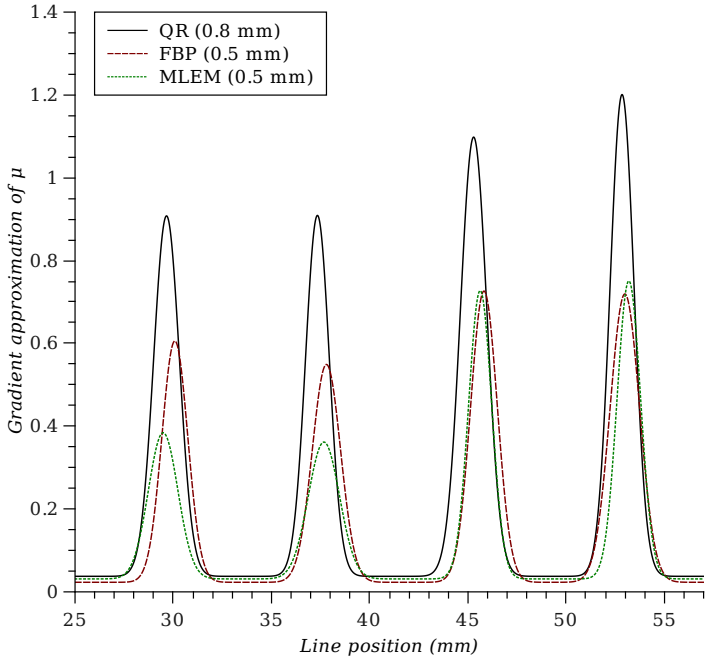


Figure 8.18: Gradient estimate fitting of line profiles through low contrast inserts. Gradient estimates of 0.8 mm voxel images reconstructed with QR (a), FBP (b), and MLEM (c) are shown. Gradient estimates of transitions involving inserts (POM and PE) and obtained from QR (d), FBP (e), and MLEM (f) are plotted in dashed lines. Gaussian fits to each peak are plotted in continuous lines. The different peaks represent, from left to right, transitions inside and outside the POM insert and inside and outside the PE insert, respectively.

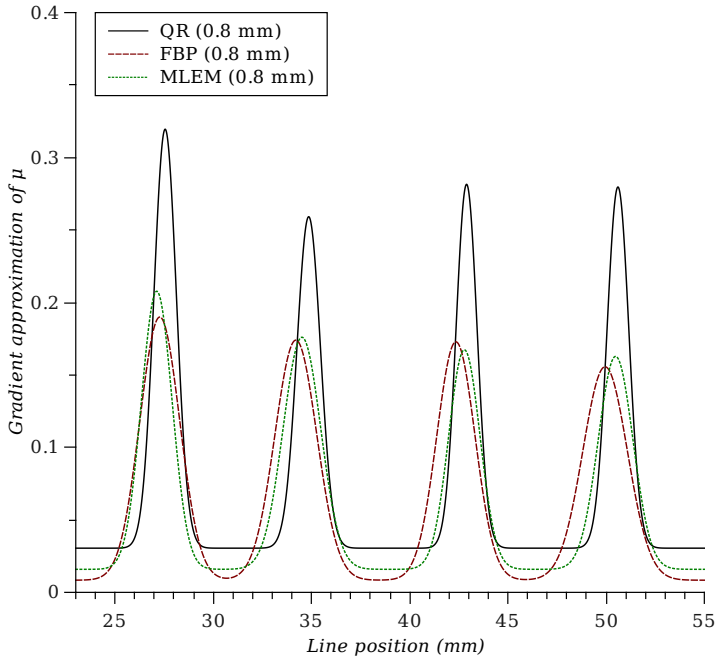


(a)

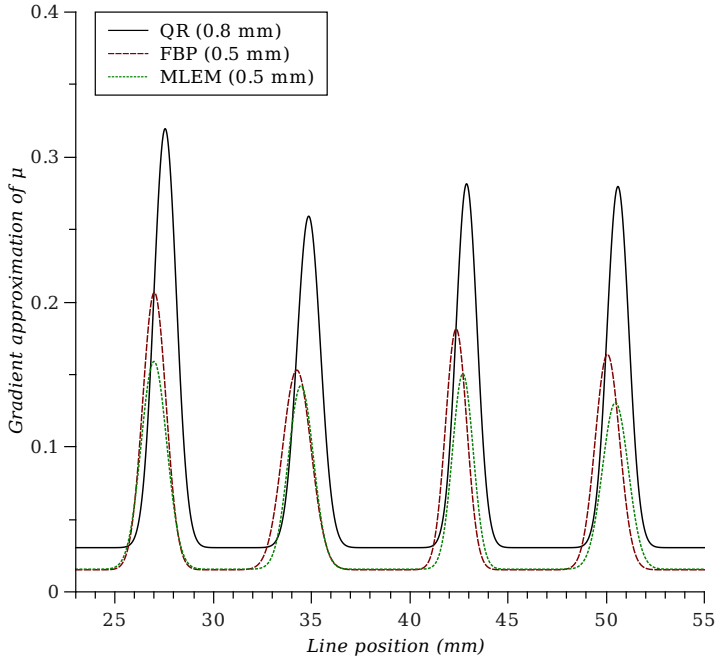


(b)

Figure 8.19: Gradient estimate fit comparison between QR, FBP, and MLEM, all from 0.8 mm voxel images (a) and QR from 0.8, and FBP and MLEM from 0.5 mm voxel images (b). Gradients represent transitions in a line profile through the high contrast inserts (air and Teflon).



(a)



(b)

Figure 8.20: Gradient estimate fit comparison between QR, FBP, and MLEM, all from 0.8 mm voxel images (a) and QR from 0.8, and FBP and MLEM from 0.5 mm voxel images (b). Gradients represent transitions in a line profile through the low contrast inserts (POM and PE).

Figure 8.22 presents the gradient estimate line profile through the low contrast inserts.

Dotted lines correspond to MLEM results. The extremely low level of noise fluctuations present in MLEM reconstructions allow to identify gradient estimates even from low contrast profiles of images using 0.2 mm voxels (while it is not possible with FBP).

As mentioned before, the FWHMs plotted for each material are the result of the average of two FWHM (produced entering and exiting each insert) and moreover, FWHM is not intended to quantify the exact distance needed for a transition. FWHM should be considered as an estimator that remains *almost* invariable to changes in noise level and other characteristics (introduced by changing the reconstruction algorithm or the voxel size, for example), and therefore, is specially useful in our case.

Similar results have been obtained for all materials aside of the air insert. Specifically, using 0.8 mm voxels, FBP needs approximately 2.5 mm to perform a transition, MLEM needs approximately 2 mm, and QR needs approximately 1.5 mm to change between materials. The case of air is slightly different. As in the case of Teflon with 0.8 mm in which FBP performed above 2.5 mm (smoother than with other materials), in the case of air with 0.8 mm, MLEM performed above 2 mm, obtaining approximately the same result as FBP. However, QR shown gradients around 1.5 mm width for all materials.

Concluding, considering 0.5 mm voxels, FBP and MLEM carry out transitions approximately in 1.5 mm. These results show that QR reproduces image edges as sharp as FBP or MLEM using almost twice the voxel size.

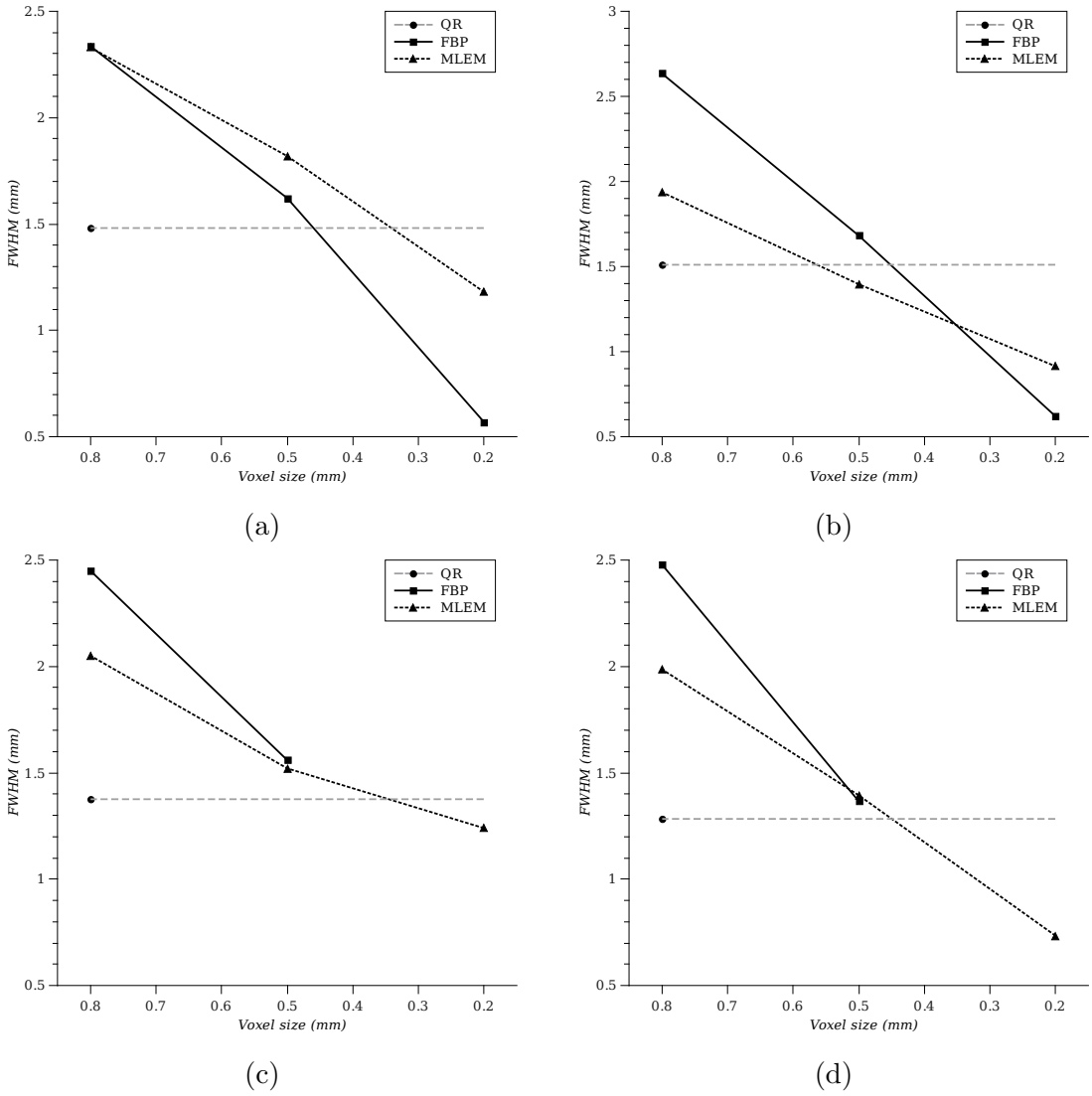


Figure 8.21: The average FWHM of the transition obtained for air (a), Teflon (b), POM (c) and PE (d) as a function of voxel size. QR results correspond to images using 0.8 mm voxels and kept constant for the rest of voxel sizes for a better visual comparison. The average FWHM of each material is the result of the average of its two FWHM (corresponding to the transition inside and outside the insert). POM and PE (low contrast inserts) are not available in the case of FBP (see Figure 8.22).

