Stony Brook University

ACCELERATING COMPUTED TOMOGRAPHY ON COMMODITY GRAPHICS HARDWARE

A Dissertation Presented

by

Fang Xu

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook Univeresity

December 2007

Stony Brook University

The Graduate School

Fang Xu

We, the dissertation committee for the above candidate for the

Doctor of Philosophy degree,hereby recommend

acceptance of this dissertation.

Klaus Mueller - Dissertation Advisor
Associate Professor, Department of Computer Science

Arie Kaufman - Chairperson of Defense
Distinguished Professor, Department of Computer Science

Jerome Zhengrong Liang, Committee Member
Professor, Departments of Radiology, Computer Science and Physics

Gene Gindi, External Committee Member
Associate Professor, Departments of Radiology and Electrical Engineering

This dissertation is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Accelerating Computed Tomography on Commodity Graphics Hardware

by
Fang Xu

Doctor of Philosophy
in
Computer Science

Stony Brook University

2007

The task of reconstructing an object from its projections via tomographic methods is a time-consuming process due to the vast complexity of the data. For this reason, manufacturers of equipment for medical computed tomography (CT) rely mostly on special ASICs to obtain the fast reconstruction times required in clinical settings. Although modern CPUs have gained sufficient power in recent years to be competitive for 2D reconstruction, this is not the case for 3D reconstructions, especially not when iterative algorithms must be applied. The recent evolution of commodity programmable PC computer graphics boards (GPUs) has the potential to change this picture in a very dramatic way.

In this thesis, we first show that many types of CT algorithms, both iterative and non-iterative, can greatly benefit from the high degree of SIMD (Same Instruction Multiple Data) parallelism these platforms provide. By doing so, results of high-fidelity can be obtained at speedups of over an order of magnitude.

In addition to describing theories and implementation details, we further show dedicated solutions for resolving various challenges presented in cone-beam reconstruction using Feldkamp's method on GPU. We also propose optimization techniques specifically targeting the latest GPU architecture that enables the implementation of our streaming-CT notion.

Next, we use electronic microscopy tomography as an example to demonstrate the power of GPU's computational capability which is even more

important here due to EMT's extensive usage of iterative algorithms. Here a sinogram-based method was designed to achieve the maximum speedup and system scalability.

Last, a new rendering method D^2VR that can produce higher visualization quality than traditional volume rendering algorithms but suffers from high computational complexity, is accelerated by our rendering-driven rapid-CT framework to obtain near-interactive framerates.

To My Parents, Xijian Xu and Rusi Huang
My Grandma, Caiyun Chen
with My Love!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Various methods for 3D computed tomography (CT) reconstruction have been devised in the past three decades. While analytical approaches can be traced back to the Radon Transform [44], iterative algorithms seek to optimize some objective function, such as maximum likelihood or minimal error. All of these algorithms have in common a series of backprojection operations which dominate the computational cost. In addition, iterative algorithms also incorporate a series of forward projections, which incur similar computational expense. Thus, to be useful in clinical practice, the backprojections (and projections) have to be made as efficient as possible. However, this goal stands in stark conflict with the complexity of these operations. Each projection/backprojection has a complexity on the order of the size of the volume dataset, which is $O(N^3)$. This complexity is always present, unless recursive [5] or Fourier space [3] approaches are employed. These, however, have their own limiting constraints, since the need to reduce domain interpolation artifacts [5] [62] via oversampling increases the multiplicative constant in the complexity term. Here, we shall assume straightforward projection/backprojection in the spatial domain, at complexity $O(N^4)$. In this case, the only way to reduce the actual computational cost is to reduce the constant factor k

that relates the complexity $O(N^3)$ to the computational cost $k \cdot N^3$. Unfortunately, even the most clever programming with cache-aware algorithms and fast differencing schemes can only reach a limited peak performance, when implemented on general purpose CPUs. The general practice of pre-computing the weight matrices in iterative reconstruction can yield tremendous speedups in 2D reconstruction, but the memory cost involved makes the use of such pre-computed matrices infeasible for 3D reconstruction. For this reason, a number of commercial custom-hardware based solutions have become available. One such approach (by TeraRecon, Inc.) uses an ASIC (Application Specific Integrated Circuit), while another (by Mercury Systems, Inc.) uses an FPGA (Field Programmable Logic Array). Both reach very impressive speeds for Feldkamp's cone-beam algorithm [26], but they do not implement any iterative algorithms, such as EM (Expectation Maximization) [82] or ART (Algebraic Reconstruction Technique) [33], which are preferable for functional imaging, such as SPECT and PET. The special-purpose proprietary boards are also quite expensive, in the range of 5- digit $-figures, and furthermore, their static custom hardware design makes them inflexible for modification and generalization. Hence, while it is economically viable to augment already expensive tomography scanners in need for stable and proven reconstruction algorithms with such hardware boards, less expensive and more flexible solutions are desirable for researchers and experimental clinicians.

When defining an appropriate platform, it helps to realize that the projection/backprojection operations, as well as the other operations involved in the grid updates and correction computations, are straightforward voxel- and pixel-based operations, which have few dependencies and are usually computed as array operations within a long loop. A very suitable platform for these kinds of calculations are vector processors or massively parallel architectures [12]. Vector processors view their input data as streams, which are combined by operators to produce an output stream. Also, while CPUs must decode every instruction in a loop, vector processors execute the entire array operation within one instruction, amortizing the cost for the single instruction decode over the entire loop. Paired with extremely high memory bandwidth, programmable vector processors can accomplish array-based computations at impressive speeds. Unfortunately, vector processors, such as the Cray supercomputer family, are expensive machines and very few people have access to them. An exciting

new development in this regard is the emergence of a main-stream computing platform that bears many features of vector processors - Graphics Processors Units(GPUs). Graphics applications fit the SIMD (Same Instruction Multiple Data) programming model of vector processors well. They typically consist of largely independent compute and data-intensive operations - the screen-rasterization of large numbers of texture-mapped polygons - which expose both small-grain (per-polygon calculations) and large-grain (per-polygon list calculations) parallelism. Graphics-heavy applications, such as computer games or engineering design, require ever-increasing complex scenes to be rendered at rates of 30 frames per second, and these large, consumer-driven demands have led to an unparalleled growth in the development of platforms that can satisfy these needs. The development of graphics hardware grows so fast that the chip performance doubles every 6 months, tripling Moore's law. These GPU boards gain their speed by devoting significantly more chip real estate to the computational engine than a general-purpose CPU, such as the Intel Pentium processor. They implement what is referred to as a stream processor, which has become a widely researched computing paradigm for high performance computing [45]. By casting the projection/backprojection operations as well as all other CT calculations in terms of stream operations (or fragment rasterization operations, in graphics parlance), we can exploit these affordable mainstream architectures to achieve rapid CT.

1.1 Computed Tomography Methods

1.1.1 Feldkamp's Algorithm

The standard Feldkamp filtered backprojection (FDK) algorithm was devised by Feldkamp, Davis, and Kress [26]. Different from exact 3D cone-beam methods, such as [36] [35], which generalize the 2D filtered backprojection by using an inverse 3D Radon transform, Feldkamp's algorithm is approximate (or non-exact) but for small cone angles, it manages to produce results very close to what can be obtained from exact methods. It does not require non-truncated projections that are difficult to obtain in clinical practice and is simple and efficient to perform, which makes it

by far the most popular reconstruction method for 3D circular cone-beam tomography. Basically, the Feldkamp algorithm consists of three stages: projection-space filtering, backprojection, and volume-space weighting. This is captured in the following three equations:

$$f(r) = \frac{1}{4\pi^2} \int_0^{2\pi} \frac{d^2}{(d + r \cdot x_\phi)^2} \hat{P}_\phi(r) d\phi \tag{1.1}$$

where

$$\hat{P}_\phi(r) = \hat{P}_\phi(Y(r), Z(r)), \quad Y(r) = \frac{r \cdot y_\phi}{d + r \cdot x_\phi} D, \quad Z(r) = \frac{r \cdot z_\phi}{d + r \cdot x_\phi} D \tag{1.2}$$

and

$$\hat{P}_\phi(Y, Z) = \frac{D}{\sqrt{D^2 + Y^2 + Z^2}} P_\phi(Y, Z) * * g(Y) \tag{1.3}$$

In these equations (also shown in Figure 1.1), a voxel is denoted by a vector $r = (x, y, z)$ in the (reconstruction) volume space defined by (X_v, Y_v, Z_v), with Y_v being the rotation axis. The elements on the projection (detector) plane oriented at ϕ are represented by $P_\phi(Y, Z)$, where Y and Z represent an element's spatial location in detector coordinates. The vector X is orthogonal to the detector plane and connects the detector center with the source S. The two orthogonal vectors Y and Z complete the 3D coordinate system of the detector space, which is related to (X_v, Y_v, Z_v) by ways of a transformation matrix composed of gantry rotation and possible gantry warp. Finally, the distances from the source to the rotation center O and the detector center are defined as d and D, respectively.

Equation (1.3) constitutes the projection-space filtering, while equation (1.2) represents the backprojection operation, that is, the mapping of a voxel to a location on the projection plane and the subsequent assignment of the value interpolated there. Finally, equation (1.1) integrates the backprojection contributions for a given voxel over all projections, properly weighted in volume space.

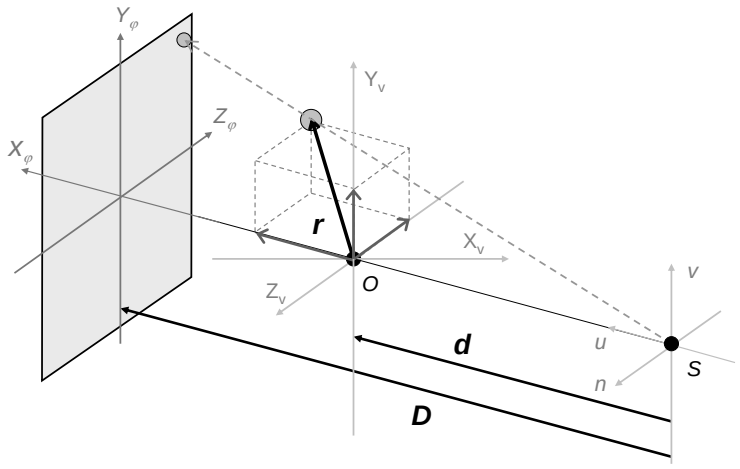


Figure 1.1: The Feldkamp filtered backprojection algorithm.

1.1.2 Iterative Methods

By defining the volume and projections (images) as discretized grids/arrays, we can model the imaging procedure as the following equation:

$$p_i = \sum_{j=1}^{N} v_j \cdot w_{ij} \qquad i = 1, 2, ..., M \tag{1.4}$$

where p_i is the approximate line integral on the i^{th} ray/pixel, v_j are those voxels involved in the ray traversing and w_{ij} is the weight factor that voxel v_i contributes to the value of p_i, while M and N are the number of pixels/voxels. For large values of M and N presented in the tomography study, it is difficult to solve the equation system by directly figuring out the inverse weight matrix. Therefore, iterative methods are employed by repeatedly modifying the estimated volume, approximating the ground truth (given images) and eventually minimizing the measurement

difference to reach convergence. Popular iterative algorithms include algebraic methods, such as SIRT (Simultaneous Iterative Reconstruction Technique) [32], SART (Simultaneous Algebraic Reconstruction Technique) [2], as well as statistical methods such as ML-EM (Maximum Likelihood / Expectation Maximization) [82] and their subset variants.

The definition of algebraic methods is shown in equation (1.5). Here acquired projections are pre-grouped into sets of certain size and the current estimated volume v_k keeps being updated by the computed correction calculated from all projections within the set. The update from all subsets completes an iteration of the reconstruction. Within each set update, the correction is computed as the difference between the measured projection (ground truth) pixels p_i and the estimated reprojection pixels r_i, weighted by the sum of weights, which is added to current k^{th} volume $v^{(k)}$ to yield the latest estimation $v^{(k+1)}$. The reprojection r_i can be calculated by simulating the ray integration procedure with raycasting approach (ray-driven method), where the discretized volume is sampled along the ray path at equidistant steps with appropriate kernels deployed to obtain interpolated values that are accumulated in the end to form the pixel. The above process repeats until convergence, where the difference between projections and reprojections is small enough. Here, the number of projection contained in each subset P_{set} can be adjusted, where SART and SIRT are special cases when P_{set} is set to 1 and the total number of projections, respectively. λ is the relaxation factor that is used to prevent instability when the subset size is small.

$$v_j^{(k+1)} = v_j^{(k)} + \lambda \frac{\sum\limits_{p_i \in P_{set}} \left(\frac{p_i - r_i}{\sum\limits_{l=1}^{N} w_{il}} \right)}{\sum\limits_{p_i \in P_{set}} w_{ij}} \qquad r_i = \sum_{l=1}^{N} w_{il} \cdot v_l^{(k)} \tag{1.5}$$

Derived from statistical model, ML-EM and its variant OS-EM (Ordered Subsets Expectation Maximization) [40] performs well on models that conforms to the Poisson distribution. It differs from algebraic methods in computing the correction with division of the projection values

over the reprojection values and updating the current estimated volume through multiplication (see equation (1.6)).

$$v_j^{(k+1)} = \frac{v_j^{(k)}}{\sum_{p_i \in P_{set}} w_{ij}} \left(\sum_{p_i \in P_{set}} \left(\frac{p_i}{r_i} \right) w_{ij} \right) \qquad r_i = \sum_{l=1}^{N} w_{il} \cdot v_l^{(k)} \tag{1.6}$$

1.2 Graphics Hardware

CPUs have been designed to provide a maximum of flexibility for the programs that run on them. In CPUs, micro-instructions are fetched and executed in a sequential fashion. This allows for any type of program flow, but performance does not benefit per se from the more regular program flow inherent to loops. The only benefit comes from the fact that loops tend to localize data access, and CPUs typically provide two levels of fast memory caches of significant size to exploit this data locality. These caches pre-fetch and store data close to previously used data items and thus provide faster access to them when needed soon after.

GPUs, on the other hand, are not known to be overly flexible in terms of program flow, but they are extremely powerful when it comes to the repetitive data processing often exhibited in loops. In fact, this form of computational model is very typical for graphical objects, which consist of large meshes of polygons, with each such polygon being defined as a set of three or more vertices. Here, each such vertex is represented as a 3D floating point (x, y, z) coordinate triple. To view such a graphical object from an arbitrary position, all vertices must first be transformed into viewing-space (defined by the viewing direction), orthographically or perspectively, and then reconnected to form the (projected) polygons. If only these connections (or edges) are displayed, then the object appears as a wireframe model. However, if all screen pixels subtended by a given polygon are assigned a value, then a process called rasterization is invoked. More generally speaking, rasterization produces data (called *fragments*), which are associated with the corresponding screen pixels. Also, apart from shading effects, one can add more surface detail by assigning each vertex a certain location in one or more images or *textures*. Each polygon

then subtends a closed (polygonal) region in this texture, and the rasterization process provides the coordinates (as part of the fragment) into these textures (Figure 1.2). Since these do not necessarily coincide with a discrete texture pixel, the texture must be interpolated (either with linear or nearest neighbor interpolation) to yield the value assigned to or combined with the screen pixel (Figure 1.3).

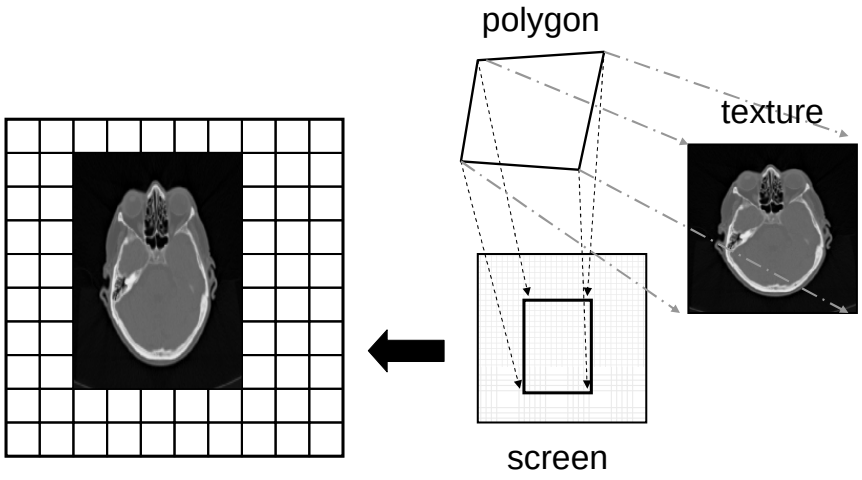


Figure 1.2: Rasterization.

Thus there is a strict computational pipeline governing the display of a graphical object: (1) the vertex transformations, (2) the generation of fragments, and (3) the computations imposed on the fragments, such as texture interpolations or more complex tasks, which result in the RGBA colors assigned to the screen pixels (see Figure 1.4). Since screen pixels are numerous and typically independent, a high degree of SIMD (Same Instruction Multiple Data) parallelism exists. Further, since this basic graphics pipeline has no loop dependencies, the objects simply *stream* across the pipeline, starting as a list of vertices, which generate fragments, which in turn form the basis for computing the visual attributes assigned to the corresponding screen pixels. GPUs are hence a *streaming architecture*, processing massive sets of graphics primitives, i.e., polygons and textures, in a highly parallelized fashion. This dedicated computational model greatly

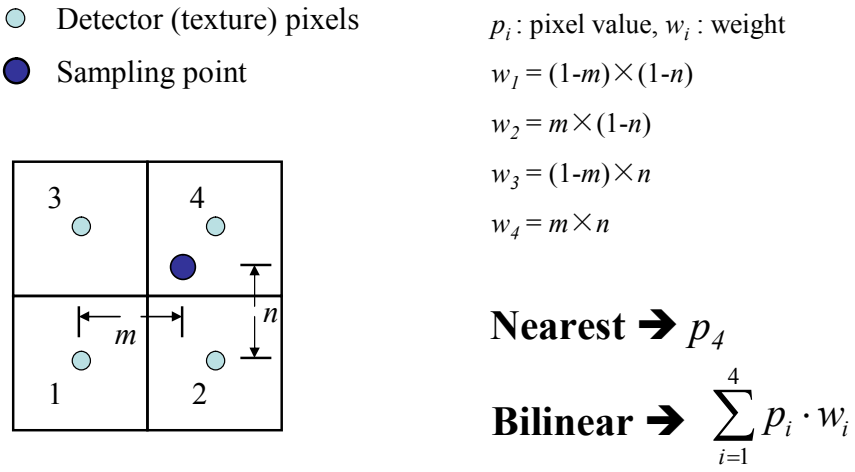


Figure 1.3: Interpolation.

simplifies the control circuitry which intensifies data processing and also enables optimized memory access for maximum throughput with significantly reduced latency.

