

Three-Dimensional Cone-Beam Computed Tomography ActiveX Style

Stefano Agostinelli

Three-Dimensional Cone-Beam Computed Tomography ActiveX Style

Stefano Agostinelli

Corso di Dottorato in Fisica XIII Ciclo
Università degli Studi di Genova



Genova, December 2000

Supervisors:

Prof. Dr. M. Bertero

Prof. Dr. A. K. Louis

Dr. P. Boccacci

Dr. F. Foppiano

“THE BLUE PAIN FADES TO A POINT WHERE
IT DOESN’T FADE IT STAYED. . . BLUE
STIRRED HIS RED COAT HEART TO
THIS STRANGE ENGINE”

MARILLION - This Strange Engine

To my strange engine: Stefania

Abstract

Computed Tomography (CT) is nowadays a mature medical imaging technique which has met a huge success since its introduction in the 70s by Nobel prize Hounsfield. While the principles underlying tomography are exactly the same described by Hounsfield in his patent, during the years CT scanners have been constantly evolving by adopting faster volume scanning schemes, improved beam shaping and more efficient X-ray detectors. The end of this quest, a sort of holy grail of computed tomography, is three-dimensional Cone-Beam CT which promises the real-time volumetric reconstructions needed to fulfil the increasing demand of accurate on-line diagnostic examinations.

The tentative state of cone-beam computed tomography requires a new generation of advanced software tools to lead the research and development of proprietary reconstruction algorithms. This issue is addressed in this thesis by *Strange Engine*, our object-oriented visual toolkit dedicated to the housing, evaluation and simulation of cone-beam CT reconstruction tasks. By offering everything necessary for the testing of user-written reconstruction code within a modern, user-friendly environment exploiting state-of-the-art ActiveX software technologies, Strange Engine allows algorithm developers to focus on their real job, leaving all the other duties to the toolkit. The discussion includes an introduction to cone-beam tomography, the detailed specification of our software toolkit and the analysis of four different reconstruction algorithms for both simulated and real-world data.

Key Words: 3D, Computed Tomography, Cone-Beam, Object-Oriented

Acknowledgments

In this space I would like to thank all those who contributed to this thesis and have supported my work during the last three years.

First of all I would like to thank Prof. Dr. Mario Bertero and Dr. Patrizia Boccacci for giving to me the opportunity to be involved in the medical imaging field and for paying so much attention to my education and research interests. A big thank you goes to Patrizia for all the help offered in many occasions.

I am really much grateful to Prof. Dr. Alfred K. Louis because he let me participate in his research on cone-beam reconstruction based on approximate inverse. Dear Prof. Louis thank you very much for the great support I have received in Saarbrücken, I have been really well during my stays in Germany! I would like also to express gratitude to everyone in the staff (Rainer, Roman, Thomas, the Peters and Eva) for their kindness and hospitality.

Another thank you goes to Prof. Dr. Michel Defrise. During my short stay in Bruxelles he offered me very inspiring insight of the cone-beam reconstruction arena including radiotherapy applications and related bibliography.

Dr. Franca Foppiano deserves a special mention because she has been really a friend for many years now and I am much indebted with her. She let me in medical physics and have taught me so many things... Thanks a lot Franca!

A thank you very much also to Andrea for all the time he spent for the beautiful artwork appearing in Strange Engine's object icons.

Some other people I would like to thank (in sparse order) are: Stefania, Gabriella, Prof. Dr. Pasquale Ottonello, Prof. Dr. Spanò.

Notation

Symbols

bolds (e.g. \mathbf{r})	denote vectors
$(\mathbf{r} \cdot \mathbf{r}')$	denotes the dot product $(xx' + yy' + zz')$
$\int f(x) dx$	is the integral $\int_{-\infty}^{+\infty} f(x) dx$
$\text{sinc}(x)$	$= \sin(x)/x$
the hat $\hat{}$	denotes Fourier transformed functions
the tilde $\tilde{}$	denotes filtered functions

Operators and Functions

\mathcal{R}	denotes the Radon transform
\mathcal{F}	denotes the Fourier transform
δ	is the Dirac delta
μ	is a map of the attenuation coefficient
d_p	are parallel-beam projections
d_f	are fan-beam projections
d_c	are cone-beam projections
d_{op}	are oblique-parallel beam projections
k	is a reconstruction filter

Abbreviations (Physics)

1D/2D/3D	One/Two/Three-Dimensional
ART	Algebraic Reconstruction Technique
BGO	Bismuth Germanate
CBCT	Cone-Beam Computed Tomography
CCD	Charge Coupled Device
CT	Computed Tomography
DFM	Direct Fourier Methods
EPID	Electronic Portal Imaging Device
EM	Expectation Maximization
FBP	Filtered Back Projection
FDK	Feldkamp–Davis–Kress
FFBP	Fast Filtered Back Projection
FFT	Fourier Fast Transform
FPI	Flat-Panel Imager
FST	Fourier Slice Theorem
MCT	Micro Computed Tomography
NDT	Non Destructive Testing
RT	Radiation Therapy
SCT	Spiral Computed Tomography
SNR	Signal to Noise Ratio
SVD	Singular Value Decomposition
TRML	TRansmission Maximum Likelihood

Abbreviations (Computing)

API	Application Programming Interface
ATL	Active Template Library
BMP	Window Bitmap
COM	Component Object Model
DCF	Device Configuration File
DICOM	Digital Imaging and Communications in Medicine
FFTW	Fastest Fourier in The West

Abbreviations (Computing) . . . continued

GDI	Graphics Device Independent
GUI	Graphical User Interface
HDF	Hierarchical Data Format
IDL	Interface Definition Language
LUT	Look Up Table
MAC	Multiply And aCcumulate
MDI	Multi Document Interface
MFC	Microsoft Foundation Classes
MKL	Math Kernel Library
MIL	Matrox Imaging Library
OO	Object-Oriented
OpenGL	Open Graphics Library
OLE	Object Linking and Embedding

Contents

1	Introduction	1
1.1	Cone-beam CT revealed	1
1.1.1	Scanning techniques	3
1.1.2	Reconstruction algorithms	5
1.1.3	Some considerations on scatter	7
1.2	Objectives and outline of this thesis	8
1.3	Author references	9
2	2D Reconstruction	11
2.1	Projections and the Radon transform	11
2.1.1	Projections measurement	11
2.1.2	The 2D Radon transform	12
2.1.3	The Fourier Slice Theorem	14
2.1.4	The fan-beam projection geometry	14
2.2	2D reconstruction algorithms overview	16
2.3	Direct Fourier Methods	17
2.4	Filtered Back Projection	18
2.4.1	FBP parallel-beam	18
2.4.2	FBP fan-beam	19
2.4.3	Filtering	20
2.4.4	Aliasing	21
2.5	Rebinned FBP	22
2.5.1	Besson's fan-parallel formula	23
2.6	Fast Filtered Back Projection	24

2.7	Approximate Inverse	24
2.7.1	Definition	24
2.7.2	Application to 2D computer tomography	25
2.7.3	Filter design	26
2.7.4	Reconstruction formula for the fan-beam geometry	27
2.8	Iterative Methods	28
3	3D Cone-Beam Reconstruction	29
3.1	Cone-beam projections and the 3D Radon transform	29
3.1.1	The cone-beam geometry	29
3.1.2	The 3D Radon Transform	31
3.1.3	The 3D Fourier Slice Theorem	31
3.1.4	The Central Slice Theorem	32
3.1.5	Plane integrals from cone-beam data	33
3.2	3D Radon transform inversion	33
3.2.1	Completeness conditions	34
3.3	3D reconstruction algorithms overview	36
3.4	Grangeat's method	36
3.4.1	Grangeat's fundamental relation	37
3.4.2	Radon inversion through the first derivative of the 3D Radon transform	39
3.4.3	Computation of the first derivate of the 3D Radon trans- form	39
3.4.4	Interpolation.	40
3.4.5	Radon inversion with the two-stage approach.	40
3.4.6	Radon inversion and 3D Direct Fourier Methods	41
3.5	3D Filtered Back Projection	42
3.5.1	The Feldkamp-Davis-Kress method	42
3.6	Rebinned 3D FBP	46
3.6.1	Oblique-parallel rebinned 3D FDK	46
3.6.2	Besson's fan-parallel extension to 3D cone-beam	46
3.7	3D Fast Back Projection	48
3.8	3D Approximate Inverse	48
3.8.1	Reconstruction formula for the cone-beam geometry	48
3.9	3D Iterative Methods	50
4	The Strange Engine toolkit	51
4.1	Toolkit design	52
4.1.1	A bit of history	53
4.1.2	Strange Engine constraints	53

4.1.3	The object-oriented point of view	54
4.2	Toolkit user interface	58
4.2.1	Object persistency	58
4.2.2	User interaction	58
4.2.3	Object-to-object interaction	59
4.3	The ActiveX edge	60
4.3.1	Strange Engine's COM interfaces.	61
4.3.2	Plug-in request and plug-in user interface	63
4.4	Toolkit usage	64
4.4.1	Data display	65
4.4.2	Acquisition and reconstruction	65
4.4.3	Algorithm test	67
4.4.4	Performance	67
4.5	Toolkit internals	71
4.5.1	The main project	71
4.5.2	Custom link libraries	72
4.5.3	Other link libraries	72
4.5.4	ActiveX plug-ins	73
4.6	The build 12 distribution	73
5	ActiveX plug-ins implementation	75
5.1	Introduction	75
5.2	Evaluation of different Fast Fourier Transform codes	76
5.3	The FDK plug-in	78
5.3.1	General remarks	78
5.3.2	Computational cost	79
5.3.3	Interface implementation	79
5.3.4	Reconstruction implementation	80
5.3.5	User interface	80
5.4	The FDK+ plug-in	81
5.4.1	Interface implementation	81
5.4.2	Reconstruction implementation	81
5.4.3	Filtering and zero padding	84
5.4.4	User interface	85
5.5	The TFDK plug-in	85
5.5.1	Interface implementation	87
5.5.2	Reconstruction implementation	87
5.5.3	User interface	88
5.6	The AInv plug-in	88
5.6.1	Interface implementation	89

5.6.2	Reconstruction implementation	89
5.6.3	User interface	91
5.7	Reconstruction results	92
5.7.1	Simulated projections	92
5.7.2	RT simulator projections	96
5.7.3	Performance comparison	98
5.7.4	Pre-processing performance	99
6	Conclusions	101
	References	103

Introduction

Computed Tomography (CT) is nowadays a mature medical imaging technique which has met a huge success since its introduction in the 70s by Nobel prize Hounsfield. While the principles underlying tomography are exactly the same described by Hounsfield in his patent, during the years CT scanners have been constantly evolving by adopting faster volume scanning schemes, improved beam shaping and more efficient X-ray detectors. The end of this quest, a sort of holy grail of computed tomography, is three-dimensional (3D) Cone-Beam CT (CBCT) which promises the real-time volumetric reconstructions needed to fulfil the increasing demand of accurate on-line diagnostic examinations.

In this thesis we describe our Object-Oriented (OO) visual toolkit, named *Strange Engine*, dedicated to evaluation, simulation and ActiveX housing of 3D CBCT reconstruction algorithms. ActiveX implementations of four CBCT reconstruction algorithms are presented and analysed by comparison of restored images and speed benchmarks for both simulated and real-world data.

1.1 Cone-beam CT revealed

Computed tomography is a digital technique which allows the reconstruction of an object using a series of projections taken from different angles around it. As pointed out by Webb in [85], tomography is really like guessing an object's shape from the shadows it casts when it is lit from some light source.

Unlike conventional tomography, which makes use of one-dimensional (1D) X-ray projections to reconstruct a slice of the exposed object, cone-beam CT provides a fully volumetric reconstruction by processing two-dimensional (2D)

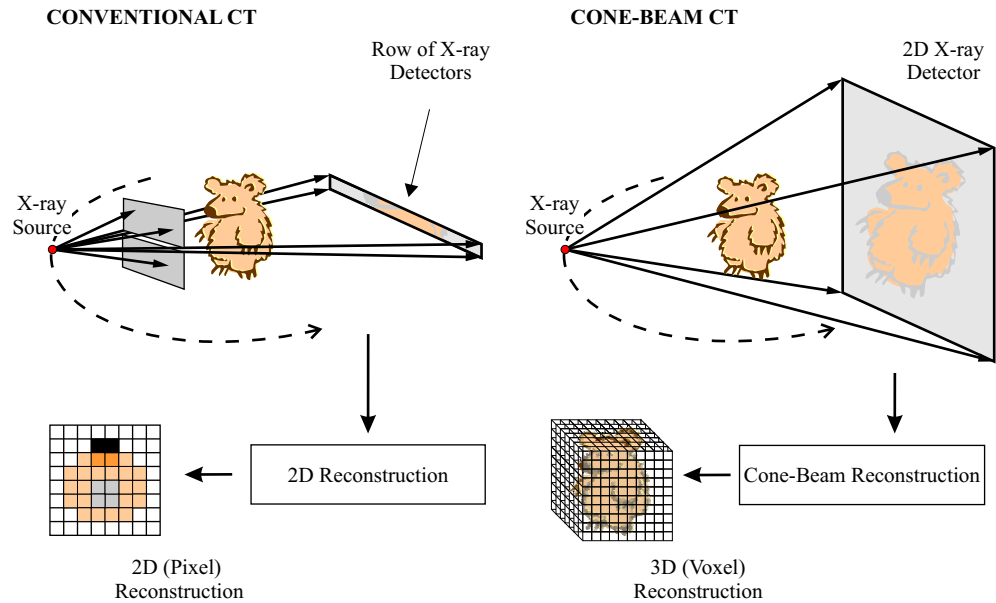


Figure 1.1. Cone-beam computed tomography makes use of X-rays efficiently by exploiting the full 3D exposure. In order to reduce the problem to planar geometry, conventional CT introduces considerable collimation to form X-ray fan-beams.

cone-beam projections. These projections can for instance be transmission type radiographs generated by a X-ray point source as shown in Figure 1.1.

We can list a number of advantages of cone-beam CT versus conventional CT:

- more efficient use of radiation, which means less X-ray source stress and complete and more productive exposure usage;
- much shorter examination times because, in a single rotational scan, it is possible to obtain tens of slices not just one;
- less motion artefacts and better axial resolution;

as well as a number of disadvantages:

- more complex scanning set-up;
- reconstruction algorithms more difficult to write and implement;
- somewhat greater radiation dose given to the patient per examination and lower Signal to Noise Ratio (SNR), due to increased scatter components.

Let us briefly have a quick look at each of these points to have an overall idea about cone-beam CT features and capabilities.

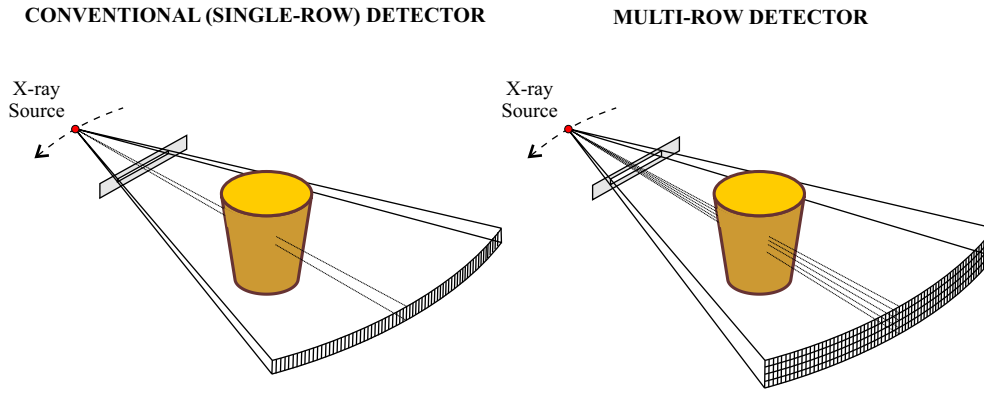


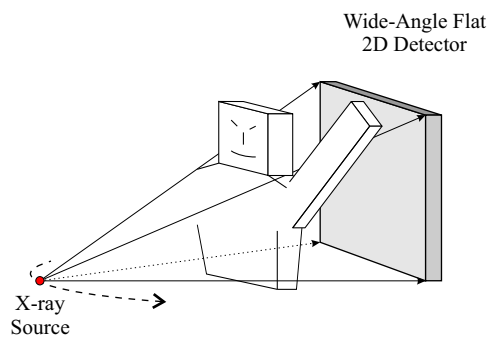
Figure 1.2. The new generation CT multi-row detectors, for each source position, are able to record up to five (four in this figure) tilted X-ray fans. This allows speed-up of data acquisition, better axial resolution and reduction of motion artefacts.

1.1.1 Scanning techniques

The first CT scanner was fabricated and tested in 1972 by Hounsfield [33]. An up-to-date summary of CT history is given by Carlsson in [15]. In modern computed tomography scanners, X-ray radiation is shaped by collimators in fan-beams encompassing the whole transversal section of the object being examined. In Spiral CT (SCT) scanners the patient is slowly translated in the axial direction while the X-ray source gantry is rotated around it; this allows the acquisition of many transmission profiles in a continuous manner, shortening examination times by a factor 10 in comparison to previous step-and-shoot scanners. Reconstruction, which is based on two-dimensional analysis, requires some sort of axial interpolation and a compromise between scanning speed, axial resolution, noise and motion artefacts has to be found.

To improve axial resolution, alleviate motion artefacts and cut even further ($3\times$ speed-up) examination times while scoring high SNR, manufacturers have been introducing in the recent years spiral CT scanners taking advantage of multi-row detectors. 3-rows, 4-rows and 5-rows scanners are currently on the market; see Wang [80] and [54, 35] for some theoretical and practical details. In Figure 1.2 a conventional detector is compared to a new generation 4-rows detector. The detector rows record at the same time slightly tilted transmission profiles and this allows better axial resolution and/or larger spiral pitch (higher scanning speed). While the reconstruction problem is in theory 3D, the industry seems oriented to use modified 2D algorithms to take into account the small divergence (angular aperture of about 1°) of X-ray fans.

CIRCULAR PATH



HELICAL PATH

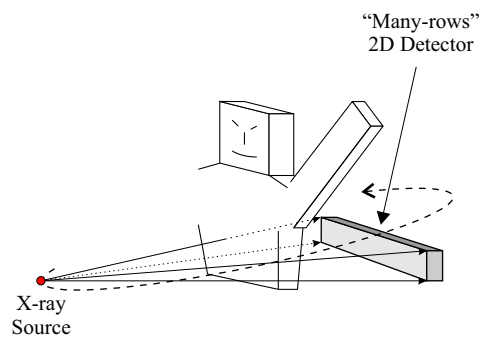


Figure 1.3. In cone-beam computed tomography two-dimensional X-ray detectors are used to collect transmission data. In the “circular path” set-up a single 2π scan is sufficient to acquire a set of cone-beam projections. A wide-angle flat detector is used. The “helical path” set-up combines cone-beam and spiral scanning. Detector can be flat or curved as in conventional SCT.

In conventional and multi-row SCT scanners X-rays are detected by high pressure Xenon gas chambers or solid state devices (photodiodes) which are optically coupled to phosphors or scintillators such as bismuth germanate (BGO), caesium iodide (CsI) and most commonly cadmium tungstate (CdWO_4). The detector elements (channels) are positioned along a curved circular arc lying on the X-ray source path.

Cone-beam computed tomography is strictly three-dimensional because the examined locus is scanned by the beam in its fully 3D extension. The divergence of the cone-beam is used to collect spatial information on a two-dimensional detector. Usage of a two-dimensional detector allows to retrieve concurrently much more object information in comparison to conventional computed tomography. We can imagine for example to expose a full human thorax as depicted in Figure 1.3 (left) to reconstruct the entire thorax morphology with a single gantry scan. The X-ray source moves along a circular path and the detector is wide-angle and flat. In the alternative CBCT set-up shown in Figure 1.3 (right), the X-ray source moves along a helical (spiral) path. A curved or flat “many-rows”¹ detector is adequate to gather all the data, but the problem, given the large cone aperture, is still 3D and the reconstruction cannot be brought back to 2D as done in conventional SCT.

¹We speak of “many-rows” detectors when the number of rows is far less than the number of channels. In the other cases, i.e. when the number of rows is just a few or it is comparable to the number of channels, we speak respectively of “multi-row” and wide-angle detectors.

Cone-beam computed tomography needs two-dimensional detectors. In general 2D detectors provide fast and efficient 3D scanning, but we should be aware of a couple of practical aspects:

- a) 2D detectors may be heavy and bulky, mechanical problems can arise and gantry rotational speed may be severely limited;
- b) data throughput may be high, so computational power should be enough to handle it;
- c) 2D detectors technology is expensive and rapidly progressing: what is high-priced and new today may be cheap and obsolete tomorrow.

That said, let us have a look to the detector arrangements which have been proposed for cone-beam CT. Systems based on phosphors or scintillators plates optically coupled to Charge Coupled Device (CCD) or other types of video cameras are very common for industrial Non Destructive Testing (NDT) and Micro CT (MCT) [92, 42]. An Image Intensifier (II) is in some case added to amplify the light signal. Experimental usage of camera based CBCT systems for medical applications have also been discussed by Webb in [86] and reported in [16, 5]. Recent technology developments have led to the introduction of Flat-Panel Imagers (FPI) based on phosphors or scintillators directly coupled to amorphous silicon (a-Si:H) photodiodes array. This kind of detectors promises to be both large-area, up to $40 \times 40 \text{ cm}^2$, and with very good spatial resolution, up to 3000×3000 pixels. Applications of FPI based cone-beam computed tomography in the medical field include real-time X-ray diagnostic examinations such as digital angiography and interventional procedures [87]. Usage of cone-beam CT is also under investigation in Radiation Therapy (RT) to address treatment registration and treatment verification [38, 39]. This also includes mega-voltage cone-beam CT [74, 75, 55, 67, 31] in which high energy (4-10 MeV) X-ray beams coming from RT linear accelerators are used to produce cone-beam radiographs on Electronic Portal Imaging Devices (EPID). While these devices are at present based on cameras or ionisation chambers, flat-panel imagers are probably going to be the next generation EPIDs as well.

1.1.2 Reconstruction algorithms

Cone-beam computed tomography requires three-dimensional reconstruction algorithms. Theoretical derivation and actual implementation of CBCT algorithms are not easy. In fact, as lucidly stated by Defrise and Clack in [20], tomographic reconstruction from cone-beam data is a fascinating problem for which a fully satisfactory solution has not yet been given.

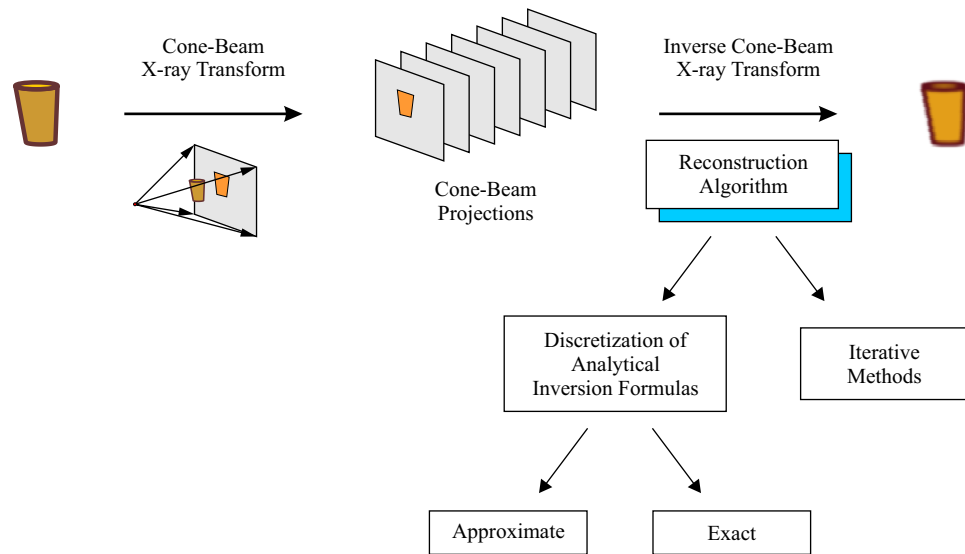


Figure 1.4. High-level overview on CBCT reconstruction algorithms.

Complications in cone-beam tomographic reconstruction algorithms mainly arise from the following aspects:

- the scanning geometry, the X-ray beam divergence and the reconstructed object representation are 3D, this means that vector math has to be used;
- the available projection data may be incomplete, because the scan technique may not be adequate to describe fully the examined object;
- the computational load is in general very high, approximations and/or fast reconstruction schemes are needed.

An analysis from an algorithmic point of view of CT imaging techniques is given by Louis in [46]. Other remarkable summaries are given among others by Kak and Slaney in [40], by Wang and Vannier in [83], by Schaller in [68] and by Turbell in [78]. Figure 1.4 shows a high-level overview on 3D CBCT reconstruction algorithms. Basically we can individuate two distinct families of reconstruction algorithms: a) discretization of analytical inversion formulas and b) iterative methods. By a mathematical point of view, CBCT reconstruction is a particular case of the inversion of the three-dimensional cone-beam X-ray transform. Algorithms based on discretization of analytical inversion formulas attack the inversion problem directly by computing a discretized inversion formula which can be either exact or approximate. These algorithms

suffer basically two problems: 1) an inversion formula must exist, i.e. the problem should admit a unique and stable solution and this is not always the case and 2) discretization of inversion formulas can be very painful. Iterative methods, which achieve tomographic reconstruction by iterative convergence to the solution, are on the other hand intrinsically discrete. The iterative nature of these algorithms make them very flexible but convergence to a satisfactory solution may be a very slow process.

1.1.3 Some considerations on scatter

As X-ray photons travel through the examined object, scattered photons are generated as depicted in Figure 1.5. Scattered photons are bad because they a) increase the patient radiation dose, b) introduce non-linearity in the X-ray transform equation and c) deteriorate the power spectrum of the detected signal. For low energy X-rays the SNR can be kept high by using anti-scatter grids or air-gaps. These methods are unfortunately not effective for high energy X-rays and more sophisticated techniques have been proposed [77].

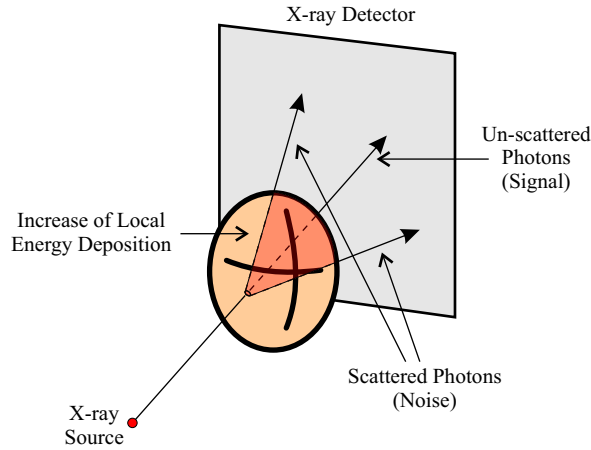


Figure 1.5. Scattered photons increase local energy deposition (radiation dose given to the patient) and degrade detected signal.

The radiation dose due to scattered X-rays is evidently somewhat proportional to the irradiated volume. Carlsson in his tomography survey [15] reports a scatter-to-primary ratio of about 10% for conventional and spiral CT and as high as 40-200% for cone-beam CT. This basically means that to obtain the same image “quality” (same SNR) in CBCT as in SCT, we expect the dose per examination to increase by some amount.

1.2 Objectives and outline of this thesis

The objectives of this thesis were twofold. Our first task was to develop a software toolkit dedicated to evaluation, simulation and computation housing of cone-beam computed tomography reconstruction algorithms. The final product of this task is *Strange Engine*, a modern object-oriented visual software platform in which CBCT reconstruction algorithms are encapsulated in Microsoft ActiveX plug-ins. The mathematical framework for the toolkit is presented in chapters 2 and 3. A schematic outline of Strange Engine features is illustrated in Figure 1.6, a detailed description will be given in chapter 4.

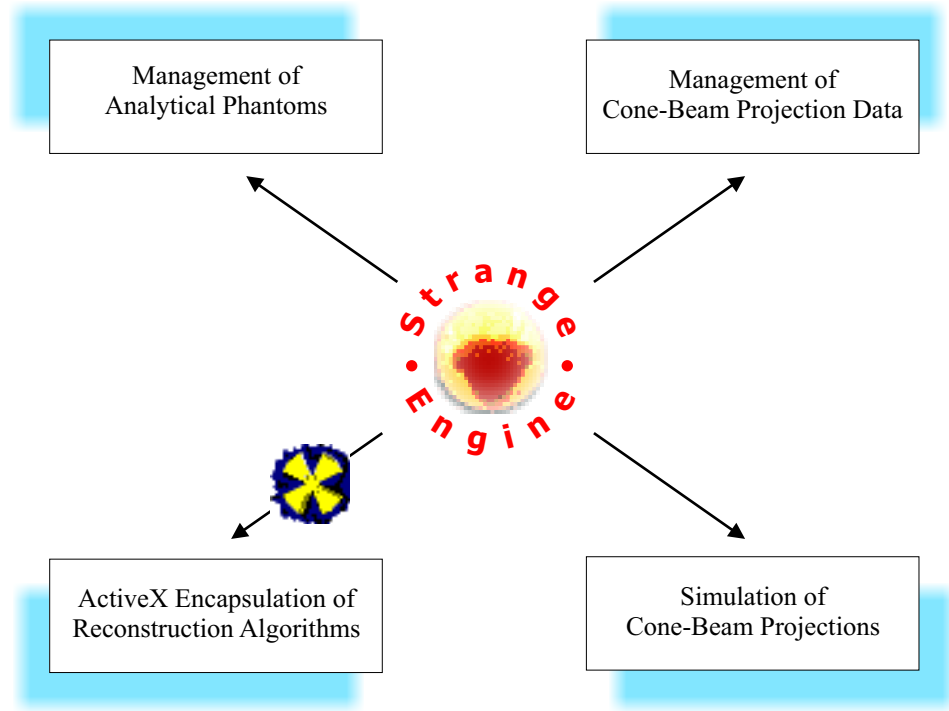


Figure 1.6. Schematic outline of the cone-beam reconstruction facilities available in our object-oriented visual toolkit Strange Engine.

Our second task was to use Strange Engine to implement and test different cone-beam reconstruction algorithms. Chapter 5 is devoted to the description and benchmarking of our custom implementations of the Feldkamp, rebinned Feldkamp and Louis reconstruction formulas. Simulated and real-world projection data acquired thanks to our collaboration with the National Cancer Research Institute of Genova is used to validate reconstruction results.

1.3 Author references

Certain results of this thesis have been discussed by Agostinelli in [2] and published by Agostinelli and Foppiano in [4, 5], Agostinelli and Paoli in [7] and by Agostinelli et al. in [6]. Strange Engine won the third prize for the communication “*3D Tomographic Reconstruction Active Style*” [3] at the ACM International Graduate Student Contest held in Austin, Texas, May 8-12 2000.

2D Reconstruction

In the following sections we introduce the mathematical principles of two-dimensional tomographic reconstruction. This will serve as a background for the next chapter *3D Cone-Beam Reconstruction*. Crucial parts of this discussion will include the definition of the Radon transform, the Fourier Slice Theorem (FST) and the Filtered Back Projection (FBP) reconstruction algorithm for parallel-beam and fan-beam geometries. For the sake of simplicity we will study the problem in the third-generation scanner set-up in which a planar detector is used to measure the transmitted beam profile.

2.1 Projections and the Radon transform

2.1.1 Projections measurement

Let us consider the simple experiment presented in Figure 2.7: a monochromatic pencil beam of X-rays goes through a sample of homogeneous material. A X-ray detector, put far away from the sample, measures the photons which did interact with the sample matter. In these conditions the attenuation of the beam is described by a simple macroscopic coefficient μ called the linear attenuation coefficient and defined according to the Beer's equation

$$I_t = I_i e^{-\mu \Delta l} \quad (2.1)$$

where I_t and I_i are respectively the transmitted and incident intensities of the beam and Δl is the length of the sample along the beam path. In the general

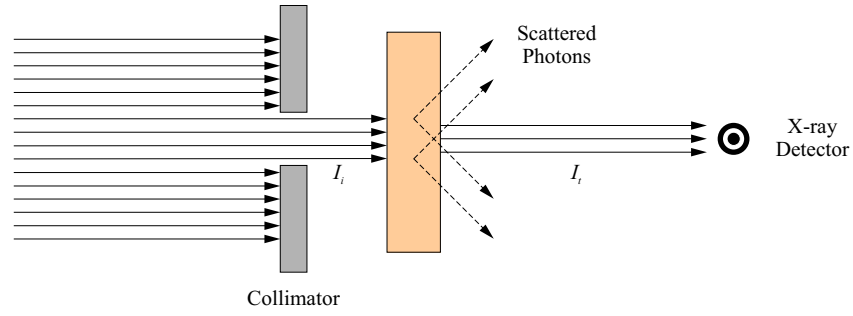


Figure 2.7. Transmission experiment for monochromatic X-rays by which the linear attenuation coefficient μ can be defined.

case of a sample described by the function $\mu(x, y)$ we can write the transmitted intensity along the monochromatic l -ray as

$$I_t = I_i e^{-\int_{l\text{-ray}} \mu(x, y) dl} \quad (2.2)$$

Equation (2.2) is the first milestone of computed tomography. Basically it connects ray integrals or projections of the object function $\mu(x, y)$ to measurable quantities, i.e. photon intensities, through the relation

$$\int_{l\text{-ray}} \mu(x, y) dl = -\ln \frac{I_t}{I_i} \quad (2.3)$$

This equation in reality is just an approximation because a) it does not take into account the energy spectrum and the hardening of the X-ray beam, b) models X-ray beams as infinitely thin, c) assumes ideal X-ray detectors and perfect scatter rejection. In fact deviations from the aforementioned relation are quite common and produce image artefacts which will not be investigated here and for which we refer to what said in [40, 83].