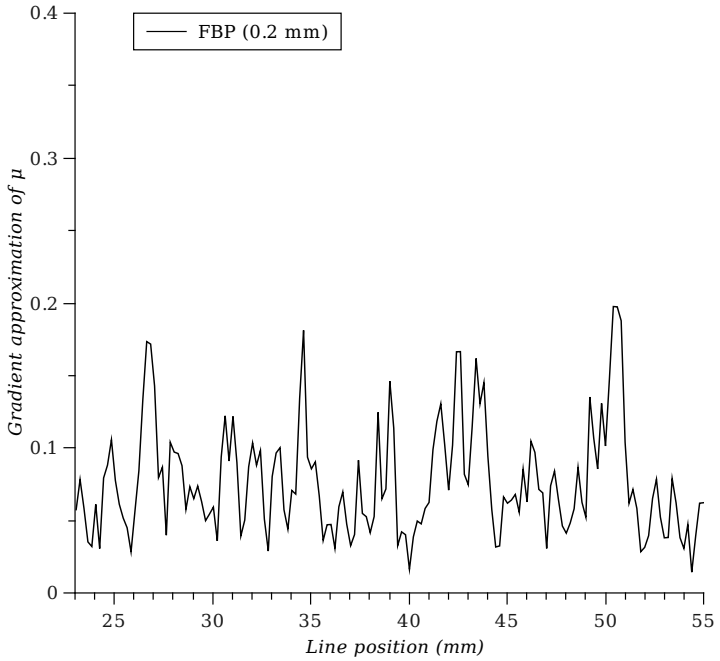


Figure 8.22: Line profile through low contrast inserts of the gradient estimate image obtained from the FBP reconstruction with 0.2 mm voxels. In this case, image noise produces gradients that obscure the transitions between materials and a Gaussian fit will provide FWHMs not related to the edge spreading. Nevertheless, lower edge spreading than QR (with 0.8 mm) can be assumed.

8.3.1 *In vivo* performance

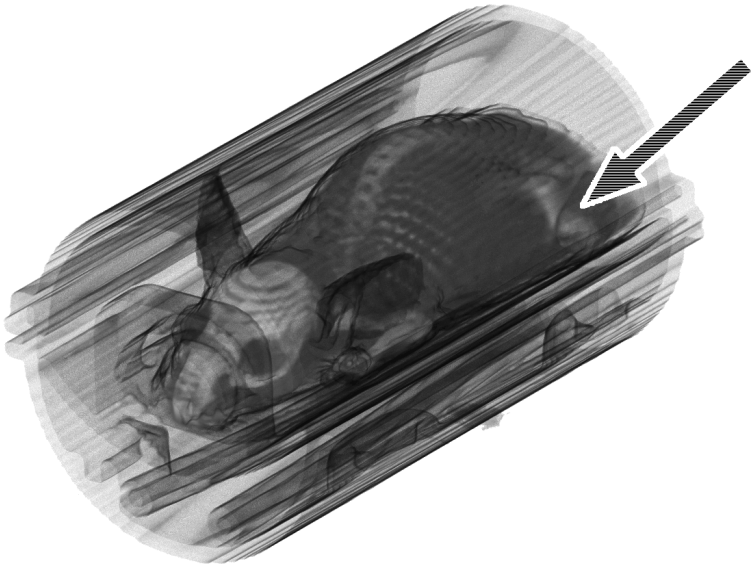
A common definition of image resolution is the smallest distance between two distinguishable objects. Image resolution is influenced by the spread of the edges, in the sense that two close peak could be indistinguishable if their transitions are smooth enough.

An *in vivo* measurement of a mouse has been reconstructed with all three methods, in order to verify the previously detailed results. This verification is carried out selecting a line profile through a fine feature region in knee of the mouse (see Figure 8.23). Specifically, 3D image reconstructions have been performed for each method (QR, FBP, and MLEM) and for 0.8, 0, 5, and 0.2 mm voxel sizes. The same slice in the axial direction has been selected in all images, yielding a cross section in the mouse femur, tibia, and fibula (see Figure 8.24).

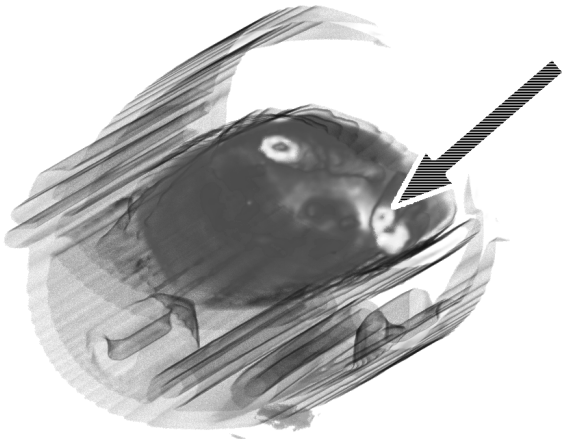
Considering the size of the detectors (0.2 mm) used for these reconstructions, the soft bone tissue of the mouse femur ($\sim$ 1 mm) must be observed, surrounded by the femur hard bone tissue ($\sim$ 1 mm to the left and to the right), and hard bone tissue for the tibia and fibula ($\sim$ 0.7 mm) must also be present. The femur has been considered as a coarse feature and the fibula has been considered as a fine feature. A line profile through the femur and fibula is enough to observe the presence of these coarse and fine features. This line profile must present peaks in hard bone regions and valleys in soft bone regions (or between bone pieces).

Three different resolution levels are expected. One, in which the femur is present ($\sim$ 3 mm total width), although, its interior soft bone tissue is not observable, due, partly, to the spread of the edges of the hard bone peaks. Other, in which the femur is present, its interior soft bone tissue is observable, but the fibula is not, due, partly, to the impossibility of the production of thin peaks with smooth transitions. Finally, a level of resolution in which all features are observable, namely, femur soft and hard bone and fibula hard bone tissues.

In Figure 8.25, the results of this test are summarized. Figure 8.25 (a) and Figure 8.25 (b) show axial slices obtained from 3D reconstructions performed with QR (0.8 mm voxels) and MLEM (0.5 mm voxels). As previous results indicated, these two images have comparable sharpness. Visually, almost the same features can be observed in the two images, with the exception of some regions with low attenuation coefficients, which are translated differently by these two algorithms. As the FWHMs obtained previously, QR can produce images as sharp and feature rich as MLEM, while using almost twice its voxel size. Numerically, in Figure 8.25 (f), the line plots for these two images are shown (black thick line for QR and joined squares with a green dashed line for MLEM). Its numerical values are similar, while MLEM values are closer to the desired result. The hard bone peaks translated by QR are not exactly where they should, due to the undersampling produced for the use of large voxel sizes. Also, MLEM numerical results show a small peak in the



(a)



(b)

Figure 8.23: 3D reconstruction of a mouse using QR method. 0.8 mm voxels have been used. The entire mouse (a) and an axial cut (b) are shown. The mouse is on a stretcher that provides it with anesthesia. The mouse left knee is pointed by an arrow in both cases. In the axial cut can be appreciated the soft bone tissue surrounded by hard bone in the mouse femur section. Soft and hard tissues have been enhanced for a better visibility of this feature.

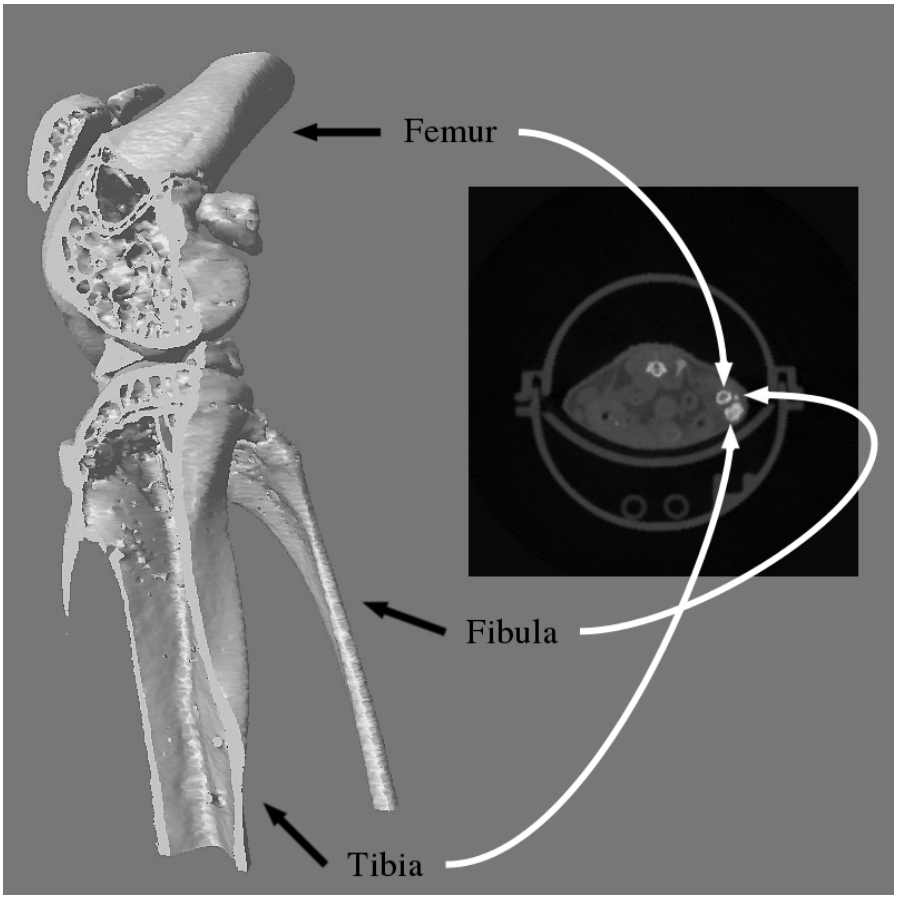


Figure 8.24: Illustration of a mouse knee anatomy† features present in the axial cut under study. As the mouse has been scanned with its leg bent, three bone sections will be appreciated: femur, tibia, and fibula. FBP with 0.2 mm voxels reconstruction (axial section) is also show to illustrate the bone piece correspondence. FBP has been used for this figure because, together with MLEM, are able to reconstruct all the features of interest (using 0.2 mm).