More concretely, vertices and fragments that share similar operations constitute the elements of a stream, where these similar operations on them are implemented via user-defined shader programs, one at the vertex stage and one at the fragment stage of the pipeline (see the orange boxes shown in Figure 1.4). When rendering begins, a stream of vertices generated on the CPU, or stored in GPU vertex memory, is passed into the GPU's geometry processing stage. These vertices usually carry multiple properties to describe the target object, such as coordinate (xyz), color (RGBA), normal, etc. A *vertex shader* can then be applied to transform and map the 3D coordinates of each vertex to homogenous space and eventually screen space, while possibly complex vertex lighting effects can also be calculated. Next, rasterization, which is performed in fast special GPU hardware circuitry, reassembles each polygon in screen space and quickly fills the space enclosed by it. This generates a stream of fragments, which map to corresponding screen pixels. This fragment stream is processed in the *fragment shader*, which performs a series of user-defined per-pixel calculations. In this effort, the fragment shader also gives rise to another data

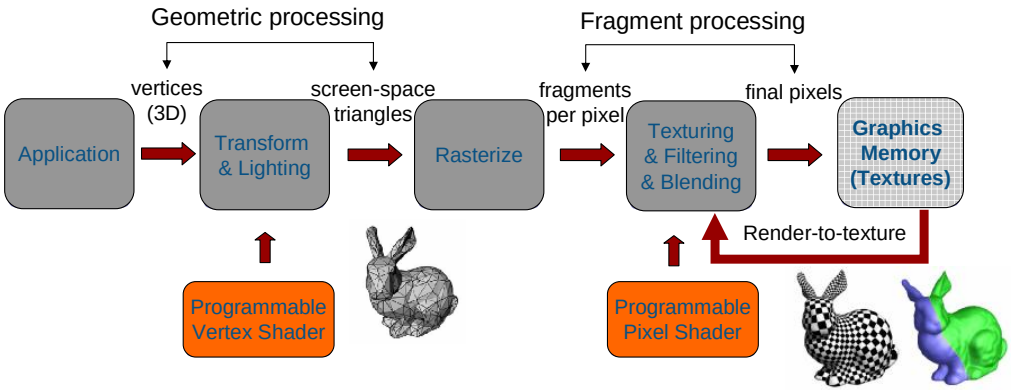


Figure 1.4: The graphics pipeline.

stream, consisting of textures, which are stored in GPU texture memory and provide the data used in the fragment shader. After fragment program completion, the result is written to the corresponding pixels on the screen (or the *framebuffer*).

Textures are the dominating data representation and storage primitive used by GPUs. A texture is essentially a 1-3D array of data. Each texture element can be represented in a certain format, ranging from a scalar to a vector of up to four components. In a graphics context, the latter usually describes a Red/Green/Blue/Alpha (RGBA) value, but note that for general-purpose GPU-computing this 4-channel representation can provide an additional 4-way parallelism. This parallelism can be exploited by packing data sharing common operations together, and good speedups can be obtained since GPUs tend to be very efficient at processing this type of vectors. Current GPUs also support various texture precisions ranging from basic 8-bit integer to advanced 32-bit floating point. There are, however, still constraints on the use of textures in certain combinations of formats and precisions. For example, 4-channel 16-bit fixed point textures are not yet supported. Further, for most GPU-based applications, 2D textures are the preferred primitive since they are naturally optimized by the hardware and support efficient read/write actions. 3D textures, in contrast, are used mainly in (read-only) visualization applications, since they do not support direct-write operations. When write-access is desired they are better represented as stacks of 2D textures.

The latest generations of GPUs (for example, the NVIDIA GeForce 8800

GTX) provide 128 stream processors, with a theoretical memory bandwidth throughput of 86.4GB/s and a pixel fill rate of 13.8GPixel/s. This is equivalent to a computational power of 520GFlops, compared to a number of 25Gflops and 12GB/s that can be sustained on a dual-core Pentium 4 3GHz CPU. The performance growth rate of GPUs is consistently maintained at about triple of the Moore's law while its cost is successfully kept low thanks to the intense competitions on the market. Finally, dual-GPU boards or even quad-GPU designs have also become recently available. These designs use the Scalable Link Interface (SLI) to connect the multiple GPUs together, and such a setup can double or quadruple, respectively, the amount of performance of a single GPU, on a single PC.

1.3 GPU for Medical Imaging Computing

The emergence of modern Graphics Processing Units (GPU) not only dramatically enhances the 3D object rendering speed and quality that enables the production of high-quality images in real time, it also propels a new wave of commodity high-performance computing (HPC) research in the last few years. This new trend is called General Purpose Computing on GPUs (GPGPU) [74], where the computational power of GPU is no longer limited to computer game design but extended into a more general field of scientific computing. This is enabled by the new features carried by the current graphics hardware whose underlying SIMD (Single Instruction/Multiple Data) architecture has evolved to a very flexible and high precision pipeline which contains programmable engines with high-level language support and full 32-bit floating point computation capabilities. These added features greatly reduce the effort users need to spend for harnessing the GPU's huge computational power that derives from simplified control circuitry and reduced memory latency.

Two fundamental computational kernels, conjugate gradient and multigrid solvers were implemented [9] for solving partial difference equations (PDE). Linear algebraic calculations in numerical computing were also mapped onto GPUs [9] [53] for solving sparse matrices and multi dimensional finite difference equations, often used in fluid dynamics. Harris et al. [38] [37] accelerated the performance of the couple map lattice (CML) model for rendering physical phenomena such as reaction/diffusion and

3D cloud, while Li et al. [58] experimented on Lattice Boltzmann model. They both achieved interactive simulation speed on sophisticated physical models. Scientific visualization as well as volume rendering research that involves processing on massive data also benefit from GPU's power [23] [49] [77]. In the following we will concentrate on reviewing the previous work contributed to the investigation of using GPU for medical imaging applications.

In the 1990s, only midrange workstations, such as the SGI Octane or Onyx, which are available at a cost of over $20,000, had the level of graphics hardware necessary for CT reconstruction. The first works that sought to exploit this hardware for the acceleration of CT was by Cabral et al. [13], who implemented an analytical Feldkamp-type algorithm, and Mueller and Yagel [70], who described the implementation of an iterative method – the Simultaneous Algebraic Reconstruction Technique (SART) [2]. With the emergence of low-cost PC-based graphics hardware of similar capabilities than that of the SGI, more recent work by Chidlow and Möller [16] focused on this platform. Using a NVIDIA GeForce 4, these authors implemented another iterative algorithm the maximum likelihood expectation maximization method (ML-EM) [82], and its faster cousin, Ordered-Subsets EM (OS-EM) [40]. However, all of the above approaches suffered from the circumstance that the graphics hardware they employed only had integer-arithmetic at 8-bit precision (PC) or 12-bit precision (SGI). This severely limited their accuracy and performance. With integer arithmetic at this precision one cannot perform the accumulation operations of the projection and backprojections in hardware. Also, the short precision limits the accuracy of the (sometimes small) grid corrections in iterative algorithms. For this reason, the accumulation operations had to be performed outside the GPU, on the CPU, which involved expensive data transfers between these two entities. A (virtual) 16-bit extension of the precision could be achieved by splitting high-precision calculations among two of the four color channels (Red, Green, Blue, Alpha) [70]. A similar mechanism could also be employed to facilitate a subset of the accumulations (16 for a 4-bit virtual extension) in hardware [16]. Although quite effective, this mechanism was only partially accurate since it dropped the lower 8 bits of the high-end channel. The reconstruction of the Shepp-Logan brain phantom using the hardware-accelerated SART algorithm as shown in [70] is clearly not satisfactory for the 0.5% contrast and only acceptable for higher contrasts $(1\%, 2\%)$, at speedups in the range of 35-68 when compared to a

CPU-based method.

A further limitation of these older generations of graphics hardware was their lack of programmability. For example, divisions are necessary for the normalization step in the iterative algorithms, but were not supported on these older platforms. The major leap forward made by new generations of GPUs is the fact that they offer programmability at floating point precision at two stages in the graphics pipeline. A direct consequence of this added functionality is that now the entire reconstruction can be performed within the GPU, at CPU precision. Thus, there is no longer a need to export and import data from and to the CPU, which overcomes the severe bottlenecks inherent in these data transfers. Also, a direct consequence of the GPU-resident computation is that the generated data can be easily visualized. Since in "normal" settings the GPU's main job is the rendition of graphics images, one can simply inject a volume rendering or a volume slicing cycle into the reconstruction and then map the resulting image to the screen-visible portion of the GPU's framebuffer.

Finally, the emergence of GPUs (for example, the NVIDIA FX series) with full programmability and 32-bit floating point precision enabled complete GPU-resident CT reconstruction, with both analytical and iterative methods [98] and with large data [69] at a fidelity comparable to CPU-based methods. Following these more fundamental works were a number of papers targeting specific CT applications, all with impressive speedup factors. Kole and Beekman [50] accelerated the ordered subset convex reconstruction algorithm, Xue et al. [102] accelerated fluoro-based CT for mobile C-arm units, and Schiwietz et al. [80] accelerated the backprojection and FFT operations employed for MR k-space transforms. In [97], the GPUs helped to achieve a fast calculation of PSF (point-spread function) matrices used for OS-EM in SPECT imaging. One of our papers, [99] used the GPU-accelerated CT framework to enable interactive volume visualization directly from a full set of projection data. Finally, a real-time reconstruction framework was proposed in [101] and advanced effects involved in emission CT were implemented in [100].

Besides these GPU-based efforts, there have also been recent works that exploited other high performance computing platforms for CT, in particular the Cell BE processor [43]. While the performance is quite good, the Cell BE does not fit (at least not currently) the profile of a commodity platform. Furthermore, it turns out that GPUs are in fact an excellent match for CT reconstruction, as the (back-) projection operations of CT have much

in common with traditional graphics operations, which receive super-fast hardwired acceleration support in GPUs. Exploiting this fact represents a major source of speedups, resulting in overall superior performance of our approach. Further, given the time-critical interaction of the various GPU pipeline components within our graphics-oriented framework, a careful load-balancing among these components is also needed to maximize the performance.

Chapter 2

Theory and Framework

2.1 Theoretical Considerations

We use the volume representation of Lewitt [57] and others, who model a volume as a collection of point samples, positioned at the grid points. In this model, values at off-grid positions are estimated from the grid samples via interpolation with some kernel function. While Lewitt has proposed the use of pre-integrated Bessel functions (so-called *blobs*) for this purpose, we will employ linear functions, which have also found widespread use in backprojectors and, as we shall see later, lend themselves well for implementation in graphics hardware.

Before describing how GPUs can be exploited to perform all calculations occurring in a variety of popular CT algorithms, it is helpful to establish a common notation for these. For this purpose, let us assume a volumetric object composed of a material with attenuation function $\mu(x, y, z)$ and separately irradiated by two imaging modalities: transmission and emission X-ray. In transmission X-ray (see Figure 2.1a), the source is located outside the object and a ray emanating with initial (source) intensity Q_0, traversing the object, and collecting in bin (u, v) of a 2D detector oriented at angle ϕ will be recorded with intensity:

$$C_\phi^Q(u, v) = Q_0 \cdot e^{-\int_0^L \mu(t)dt} \tag{2.1}$$

Here, t is a parametric variable defined along the ray, and L is the distance between the source and the detector bin. On the other hand, in emission X-ray (see Figure 2.1b) the sources are the metabolic activities $E(x, y, z)$ located inside the object, each attenuated by the material between it and the detector. Integrating over all metabolic sources along the ray orthogonal to detector bin (u, v) gives the energy:

$$C_\phi^E(u, v) = \int_0^L E(s) \cdot e^{-\int_0^s \mu(t)dt} \tag{2.2}$$

where s is a parametric variable defined along the ray, and L and t are defined as in (2.1). To illustrate the amenability of CT for vector processing, let us choose an appropriate notation. For this, we denote $C_i^Q = C_\phi^Q(u, v)$ and $C_i^E = C_\phi^E(u, v)$ for $0 \leq i < M_\phi$, where M_ϕ is the total number of pixels (rays) in the projection acquired at detector angle ϕ.

By further setting $q_i = -log(C_i^Q/Q_0)$, the transmission X-ray equation (2.1) can be written as follows:

$$q_i = \int_0^L \mu(t)dt \tag{2.3}$$

Since we would like to reconstruct the values at the volume grid positions, it makes sense to rewrite (2.3) in an alternative, voxel-centric form:

$$q_i = \sum_{j=0}^{N^3-1} \mu_j w_{ij} \tag{2.4}$$

Here, a w_{ij} is the weight with which the object voxel j (of value μ_j) contributes to detector pixel i (with final value q_i). These weights are determined by the interpolation filter [57] and the integration rule. On the other hand, the emission X-ray equation (2.2) indicates that the emissive quantity $E(s)$ is attenuated by the materials μ between site s and the detector. Returning to the voxel-centric representation of (2.4), now using the E_j as the values stored at the grid points, the projected emissive contribution originating at any s is:

$$e_i(s) = \sum_{j=0}^{N^3-1} E_j w_{ij}(s) \tag{2.5}$$

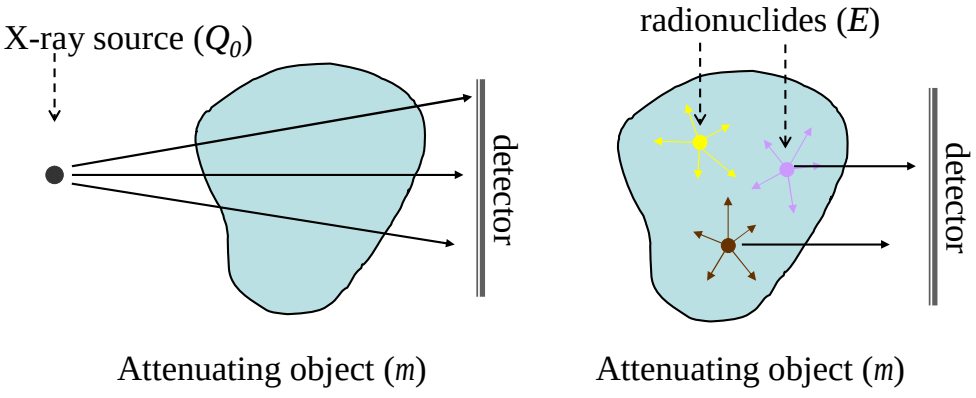


Figure 2.1: (a)Transmission imaging: an external X-ray source emits X-rays. (b)Emission imaging: internal radionuclides emit photons at sites of biochemical (metabolic) activity. Both are attenuated by the object's densities.

Note that here the $w_{ij}(s)$ are not only given by the voxel weights, as in the transmission case, rather they now also incorporate the attenuation integral up to s. The equation for the total projected emissive energy is then given as:

$$e_i = \int_{s=0}^{L} (\sum_{j=0}^{N^3-1} E_j w_{ij}(s)) \tag{2.6}$$

Re-ordering the integral yields:

$$e_i = \sum_{j=0}^{N^3-1} E_j \int_{s=0}^{L} w_{ij}(s)ds = \sum_{j=0}^{N^3-1} E_j w_{ij(a)} \tag{2.7}$$

Here, the $w_{ij(a)}$ combines the voxel weights of (2.4) and the attenuation factors. The subscript (a) is used to denote that the w_{ij} contain a factor for attenuation correction. We observe that this equation is very similar

to (2.4). Therefore we conclude that we can project the emission volume E with methods similar to those that reconstruct the attenuation volume μ, given knowledge about the more complicated $w_{ij(a)}$ (in case we do not care about the ray attenuation, we simply use the basic w_{ij} of (2.4) for the projection of the emission volume). Thus, by generalizing (E_j, m_j) to v_j and (e_i, q_i) to p_i we can formulate a generalized projector and, by exchanging the roles of v_j and p_i, we obtain a generalized backprojector:

$$p_i = \sum_{j=0}^{N^3-1} v_j w_{ij} \qquad v_j = \sum_{i=0}^{M_\phi-1} p_i w_{ij} \tag{2.8}$$

In the following, we will denote the projection operator in the first part of (2.8) by $P_\phi(V)$ and the backprojection operator in the second part of (2.8) as $B_\phi(I)$. Here, V is the volume data vector (subject to reconstruction), I is an image data vector, and the projectors/backprojectors are matrices operating on them. However, in our framework the matrix elements, i.e., the w_{ij}, will not be stored explicitly, but computed on the fly, using the interpolators in the rasterization hardware. We will now express the various reconstruction methods by ways of these operators.