2.1.2 The 2D Radon transform

Let us consider the ideal projection system depicted in Figure 2.8 in which X-rays travel along parallel lines. In this parallel-beam geometry each l -ray is a line conveniently described by the angle $\theta \in [0, \pi)$ and its signed distance from origin t such that

$$x \cos \theta + y \sin \theta = t \quad (2.4)$$

which can also be expressed using polar coordinates as

$$\mathbf{r} \cdot \boldsymbol{\xi} = t \quad (2.5)$$

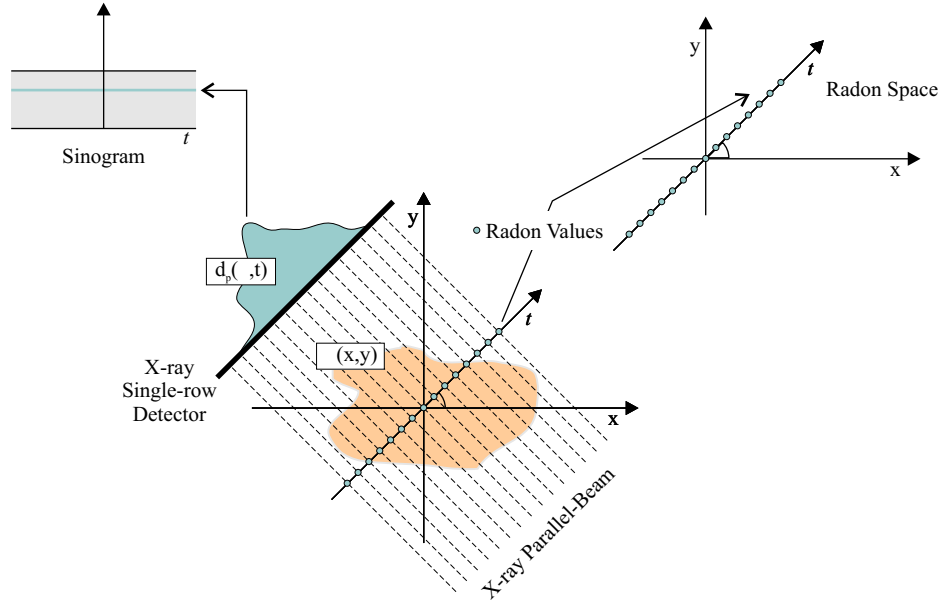


Figure 2.8. The parallel-beam projection geometry and the 2D Radon transform.

where $\mathbf{r} = (x, y)$ and $\boldsymbol{\xi} = (\cos \theta, \sin \theta)$. Let us denote as $d_p(\theta, t)$ a parallel-beam projection for a given θ and t . Using the Dirac delta as a l -ray selector, $d_p(\theta, t)$ can be written as

$$d_p(\theta, t) = \iint \mu(x, y) \delta(x \cos \theta + y \sin \theta - t) dx dy \quad (2.6)$$

The Cartesian representation of the parallel-beam projections $d_p(\theta, t)$ with (θ, t) in $[0, \pi) \times \Re$ is said to be the *sinogram* of the object.

Another way to look at the projections is the 2D Radon space associated to the 2D Radon transform which we will denote as \mathcal{R} . The two-dimensional Radon transform of $\mu(\mathbf{r})$ at some point \mathbf{P} is defined as the line integral along the l -ray passing through it with direction orthogonal to the distance from origin vector $(\mathbf{P} - \mathbf{O})$. We can write the 2D Radon transform as

$$\mathcal{R}\mu(t, \boldsymbol{\xi}) = \iint \mu(\mathbf{r}) \delta(\mathbf{r} \cdot \boldsymbol{\xi} - t) d\mathbf{r} \quad (2.7)$$

Radon transform values can be merely recovered by the knowledge of the parallel-beam projections because

$$\iint \mu(\mathbf{r}) \delta(\mathbf{r} \cdot \boldsymbol{\xi} - t) d\mathbf{r} = \iint \mu(x, y) \delta(x \cos \theta + y \sin \theta - t) dx dy = d_p(\theta, t)$$

2.1.3 The Fourier Slice Theorem

By a mathematical point of view, the two-dimensional reconstruction problem is to recover the object function $\mu(x, y)$ from its 2D Radon transform.

The Fourier Slice Theorem (FST) establishes a fundamental relation connecting the object space to the 2D Radon space. This connection is provided by the Fourier transform \mathcal{F} in the following way

$$\mathcal{F}_t \mathcal{R}\mu(\omega, \xi) = \mathcal{F}_2 \mu(\omega \xi) \quad (2.8)$$

where \mathcal{F}_t denotes the 1D radial Fourier transform and \mathcal{F}_2 the 2D Fourier transform. In terms of the sinogram this can also be written as

$$\hat{d}_p(\theta, \omega) = \hat{\mu}(\omega \cos \theta, \omega \sin \theta) \quad (2.9)$$

where, with a slight abuse of notation, we have introduced hats ($\hat{\cdot}$) to denote Fourier transformed functions.

Proof. By the definition of the Fourier transform we can write

$$\mathcal{F}_t \mathcal{R}\mu(\omega, \xi) = \int \mathcal{R}\mu(t, \xi) e^{-2\pi i \omega t} dt$$

Substituting the Radon transform definition formula (2.7) and using Dirac delta properties we obtain

$$\begin{aligned} \mathcal{F}_t \mathcal{R}\mu(\omega, \xi) &= \int \int \mu(\mathbf{r}) \delta(\mathbf{r} \cdot \xi - t) d\mathbf{r} e^{-2\pi i \omega t} dt \\ &= \int \mu(\mathbf{r}) d\mathbf{r} \int \delta(\mathbf{r} \cdot \xi - t) e^{-2\pi i \omega t} dt \\ &= \int \mu(\mathbf{r}) e^{-2\pi i \omega \mathbf{r} \cdot \xi} d\mathbf{r} = \mathcal{F}_2 \mu(\omega \xi) \end{aligned}$$

□

2.1.4 The fan-beam projection geometry

Till this point we have considered parallel-beam projections. A much clever and more realistic way of acquiring Radon data is to employ rays departing from a X-ray source while this rotates on a circle around the object.

The geometry under consideration is illustrated in Figure 2.9: rays form a so called fan-beam diverging from the source positioned at point

$$\mathbf{S}(\beta) = R(\sin \beta, -\cos \beta)$$

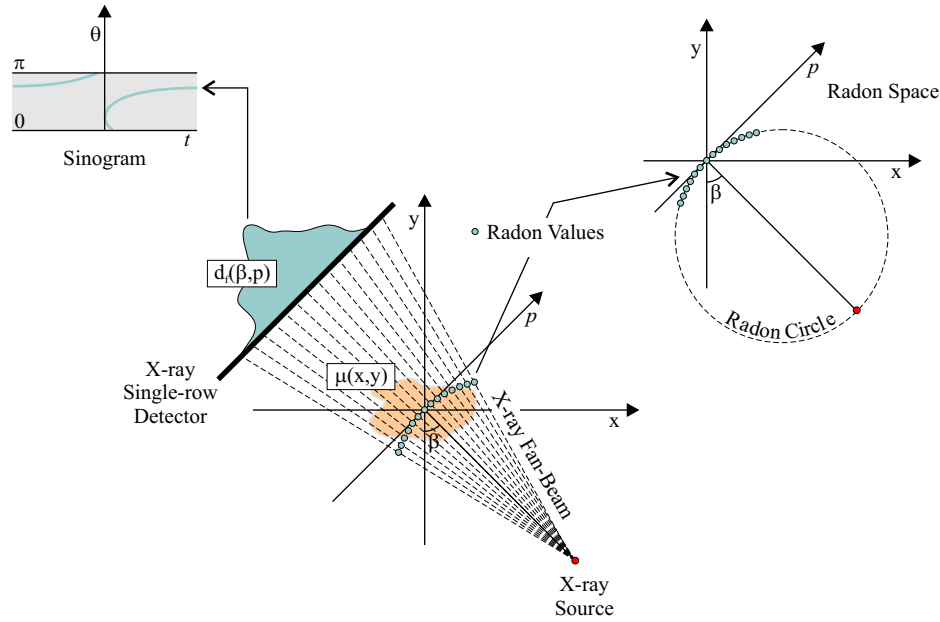


Figure 2.9. The fan-beam projection geometry.

where β is the source angle and R is the source-origin distance. Each ray of a fan-beam projection d_f is conveniently described by the coordinates pair $(\beta, p) \in [0, 2\pi) \times \mathfrak{R}$, where the p axis is orthogonal to the source-origin vector.

To connect fan-beam projections to the two-dimensional Radon transform we rewrite the parameterisation of fan-beam projections $d_f(\beta, p)$ in terms of parallel-beam coordinates (θ, t) :

$$\begin{cases} \theta = \beta - \gamma \\ t = p \cos \gamma \end{cases} \quad (2.10)$$

where we introduced the ray angular aperture γ defined in Figure 2.13 and such that $\gamma = \arctan(p/R)$.

We have seen that a parallel-beam projection taken at θ angle produces Radon data along the same t axis direction. A fan-beam projection taken at β angle produces Radon data along the Radon circle passing through the source and the origin. This can be easily demonstrated noting that, because of the relationship (2.10), the following equality holds

$$|t\xi - \frac{1}{2}\mathbf{S}|^2 = \frac{1}{4}R^2$$

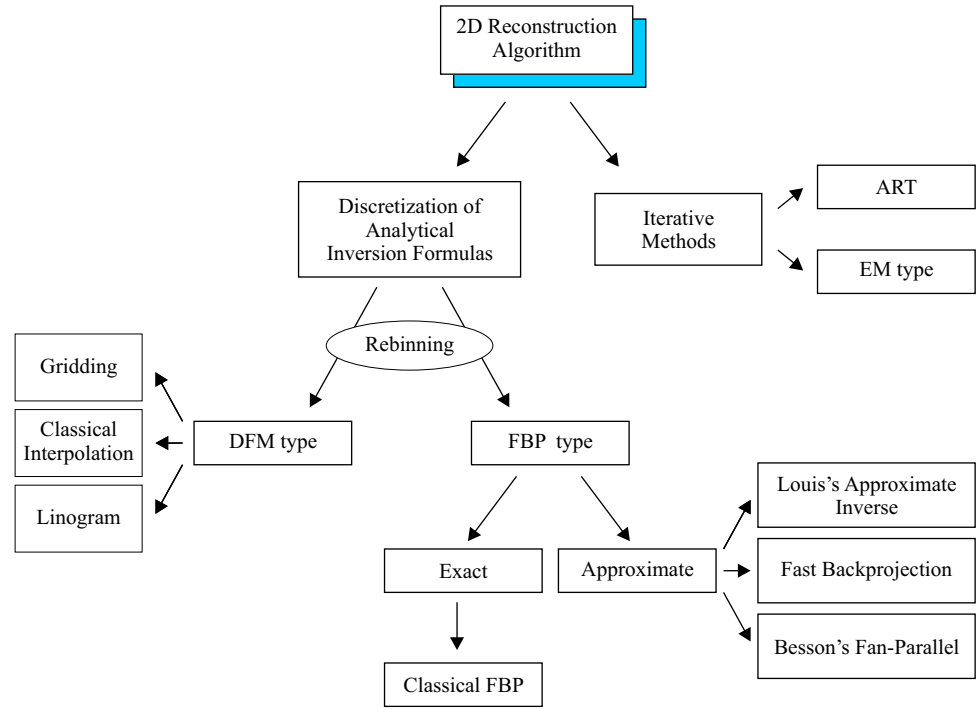


Figure 2.10. Detailed overview of 2D tomographic reconstruction algorithms.

2.2 2D reconstruction algorithms overview

Figure 2.10 presents a schematic classification of 2D reconstruction algorithms. A first distinction is made for algorithms derived from analytical inversion formulas based on the Fourier slice theorem and algorithms in which an explicit discrete version of the problem is solved by an iterative method. Algorithms based on discretization of analytical inversion formulas are then split in algorithms in which the reconstruction is performed through two-dimensional Fourier inversion and algorithms based on filtered back projection in which the reconstruction is done projection by projection. Several variants of algorithms based on FBP differing in filtering, interpolation and approximation schemes are also investigated. These include Louis's approximate inverse reconstruction based on mollifiers and fast filtered back projection introduced by the Linköping group to reduce computational complexity of FBP.

In the following sections we will present the basic principles of each 2D reconstruction algorithm. Greater coverage will be provided for arguments functional for the presentation of the cone-beam algorithms given in next chapter.

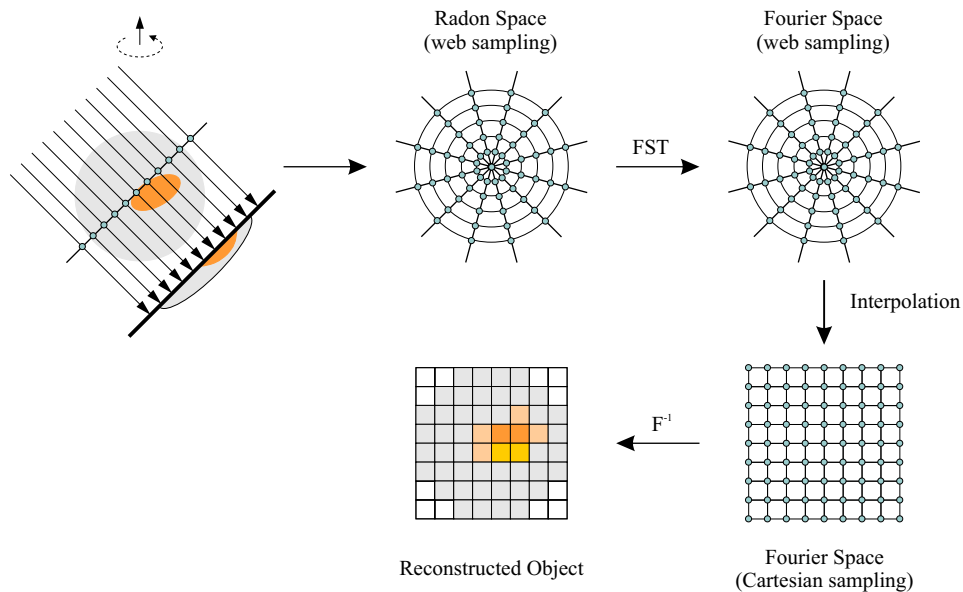


Figure 2.11. Reconstruction steps for Direct Fourier Methods.

2.3 Direct Fourier Methods

Reconstruction algorithms which pursue the inversion of the Radon transform by means of the straight application of the Fourier slice theorem are called Direct Fourier Methods (DFM). As shown in Figure 2.11 for the parallel-beam geometry, the steps followed in a DFM reconstruction are essentially three:

1. projections are Fourier transformed. By the FST this is equivalent to sample the Fourier space on a not-Cartesian grid;
2. the Fourier space is re-sampled in a Cartesian grid by interpolation,
3. the object is reconstructed applying the 2D Fourier inversion formula.

The critical part of any DFM is the interpolation step: not trivial interpolation schemes have to be used to obtain good reconstruction results. For example Magnusson in [50] reports success using an 8-point interpolation kernel in the radial direction and linear interpolation along the angular direction. Other approaches include fast gridding [59, 68] and linogram sampling [23, 24, 50]. Direct Fourier Methods computational complexity is of order $O(N^2 \log N)$. This does not take into account real computer code complexity which makes DFM not much faster than other 2D reconstruction algorithms.

2.4 Filtered Back Projection

A problem with DFM is that the reconstruction is not done per projection but on the whole set of projections. This can create memory consumption problems and make direct Fourier methods somewhat cumbersome numerically. A per projection reconstruction algorithm is the Filtered Back Projection (FBP) which is in fact the most commonly used reconstruction algorithm. While the computational complexity of FBP is of order $O(N^3)$ it comprises some advantages in comparison of DFM including that a) no special interpolation technique is needed, b) reconstruction is per projection so it is possible to reconstruct while continuing to acquire new projections and c) computer implementation is easier and more efficient.

FBP reconstruction for the parallel-beam geometry is depicted in Figure 2.12. Basically two steps are necessary:

1. projections are filtered using a convolution kernel,
2. projections are back projected.

Filtering is usually performed in the Fourier space. Back projection is usually pixel driven and involves ray tracing and simple interpolation of detector data.

2.4.1 FBP parallel-beam

Using equation (2.9) we can write

$$\begin{aligned}
 \mu(x, y) &= \mathcal{F}_2^{-1} \hat{\mu}(x, y) = \iint \hat{\mu}(u, v) e^{2\pi i(ux+vy)} du dv \\
 &= \int \int_0^\pi \hat{d}_p(\theta, \omega) e^{2\pi i\omega(x \cos \theta + y \sin \theta)} |\omega| d\omega d\theta \\
 &= \int_0^\pi d\theta \int \hat{d}_p(\theta, \omega) |\omega| e^{2\pi i\omega(x \cos \theta + y \sin \theta)} d\omega \\
 &= \int_0^\pi \tilde{d}_p(\theta, x \cos \theta + y \sin \theta) d\theta
 \end{aligned} \tag{2.11}$$

where we have defined the filtered projections

$$\tilde{d}_p(\theta, t) = \int \hat{d}_p(\theta, \omega) |\omega| e^{2\pi i\omega t} d\omega \tag{2.12}$$

and where the term $(x \cos \theta + y \sin \theta)$ represent the actual backprojection.

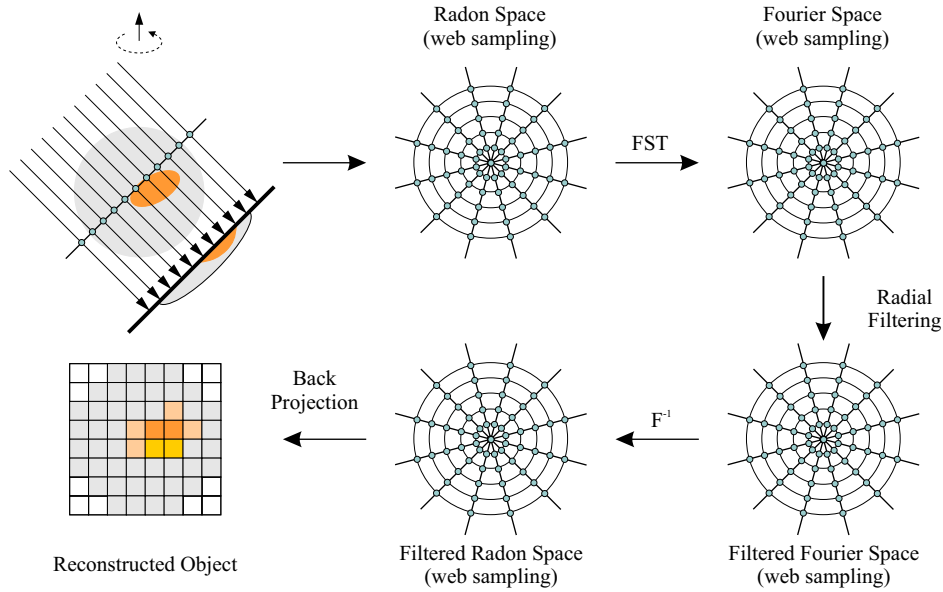


Figure 2.12. Reconstruction steps for Filtered Back Projection.

2.4.2 FBP fan-beam

Using the relationship (2.10) connecting the fan-beam geometry to the parallel-beam geometry and the previous FBP formula (2.11), we can re-formulate [40, 83] the FBP algorithm for the fan-beam geometry as follows

$$\begin{aligned} \mu(x, y) &= \frac{1}{2} \int_0^{2\pi} \frac{R^2}{(R-s)^2} \int \frac{R}{\sqrt{R^2 + p^2}} d_f(\beta, p) k\left(t \frac{R}{R-s} - p\right) dp d\beta \\ &= \frac{1}{2} \int_0^{2\pi} \frac{R^2}{(R-s)^2} \tilde{d}_f\left(\beta, \frac{R}{R-s}(x \cos \theta + y \sin \theta)\right) d\beta \end{aligned} \quad (2.13)$$

where $s = -(y \cos \beta - x \sin \beta)$ is the distance of the (x, y) pixel from a virtual detector centred on the origin and where we have defined the filtered pre-weighted projections

$$\tilde{d}_f(\beta, p) = \int \mathcal{F}\left(\frac{R}{\sqrt{R^2 + p^2}} d_f\right)(\theta, \omega) |\omega| e^{2\pi i \omega t} d\omega \quad (2.14)$$

Note that in the FBP fan-beam formula a) the backprojection term includes the magnification factor $R/(R-s)$ and that b) each filtered pre-weighted projection has to be multiplied by a pixel dependent $1/r^2$ factor.

2.4.3 Filtering

Let's now briefly examine the filtering step involved in FBP reconstruction. As showed above for both parallel-beam and fan-beam geometries, filtering reduces to multiplication in Fourier space of transformed projections by the *ramp filter*

$$\hat{k}_{B_\infty}(\omega) = |\omega|$$

In the object space this is equivalent to convolution with the impulse response or kernel function

$$k_{B_\infty}(t) = \int \hat{k}_{B_\infty}(\omega) e^{2\pi i \omega t} d\omega = \int |\omega| e^{2\pi i \omega t} d\omega \quad (2.15)$$

Unfortunately the ramp filter is not in \mathcal{L}_2 and the kernel $k_{B_\infty}(t)$ does not exist. Moreover in computer code we have to take into account that projections are sampled with τ steps. The solution is to band limit the reconstruction filter to some frequency w :

$$k_{B_w}(t) = \int_{-w}^{+w} |\omega| e^{2\pi i \omega t} d\omega \quad (2.16)$$

Integrating by parts we obtain the Ram-Lak [61, 40] class filters

$$k_{B_w}(t) = \left[2 \frac{\sin(2\pi w t)}{2\pi w t} - \left(\frac{\sin(\pi w t)}{\pi w t} \right)^2 \right] w^2 = [2 \operatorname{sinc}(2\pi w t) - \operatorname{sinc}^2(\pi w t)] w^2$$

To this regard we should also mind the Shannon's sampling theorem which states that to avoid aliasing it's necessary to band limit up to the Nyquist frequency given by $\Omega = 1/(2\tau)$. Band limiting the filter to Ω we get

$$\begin{aligned} k_{B_\Omega}(t) &= \int_{-\Omega}^{+\Omega} |\omega| e^{2\pi i \omega t} d\omega \\ &= \frac{1}{2\tau^2} \operatorname{sinc}(2\pi t/2\tau) - \frac{1}{4\tau^2} \operatorname{sinc}^2(\pi t/2\tau) \end{aligned} \quad (2.17)$$

which can conveniently be expressed in the discretized form

$$k_{B_\Omega}(n\tau) = \begin{cases} 1/4\tau^2 & n = 0 \\ 0 & n \text{ even} \\ -\frac{1}{(n\pi\tau)^2} & n \text{ odd} \end{cases} \quad (2.18)$$

where n is the sampling index.

Nyquist band limited Ram-Lak filter is probably the most commonly used reconstruction filter for FBP. However when projection data is noisy, the high

frequency exaltation performed by pure ramp filters may lead to noise amplification and hence poor reconstruction quality. In this situation it is better to opt for a less aggressive filter in which reconstruction resolution is traded for better noise suppression. To this end we can introduce the windowed or apodized ramp filter

$$\hat{k}_W(\omega) = |\omega| W(\omega) \quad (2.19)$$

where $W(\omega)$ is called the *apodization window*. If

$$W(\omega) = B_w(\omega) = \begin{cases} 1 & \text{if } |\omega| < w \\ 0 & \text{otherwise} \end{cases}$$

the apodized ramp filter equates to the \hat{k}_{B_w} Ram-Lak filter. Other apodization windows include the well-known Hann and Hamming filters, used in digital signal processing, and the popular Shepp–Logan window [73]

$$W(\omega) = W_{sl}(\omega) = \begin{cases} \frac{\sin(\omega\pi\tau)}{\omega\pi\tau} & \text{if } |\omega| < \Omega = 1/2\tau \\ 0 & \text{otherwise} \end{cases} \quad (2.20)$$

which corresponds in discretized form to the kernel

$$k_{W_{sl}}(n\tau) = \frac{1}{\pi^2(1 - 4n^2\tau^2)} \quad (2.21)$$

Choosing an appropriate window is often not easy because one have to balance the over-smoothing of fine details (i.e. resolution) with noise reduction. To address this problem new tools based on multi-resolution wavelets [14, 89] and approximate inverse (see § 2.7) have been recently proposed.

2.4.4 Aliasing

In this section we will present some simple but crucial observation about aliasing and sampling conditions for filtered backprojection. The same results apply similarly for the other 2D reconstruction methods derived from the Fourier slice theorem while iterative algorithms are much more robust to this regard.

The sampling conditions are defined specifying the following quantities:

$N_x \times N_y$ = number of pixels of the reconstructed image

N_t or N_p = number of rays per projection in parallel or fan-beam geometry

N_θ or N_β = number of projections in parallel or fan-beam geometry

The reconstructed image can suffer two different kinds of artefacts due to discretization and bad sampling: a) if the resolution of the reconstructed image does not match the bandwidth of the filtered projections aliasing produces Moiré patterns, b) if the number of projections does not match the number of rays per projection dark/light streaks are produced.

Number of rays per projection. Let $N = N_x = N_y$. We have seen in the preceding section that to avoid aliasing during the filtering stage we have to band limit the reconstruction kernel up to Nyquist frequency $\Omega = 1/2\tau$. By Fourier slice theorem the filter bandwidth w is also the bandwidth of the 2D Fourier transform of the object. This basically means that filter bandwidth should match the bandwidth required to reconstruct the object on N^2 pixels. In parallel-beam geometry the detector dimension is roughly equal to the linear dimension of the reconstruction region. This implies that the best reconstruction will be provided by $N_t = N$ and $w = \Omega$. For the fan-beam geometry the previous result is modified inserting a magnification factor.

Number of projections. If the number of projections is low the reconstruction of a point is a star-shaped object. In fact in the backprojection step a projection value is smeared along the strip of pixels traversed by the back projected ray. Increasing the number of projections this effect vanishes but if N_θ does not match the number of rays, dark or light streaks may still show in the reconstructed image. To see how to prevent these effects to appear, let us look again to the Radon pattern in Fourier space for parallel-beam geometry in Figure 2.11. Radon values are distributed more densely at low frequencies and become sparser and sparser as frequency increases. A well balanced interpolation to Cartesian coordinates is possible only if even at the Nyquist frequency, i.e. the highest allowed frequency, the azimuthal resolution matches the radial resolution. This means that:

$$\Omega \Delta\theta = \frac{1}{2\tau} \Delta\theta = \frac{1}{2\tau} \frac{\pi}{N_\theta} \simeq \frac{1}{\tau N_t} \rightarrow N_\theta = \frac{\pi}{2} N_t$$

2.5 Rebinned FBP

Another way to look at the fan-beam FBP reconstruction is to use *rebinning*. With this term we refer to the rearrangement of rays which produce parallel-beam projections data from fan-beam projections data. Rebinning basically involves the coordinates change described in the rebin equation (2.10) and some sort of azimuthal and radial interpolation:

$$d_p(\theta, t) = d_f(\theta + \gamma, t/\cos \gamma) \quad (2.22)$$

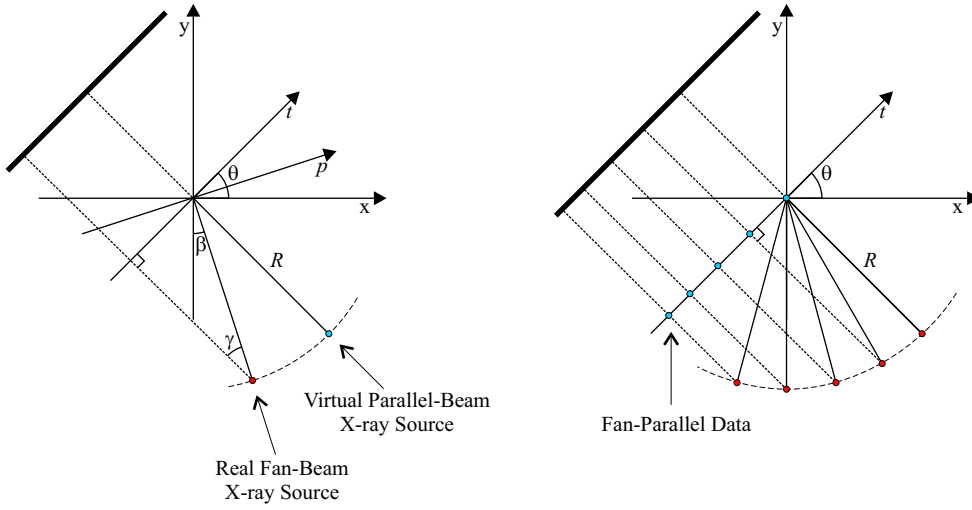


Figure 2.13. Left: geometric sketch for the re-parameterisation of fan-beam coordinates to parallel-beam coordinates. Right: the angular rearrangement of fan-beam rays leads to Besson’s fan-parallel projections.

where the angular aperture can be conveniently computed as

$$\gamma = \arcsin(t/R)$$

and where t is equispatially sampled. After rebinning the usual parallel-beam filtered backprojection reconstruction formula can be used. This has a significant advantage in terms of reconstruction speed because of the removal of the $1/r^2$ factor which slows down the backprojection step. Clearly the trade-off is the additional computational cost of rebinning and also the resolution loss introduced by interpolation.

2.5.1 Besson’s fan-parallel formula

In [9, 10] Besson’s describes a modified FBP algorithm using a one step rebinning method in which only azimuthal interpolation is performed. The rebinning step consists essentially of fan-beam data reorganization to form so called fan-parallel projections as depicted in Figure 2.13. Contrary to parallel-beam projections, in fan-parallel projections $d_{fp}(\theta, t)$ we do not force t to be sampled equispatially so no radial interpolation is necessary. This should enhance reconstruction resolution because radial interpolation introduces some attenuation of the higher frequencies.

2.6 Fast Filtered Back Projection

Introduced by Brady in [13] and by Nilsson in [57], fast filtered back projection has been investigated by Ingerhed in [36]. Fast FBP (FFBP) achieves reconstruction with $O(N^2 \log N)$ complexity compared to $O(N^3)$ for classical FBP. The FFBP algorithm is not exact and works for parallel-beam geometry only, so in the case of fan-beam projections a rebinning step is necessary.

Speed-up is accomplished in the backprojection stage by approximating the sinusoidal backprojection path in the sinogram with a segmented curve whose values can be computed with an efficient divide-and-conquer strategy. This is also the reason why in fast filtered backprojection the number of projections should be a power of 2.

2.7 Approximate Inverse

The concept of the approximate inverse was introduced by Louis in [47]. Basically the approximate inverse is a general scheme to obtain stable inversion of ill-posed linear problems such as the inversion of the 2D Radon transform.

2.7.1 Definition

Let us consider a linear and continuous operator $A : X \rightarrow Y$ connecting the Hilbert spaces X and Y . We want to find f such that

$$A f = g \quad (2.23)$$

The key idea of the approximate inverse is to compute instead of f an approximate version f_γ which is defined introducing the *mollifier* e_γ in the following way

$$f_\gamma(\mathbf{x}) = \langle f, e_\gamma(\mathbf{x}, \cdot) \rangle = \int f(\mathbf{y}) e_\gamma(\mathbf{x}, \mathbf{y}) d\mathbf{y} \quad (2.24)$$

where γ is a scalar regularization parameter. Obviously the perfect mollifier would be the Dirac delta

$$e(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{x} - \mathbf{y})$$

but as our problem is ill-posed we are forced to introduce some sort of regularization. In the approximate inverse scheme this regularization is expressed in terms of the mollifier and of the mollified solution. Louis has shown in [47, 48] this approach to be very general and that classical regularization techniques such as Tikhonov-Phillips and even iterative methods such as conjugate gradient and Landweber, can be written as approximate inverse instances.

In the following we show how to calculate f_γ from g . Suppose there exists a reconstruction kernel function k_γ such that

$$A^* k_\gamma(\mathbf{x}) = e_\gamma(\mathbf{x}) \quad (2.25)$$

is solvable, then we can write the solution to (2.23) in the approximate inverse sense as

$$\begin{aligned} f_\gamma(\mathbf{x}) &= \langle f, e_\gamma(\mathbf{x}, \cdot) \rangle = \langle f, A^* k_\gamma(\mathbf{x}) \rangle \\ &= \langle A f, k_\gamma(\mathbf{x}) \rangle = \langle g, k_\gamma(\mathbf{x}) \rangle = S_\gamma g(\mathbf{x}) \end{aligned} \quad (2.26)$$

where the operator A^* is the adjoint of A and S_γ is the so called approximate inverse operator.