†Mouse knee 3D render was obtained from Bruker Corp. website: www.bruker.com .

fibula region while QR not, and both translated the femur soft bone tissue.

Figure 8.25 (a) and Figure 8.25 (c) are axial slices obtained from 3D reconstructions performed with QR (0.8 mm voxels) and FBP (0.5 mm voxels) respectively. While previous results estimated that these two configurations are similar, there are wider differences between QR and FBP than between QR and MLEM. Visually, FBP slice presents a lower edge definition than QR slice and, therefore, less features can be observed or there are more difficulties to observe them. Using almost twice the voxel size, QR reconstructs a sharper image than FBP. Numerically, in Figure 8.25 (e), the line plots for these two images are shown (black thick line for QR and joined squares with a green dashed line for FBP). The main numerical difference is in the transition involving the femur soft bone tissue. While QR is able to reproduce it, the FBP reconstruction is not. Numerical results obtained for the rest of tested voxel sizes are also included in Figure 8.25 (e) and 8.25 (f).

The highest resolution level proposed previously has not been achieved by the QR reconstruction. As FBP and MLEM are able to produce images with smaller voxel sizes, these two methods achieved it flawlessly while QR not. Although, one of the QR advantages is that has been able to produce images with approximately the same quality than MLEM or FBP with approximately twice its voxel size (0.8 mm *vs.* 0.5 mm). In conclusion, the reconstruction performed by the QR algorithm is more efficient in terms of image features (image information) / space, but always considering its limitation on the voxel size. In other words, with eight times (half in each dimension) less image sizes (or image points).

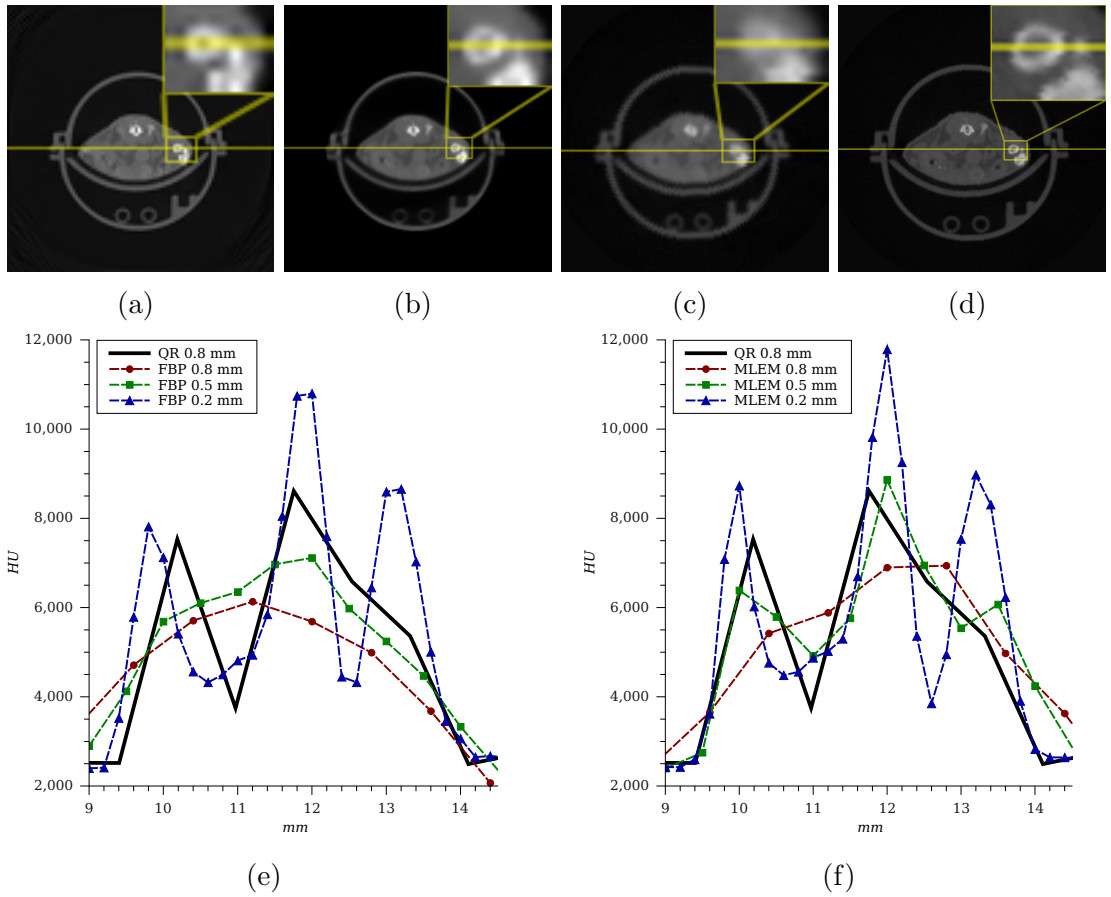


Figure 8.25: Line profiles through mouse femur and fibula. Profiles have been measured in the same axial coordinate from various images reconstructed with QR, MLEM, and FBP, using 0.8, 0.5, and 0.2 mm voxels (QR with 0.8 (a), MLEM with 0.5 (b), FBP with 0.5 (c), and FBP 0.2 mm (d) are shown). Profile plots begin right before entering the femur and end right after leaving the fibula. FBP plots (e) and MLEM plots (f) are separated for better visibility. The QR line profile is present in both plots for reference (thick black line). From left to right, in the 0.2 mm voxel reconstructions (joined triangles with blue dashed lines), three peaks can be appreciated. First and second correspond to hard bone entering and leaving the femur and third represents the fibula.

8.4 Noise power spectrum analysis

The power spectrum of an image determines the power of image signal at each spatial frequency and it is obtained through the discrete Fourier transform. When obtained from a noise image is called noise power spectrum (NPS) and determines the noise power at each spatial frequency. NPS is considered an analysis that completes the statistical noise analysis in an image quality study [77]. Information derived from NPS can be used to quantify image quality parameters such as noise grain (fine or coarse) or object detectability among others. Different frameworks that make use of the NPS have been treated in the literature: to evaluate the effects produced by sampling [78], by the aperture of the fan beam [79], or optimizing the image quality for a certain spatial frequencies of interest [80]. In the next section a more general multidimensional framework, also present in the literature, will be introduced. This framework will be used to compute the NPS of the images reconstructed with QR algorithm.

8.4.1 Multidimensional NPS framework

A multidimensional analysis of the NPS has been performed, following the guidelines described in [81]. Volumetric realizations have been extracted form 3D reconstructed images, their 3D NPS have been obtained and averaged to avoid statistical fluctuations, and, finally, the resulting 3D NPS has been analyzed. Reconstructed images have been converted to Hounsfield units (HU), in order to obtain the NPS in a more understandable units.

Specifically, three consecutive air measurements (I_A, I_B, and I_C) have been obtained in a short period of time to preserve similar scanning conditions. Each measurement has been reconstructed with QR, FBP, and MLEM. Real measurements are not ideal, and even from air measurements, result in reconstructions with non random noise structures. These noise structures would produce an increase in the low frequency regions when applying the discrete Fourier transform and even more, would led to inaccuracy in the NPS computation. Most of these noise structures will be eliminated if D_A^{QR}, D_B^{QR}, and D_C^{QR} are considered for the NPS computation, defined as:

$$\begin{array}{ll} D_A^{QR} & = I_A^{QR} - I_B^{QR} \\ D_B^{QR} & = I_A^{QR} - I_C^{QR} \\ D_C^{QR} & = I_B^{QR} - I_C^{QR} \end{array} \tag{8.5}$$

where I_A^{QR}, I_B^{QR}, and I_C^{QR} are the 3D reconstructed images with the QR algorithm, from I_A, I_B, and I_C measurements (and analogously for the FBP and MLEM algorithms).

Even with these cancellations, some noise structures would be present. In Figure 8.26 a slice of the air reconstruction using the QR algorithm is shown in the background. Contrast has been enhanced around air values to better appreciate noise patterns. A non random noise structure can be easily identified: a circle of approximately the size of the FoV (pixels outside of this circle are said to be outside the FoV in the sense that they are not covered by all the projections). Also, from this circle towards the image edges a variety of artifacts can be observed with this contrast level. These image regions, including the circle, must be avoided in the computation of the NPS. Moreover, a very attenuated ring artifact could appear in some slices producing an almost inappreciable noise structure in the center of the slice. Previous considerations motivated the election of the volumetric realization locations.

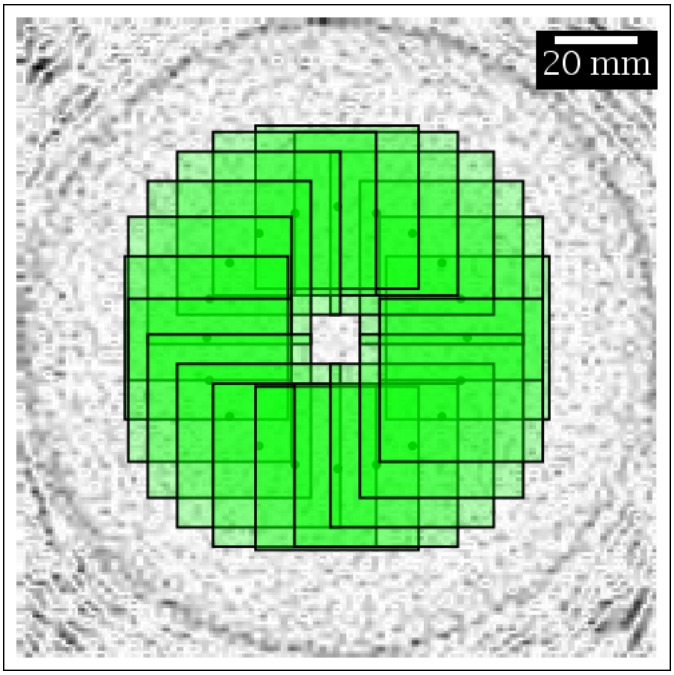


Figure 8.26: Illustration of twenty transaxial locations of the 3D realizations (transaxial cut). Image slice shown correspond to an air measurement reconstructed with QR method. This realization has been kept through the axial direction (perpendicular to the image).