In the Feldkamp algorithm [26] the w_{ij} are multiplied by a depth correction factor during backprojection (see Figure 1.1 and Equation (1.1)):

$$w_{ij(d)} = w_{ij} \frac{D}{\sqrt{D^2 + Y^2 + Z^2}} \tag{2.9}$$

Here, Y and Z return a voxel y and z coordinate and D is the distance from the source to the rotation center. Finally, $w_{ij(d)}$ is the depth-weighted w_{ij} in (2.9). Using our shorthand notation, the backprojection process is written as:

$$V = \sum_{\phi \in S} B_{\phi(d)}(I_\phi) \tag{2.10}$$

where I_ϕ is the image obtained from the scanner at angle ϕ. The iterative method SART [2] updates the grid on a projection-basis. This turns out to be more convenient than the related (ray-based) ART [33] when used in conjunction with texture mapping hardware. Using our notation, SART's

grid update equation is:

$$V = V + \frac{B_\phi(\lambda \frac{I_\phi - P_\phi(V)}{P_\phi(W)})}{B_\phi(W)} \tag{2.11}$$

where λ is a relaxation factor. $P(W)$ and $B(W)$ denote the projection and backprojection of the weights for normalization, which can be performed using a unity I and V, respectively. Finally, the OS-EM [40] algorithm is written as:

$$V = \frac{V}{\sum_{\phi \in OS} B_{\phi(a)}(W)} \sum_{\phi \in OS} B_{\phi(a)}(\frac{I_\phi}{P_{\phi(a)}(V)}) \tag{2.12}$$

where OS is one of the ordered subsets of S. We observe that, computation-wise, the only real difference among these reconstruction methods is how the results of the projection/backprojection operators are combined. However, these combination operations are straightforward vector calculations. We will now discuss how equations (2.10) - (2.12) can be efficiently realized in GPU hardware.

2.2 Implementation

In graphics hardware, just as images, volumes can also be represented as textures. There are two choices: a stack of 2D textures or a single 3D texture. While both allow projection, there is currently no convenient and efficient facility that would allow a backprojection into a 3D texture. We therefore store a volume as two stacks of 2D textures (see Figure 2.2), one each for projections along the X and the Z main viewing axes. We do not need a Y-major texture stack, since we only acquire data in a circular orbit about the Y-axis.

2.2.1 Forward Projection

Perspective (cone-beam) projection is a straightforward operation with 2D textures (we shall consider parallel-beam a subset of perspective). We

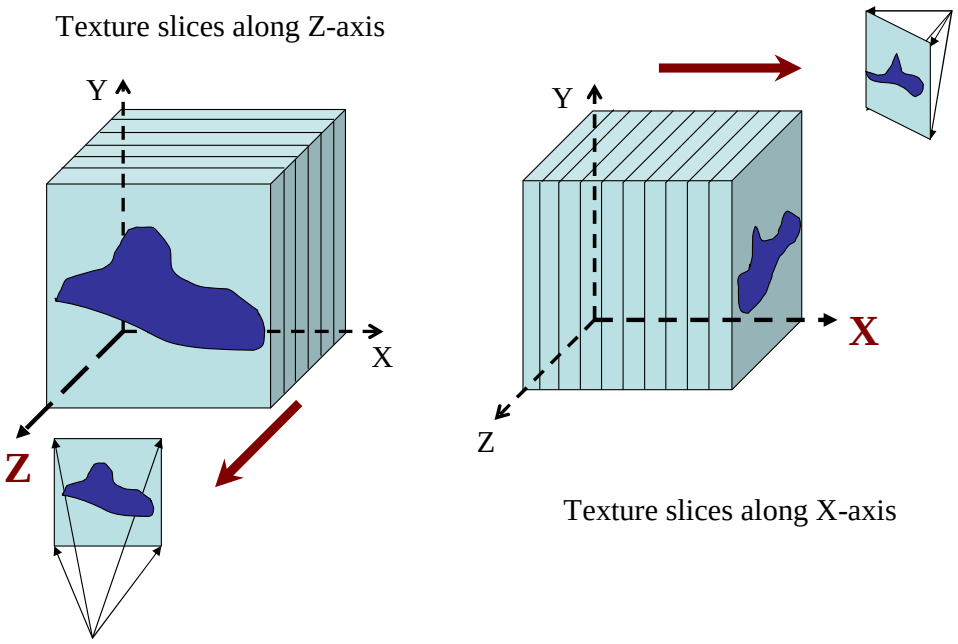


Figure 2.2: Volume representation as a stack of 2D textures

can approximate equations (2.3) and (2.4) using 2D textures as follows (see Figure 2.3):

$$\hat{q}_i = \sum_{k=0}^{L/\Delta t} \mu(k\Delta t)\Delta t \rightarrow \hat{q}_i = \sum_{k=0}^{N-1} \sum_{l=0}^{N^2-1} \mu_{lk} w_{ilk} \Delta t_i \tag{2.13}$$

The left part is a discretized form of (2.3), which is further approximated into the form of the right part by adapting (2.4), grouping the N^2 voxels within each of the N volume slice textures. The true voxel index j can be derived from the index lk used in (2.13), where k indexes the slice and l the voxels in the slice. There are two approximations here. First, the integration is now a discrete trapezoidal one, with the stepsize Δt varying across the slice (denoted as Δt_i in (2.13)). Since Δt is never greater than $\sqrt{3}$ this is a reasonable approximation. Second, the w_{ilk} are computed via bilinear interpolation – a square neighborhood of 4 slice voxels will contribute to each. (see Figure 1.3). We can compensate for the varying Δt

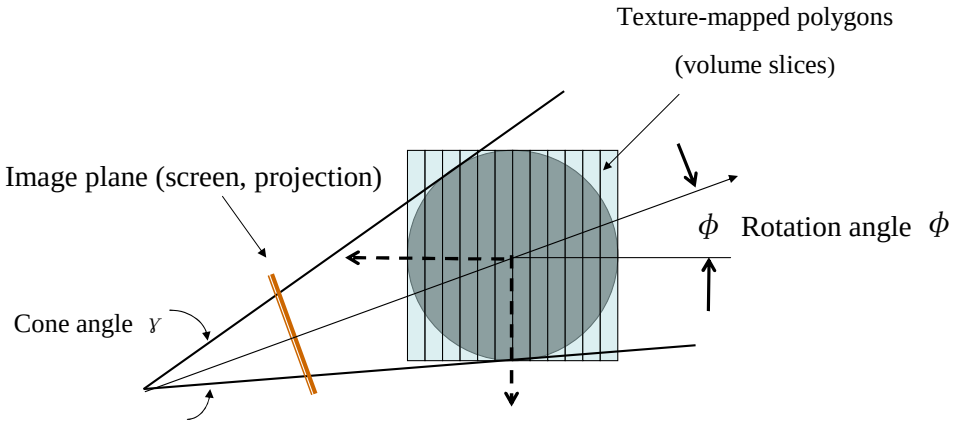


Figure 2.3: 3D forward projection with 2D texture slices (for simplicity of illustration, only the 2D case is shown)

by pre-computing a sampling interval texture for each orientation angle in the set and multiplying this texture with the texture of the projection result, on the GPU.

We now look into the projection of the emission volume. If attenuation is not modeled, then the mechanism of (2.13) will readily apply, simply substituting (m, q) by (E, e). However, attenuation modeling can improve reconstruction results considerably (see e.g. [47]), and our hardware-based approach can realize this efficiently. We first discretize equation (2.2), in a fashion similar to the first part of (2.13) (here, we use our notational identity $C_\phi^E(u, v) = C_i^E = e_i$):

$$C_i^E = \int_{s=0}^{L} E(s) \prod_{n=0}^{s-1} e^{-\int_{t=n} \mu(t)dt} \tag{2.14}$$

$$\approx \int_{s=0}^{L} E(s) \prod_{n=0}^{s-1} (1 - \int_{t=n}^{n+1} \mu(t)dt)ds \tag{2.15}$$

$$\approx \sum_{k=0}^{L/\Delta s} E(k\Delta s) \prod_{n=0}^{k-1} ((1 - \mu(n\Delta t))\Delta t)\Delta s \tag{2.16}$$

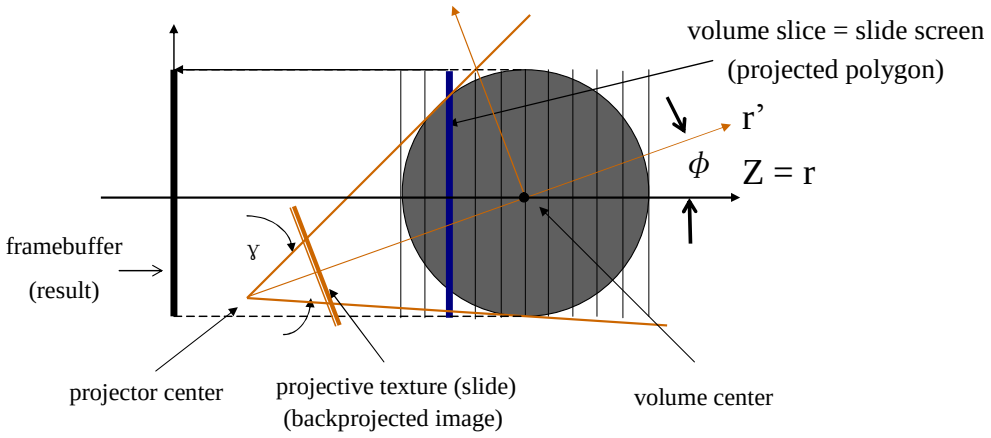


Figure 2.4: Backprojection of a correction image texture onto a volume slice

Here, we map the m volume to a range $[0.0, ..., 1.0]$. The error of the Taylor series approximation of the exponential is within reasonable bounds since the interval Δt is never greater than $\sqrt{3}$. The final expression in (2.14) allows us to convert (2.7) into the texture slice-based representation, similar to the second part of (2.13):

$$\hat{e_i} = \sum_{k=0}^{N-1} \sum_{l=0}^{N^2-1} E_{lk} w_{ilk(a)} \Delta s_i \tag{2.17}$$

where $w_{ilk(a)}$ is the product of the interpolation weight w_{ilk} for the emissions in slice k, and the product of the slice-wise interpolated attenuations up to slice $(k-1)$:

$$w_{ilk(a)} = w_{ilk} \prod_{n=0} (1 - \mu(n\Delta t))\Delta t = w_{ilk} \prod_{n=0}^{k-1} (1 - \sum_{m=0}^{N^2-1} (\mu_{nm} w_{imn}))\Delta t_i \tag{2.18}$$

Here, the w_{imn} are also determined by the interpolation filter, and n indexes the slices and m the voxels in the slices. We compute the attenuation

part of the $w_{ilk(a)}$ in (2.17) recursively, via implementation of a variant of the familiar volume rendering *front-to-back compositing* equation [60]:

$$e_f = e_f + e_b(1 - \mu_b) \qquad \Rightarrow \qquad e_f = e_f + e_b \cdot t_f \tag{2.19}$$
$$(1 - \mu_f) = (1 - \mu_f)(1 - \mu_b) \qquad \Rightarrow \qquad t_f = t_f \cdot (1 - \mu_b) \tag{2.20}$$

Here, the two columns hold equivalent expressions, with t denoting transparency. The e_b and μ_b are the newly interpolated values, while e_f and μ_f are the recursive variables. Note, that in contrast to volume rendering, e can grow past 1.0. In the end, e holds the emission volume projection, properly attenuated by μ. Equation (2.19) states that we must maintain a texture buffer for transparency t and one for emission e, and that we must multiply t with the newly interpolated emission. Two texture volumes are required, one for the emission volume that is being reconstructed and one for the attenuation volume, possibly obtained via a prior transmission CT.

An alternative form is *back-to-front compositing*:

$$e_b = e_b(1 - \mu_f) + e_f = e_b t_f + e_f \tag{2.21}$$

The new (back) emission e_b is calculated by adding the previous back emission e_b to the newly interpolated front emission e_f. But before it is added it must be attenuated by the interpolated transparency t_f at this point.

2.2.2 Back-projection

The grid updates in equations (2.9)-(2.12) all have a similar backprojection term, which can be written as:

$$dv_j = \sum_{i \in I_\phi} d_i w_{ij} \tag{2.22}$$

where dv_j is the update to a voxel j, derived from grid update factors d_i. The w_{ij} are determined similarly as outlined for the projection case. For

emission tomography, matched projector/backprojector pairs [103] that use full attenuation modeling (and other effects) only for the projection phase, but not for the backprojection phase, have been proposed and can be implemented by using w_{ij} in place of $w_{ij(a)}$. However, it is desirable to use the same w_{ij} in both projection and backprojection. The 2D slice texture approach allows us to achieve the desired equivalent mapping by using *projective textures* [81] for the backprojection, which is illustrated in Figure 2.4. Essentially, projective textures work similar to a slide projector. The backprojected image forms the "slide", which is perspectively projected onto the "screen" formed by a polygon that is placed at the location of the volume slice to be updated. The "slide projection" is then "viewed" in parallel projection mode on the screen. The perspective transform is given by the viewing geometry at which the projection was originally obtained from the scanner. Using this mapping, the weight with which a voxel j contributes to a projection image pixel i is identical to the weight that a correction d_i coinciding with i has on j. Projective textures can be implemented by filling the hardware texture mapping matrix with the appropriate values in a vertex program (see [70] for further detail) and performing the actual projective mapping in a fragment program.

The attenuation weighting can be implemented in two ways: (1) as an interleaved projection/backprojection procedure, or (2) as a projection followed by a backprojection. The former can be formulated as follows, with D being initialized as the grid update image (computed from scanner image and projection), DV being the volume that accumulates the updates, and μ being the attenuation volume:

```
for each volume slice k = 0,...,N−1, going in front-to-back order
   backproject D onto DV_k
   project μ_k onto D executing blending D = D · (1 − μ_k)
```

For the alternative, second method, we pre-compute a new set of textures D_k by first rendering all projections of the μ_k slices (with the blending), saving them in texture memory, and then performing all backprojections using these D_k. This saves the somewhat expensive projection and backprojection context switches, but it consumes more storage in texture memory. The algorithm is written as:

```
Initialize D_0 to D
```

```
for each volume slice k = 1,...,N−1, going in front-to-back order
   project μ_k onto D_{k−1} executing blending D_k = D_{k−1} · (1 − μ_k)
for each volume slice k = 0,...,N−1, going in front-to-back order
   backproject D_k onto DV_k
```

It is left to mention that both SART and EM also require a volume that stores the w_{ij} for each updated voxel, to be used later for normalization. In practice, we have found that SART does not require normalization, due to the bilinear filter weights, while for EM we can just normalize by the number of projections in the subset (similar to [16]). However, if attenuation correction is applied, a weight volume W must be accumulated, which we accomplish by backprojecting a two-channel texture (D, μ) into a two-channel texture stack (DV, W).

2.2.3 Pixel-wise Components

In the iterative schemes, both the computation of the correction texture D and the new state of the voxel textures V (i.e., E or μ) are pixel-wise operations, implemented as simple texture blends. Denoting an original, acquired projection as OP, the calculated projection as P, and a projection of the weights as W, the (vector or stream) calculation of D can be written as:

1. D = DIV(SUB(OP, P), W) for **SART**
2. D = DIV(OP, P) for **EM**

The voxel update after backprojection of all projections in the set has occurred can be written as:

1. $V = V + DV$ for **SART**
2. $V = (V \cdot DV)/W$ for **EM**

where W is the accumulated weight volume, if attenuation correction is used.

In the iterative algorithms, our use of two stacks of 2D textures will lead to inconsistencies if one stack of textures is updated by ways of backprojection but the other is not. Therefore we must update a texture stack

whenever its projection proceeds an update of the other texture stack. This is frequently the case since two subsequent projections should be close to orthogonal to maximize the rate of convergence. We implement each texture stack as a single large 2D texture, with one tile per slice. We can then accomplish a stack update by adding an up-to-date column in the source stack texture to the corresponding out-of-date column in the destination stack texture (see Figure 2.5).

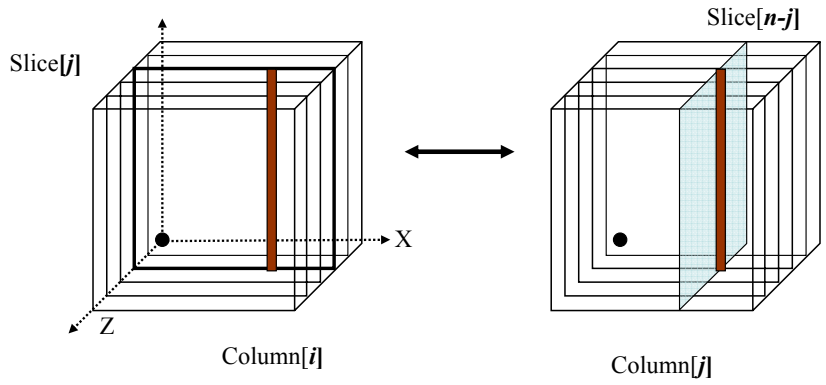


Figure 2.5: Texture stack updates when the major projection direction switches from one stack to the other

2.2.4 Parallel Execution via the RGBA Color Channels

All three CT algorithms can exploit the inherent parallelism offered by the four color channels. In Feldkamp's filtered backprojection, we may pack the data of four orthogonal projections into the RGBA channels, since they share the same projection matrices. This gives rise to a 4-way (RGBA) texture stack, one each for the four 90° intervals processed in the four channels. Then, after all backprojections are completed, the volume is assembled by adding the data in the four channels, using a technique similar to the stack update described before. This speedup strategy requires that the projections were acquired at orthogonal angle increments.

In SART, we cannot project/backproject orthogonal projections in parallel due to the projection-wise volume update. Here, we *fold* the upper and lower halves of the volume and the projections into the RG channels, while the BA channels are used to accumulate the weights in the projection phase. The two halves share the same projection matrices, just reflected about the mid-line. Projections, backprojections, and texture stack updates all use this decomposition and the complete volume is only assembled at the end, by merging the RG channels. The Feldkamp algorithm can also use this folded partition, which reduces the number of required texture stacks to two.

The unattenuated EM algorithm can employ a 4-times parallelism, either using the folded decomposition in conjunction with two orthogonal projections or the unfolded 4-way scheme. However, the latter incurs significant overhead, both for volume assembly and volume distribution from and to the four color channels each time a subset has been processed, which only amortizes when the subsets are large. We therefore chose to use the folded 2-way approach. Note that this EM parallelism poses certain constraints on the composition of the subset. The parallelism in the attenuated EM algorithm is only two-fold since two channels are needed for each projection/backprojection to hold (μ, e) and (DV, W), respectively.