If equation (2.25) is not solvable we can in any case use the kernel minimizing the defect in least squares sense, that is the k_γ such that

$$A A^* k_\gamma(\mathbf{x}) = A e_\gamma(\mathbf{x}) \quad (2.27)$$

and compute $\tilde{f}_\gamma = S_\gamma g(\mathbf{x}) = \langle g, k_\gamma(\mathbf{x}) \rangle$, the approximate inverse solution in least squares sense of (2.23) or

$$A^* A \tilde{f} = A^* g \quad (2.28)$$

2.7.2 Application to 2D computer tomography

Let us now see how the 2D tomographic reconstruction problem can be solved within the approximate inverse framework. First of all we will consider the parallel-beam geometry and the associated 2D Radon transform definition (2.7)

$$d_p(\theta, t) = \mathcal{R}\mu(\theta, t) = \int \mu(\mathbf{r}) \delta(\mathbf{r} \cdot \boldsymbol{\xi} - t) d\mathbf{r}$$

This equation is exactly of the kind $A f = g$ with $f = \mu$, $g = d_p$ and $A = \mathcal{R}$. We now choose the mollifier e_γ to be space-invariant and such that

$$e_\gamma(\mathbf{x}, \mathbf{y}) = \bar{e}_\gamma(\mathbf{x} - \mathbf{y})$$

where \bar{e}_γ is a sufficiently smooth and circular symmetric function. Under these assumptions it can be shown [49] that the approximate inverse leads to the filtered back projection formula:

$$f(x, y) \simeq f_\gamma(x, y) = \int \int d_p(\theta, t) k_\gamma(x \cos \theta + y \sin \theta - t) dt d\theta \quad (2.29)$$

The preceding equation means that by using approximate inverse formalism we can try to master or at least improve the design of FBP filters.

2.7.3 Filter design

We have discussed in § 2.4.3 the filtering operation in the FBP algorithm. The approximate inverse scheme allows to look at filter design from a different perspective. This is possible thanks to relations (2.27) and (2.29) which connects shift-invariant circular symmetric mollifiers to FBP reconstruction kernels. Before proceeding further let us introduce an additional simplification of the problem by supposing the \bar{e}_γ mollifier to be dilation invariant:

$$\bar{e}_\gamma(\mathbf{x}) = T_1^\gamma e(\mathbf{x}) = \gamma^2 e(\gamma \mathbf{x})$$

It can be seen [49] that under these conditions the reconstruction kernel follows the same rule, that is

$$k_\gamma(t) = \gamma^2 k(\gamma t) \quad (2.30)$$

where $\mathcal{R}\mathcal{R}^*k = \mathcal{R}e$. Two possible pathways can be followed to produce approximate inverse reconstruction kernels: a) a suitable Singular Value Decomposition (SVD) of the Radon transform is used to compute the reconstruction kernel as a truncated series. For example the SVD of the Radon transform in parallel-beam geometry for objects defined in the unit ball leads [22] to

$$k(t) = \frac{1}{\pi^2} \sum_{m=0}^{\infty} (2m+1) U_{2m}(t) I_{2m} \simeq \frac{1}{\pi^2} \sum_{m=0}^M (2m+1) U_{2m}(t) I_{2m}$$

where $e(x)$ is a space, rotation and dilation invariant mollifier, U_m are Tschebyschew polynomials of second kind and

$$I_{2m} = \int_0^1 e(x) U_{2m}(x) dx$$

b) Usage of explicit filtered backprojection formula for fast decaying functions. As discussed in [56, 22, 65] introducing the Riesz potential I^{-1} such that $\mathcal{F}I^{-1}f(\boldsymbol{\omega}) = |\boldsymbol{\omega}| \hat{f}(\boldsymbol{\omega})$ it's possible to write explicitly the Radon transform inversion formula for fast decaying functions as

$$e = \frac{1}{2\pi} \mathcal{R}^* I^{-1} \mathcal{R} e$$

From reconstruction kernel definition (2.27) it follows that

$$k = \frac{1}{2\pi} I^{-1} \mathcal{R} e$$

which can be conveniently written in Fourier space

$$\hat{k}(\omega) = \frac{1}{2\pi} |\omega| \mathcal{R} e(\omega)$$

Example: Ram-Lak mollifier. Consider the mollifier

$$e^j(\mathbf{x}) = \frac{1}{2\pi} \frac{J_1(|\mathbf{x}|)}{|\mathbf{x}|}$$

where $J_1(x)$ is the Bessel function of first kind and first order. Using Riesz potential the Fourier transform of the reconstruction kernel is

$$\hat{k}^j(\omega) = \frac{1}{2}(2\pi)^{-3/2}|\omega|B_\Omega(\omega)$$

which corresponds exactly to the Ram-Lak filter band limited to Nyquist frequency. Every other Ram-Lak filter is obtained by scaling the mollifier with the parameter γ . Applying rule (2.30) we see that γ is in fact correlated to the filter effective bandwidth w

$$k_\gamma^j(t) = \gamma^2 k^j(\gamma t) = k_{B_w} \text{ with } w = \gamma\Omega \quad (2.31)$$

Example: Gaussian mollifier. The Gaussian mollifier is defined as

$$e^g(\mathbf{x}) = \frac{1}{2\pi} e^{-|\mathbf{x}|^2/2}$$

Using Riesz potential we can write

$$k^g(t) = \frac{1}{\pi} \int_0^\infty |\omega| \hat{e}(\omega) \cos \omega t dt$$

Integrating by parts we obtain

$$k^g(t) = \frac{1}{2\pi^2} \left[1 + it\sqrt{\pi/2} \operatorname{erf}\left(it\sqrt{1/2}\right) e^{-t^2/2} \right]$$

where $\operatorname{erf}(t) = 1/\sqrt{\pi} \int_{-t}^t e^{-z^2} dz$ is the error function.

2.7.4 Reconstruction formula for the fan-beam geometry

Derivation of an approximate inverse reconstruction formula for the fan-beam geometry is more involved than the one for parallel-beam geometry. The problem is essentially that the Radon transform of a mollifier in fan-beam geometry is space-variant because the magnification factor depends on the source-to-mollifier distance. This would imply a space-variant reconstruction kernel. Dietz have shown in [22] that for small fan aperture angles, the approximate inverse reconstruction formula can still be written in the form of fan-beam

filtered backprojection (2.13) with the pre-weighted filtered projections given by

$$\tilde{d}_f(\beta, p) = \int \mathcal{F}\left(\frac{R}{\sqrt{R^2 + p^2}} d_f\right)(\theta, \omega) \hat{k}_\gamma(\omega) e^{2\pi i \omega t} d\omega \quad (2.32)$$

where $k_\gamma(\omega)$ is the parallel-beam reconstruction kernel associated to a space, rotation and dilation invariant e_γ mollifier.

2.8 Iterative Methods

Iterative methods are based on successive refinements of the reconstruction. In general the reconstruction starts with an initial guess and enters an iteration cycle which follows three steps:

1. the current estimation of the reconstructed object is used to compute expected projections,
2. the comparison of measured projections with expected projections is used to calculate update factors,
3. update factors are applied to get a new current estimation.

A plethora of iterative methods have been proposed. They differ mostly by the update rule and by the model used to compute expected projections. Among them we would like to mention Algebraic Reconstruction Techniques (ART) introduced by Gordon and Herman in [27] and methods based on Expectation Maximization (EM) like the TRansmission Maximum Likelihood (TRML) proposed by Lange and Carson in [44]. Most of the iterative methods are not used to perform real tomographic reconstruction because they tend to be much slower than algorithms based on discretization of analytic formulas like FBP. However new optimised ordering techniques [51, 32] and special applications like metal artefacts removal [82], beam hardening correction [34] and reconstruction from incomplete data have revamped interest in these methods.

3D Cone-Beam Reconstruction

We have motivated three-dimensional cone-beam reconstruction in the introductory chapter. Essentially we have showed that the cone-beam geometry allows to improve the overall effectiveness of X-ray examinations both in terms of speed and image quality. One big issue posed by cone-beam computed tomography is the research and development of an adequate reconstruction algorithm. In this chapter we review the principles of three-dimensional cone-beam computed tomography and introduce the most widespread 3D CBCT reconstruction algorithms that have been proposed in the literature. The discussion will focus on the circular path set-up with planar detectors and on algorithms currently implemented in our Strange Engine software distribution.

3.1 Cone-beam projections and the 3D Radon transform

3.1.1 The cone-beam geometry

First of all let us analyse the cone-beam geometry that will be considered in the remainder of this thesis. Figure 3.14 depicts a cone-beam X-ray source which exposes a fully 3D object volume described by the function $\mu(x, y, z)$. The cone-beam projections, which will be denoted as $d_c(\beta, p, q)$, are measured by a flat-panel detector with central point \mathbf{D} orthogonal to the \mathbf{SD} segment. The source and detector are assumed to rotate synchronously around the y -axis

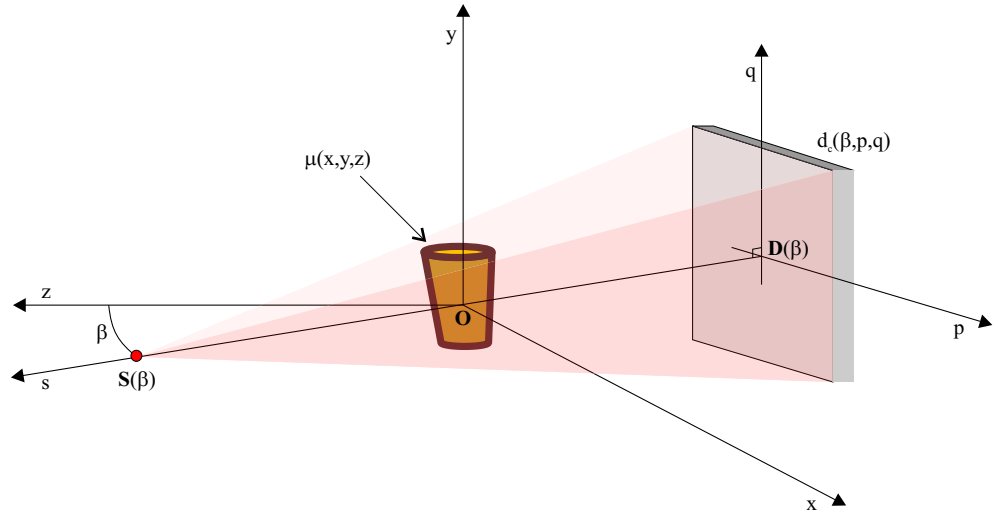


Figure 3.14. Geometry for three-dimensional cone-beam computed tomography.

of the object on circular paths. Let

$$\begin{aligned}\mathbf{S}(\beta) &= R(\sin \beta, 0, \cos \beta) \\ \mathbf{D}(\beta) &= \frac{R-D}{R} \mathbf{S}(\beta)\end{aligned}$$

so that R is the radius of the source circular path and D is the length of the \mathbf{SD} segment. The source angle β has been taken between the positive z -axis and the segment \mathbf{SO} to conform to standard 3D computer graphics geometry conventions. A Cartesian tern defined by the rotated axes p , q and s has also been introduced for convenience. A single pixel \mathbf{Q}_β , on the cone-beam detector at angle β , is selected by the coordinates pair (p, q) while its position is

$$\mathbf{Q}_\beta(p, q) = \mathbf{D}(\beta) + (p \cos \beta, q, -p \sin \beta)$$

In these terms a l -ray is therefore the line passing through some pixel $\mathbf{Q}_\beta(p, q)$ and the X-ray source $\mathbf{S}(\beta)$. That said in this particular geometry we can write the cone-beam projections as:

$$d_c(\beta, p, q) = \int_0^1 \mu(\mathbf{S}(\beta) + \frac{l}{\sqrt{p^2 + q^2 + L^2}} [\mathbf{Q}(\beta) - \mathbf{S}(\beta)]) dl \quad (3.33)$$

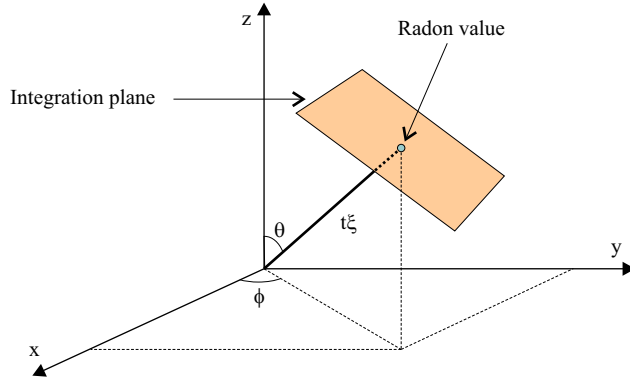


Figure 3.15. The 3D Radon transform at point $t\xi$ is the plane integral of the object function on the l -plane passing perpendicularly through it.

3.1.2 The 3D Radon Transform

Analogously to the two-dimensional case examined in § 2.1.2, we define the three-dimensional Radon transform of $\mu(x, y, z)$ as

$$\mathcal{R}\mu(t, \xi) = \iiint \mu(\mathbf{r}) \delta(\mathbf{r} \cdot \xi - t) d\mathbf{r} \quad (3.34)$$

where in spherical coordinates $\xi = (\sin \theta \cos \phi, \sin \theta \sin \phi, \cos \theta)$. In other words the Radon value at point \mathbf{P} corresponds to the plane integral of the object function along the l -plane passing through it and orthogonal to the distance from origin vector $(\mathbf{P} - \mathbf{O})$. Figure 3.15 shows this situation.

Similarly to the 2D fan-beam geometry, Radon values for a cone-beam projection are positioned on the surface of the Radon sphere or Radon shell. The central section of the Radon shell lying on the source trajectory plane is of course the Radon circle. Given the finite extension of the cone-beam detector, Radon values do in fact cover a limited spherical cap. This particular Radon values positioning forms so called Radon “umbrellas” (see Figure 3.16).

3.1.3 The 3D Fourier Slice Theorem

As in the two-dimensional case it is possible to connect the Radon space with the object space through the Fourier transform. The 3D FST states that the radial Fourier transform of the 3D Radon transform equals to the 3D Fourier transform of the object along the same radial direction. Mathematically this means that

$$\mathcal{F}_t \mathcal{R}\mu(\omega, \xi) = \mathcal{F}_3 \mu(\omega \xi) \quad (3.35)$$

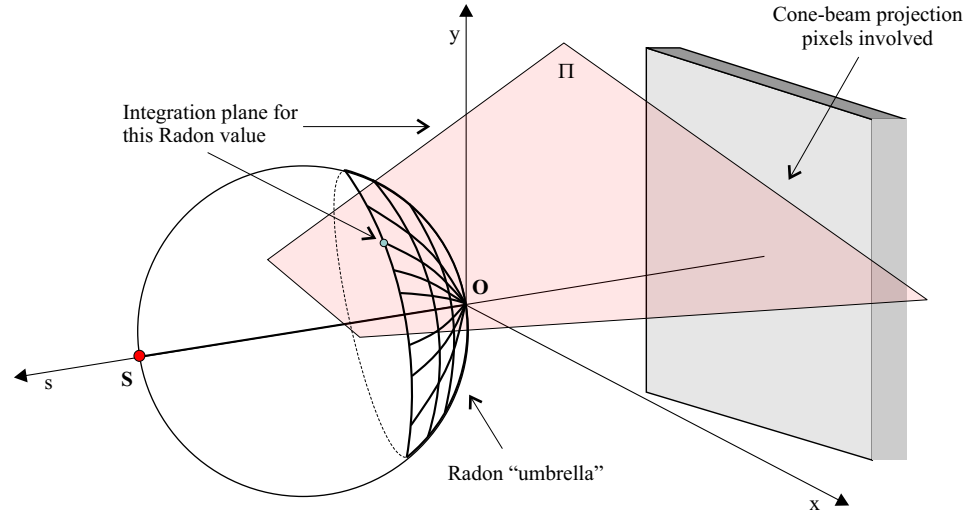


Figure 3.16. 3D Radon transform values for cone-beam projections are positioned on a spherical cap known as a Radon “umbrella”. Each Radon value on the umbrella is the plane integral calculated on the plane passing through it and the source point and orthogonal to the Radon value vector.

Proof. It follows directly from Dirac delta properties and 3D Radon transform and Fourier transform definitions:

$$\begin{aligned}
 \mathcal{F}_t \mathcal{R}\mu(\omega, \xi) &= \iiint \mu(\mathbf{r}) \delta(\mathbf{r} \cdot \xi - t) e^{-2\pi i \omega t} d\mathbf{r} dt \\
 &= \int \mu(\mathbf{r}) d\mathbf{r} \int \delta(\mathbf{r} \cdot \xi - t) e^{-2\pi i \omega t} dt \\
 &= \int \mu(\mathbf{r}) e^{-2\pi i \omega \mathbf{r} \cdot \xi} d\mathbf{r} = \mathcal{F}_3 \mu(\omega \xi)
 \end{aligned}$$

□

3.1.4 The Central Slice Theorem

The 3D Fourier slice theorem connects the 3D Radon transform with the 3D Fourier transform of the projected object. A very noteworthy simplification occurs in the case of 2D parallel-beam projections because in that geometry a single projection is able to fill completely the corresponding Radon plane. In this situation actually the 3D FST reduces to the Central slice theorem which states that the 2D Fourier transform of a 2D parallel-beam projection equals to the central slice of the 3D Fourier transform of the object orthogonal to the beam direction. Unfortunately this is not valid for cone-beam projections.

3.1.5 Plane integrals from cone-beam data

Two-dimensional projections are essentially collections of line integrals in the sense of (2.3). Computation of a plane integral from 2D projections involves basically an integration along the plane direction. This computation is straightforward for 2D parallel-beam projections. The same is not true for projections obtained with divergent beams like cone-beams. Let us have a look at Figure 3.17 in which we have depicted the section of parallel and divergent beams along the integration plane Π . The 3D Radon transform for this plane is

$$\mathcal{R}_{\Pi} = \int \mu(\mathbf{r}) ds dt$$

In the case of the parallel-beam the plane integral is clearly the ray sum

$$\mathcal{R}_{\Pi} = \int d_p(\theta, t) dt$$

We could think that this trick works also for the divergent beam and that the plane integral is given by the ray sum

$$\mathcal{R}_{\Pi} \stackrel{?}{=} \int d_f(\beta, p) d\gamma$$

where γ is the angular aperture of the rays. Unfortunately this is not true. In fact rewriting the plane integral using polar coordinates we obtain

$$\mathcal{R}_{\Pi} = \int \mu(\mathbf{r}) l dl d\gamma$$

which is not equal to the ray sum for the divergent beam because of the presence of the Jacobian factor l . In essence this is the key problem which any cone-beam reconstruction method has to take into account.

3.2 3D Radon transform inversion

Inversion of the 3D Radon transform is possible if the Radon space is adequately sampled. The inversion formula [56] can be written as:

$$\mu(\mathbf{r}) = -\frac{1}{8\pi^2} \int_0^{2\pi} \int_0^{\pi} \frac{\partial^2}{\partial t^2} \mathcal{R}\mu(\mathbf{r} \cdot \boldsymbol{\xi}, \boldsymbol{\xi}) \sin \theta d\theta d\phi \quad (3.36)$$

Direct usage of this inversion relation is not possible in the cone-beam geometry because, as showed in the preceding section, computation of the 3D Radon transform (i.e. plane integrals) is not achievable for divergent beams. We will see how some cone-beam reconstruction algorithms counteract for this situation in the reconstruction part starting from § 3.3.

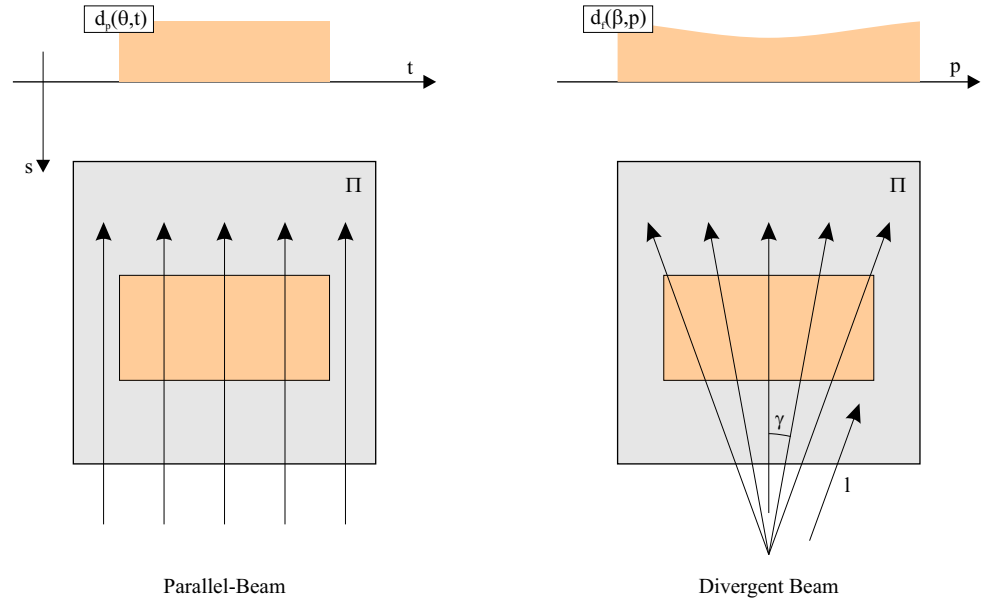


Figure 3.17. Plane integrals for parallel and divergent beams.

3.2.1 Completeness conditions

Before examining reconstruction strategies, let us have a look to another important problem connected to 3D Radon transform inversion: Radon space sampling. The inversion formula (3.36) tells us that if the 3D Radon transform is known for all the planes passing through a neighbour of some point \mathbf{r} then reconstruction of the object function in that point is achievable. The 3D Fourier slice theorem also points out that to perform an exact reconstruction on the object support, say the unit ball, the 3D Radon transform should be known for that space. This fact reverberates on the sampling scheme of the Radon space and in particular on the X-ray source trajectory. A trajectory which achieves complete sampling of the Radon space on the object support is said to fulfil the completeness conditions for exact cone-beam reconstruction.

In the literature the completeness conditions for source trajectory have been proposed in several ways. A good overview of these different formulations is given by Eriksson in [24]. Here we report the condition of Grangeat stated in his ground-breaking work [29]: *every plane intersecting the object support must intersect the source trajectory in at least one point.*

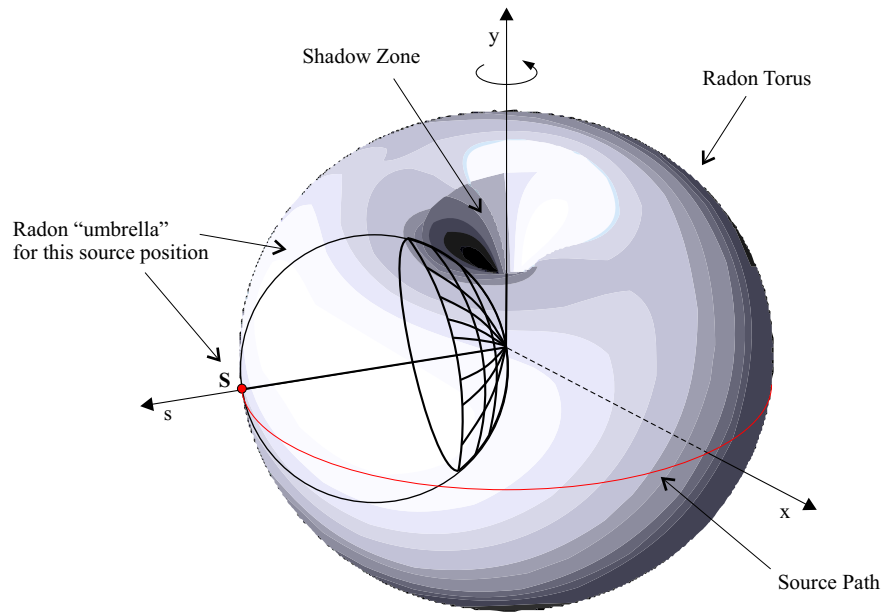


Figure 3.18. For the circular path, Radon values are positioned on the internal surface of the Radius torus, i.e. on the revolution volume generated by the rotation of Radon “umbrellas” around the trajectory axis.

The circular path. The circular path does not fulfil Grangeat completeness condition for source trajectory. Intuitively this is somewhat obvious because at least every plane that intersects the object support but is parallel to the trajectory plane does not intersect the source orbit for sure. Missing Radon data lies in the inner part of a torus. To see this consider that, as showed before, a singular cone-beam projection samples the Radon space on an “umbrella”. As the source moves along the circular path, Radon shells accumulates and form a Radon torus. The Radon space is then sampled inside the revolution volume generated by the rotation of “umbrellas” around source trajectory axis. Figure 3.18 represents this situation.

It is important to note that some volume on the inner part of the torus is not sampled at all, so some Radon data is missing. This volume, whose shape remembers a sort of sandglass, is called the *shadow zone*. The more we move away from the trajectory plane the bigger this volume becomes. This means that reconstruction of off-plane sections cannot be exact because exact Radon inversion is not possible. Moreover we expect reconstruction errors to be bigger for large cone angles as distance from the central section and size of the shadow zone increase.

Other paths. To address circular path deficiency to cover completely Radon space, other source paths have been proposed. Among these: two orthogonal circular paths [42], the line + circle path [91] and the helix path [24, 71], the latter being the most interesting. A general formulation for arbitrary discrete sets of source positions is also available in [58].

3.3 3D reconstruction algorithms overview

Figure 3.19 illustrates a schematic classification of 3D cone-beam reconstruction algorithms. As in the 2D case a first distinction is made for algorithms derived from analytical inversion formulas based on the 3D Radon inversion formula and the 3D Fourier slice theorem and algorithms in which the problem is modelled in discrete terms and solved by an iterative method. Algorithms based on discretization of analytical inversion formulas are then split in algorithms in which the reconstruction is performed through three-dimensional Fourier inversion or Radon inversion (through Grangeat’s fundamental relation) and algorithms based on FBP. Regarding the latter methods we should notice the distinction between “exact” and approximate algorithms. The first ones are essentially derived from the 3D Radon inversion formula; this derivation is a lot more tricky in respect to 2D FBP though. These algorithms are exact because do not introduce any approximation but just use a mathematical inversion formula. Nonetheless we should mind that every cone-beam reconstruction algorithm does not produce exact results if the completeness conditions are not satisfied. Approximate filtered back projection algorithms are on the other hand based on an explicit approximation and therefore contain an intrinsic inexactness. Among approximate methods the Feldkamp-Davis-Kress (FDK) method is without any doubt the most popular.

3.4 Grangeat’s method

Grangeat’s method pursue CBCT reconstruction through direct usage of 3D Radon transform inversion formula. The big issue here is posed by the actual computation of the 3D Radon transform values from cone-beam projections. We have seen in fact that, in the cone-beam geometry, the calculation of plane integrals is not a trivial task. To overcome this problem Radon inversion can be performed not starting from the 3D Radon transform but from some intermediate function [63] which can be connected to it. Let us examine how this can be done presenting in some details Grangeat’s reconstruction method and other sub-variant algorithms in which the intermediate function is the first derivative of the 3D Radon transform.

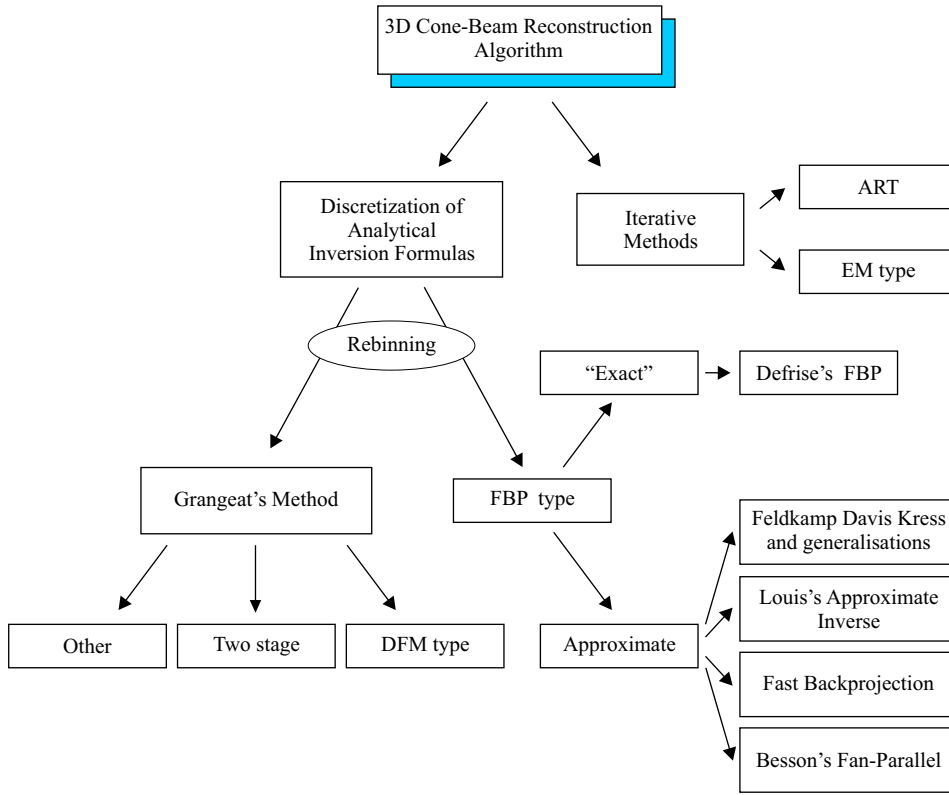


Figure 3.19. Detailed overview of 3D cone-beam reconstruction algorithms.

3.4.1 Grangeat's fundamental relation

Grangeat established in his Ph.D. thesis [29] (see also [28]) an important relation between the cone-beam X-ray transform and the first derivative \mathcal{R}' of the 3D Radon transform. Let us observe Figure 3.20. Consider a certain Radon “characteristic” point $\mathbf{P} = t\xi$ whose value $\mathcal{R}(t, \xi)$ is given by the plane integral on plane Π . Note that Π passes through the source position $\mathbf{S}(\beta)$ because in the cone-beam geometry information on plane integrals and hence Radon values is available only for the planes passing through the source. This also means that t depends on ξ and vice versa, in fact it's easy to see that $t = \mathbf{S}(\beta) \cdot \xi$.

Let $L(\tau, u)$ be the line who represents the intersection of the Π integration plane with the detector plane. This line is described by the angle τ and by the signed distance u from the detector's central point D . Grangeat introduces the function

$$S(\beta, \tau, u) = \int g(\beta, u, v) dv \quad (3.37)$$

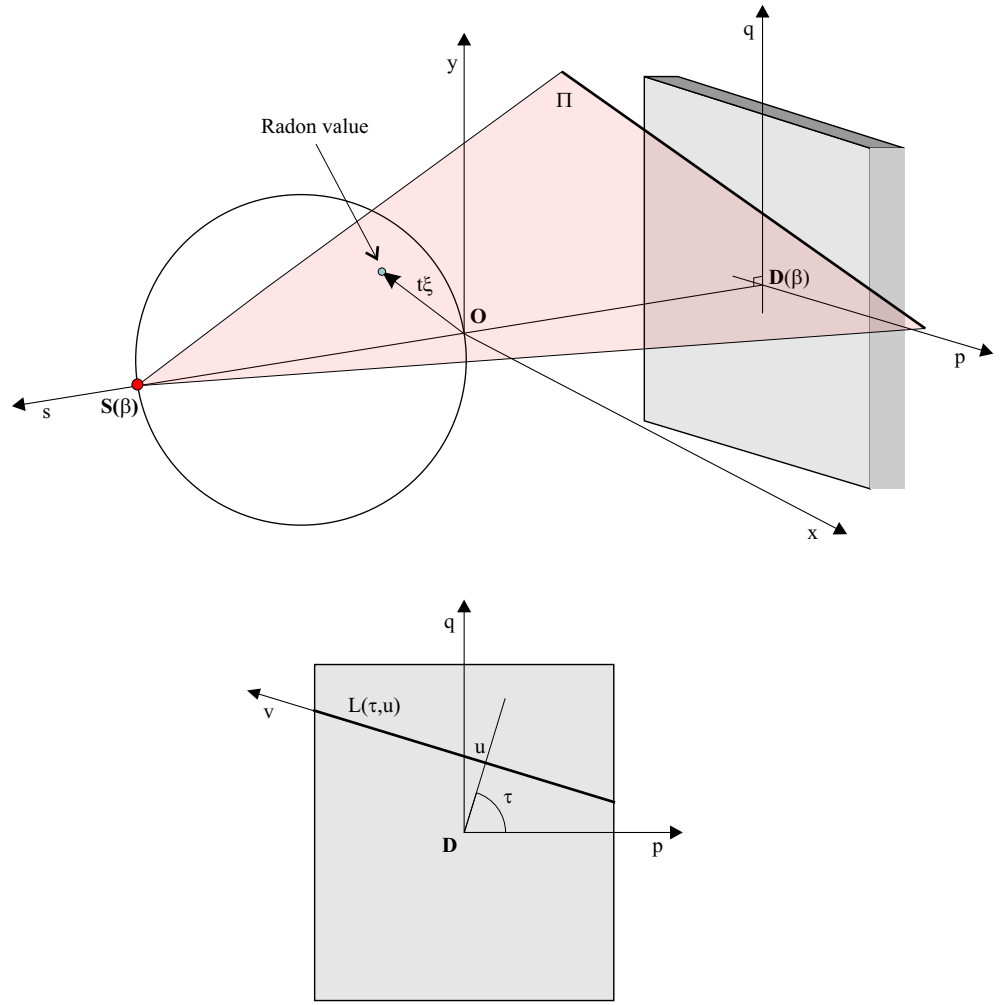


Figure 3.20. Geometric representation for Grangeat's fundamental relation. Top: the Radon value at $t\xi$ is given by the plane integral on the Π plane passing through $t\xi$ and \mathbf{S} and orthogonal to $t\xi$. Bottom: Π intersects the detector on line $L(\tau, u)$.

with

$$g(\beta, u, v) = \frac{D}{\sqrt{D^2 + u^2 + v^2}} d_c(\beta, u \cos \tau - v \sin \tau, u \sin \tau + v \cos \tau) \quad (3.38)$$

The g function represents cone-beam projections weighted by the distance $|\mathbf{S}\mathbf{Q}|$ from the source to individual pixels lying on line $L(\tau, u)$. Therefore $S(\beta, \tau, u)$ is essentially the integral of weighted cone-beam projections on line $L(\tau, u)$.