In Figure 8.26 the transaxial locations of the realizations are illustrated. The realizations consist in $51 \times 51 \times 51$ image elements and are located forming a circle through the transaxial direction, at the same radial distance (32 mm) to the image center, and have been allowed to overlap between them (as in [82]). This transaxial distribution has been kept along the axial direction and conflicting image regions

previously described have been avoided.

A total of 120 realizations have been obtained for each reconstruction algorithm and have been zero-mean detrended by a 3D polynomial fit to

$$p(x,y,z) = a_1x^3 + a_2x^2 + a_3x + a_4y^3 + a_5y^2 + a_6y + a_7z^3 + a_8z^2 + a_9z + a_{10} \quad (8.6)$$

where x, y, and z are coordinates inside a realization. A window function has been applied to the realizations, in order to smoothly bring its edges down to zero. The Hann window function has been considered for this matter:

$$h(x) = \begin{cases} \frac{1}{2} + \frac{1}{2}\cos(2\pi x) & : \quad |x| \leq \frac{1}{2} \\ 0 & : \quad |x| > \frac{1}{2} \end{cases} \quad (8.7)$$

where x is the distance to the center of the realization, considered to range from -1 to 1. While the result of the application of this window function to an 1D realization is intuitive, the 2D and 3D cases are illustrated in Figure 8.27 for clarification.

At this point, a collection of zero-mean detrended data has been gathered and its power spectrum can be evaluated according to

$$S = \frac{b_x b_y b_z}{2N_x N_y N_z} \langle |DFT(d)|^2 \rangle \quad (8.8)$$

where d is a realization, b is the voxel width, N is the number of voxels in the realization, and brackets indicate the ensemble average to reduce fluctuations in the computed NPS. It must be noted that the normalization coefficient in Equation 8.8 is divided by 2. This accounts for the definition of images in Equation 8.5 in which each image contains the noise of two measures.

As a summary of the whole process, the square of the modulus of the 3D discrete Fourier transform of each 3D realization is computed. These results form a statistical ensemble, and are averaged in order to reduce statistical fluctuations. Finally, a normalization is applied, and the 3D NPS is obtained.

8.4.2 NPS results

NPS is obtained in volumetric space and there are several representations extracted from the 3D NPS that help with the interpretation of the spectral density. For example, the central slice in the axial axis (XY plane), contains information about the distribution of noise in the transaxial plane. Usually, this information is presented as a line profile of the NPS from the zero frequency to the Nyquist frequency. In Figure 8.28, this plot is shown containing the results obtained from the same measurement reconstructed with QR (black joined triangles), FBP (red joined circles), and MLEM (green joined squares). In all cases, a small spectral

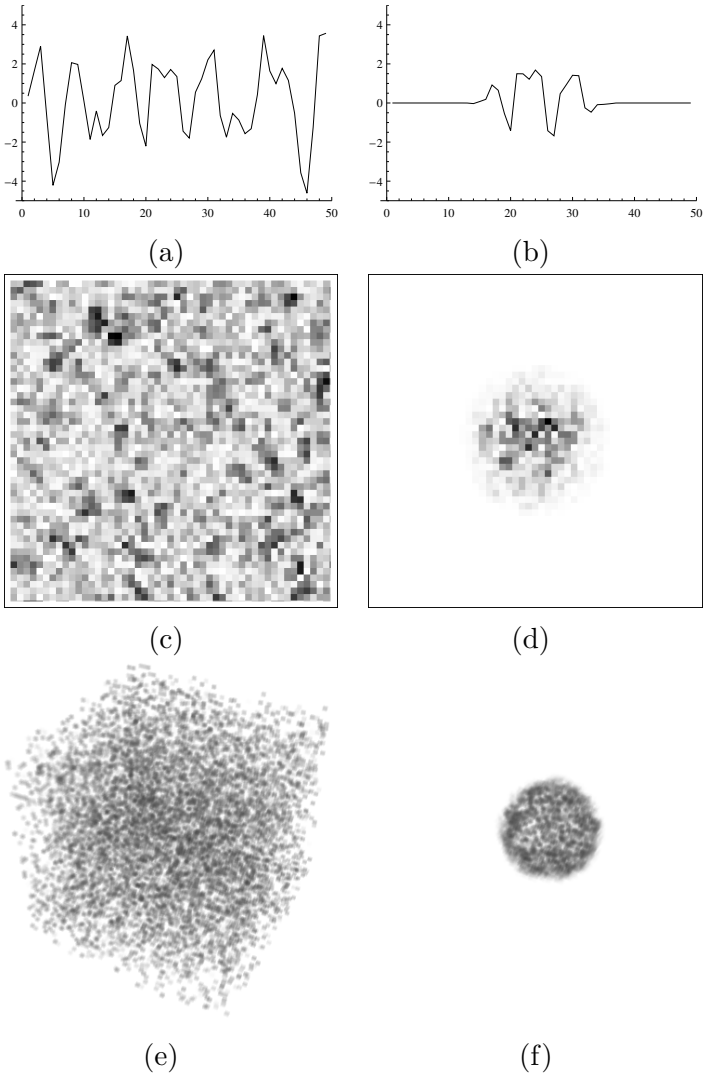


Figure 8.27: Realization examples. 1D (a), 2D (c), and 3D (e) zero mean detrended realizations and its results after the application of the Hann window (see Equation 8.7) (b), (d), and (f) respectively.

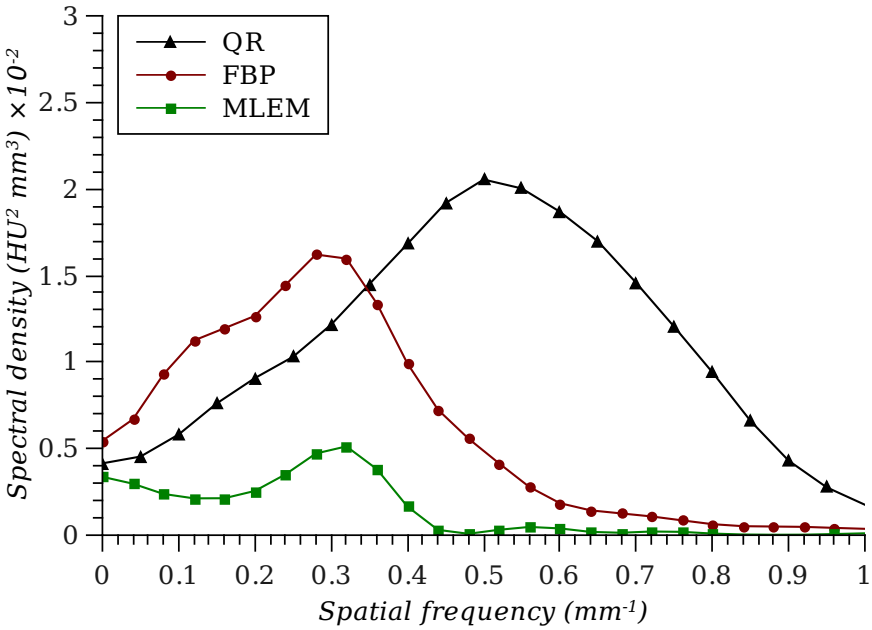


Figure 8.28: Profile plot of NPS through X axis in the central slice of the 3D NPS (S_X). Noise predominance can be observed around mid frequencies for FBP and MLEM (although with a very low spectral density in the case of MLEM). QR noise is shifted to higher frequencies and with the highest spectral density of all three methods.

density can be observed in the zero frequency, due to the failure to remove completely all noise trends.

Regarding the FBP, its shape corresponds to previous analysis performed in the literature by other authors [83], characteristic of images reconstructed by cone-beam FBP. Spectral density increases at low and mid-frequencies because of images reconstructed with FBP are ramp filtered. The spectral density decay at high frequencies is due to band-limiting processes such as image blur or the application of the Hann window (the same Hann window has been applied to the realizations of all three algorithms). The FBP NPS can be classified as spectrally *green* (spectrum has its highest values at mid-frequencies). This expected result will be of use as a reference to compare against QR, as image characteristics of cone-beam FBP reconstructions are well known [7].

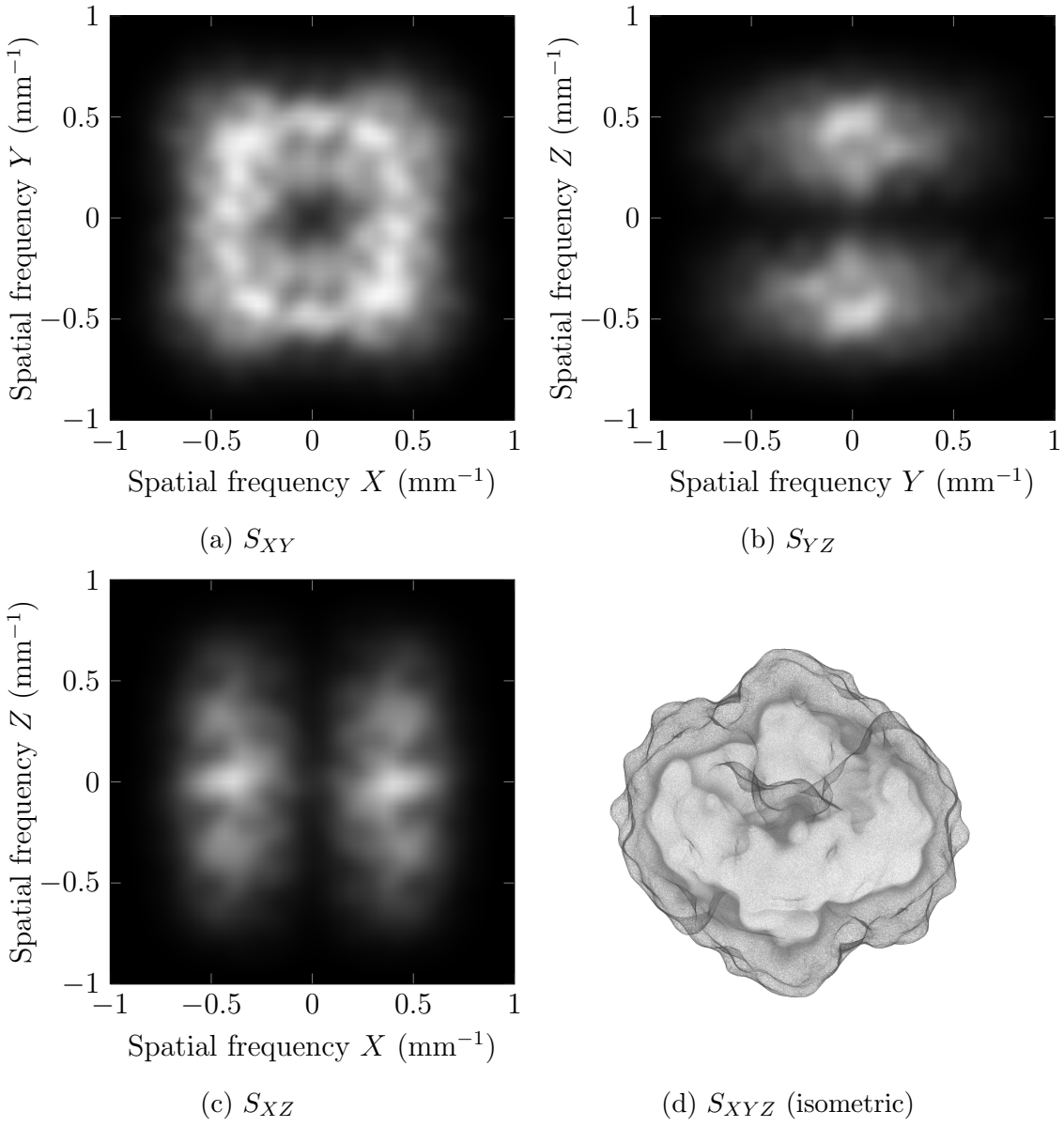
QR curve, shown in Figure 8.28 reveals a shifted spectral density to high frequencies. With comparable (although slightly lower) mid frequencies, high frequency values are notably higher than FBP. The QR NPS can be viewed as spectrally *bluer* (spectrum has its highest values at high-frequencies) than NPS of FBP