2.3 Modeling Scattering Effects

2.3.1 Background

Scattering effect is another important phenomena to be modeled in functional imaging, which usually occurs along the photon paths [59]. These are mostly due to photon interactions with the traversed tissue, leading to stray photons that are eventually counted in detector bins neighboring the intended one. Proper modeling of these effects in the forward projection step of iterative algorithms, such as OS-EM, can yield a significantly more accurate estimate of the required grid correction in the subsequent back projection step. This, in turn, leads to faster and more accurate reconstruction of the emission image.

Here, the amount of detector blurring a particular emission site causes

is influenced by the local scattering properties of the traversed tissue as well as the distance of this site to the detector. The most appropriate technique to model scattering is a Monte-Carlo simulation, using the current reconstruction instance as the source. However, such a simulation is (currently) computationally infeasible to conduct within an iterative procedure, at least when it comes to clinical routine. A good approximation can be obtained by the slice-by-slice blurring method that has been proposed for emission CT [4] as well as for volume rendering [48]. In both approaches, the blurring is guided by the local scattering properties of the tissue. The scattering as well as the attenuation modeling rely on the existence of a prior CT scan to provide an estimate of the attenuation and scattering properties of the tissue.

To describe the approximate, non-Monte Carlo technique used to model the scattering, let us first consider a scattering event in a differential volume patch dV. Here, depending on the patch's scatter properties, a ray of photons suffers some amount of diffusion, which can be modeled by a suitable diffusion function kernel, such as a box, tent, or Gaussian, where the extent of this kernel is determined by the amount of local scattering potential, estimated from a map indexed by the underlying CT data. This local model can be extended to a global one by recursion, where a detector-aligned slice buffer is advanced step-by-step from the rear of the volume to the detector, and the scatter-diffusion process is modeled, at each such step, by convolving the slice image with a variable extent blurring function. Here, the size of the filter is dependent on the local scattering properties (higher scattering coefficients widen the filter). Also at each step, an emission volume slice is interpolated and added to the advancing slice buffer. In fact, this scheme can be combined with the attenuation modeling.

Modeling these effects in the forward projection provides a better estimate of the actual image generation process, given the current state of the emission volume under reconstruction. This in turn provides a better estimate of the required grid correction for back-projection (which favors reconstruction speed and quality). This estimate can then be back-projected via regular means (without modeling attenuation and scattering effects) or by including these effects. The former leads to the concept of unmatched projector/back-projector pair, which has been frequently used.

2.3.2 Forward Projection

Figure 2.6a shows the blurring kernel in the ideal detector-aligned buffer configuration. Here the recursive blurring (the red kernel) can occur at any arbitrary buffer (blue dashed lines) distance Δs(Δs = 1 is reasonable). This is the configuration used when a 3D texture is interpolated.

Next, Figure 2.6b shows the texture-slice (or axis) aligned situation (assuming the texture slice distance is 1). In this case the scattering will occur at a larger distance and therefore the width of the blurring kernel should be scaled up accordingly, that is, by a factor $1/\cos\alpha$, where α is the rotation angle of the detector.

Due to the perspective (cone-beam) distortion, the scattering on one side of the principal direction (here the right half-cone as seen from the scattering source) will occur in a larger distance before entering the slice buffer than the other (the left half-cone), see Figure 2.6c. We can correct for this as well, by additionally scaling the two kernel side lobes according to their subtended half-cone volumes, V_1 and V_2.

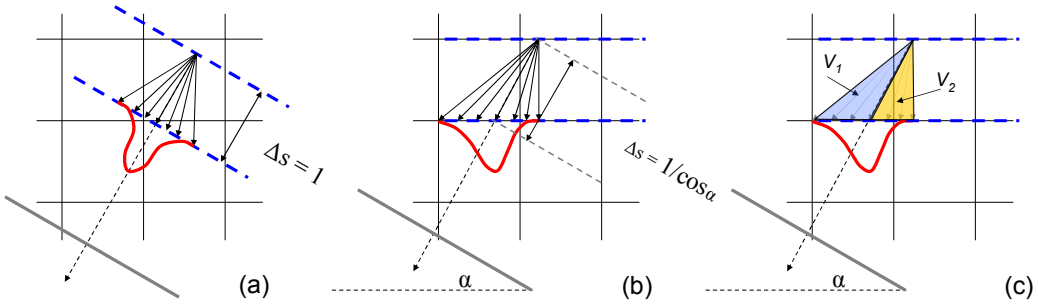


Figure 2.6: Modeling scattering effects using geometry dependent blur kernels. (a)ideal case when 3D texture is used; (b)compensated for 2D textures; (c)compensated for cone angle.

Platform	Algorithm	Volume	Projection	1 iter.	3 iter.
SGI-hardware	SART	128^3	80×128^2	1.1 min	3.1 min
PC - GPU	SART/OS-EM	128^3	80×128^2	1s/2.5s	3.1s/5s
PC - GPU	SART/SIRT	256^3	160×256^2	7.5s/5.0s	22s/15s

Table 2.1: Reconstruction timings for various configurations. Nvidia Geforce 8800GTX is employed to obatin GPU timings.

2.4 Results

2.4.1 Basic Framework

Table 2.4.1 lists the timings obtained in our experiments. All GPU results were produced on a Athlon 2.2GHz PC hosting a NVIDIA 8800 GTX GPU with 768MB on-board memory. For results shown in Figure 2.7 we employ a 3D version of the Shepp-Logan brain phantom (of size 128^3) [71] at the original 0.5% contrast level to demonstrate reconstruction quality. Figure 2.7 shows slices across the reconstructed phantoms, while line plots provide further insight into reconstruction fidelity and noise. These plots are obtained from the intensity profile along the line indicated in Figure 2.7a [and Figure 2.7g for EM]. Finally, Figure 2.8 presents a formal error analysis, where we compute the correlation coefficient (CC) of the phantom with the reconstruction, both within the entire skull and within an ellipsoid just enclosing the three small "tumor" at the bottom. We also compute the coefficient of variation (CV) over four ellipsoidal regions with uniform content. Here, the CV for region i is $CV_i = \delta_i/\mu_i$, where μ_i is the average and δ_i is the standard deviation of the region's voxel values [71] [78].

We observe that a current, fairly optimized CPU implementation, using first-order (linear) interpolation filters, runs at about 1-2 order of magnitude faster than the (older) SGI texture mapping hardware implementation. Meanwhile, the GPU reconstruction quality (Figure 2.7d) is nearly equivalent to that obtained with the software implementation (Figure 2.7c), which was infeasible with the integer-based SGI hardware (Figure 2.7b). We suspect that the remaining artifacts for GPU SART may be due to the

coarser sampling due to the fixed slice distance and the trapezoidal interpolation rule.

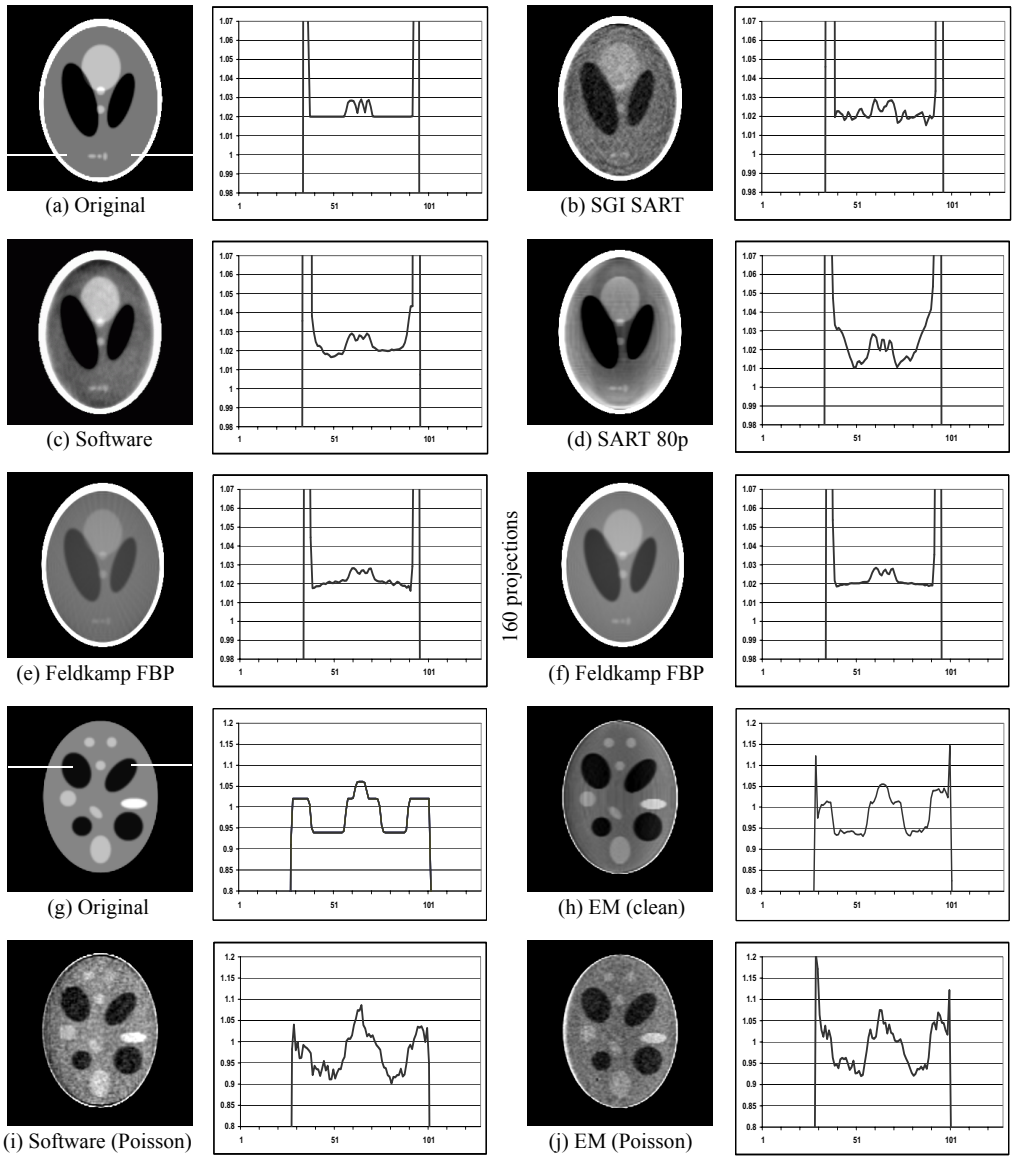


Figure 2.7: Slice across the 3D (a)-(f) Shepp-Logan brain phantom and (g)-(j) ellipsoid phantom, reconstructed as a 128^3 volume from a set of 80 analytically computed projections of size 128^2 each [160 projections for (f)]. The iterative algorithms used three iterations of projection/backprojections. (i) and (j) are from the ellipsoid phantom, with random Poisson noise added to the projections. The plots show the intensity profiles across the center of three small ellipsoids near the bottom of the phantom in (a)-(f) and near the top of the phantom (g)-(j), as indicated by the white line in (a) and (g).

The GPU implementation of Feldkamp FBP produces nearly perfect results. As indicated in Figure 2.7e and f, doubling the number of projections from 80 to 160 can eliminate the residual streak artifacts that are common for FBP when less than projections are used. In order to test the EM implementation, we designed a volume more suited for emission studies. We also added Poisson noise to the analytically computed projections. The phantom consists of ellipsoids with four times the original Shepp-Logan contrast (Figure 2.7g). Figure 2.7h shows an EM reconstruction without noise, and Figure 2.7i and j show a CPU and GPU reconstruction, respectively, from noisy data. The GPU-based EM implementation yields fairly good results from both clean and noisy data. We notice some faint ringing and some elevated level of noise in the both GPU-reconstructed datasets. We attribute this again to the coarser sampling rate and the trapezoidal integration rule. The line plots of Figure 2.7 and the error metrics of Fig-

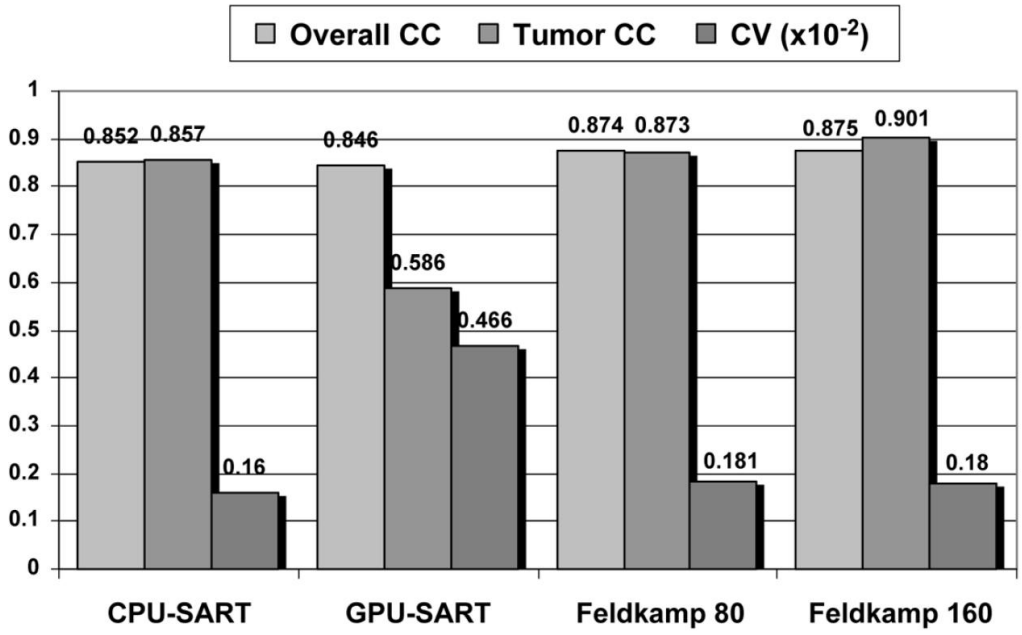


Figure 2.8: CC and background CV for various reconstructions.

ure 2.8 indicate that the GPU reconstructions have generally greater noise and structural artifacts, but only at moderate levels, but distinguish the phantom features quite well. The remaining artifacts are greater for the iterative algorithms than for Feldkamp, which we believe is due to the fact

that projection errors accumulate through the iterative process.

2.4.2 Advanced Effects

Further results are obtained to validate the accuracy and performance of the attenuation correction and scattering effects. We employed a different 3D phantom composed of a set of ellipsoids of varying emission values.

Figure 2.9 compares the slice-by-slice blurring results obtained with the detector-aligned vs. the axis-aligned projector for five orientations α: $0°$, $10°$, $20°$, $30°$, $40°$. Both simulations use the same scattering model and volume. Next to the projections we show the difference images and the intensity profile along the line indicated in the first image. A good fit is observed. We also computed the overall RMS error and found it to be between 1-2% of the maximum value in a projection. We conclude from this study that the axis-aligned projector is well suited for the recursive scatter simulation we use in our GPU-accelerated iterative emission reconstruction framework.

Figure 2.10 shows a set of representative projections obtained with our simulator, with emissions only, (E) emission with attenuation (E+A), emission with scattering (E+S), and emission with scattering and attenuation (E+S+A). Scattering creates substantially more blur, while attenuation weakens the projections of emissions traversing highly attenuating material, both with and without scattering.

Figure 2.11 shows a representative slice from a 3D reconstruction of our phantom (10 SART iterations), for various modeling scenarios arranged into rows. The first column shows the reconstructed slice when the (row) effects are not modeled, while the second column shows the slice when modeling took place. In the final column we show the intensity profiles for the line indicated in the first image. We compare the original phantom profile (solid grey), the profile obtained when the effect has not been modeled (dotted) and the profile when the effect has been modeled (solid black). We see that in all cases the contrast is greatly improved, the features are sharper, and the profiles match the original better when the effect is modeled. We also observe that without attenuation/scattering (A+S) modeling, the small ellipsoid between the two large ones in the upper third of the phantom can not be detected.

	Projection	Backprojection	1 iter.	10 iter.
Transmission CT	0.7s	0.7s	1.7s	18s
Emission CT (matched)	1.2s	7.3s	8.7s	90s
Emission CT (unmatched)	1.2s	0.7s	2.2s	23s

Table 2.2: Reconstruction timings for the transmission and emission CT on 8800GTX: 128^3 volume, 160 projection angles. For emission CT, attenuation correction and scattering effects are modeled.

Finally, Table 2.4.2 shows the detailed performance of running reconstruction with both attenuation correction and scattering effects on a Nvidia 8800 GTX. We observe that adding attenuation and scattering only to the projector (in an unmatched projector/back-projector reconstruction framework) has a relatively small impact on performance (less than a factor of 2). This verifies the observations of [4]. The results shown in Figure 2.11 were all obtained with configuration. On the other hand, modeling these effects for the back-projection operator is about 10 times more expensive. Overall, it takes 5 times longer if attenuation and scattering are modeled in both he projection and backprojection stages (in the matched projector/back-projector).