Grangeat's fundamental relation is written as

$$\mathcal{R}'\mu(t, \boldsymbol{\xi}) = \frac{\partial}{\partial t} \mathcal{R}\mu(t, \boldsymbol{\xi}) = \frac{D^2}{u^2 + D^2} \frac{\partial}{\partial u} S(\beta, \tau, u) \quad (3.39)$$

where u and τ depends on plane orientation $\boldsymbol{\xi}$. For proof of this relation the reader is referred to [28]. In conclusion Grangeat's formula yields the values of the first derivative of the 3D Radon transform corresponding to all the planes passing through the source orbit. Let now examine how this information can be utilised to perform reconstruction.

3.4.2 Radon inversion through the first derivative of the 3D Radon transform

We can rewrite the 3D Radon inversion formula (3.36) in terms of the first derivative \mathcal{R}' as:

$$\begin{aligned} \mu(\mathbf{r}) &= -\frac{1}{8\pi^2} \int_0^{2\pi} \int_0^\pi \frac{\partial^2}{\partial t^2} \mathcal{R}\mu(\mathbf{r} \cdot \boldsymbol{\xi}, \boldsymbol{\xi}) \sin \theta \, d\theta \, d\phi \\ &= -\frac{1}{8\pi^2} \int_0^{2\pi} \int_0^\pi \frac{\partial}{\partial t} \mathcal{R}'\mu(\mathbf{r} \cdot \boldsymbol{\xi}, \boldsymbol{\xi}) \sin \theta \, d\theta \, d\phi \end{aligned} \quad (3.40)$$

This formula is in essence a backprojection in which each reconstruction point is given a value accumulated from the different planes passing through it. In practice cone-beam reconstruction can then be carried out in three steps:

- computation of the first derivative \mathcal{R}' from cone-beam projections using Grangeat's fundamental relation,
- interpolation of \mathcal{R}' values to a spherical coordinates grid and
- backprojection as described by (3.40).

We call this reconstruction method the Grangeat's method.

3.4.3 Computation of the first derivative of the 3D Radon transform

Grangeat's fundamental relation allows to compute the first derivative of the Radon transform from cone-beam projections. This is usually performed in four steps:

- pre-weighting of cone-beam projections with pixel distance $|\bar{\mathbf{S}}\mathbf{Q}|$,
- integration of data along a set of L lines,

- differentiation along the u axis,
- post-weighting with $L^2/(u^2 + L^2)$ factor.

The complexity of this computation is $O(N^4)$ because for each of the N^3 Radon values required for inversion it is necessary to perform a $O(N)$ line integration. It is possible to bring down the theoretical complexity to $O(N^3 \log N)$ using a direct Fourier method or a linogram method as described in [19, 8].

3.4.4 Interpolation.

As explained in § 3.2.1, for each projection, plane integral information and hence \mathcal{R}' values do reside on a Radon “umbrella”. This means that the Radon derivative data obtained using the four steps method sketched above is not suitable for the inversion formula of Grangeat’s method which requires Radon derivative data to be available in spherical coordinates. Therefore an interpolation scheme is necessary. Interpolation is also important to provide proper handling of the shadow zone in the case of source trajectories not satisfying completeness conditions. For further details on how interpolation can be actually carried out we refer to [24] and to [28].

3.4.5 Radon inversion with the two-stage approach.

The last step of Grangeat’s method consists in 3D Radon transform backprojection through inversion formula (3.40). For each reconstruction point it is necessary to perform a radial differentiation and an integration over the surface of the unit ball. Consequently the overall complexity of this formula appears to be very high: $O(N^5)$.

Efficient implementations of the aforementioned inversion formula which reduce the computational effort are available. A popular approach is based on the so called two-stage approach proposed by Marr et al. in [52]. This approach is based on the 3D Fourier slice theorem. First of all let us reconsider the relation (3.36) by introducing the radial Fourier transform of the 3D Radon transform $\hat{R}\mu(\omega, \boldsymbol{\xi})$

$$\begin{aligned} \mu(\mathbf{r}) &= -\frac{1}{8\pi^2} \int_0^{2\pi} \int_0^\pi \frac{\partial^2}{\partial t^2} \mathcal{R}\mu(\mathbf{r} \cdot \boldsymbol{\xi}, \boldsymbol{\xi}) \sin \theta \, d\theta \, d\phi \\ &= \frac{1}{4\pi} \int_0^{2\pi} \int_0^\pi \int \omega^2 \hat{\mathcal{R}}\mu(\omega, \boldsymbol{\xi}) e^{2\pi i \omega(\mathbf{r} \cdot \boldsymbol{\xi})} d\omega \sin \theta \, d\theta \, d\phi \end{aligned}$$

In terms of the first derivative this can be written as

$$\mu(\mathbf{r}) = -\frac{1}{4\pi} \int_0^{2\pi} \int_0^\pi \int i\omega \hat{\mathcal{R}}'\mu(\omega, \boldsymbol{\xi}) e^{2\pi i \omega(\mathbf{r} \cdot \boldsymbol{\xi})} d\omega \sin \theta \, d\theta \, d\phi \quad (3.41)$$

The two-stage approach subdivides this calculation in two stages. Basically the stage one performs the integration over the polar angle θ on meridian planes keeping ϕ constant, while stage two carries out the final azimuthal summation. On the meridian plane Π_ϕ at angle ϕ , let (\tilde{x}, \tilde{y}) be a pair of Cartesian coordinates corresponding to the rotated x and z axes. A point onto this plane can be written as $\mathbf{r} = (\tilde{x} \cos \phi, \tilde{y} \sin \phi, \tilde{y})$. Stage one for this plane is

$$\mu_\phi(\tilde{x}, \tilde{y}) = -\frac{1}{2\pi} \int_0^\pi \int i\omega \hat{\mathcal{R}}\mu(\omega, \boldsymbol{\xi}) e^{2\pi i\omega(\tilde{x} \sin \theta + \tilde{y} \cos \theta)} d\omega \sin \theta d\theta \quad (3.42)$$

which corresponds to a modified 2D filtered backprojection formula. Physically the μ_ϕ integral describes a 3D parallel-beam projection along the direction perpendicular to plane Π_ϕ . By equation (3.41) reconstruction in point \mathbf{r} is accomplished by summing $\mu_\phi(\tilde{x}, \tilde{y})$ contributions with $\tilde{y} = z$ and \tilde{x} corresponding to the orthogonal projection of \mathbf{r} on Π_ϕ plane. Hence stage two is

$$\mu(\mathbf{r}) = \frac{1}{2} \int_0^{2\pi} \mu_\phi(x \cos \phi + y \sin \phi, z) d\phi = \int_0^\pi \mu_\phi(x \cos \phi + y \sin \phi, z) d\phi$$

where we have used the symmetry of parallel-beam projections $\mu_{\pi+\phi} = \mu_\phi$.

The computational complexity of Grangeat's inversion step with the two-stage approach is $O(N^4)$. This is due essentially to stage one in which for each of the N meridian planes a $O(N^3)$ filtered backprojection is necessary. 2D direct Fourier methods or linogram methods can again be used to reduce the theoretical computational complexity even further to $O(N^3 \log N)$ [37, 8, 24].

3.4.6 Radon inversion and 3D Direct Fourier Methods

The strategy followed by volumetric direct Fourier methods is essentially the same followed by 2D DFM in which the reconstruction is achieved using the Fourier slice theorem and Fourier inversion.

Relation (3.41) allows to compute the radial Fourier transform of the object function $\mu(x, y, z)$. Interpolation [72] or gridding [59] techniques are then employed to compute a Cartesian representation of the 3D Fourier transform $\hat{\mu}(\omega_x, \omega_y, \omega_z)$ from the radial Fourier samples. This has to be done with a certain care as discussed in [68, 70]. Reconstruction is finally achieved through 3D Fourier inversion. Computational complexity of 3D direct Fourier methods is $O(N^3 \log N)$. As usual this does not take into account real computer code complexity which can be much greater.

3.5 3D Filtered Back Projection

In comparison to methods based on 3D Radon inversion, e.g. the Grangeat's method with the two-stage approach, 3D FBP can perform reconstruction on a per projection basis. This property makes efficient and fast reconstruction easier to implement and optimise.

In general we can distinguish “exact” and approximate 3D FBP algorithms. “Exact” FBP algorithms rewrite the 3D Radon inversion formula, through usage of Grangeat's fundamental relation, in filtered backprojection form. The 3D FBP reconstruction formula for cone-beam projections was written independently by Defrise and Clack in [20] and by Kudo and Saito in [43]. Unfortunately this formula uses two-dimensional shift-variant filtering which makes it unattractive. Approximate FBP algorithms makes use of explicit approximations to simplify the reconstruction problem and hence speed-up the reconstruction. In the following sections we are going to present some of the most popular approximate 3D FBP reconstruction algorithms.

3.5.1 The Feldkamp-Davis-Kress method

Originally formulated for the circular path by Feldkamp, Davis and Kress in [25] this method has been the most popular cone-beam reconstruction algorithm since its introduction. Generalisations to more complex paths of the FDK method have been proposed by Wang et al. [81] and by Schaller [68] among others. These will not be considered in this discussion.

Let us consider Figure 3.21 in which the source moves along a circular path. The reconstruction on the $x - z$ mid-plane can be performed exactly using the standard fan-beam filtered backprojection formula (2.13)

$$\mu(x, 0, z) = \frac{1}{2} \int_0^{2\pi} \frac{R^2}{(R-s)^2} \int \frac{D}{\sqrt{D^2 + p^2}} d_c(\beta, p, 0) k(t \frac{D}{R-s} - p) dp d\beta$$

where (t, s) refers to the rotated coordinates frame.

The idea of the FDK algorithm is to perform reconstruction for tilted fan-beams with an adaptive fan-beam formula (slightly) modified to take into account the tilt angle ζ . To see how this can be done, let us examine the situation for the point object A of Figure 3.21. For the source position $\mathbf{S}(\beta)$ the ray to be considered in the backprojection belongs to the tilted plane $\Pi_A(\beta)$ which intersects the detector along some horizontal line $q = \text{const}$. Note that as β varies for each projection, the line touched by the ray passing through A goes up and down because the magnification factor varies. It's easy to show that

$$q = \frac{D}{R-s} y$$

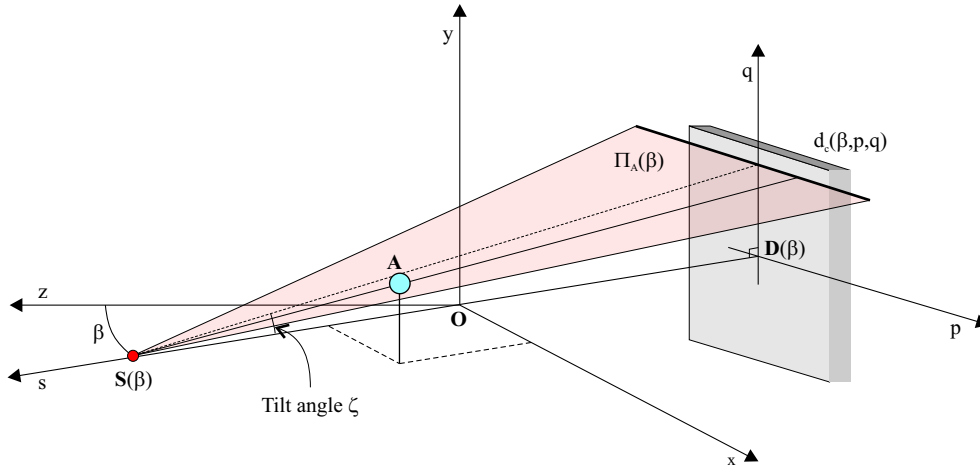


Figure 3.21. The Feldkamp-Davis-Kress reconstruction method makes use of an adaptive fan-beam formula for tilted planes.

We can write the fan-beam reconstruction formula on the $\Pi_A(\beta)$ plane. To do this we can use a trick: scale all the geometrical quantities that appear in the fan-beam formula for the mid-plane (3.5.1) with a factor $\cos \zeta$ which accounts for the mid-plane $\rightarrow \Pi_A$ passage. This means that:

$$\begin{aligned} R &\rightarrow R / \cos \zeta \\ D &\rightarrow D / \cos \zeta \\ s &\rightarrow s / \cos \zeta \\ d\beta &\rightarrow d\beta \times \cos \zeta \end{aligned}$$

where the scale factor $\cos \zeta = D / \sqrt{D^2 + q^2}$ provides compensation for the tilting of the plane. Note that as ζ depends on the angular position β of the source, the reconstruction of A can nonetheless be performed adapting the 2D fan-beam reconstruction formula to each different cone-beam projection. Putting the geometrical scaling rules into (3.5.1) we obtain the Feldkamp-Davis-Kress algorithm

$$\begin{aligned} \mu(x, y, z) = \frac{1}{2} \int_0^{2\pi} \frac{R^2}{(R-s)^2} \int \frac{D}{\sqrt{D^2 + p^2 + q^2}} \cdot \\ \cdot d_c(\beta, p, y \frac{D}{R-s}) k(t \frac{D}{R-s} - p) dp d\beta \end{aligned} \quad (3.43)$$

While this algorithm is exact for any point object like A, it's clearly approximate for more complex objects because it ignores the missing of Radon data

in the shadow zone. However we expect this approximation to be fair for small cone angles because the shadow zone is small compared to the object support. Rizo et al. in [66] have shown FDK reconstruction for some real objects to be reliable for cone apertures inferior to about 10° .

Properties. The FDK method possesses some interesting properties:

- it provides exact reconstruction on the mid-plane of the object, on every other plane the reconstruction error increase with the cone angle,
- it provides exact reconstruction for objects homogeneous in the axial direction,
- the integral of reconstructed values along lines parallel to the rotation axis is exact,
- it has been proven in [28, 20] to be the best approximation of “exact” algorithms for the circular path.

Implementation. The Feldkamp-Davis-Kress algorithm is usually implemented in three consecutive steps (see also Figure 3.22):

1. cone-beam projections are pre-weighted

$$g(\beta, p, q) = d_c(\beta, p, q) \times \frac{D}{\sqrt{D^2 + p^2 + q^2}}$$

2. filtered row-by-row using a one-dimensional kernel function $k(p)$

$$\tilde{d}_c(\beta, p, q) = \int g(\beta, p_1, q) k(p - p_1) dp_1$$

3. and backprojected

$$\mu(x, y, z) = \frac{1}{2} \int_0^{2\pi} \frac{R^2}{(R - s)^2} \tilde{d}_c(\beta, \frac{D}{R - s} t, \frac{D}{R - s} y) d\beta$$

with $t = x \cos \beta - z \sin \beta$ and $s = x \sin \beta + z \cos \beta$.

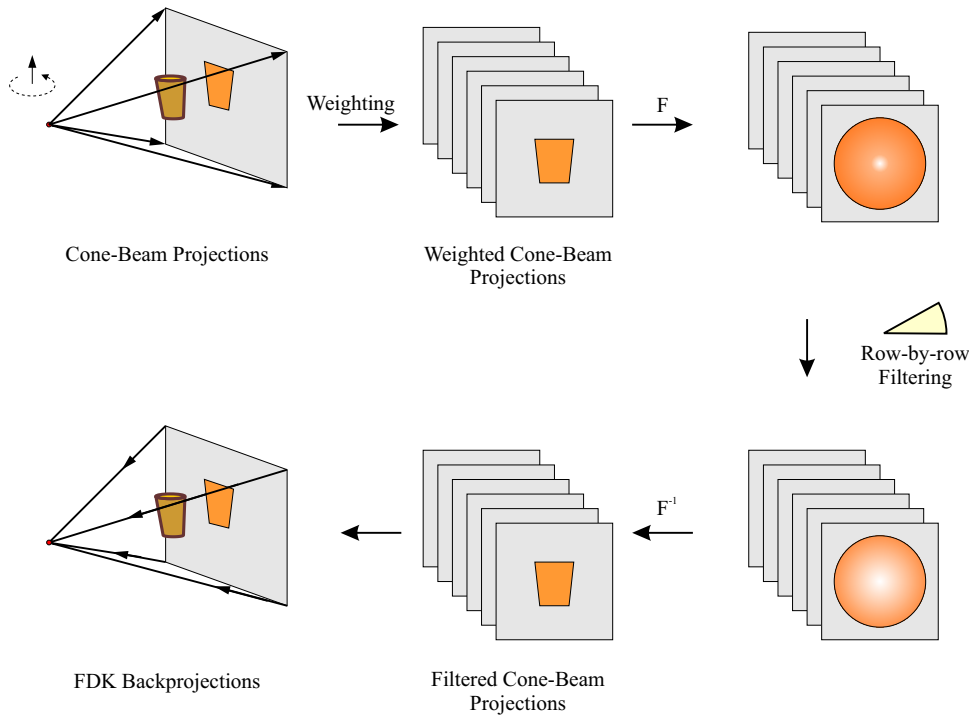


Figure 3.22. Steps of the Feldkamp-Davis-Kress reconstruction method. Cone-beam projections are pre-weighted by a pixel distance factor, filtered (in Fourier space by ramp filter) and backprojected. Reconstruction is per projection.

Notes. We would like to add a couple of comments regarding Feldkamp-Davis-Kress reconstruction:

- filtering is done row-by-row. Any filter used for fan-beam reconstruction is suitable. Popular choices are the ramp filter with or without apodization window and the Shepp-Logan filter (see § 2.4.3). Alternative approaches for filter design, based on wavelets [14, 90] and approximate inverse (more about this in § 3.8), are also available.
- as in the 2D case a pixel dependent $1/r^2$ factor is present in the calculation of the final backprojection step.
- the computational complexity is $O(N^4)$ but its structure makes it easy to optimise its implementation and achieve fast reconstruction.

3.6 Rebinned 3D FBP

One way to speed-up Felkamp-Davis-Kress reconstruction is to get rid of the pixel dependent $1/r^2$ factor appearing in the backprojection formula. This is made possible by the rebinning of cone-beam projections into so called oblique-parallel projections or into oblique fan-parallel projections.

3.6.1 Oblique-parallel rebinned 3D FDK

An oblique-parallel projection [69, 78, 30] is a two-dimensional projection in which each row corresponds to a tilted planar parallel-beam projection. The rebinning of cone-beam projections into oblique-parallel projections for the circular path is illustrated in Figure 3.23 in which a set of cone-beam projections $d_c(\beta, p, q)$ is used to produce an oblique-parallel projection $d_{op}(\beta, p, q)$ rising from a virtual X-ray source. To this end the fan-beam to parallel-beam rebinning equation (2.22) is applied to each projection row. The cone-beam to oblique-parallel beam rebinning $d_c(\beta, p, q) \rightarrow d_{op}(\bar{\beta}, \bar{p}, \bar{q})$ operation can therefore be written as:

$$\begin{aligned}\beta &= \bar{\beta} + \arcsin \frac{\bar{p}}{R} \\ p &= D \tan(\arcsin \frac{\bar{p}}{R}) \\ q &= \bar{q} \sqrt{\frac{D^2 + p^2}{R^2 - \bar{p}^2}}\end{aligned}\tag{3.44}$$

Careful interpolation (both azimuthal and on detector pixels) is required. After rebinning reconstruction can be performed. Following the same reasoning as for the non-rebinned FDK algorithm we can deduce the oblique-parallel rebinned Feldkamp-Davis-Kress algorithm:

$$\mu(x, y, z) = \frac{1}{2} \int_0^{2\pi} \int \frac{D}{\sqrt{D^2 + q^2}} d_{op}(\beta, p, y \frac{R_t}{R_t - s}) k(t - p) dp d\beta \tag{3.45}$$

where $R_t = \sqrt{R^2 - t^2}$. Note that while a pre-weight factor is still there, the computational expensive pixel dependent $1/r^2$ factor is gone. Obviously the trade-off is the rebinning operation but this in general is not a serious issue because it can be performed during the acquisition of cone-beam projections.

3.6.2 Besson's fan-parallel extension to 3D cone-beam

An extension to the 3D cone-beam case of the original fan-parallel reconstruction formula has been proposed by Besson in [11]. Basically this extension is

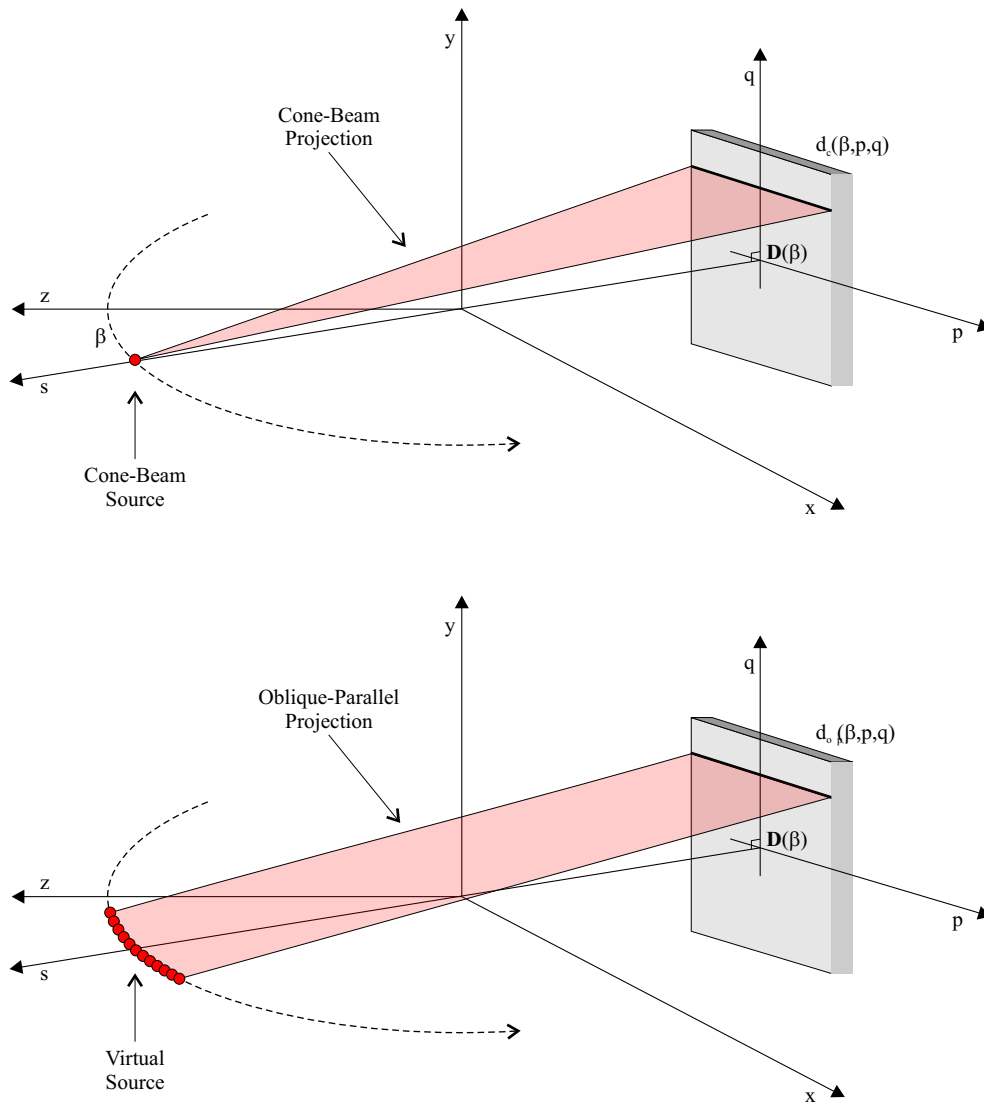


Figure 3.23. Rebinning of cone-beam projection into oblique-parallel projections.

based on an hybrid between fan-beam and parallel-beam FDK reconstruction. Each fan-parallel projection is obtained from a set of cone-beam projections through azimuthal interpolation only, avoiding the resolution loss implicit in the radial interpolation process required in the cone-beam to oblique-parallel rebinning.

3.7 3D Fast Back Projection

Turbell and Danielsson have recently proposed in [78, 79] a modified Feldkamp-Davis-Kress reconstruction algorithm which performs the backprojection in $O(N^3 \log N)$ steps instead of the $O(N^4)$ steps required for the traditional backprojection. This algorithm is the 3D extension of the two-dimensional fast filtered backprojection method presented in § 2.6. It requires oblique-parallel rebinning and a number of projections power of 2 to be efficient. Another disadvantage is that it requires an additional $O(N^3)$ matrix to store summation links and that it's not a per projection algorithm. A detailed analysis including real world performance tests is available in Linköping Ph.D. thesis [78, 36].

3.8 3D Approximate Inverse

The approximate inverse framework presented in § 2.7 can be exploited to design improved filtering strategies for 3D filtered backprojection algorithms. For 3D parallel-beam geometry the approximate inverse is written as:

$$f_\gamma(x, y, z) = \int \int d_p(\theta, t, y) k_\gamma(z \cos \theta + x \sin \theta - t, y) dt d\theta \quad (3.46)$$

where as usual we have assumed that rotation is around the y -axis and that we deal with a translation and rotation invariant mollifier $\bar{e}_\gamma(x, y, z)$. $k_\gamma(t, q)$ is the reconstruction kernel corresponding to the parallel-projection of the mollifier. Note that k_γ is 2D. Like in the two-dimensional case the reconstruction kernel can be computed from a truncated series of a suitable SVD or by using the explicit filtered backprojection formula based on Riesz potential.

3.8.1 Reconstruction formula for the cone-beam geometry

As for the 2D fan-beam geometry, the problem of adapting the approximate inverse scheme to the cone-beam geometry resides in the space-variant magnification factor to be applied to the mollified representation of the object to be reconstructed. For small cone angles Dietz has proven in [22] that a good approximation for the circular path is

$$\tilde{f}_\gamma(\mathbf{r}) = \frac{(R-1)^2}{8} \int_0^{2\pi} \frac{1}{|\mathbf{r} - \mathbf{S}(\beta)|^2} \tilde{d}_c(\beta, p, q) d\beta \quad (3.47)$$

where p and q are the projection coordinates for \mathbf{r} ($p = \frac{D}{R-s} t$, $q = \frac{D}{R-s} y$, $t = x \cos \beta - z \sin \beta$ and $s = x \sin \beta + z \cos \beta$) and where \tilde{d}_c denotes 2D filtered cone-beam projections

$$\tilde{d}_c(\beta, p, q) = \iint d_c(\beta, p_1, q_1) \tilde{k}_\gamma(p - p_1, q - q_1) dp_1 dq_1 \quad (3.48)$$

Here $\tilde{k}_\gamma(p, q)$ is the approximate reconstruction kernel given by

$$\begin{aligned} \tilde{k}_\gamma(p, q) = \tilde{k}(\boldsymbol{\theta}) = & -h \mathbf{S}_0 \cdot \boldsymbol{\theta} \int_{\boldsymbol{\theta}^\perp} \psi_\gamma(h \mathbf{S}_0 \cdot \boldsymbol{\omega}) |\mathbf{S}'_0 \cdot \boldsymbol{\omega}| d\boldsymbol{\omega} - \\ & - \mathbf{S}'_0 \cdot \boldsymbol{\theta} \int_{\boldsymbol{\theta}^\perp} \Psi_\gamma(h \mathbf{S}_0 \cdot \boldsymbol{\omega}) \text{sign}(\mathbf{S}'_0 \cdot \boldsymbol{\omega}) d\boldsymbol{\omega} \end{aligned} \quad (3.49)$$

with

$\boldsymbol{\theta}$ the vector connecting the X-ray source to pixel $\mathbf{Q} = (p, q)$,

$h = (R - 1)/(2R)$,

$\mathbf{S}_0 = \mathbf{S}(\beta = 0)$,

\mathbf{S}'_0 the tangent to the source path for $\beta = 0$

and with ψ_γ and Ψ_γ defined as

$$\begin{aligned} \psi_\gamma(t) &= \text{reconstruction kernel for } \bar{e}_\gamma \\ \Psi_\gamma(t) &= \int_0^t \psi_\gamma(t_1) dt_1 \end{aligned}$$

where it was assumed that the mollifier \bar{e}_γ is also dilation invariant.

In conclusion the approximate inverse scheme leads to a filtered back projection algorithm. In contrast to the Feldkamp-Davis-Kress method, filtering is two-dimensional and shift-variant. However it is possible to make it shift-invariant by introducing some approximation in the computation of the reconstruction kernel. Approximation errors are expected to be small for small cone angles.

Example : 3D Gaussian mollifier. Let us consider the family of three-dimensional Gaussian mollifiers

$$e_\gamma^g(\mathbf{x}) = \frac{1}{(2\pi\gamma^2)^{3/2}} e^{-|\mathbf{x}|^2/2\gamma^2}$$

It has been showed by Dietz in [22] that:

$$\psi_\gamma(t) = \frac{\gamma^2 - t^2}{2(2\pi\gamma^2)^{5/2}} e^{-t^2/2\gamma^2}$$

The integral of ψ_γ has a closed expression too. It's easy to see that

$$\Psi_\gamma(t) = \frac{\gamma^2 t}{2(2\pi\gamma^2)^{5/2}} e^{-t^2/2\gamma^2}$$

3.9 3D Iterative Methods

Three-dimensional iterative methods are based on the same estimation-update concept we have encountered in the 2D iterative methods analysis. Slowness of reconstruction was the main problem for two-dimensional iterative methods and obviously this same problem is even worse for volumetric reconstruction. Nonetheless interest in 3D iterative methods is increasing. ART, EM, TRML and even Monte Carlo based methods are available for 3D reconstruction. Look to § 2.8 for some interesting references covering this topic.

The Strange Engine toolkit

The tentative state of cone-beam computerized tomography requires advanced software tools to lead the research and development of proprietary reconstruction algorithms. The common approach to address this issue, is to write small in-house software routines which tailor specific research needs. To speed-up the development, usually these routines are written with a procedural programming language, e.g. the C language; public domain libraries and/or commercial numerical software environments can then be used to glue them together and to display results. This approach is seriously flawed because it produces monolithic software pieces which are difficult to reuse and maintain.

Enter now *Strange Engine*, our dedicated software toolkit which provides a comfortable and user friendly common ground for research and development of cone-beam computed tomography reconstruction algorithms. Strange Engine allows algorithm developers (the users) to focus on their real job, leaving all the other duties to the toolkit. Strange Engine in fact offers everything needed for the testing of proprietary reconstruction code; this includes:

- management, analysis and simulation of cone-beam projections,
- management of software phantoms and of 3D volumes,
- reconstruction plug-in ActiveX facility,
- uniform data input and output from and to storage devices,
- import and export of data from and to industry standard formats with real-time frame grabbing for video based devices.

Strange Engine is modern and fully object-oriented: it has been written in the C++ language and provides a problem domain abstraction through a component driven interface specified using state-of-the-art software technologies.

Strange Engine is configurable and expandable: its component driven interface allows users to add new custom reconstruction plug-ins.

Strange Engine is easy to use and visually appealing: it features a full-fledged Microsoft Windows 2000 graphical user interface with drag & drop support.

4.1 Toolkit design

Strange Engine was born by our necessity to write and test, on the Microsoft Windows platform, cone-beam CT reconstruction algorithms. We have found that the development of a reconstruction algorithm is not a linear process, but more like a parallel pipeline in which particular tasks have to be repeated in different stages concurrently. Nonetheless algorithm development goes through some distinct phases of advancement:

Phase 1. The initial task is to assess that the reconstruction algorithm is well-coded (no programming errors) and that it really produces data resembling an object reconstruction. In this phase the user basically needs to check his code using some cone-beam projections test data and see if it does a good reconstruction job. Usually test data is supplied by a cone-beam projections simulator which computes cone-beam projections starting from an analytical description of a test object known as software phantom.

Phase 2. Once the reconstruction algorithm code has been debugged and tested against reconstructions of some simple software phantom it can enter phase 2 in which more in-depth analysis has to be performed. This includes some reconstruction tests for both complex software phantoms and real-world data and a comparison with other trusted reconstruction algorithm codes.

Phase 3. Validated reconstruction algorithm codes can finally be benchmarked and optimised against competing codes.