due to spectrum peak position, but there it is also a decay at high frequencies due to the application of the Hann window.

On one hand, in Section 8.2 where a comparative study of CV and CNR between QR, FBP, and MLEM is shown, QR exhibits notably higher noise level than FBP, but NPS locates this noise at higher frequencies. Therefore, QR produces a higher level of noise than FBP but finer grained. On the other hand, QR seems to produce less blurry images, as supported in Section 8.3 (where a comparative study of the edge spreading between QR, FBP, and MLEM is shown), where QR presented finer feature translation than FBP. This decrease in blurring could be one of the factors of the accentuated spectral density at high frequencies. In other words, higher NPS at high frequencies could reflect the ability to reproduce fine structures.

Regarding the MLEM, its spectral density peak is close to the peak of the FBP. Although, the most remarkable feature is the lower spectral density in all frequencies, compared to FBP or QR. Nevertheless, not all the high-frequency spectral density should be attributed to the fine feature translation. In fact, in Section 8.3, MLEM showed transitions as fine as those produced with QR (or even finer) and its spectral density at high frequencies is almost null. This could be due to an incomplete background trend removal. Evidences that its NPS is not radially symmetric in the transaxial plane can be found in the corresponding NPS slice (see Figure 8.31). Considering gathered data from Figures 8.28 and 8.31 it could be possible that incomplete background trend removal may have corrupted the NPS spectrum [84].

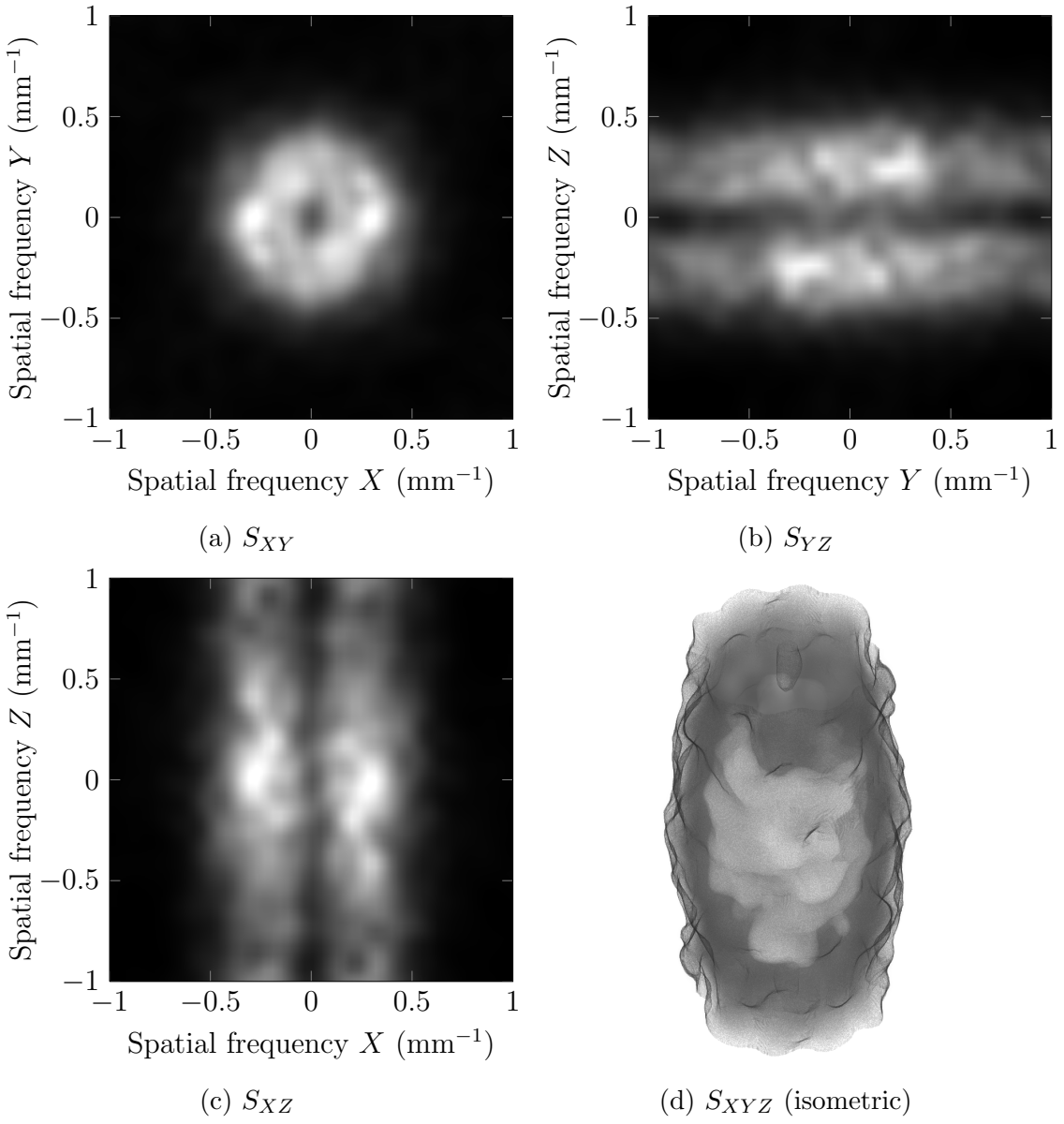
In Figures 8.29 and 8.30 orthogonal slices of 3D NPS obtained from QR and FBP reconstructions are shown. The QR and FBP distributions of the 3D NPS are similar, except for the shift to higher frequencies in the XY plane (as seen in Figure 8.28) and the shape of the NPS in the Z axis. In the case of FBP, the shape of the NPS in the Z axis tends to be *whiter* (keeps certain spectral power through almost all frequencies) since FBP ramp filter is not applied over Z. Although, in the case of QR, NPS shape along Z tends to decrease at high frequencies (as in X or Y directions). This produces a cube shaped 3D NPS for QR, while a cylinder shaped 3D NPS is obtained with FBP.



(a) S_{XY} (b) S_{YZ}

(c) S_{XZ} (d) S_{XYZ} (isometric)

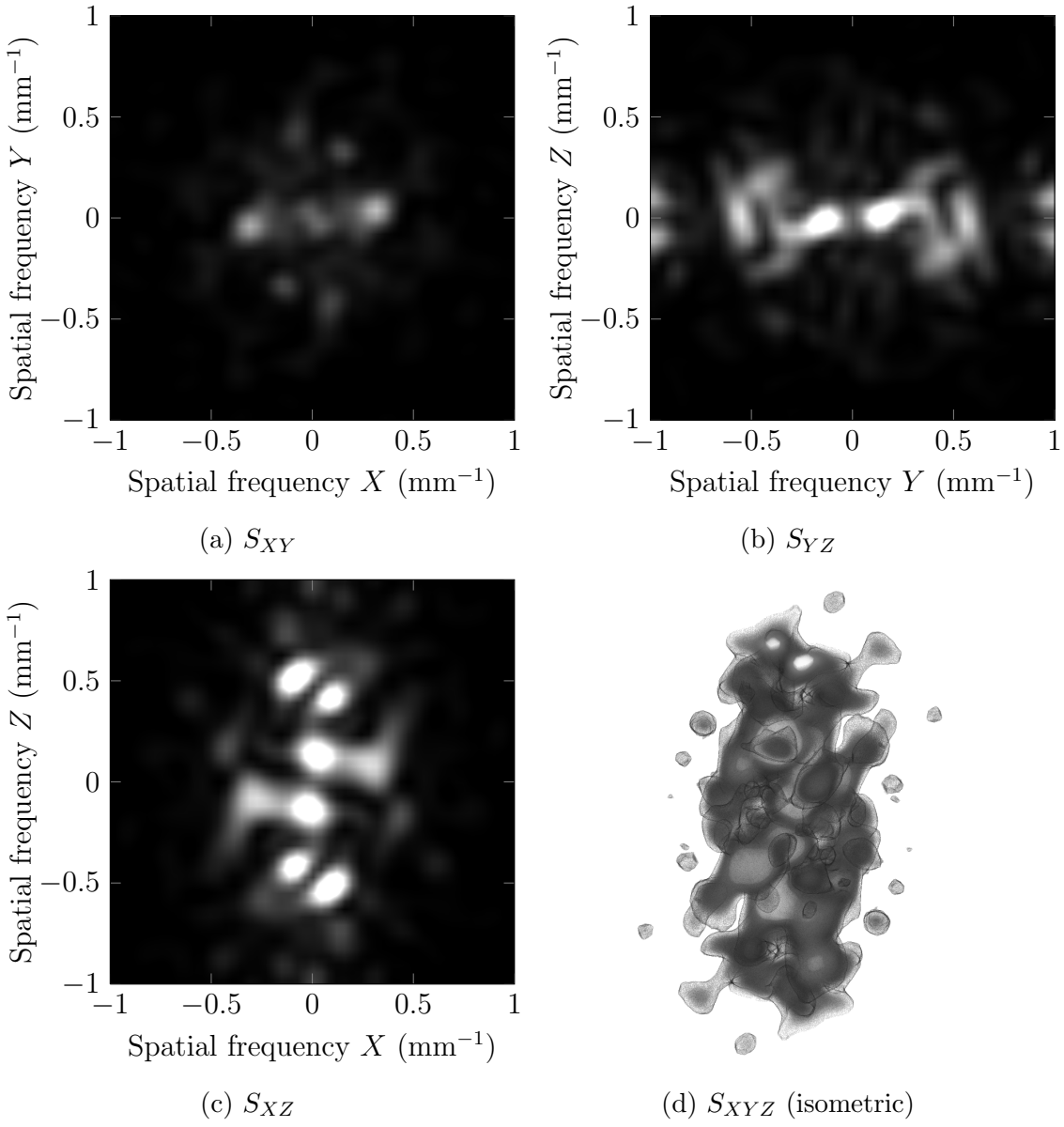
Figure 8.29: Representation of the 3D NPS obtained from the QR air reconstruction. Orthogonal views are presented, corresponding to XY (a), YZ (b), and XZ (c), along with a 3D render (d) of the 3D NPS in a isometric view. The 3D NPS render has emphasized and slightly shaded surface for visualization purposes.