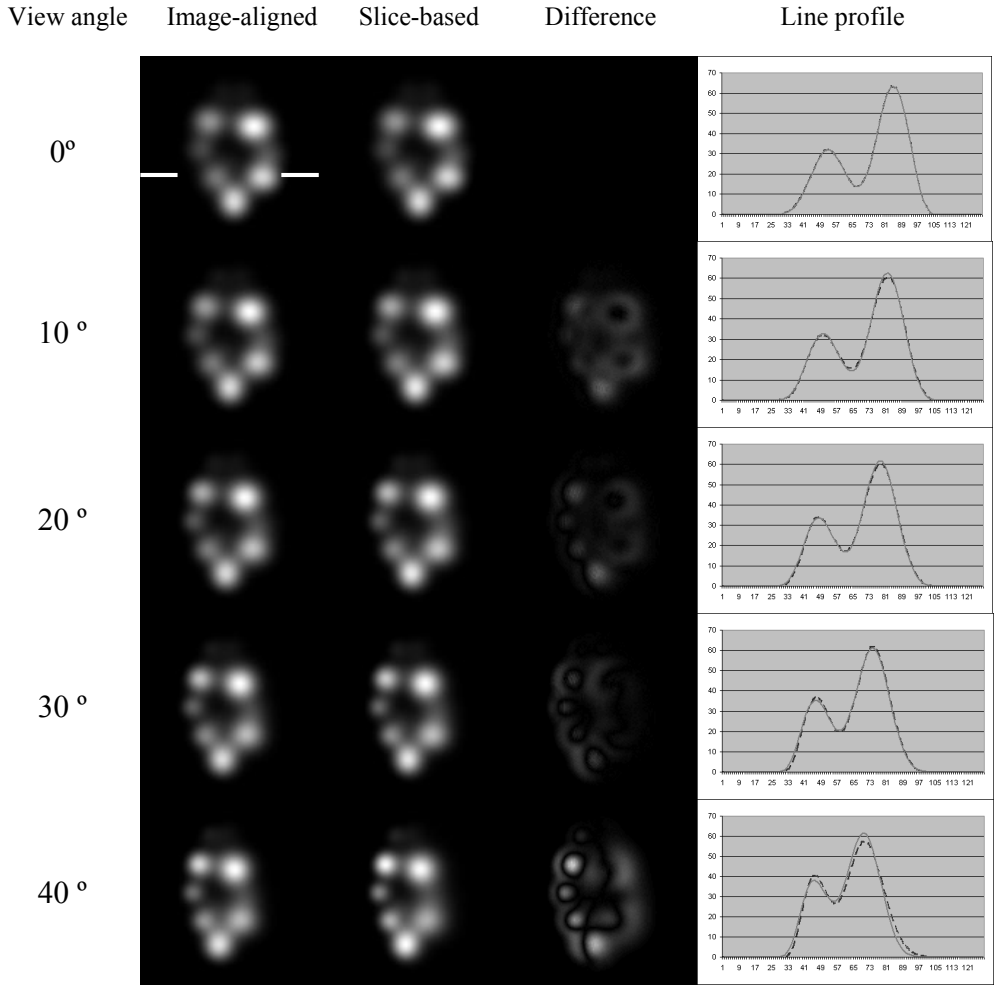


Figure 2.9: Projection evaluation of detector and axis-aligned scattering models. Profiles: detector-aligned (solid), axis aligned (dashed) projections

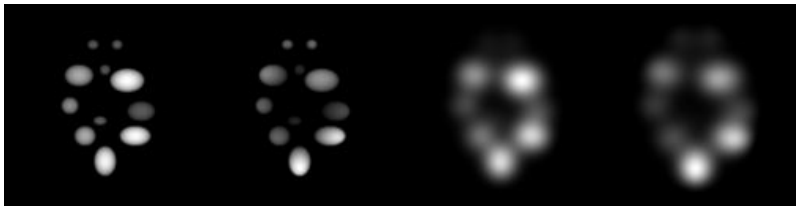


Figure 2.10: Projections obtained with different effects. **E**: emission; **A**: attenuation; **S**: scattering.

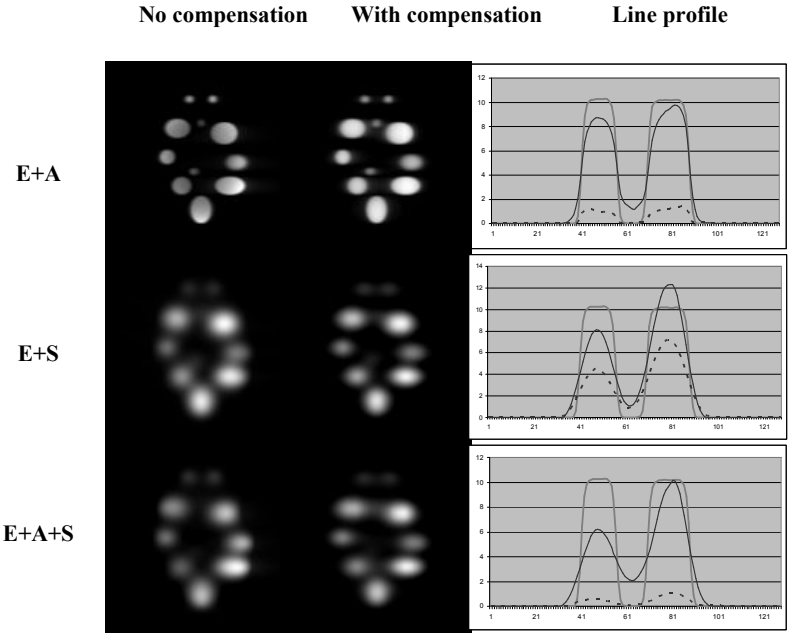


Figure 2.11: Reconstructions results obtained with different effects. **E**: emission; **A**: attenuation; **S**: scattering. Solid grey line: original phantom profile; Dotted line: when the effect has not been modeled; Solid black line: when the effect has been modeled.

2.5 Signal Analysis and Evaluation

2.5.1 Popular Interpolation and Integration Methods

Background

A plethora of methods for projection and backprojection methods has become available for CT through the past three decades. One way to distinguish these is in the manner they relate the data to be generated (the reconstruction) to the data provided (the scanner data). With 3D reconstruction in mind, we shall refer to the former as the volume, composed of voxels, and the latter as (projection) image, composed of pixels. While analytical CT algorithms, such the approximation by Feldkamp (FDK) [26] mainly involve a backprojection, iterative algorithms, such as SART [2] and EM [82], involve both projections and backprojections. In most cases, backprojection is simply the inverse of a projection, unless unmatched projector pairs are used [103]. The target of a projection is a (projection) image P, while the target of a backprojection is the volume V. Iterative algorithms work by projecting V to create an image estimate P which is then compared with the acquired image P and a correction image $P'' = s(P-P)$ is backprojected – s being a scale factor – which brings V closer to an accurate estimate. The projection operator is written as: $P = WV$ and the backprojection operator is written as $V = W^{-1}P$. Here, P is a N^2-long vector of pixels, V is a N^3-long vector of voxels, and W is a $N^2 \times N^3$ weight matrix. In the general case, in particular for 3D cone-beam reconstruction, the size of W prohibits its storage, and thus its elements have to be computed on the fly. In fact, W is a sparse matrix, since generally a pixel in P traces out a ray line or a ray beam, which only cover a subset of connected voxels (see Figure 2.13).

The remaining issue is how the voxels that fall within the influence of this beam or line contribute to the pixel emitting it. This is where the various methods differ, and they do so in three, mostly orthogonal ways:

- Either they assume that the voxels are solid blocks or that the voxels are infinitesimal thin spikes (or sample points). In fact, this just leads to different kernels (a box for the former).
- Either they trace rays emerging from the pixels, or a few subpixels

within a pixel, or they trace beams, usually bounded by the pixel (detector) boundaries.

- Either they trace the rays (or beams) across the volume from the pixels or they project the voxels onto the image plane.

Finally, they either perform a piecewise constant, piecewise linear, or continuous integration. We should note that not all of the combinations have actually been proposed and implemented, and we shall restrict our discussion to only those that have been.

We have studied these methods in a common framework designed to perform these tests in a strict signal processing context. This is justified by the fact that interpolation, and to a somewhat lesser extent, integration, are operations that seek to estimate a continuous function from a set of discrete sample points. While during data acquisition (the X-ray projection in the scanner) the rays have traversed an object that was naturally defined everywhere, this is not the case in the simulation of this process, where the object under reconstruction is only defined at discrete data points. The circumstance that a collection of rays that is averaged (before the log-operation) within a certain detector bin, poses another challenge for this simulation during reconstruction. A faithful simulation of the projection process, however, is key to a faithful reconstruction, and we shall compare the different methods with regards to this viewpoint (neglecting any other adverse effects, such as scattering, beam hardening, and polychromaiticity). But ultimately, we are interested in a faithful reconstruction, and this motivates an extension of this comparative study to a reconstruction task. In order to avoid possibly confounding perspective effects implied by fan or cone-beam, we perform our study in a parallel-beam reconstruction scenario, using our strict signal processing-motivated test function.

A wealth of publications exist that discuss and compare interpolation filters, and the limited space only permits to mention a few of these here. Thévenaz et al. [92] provide a comprehensive study on interpolation filters using frequency domain arguments, while Möller et al. [66] view this task from a numerical standpoint, via a Taylor series expansion. Siddon [83], Joseph [42], Herman [39], and Lewitt [57] have described interpolation and integration mechanisms that are frequently used in CT today. Other, more recent papers, have enhanced and augmented these basic approaches, and the reader is referred to [20] for a more complete list of references. CT reconstruction is in some ways similar to volume rendering, where the goal

is to project a volume dataset for visualization purposes. A seminal paper with respect to the study and comparison of interpolation filters in the context of volume rendering was presented by Marschner and Lobb [63]. To conduct their comparison they designed a rigorous test function, now known as the Marschner-Lobb (ML) function, which has a near-uniform frequency content that extends very close to the Nyquist rate and is contained within that interval to 99.8%. Its equation is given here:

$$\rho(x,y,z) = \frac{(1-\sin(\pi z/2)+\alpha(1+\rho_r(\sqrt{x^2+y^2})))}{2(1+\alpha)} \quad (2.23)$$

$$\rho_r(r) = cos(2\pi \cdot f_M \cdot \cos(\frac{\pi r}{2})) \quad (2.24)$$

Here, $\alpha = 0.25$ and f_M controls the frequency bandwidth for a given volume size. We set it $f_M = 18$, which provides the desired full frequency range for a volume size of 128^3 when $-1 < x, y, z < 1$.

A volume iso-surface rendering of the function, sampled into a volume of 40^3 with $f_M = 6$ and rendered at an iso-value of 0.5 is shown in Figure 2.12. The interpolation filter used in the rendering (using raycasting) was of good quality and only caused little aliasing, as is evident from the modest deformations at the sinusoid rims. Since in CT we are reconstructing an *estimate* of the volume dataset from a set of near-analytical projections, we find it more useful to compare the reconstructed dataset with the true function in a numerical sense, via its RMS, and not via a visualization, which was done in the original ML work.

Methods

Figure 2.13 illustrates various interpolation and integration strategies. Although we only show the interpolation for the 2D case, for 3D rendering, the drawings would extend into 3D (which turns every linear interpolation into a bilinear interpolation).

- The *slice-interpolated* method uses bilinear interpolation within each slice and integrates the results in a trapezoidal fashion. This is a 3D extension to Joseph's method [42] and is also the method our GPU projective-texture renderer employs [98]. Depending on the viewing

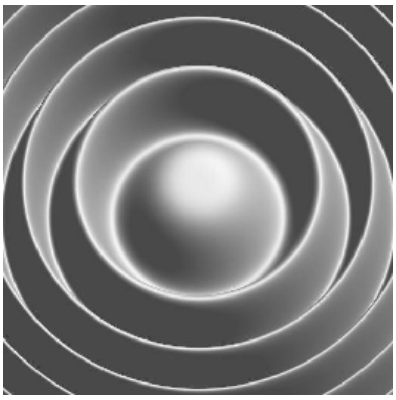


Figure 2.12: The Marschner-Lobb function.

angle, the sampling rate along the ray varies $(1...\sqrt{2})$, and so does the integration interval. One way to increase the integration quality is to use a more sophisticated quadrature method, such as Simpson's method along with an intermediate sample.

- The *grid-interpolated* scheme can space the samples at the same distance, independent of viewing angle, usually at 1.0. Interpolation is commonly performed with a trilinear filter, and integration is again according to the trapezoidal rule.

- The *box-line-integrated* scheme was proposed by Siddon [83]. This corresponds to a box interpolation filter (kernel) and a continuous integration (assuming a piecewise constant signal).

- The *RBF-line-integrated* method (RBF = Radial Basis Function) was proposed by Lewitt [57] and is used in conjunction with a forward projection (splatting) of a pre-integrated kernel function. The integration is continuous, similar Siddon's method, since the kernel (usually a Gaussian or a Bessel function) is superior to a box, but also much wider, taking longer to project.

- The *box-beam-integrated* method is an extension to Siddon's method, where now the entire box volume that falls inside the extended pixel

boundaries is added to the integral. It still assumes an inferior box kernel, but it captures the width-integrating nature of the X-ray beam better than a line. The distance-driven method of [20] works along these lines.

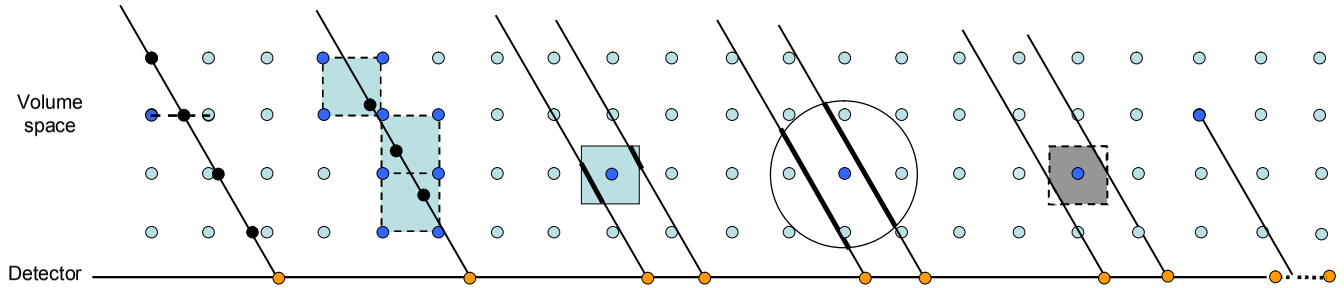
- In the *pixel-driven* method (we call it *voxel-driven*) [39], each voxel is projected to the screen and its contribution written to the nearest pixel or distributed with bilinear weighting among the four pixels in the screen square. The latter is similar to the splatting of a bilinear kernel.

Since the acquired data are proportional to the average to the beam of rays that end in a given detector bin, a beam-integrated method is at least potentially a better choice. We have just discussed the box-beam-integrated (distance-driven) method, which uses an underlying box filter. The task of computing the solid encapsulated within a beam is feasible to compute. It is more difficult to do so for higher-order functions, although it can be done with the radially-symmetric functions, such as the RBF method, using a Summed-Area-Table [18]. An approximation of the beam-tracing that can use better interpolation kernels is to trace extra rays and then downsample the result, using some sort of lowpass filter tuned to the sampling rate of the output grid. The most obvious way is the *cartesian-2×-oversampled* scheme shown on Figure 2.15. However, this requires the tracing of 4-times the number of original rays, which can be slow. A better solution in this regard is what we call the *hexagonal-2×-oversampled* scheme, which will achieve comparable results with much less samples. We will describe this in details in the following section.

2.5.2 Optimal Sampling Grid

Background

Regular lattices typically are Cartesian lattices with grid samples distributed on a separable, orthogonal raster of most often equal grid spacing in all dimensions. This type of lattice is very convenient for representation, indexing, and interpolation, and it is also easy to conceptualize. It is mostly for these reasons that the regular Cartesian lattice has become the most dominantly used regular raster structure today. But recent years have brought



Method	Slice-interpolated	Grid-interpolated	Box-line-integrated	RBF-line-integrated	Box-beam-integrated	Pixel-driven
Interpolation	Bilinear (non-uniform)	Trilinear (uniform)	Box	Radial Basis Function	Box	Bilinear or nearest
Integration	Trapezoidal (non-uniform)	Trapezoidal (uniform)	Continuous (line)	Continuous (line)	Continuous (box)	
Notes	Embedded in GPU renderers	Uniform sampling rate	Siddon's method	Gaussian or Bessel kernel	Extended Siddon's method	

Figure 2.13: Various interpolation and integration methods. The lines are rays traversing the volume grid, with the empty dots indicating the voxels.

an increased awareness with respect to the sub-optimality of this grid topology. Early efforts in this direction have mostly concentrated on efficiency, and the key observation motivating this is made in the frequency domain. When assuming a radially symmetric (spherical) frequency spectrum of the rasterized signal, then the optimal packing of the alias spectra is not a Cartesian lattice but a hexagonal one, since such a lattice packs the frequency spectrum spheres closest together (which directly follows from optimal sphere packing theory [17]). This, in turn, using the Fourier scaling theorem, stretches the samples in the co-domain, here the spatial domain, furthest apart, leading to the coarsest possible sampling pattern without risking (pre-) aliasing. This makes possible a reduction of grid points to 87% in 2D, 71% in 3D, and 50% in 4D, with a direct consequence being a reduction in storage by these amounts, which can affect cache behavior as well. But possibly more importantly, grid processing costs are reduced by these amounts as well, if these costs are strongly related to the number of grid points. This is the case whenever a point-based projection approach is used. We call such a scheme voxel-driven, in contrast to pixel-driven schemes in which rays originating from projection pixels traverse and interpolate the lattice.

It turns out that cost savings are only one aspect of optimal lattices. They also provide a more uniform and isotropic sampling of the space [86], under the condition of a tighter space sampling than that implied by the sphere-packing results. Our research shows the important implications this has for CT, and in fact, these aspects come into play in CT on two occasions: (1) the initial data acquisition on the detector lattice, capturing the object-attenuated transmission X-ray radiation, and (2) the object formation on the reconstruction lattice accumulating the back-projected contributions. In the subsequent sections, we will give theoretical arguments as well as show practical examples for both.

Prior Work

The optimal lattice in 2D is the hexagonal lattice, and in 3D it is the Body Centered Cartesian (BCC) lattice. These are illustrated in Figure 2.14, with the grid distances expressed in terms of its frequency bandwidth-equivalent Cartesian (CC) lattice. Note that this assumes frequency spectra that fit into a radially symmetric hull (circle, sphere) with the radius

given by the frequency bandwidth.