From this analysis it is clear that in order to test a cone-beam computed tomography reconstruction algorithm code the user needs a series of ancillary tools. The minimum set-up involves a data displayer to visualize cone-beam projections and 3D volume data, a cone-beam projections simulator with software phantoms and a trusted reconstruction code. All these tools have to make

input and output of data in the same digital format. Regarding this point, we should bear in mind that while usage of simple data formats like raw binary is trivial, it leads to hard-coding of reconstruction parameters. In addition import and export of data to industry standard formats is a definite must as well as the capability to pre- and post-process data. Strange Engine addresses all these issues offering a modern solution based on object-oriented analysis.

4.1.1 A bit of history

The current release of Strange Engine is version 1.0a build 12 distribution dated December 14, 2000. Before examining actual Strange Engine's architecture, let us briefly review its development history which goes back to 1998 fall.

Strange Engine started as a personal tool for the development and test of cone-beam computed tomography reconstruction algorithm code on Microsoft Windows NT platform. In 1999 spring an early internal version was used for [2]. This early version was severely limited and contained only built-in code for Feldkamp-Davis-Kress reconstruction. The first public release of Strange Engine (build 6) was released on September 1999. This version added lots of important features: ActiveX reconstruction plug-in support, import and export to foreign data formats and refined image display. Build 7 won the 3rd prize for the communication "*3D Tomographic Reconstruction ActiveX Style*" [3] given for the ACM International Graduate Student Contest held in Austin, May 2000. Later builds added an improved ActiveX component driven interface, additional off-the-shelf reconstruction plug-ins and some minor new display feature.

4.1.2 Strange Engine constraints

Strange Engine is very versatile nonetheless some constraint does exist. During the design phase we decided to introduce some initial assumption that as the development progresses could be removed. Currently the constraints are:

1. cone-beam projections are assumed to be taken on a circular path,
2. cone-beam projections are assumed to be planar,

so that the cone-beam set-up is the one we already inspected in Figure 3.14. We would like to point out again that Strange Engine is an open project. New builds can quite easily extend the current architecture to address specific needs like e.g. reconstruction for the helical path, curved and other not-planar detectors, 2D reconstruction and so on. This is made possible by the toolkit's modern object-oriented design we are going to examine next.

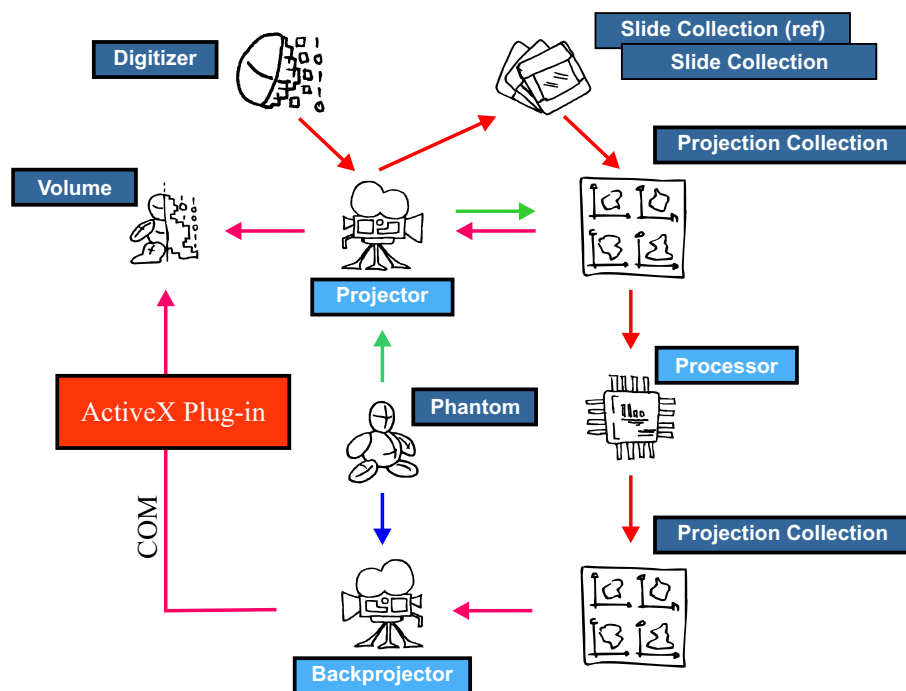


Figure 4.24. Strange Engine object framework.

4.1.3 The object-oriented point of view

A serious rethinking of cone-beam reconstruction algorithm development can effectively be made using modern object-oriented techniques. The result of our analysis of the problem domain led to the object abstraction (framework) depicted in Figure 4.24. In the Strange Engine framework each development task have been subdivided into “atoms” called objects. The object abstraction has been inspired by component-driven object-oriented programming languages such as Smalltalk, Component Object Model (COM) and Java. In the following sections we will examine in details the framework’s main features and goals.

Data- and op-objects. In the Strange Engine framework an object can describe a data entity or a data operator. We will call these two kinds of objects data-objects and op-objects respectively. Data-objects contain raw image data and every parameter necessary to describe what the raw image data refers to. For example a set of cone-beam projections is represented by an instance of the “Projection Collection” data-object which contains projection images and all the information needed to explain in what situation these images were taken.

Op-objects on the other hand do not contain any image data but are merely data operators which manipulates data entities to perform specific tasks. An example of this is the “Backprojector” op-object: it encapsulates a reconstruction algorithm which produces volume data from cone-beam projections.

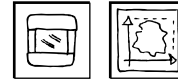
Data-objects list. In the current Strange Engine 1.0a build 12 distribution four different data-objects are defined:

Phantom



A Phantom represents an analytical 3D software phantom composed by ellipsoids and/or truncated cylinders (phantom elements). Each element is characterised by a signed attenuation coefficient $\pm\mu$. The size, position and orientation of elements in space is specified through appropriate shift, scaling and rotation (with Euler angles) transformations. To simplify the design of phantom with “holes” at element intersections the signed attenuation coefficient is summed.

Surface

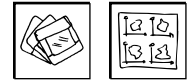


A Surface encapsulates 2D image pixel data and the geometry for the image plane. This includes dimensions, position and orientation of the image plane, number of rows, number of columns and depth of pixel data. Each pixel is represented by a floating-point value. Three surface child objects are implemented:

- Slices,
- Projections,
- Slides.

Slices are sections of volumetric data. Projections are cone-beam projections in the sense of (3.33). Slides are essentially cone-beam intensity projections in which each pixel value measures the X-ray intensity hitting the pixel. Contrary to projections each pixel value is represented by an integer; this is useful to spare memory in low dynamic range situations. A slide can be converted into a projection by using a reference slide and the Lambert-Beer’s relation (2.3).

Surface Collection



A Surface Collection represents a set of different surfaces. Two kinds of collections are available: in-place and linked. In-place collections store surfaces one after the other in a single big memory space. Linked collections on the other hand store only links to surface files residing in storage space. Strange Engine provides a built-in garbage manager which allows to load, restore and offload single surfaces from the system memory pool. This makes possible saving of memory and speed-up of surface processing.

Volume



A Volume encapsulates 3D voxel data and the geometry for the voxel cube. This includes dimensions of the voxel cube, number of planes, number of rows, number of columns and depth of voxel data. Each voxel is represented by a floating-point value. Starting from a volume, a slice collection representation can be generated.

Op-objects list. In the current Strange Engine 1.0a build 12 distribution four different op-objects are defined:

Projector



A Projector serves a double function. It can generate an in-place or linked projection collection by ray-tracing a phantom or work in conjunction with a Digitizer to grab a slide collection from a video source (more on this later). The phantom ray-tracer makes use of a single ray per detector pixel, which connects the X-ray source to the pixel centre. Optional anti-alias is also available by adding four extra rays (each pixel is the average of a pyramidal beam).

Backprojector



A Backprojector acts like a front-end for cone-beam reconstruction code provided by the user. The reconstruction code has to be encapsulated by the user into an external ActiveX plug-in. Communication between Strange Engine backprojector and the user written plug-in is granted by two Strange Engine custom COM interfaces IBackProjector and IBackProjectorRenderer (more on this later). A second (optional) service which can be offered by user written ActiveX plug-ins is voxel synthesis of phantoms. Voxel synthesis refers to voxel sampling of the intrinsic analytical representation of phantoms. This is again done through other two custom COM interfaces called IPhantom and IPhantomElement.

Processor



A Processor is a simple pre-processing tool for projection collections. It provides three basic image processing functions:

- shift,
- rotate and
- spherize.

These functions are sometimes useful to correct image distortions present in projections grabbed from video devices (e.g. X-ray detectors based on a camera + image intensifier set-up).

Digitizer



A Digitizer works in conjunction with a projector and a Matrox Genesis advanced digitizer card. Its job is to grab a linked slide collection from a video source connected to the Genesis card and save it to disk in real-time. Grabbed slides can afterwards be converted into projections by using a reference slide collection and the Lambert-Beer's relation (2.3).

4.2 Toolkit user interface

Strange Engine is not just a software toolkit, it's a visual software toolkit. Strange Engine's objects are visualised through a standard Microsoft Windows Graphical User Interface (GUI) which provides:

- object persistency,
- user interaction and
- object-to-object interaction.

4.2.1 Object persistency

Every Strange Engine's object can be saved and loaded to and from storage devices in proprietary file formats. Lossless binary compression is used, whenever it makes sense, to spare disk space. Import and export (some restrictions apply) to the following industry standard formats is also available:

- raw binary,
- Windows Bitmap (BMP),
- Windows Video (AVI),
- NCSA² HDF³,
- ACR/NEMA⁴ DICOM⁵ 3.

4.2.2 User interaction

Strange Engine acts like a container. At start it looks like an empty space ready to be populated with Strange Engine's objects. Each object sports a full fledged Microsoft Windows 2000 graphical user interface. In Figure 4.25 for example Strange Engine's multi document dialog is used to create a backprojector object which interacts with the user through a panel window.

²National Center for Supercomputing Applications.

³Hierarchical Data Format. <http://hdf.ncsa.uiuc.edu>.

⁴American College of Radiology/National Electrical Manufacturers Association.

⁵Digital Imaging and Communications in Medicine. <http://www.xray.hmc.psu.edu>.

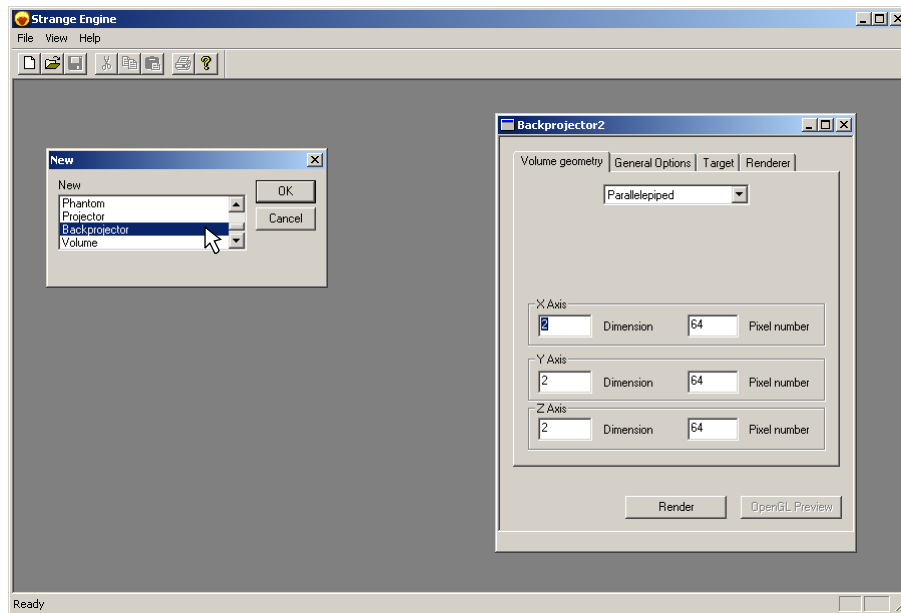


Figure 4.25. At start Strange Engine looks like an empty container ready to be populated with objects. Big window: Strange Engine application. Left window: the multi document dialog. Right window: a backprojector object.

4.2.3 Object-to-object interaction

An essential feature of Strange Engine is object-to-object interaction. We have seen that op-objects perform specific actions on data-objects. This means that the user creates a new op-object and then ask it to act on a input data-object. The input data-object is modified (this is the case of the processor object) or used to create another output data-object: e.g. the backprojector object takes a projection collection object and produce a volume object. An exception to this rule is the digitizer object which does not require an input data-object but works in conjunction with a projector object.

Object-to-object interaction is performed by the user through Microsoft Windows drag & drop. Op-objects and data-objects can be reduced to icons. Op-objects have “hot spots” on which the user can drop a data-object (or op-object) icon. Every “hot spot” is sensitive only to appropriate objects and rejects all the others (see Figure 4.26).

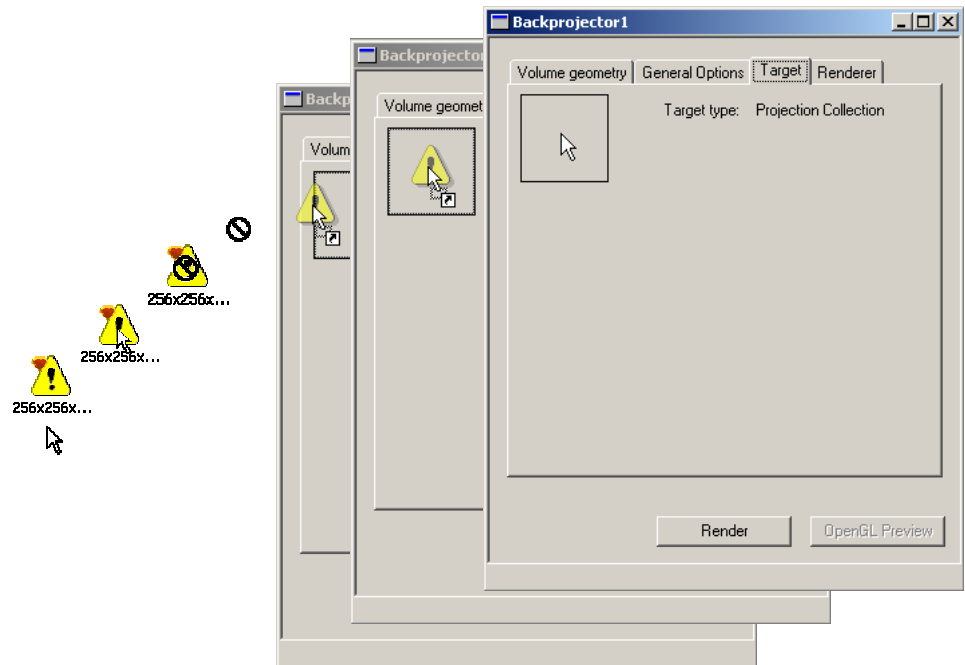


Figure 4.26. Object-to-object interaction is performed through Windows drag & drop. In this example the icon on the left represents a projection collection which is dragged over a backprojector object and dropped on its hot spot. The hot spot recognizes the dragged object to be a projection collection and accept it for drop.

4.3 The ActiveX edge

The crucial feature of the Strange Engine toolkit resides in its capability to fit user-written reconstruction code. Basically this works in the following way. The user encapsulates his reconstruction code in a sort of black box, a so called plug-in, which plugs into Strange Engine main application. The plug-in has to follow some programming specification which are dubbed by Microsoft as ActiveX. Strange Engine receives and sends “messages” from and to ActiveX plug-ins according to user input. This communication happens through documented COM interfaces (for technical details see references cited in § 4.5) specified in Strange Engine’s mini Application Programming Interface (API). In addition to user-written reconstruction code, each plug-in can also contain (optional) user interface fragments. This is very useful for example to expose to the end user plug-in specific reconstruction options and parameters.

4.3.1 Strange Engine's COM interfaces.

The Strange Engine's mini Application Programming Interface defines 8 simple COM interfaces:

- IBackProjector,
- IBackProjectorRenderer,
- IPantom,
- IPantomElement,
- IProjectionCollection,
- IListElementStub,
- IProjection,
- IVolume.

Figure 4.27 describes the COM mediated interactions of Strange Engine objects with a reconstruction ActiveX plug-in. Let us examine how this happens.

IBackProjector and IBackProjectorRenderer are the two primary interfaces. A projector object exposes the IBackProjector interface to a plug-in. The plug-in exposes the IBackProjectorRenderer interface to the projector. Note that the ActiveX plug-in is contained within the projector object.

The IBackProjector methods SetComputation() and StepComputation() are used by the plug-in to inform the projector object that reconstruction is proceeding well, so that the projector can refresh its graphical user interface and compute simple estimates of time required for each reconstruction step.

The plug-in must implement IBackProjectorRenderer interface. The three methods BackProjectPhantom(), BackProjectProjection() and the optional BackProjectProjectionCollection() are invoked by the plug-in projector container to fill voxel data of a given volume object which represents the reconstruction region. BackProjectPhantom() should provide a voxel representation of a phantom, while the other two methods should provide cone-beam reconstruction obtained from a set of projections. The method invoked depends obviously by user input. The difference between BackProjectProjection() and BackProjectProjectionCollection() is this: Strange Engine first of all tries to invoke the latter which directly operates on a projection collection object. If BackProjectProjectionCollection() is not implemented (it can because it is declared as optional in Strange Engine's mini API), the mandatory method BackProjectProjection(), which operates on a per projection basis, is invoked.

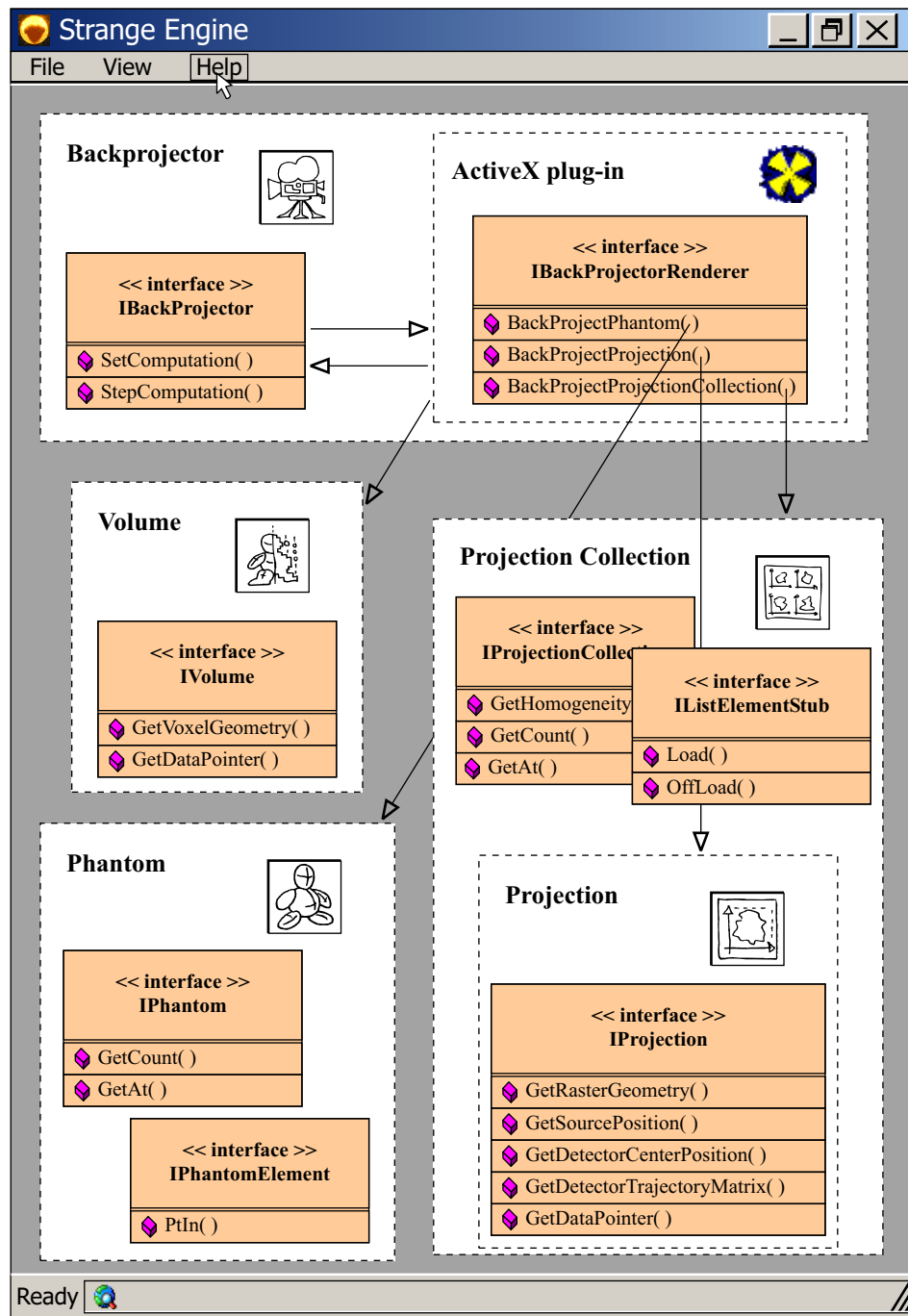


Figure 4.27. Strange Engine's COM interfaces.

IPhantom and IPhantomElement are used by plug-ins to access phantom data. Each phantom contains a set of phantom elements which the plug-in can enumerate through the `GetCount()` and the `GetAt()` methods. Phantom elements properties are then read through the `PtIn()` method.

IProjectionCollection and IListElementStub are used by plug-ins to access projection data residing in a projection collection. Each projection collection contains a set of projection “stub” objects which the plug-in can enumerate through the `GetCount()` and the `GetAt()` methods. Stubs are sort of projection surrogates which mediates loading and unloading of linked projections residing in disk space through methods `Load()` and `OffLoad()`. The `GetHomogeneity()` method can be used by the plug-in to check if all the projections contained in the collection, were taken by the same detector (same detector plane geometry and same number of pixels) so that caching of some reconstruction intermediate quantity is possible.

IProjection describes a Strange Engine’s projection object. This includes the detector plane geometry (`GetRasterGeometry()` method), source and detector position (`GetSourcePosition()` and `GetDetectorCenterPosition()` methods), detector orientation (`GetTrajectoryMatrix()` method) and image pixel data (`GetDataPointer()` method).

IVolume describes a Strange Engine’s volume object. This includes volume dimensions, number of voxels (`GetVoxelGeometry()` method) and voxel data (`GetDataPointer()` method).

4.3.2 Plug-in request and plug-in user interface

When a Strange Engine’s backprojector object requests a plug-in, the system ActiveX list is scanned and available Strange Engine plug-ins are shown. The selected plug-in is then loaded and injected into Strange Engine. In addition to implementation code related to COM `IBackProjectorRenderer` interface, each ActiveX plug-in can contain user interface fragments. These fragments can be used by the end user to modify plug-in specific reconstruction options and parameters, trigger plug-in actions, etc.... Fragments are organized in panel windows which are appended (see Figure 4.28) to the panels of the Strange Engine backprojector object requesting the plug-in.

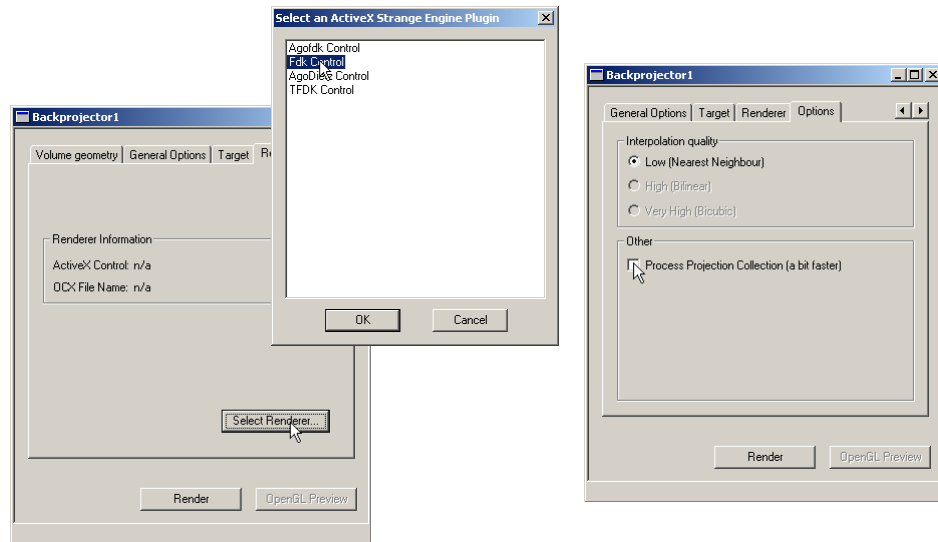


Figure 4.28. On backprojector request (left window) installed Strange Engine plug-ins are shown (mid window). The selected plug-in is loaded and injected into Strange Engine main application. If available, plug-in's user interface fragments are appended to the backprojector object GUI (right window).

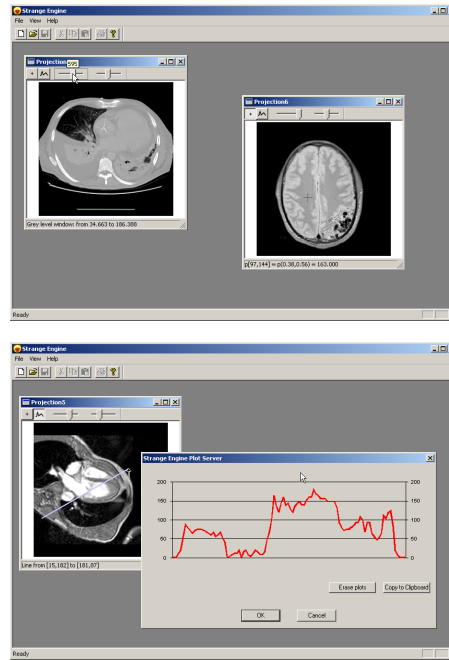
4.4 Toolkit usage

Till now we have examined Strange Engine by the developer point of view. The end-user is not required to know every single detail of Strange Engine API to use it as every other Windows application. To see this in practice let us look at three different examples of Strange Engine usage:

- Data display
 - visualise image data in full grey shades with windowing,
 - analyse image data pixel values and produce line profile plots.
- Acquisition and reconstruction
 - grab slides from a video source in real-time,
 - use an ActiveX plug-in to perform reconstruction.
- Algorithm test
 - assemble test phantoms and produce simulated projections,
 - test user written algorithm on simulated projections,
 - compare results with voxel representation of the test phantom.

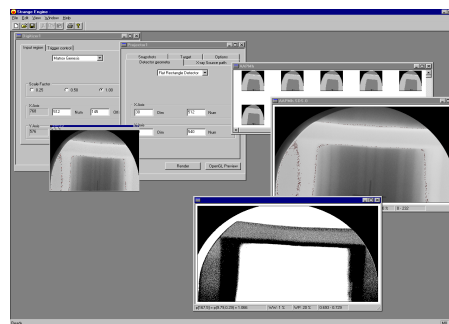
4.4.1 Data display

This is the most basic usage of the Strange Engine toolkit. The user can load from a storage device any previously saved object, e.g. a projection, or import a raw binary matrix into a new data-object. As depicted in Figure 4.29 (see figure cameo on the right and float pages) image data is visualised with 256 grey shades. Windowing (position and width) of data is available through two toolbar cursors. Other image functions include 8 levels of zoom and report of pixel values (pointed by cursor, image minimum and maximum). Generation of a line profile plot with optional export to Excel spreadsheet text format is also provided.



4.4.2 Acquisition and reconstruction

At start Strange Engine checks the system to find an installed Matrox Genesis digitiser card, if found the user can create a digitizer object. A digitizer object provides controls to define the video input region, scale and trigger options. A live preview window (see Figure 4.30) is opened to show to the user what it's going to be digitised. After digitizer configuration the grabbing begins and slides are saved on disk as they are acquired. When all the slides have been grabbed, a reference slides collection can be used to convert the acquired slides into projections from which cone-beam reconstruction can be performed using any Strange Engine's ActiveX plug-in. If needed, before reconstruction, a processor object can provide projection pre-processing.



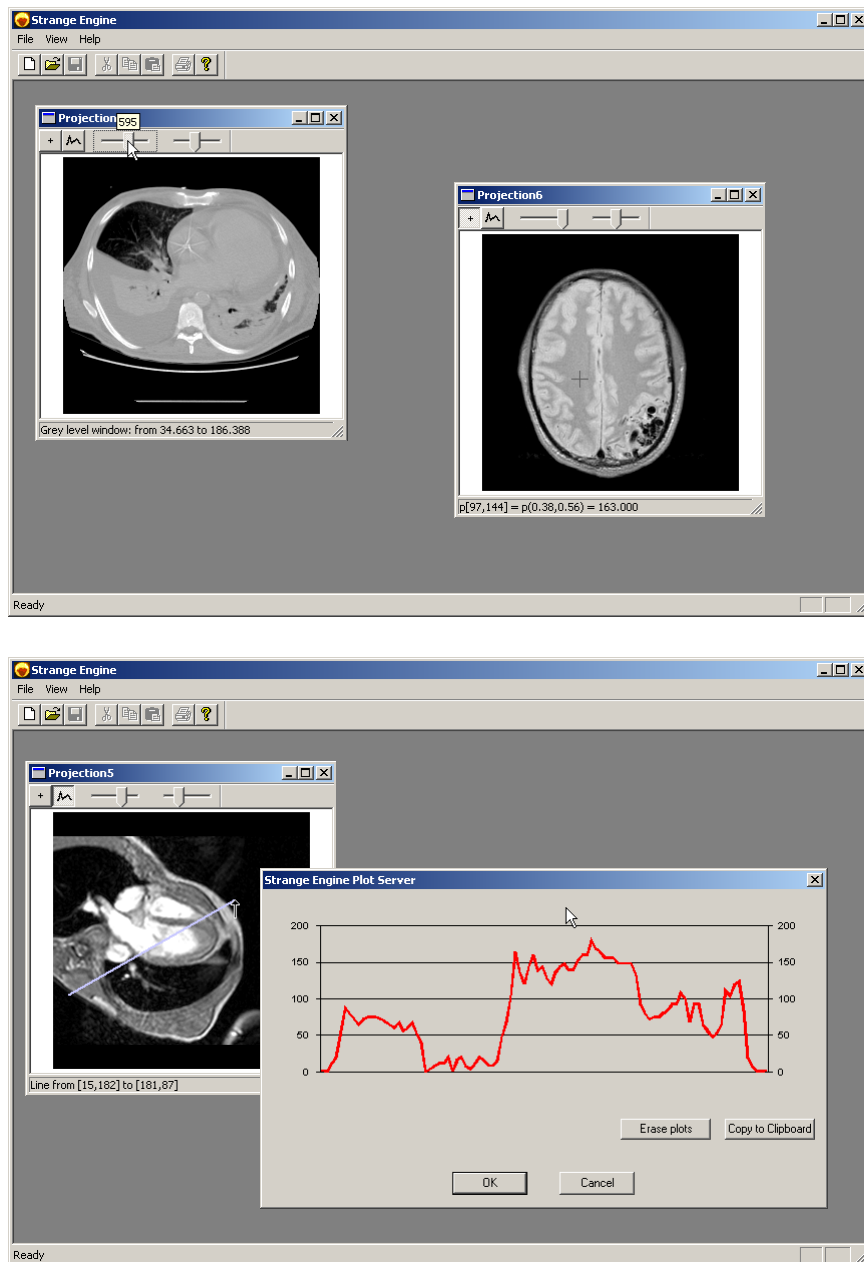
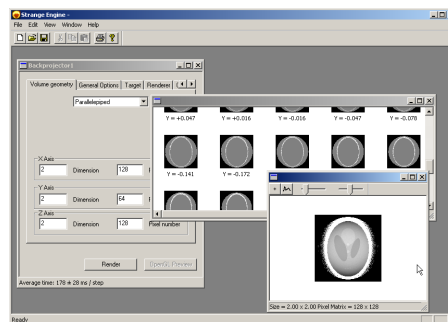
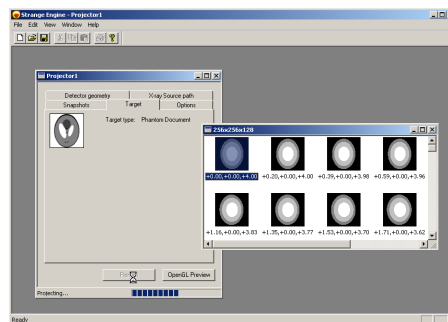
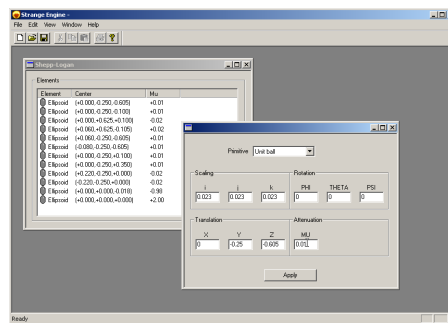


Figure 4.29. Strange Engine as a data viewer. Top: Image data is visualised with 256 grey shades and windowing. Status bar reports: window position/width and pixel values (pointed, minimum and maximum). Bottom: a “profiler” tool lets the user examine image values along a line traced on the image picture.

4.4.3 Algorithm test

This is the premiere task for which Strange Engine has been developed. It's possible to test custom reconstruction algorithms encapsulated by the user into a Strange Engine compliant ActiveX plug-in or just use an off-the-shelf plug-in. To do this we load or create a test phantom by putting together ellipsoids and truncated cylinders as shown in Figure 4.31.