(a) S_{XY} (b) S_{YZ}

(c) S_{XZ} (d) S_{XYZ} (isometric)

Figure 8.30: Representation of the 3D NPS obtained from the FBP air reconstruction. Orthogonal views are presented, corresponding to XY (a), YZ (b), and XZ (c), along with a 3D render (d) of the 3D NPS in a isometric view. The 3D NPS render has emphasized and slightly shaded surface for visualization purposes.



(a) S_{XY} (b) S_{YZ}

(c) S_{XZ} (d) S_{XYZ} (isometric)

Figure 8.31: Representation of the 3D NPS obtained from the MLEM air reconstruction. Orthogonal views are presented, corresponding to XY (a), YZ (b), and XZ (c), along with a 3D render (d) of the 3D NPS in a isometric view. The 3D NPS render has emphasized and slightly shaded surface for visualization purposes.

8.5 Time complexity of the QR solution

Time required to obtain the reconstructed image is a key feature of the reconstruction algorithm. Nowadays, reconstructed images are demanded in a short period of time after the measurement is finished, and this is crucial to the efficiency of a CT service. QR algorithm offers a valuable advantage in this matter. As seen in Section 2.2, its computational load is performed once the CT system is modeled as a matrix (A) to compute the factorization ($A = QR$). In the reconstruction time only two processes need to be addressed: the update of the measurement, b, (applying the stored rotations to b) obtaining $Q^t b$ and a back substitution to obtain the reconstructed image (x) for the previously computed vector.

Usually, time complexity of the whole process is presented in the literature for dense matrices [27]. Let $A \in \mathbb{R}^{m \times n}$ and $m > n$ then, the whole process requires $\mathcal{O}(mn^2)$ (for the factorization), plus $\mathcal{O}(mn)$ (for the update of b), plus $\mathcal{O}(n^2)$ (for the back substitution). This cost is governed by the factorization time and it is said that the entire computation of a solution requires $\mathcal{O}(mn^2)$. However, in the CT image reconstruction, the factorization is computed *a priori* and therefore, image reconstruction time is governed by the update of b that requires $\mathcal{O}(mn)$.

This time complexity analysis overestimates the time required to obtain a solution in the sparse case. The number of Givens rotations needed for the entire factorization should be considered to characterize a tighter bound to the sparse case. In fact, one of the advantages of Givens QR in the sparse case is its flexibility to introduce zeros anywhere in A and Section 6.2 is devoted to alternatives to minimize the fill-in during the factorization process of CT system matrices.

Let r be the number of rotations needed for the factorization $A = QR$ then the whole process requires $\mathcal{O}(rn)$ (for the factorization), plus $\mathcal{O}(r)$ (for the update of b), plus $\mathcal{O}(n^2)$ (for the back substitution). In the dense case $\mathcal{O}(r) = \mathcal{O}(mn)$, required times agree with those previously mentioned, and as $m > n$ the image reconstruction process is still governed by the update of b. In the sparse case (for CT model matrices), and considering the best case scenario where an ideal strategy that avoided all fill-in during the factorization is applied, $\mathcal{O}(r) = \mathcal{O}(n_{nnz})$, where n_{nnz} is the number of non-zeros present in the CT model matrix. Then,

$$\mathcal{O}(n_{nnz}) < \mathcal{O}(r) < \mathcal{O}(mn) \tag{8.9}$$

is the time complexity of the reconstruction process expressed as a function of the number of Givens rotations needed.

From this point of view follows that time required for image reconstruction strongly depends on the model and the fill-in avoidance strategy. CT system model matrices that present symmetries (such as presented in Section 3.3), combined with well suited fill-in reduction strategies that take advantage of its non-zero structure would drastically decrease the time required to produce images.

In order to clarify the performance of the implemented algorithm, reconstruction times have been measured in a computer with an Intel Core i7–4930K with 12 threads at 3.9 GHz clock frequency. FBP and MLEM algorithms are fully parallelized, making use of 11 of the 12 available threads in the processor, while the current implementation of the QR algorithm has a *naive* back substitution process without any parallelization and a $Q^T b$ product using one thread for computation and other for data loading into memory. Obtained computation times are 0.125 s in the case of FBP, 9 s in the case of MLEM (0.225 s / iteration), and 163 s in the case of QR (79 s for the $Q^t b$ product and 84 s for the back substitution). Although obtained reconstruction times may seem to be a drawback of the QR-decomposition algorithm, the difference between the FBP and MLEM, and QR times is not only an inferior (or non) parallelization of the current implementation, but to the amount of fill-in produced during the reduction of the system matrix to triangular form.

This computation times, together with the complexity analysis carried out in this section, offer a view of the potential reconstruction time reduction with the research of better fill-in reduction strategies.

Chapter 9

Conclusions

Considering the proposed objectives, the major solutions implemented to achieve them will be summarized in this section.

The computation of the volume intersection between a LoR (in an arbitrary projection) and a voxel is not straightforward. In §3.1.5, an algorithm has been defined for the computation of the exact volume intersection between a LoR and a voxel. This procedure only requires the LoR and a voxel polyhedra with their faces sored in CCW order, which can be trivially defined when coming from a regular grid as the voxellation of the space occupied by the object under a CT scan, or the matrix of detector elements that form a CT flat panel. Moreover, this algorithm allows more complicated voxellations that could be of some advantage, being the only problem for the exact volume computation, the definition of the voxel and LoR polyhedra. To the best of our knowledge, this is the first work that proposes an algorithm with such a general input.

In §3.3, during the definition of the linear system that solves the CT reconstruction problem, a symmetry common to all third generation CT has been detailed. The exploit of this symmetry allows solving the entire system by the reduction to triangular form of a half rows, half columns matrix. Its drawback is the resolution of the same system for two right hand sides, but this additional cost is much lower than the cost of the reduction to triangular form of a matrix four times larger. To the best of our knowledge, this is the first work that describes a symmetry that expresses the system matrix as a block matrix and therefore, allows methods like QR-decomposition to take advantage of it.

The evolution of the condition number of the linear system has been also studied. The accuracy in the solution depends on the condition number, and the condition number tends to decrease as the number of equations increases. This evolution performs at different speed, depending on the equations coming from new projections or from the reduction of the size of the detector elements in the flat panel. Results show advantageous condition numbers when considering equa-

tions defined by a large number of detector elements rather than a large number of projections.

Any reduction process, as the QR decomposition, requires a large amount of modifications in the values of the original matrix. In §4.3, a sparse matrix scheme has been entirely detailed, from its elementary operations to a `get`(i,j) / `set`(i,j) interface to operate with the matrix as a black box. This scheme is easy to implement (maintain and update) and C++ source code has been provided to avoid ambiguities. The main advantage of this scheme over other existing schemes is its fully orientation to offer balanced and reduced costs in all operations, including the element deletion (storing a zero and releasing memory) or creation (replacing a zero with a non-zero value). All operations run in $\mathcal{O}(\log n)$, where n is the number of non-zero elements of the row involved in the operation.

Code for the rotation of two rows of a generic matrix has been provided in §5.3. The code assures the continuity in the computation of the rotation as up-to-date implementations in BLAS or LAPACK. Also, an heuristic strategy to minimize the fill-in during the reduction has been detailed in §6.2. This strategy selects the rows to be rotated exploiting the particular structure of the system matrix. A bound of the fill-in produced using this strategy has been obtained and its performance has been compared to the standard fill-in reduction strategies in realistic CT model matrices, outperforming standard methods in this case. To the best of our knowledge, this is the first work that presents an specialized fill-in reduction strategy for sparse matrices derived from the CT image reconstruction problem. All implementation details have been addressed and C++ source code has been provided to remove ambiguities.

A basic and simple-to-implement strategy to parallelize the $Q^T b$ product, critical in the time consumption of the image reconstruction process, has been provided in §7.3. The parallelization consists of a circular buffer to address the loading cost of the likely large Q^T file.

The details of the image quality of the solution of the linear system through the QR decomposition are shown in Chapter 8. A complete analysis has been performed considering a linearity check of the attenuation coefficients, noise measures based on the standard deviation, noise power spectrum analysis, and performance with measured CT data of a mouse. All results have been compared against the CT image reconstruction dominant algorithm, FBP, and a state of the art model based iterative reconstruction algorithm, MLEM. Also, an image blurring analysis has been performed, showing that the QR algorithm introduces less burring in the reconstructed images if an equal voxellation is imposed, and showing the potential of the QR algorithm.

A bound of the required time for the image reconstruction process as a function of the number of rotations needed to reduce A to triangular form is shown at the

end of Chapter 8. This bound shows the potential cost reduction if better fill-in reduction strategies are developed.

Finally, as a result, a proof of concept implementation of a CT image reconstruction algorithm has been obtained. This algorithm is model based, and does not relies on an iterative procedure to obtain the reconstructed image. Instead, the image is obtained by a matrix vector multiplication and a backward substitution process. A QR decomposition must be previously performed, but it is required only once, since each image reconstruction corresponds to the resolution of the same CT system for a different right hand side. Although high resolutions could not have been achieved yet, obtained results also demonstrate the prospective of this algorithm, as great performance and scalability improvements would be achieved with the success in the development of better fill-in strategies or additional symmetries in CT geometry.