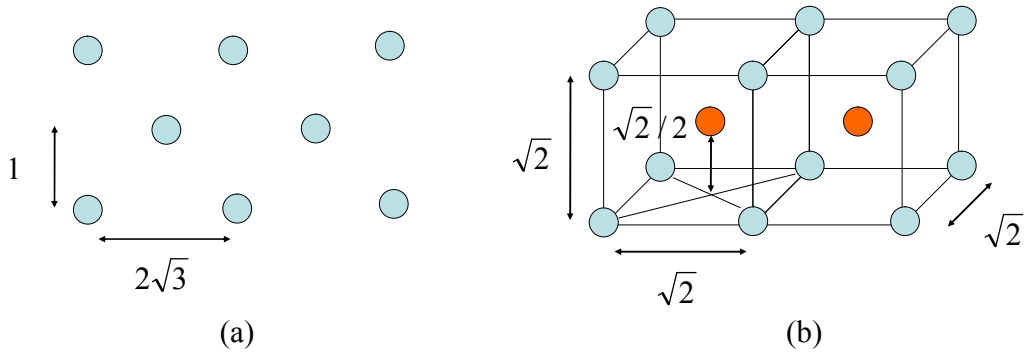


Figure 2.14: (a) 2D hexagonal lattice (b) 3D BCC lattice

As mentioned above, optimal sampling lattices have been used for quite some time for discrete object representation, both for 3D [91] and 4D volume datasets [72]. These works were mainly motivated by the reduction in the number of required lattice samples. Since the generation of projective, volume-rendered images requires the interpolation of lattice samples, now in this optimal lattice space, appropriate filters for this operation have been proposed, such as box-splines [24], hex-splines [96], and a pre-filtering scheme that operates in conjunction with a Gaussian filter [19]. Finally, the theoretical finding of space-optimality without quality loss was confirmed in a user study which compared images rendered from a Cartesian lattice and images rendered from the same object rendered from an optimal lattice [65].

There were also researchers in the CT reconstruction arena who have used optimal lattices [64] [68], and these works were all within iterative reconstruction frameworks. Just as in volume rendering, these efforts were mainly driven by the reduction of grid samples and the subsequent speedup in terms of reconstruction time. Using optimal lattices brought significant savings here since iterative algorithms typically project and back-project the evolving reconstruction many times, and thus a reduction in grid complexity can make a considerable difference in running time.

We will show that due to these intrinsic properties, optimal lattices can reconstruct and acquire fine detail significantly better than the standard Cartesian lattices, without a loss of performance. This is important, for

example, when it comes to the detection of small lesions in CT. At the same time, optimal lattices can also provide a higher-fidelity sampling of the incoming X-ray signal on the detector plane.

Theory of Lattice Uniformity

In a uniform lattice the maximum distance of an arbitrary sample in space to a lattice point is independent of direction. Thus, the Voronoi cell of such a uniform lattice must be a sphere. A collection of spheres, however, cannot be space-filling and thus optimal uniformity cannot be achieved in practice. We therefore seek a lattice with a Voronoi cell that is closest to a sphere. The sphere has the smallest surface area enclosing a given volume. Setting the volume of a sphere to unity, the surface area S_s is:

$$S_s = 4\pi(\frac{1}{\sqrt[3]{4/3 \cdot \pi}})^2 = 4.83 \tag{2.25}$$

The surface area S_{cc} of a unit cube is an obvious $S_{cc} = 6$. The Voronoi cell of the BCC lattice is the truncated octrahedron, and the surface area S_{bcc} of its unit cell is:

$$S_{bcc} = (6 + 12\sqrt{3}) \cdot (\frac{1}{\sqrt[3]{8\sqrt{2}}})^2 = 5.31 \tag{2.26}$$

Here the second term is the lattice parameter value setting for a unit cell. We see that the BCC lattice is about 10% worse than the sphere, but 12% better than the CC lattice. Finally, let us have a look at the Face-Centered Cartesian (FCC) lattice, which is the dual of the BCC lattice. Its Voronoi cell is the rhombic dodecahedron, and the surface area Sfcc of its unit cell is:

$$S_{fcc} = (8\sqrt{2}) \cdot (\frac{1}{\sqrt[3]{(16/9)\sqrt{3}}})^2 = 5.34 \tag{2.27}$$

Again, the second term is the lattice parameter value setting for a unit volume cell. Thus, the BCC lattice is slightly more isotropic than the FCC lattice, under this metric. We therefore choose the BCC lattice.

Implementation

Data Acquisition Detectors are typically composed of an array of square pixels. There has been a recent trend to CMOS flat-panel detectors, and parallel to this, optical video cameras have also been introduced that adopted hexagonal lattices for more isotropic sampling. This has led us to explore these types of lattices also for X-ray detection. However, as we have seen in the previous section, GPU-accelerated reconstruction requires a standard Cartesian grid for the mapping of the voxels onto the projections. Else, the added overhead incurred by application of fragment shader-bound interpolation filters would cause a significant loss in performance, as this would occur at a complexity of $O(N^4)$, assuming we have $O(N)$ projections to reconstruct N^3 voxels. We therefore resample the acquired projections from the hexagonal grid onto a double-resolution Cartesian grid before reconstruction begins, using a high-quality filter. This enables fast hardware accelerated bilinear interpolation, and it also lowers the complexity of the hex-grid interpolation by an order of magnitude (now it is a pre-processing step). In fact, we couple this operation with a projection of the raw projection images into a standard axis-parallel configuration which allows for a faster voxel mapping in the back-projection stage [43]. The result are projection images of resolution $2N^2$, which can even be back-projected using with nearest neighbor interpolation (but we use bilinear since there is almost no performance penalty on GPUs).

In practice, assuming a unit grid spacing for the *direct-sampled* case, the hexagonal grid would sample at $1/\sqrt{3}$ along x and at 0.5 along y (see Figure 2.15). This would require 13.4% less samples, that is, for a 100^2 grid, the cartesian scheme would require $40,000$ rays, compared to $34,640$ rays for the hexagonal scheme.

Volume Reconstruction Essential for the work described here is the fact that the back-projection operations are independent of the underlying lattice. Lattice points (voxels) (i) are backprojected (mapped to a projection image), (ii) then interpolate their updates from this projection image, and (iii) finally receive their depth-weighting, all according to their individual coordinates. It is the lattice's "responsibility" that it can represent, in terms of aliasing, the signal that is being compiled that way.

Our GPU-accelerated reconstruction algorithm only requires a slight

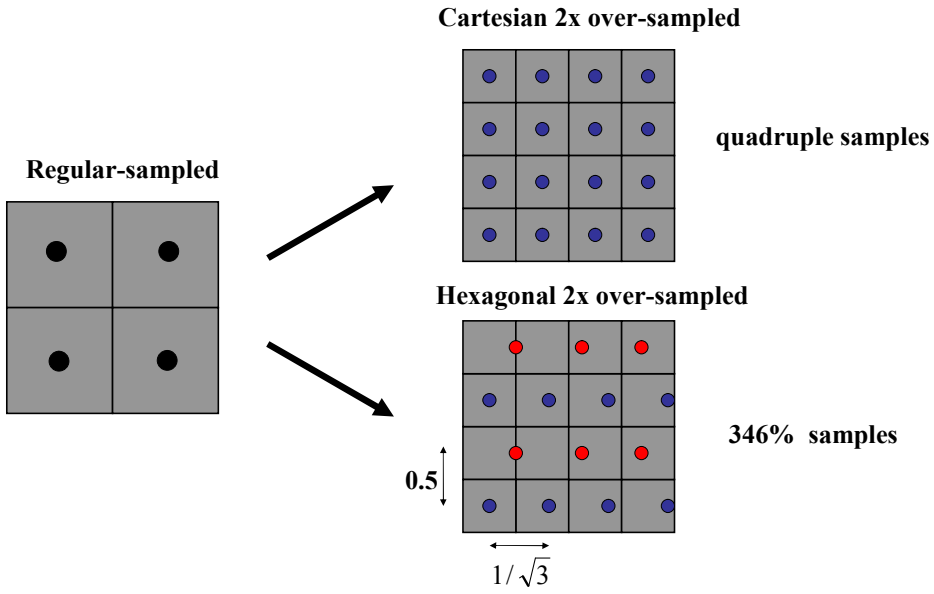


Figure 2.15: The empty dots are the pixels (from which the rays emerge), and the full dots are the final pixels stored – the oversampled methods downsample the images obtained with the traced rays.

change for handing the BCC lattice. Since for BCC each slice is a standard Cartesian lattice, we can employ our interpolation fragment program unchanged. The only item requiring change is the vertex program that needs to shift the slice polygon for each odd slice index.

We have chosen to keep the total number of voxels the same. This reduces the in-slice lattice spacing from the space-optimal $\sqrt{2}$ to $\sqrt{3} = 1.26$ (and an inter-slice distance of $\sqrt[3]{2}/2$), yielding a system where the reconstruction performance does not suffer at all. However, as we see next, this arrangement is able to resolve small feature better than the standard lattice with both in-slice and inter-slice spacing of 1.

2.5.3 Results

Evaluation of Interpolation and Integration Methods

We conducted both a projection and a reconstruction study to evaluate the performance of various interpolation and integration methods. All results are produced on a 2.2GHz dual core AMD Athlon PC with 1GB RAM.

Projection Study We generated both line-integrated and beam-integrated reference images from the ML dataset. The latter are produced by calculating 16 sub-rays for each detector element and box-filtering the values. Both types of reference images are computed by sampling the object function with a dense step size of 0.2 unit distances. A discretized 3D floating point dataset (128^3 resolution) sampled from the object function is used to generate all other projections. We measure the RMS error from different viewing angles, particularly focusing into oblique views in the range of 40° to 45° where slice-integrated schemes are prone to have artifacts due to their insufficient sampling rate along the ray direction from those views. Note that we compare all strategies in the beam-integrated study, but only line-integrated schemes in the other.

Since the detector elements in the optimal lattices are regular hexagons, let us first review some (regular) hexagon geometry metrics. The area of a hexagon $A_{hex} = 3\sqrt{3}/2 \cdot a^2$, where a is the side length of the hexagon, and the maximal diameter is $2a$ and the minimal diameter is $\sqrt{3}a$. The minimal diameter is the horizontal in-slice lattice spacing, and the vertical spacing is $\sqrt{3}/2$ of this. With this in mind, we explored the following configurations:

1. The standard Cartesian case, assuming a lattice spacing of 1, and a (square) detector element area of 1 (*cartesian*).

2. A standard Cartesian lattice with twice the resolution of #1, which gives a detector element area of 0.25 (*cartesian -2×-oversampled*).

3. The space-optimal (hexagonal) lattice version of #2, which has 86% ($\sqrt{3}/2$) of the elements of #2. Here, the detector element area is increased by 23% to 0.28 (*hexagonal-2×-oversampled*).

Figure 2.16 and 2.17 plot the evaluation results from each strategy against line-integrated and beam-integrated reference images, respectively. We observed that in the line-integrated scanner projections study, Siddon's method (box-line-integrated scheme) produces overall better projections than all other methods, which we believe is partly due to its more accurate continuous integration path. Slice-interpolated approaches, while having less sampling density along the ray direction compared to grid-interpolated methods, have nevertheless similar image quality. The voxel-driven scheme with bilinear kernel has similar performance but suffers a severe deterioration in image quality at angles close to 45°. The blurry effect from the RBF-line-integrated method leads to a low score when compared to line-integrated scanner projections. Finally, we also observe that Simpson's integration does not help improve image quality for slice-integrated schemes.

While in the beam-integrated study, the RBF-line-integrated method using the Bessel kernel achieves the best image quality, slice-integrated methods and grid-integrated approach perform as just as good as Siddon's method. Also it helps to generate more accurate projections by computing extra rays for the ray-driven methods, where hexagonal-2×-oversampled and trilinear-2×-oversampled schemes surpass the cartesian-2×-oversampled scheme. If the trapezoidal integration is replaced by Simpson's rule in slice-integrated and hexagonal-2×-oversampled schemes, further improvement on image quality can be obtained. However, oversampling in the object space via computing intermediate slices has little impact for slice-interpolated approaches. Similarly, the voxel-driven scheme with the bilinear kernel yields results close in quality to Siddon's method, except at views around 45°.

Reconstruction Study We employed the Simultaneous Algebraic Reconstruction Technique (SART) as our test algorithm to perform this study. In addition to the ML dataset, a human CT skull volume is also used for evaluation. Both datasets are discretized into a grid of 128^3, while 80 views of reference images of detector elements are acquired uniformly within 360 degrees. A relaxation parameter of 0.1 and 20 iterations are applied throughout the experiment. Both line-integrated and beam-integrated reference images are generated for the ML study, while only line-integrated references are derived for the human CT skull study. Fig 2.18 and 2.19

demonstrate the performance of all methods for the ML dataset in the line- and beam-integrated settings, respectively.

We notice that the slice-integrated schemes have at least comparable performance with Siddon's method and the grid-integrated method, in terms of both reconstruction quality and convergence speed. Similar to the projection study, given beam-integrated scanner projections, over-sampling on the detector plane helps to reduce the error, and here the hexagonal-2$\times$-oversampled scheme outperforms the cartesian-2$\times$-oversampled scheme, and even the RBF-line-integrated method. In contrast, there does not appear to be an advantage in applying Simpson's rule over the trapezoidal integration.

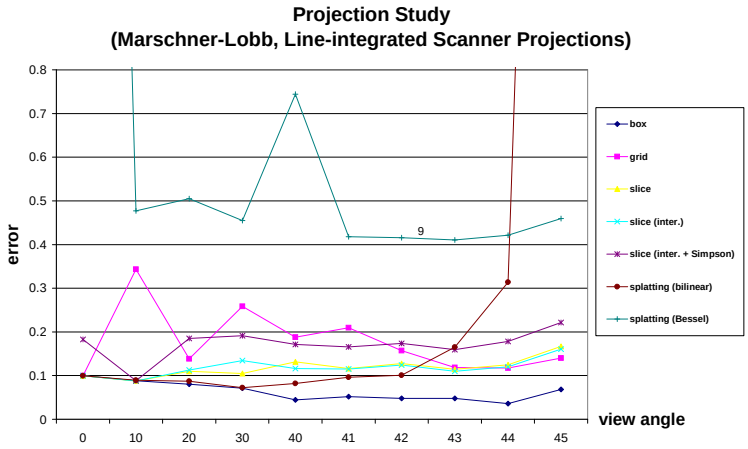


Figure 2.16: A projection study with line-integrated scanner projections.

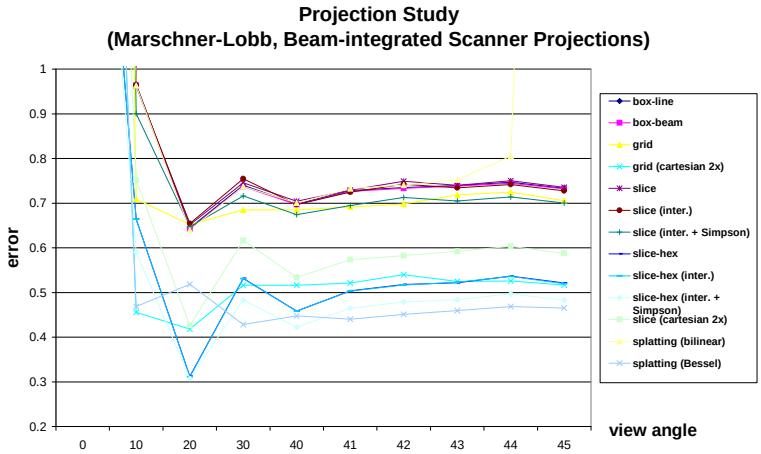


Figure 2.17: A projection study with beam-integrated scanner projections.

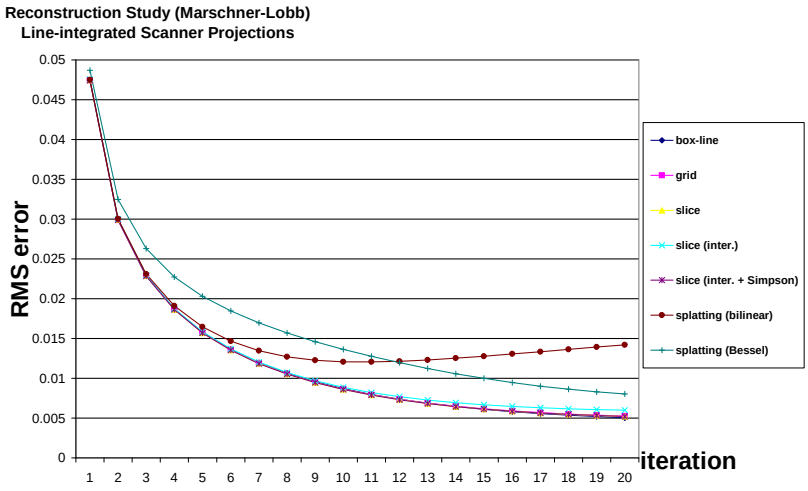


Figure 2.18: A reconstruction study with line-integrated scanner projections. RMS errors after 10 iterations are plotted in the inset

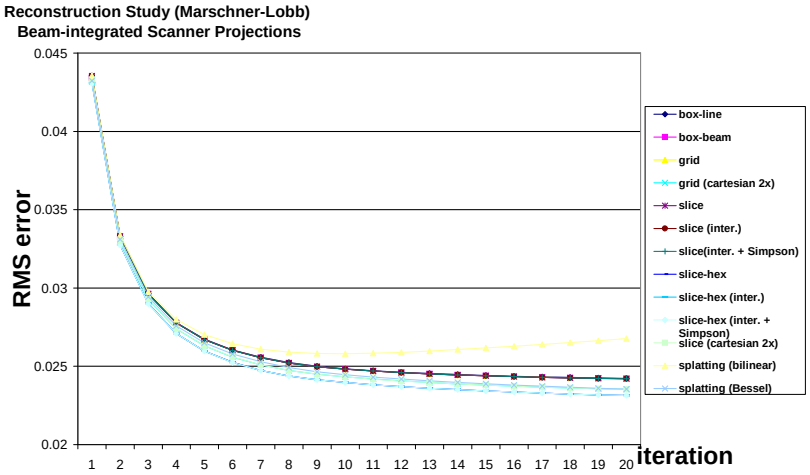


Figure 2.19: A reconstruction study with beam-integrated scanner projections. RMS errors after 10 iterations are plotted in the inset

Optimal Lattice for Volume Reconstruction

We then used the high-resolution detector images for 3D reconstruction. We first reconstructed the toes dataset, both onto a standard Cartesian lattice and then on the BCC lattice with the same number of elements, using our streaming CT reconstruction application (Chapter 3). Fig 2.20 shows visualizations from the same viewpoint and at the same iso-surface setting. We notice a slightly better recovery of the features and an overall smoother surface quality. But the differences are not overly dramatic.