A projector is then used to compute a simulated cone-beam projection collection which is displayed as a projection icon list. Projector options include number of projections, geometry of the detector, detector path, number of pixels and pyramidal beam anti-alias. Once simulated projections have been calculated we can create a backprojector, select our choice for the reconstruction plug-in and ask for a render: a volume object is produced. The volume object is visualised as a slice collection (see Figure 4.32). We can view each slice and examine them through the image functions examined in the data display example. Comparison with a voxel representation of the phantom is also possible.



4.4.4 Performance

Strange Engine performance is heavily hardware dependent. Table 4.1 reports computation times for some typical task. Two studies have been investigated:

- the S study : 3D Shepp-Logan phantom projection to 64 projections (64×64 pixels) which are backprojected into a 64^3 voxel cube;
- the V study : video acquisition and backprojection of 300 projections (512×540 pixels) into a $256 \times 256 \times 30$ voxel cube.

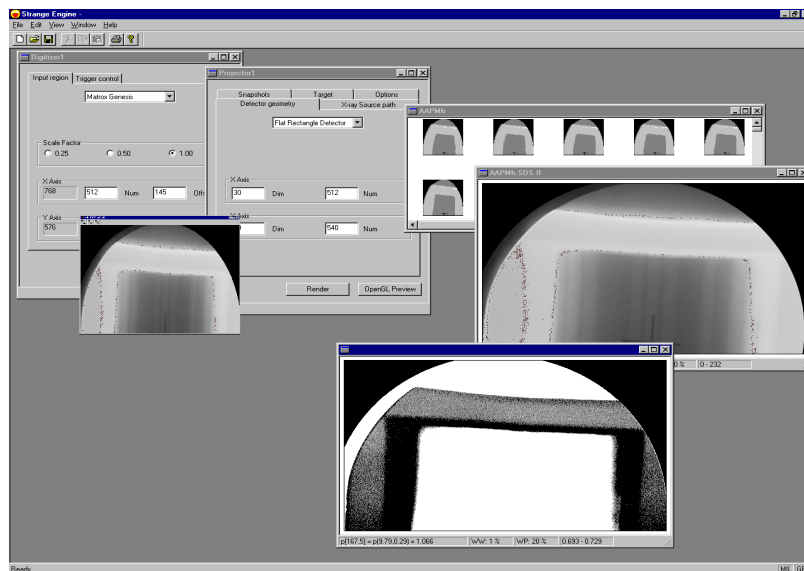


Figure 4.30. The Strange Engine's digitizer object provides real-time frame grabbing for video sources. Video input region, image scale and trigger are user defined. A live preview window (small window on the left) shows live video.

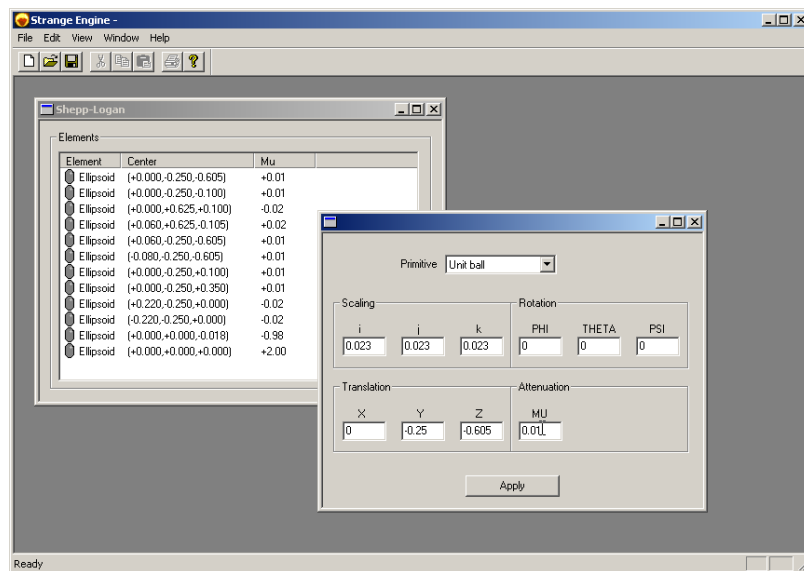


Figure 4.31. Strange Engine allows the user to assemble software phantoms by defining ellipsoids and truncated cylinders elements. This picture refers to the 3D Shepp-Logan phantom composed of 12 ellipsoids.

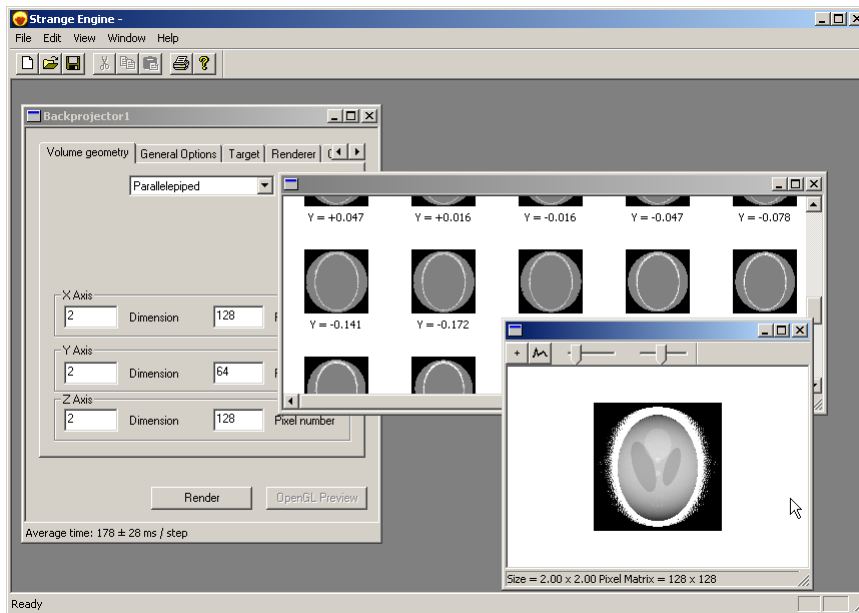
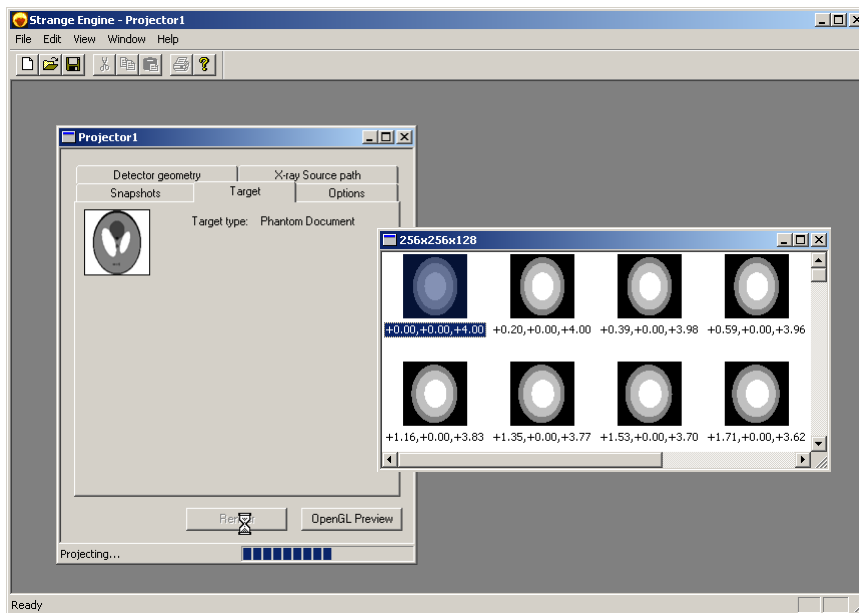


Figure 4.32. A Strange Engine's projector object is used to produce simulated cone-beam projections of a test phantom. Top: projections are organized as a projection collection (large window with multiple icons inside). Bottom: a projection collection is backprojected into a volume object which is visualised as a slice collection. A slice (small window on bottom right) is opened and examined.

Task	S study (seconds)	V study (seconds)
Full disk load	0 (inst.)	180/90/60/2*
Disk save	0	1/1/1/0
Projection	11.7/14.9/6.7/1.3	900/600/400/260
Backprojection	3.8/1.9/1.3/1.0	575/363/235/95

Table 4.1. Performance for some typical Strange Engine task. * = a different build has been used. For the V study “Projection” refers to slides to projections conversion.

Image size (pixels)	Maximum throughput (msec-fps)	
	to memory	to disk
768×576 (full CCIR)	123-8	300-3
512×540 (full FOV)	120-8	200-5
512×270 (half FOV)	85-12	120-8

Table 4.2. Maximum “safe” throughput for the PC_3 personal computer connected to CCIR video. Throughput is reported in minimum time per grab in msec and in frames per second (fps). “Safe” here means that the throughput variance calculated on 1 minute of continuous acquisition is inferior to 5% of average throughput.

The tasks examined were:

- full disk load: time required to load the projection collection from disk;
- disk save: time required to save the projection collection on disk;
- projection: time required to produce a projection collection;
- backprojection: time required to perform a reconstruction.

For both studies the reconstruction has been performed using the FDK ActiveX plug-in. To stress hardware dependence, four different personal computers have been considered:

- PC_1 a Pentium 133 with 32 MB RAM,
- PC_2 a Pentium Pro 200 with 128 MB RAM,
- PC_3 a Celeron 300 with 64 MB RAM and
- PC_4 an Athlon 600 with 256 MB RAM.

Acquisition maximum video throughput for the PC_3 personal computer have also been examined (see Table 4.2) for a CCIR video source.

4.5 Toolkit internals

Strange Engine is written in C++ and has been developed on Microsoft Windows NT 4.0 and Windows 2000 using the Integrated Development Environment (IDE) Microsoft Visual C++ 6.0. Technically the build 12 distribution comes from ten projects: a main project defining the Strange Engine application and the object abstraction, two custom link libraries, three public domain link libraries and four ActiveX plug-ins.

4.5.1 The main project

The main project builds to the Strange Engine application. It is written using the Microsoft Foundation Classes (MFC) version 4.2 object framework [41, 60] and consists of more than 80 C++ classes.

User interface provides multiple documents management through standard Multi Document Interface (MDI). Drag & drop is achieved through Windows Object Linking and Embedding (OLE).

Graphical display and rendering is provided by the standard Windows library Graphics Device Independent (GDI). Experimental three-dimensional display based on Silicon Graphics OpenGL [88] is also partially available.

Digitizer support is implemented through Matrox Imaging Library (MIL) and a camera specific Device Configuration File (DCF) “StrangeEngine.dcf”. For further information consult Matrox documentation [53].

COM support has been implemented following the treatment given in [18], recommendations found in [12] and the Microsoft ActiveX specifications [41, 45] for in-place controls. The Windows 2000 Category Manager is used to speed-up the search of suitable Strange Engine plug-ins installed in the system.

Plug-in user interface support is implemented through the two standard Windows ISpecifyPropertyPages and IPropertyPage COM interfaces [41]. This makes very easy to develop plug-in graphical user interface through ActiveX component IDEs like Microsoft Visual C++.

Mini API is specified using ActiveX Interface Definition Language (IDL).

Inspiration for some parts of the project and for some specific aspects of the implementation was taken from the stimulating reading of C++ programming technical books [17] and 3D graphics bibliography [76, 84].

4.5.2 Custom link libraries

Two C++ custom link libraries have been built to ease Strange Engine and companion plug-ins development: **VectorMath** and **FourierFarm**. While the former is for internal usage only, the latter is publicly available in Microsoft Windows Dynamic Link Library (DLL) format [64].

VectorMath is a static link library for Strange Engine and companion plug-ins internal usage only. It provides C++ classes for object-oriented management of vector algebra, i.e. points, vectors, quaternions and matrices. It is basically a partial transposition of Java 3D vector math API [76].

FourierFarm is an intelligent DLL wrapper which encapsulates filtering and 1D and 2D Fast Fourier Transform (FFT) functions. Depending on requested Fourier transformation and on libraries installed in the system it switches from our custom implementation based on Duhamel-Hollman split-radix algorithm, to the Faster Fourier in The West (FFTW) proposed by MIT researchers in [26]. A dedicated version for Intel Math Kernel Library⁶ (MKL) is also available.

4.5.3 Other link libraries

Three other link libraries, derived from public domain distributions, have been included in the current Strange Engine 1.0a build 12 project:

- papyrus [62] for DICOM 3 support (see also [1]),
- hdf for Hierarchical Data Format support⁷,
- zlib⁸ for binary gzip compression.

These libraries have been built as separate static link libraries. They are used internally by Strange Engine main application for file formats exchange.

⁶<http://developer.intel.com>

⁷<http://hdf.ncsa.uiuc.edu>.

⁸<http://www.cdrom.com/pub/infozip/zlib/>.

4.5.4 ActiveX plug-ins

Four ActiveX reconstruction plug-ins are provided with Strange Engine 1.0a build 12 distribution:

- FDK,
- FDK+,
- TFDK and
- AInv.

All the four plug-ins are supplied as ActiveX libraries (.ocx files). The FDK ActiveX plug-in is also available as source code to serve as example for users willing to write their own reconstruction plug-ins (as described in § 4.3).

FDK is our straightforward implementation of the Feldkamp-Davis-Kress algorithm which have been formulated in § 3.5.1.

FDK+ is our optimised and extended implementation of the FDK method.

TFDK is our implementation of oblique-parallel rebinned FDK of § 3.6.1.

AInv is our implementation of Dietz's reconstruction formula (3.47) and reconstruction kernel (3.49) for Louis's approximate inverse algorithm.

An in-depth analysis of plug-ins features and performance and a comprehensive comparison follows in the next chapter.

4.6 The build 12 distribution

Strange Engine runs on Microsoft Windows NT 4.0 and Windows 2000 PCs. The minimum recommend system set-up includes a Pentium II class computer, 128 MB of RAM and GB sized storage disk. Video support requires a Matrox Genesis card and Matrox MIL library.

Strange Engine 1.0a build 12 distribution is available on the Internet at:

<http://www.bigfoot.com/~agostinellis>

ActiveX plug-ins implementation

The Strange Engine software distribution currently contains four proprietary ActiveX reconstruction plug-ins: FDK, FDK+, TFDK and AInv. The custom filtering Fourier library FourierFarm is also provided. In the following sections we are going to examine implementation details for each of these solutions, show reconstruction results and make a comparison of the computational and overall performances.

5.1 Introduction

The ActiveX plug-ins included in the Strange Engine 1.0a build 12 distribution implement three different cone-beam reconstruction algorithms for the circular-path + planar detector geometry of Figure 3.14:

- FDK and FDK+ \rightarrow Feldkamp-Davis-Kress method,
- TFDK \rightarrow oblique-parallel rebinned Feldkamp-Davis-Kress,
- AInv \rightarrow Louis's approximate inverse through Dietz's formula.

All three algorithms are of filtered backprojection kind. Figure 5.33 shows how the reconstruction is actually performed. In general three consecutive steps are done: pre-weighting, filtering and backprojection. Note that the AInv and the TFDK plug-ins provide in addition internal code for Dietz's kernel computation and cone-beam to oblique-parallel rebinning respectively.

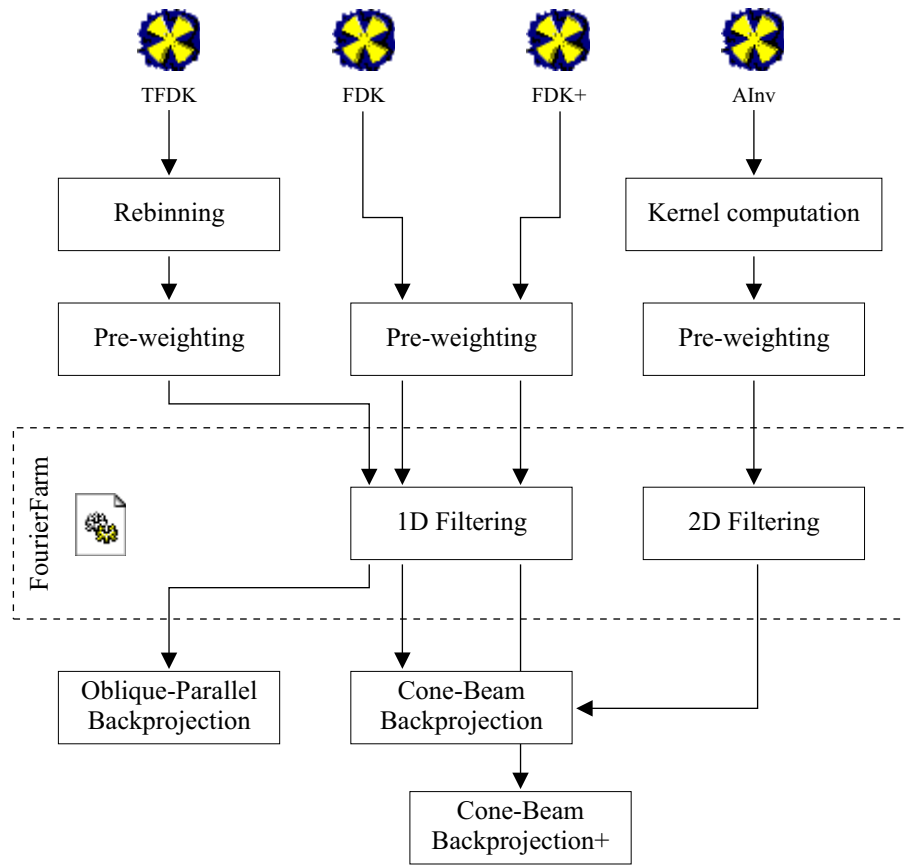


Figure 5.33. Reconstruction steps for Strange Engine build 12 ActiveX plug-ins.

5.2 Evaluation of different Fast Fourier Transform codes

A crucial step for filtered backprojection algorithms is obviously filtering. Strange Engine's ActiveX plug-ins employ our object-oriented FourierFarm DLL to perform fast filtering through Fourier transformation. This allows transparent and future-proof exploitation of new filtering techniques and codes as they become available. Four different FFT codes have been evaluated:

- our “My FFT” code based on Duhamel-Hollman split-radix algorithm,
- MIT's Fastest Fourier in The West version 2.1,
- Intel Math Kernel Library 3.1 and
- AMD ADSP library 3.1.

The FFTW and MKL libraries provide both complex-to-complex as well as real-to-complex fast Fourier transformation. My FFT code and ADSP library provide only complex-to-complex transformation. MKL, ADSP and My FFT codes require an input array of 2^N points, while FFTW manages an arbitrary number of points.

The MKL library is optimised for Intel DSP instructions and makes use of an adaptive scheme to load different libraries for different Intel processors. The ADSP library is optimised for AMD DSP instructions.

A series of speed tests have been performed through our BenchFFT routine. Systems considered were a Intel Pentium Pro 200 Mhz and an AMD Athlon 600 MHz with plenty of memory. Some results are shown in Figure 5.34 and Figure 5.35. We have found MKL to be in general a bit faster than FFTW and both to be much faster than our My FFT code. MKL is also expected to be somewhat faster on more recent Intel processors. ADSP library is strangely not giving good scores even on the recent AMD Athlon processor. We have also tested ADSP convolve function and found it not be brilliant as well.

In conclusion we can say that FFTW is a very good FFT solution though the MKL library should be used whenever possible to achieve maximum performance. Our My FFT is not bad as a last alternative solution. This is exactly the strategy followed by our FourierFarm DLL.

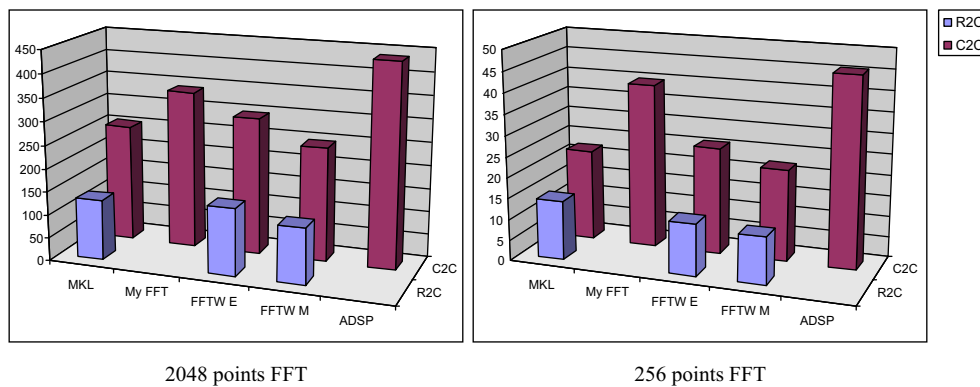


Figure 5.34. Performance for four different FFT codes on an Athlon 600 MHz system. Back histogram: complex-to-complex (C2C). Front histogram: real-to-complex (R2C). On ordinates time per transformation is reported in microseconds. FFTW E and FFTW M scores were taken using “Estimate” and “Measure” operation modes. Here the FFTW library was compiled with maximize speed optimisation on Visual C++6.0.

5.3 The FDK plug-in

5.3.1 General remarks

First of all let us rewrite once again the Feldkamp-Davis-Kress method we already examined in § 3.5.1. The FDK formula is:

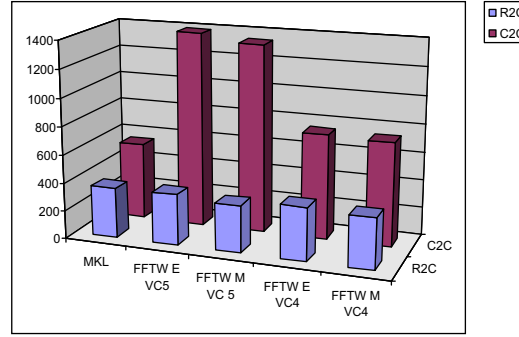
$$\mu(x, y, z) = \frac{1}{2} \int_0^{2\pi} \frac{R^2}{(R-s)^2} \int \frac{D}{\sqrt{D^2 + p^2 + q^2}} \cdot d_f(\beta, p, y \frac{D}{R-s}) k(t \frac{D}{R-s} - p) dp d\beta \quad (5.50)$$

The straightforward three-steps implementation of this formula follows.

Step 1 - Pre-weighting:

$$g(\beta, p, q) = d_c(\beta, p, q) \times \frac{D}{\sqrt{D^2 + p^2 + q^2}}$$

This means that for every cone-beam projection, each pixel is weighted by its distance from the X-ray source. In our geometry these factors are the same for every projection so they can be cached. The cost of this step is hence $N_p \times N_q \times N_\beta = N^3$ multiplications with obvious meaning of the symbols.



2048 points FFT

Figure 5.35. Performance of MKL and FFTW libraries on a PPro 200 MHz system. Back histogram: complex-to-complex (C2C). Front histogram: real-to-complex (R2C). On ordinates time per transformation is reported in microseconds. FFTW E and FFTW M scores were taken using “Estimate” and “Measure” operation modes. Here the FFTW library was compiled with maximize speed optimisation on Visual C++4.0 (VC4 scores) and Visual C++5.0 (VC5 scores).

Step 2 - Filtering:

$$\tilde{d}_c(\beta, p, q) = \int g(\beta, p_1, q) k(p - p_1) dp_1$$

Each projection is filtered row-by-row. The filter is the same for all the rows so it can be computed only once. However it's necessary to perform $N_\beta \times N_q$ convolutions of N_p points. The theoretical cost for this is $N_\beta \times N_q \times N_p^2 = N^4$. A cost reductions can be achieved through Fourier transformation: it is easy to see that $2N_\beta \times N_q$ 1D FFTs plus $N_\beta \times N_q \times N_p$ multiplications are required. An overall cost of $2N^3 \log N$ is hence expected.

Step 3 - Backprojection:

$$\mu(x, y, z) = \frac{1}{2} \int_0^{2\pi} \frac{R^2}{(R-s)^2} \tilde{d}_c(\beta, \frac{D}{R-s} t, \frac{D}{R-s} y) d\beta$$

with $t = x \cos \beta - z \sin \beta$ and $s = x \sin \beta + z \cos \beta$. This step is the most computationally expensive. For each of $N_x \times N_y \times N_z = N^3$ reconstruction points, N_β Multiply And aCcumulate (MAC) operations have to be performed. The $1/r^2$ term is y -independent so it's the same for all the slices parallel to the source rotation plane. Therefore its calculation takes N_β 2D matrix-vector multiplications. If projections are taken equi-angularly this can be reduced to $2N_\beta \times N_x \times N_z$ multiplications. The last cost is hidden in the voxel backprojection terms

$$p = \frac{D}{R-s} t, \quad q = \frac{D}{R-s} y$$

For nearest-neighbour interpolation it requires $2N_\beta \times N_x \times N_y \times N_z = 2N^4$ multiplications and $2N^4$ (very time consuming) floating-point to integer math conversions. This cost is even greater for more elaborate interpolation schemes like bilinear interpolation.

5.3.2 Computational cost

Table 5.3 summarizes the computational cost for the FDK algorithm. As already noticed in chapter 3, the total cost is $O(N^4)$. This comes essentially from the final backprojection step.

5.3.3 Interface implementation

The FDK plug-in has been developed on Microsoft Visual C++ 4.0 but the version contained in build 12 comes from a 6.0 compile. It implements all the three

Step	Computational cost
Pre-weighting	N^3 mults.
Filtering	$2N^3 \log N$ 1D FFTs + N^3 mults $\simeq 2N^3 \log N$ mults.
Backprojection	N^4 MACs + $2N^4$ mults. $\simeq 3N^4$ mults.
TOTAL	$O(N^4)$

Table 5.3. Computational costs for the Feldkamp-Davis-Kress method.

methods of Strange Engine's IBackProjectorRenderer interface: the mandatory IBackProjectProjection and IBackProjectPhantom and also the optional IBackProjectProjectionCollection.

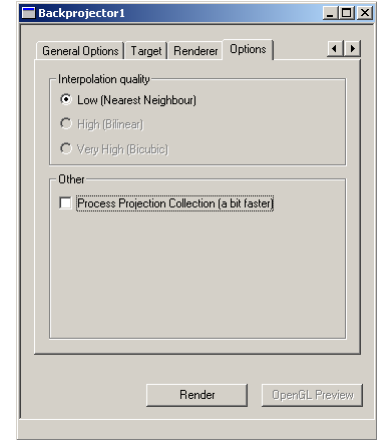
5.3.4 Reconstruction implementation

The reconstruction code is a no frills implementation of the FDK method. The three steps have been written directly as they are in the original formula:

- No pre-weight matrix is cached.
- Filtering is performed through FourierFarm. The kernel is a Ram-Lak filter band-limited to Nyquist frequency (see filter k_{B_Ω} in § 2.4.3).
- Filter is computed only once in IBackProjectProjectionCollection.
- Reconstruction region is limited to the unit ball.
- Interpolation is nearest-neighbour.

5.3.5 User interface

The Figure on the right shows the user interface of the FDK plug-in. Two user controls are available. A radio control selects the interpolation scheme (currently only nearest-neighbour). A checkmark control let the user choose to call the IBackProjectProjection method or the IBackProjectProjectionCollection method (which is a bit faster).



5.4 The FDK+ plug-in

The FDK plug-in has been thought as pedagogical and in fact is available as source code. The FDK+ plug-in builds on FDK plug-in experience to achieve faster reconstruction. Speed-up is essentially gained rewriting the backprojection implementation.

5.4.1 Interface implementation

The FDK+ plug-in has been developed on Microsoft Visual C++ 6.0. It implements `IBackProjectPhantom` and `IBackProjectProjectionCollection`.

5.4.2 Reconstruction implementation

The pre-weighting and filtering routines are almost identical to the ones in the FDK plug-in. Higher reconstruction speed is reached using a series of “tricks” in the crucial final backprojection step:

- angular symmetry,
- axial symmetry,
- axial interpolation skip for nearest-neighbour interpolation,
- fixed point math,
- index Look Up Table (LUT) boost.

The flow diagram of the FDK+ tricks is showed in Figure 5.36. Each one is presented in the following paragraphs.

Angular symmetry. If the number of projections is even, an obvious azimuthal symmetry is fulfilled because given the projection at angle β the complementary $\beta + \pi$ projection always exist in the available projection collection. In these conditions it is advantageous to carry on backprojection for a projection and its complementary at the same time. This is because for a certain reconstruction point (x, y, z) :

$$\begin{aligned}s(\beta + \pi) &= -s(\beta) \\ t(\beta + \pi) &= -t(\beta)\end{aligned}$$

Computation time for s and t can then cut by half. Please note that while the saving of a multiplicative factor in the backprojection step does not certainly

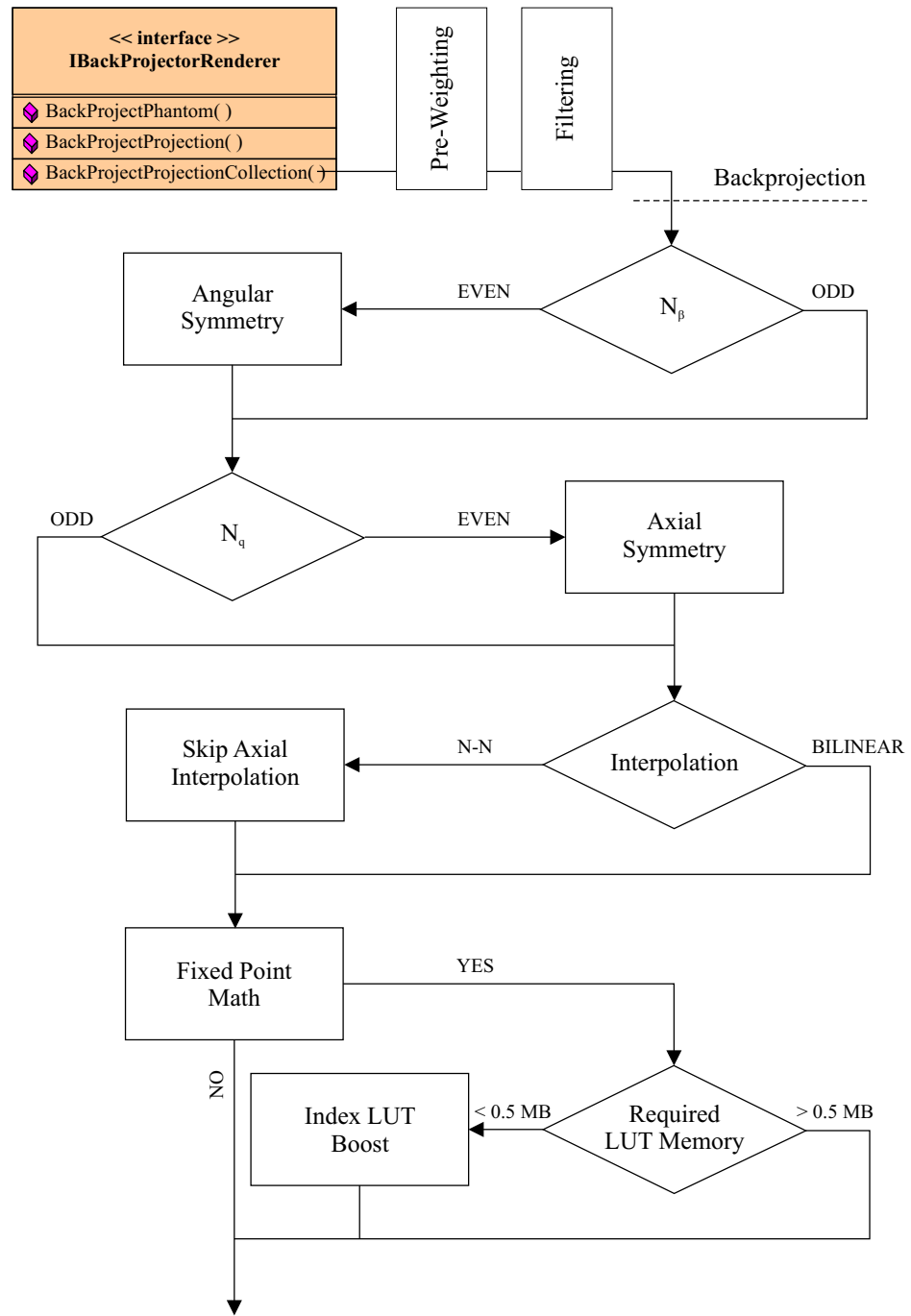


Figure 5.36. Flow diagram of backprojection tricks used in the FDK+ plug-in.