Bibliography

[1] "Normal barium swallow." Obtained from Wikimedia Commons. Available at: https://commons.wikimedia.org/wiki/File:Normal_barium_swallow_animation.gif [Accessed Nov. 2015].

[2] J. Langner, *Event-Driven Motion Compensation in Positron Emission Tomography: Development of a Clinically Applicable Method*. PhD thesis, Faculty of Medicine Carl Gustav Carus, Technische Universität Dresden, Germany, November 2008.

[3] J. Cal-González, J. L. Herraiz, S. España, E. Vicente, A. Sisniega, J. J. Vaquero, and J. M. Udías, "Extraction of Information from CT Images for PET and Radiotherapy Planning." IV Encuentro de Física Nuclear, El Escorial, Madrid, Spain, September 2010.

[4] R. Gordon, R. Bender, and G. T. Herman, "Algebraic Reconstruction Techniques (ART) for three-dimensional electron microscopy and X-ray photography," *Journal of Theoretical Biology*, vol. 29, pp. 471–481, Dec. 1970.

[5] J. Radon, "On determination of functions by their integral values along certain multiplicities," *Ber. der Sachische Akademie der Wissenschaften Leipzig,(Germany)*, vol. 69, pp. 262–277, 1917.

[6] L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone-beam algorithm," *Journal of the Optical Society of America A*, vol. 1, p. 612, June 1984.

[7] J. Hsieh, *Computed Tomography, Second Edition*. 1000 20th Street, Bellingham, WA 98227-0010 USA: SPIE, 2nd ed. ed., Oct. 2009.

[8] S. Basu and Y. Bresler, "O(N(2)log(2)N) filtered backprojection reconstruction algorithm for tomography.," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 9, pp. 1760–73, Jan. 2000.

[9] Y. Ye, S. Zhao, H. Yu, and G. Wang, "A general exact reconstruction for cone-beam CT via backprojection-filtration," *IEEE Transactions on Medical Imaging*, vol. 24, pp. 1190–1198, Sept. 2005.

[10] A. Katsevich, "Theoretically Exact Filtered Backprojection-Type Inversion Algorithm for Spiral CT," *SIAM Journal on Applied Mathematics*, vol. 62, pp. 2012–2026, Jan. 2002.

[11] A. Katsevich, "Analysis of an exact inversion algorithm for spiral cone-beam CT," *Physics in Medicine and Biology*, vol. 47, pp. 2583–2597, Aug. 2002.

[12] Y. Zou and X. Pan, "Exact image reconstruction on PI-lines from minimum data in helical cone-beam CT," *Physics in Medicine and Biology*, vol. 49, pp. 941–959, Mar. 2004.

[13] H. Yu, Y. Ye, S. Zhao, and G. Wang, "A backprojection-filtration algorithm for nonstandard spiral cone-beam CT with an n -PI-window," *Physics in Medicine and Biology*, vol. 50, pp. 2099–2111, May 2005.

[14] Y. Zou, X. Pan, and E. Y. Sidky, "Theory and algorithms for image reconstruction on chords and within regions of interest," *Journal of the Optical Society of America A*, vol. 22, no. 11, p. 2372, 2005.

[15] S. Cho, D. Xia, C. A. Pelizzari, and X. Pan, "Exact reconstruction of volumetric images in reverse helical cone-beam CT," *Medical Physics*, vol. 35, pp. 3030–3040, July 2008.

[16] M. Beister, D. Kolditz, and W. A. Kalender, "Iterative reconstruction methods in X-ray CT.," *Physica medica : PM : an international journal devoted to the applications of physics to medicine and biology : official journal of the Italian Association of Biomedical Physics (AIFB)*, vol. 28, pp. 94–108, Apr. 2012.

[17] P. Gilbert, "Iterative methods for the three-dimensional reconstruction of an object from projections," *Journal of Theoretical Biology*, vol. 36, pp. 105–117, July 1972.

[18] A. H. Andersen and A. C. Kak, "Simultaneous Algebraic Reconstruction Technique (SART): A Superior Implementation of the Art Algorithm," *Ultrasonic Imaging*, vol. 6, pp. 81–94, Jan. 1984.

[19] L. A. Shepp and Y. Vardi, "Maximum likelihood reconstruction for emission tomography.," *IEEE transactions on medical imaging*, vol. 1, pp. 113–22, Jan. 1982.

[20] G. Kontaxakis and L. G. Strauss, "Maximum likelihood algorithms for image reconstruction in positron emission tomography," *Mediterra, Athens.*, pp. 73–106, 1998.

[21] K. Lange and R. Carson, "EM reconstruction algorithms for emission and transmission tomography.," *Journal of computer assisted tomography*, vol. 8, pp. 306–16, Apr. 1984.

[22] C. A. Bouman and K. Sauer, "A unified approach to statistical tomography using coordinate descent optimization.," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 5, pp. 480–92, Jan. 1996.

[23] F. Xu, W. Xu, M. Jones, B. Keszthelyi, J. Sedat, D. Agard, and K. Mueller, "On the efficiency of iterative ordered subset reconstruction algorithms for acceleration on GPUs.," *Computer methods and programs in biomedicine*, vol. 98, pp. 261–70, June 2010.

[24] S. H. Manglos, G. M. Gagne, A. Krol, F. D. Thomas, and R. Narayanaswamy, "Transmission maximum-likelihood reconstruction with ordered subsets for cone beam CT," *Physics in Medicine and Biology*, vol. 40, pp. 1225–1241, July 1995.

[25] S.-J. Lee, "Accelerated coordinate descent methods for Bayesian reconstruction using ordered subsets of projection data," vol. 4121, pp. 170–181, Oct. 2000.

[26] J. Hsieh, B. Nett, Z. Yu, K. Sauer, J.-B. Thibault, and C. a. Bouman, "Recent Advances in CT Image Reconstruction," *Current Radiology Reports*, vol. 1, pp. 39–51, Jan. 2013.

[27] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Johns Hopkins University Press, 2013.

[28] D. S. Watkins, *Fundamentals of Matrix Computations*. John Wiley and Sons. Inc., 2002.

[29] T. A. Davis, *Direct methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2006.

[30] A. k. Björck, *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, Jan. 1996.

[31] J. W. Demmel, *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Jan. 1997.

[32] N. J. Higham, "A Survey of Condition Number Estimation for Triangular Matrices," *SIAM Review*, vol. 29, pp. 575–596, Dec. 1987.

[33] J. T. Bushberg, J. A. Seibert, E. M. Leidholdt, and J. M. Boone, *The Essential Physics of Medical Imaging*. Lippincott Williams & Wilkins, Dec. 2011.

[34] W. Yao and K. Leszczynski, "Analytically derived weighting factors for transmission tomography cone beam projections.," *Physics in medicine and biology*, vol. 54, pp. 513–33, Feb. 2009.

[35] R. A. Brooks and G. D. Chiro, "Beam hardening in X-ray reconstructive tomography," *Physics in Medicine and Biology*, vol. 21, pp. 390–398, May 1976.

[36] R. J. Jennings, "A method for comparing beam-hardening filter materials for diagnostic radiology," *Medical Physics*, vol. 15, p. 588, July 1988.

[37] R. C. Chase, "An improved image algorithm for CT scanners," *Medical Physics*, vol. 5, p. 497, Nov. 1978.

[38] G. T. Herman, "Correction for beam hardening in computed tomography," *Physics in Medicine and Biology*, vol. 24, pp. 81–106, Jan. 1979.

[39] J. Hsieh, R. C. Molthen, C. A. Dawson, and R. H. Johnson, "An iterative approach to the beam hardening correction in cone beam CT," *Medical Physics*, vol. 27, p. 23, Jan. 2000.

[40] M.-J. Rodríguez-Álvarez, F. Sánchez, A. Soriano, A. Iborra, and C. Mora, "Exploiting symmetries for weight matrix design in CT imaging," *Mathematical and Computer Modelling*, vol. 54, pp. 1655–1664, Oct. 2011.

[41] M. B. Stephenson and H. N. Christiansen, "A polyhedron clipping and capping algorithm and a display system for three dimensional finite element models," *ACM SIGGRAPH Computer Graphics*, vol. 9, pp. 1–16, Sept. 1975.

[42] F. Sánchez, A. Orero, A. Soriano, C. Correcher, P. Conde, A. González, L. Hernández, L. Moliner, M. J. Rodríguez-Alvarez, L. F. Vidal, J. M. Benlloch, S. E. Chapman, and W. M. Leevy, "ALBIRA: A small animal PET/SPECT/CT imaging system," *Medical physics*, vol. 40, p. 051906, May 2013.

[43] A. Iborra, M. J. Rodriguez-Alvarez, A. Soriano, F. Sanchez, P. Bellido, P. Conde, E. Crespo, A. J. Gonzalez, L. Moliner, J. P. Rigla, M. Seimetz,

L. F. Vidal, and J. M. Benlloch, "Noise Analysis in Computed Tomography (CT) Image Reconstruction using QR-Decomposition Algorithm," *IEEE Transactions on Nuclear Science*, vol. 62, pp. 869–875, June 2015.

[44] C. Mora, M. J. Rodríguez-Álvarez, and J. V. Romero, "New pixellation scheme for CT algebraic reconstruction to exploit matrix symmetries," *Computers and Mathematics with Applications*, vol. 56, no. 3, pp. 715–726, 2008. Mathematical Models in Life Sciences and Engineering.

[45] I. Duff, A. Erisman, and J. Reid, *Direct Methods for Sparse Matrices*. Clarendon Press, 1986.

[46] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[47] C. Shaffer, *Data Structures & Algorithm Analysis in C++*. Dover Publications, 2011.

[48] Q. F. Stout and B. L. Warren, "Tree rebalancing in optimal time and space," *Communications of the ACM*, vol. 29, pp. 902–908, Sept. 1986.

[49] A. Drozdek, *Data Structures and Algorithms in C++*. Cengage Learning, Sept. 2004.

[50] A. S. Householder, "Unitary Triangularization of a Nonsymmetric Matrix," *Journal of the ACM*, vol. 5, pp. 339–342, Oct. 1958.

[51] W. Givens, "Computation of Plain Unitary Rotations Transforming a General Matrix to Triangular Form," *Journal of the Society for Industrial and Applied Mathematics*, vol. 6, pp. 26–50, Mar. 1958.