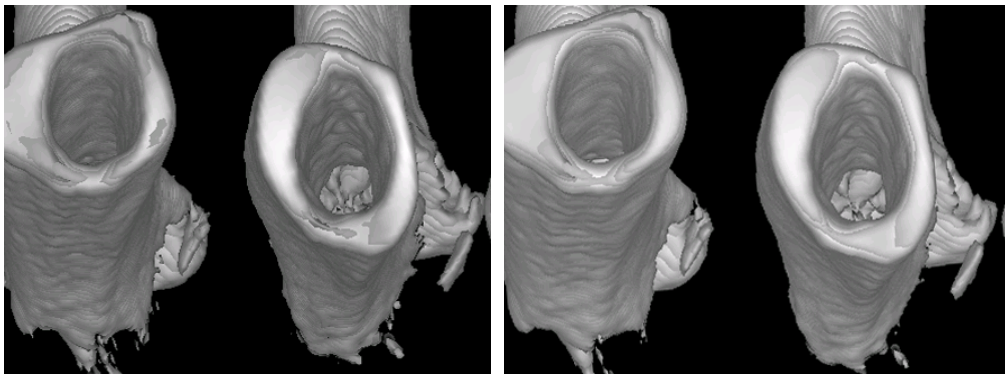


Figure 2.20: Iso-surface rendering of the Toes dataset: (left) CC-lattice reconstruction, (right) BCC-lattice reconstruction

The true advantages of optimal lattices lie in their ability to recover small features better, and with less sensitivity to orientation (due to its more isotropic sampling). A common task in medical imaging is the detection of small lesions and tumors. To explore the performance of the two candidate lattices in this context, we generated a 3D phantom dataset consisting of blobs of various sizes and generated (standard Cartesian) projections of these. The projections had a resolution such that the smallest blobs would still be detectable to at least 1-2 pixels. We then reconstructed these blobs on both a CC and the equivalent BCC lattice at a resolution matching that of the projections. At this lattice resolution the size of the smallest blobs amounted to about 1.5 voxels. The reconstruction results are shown in Figure 2.21. We see that only the BCC lattice is able to recover the smaller blobs, and overall the blobs appear better refined.

Next, we generated a tumor brain phantom by embedding a selection of randomly distributed small tumors into an existing brain volume dataset. We generated projections and reconstructed them as well, using the same protocol than for the blob phantom. Figure 2.22 shows a slice of this reconstruction. Again, we observe that the BCC lattice is able to recover almost all of the tumors, while the CC lattice fails in many cases.

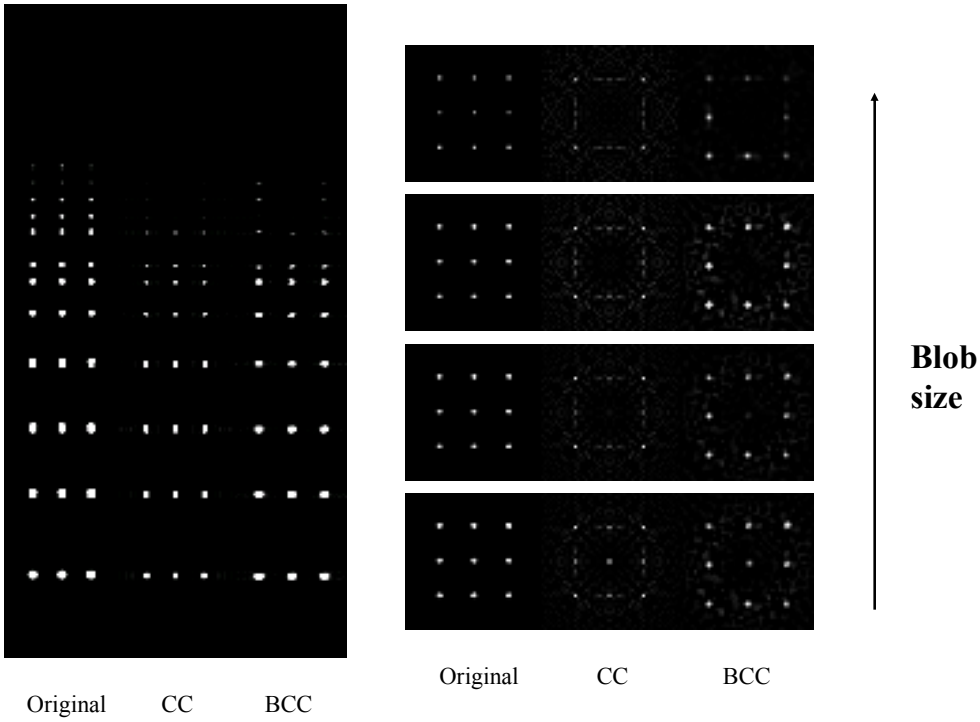
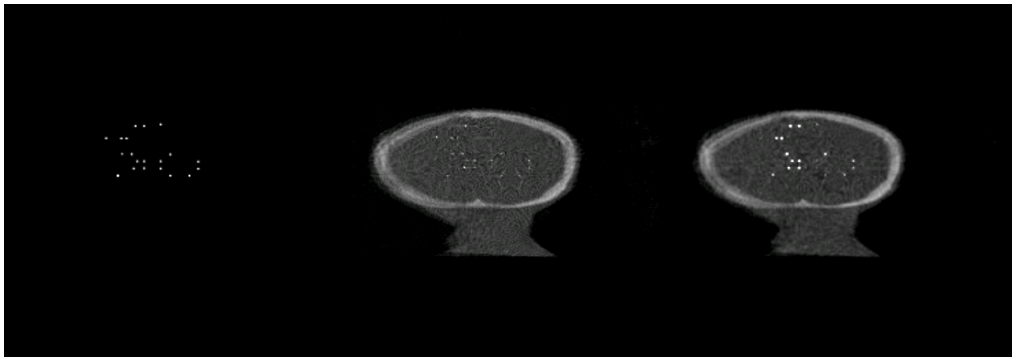


Figure 2.21: Blob phantom study. (Left): a row of blobs of decreasing size extends diagonally across space. The smallest blobs have the equivalent size of 1.5 voxels. The CC and BCC lattice reconstructions of these are shown to the right. (Right): Cross-sections of the blob phantom and its reconstructions. We observe that the BCC lattice recovers the small blobs significantly better, and in some cases is the only lattice to recover them.



Original CC BCC

Figure 2.22: Tumor phantom study, which is a more realistic experiment than the blob study. A set of small tumors (a slice of these is shown on the left) was projected into existing brain projections images. The corresponding slice of the reconstruction results using the CC and the BCC lattices are shown to the right, respectively. We see that the BCC lattice cannot recover all them either (due to their small size), but it recovers significantly more of them and at higher intensity.

Chapter 3

Streaming-CT

3.1 Introduction

Spiral (helical) CT has become the de-facto equipment for volumetric diagnostic CT, and scanners of this nature can be found in almost abundant quantities both in radiology departments of hospitals as well as at commercial imaging services. In recent years, multi-row detectors have given rise to a spiraling cone-beam data acquisition, shortening acquisition time to single breath-hold imaging which is useful for pulmonary and cardiac CT (see [10] for a comprehensive overview on the theoretical aspects of dynamic CT). Scanners with 64 such rows are currently available, but scanners with 256 detector rows have already been prototyped in research labs. Nevertheless, despite their obvious advantages, spiral CT scanners can be large and costly, and at the same time less suited for interactive applications, such as planning or monitoring applications occurring in radiotherapy, image-guided surgery, patient positioning, instrument navigation tasks, trauma units with mobile scanners, and others. In that context, the emergence of technologically advanced flat-panel displays, manufactured from either amorphous silicon or CMOS, have created alternative image acquisition platforms seeking to fill these application areas in which spiral CT equipment is less suitable. The traditional platform for cone-beam CT is the C-arm gantry, which typically provides a spin-range

of 270° at 30-50° /s. C-arm gantries often suffer from instability problems, which need to be quantified and corrected, and novel closed-gantry scanners have recently become available to eliminate these problems. On the other hand, image acquisition speed and resolution are both dictated by the flat-panel detector in use. Here, 30 frames/s are typically acquired, at a matrix size of up to 1024^2. C-arm scanners have been the platform of initial research in this direction, leading to the development of computed rotational angiography where images acquired with standard rotational angiography were used for volumetric reconstruction (see earlier research by Fahrig et al. [25], which still used detectors based on X-ray image intensifier (XRII) technology). Micro-tomography for small animal imaging has also traditionally used this type of platform (see, e.g., Lee et al. [56]). Finally, more recent applications include CT-guided radiation therapy [41], trauma imaging [95], 4D imaging for cardiac CT and others [51] [89] [67], and the integration of CT with interventional procedures [61]. Further applications are in dentistry, but also in non-medical scenarios in industry and security. The majority of these approaches use the cone-beam reconstruction algorithm devised by Feldkamp, Davis and Kress [26], which is a special case of the exact cone-beam reconstruction algorithm pioneered by Grangeat [36].

For any of these applications, all one requires is an X-ray source, a (2D) flat-panel detector, a gantry with rotation capabilities (C-arm or closed), a computer programmed to perform both CT reconstruction and subsequent visualization of the results, and a high-speed interconnect to transmit the detector data to the computer. X-ray sources, detectors, and gantries are now readily available from various companies (small and large), either in form of components or already assembled into turn-key systems. Common to many of the aforementioned cone-beam CT applications is the expectation to achieve reconstructions in an expedient manner. When used in an interventional application, a near-interactive reconstruction is desirable. The overall goal motivating CT in all of these settings is to obtain a visualization of the acquired data in a form different from what has been originally acquired by the detector. Rather than viewing the scanned X-ray projection images directly, as in standard fluoroscopic angiography, clinicians seek to create novel views and insight into the scanned subject. Once the region of interest has been 3D reconstructed, these novel views can be obtained via 3D slicing, goal-directed segmentation, and shaded volume rendering. Ideally, clinicians also would like to obtain these novel views

quickly, as if the scanner had produced this imagery directly. This is obviously important in interactive applications, such as image-guided surgery, radiation therapy, and other tasks in which the imaged object, here the tissue or bones, is manipulated or modified in real time. Being able to get a quick volumetric snapshot of the current state of the object represents a significant advancement over methods in which the object was imaged before in a pre-operative scan, and in which only the tracked instruments move, in this static volume. For example, Bonnet et al. [10] have described the *Sliding Window Principle* for dynamic (4D) CT, in which the reconstruction is continuously updated with newly acquired projections while the influence of older projections is phased out at the same rate.

To enable this type of operation for our commodity CT scenario, the speed of the data processing must match that of the data acquisition. More concretely, assuming an image acquisition rate of 30 2D projections/s, the reconstruction must proceed at the same bandwidth, that is, at a rate of 30 projection/s. This then will result in a pipeline in which projection data are produced by the acquisition process (the detector) and are immediately consumed by the reconstruction process (the computer), without any growing intermediate buffers. In that case the time delay incurred for 3D viewing is only constrained by the speed of the gantry. This is true in most types of scanning protocols, such as full-scan, short-scan, sliding window, and others [89] [10]. If the reconstruction speed matches the acquisition speed (assuming properly dimensioned network interconnects), then the reconstruction is obtained and can be viewed immediately after the protocol allows for it, plus the time a projection takes to flow (or stream) across the computational pipeline. Therefore, in order to reflect this type of dataflow architecture, we refer to this reconstruction paradigm as streaming CT.

Unfortunately, in today's practice even highly optimized CPU-based reconstruction engines cannot achieve this degree of computational bandwidth. Typically, reconstructing a 512^3 volume from 360 512^2 projections still consumes 60-100 seconds on a dual Pentium PC, while streaming CT would require reconstruction speeds an order of magnitude higher. The usual resort to bridge this performance gap is to employ either high performance, specialized proprietary hardware (ASIC or FPGA) or a multi-node cluster, as for example the CPU/FPGA inline reconstruction architecture by Brasse et al. [11]. However, all of these implementations are expensive to develop and can be difficult to modify.

With the help of modern programmable computer graphics boards (GPUs) we are able to fulfill the requirement of streaming CT at a very low cost. Although GPUs were mainly (and continue to be) designed to produce dazzling visual and physical effects for computer games and entertainment, they have also given rise to the strong trend of GPGPU (General Purpose Computing on GPUs). The advantage of GPUs (over proprietary platforms) is that they are widely available at a price of less than $400, their programs are easily modified and updated, their programming model is well understood and supported by a large user base, and their existence is immensely boosted by the commercial power of interactive entertainment. In fact, due to both the market pressure and their favorable general architecture, the performance of GPUs has consistently doubled every six months, which is triple of Moore's law governing the growth of the performance of CPUs. Therefore, in terms of the streaming CT paradigm, a GPU-based platform is much better prepared to scale with growing gantry speeds, and projection and volume dimensions. At the same time, GPU on-board memory has also substantially increased (to 1GB and more), enabling reconstructions of realistically-sized volume datasets (512^3 and more) at floating point precision without incurring any time-consuming data communication to main memory. Finally, since GPUs are already frequently used for the visualization of the final reconstruction results, they naturally lend themselves also for the visual monitoring of the ongoing reconstruction.

3.2 Mapping Feldkamp's Algorithm

The three equations in Section 1.1.1 constitute a natural decomposition of the FDK algorithm, forming a pipeline consisting of filtering, backprojection, and weighting. Depending on how the stream is defined, we have two modules that have different processing sequences with respect to both projections and the volume being reconstructed. The first module, a rigorous *Streaming-CT*, processes the incoming individual acquisition completely before going to the next. This essentially eliminates the necessity of keeping projections in memory but the reconstructed volume has to remain in memory as a whole. There is no space concern for large projections, since projections are deleted from GPU memory immediately after

they have been applied (consumed) for reconstruction. The other module, which we call *Slice-CT* takes priority in processing the volume slices. Here all projections should be ready before any reconstruction of slices of the volume starts. The advantage is that reconstructed volume slices can be swapped out of the pipeline and examined immediately. Despite the order of processing, the underlying reconstruction operations with respect to each voxel are the same, which constitute of the three steps mentioned above.

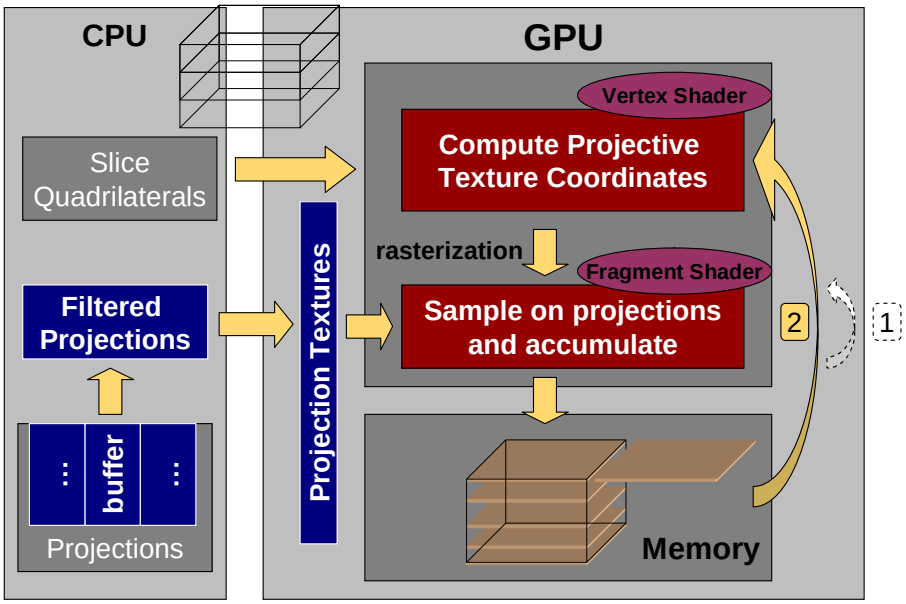


Figure 3.1: Streaming-CT pipeline

3.2.1 Projection-space Filtering

We choose to perform projection space filtering on the CPU, using the cache-optimized FFTW library [30]. This is reasonable since the FFT is a relatively sequential algorithm which involves many non-GPU-friendly operations, such as sorting, indexing and bit-wise calculations. Although a number of GPU-based FFT implementations are available [34] [88], significant speedups are obtained only for 2D and 1D FFTs with sufficiently

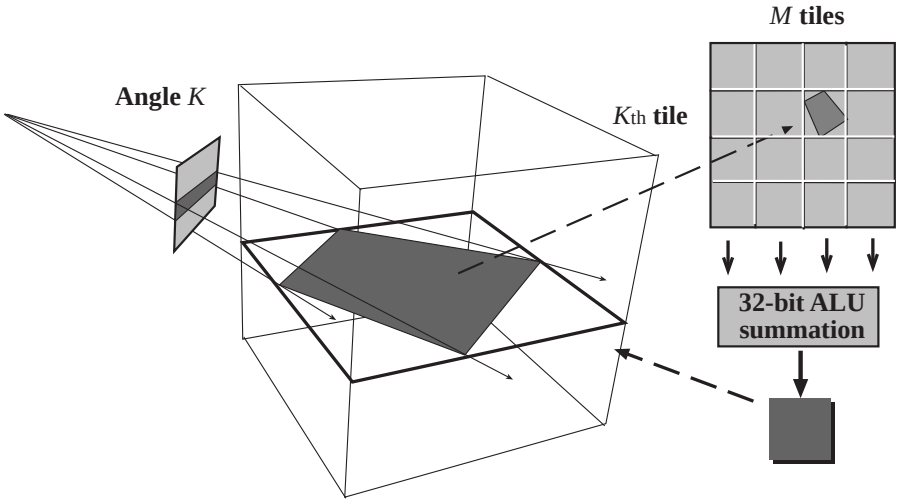


Figure 3.2: Slice-CT pipeline

large arrays (over 10k elements) - larger than those typically encountered in CT applications. This also resonates well with the declared goal to integrate the CPU as part of the computing pipeline, in order to make optimal use of all available resources. It is also theoretically justifiable, considering that the complexity ratio of a projection filtering vs. its backprojection ($O(N^2 logN)$ vs. $O(N^3)$) is in good correspondence to the performance ratio of CPU vs. GPU (1-2 orders of magnitude). Using the Pixel Buffer Object (PBO) and Vertex Buffer Object (VBO) software interface, in conjunction with the PCI-Express memory bus, this streaming can be well overlapped with ongoing backprojections and thus does not cause significant pipeline delays.