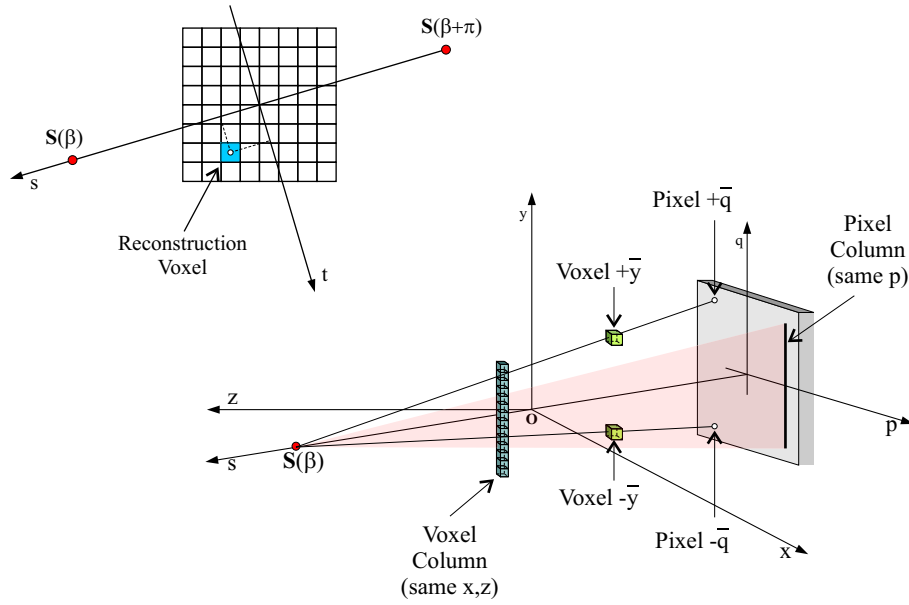


Figure 5.37. Angular/axial symmetry and interpolation skip acceleration. Top left: computation of s and t is done only once for complementary projections. Bottom : a voxel column projects to a pixel column and two voxels with $y = \pm \bar{y}$ project to $\pm \bar{q}$.

modify the order of the computational effort it can really provide real acceleration because it has to be multiplied by a huge N^4 term. So even a small factor can produce big time savings.

Axial symmetry. Another symmetry does exist if the number of planes in the axial direction is even. Given a certain reconstruction point (x, \bar{y}, z) the same calculation of $\bar{q} = \bar{y} D / (R - s)$ can be used with sign reversed for point $(x, -\bar{y}, z)$. This cuts computation of q by half.

Axial interpolation skip. FDK+ provides nearest-neighbour interpolation only. Backprojection using this interpolation is particularly suited to be optimised. We found that for a given column of voxels (parallel to the axial direction) it's possible to skip the backprojection on the row direction because their projection on the detector plane is a line parallel to the axial direction. In other words the pixel index in the row direction is the same for all the voxels of a column so that we can compute row backprojection (and floating-point to integer conversion) only once. This makes possible a saving of a N_y factor in row backprojection.

Fixed point math. While in the recent years the floating-point performance of processors has been increasing dramatically, integer units are still much faster. This means that in general processing speed for integer numbers will be much greater than for floating-point numbers. This simple observation suggested us to investigate usage of fixed-point math in the backprojection step. Fixed-point math uses integers to represent decimal numbers.

Let $T_{fp} = 1.23456\dots$ be a floating-point value. If we multiply T_{fp} by a “big” integer number X (base) and take the integer part of the result, we end with an integer number that we call a fixed-point version of T_{fp} and which will be denoted as T_{xp} . We can write:

$$T_{xp} = \text{int}(X \times T_{fp})$$

Obviously T_{fp} can be recovered by dividing its fixed-point version by X . Note that this introduces an error which depends on the base and on the order of magnitude of T . This error can be reduced increasing X . However if we increase the base too much we will produce very big integer numbers which will not fit processor registers. Usually this means that X has to be such that T_{xp} fits into 32 bits. Clearly a compromise between small base values (great errors) and big base values (too many bits) has to be established.

To exploit fixed-point math acceleration all the involved quantities in the backprojection step have to be converted to fixed-point. In FDK+ all the geometric quantities, like the scale factors, are computed as floating-point values and then converted into fixed-point math. The usual value of the base is $X = 8192$. In the case in which fixed-point would overflow 32 bits, FDK+ automatically switches to floating-point backprojection. Usage of 64 bits integers on Intel processors has also been investigated, but was found not to be any faster than usage of floating-point numbers.

Index LUT boost. If fixed-point acceleration has been turned on, an additional small speed-up can be achieved in the backprojection step by pre-computing voxel indexes and storing these values in a Look Up Table. Index LUT boost is automatically switched on by FDK+ if memory required for the LUT is inferior to 512 KB.

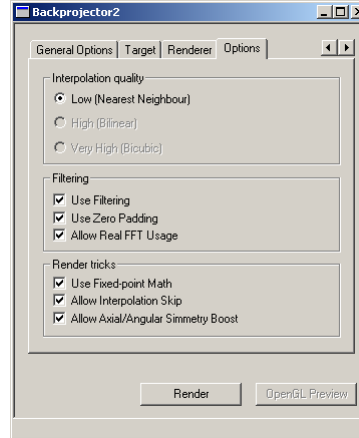
5.4.3 Filtering and zero padding

Another FDK+ feature is zero padding. To “zero pad” an array of numbers means to append to it a certain number of zeros. Zero padding finds application in the digital filtering in which arrays are convolved through circular

convolution. In computed tomography circular convolution can produce reconstruction artefacts like so called “cupping” (see [40]). To avoid this kind of effects the FDK+ plug-in uses zero padding through our FourierFarm DLL. Given a projection of $N_p \times N_q$ pixels, the padding produces a $2N_p \times N_q$ pixels projection which is Fourier transformed, multiplied row-by-row with a padded Ram-Lak filter and inverse transformed. Filtering ends by taking the first half of the filtered $2N_p \times N_q$ pixels projection.

5.4.4 User interface

The Figure on the right shows the user interface of the FDK+ plug-in. 7 user controls are available. A radio control selects the interpolation scheme (currently only nearest-neighbour). 6 checkmark controls let the user choose filtering options (no filtering, zero padding and real-to-complex FFT) and rendering tricks (angular and axial symmetry, fixed-point math and interpolation skip).



5.5 The TFDK plug-in

The TFDK plug-in provides an experimental implementation of the oblique-parallel rebinned FDK method presented in § 3.6.1. This method is based on a rebinning step in which cone-beam projections are used to compute oblique-parallel projections and on a modified FDK formula in which the computational expensive $1/r^2$ term is removed:

$$\mu(x, y, z) = \frac{1}{2} \int_0^{2\pi} \int \frac{D}{\sqrt{D^2 + q^2}} d_{op}(\beta, p, q) k(t - p) dp d\beta$$

Reconstruction from cone-beam projections is therefore a four steps process.

Step 1 - Rebinning: For each pixel (\bar{p}, \bar{q}) of the $\bar{\beta}$ oblique-parallel projection a cone-beam pixel $d_c(\beta, p, q)$ has to be found through the rebin relation:

$$\begin{aligned}\beta &= \bar{\beta} + \arcsin \frac{\bar{p}}{R} \\ p &= D \tan(\arcsin \frac{\bar{p}}{R}) \\ q &= \bar{q} \sqrt{\frac{D^2 + p^2}{R^2 - \bar{p}^2}}\end{aligned}$$

We expect the computational cost of this step to be quite high, because for each of the $N_{\bar{\beta}} \times N_{\bar{p}} \times N_{\bar{q}} = N^3$ oblique-parallel pixels at least 3 multiplications and 3 complex math functions have to be performed. However rebinning is not really a reconstruction step but has to be considered a pre-processing step, so its cost is not that much important.

Step 2 - Pre-weighting:

$$g(\beta, p, q) = d_{op}(\beta, p, q) \times \frac{D}{\sqrt{D^2 + q^2}}$$

This means that for every oblique-parallel projection, each row has to be weighted by its distance from the X-ray source. In our geometry these factors are the same for every projection so they can be cached. Note that unlike in the traditional FDK method the pre-weight factor is not depending on p so the cost of this step is just $N_q \times N_{\beta} = N^2$.

Step 3 - Filtering:

$$\tilde{d}_{op}(\beta, p, q) = \int g(\beta, p_1, q) k(p - p_1) dp_1$$

Is exactly the same as in the FDK method. Expected cost is then $2N^3 \log N$.

Step 4 - Backprojection:

$$\mu(x, y, z) = \frac{1}{2} \int_0^{2\pi} \tilde{d}_{op}(\beta, p, q) d\beta$$

with $p = t$ and $q = y D / (R - s)$. Note that the computationally expensive $1/r^2$ term is gone and that it's necessary to perform a multiplication only for the q coordinate. For nearest-neighbour interpolation the overall backprojection cost is then N^4 multiplications + $2N^4$ floating-point to integer math conversions. Some performance gain in comparison to traditional FDK backprojection is hence expected.

5.5.1 Interface implementation

The TFDK plug-in has been developed on Microsoft Visual C++ 6.0. It implements IBackProjectPhantom and IBackProjectProjectionCollection.

5.5.2 Reconstruction implementation

Implementation of TFDK plug-in reconstruction is much similar to the one for the FDK plug-in. Pre-weighting is not cached. Filtering is performed through our FourierFarm DLL, filter used is Ram-Lak filter band-limited to Nyquist frequency. Zero padding is not used. Backprojection code is a straightforward implementation of oblique-parallel backprojection.

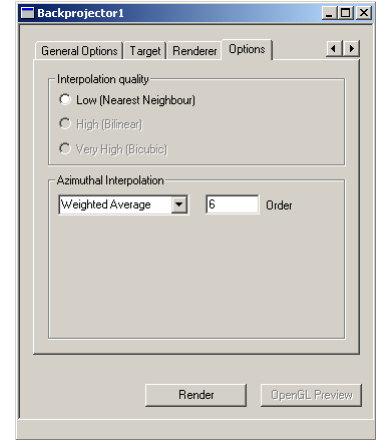
What makes TFDK plug-in apart is the cone-beam to oblique-parallel beam rebinning step. Implementation follows this scheme:

- a virtual detector:
 - is positioned on the origin,
 - has dimensions computed scaling cone-beam detector dimensions,
 - has the same number of pixels as for the cone-beam detector,
- the number of oblique-parallel projections (N_β) is taken equal to the number of cone-beam projections,
- each oblique-parallel projection is computed through azimuthal super-sampling:
 - K_{ss} temporary projections are computed, through the rebin relation and bilinear interpolation,
 - and interpolated to form a single oblique-parallel projection;
- each pixel of the virtual detector is calculated using the cone-beam to oblique-parallel beam rebinning relation reported above.

Further details are available in [21].

5.5.3 User interface

The user interface of the TFDK plug-in is shown in the Figure on the right. Three controls are available. A radio control selects the interpolation scheme (currently only nearest-neighbour). A drop-list control selects temporary projections interpolation (currently only weighted average). An edit control retrieves the order of interpolation K_{ss} .



5.6 The AInv plug-in

The AInv plug-in implements the approximate inverse reconstruction scheme for cone-beam projections using Dietz's reconstruction formula and Dietz's approximation for the reconstruction kernel:

$$\tilde{f}_\gamma(\mathbf{r}) = \frac{(R-1)^2}{8} \int_0^{2\pi} \frac{1}{|\mathbf{r} - \mathbf{S}(\beta)|^2} \tilde{d}_c(\beta, p, q) d\beta$$

with

$$\tilde{d}_c(\beta, p, q) = \iint d_c(\beta, p_1, q_1) \tilde{k}_\gamma(p - p_1, q - q_1) dp_1 dq_1$$

and

$$\begin{aligned} \tilde{k}_\gamma(p, q) = \tilde{k}(\boldsymbol{\theta}) = & -h \mathbf{S}_0 \cdot \boldsymbol{\theta} \int_{\boldsymbol{\theta}^\perp} \psi_\gamma(h \mathbf{S}_0 \cdot \boldsymbol{\omega}) |\mathbf{S}'_0 \cdot \boldsymbol{\omega}| d\boldsymbol{\omega} - \\ & - \mathbf{S}'_0 \cdot \boldsymbol{\theta} \int_{\boldsymbol{\theta}^\perp} \Psi_\gamma(h \mathbf{S}_0 \cdot \boldsymbol{\omega}) \text{sign}(\mathbf{S}'_0 \cdot \boldsymbol{\omega}) d\boldsymbol{\omega} \end{aligned}$$

The meaning of the symbols was explained in § 3.8. This algorithm is of filtered backprojection kind and basically differs from the FDK method only in the filtering part which is two-dimensional. Let us note that for each different geometry the reconstruction kernel formula provides a different filter. Hence before reconstruction we have to compute the appropriate filter. While caching can of course be used for known cone-beam geometries, in general reconstruction can be performed in three steps.

Step 1 - Filter computation Given a certain cone-beam geometry the appropriate filter can be computed using Dietz's approximate formula for the reconstruction kernel:

$$\begin{aligned}\tilde{k}_\gamma(p, q) = \tilde{k}(\boldsymbol{\theta}) = & -h \mathbf{S}_0 \cdot \boldsymbol{\theta} \int_{\boldsymbol{\theta}^\perp} \psi_\gamma(h \mathbf{S}_0 \cdot \boldsymbol{\omega}) |\mathbf{S}'_0 \cdot \boldsymbol{\omega}| d\boldsymbol{\omega} - \\ & - \mathbf{S}'_0 \cdot \boldsymbol{\theta} \int_{\boldsymbol{\theta}^\perp} \Psi_\gamma(h \mathbf{S}_0 \cdot \boldsymbol{\omega}) \text{sign}(\mathbf{S}'_0 \cdot \boldsymbol{\omega}) d\boldsymbol{\omega}\end{aligned}$$

In our case $\mathbf{S}_0 = (0, 0, R)$ and $\mathbf{S}'_0 = (R, 0, 0)$. The cost of this step is expected to be very high because it needs numerical integration. However the result can be cached and re-used for reconstruction in the same cone-beam geometry, so it has to be considered a pre-processing step.

Step 2 - 2D Filtering

$$\tilde{d}_c(\beta, p, q) = \iint d_c(\beta, p_1, q_1) \tilde{k}_\gamma(p - p_1, q - q_1) dp_1 dq_1$$

Once the reconstruction filter has been computed, two-dimensional filtering is performed on cone-beam projections. The cost of this step is in theory $N_\beta \times N_p^2 \times N_q^2 = N^5$, but can be reduced to $2N_\beta$ 2D FFTS + $N_\beta \times N_p \times N_q$ multiplications. An overall $4N^3 \log N$ cost is expected (double the expected cost for 1D filtering).

Step 3 - Backprojection

$$\tilde{f}_\gamma(\mathbf{r}) = \frac{(R-1)^2}{8} \int_0^{2\pi} \frac{1}{|\mathbf{r} - \mathbf{S}(\beta)|^2} \tilde{d}_c(\beta, p, q) d\beta$$

This formula is much similar to traditional FDK backprojection formula. For nearest-neighbour interpolation thus a $2N_\beta \times N_x \times N_y \times N_z = 2N^4$ multiplications + $2N^4$ floating-point to integer math conversions cost is expected.

5.6.1 Interface implementation

The AInv plug-in has been developed on Microsoft Visual C++ 6.0. It implements IBackProjectPhantom and IBackProjectProjectionCollection.

5.6.2 Reconstruction implementation

Implementation of AInv plug-in reconstruction is again much similar to implementation for the FDK plug-in. Pre-weighting is not cached. 2D Filtering is

performed through our FourierFarm DLL. Zero padding is not used. AInv manages reconstruction kernels through on-disk caching. On a rendering request disk is searched for a suitable kernel file:

- if kernel file is found, it is loaded and used for 2D filtering,
- otherwise the kernel is computed, through optional supersampling, and saved on disk for later use.

Kernel computation. The AInv plug-in makes use of the Gaussian mollifier

$$e_{\gamma}^g(\mathbf{x}) = \frac{1}{(2\pi\gamma^2)^{3/2}} e^{-|\mathbf{x}|^2/2\gamma^2}$$

Closed forms of the ψ_{γ} and Ψ_{γ} functions for the Gaussian mollifier were given in § 3.8. AInv computes the reconstruction kernel using straightforward numerical integration of Dietz’s kernel formula. For each pixel of the kernel:

1. $\boldsymbol{\theta}$, $(\mathbf{S}_0 \cdot \boldsymbol{\theta})$ and $(\mathbf{S}'_0 \cdot \boldsymbol{\theta})$ are computed,
2. the $\boldsymbol{\theta}^{\perp}$ plane is found,
3. the two integrals on $\boldsymbol{\theta}^{\perp}$ are computed by summing on “small” mesh steps.

To speed-up kernel computation only one quadrant of the reconstruction kernel is computed. The other three quadrants are obtained by simple mirroring. A supersampling experimental option is also available. With supersampling each pixel is obtained as the average of 2^k “superpixels” contained in the pixel. In our tests supersampling proved not to bring any real improvement in the reconstruction. However it can be used to fasten kernel computation because a kernel of size $X \times Y$ can be obtained, if all other parameters are the same, by mip-mapping of a supersampled kernel $kX \times kY$. Figure 5.38 shows two reconstruction kernels computed for $\gamma = 1^{-3}$ and $\gamma = 1^{-2}$ for the same cone-beam geometry.

Kernel file format. Kernels are stored as raw binary files. Each pixel of a kernel is represented by a 32 bit IEEE (Intel endian) value. Every kernel file name is specified in the format:

$$\text{“Dim}(p) \rightarrow \text{Dim}(q) \rightarrow N_p \rightarrow N_q \rightarrow R \rightarrow (D - R) \rightarrow \gamma.ker\text{”}$$

Where the separator \rightarrow is the underscore ‘_’ and where $\text{Dim}(p)$ and $\text{Dim}(q)$ are the detector dimensions along axes p and q .

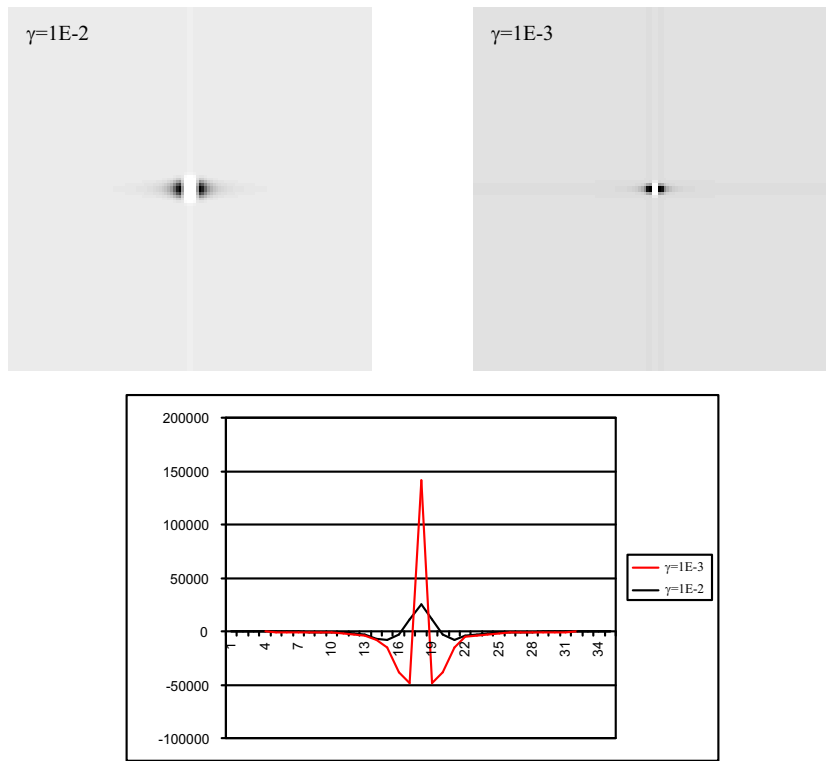
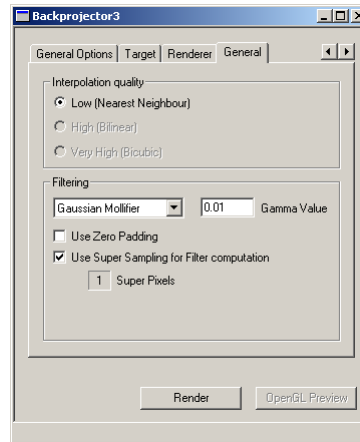


Figure 5.38. Reconstruction kernels for two γ values. Left and black line: $\gamma = 1^{-2}$. Right and red line: $\gamma = 1^{-3}$. Plots refers to profile taken on the central horizontal line. Cone-beam geometry was: $\text{Dim}(p) = \text{Dim}(q) = 4$, $N_p = N_q = 128$, $R = 4$, $D = 8$.

5.6.3 User interface

The user interface of the AInv plug-in is shown in the Figure on the right. Three controls are available. A radio control selects the interpolation scheme (currently only nearest-neighbour). A drop-list control selects the mollifier (currently only Gaussian mollifier). An edit control retrieves the γ value. Two checkmark controls selects zero padding (not implemented yet) and supersampling.



a	b	c	x_c	y_c	z_c	α ($^\circ$)	β ($^\circ$)	μ
0.69	0.92	0.9	0.0	0.0	0.0	0	0	2.0
0.6624	0.88	0.874	0.0	0.0	0.0	0	0	-0.98
0.41	0.21	0.16	-0.22	0.25	0.0	0	72	-0.02
0.31	0.22	0.11	0.22	0.25	0.0	0	-72	-0.02
0.21	0.35	0.25	0.0	0.25	0.35	0	0	0.01
0.046	0.046	0.046	0.0	0.25	0.1	0	0	0.01
0.046	0.02	0.023	-0.08	0.25	-0.605	0	0	0.01
0.046	0.02	0.023	0.06	0.25	-0.605	0	0	0.01
0.056	0.02	0.023	0.06	-0.625	-0.105	0	90	0.01
0.056	0.16	0.04	0.0	-0.625	0.1	0	90	0.02
0.046	0.046	0.046	0.0	0.25	-0.1	0	0	0.01
0.023	0.023	0.023	0.0	0.25	-0.605	0	0	0.01

Table 5.4. Definition of the 3D Shepp-Logan phantom. In this phantom 12 ellipsoids are superimposed. For each ellipsoid: a , b and c are the radii; x_c , y_c and z_c are the centre coords; α and β are the rotation angles; μ is the signed attenuation coefficient.

5.7 Reconstruction results

All the Strange Engine reconstruction plug-ins have been tested and benchmarked on different computer platforms. Tests were performed mainly through an assortment of simulated cone-beam projections and a set of real-world cone-beam projections acquired from a Varian RT simulator (see [2, 5] for details).

Reconstructions produced by FDK, FDK+, AInv and (preliminary) TFDK plug-ins were investigated. In the beginning we also examined if the fixed-point math rendering trick of FDK+ plug-in does affect the quality of the reconstructed volume. We found that the FDK+ plug-in, with fixed-point math turned on, produces reconstructions which are visually identical to the ones for the FDK plug-in (see Figure 5.39). A maximum error of 1% has been observed for a fixed-point math base equals to 8192.

5.7.1 Simulated projections

We have examined reconstruction for two test phantoms. A simple ball-shaped phantom was used for code debug and for first implementation checks. The 3D Shepp-Logan phantom, included in the Strange Engine 1.0a build 12 distribution, was used for reconstruction evaluation. This phantom is composed of 10 small and very small, low contrast ellipsoids inside two large ellipsoids. Its definition, according to [24], is given in Table 5.4.

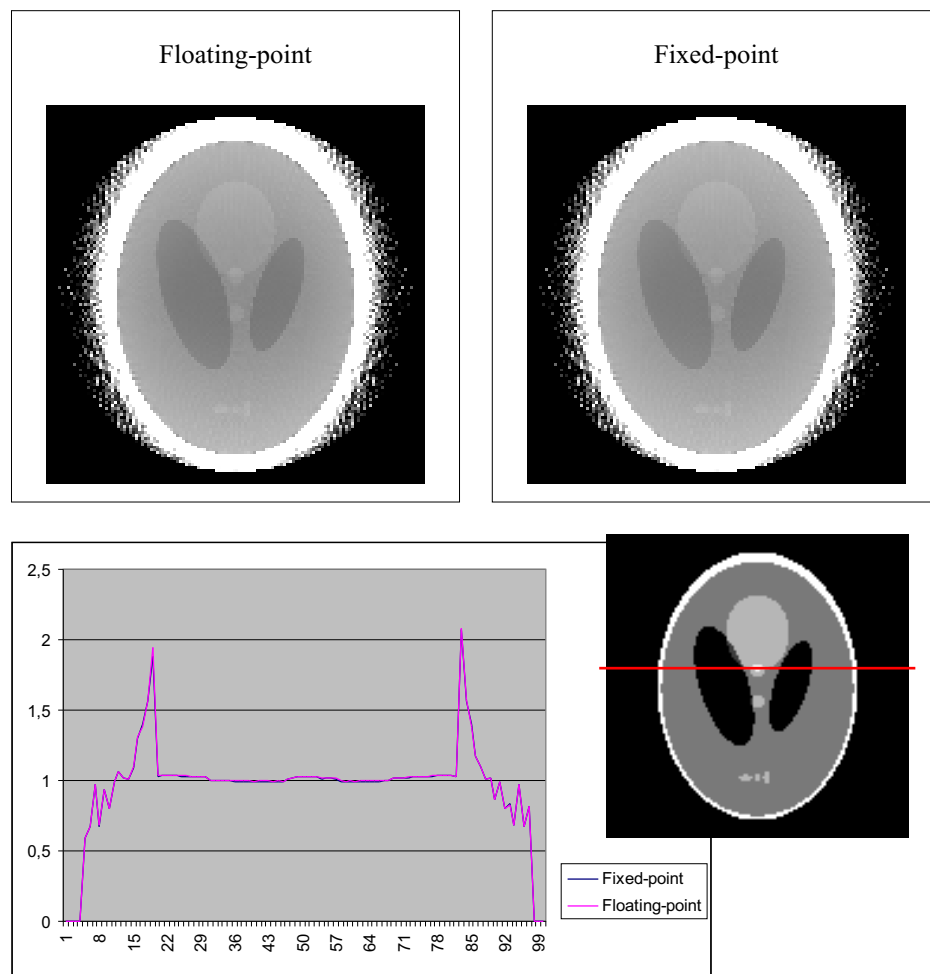


Figure 5.39. Comparison of the reconstructions of the 3D Shepp-Logan phantom section $y = -0.258$ produced by the FDK+ plug-in with fixed-point and floating-point backprojection. The reconstruction are visually identical. A profile plot shows that reconstruction error when fixed-point math is turned on is less than 1%.

Some reconstruction results for two of the most significant phantom sections ($y = -0.258$ and $z = 0.008$) are shown in Figure 5.40 and in Figure 5.41. These reconstructions were obtained using 128×128 pixels cone-beam projections computed by a Strange Engine's projector. The FDK+ reconstruction is quite good even for a small (64) number of projections and resolves the tiny three ellipsoids on the bottom of the section. The AInv reconstruction ($\gamma = 0.005$)

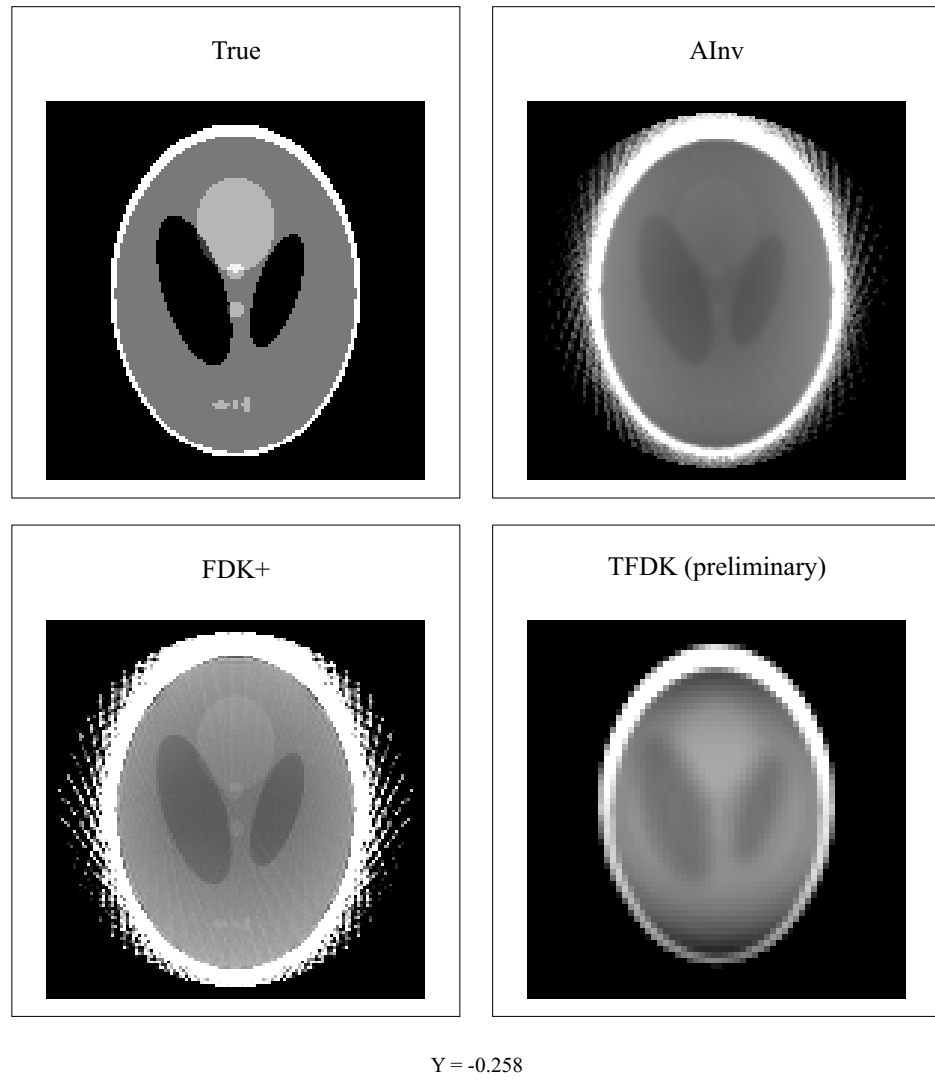


Figure 5.40. Reconstruction of the 3D Shepp-Logan phantom section $y = -0.258$. Top left: true section. Top right: AInv reconstruction for $\gamma = 5^{-3}$, 64 projections. Bottom left: FDK+ reconstruction with zero padding, 64 projections. Bottom right: preliminary TFDK for 240 rebinned projections and 12th order interpolation.

produces a much smoother (lower resolution) reconstruction with fewer details. Better resolution can be obtained decreasing the γ value. This would probably require to compute the reconstruction kernel on a finer, e.g. 256×256 pixels, grid. The (preliminary) TFDK plug-in has to make use of a much greater

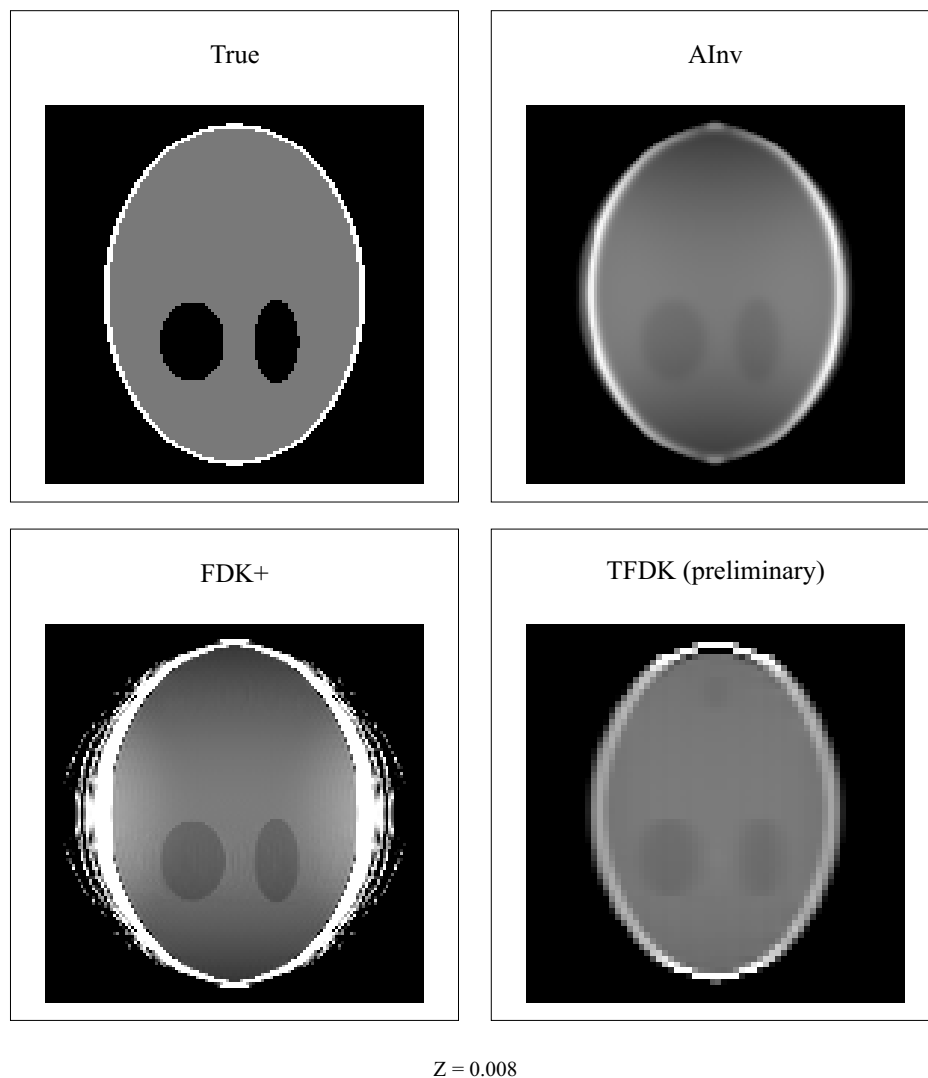


Figure 5.41. Reconstruction of the 3D Shepp-Logan phantom section $z = 0.008$. Top left: true section. Top right: AInv reconstruction for $\gamma = 5^{-3}$, 64 projections. Bottom left: FDK+ reconstruction with zero padding, 64 projections. Bottom right: preliminary TFDK for 240 rebinned projections and 12th order interpolation.