[52] J. H. Wilkinson, *The algebraic eigenvalue problem*. Clarendon Press, 1965.

[53] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Trans. Math. Softw.*, vol. 5, pp. 308–323, Sept. 1979.

[54] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, third ed., 1999.

[55] E. Anderson, "Discontinuous plane rotations and the symmetric eigenvalue problem," 2000.

[56] G. Stewart, "The economical storage of plane rotations," *Numerische Mathematik*, vol. 25, no. 2, pp. 137–138, 1976.

[57] M. Yannakakis, "Computing the Minimum Fill-In is NP-Complete," *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 1, pp. 77–79, 1981.

[58] A. George, J. Liu, and E. Ng, "Row-ordering schemes for sparse givens transformations. I. bipartite graph model," *Linear Algebra and its Applications*, vol. 61, pp. 55–81, 1984.

[59] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference on* -, (New York, New York, USA), pp. 157–172, ACM Press, 1969.

[60] J. A. George, *Computer Implementation of the Finite Element Method*. PhD thesis, 1971. Department of Computer Science, Stanford University, Stanford, California.

[61] I. P. King, "An automatic reordering scheme for simultaneous equations derived from network systems," *International Journal for Numerical Methods in Engineering*, vol. 2, no. 4, pp. 523–533, 1970.

[62] J. W. H. Liu, "Modification of the minimum-degree algorithm by multiple elimination," *ACM Transactions on Mathematical Software*, vol. 11, no. 2, pp. 141–153, 1985.

[63] Wolfram Research, Inc., "Mathematica 9," 2013. http://reference.wolfram.com.

[64] C. Mueller, "Sparse matrix reordering algorithms for cluster identification," 2004. For I532, Machine Learning in Bioinformatics.

[65] A. Lim and B. Rodrigues, "A centroid-based approach to solve the bandwidth minimization problem," in *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*, p. 6 pp., IEEE, Jan. 2004.

[66] Online Reference Manual for Intel Math Kernel Library 11.2, "Intel MKL PARDISO – Parallel Direct Sparse Solver Interface." https://software.intel.com/en-us/node/521677 – Last visited in September 2014.

[67] E. Busemann Sokole, A. Płachcínska, and A. Britten, "Acceptance testing for nuclear medicine instrumentation.," *European journal of nuclear medicine and molecular imaging*, vol. 37, pp. 672–81, Mar. 2010.

[68] J. H. Hubbell and S. M. Seltzer, "Tables of X-Ray Mass Attenuation Coefficients and Mass Energy-Absorption Coefficients from 1 keV to 20 MeV for Elements Z = 1 to 92 and 48 Additional Substances of Dosimetric Interest," 2004.

[69] R. Nowotny, "XMuDat: Photon attenuation data on PC," 1998.

[70] J.-Y. Jin, L. Ren, Q. Liu, J. Kim, N. Wen, H. Guan, B. Movsas, and I. J. Chetty, "Combining scatter reduction and correction to improve image quality in cone-beam computed tomography (CBCT)," *Medical Physics*, vol. 37, p. 5634, Nov. 2010.

[71] S. J. Erasmus and K. C. A. Smith, "An automatic focusing and astigmatism correction system for the SEM and CTEM," *Journal of Microscopy*, vol. 127, pp. 185–199, Aug. 1982.

[72] N. Zhang, A. Vladar, M. Postek, and B. Larrabee, "A kurtosis-based statistitcal measure for two-dimensional processes and its application to image sharpness," *Proc. section of physical and engineering sciences of American Statistical Society*, pp. 4730–4736, 2003.

[73] J. Caviedes and F. Oberti, "A new sharpness metric based on local kurtosis, edge and energy information," *Signal Processing: Image Communication*, vol. 19, no. 2, pp. 147–161, 2004.

[74] R. Ferzli and L. J. Karam, "A no-reference objective image sharpness metric based on the notion of just noticeable blur (JNB).," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 18, pp. 717–28, Apr. 2009.

[75] P. Marziliano, F. Dufaux, S. Winkler, and T. Ebrahimi, "A no-reference perceptual blur metric," in *Proceedings. International Conference on Image Processing*, vol. 1, pp. III–57–III–60, IEEE, 2002.

[76] I. Sobel and G. Feldman, "A 3x3 isotropic gradient operator for image processing," 1968. Unpublished, presented as a talk within the Stanford Artificial Intelligence Project.

[77] K. L. Boedeker, V. N. Cooper, and M. F. McNitt-Gray, "Application of the noise power spectrum in modern diagnostic MDCT: part I. Measurement of noise power spectra and noise equivalent quanta.," *Physics in medicine and biology*, vol. 52, pp. 4027–46, July 2007.

[78] M. F. Kijewski and P. F. Judy, "The noise power spectrum of CT images.," *Physics in medicine and biology*, vol. 32, pp. 565–75, May 1987.

[79] J. Baek and N. J. Pelc, "The noise power spectrum in CT with direct fan beam reconstruction," *Medical Physics*, vol. 37, no. 5, p. 2074, 2010.

[80] D. J. Tward and J. H. Siewerdsen, "Noise aliasing and the 3D NEQ of flat-panel cone-beam CT: Effect of 2D/3D apertures and sampling," *Medical Physics*, vol. 36, no. 8, p. 3830, 2009.

[81] J. H. Siewerdsen, I. a. Cunningham, and D. a. Jaffray, "A framework for noise-power spectrum analysis of multidimensional images," *Medical Physics*, vol. 29, no. 11, p. 2655, 2002.

[82] S. N. Friedman, G. S. K. Fung, J. H. Siewerdsen, and B. M. W. Tsui, "A simple approach to measure computed tomography (CT) modulation transfer function (MTF) and noise-power spectrum (NPS) using the American College of Radiology (ACR) accreditation phantom.," *Medical physics*, vol. 40, p. 051907, May 2013.

[83] D. A. Jaffray and J. H. Siewerdsen, "Cone-beam computed tomography with a flat-panel imager: Initial performance characterization," *Medical Physics*, vol. 27, no. 6, p. 1311, 2000.

[84] M. B. Williams, P. A. Mangiafico, and P. U. Simoni, "Noise power spectra of images from digital mammography detectors," *Medical Physics*, vol. 26, p. 1279, July 1999.

Acronyms

ART algebraic reconstruction technique. 7, 8, 19

BPF backprojection-filtration. 8

BST binary search tree. 44–54, 59, 64, 95, 99

CCW counterclockwise. 25, 26, 31–34, 149

CNR contrast to noise ratio. 114–120, 143

COO coordinate list. 41–43

CSC compressed sparse column. 42, 43, 54

CSR compressed sparse row. 42, 43, 63, 64

CT computed tomography. 6–12, 19, 20, 23, 24, 31, 34–36, 38, 55, 56, 67, 71, 98, 99, 101–103, 114, 147, 149–151

CV coefficient of variation. 114–118, 120, 143

FBP filtered backprojection. 8, 10–12, 102, 103, 105, 107–109, 115, 119, 121–123, 125–133, 135–138, 140, 142, 143, 145, 148, 150

FoV field of view. 101, 139

FWHM full width at half maximum. 121, 124, 125, 130–133

HU Hounsfield units. 138

ICD iterative coordinate descent. 8, 9

LoR line of response. 23–31, 33–35, 39, 149

MBIR model-based iterative reconstruction. 9

MLEM maximum likelihood expectation maximization. 8–12, 102, 103, 106–109, 114, 115, 119, 121–123, 125–130, 133, 135–138, 140, 142, 143, 146, 148, 150

MRI magnetic resonance imaging. 3, 4

NPS noise power spectrum. 138–140, 142–146

OS ordered subsets. 9

PE polyethylene. 103–107, 109, 112, 114–116, 123, 125, 127, 129, 131

PET positron emission tomography. 6, 101

PMMA polymethyl methacrylate. 102–107, 114, 119

POM polyoxymethylene. 103, 107, 109, 113, 115, 117, 123, 125, 127, 129, 131

PTFE polytetrafluoroethylene. 103

SART simultaneous ART. 8

SIRT simultaneous iterative reconstruction technique. 8, 9

VoI volume of interest. 102, 103, 107, 109, 114–120

Acknowledgments / Agradecimientos

Me gustaría expresar mi gratitud a mis directores. En especial a la Dra. María José Rodríguez Álvarez, que con su experiencia y conocimientos me ha proporcionado una inestimable ayuda durante toda esta investigación; y que con su infinita paciencia me ha guiado durante este largo camino, de la forma más amable y diligente. No hay espacio aquí para transmitir mi aprecio, pero le debo mi eterna gratitud.

He tenido la suerte de coincidir y colaborar con excelentes compañeros a lo largo de esta investigación, tanto en el Instituto de Instrumentación para Imagen Molecular como en el Instituto de Matemática Multidisciplinar. A todos ellos gracias. Especialmente al director del Instituto de Instrumentación para Imagen Molecular, Profesor de Investigación José María Benlloch Baviera, que siempre me ha mostrado su apoyo, ayuda y optimismo.

De entre los compañeros con los que he tenido la suerte de coincidir, quiero agradecer especialmente a Pablo Bellido Millán que me haya dedicado largas charlas en las que tuvo la paciencia y amabilidad de exponer su profundo conocimiento de la física frente a mis triviales dudas de forma que las pudiese entender, revelándome claves sin las cuales no hubiese podido llevar a buen término esta investigación y permitiéndome participar de su pragmatismo y buen hacer. Mi más sincero agradecimiento. A Pablo Conde Castellanos, me gustaría agradecerle el tiempo que me ha dedicado, en el que con su vasto conocimiento de la fisica y la electrónica, ha tenido la paciencia de arrojar luz sobre muchos de los numerosos vacíos que he ido llenando durante este camino, y conduciéndome amablemente hacia el camino de la eficiencia, rapidez y buen hacer. Mi más profundo agradecimiento. Y a Sebastián Sanchez Goez, quiero agradecerle que haya compartido su amplio conocimiento de la física conmigo, siempre tomando de forma seria y metódica mis dudas, por simples que fuesen y compartiendo razonamientos que han completado mi experiencia durante estos últimos años. Mi más sentido agradecimiento.

Quiero agradecer a mi familia, que siempre haya estado a mi lado. Particularmente, a mi compañera y mejor amiga, Blanca, su incondicional apoyo, su ánimo y su cariño; en las largas noches de trabajo, en el día a día y en todo momento. No sólo su aura de bondad e inteligencia me han hecho ser mejor persona, sino que sin su compañía, respirar sería simplemente el marcar de un reloj.