(240) number of projections and of a high order of azimuthal interpolation to achieve a similar reconstruction. For all the plug-ins better reconstructions can of course be attained using an increased number of projections.

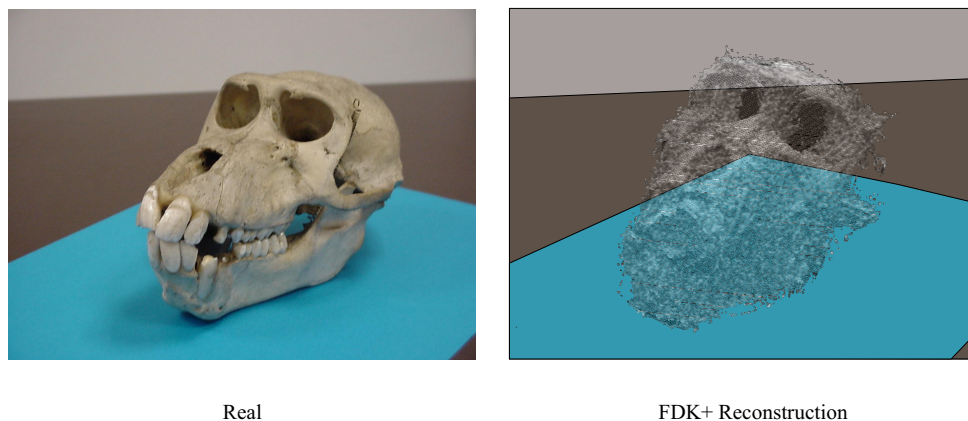


Figure 5.42. Surface rendering of the 256^3 voxels reconstruction of a baboon skull performed by the FDK+ plug-in in about 10 minutes on an Athlon 700 MHz system. 300 cone-beam projections of 512×540 pixels were grabbed by Strange Engine from the video output of a Varian Ximatron RT simulator, pre-processed to correct video chain distortions and reconstructed.

5.7.2 RT simulator projections

Thanks to the collaboration with the National Institute for Cancer Research of Genova we were also able to acquire a set of real-world cone-beam projections from the CCIR video output of the Varian Ximatron RT simulator installed in the Radiotherapeutic Oncology service of the Institute.

Projections were acquired and pre-processed through Strange Engine. A standard set of 300 projections, 512×540 pixels, was considered. Initially we examined simple high contrast objects, like plastic balls, acrylic cylinders and a metallic frontal pointer. This allowed us to determine the geometric distortions introduced by video chain faults and to prepare adequate pre-processing. Once the reconstruction of these simple objects was judged satisfactory we proceeded to investigate more complex tasks. An example of this is given in Figure 5.42 and in Figure 5.43 (float) in which we compare a real photograph of a baboon skull (about 20 cm of diameter) with the 3D surface rendering of a reconstruction performed by the FDK+ plug-in. While noisy the reconstruction shows quite good spatial resolution and resolves some of the skull's fine details. We expect better reconstruction to be easily achievable using smoother filtering. For further details on our CBCT system for RT simulators we refer to [2, 5].

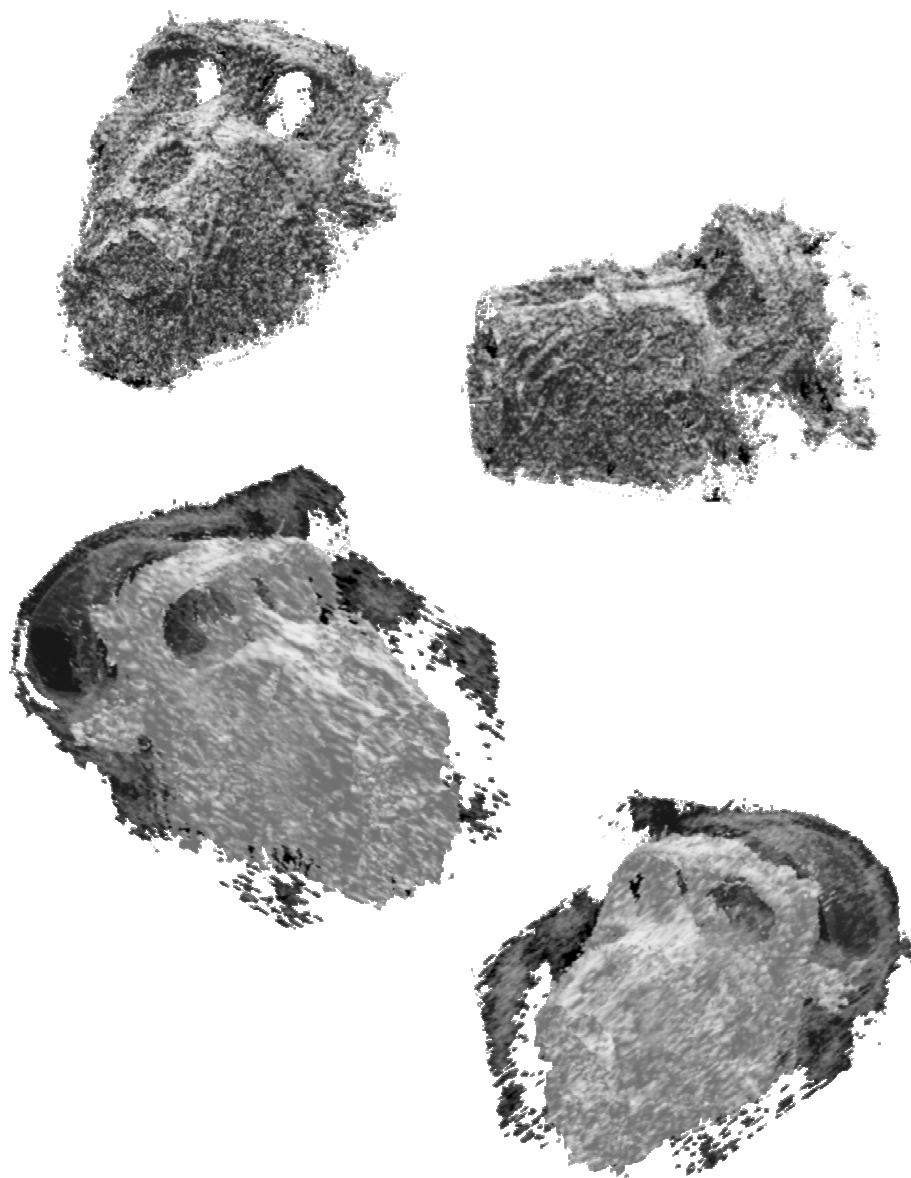


Figure 5.43. Surface renderings of the 256^3 voxels reconstruction of a baboon skull performed by the FDK+ plug-in in about 10 minutes on an Athlon 700 MHz system.

Plug-in	Plug-in options	Time (msec)	Speed scale
FDK	single proj.	264	100
FDK	multi proj.	253	104
FDK+	all on, ZP	85	168
FDK+	all on, no ZP	73	172
AInv	single proj.	271	97
TFDK	multi proj.	238	110

Table 5.5. Comparison of reconstruction performance for the four Strange Engine’s ActiveX plug-ins on an Athlon 600 MHz system. Time is the time required to backproject a 128×128 projection to a 128^3 voxel cube. Single proj. refers to per projection reconstruction as found in the BackProjectProjection method, multi proj. refers to BackProjectProjectionCollection reconstruction. ZP stands for zero padding.

5.7.3 Performance comparison

We have tested the plug-ins on different personal computers. Performance results are compared in Table 5.5 and plotted in Figure 5.44. Detailed performance of our optimised FDK+ plug-in is also reported in Table 5.6 and in Figure 5.45. Some comments are necessary:

- FDK+ can be up to 80 % faster than FDK.
- TFDK can be up to 10 % faster than FDK. Introducing FDK+ tricks into the TFDK plug-in could effectively cut by five reconstruction times.
- AInv does not take an important performance hit by 2D filtering. AInv can be made as fast as traditional FDK.

Also let us note how well reconstruction time scales with processor frequency: an Ahtlon at 600 MHz performs twice as fast as a Celeron at 300 MHz and so on. This means that performance seems not be influenced by other processor parameters such as cache as long as there is plenty of memory available.

Performance of the FDK+ plug-in is very aggressively influenced by each rendering trick. Axial interpolation skip and axial/angular symmetry add a boost of +20% and +40% respectively. Fixed-point math is however what improves the reconstruction performance the most: up to +60%. Putting all together a +80% speed gain can be achieved. When fixed-point math cannot be used, due to the 32 bit limitation, speed-up is reduced to about a +70%.

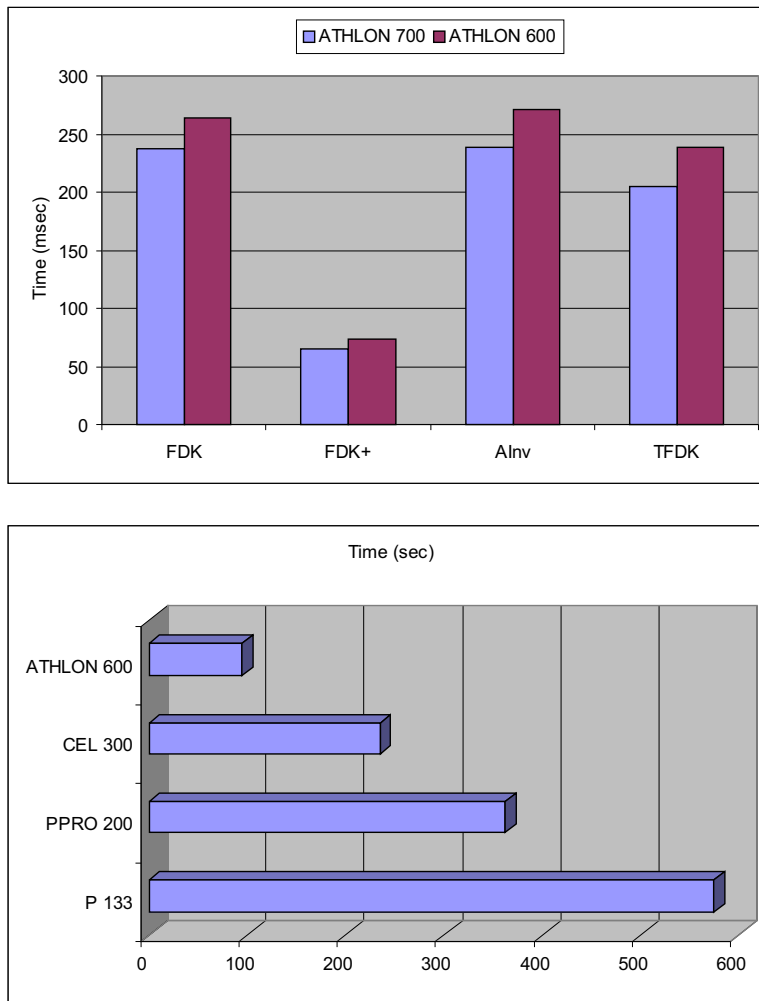


Figure 5.44. Comparison of reconstruction performance for the four Strange Engine's ActiveX plug-ins on different personal computers. Top: reconstruction time required to backproject a single 128×128 pixels projection on a 128^3 voxel cube. Bottom: reconstruction time required to reconstruct the baboon skull with the FDK+ plug-in on a $256 \times 30 \times 256$ voxel cube from 300 projections of 512×540 pixels.

5.7.4 Pre-processing performance

To end this discussion let us report some typical pre-processing times required in the AInv plug-in for filter computation and in the TFDK plug-in for the rebinning step:

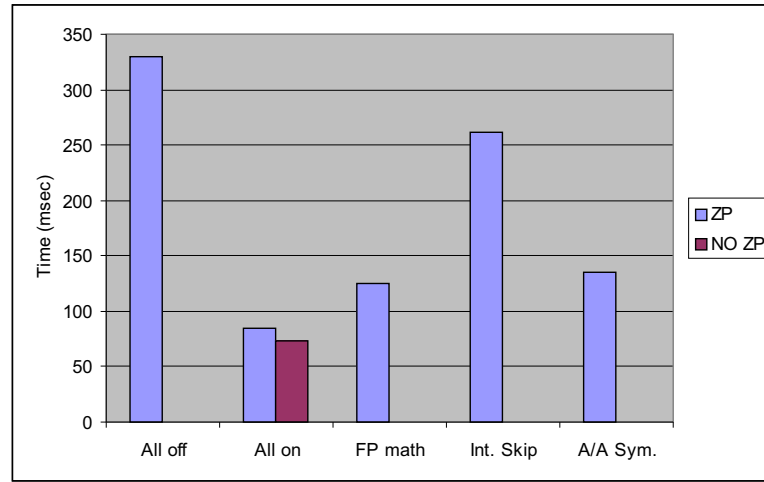


Figure 5.45. Impact of rendering “tricks” on the reconstruction time for the FDK+ plug-in. Time is the time required on an Athlon 600 MHz system to backproject a single 128×128 pixels projection on a 128^3 voxel cube.

Trick flags				Time (msec)	Speed scale
ZP	FP math	Int. Skip	Axial/Ang. Sym.		
				328	100
✓				330	99
✓		✓		261	121
✓			✓	135	159
✓	✓	✓	✓	85	174
	✓	✓	✓	73	178

Table 5.6. FDK+ plug-in reconstruction performance for an Athlon 600 Mhz system. Time is the time required to backproject a 128×128 projection to a 128^3 voxel cube. Flags refer to: usage of zero-padding for Fourier filtering, fixed-point math, axial interpolation skip and axial/angular symmetry.

Filter computation has been optimised as described in AInv documentation. As an example the calculation of a 128×128 pixels reconstruction kernel takes just 4 seconds on a Athlon 600 MHz system.

TFDK rebinning has not been optimised yet. Rebinning of 64 projections of 128×128 pixels with 12th order interpolation takes about 1 minute on a Athlon 600 MHz system.

Conclusions

This thesis describes an original software toolkit dedicated to cone-beam computed tomography, a new fast volumetric tomography technique which is going to be introduced in next generation medical scanners in a few years. We have motivated cone-beam computed tomography and presented the technical aspects related to scanner hardware and 3D cone-beam reconstruction. This analysis pointed out how difficult to implement and optimise 3D cone-beam reconstruction algorithms can be. New advanced software tools are hence needed to make algorithm research and development more efficient.

Our Strange Engine software visual toolkit provides a comfortable and user friendly environment for research and development of cone-beam computed tomography reconstruction algorithms. Strange Engine offers, through an easy to use graphical user interface, a series of services for simulation, storage and benchmarking of user-written reconstruction algorithms encapsulated into ActiveX components. We have described Strange Engine object-oriented architecture and have explained how it can improve the testing of custom reconstruction code. We have also examined four ActiveX components we have written and compared their performance for both simulated and real-world data on different personal computers. Our results have shown that cone-beam computed tomography reconstruction is indeed feasible even on common computer platforms.

References

- [1] ACR/NEMA, *Digital Imaging and Communications in Medicine (DICOM): Part 10. Media Storage and File Format for Data Interchange*, tech. rep., ACC-ACR-NEMA, 1994.
- [2] S. AGOSTINELLI, *Realizzazione di una estensione 3D CT proprietaria per simulatori radioterapici*, Tesi di specialità in Fisica Sanitaria, Scuola di Specializzazione in Fisica Sanitaria dell'Università degli Studi di Genova, Via Dodecaneso 33, July 1999.
- [3] —, *3D Tomographic Reconstruction ActiveX Style*, in The Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education, Austin, March 8-12 2000, The Association for Computing Machinery, Special Interest Group on Computer Science Education, p. 439. Third prize winner of the ACM International Graduate Student Research Contest 2000.
- [4] S. AGOSTINELLI AND F. FOPPIANO, *A 3D CT software platform for radiotherapy*, in The Use of Computers in Radiation Therapy, Heidelberg, May 22-25 2000, ICCR 2000.
- [5] —, *A prototype 3D CT extension for radiotherapy simulators*, Computerized Medical Imaging and Graphics, 25 (2001), pp. 11–21.
- [6] S. AGOSTINELLI, F. FOPPIANO, S. GARELLI, AND G. PAOLI, *Computational feasibility of 3D cone-beam tomography for radiotherapy*, in Ra-

- diotherapy & Oncology, vol. 56 Supplement 1, Istanbul, September 2000, European Society for Therapeutic Radiology and Oncology.
- [7] S. AGOSTINELLI AND G. PAOLI, *An object oriented fully 3D tomography toolkit*, Computer Methods and Programs in Biomedicine, (2001).
 - [8] C. J. (AXELSSON), *Direct Fourier Methods in 3D-Reconstruction from Cone-Beam Data*, PhD thesis, Department of Electrical Engineering, Linköpings Universitet, Linköping, 1994.
 - [9] G. BESSON, *CT fan-beam parameterizations leading to shift-invariant filtering*, Inverse Problems, 12 (1996), pp. 815–833.
 - [10] ———, *CT image reconstruction from fan-parallel data*, Medical Physics, 26 (1999), pp. 417–426.
 - [11] ———, *Fan-Parallel Cone-Beam Volumetric CT Reconstruction Algorithm Using a Flat Panel Detector*, in Proceedings of the 1999 International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Egmond aan Zee, June 23-26 1999, pp. 315–318.
 - [12] D. BOX, K. BROWN, T. EWALD, AND C. SELLS, *Effective COM*, Addison–Wesley, 1999.
 - [13] M. BRADY, *A Fast Discrete Approximation Algorithm for the Radon Transform*, SIAM Journal of Computing, 27 (1998), pp. 107–119.
 - [14] A. BRONNIKOV AND G. DUIFHUIS, *Wavelet-based image enhancement in x-ray imaging and tomography*, Applied Optics, 37 (1998), pp. 4437–48.
 - [15] C. CARLSSON, *Imaging modalities in x-ray computerized tomography and in selected volume tomography*, Physics in Medicine and Biology, 44 (1999), pp. R23–R56.
 - [16] P. CHO, K. LINDSLEY, J. DOUGLAS, K. STELZER, AND T. GRIFFIN, *Digital radiotherapy simulator*, Comp. Med. Imag. Gra., 22 (1998), pp. 1–7.
 - [17] M. CLINE, G. LOMOW, AND M. GIROU, *C++ FAQs*, Addison–Wesley, second ed., 1999.
 - [18] C. CORRY, V. MAYFIELD, J. CADMAN, AND R. MORIN, *COM/DCOM Primer Plus*, SAMS Macmillan Computer Publishing, 1999.

- [19] P. DANIELSSON, *From Cone-Beam Projections to 3D Radon data in $O(N^3 \log N)$ Time*, in Proc. of the IEEE Medical Imaging Conference, Orlando, October 27-31 1992, pp. 1135–37.
- [20] M. DEFRISE AND R. CLACK, *A Cone-Beam Reconstruction Algorithm Using Shift-Variant Filtering and Cone-Beam Backprojection*, IEEE Trans. Med. Imag., 13 (1994), pp. 186–195.
- [21] A. DEL FIANDRA, *Implementazione di un algoritmo veloce per la ricostruzione 3D cone-beam basato su rebinning*, 2001. Tesi di laurea in Scienze dell’Informazione, Università degli Studi di Genova.
- [22] R. DIETZ, *Die Approximative Inverse als Rekonstruktionsmethode in der Röntgen-Computertomographie*, PhD thesis, Mathematisch-Naturwissenschaftlichen Fakultät, Universität des Saarlandes, Saarbrücken, July 1999.
- [23] P. EDHOLM AND G. HERMAN, *Linograms in Image Reconstruction from Projections*, IEEE Trans. Med. Imag., MI-6 (1987), pp. 301–307.
- [24] J. ERIKSSON, *Problems and Experiments in Cone-Beam Tomography*, PhD thesis, Department of Electrical Engineering, Linköpings Universitet, Linköping, June 1998.
- [25] L. FELDKAMP, L. DAVIS, AND J. KRESS, *Practical cone-beam algorithm*, J. Opt. Soc. Am., A1 (1984), pp. 612–619.
- [26] M. FRIGO AND S. G. JOHNSON, *FFTW: An Adaptive Software Architecture for the FFT*, Proc. ICASSP, 3 (1998).
- [27] R. GORDON, R. BENDER, AND G. HERMAN, *Algebraic reconstruction techniques (ART) for three dimensional electron microscopy and X-ray photography*, J. Theor. Biol., 29 (1970), pp. 471–482.
- [28] P. GRANGEAT, *Mathematical Methods in Tomography*, Springer Verlag, 1991, ch. Mathematical framework of cone beam 3D reconstruction via the first derivative of the Radon transform, pp. 66–97.
- [29] —, *Analyse d’un Système d’Imagerie 3D par Reconstruction à partir de Radiographies X en Géométrie Conique*, PhD thesis, Ecole Nationale Supérieure des Télécommunications, 1997.
- [30] M. GRASS, T. KÖHLER, AND R. PROKSA, *3D cone-beam CT reconstruction for circular trajectories*, Phys. Med. Biol., 45 (2000), pp. 329–347.

- [31] B. GROH, J. SIEWERDSEN, D. DRAKE, J. WONG, AND D. JAFFRAY, *MV and kV cone-beam CT on a medical linear accelerator*, in The Use of Computers in Radiation Therapy, Heidelberg, May 22-25 2000, ICCR 2000, pp. 561–563.
- [32] H. GUAN AND R. GORDON, *A projection access order for speedy convergence of ART (algebraic reconstruction technique): a multilevel scheme for computed tomography*, Phys. Med. Biol., 39 (1994), pp. 2005–22.
- [33] G. HOUNSFIELD, *Computerized transverse axial scanning (tomography). Part I. Description of the system*, Br. J. Radiol., 46 (1973), pp. 1016–22.
- [34] J. HSIEH, R. MOLTHEN, C. DAWSON, AND R. JOHNSON, *An iterative approach to the beam hardening correction in cone beam CT*, Medical Physics, 27 (2000), pp. 23–29.
- [35] H. HU, T. PAN, AND Y. SHEN, *Multi-Slice Helical CT: Scan and Image Temporal Resolution*, in Proceedings of the 1999 International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Egmond aan Zee, June 23-26 1999, pp. 319–321.
- [36] M. INGERHED, *Fast Backprojection in Computed Tomography: Implementation and Evaluation*, PhD thesis, Department of Electrical Engineering, Linköpings Universitet, Linköping, 1999.
- [37] C. JACOBSON AND P. DANIELSSON, *Three-dimensional reconstruction from cone-beam data in $O(N^3 \log N)$ time*, Phys. Med. Biol., 39 (1994), pp. 477–491.
- [38] D. JAFFRAY, K. CHAWLA, C. YU, AND J. WONG, *Dual-beam imaging for online verification of radiotherapy field placement*, Int. J. Radiat. Oncol., Biol., Phys., 33 (1995), pp. 1273–80.
- [39] D. JAFFRAY, J. SIEWERDSEN, B. GROH, D. DRAKE, J. WONG, AND A. MARTINEZ, *Cone-beam computed tomography on a medical linear accelerator using a flat-panel imager*, in The Use of Computers in Radiation Therapy, Heidelberg, May 22-25 2000, ICCR 2000, pp. 558–560.
- [40] A. KAK AND M. SLANEY, *Principles of Computerized Tomographic Imaging*, IEEE Press, 1989.
- [41] D. KRUGLINSKI, G. SHEPHERD, AND S. WINGO, *Programming Microsoft Visual C++*, Microsoft Press, fifth ed., 1998.

- [42] H. KUDO AND T. SAITO, *Feasible cone beam scanning methods for exact reconstruction in three-dimensional tomography*, J. Opt. Soc. Am. A, 17 (1990), pp. 2169–83.
- [43] —, *Derivation and implementation of a cone-beam reconstruction algorithm for nonplanar orbits*, IEEE Trans. Med. Imag., 13 (1994), pp. 196–211.
- [44] K. LANGE AND R. CARSON, *EM reconstruction algorithms for emission and transmission tomography*, J. Comput. Assist. Tomogr., 8 (1984), pp. 306–316.
- [45] S. LI AND P. ECONOMOPOULOS, *Professional COM Applications with ATL*, Wrox Press Inc., Chicago, 1998.
- [46] A. LOUIS, *Medical imaging: state of the art and future development*, Inverse Problems, 8 (1992), pp. 709–738.
- [47] —, *Approximate inverse for linear and some nonlinear problems*, Inverse Problems, 12 (1996), pp. 175–190.
- [48] —, *A unified approach to regularization methods for linear ill-posed problems*, Inverse Problems, 15 (1999), pp. 489–498.
- [49] A. LOUIS AND T. SCHUSTER, *A novel filter design technique in 2D computerized tomography*, Inverse Problems, 12 (1996), pp. 685–696.
- [50] M. MAGNUSSON, *Linogram and Other Direct Fourier Methods for Tomographic Reconstruction*, PhD thesis, Institute of Technology, Linköpings Universitet, Linköping, 1993.
- [51] S. MANGLOS, G. GAGNE, A. KROL, F. THOMAS, AND R. NARAYANASWAMY, *Transmission maximum-likelihood reconstruction with ordered subsets for cone beam CT*, Phys. Med. Biol., 40 (1995), pp. 1225–41.
- [52] R. MARR, C. CHEN, AND L. P.C., *On two approaches to 3D reconstruction in NMR zeugmatography*, in Proc. Mathematical Aspect of Computerized Tomography, G. Herman and F. Natterer, eds., Oberwolfach, 1981, Springer-Verlag.
- [53] MATROX, *Matrox Imaging Library Programming Manual*, 1997.

- [54] MCCOLLOUGH AND ZINK, *Perfomance evaluation of a multi-slice CT system*, Medical Physics, 26 (1999), pp. 2223–2230.
- [55] S. MIDGLEY, R. MILLAR, AND J. DUDSON, *A feasibility study for mega-voltage cone beam CT using commercial EPID*, Phys. Med. Biol., 43 (1998), pp. 155–169.
- [56] F. NATTERER, *The Mathematics of Computerized Tomography*, John Wiley & Sons, Chichester, 1986.
- [57] S. NILSSON, *Application of fast backprojection techniques for some inverse problems of integral geometry*, PhD thesis, Department of Mathematics, Linköpings Universitet, Linköping, September 1997.
- [58] F. NOO, R. CLACK, AND M. DEFRISE, *Cone-beam Reconstruction from General Discrete Vertex Sets using Radon Rebinning Algorithms*, IEEE Trans. Nucl. Sci., 44 (1997), pp. 1309–16.
- [59] J. O’SULLIVAN, *A Fast Sync Function Gridding Algorithm for Fourier Inversion in Computer Tomography*, IEEE Trans. Med. Imag., MI-4 (1985), pp. 200–207.
- [60] J. PROSISE, *Programming Windows 95 with MFC*, Microsoft Press, 1997.
- [61] G. RAMACHANDRAN AND A. LAKSHMINARAYAN, *Three-dimensional reconstruction from radiographs and electron micrographs: application of convolution instead of Fourier transforms*, Proc. Nat. Acad. Sci. US, 68 (1971), pp. 2236–40.
- [62] O. RATIB, H. HOEHN, C. GIRARD, AND C. PARISOT, *PAPYRUS 3.0: DICOM compatible file format*, Med. Informatics, 19 (1994), pp. 171–178.
- [63] R. CLACK AND M. DEFRISE, *Cone-beam reconstruction by the use of Radon transform intermediate functions*, J. Opt. Soc. Am. A, 11 (1994), pp. 580–585.
- [64] J. RICHTER, *Advanced Windows*, Microsoft Press, third ed., 1997.
- [65] A. RIEDER AND T. SCHUSTER, *The approximate inverse in action with an application to computerized tomography*, SIAM J. Numer. Anal., 37 (2000), pp. 1909–29.
- [66] P. RIZO, P. GRANGEAT, P. SIRE, P. LEMASSON, AND P. MELENNEC, *Comparison of two three-dimensional x-ray cone-beam reconstruction algorithms*, J. Opt. Soc. Am. A, 8 (1991), pp. 1639–48.

- [67] K. RUCHALA, G. OLIVERA, E. SCHLOESSER, AND T. MACKIE, *Megavoltage CT on a tomotherapy system*, Phys. Med. Biol., 44 (1999), pp. 2597–2621.
- [68] S. SCHALLER, *Practical Image Reconstruction for Cone Beam Computed Tomography*, PhD thesis, Friederich-Alexander Universität, Erlangen-Nürnberg, July 1998.
- [69] S. SCHALLER, T. FLOHR, AND P. STEFFEN, *New Efficient Fourier-reconstruction Method for Approximate Image Reconstruction in Spiral Cone-Beam CT at Small Cone Angles*, in SPIE Medical Imaging Conference, Newport Beach, 1997.
- [70] ———, *An Efficient Fourier Method for 3-D Radon Inversion in Exact Cone-Beam CT Reconstruction*, IEEE Trans. Med. Imag., MI-17 (1998), pp. 244–250.
- [71] S. SCHALLER, F. NOO, F. SAUER, K. TAM, G. LAURITSCH, AND T. FLOHR, *Exact Radon Rebinning Algorithm for the Long Object Problem in Helical Cone-Beam CT*, IEEE Trans. Med. Imag., MI-19 (special issue on volumetric reconstruction of medical images) (2000), pp. 361–375.
- [72] M. M. SEGER, *Three-dimensional reconstruction from cone-beam data using an efficient Fourier technique combined with a special interpolation filter*, Phys. Med. Biol., 43 (1997), pp. 951–959.
- [73] L. SHEPP AND B. LOGAN, *The fourier reconstruction of a head section*, IEEE Trans. Nucl. Sci., NS-21 (1974), pp. 21–43.
- [74] M. SHIRAZI, *A combined cone-beam megavoltage CT scanner and portal imager for treatment verification in radiotherapy*, PhD thesis, Joint Department of Physics, Institute of Cancer Research and Royal Marsden NHS Trust, Sutton, March 1997.
- [75] M. SHIRAZI, P. EVANS, W. SWINDELL, S. WEBB, AND M. PARTRIDGE, *A cone-beam megavoltage CT scanner for treatment verification in conformal radiotherapy*, Radiotherapy and Oncology, 48 (1998).
- [76] H. SOWIZRAL, K. RUSHFORT, AND M. DEERING, *The Java 3D API Specification*, Addison–Wesley, 1998.
- [77] L. SPIES, *On scatter in megavoltage X-ray transmission imaging*, PhD thesis, Deutsches Krebsforschungszentrum, Medical Physics, Heidelberg, February 2000.

- [78] H. TURBELL, *Three-Dimensional Image Reconstruction in Circular and Helical Computed Tomography*, PhD thesis, Department of Electrical Engineering, Linköpings Universitet, Linköping, April 1999.
- [79] H. TURBELL AND P. DANIELSSON, *Fast Feldkamp Reconstruction*, in Proceedings of the 1999 International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Egmond aan Zee, June 23-26 1999, pp. 311–314.
- [80] G. WANG, P. CHENG, AND M. VANNIER, *Spiral CT: Current Status and Future Directions*, in Proceedings of 1997 SPIE, 1997, pp. 203–212.
- [81] G. WANG, T. LIN, P. CHENG, AND D. SHINOZAKI, *A general cone-beam reconstruction algorithm*, IEEE Trans. Med. Imag., 12 (1993), pp. 486–496.
- [82] G. WANG, D. SNYDER, J. O’SULLIVAN, AND M. VANNIER, *Iterative deblurring for CT metal artifact reduction*, IEEE Trans. Med. Imag., 15 (1996), pp. 657–664.
- [83] G. WANG AND M. VANNIER, *Encyclopedia of Electrical and Electronics Engineering*, John Wiley and Sons, 1998, ch. Computerized Tomography.
- [84] A. WATT, *3D Computer Graphics*, Addison–Wesley, third ed., 2000.
- [85] S. WEBB, *From the watching of shadows*, Adam Hilger, New York, 1990.
- [86] ———, *Non-standard CT scanners: their role in radiotherapy*, Int. J. Radiat. Oncol., Biol., Phys., 19 (1990), pp. 1589–1607.
- [87] K. WIESENT, K. BARTH, N. NAVAB, T. BRUNNER, AND W. SEISSLER, *Enhanced 3D-Reconstruction Algorithms for C-Arm Based Interventional Procedures*, in Proceedings of the 1999 International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, Egmond aan Zee, June 23-26 1999, pp. 167–170.
- [88] M. WOO, J. NEIDER, AND T. DAVIS, *OpenGL Programming Guide*, Addison–Wesley, second ed., 1997.
- [89] S. ZHAO, *Wavelet filtering for filtered backprojection in computed tomography*, Appl. Comput. Harmon. Anal., 6 (1999), pp. 346–373.
- [90] S. ZHAO AND G. WANG, *Feldkamp-Type Cone-Beam Tomography in the Wavelet Framework*, IEEE Trans. Med. Imag., (2000).

- [91] G. ZHENG AND G. GULLBERG, *A cone-beam tomography algorithm for orthogonal circle-and-line orbit*, Phys. Med. Biol., 37 (1992), pp. 563–577.
- [92] A. ZOLFAGHARI, *An iterative reconstruction algorithm in cone beam geometry: simulation and application in X-ray microtomography*, Nucl. Inst. Meth. Phys. Res. A, 378 (1996), pp. 326–336.