Another important issue is the precision used within the pipeline. While the projection data acquired by the detector typically have a dynamic range within 12-16 bits, the filtering potentially widens this range, and therefore maintaining full 32-bit floating point precision in later pipeline stages is most appropriate in clinical settings. For reconstruction on objects that do not require extreme accuracy such as industrial CT, we experimented with

different data formats in Section 3.3.1.

3.2.2 Backprojection

In order to describe a generalized mapping from volume space into perspective detector space, the (reconstruction) volume coordinate system (VCS) is described here by axis vectors (x_v, y_v, z_v) as the reference coordinate system, with the volume center at location (0,0,0) (see Figure 3.3). In this VCS, a given detector image P_ϕ has been acquired in a source-detector pair coordinate system (DCS_ϕ) described by axis vectors (x_ϕ, y_ϕ, z_ϕ). Here z_ϕ is orthogonal to the (flat) detector plane, the source is located at $s = -d_\phi z_\phi$ and the detector center is located at $(D_\phi - d_\phi)z_\phi$. A backprojection is the mapping of a voxel with VCS coordinates $r = (r_x, r_y, r_z)$ onto the detector plane, yielding coordinates $P_\phi(X(r), Y(r))$. Here, $X(r)$ and $Y(r)$ are scaling functions from VCS coordinates into detector pixel coordinates. After the mapping, an interpolation operator $Int()$ yields the backprojected voxel update $v_\phi(r)$, which is then depth-weighted according to the FDK equation:

$$\begin{aligned} v_\phi(r) &= \frac{d_\phi^2}{(d_\phi + r \cdot z_\phi)^2} \cdot Int(P_\phi(X_\phi(r), Y_\phi(r))) \\ X(r) &= \frac{r \cdot x_\phi}{d_\phi + r \cdot z_\phi} D_\phi, Y(r) = \frac{r \cdot y_\phi}{d_\phi + r \cdot z_\phi} D_\phi \end{aligned} \tag{3.1}$$

In the GPU context, the target volume is represented as an axis-aligned stack of 2D textures, since a single 3D texture does not support an efficient update mechanism, as mentioned before. As illustrated in Figure 3.1, the streaming-CT pipeline begins with generating a series of quadrilaterals P_i (called *proxy polygons*) which define the location and spatial extent of each volume slice as well as corresponding 2D textures T_i that contains voxel values to be reconstructed (initialized to zero). Then from a specific projection angle, for each such slice P_i, viewing its host polygon face-on in orthographic viewing mode produces the fragments that relate to the slice voxels (solid arrows pointing downwards in Figure 3.4). The vertex shader

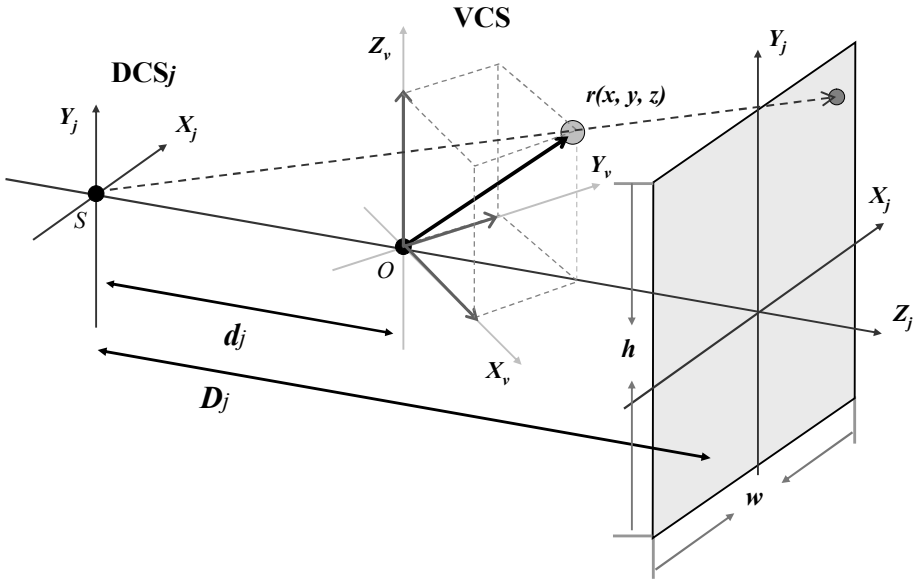


Figure 3.3: Geometry defined in the Feldkamp filtered backprojection algorithm. $\mathbf{VCS}(x_v, y_v, z_v)$: volume coordinate system; $\mathbf{DCS}_\phi(x_\phi, y_\phi, z_\phi)$: source-detector coordinate system; $\mathbf{r}$: voxel to be reconstructed; $\mathbf{O}$: rotation center (origin); $\mathbf{S}$: source; $\mathbf{w}/\mathbf{h}$: detector width/height in pixel counts; $\mathbf{d_{phi}}$: source-origin distance; $\mathbf{D_{phi}}$: source-detector distance.

then performs the mapping of these fragments into the perspective coordinate space of the back-projected image (the detector space). This is often referred to as *projective-textures mapping* (the dash arrow in Figure 3.4). An interpolation of this image at the mapped location produces the fragment value, which is then added/written to the corresponding output texture T_i representing the slice. These operations represent one rendering cycle (pass) on the GPU. New passes will be initialized and executed repeatedly until every volume slice is processed, from every projection angle.

In practical applications, the detector plane may not always be oriented perfectly collinear with the rotation axis for all angles. This may be due to (measurable) gantry instabilities, or it may be intended, in order to implement specific acquisition protocols on non-circular orbits. A series of matrix operation is used to represent the mapping, as shown in Equation (3.3). Since the mapping is perspective, 4D(homogenous) vectors and 4×4

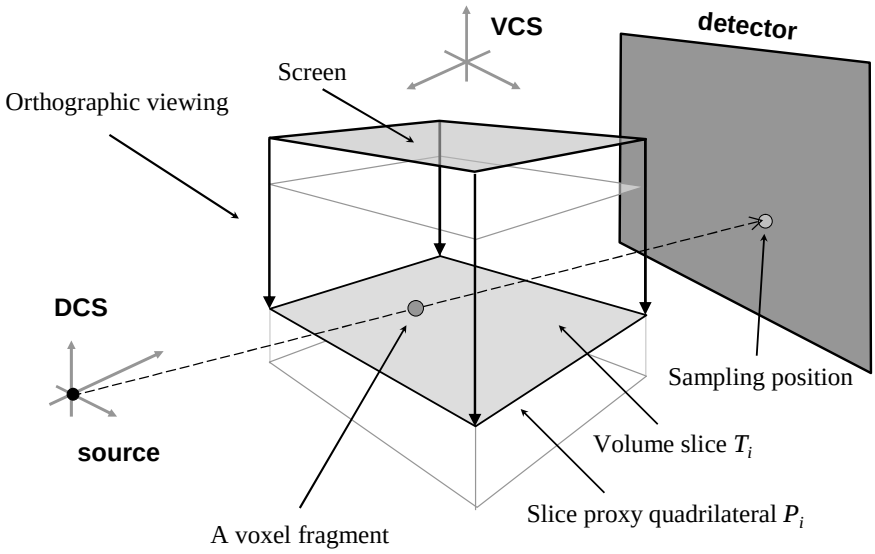


Figure 3.4: Backprojection of a volume slice on the GPU. An orthographic view (screen) onto the volume slice generates voxel fragments, each of which will then sample the detector (one fragment is shown here).

matrices are employed.

$$S \otimes T \otimes P \otimes M \otimes \vec{v} = \vec{v_h} \tag{3.2}$$

$$\begin{bmatrix} \frac{w}{2} & 0 & 0 & 0 \\ 0 & \frac{h}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1.0 \\ 0 & 1 & 0 & 1.0 \\ 0 & 0 & 1 & 1.0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{2D_\phi}{w} & 0 & 0 & 0 \\ 0 & \frac{2D_\phi}{h} & 0 & 0 \\ 0 & 0 & d/c & d/c \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_\phi^x & x_\phi^y & x_\phi^z & -x_\phi \cdot s \\ y_\phi^x & y_\phi^y & y_\phi^z & -y_\phi \cdot s \\ z_\phi^x & z_\phi^y & z_\phi^z & -z_\phi \cdot s \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} = \begin{bmatrix} x_h \\ y_h \\ z_h \\ w_h \end{bmatrix}$$

$$D_\phi = \frac{h}{2 \cdot \tan(\theta_\phi/2)} \qquad P_\phi(X,Y) = (\frac{x_h}{w_h}, \frac{y_h}{w_h})$$

Here, θ is the cone-angle and d/c indicates terms that are not needed, since we do not require z_h. The model-view matrix $\boldsymbol{M}$ transforms a voxel coordinate $\boldsymbol{r}$ from the VCS into the DCS. Another 4×4 matrix, $\boldsymbol{P}$, determined by D_ϕ, and detector dimensions w and h implements the subsequent perspective projection. $\boldsymbol{M}$ and $\boldsymbol{P}$ map $\boldsymbol{r}$ into a canonical viewing space, which is essentially a volume whose Cartesian coordinates are in [-1,1]. The following two transformations, translation matrix $\boldsymbol{T}$ and scaling matrix $\boldsymbol{S}$, are determined by the detector size (w and h) in pixels. Next comes the perspective divide, using the w_h term of the resulting 4D vector. This produces the (floating point) coordinates $P_\phi(X(r), Y(r))$ in detector pixel space. After interpolating the detector image the FDK weighting is performed, re-using the w_h term. This weight is essentially computed as $\mid z_\phi \cdot r - z_\phi \cdot s \mid$ representing the voxel's depth with respect to the source.

Note that this matrix only accommodates the case illustrated in Figure 3.3, that is, the source-detector pair can rotate in any (non-circular) orbit and orientation, but the center ray must pass through the rotation center (here the volume origin) and it must be orthogonal to the detector plane. However, generalizations of this can be easily incorporated into the $\boldsymbol{P}$ matrix by implementing a general viewing frustum (for more details of this mapping see [28] and [81]. This can accommodate any type of instable gantry situation, which often occur in practice.

For streaming-CT, once the fragment values have been generated, we accumulate them into the corresponding slice voxels. The above procedure is repeated for each volume slice until the entire volume is updated, and we then move to the next filtered projection generated by the scanner. The projection-processing iteration over all slice is illustrated as the solid arrow in Figure 3.1. For Slice-CT, the accumulation is postponed until all backprojection are completed from every projection. A large 2D texture sheet to hold $\boldsymbol{M}$ 2D texture tiles computed from $\boldsymbol{M}$ projection angles is prepared and each backprojected result is then directed to a designated area on the sheet, according to its projection angle $\boldsymbol{K}$ (see Figure 3.2). After all backprojections onto a given volume slice from all rotation angles have been completed and written to the sheet, an accumulation stage that consists of several passes is executed to combine multiple tiles into one. The separation of the backprojection and accumulation stages within a slice reconstruction gives rise to a reduced number of rendering passes of $\lceil M/L \rceil$ for accumulation, instead of $\boldsymbol{M}$.

With recent GPUs such as Geforce G80 series, the pipeline distinction

between the vertex and pixel engines is no longer made explicit in the hardware itself. For example, the NVidia 8800 GTX features 128 uniform SIMD (Same Instruction Multiple Data) processors. These can then be viewed either as a 128-way parallel processor, in the spirit of GPGPU and in association with the CUDA (Compute Unified Device Architecture) programming interface, or as a traditional graphics pipeline, in which case the processors are dynamically assigned to vertex and fragment operations in the manner described above. Therefore two choices are available: implement backprojection (i) as a Multi-Processing task using the 128-way parallel configuration (MP-GPU), or (ii) as a graphics task by ways of the Accelerated Graphics pipeline shown above (AG-GPU). Most previous GPU-accelerated CT reconstruction solutions, as the one of [102] mentioned before, have relied on the MP-GPU configuration, but as the comparison will show, the AP-GPU configuration is by far preferable, due to the fact that it benefits from the fast parallel ASIC rasterization hardware for the compute-intensive texture coordinate generation.

Using the Accelerated Graphics Pipeline (AG-GPU)

In the AG-GPU configuration both vertex and fragment shaders are used (see Figure 3.5(a)). First the matrix $T_r = S \cdot T \cdot P \cdot M$ for the specific projection is compiled and loaded into the vertex shader. Then, for each slice, the vertices of its proxy polygon are passed into the vertex shader, and the subsequent transformation produces the mappings of the slice vertices into the detector plane (the 4D coordinate vector in Equation 3.3). The rasterizer, in turn, produces bilinear interpolations of these coordinates, one for each slice voxel (mapped to its fragment). These interpolated 4D coordinates are the correct mappings for the slice voxels, since the matrix T_r originates in linear algebra and linear transformations. With each such fragment containing the vector $[x_h, y_h, z_h, w_h]$ for its slice voxel r, the fragment program then performs the final perspective division. This produces the detector coordinates needed for the sampling of the (filtered) projection image, which is streamed in as a texture from the CPU. The sampling position usually does not coincide with the detector pixels and a sampling kernel needs to be applied to produce final values. Here, the method employed for this sampling is important. We use bilinear interpolation, which in fact runs nearly at the same speed than

nearest neighbor interpolation on the Nvidia 8800 GTX, but produces superior results, especially in cases where the projection resolution is close to the volume resolution. The last portion of the fragment program is the computation of the FDK depth-weighting factor $(d_\phi/w_h)^2$ and the result is then written to the texture accumulating the backprojections. Thus, the overall length of the fragment program is quite short: two divisions, two multiplications, one (hardwired) interpolation, and one addition.

Using the Multi-Processor Configuration (MP-GPU)

In MP-GPU the generated fragments are processed in one uniform SIMD multi-processor of the GPU, most appropriately called the fragment shader (the vertex shader is not implemented). This mode is shown in Figure 3.5(b). Here, the processors, and not the built-in hardware, must perform the matrix-vector multiplication for each slice-voxel (represented by a fragment), in addition to the other operations also performed in the AG-GPU configuration. These additional calculations amount to 12 multiplications and 9 additions, which requires nearly twice as many clock cycles than the shader program of the AG-GPU configuration.

3.2.3 Additional Acceleration Strategies

RGBA Packing

Originally designed to process graphics primitives, the GPU architecture is capable of efficiently processing 4-component vectors containing red, green, blue and alpha properties of such a primitive. This feature offers an opportunity to obtaining a second level of parallelism, that is, channel parallelism. There are two different ways of implementation.

- When projections are acquired under parallel-beam geometry and have the same resolution than the object grid, adjacent volume slices share similar sampling patterns on the detector, which is independent of the rotation axis Y. Therefore, every four neighboring projection rows and corresponding volume slices can be packed into a texture of 4 channels and all the transformation and sampling can

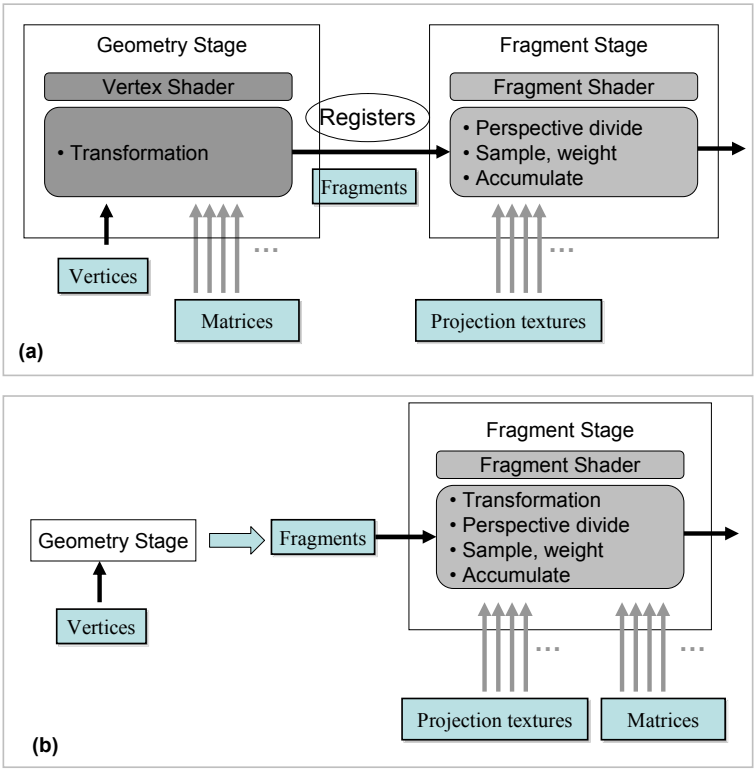


Figure 3.5: Two options for GPU-based accelerated CT reconstruction: (a) AG-GPU: accelerated graphics pipeline using both vertex and fragment engines; (b) MP-GPU: multi-processor configurations using fragment engine only.

be implemented concurrently on the GPU. This method reduces the number of rendering passes in both Streaming-CT and Slice-CT by four, and in practice yields a 2-3 fold speedup.

- When projections are obtained in cone-beam setup and collected at regular intervals over $360°$, we can process four projections spaced apart by $90°$ in four separate channels, without committing any errors (see Figure 3.6). These projections will share the GL transformation matrix, only the resulting backprojection result needs to be rotated by 90, 180, or 270 degrees. For this purpose the tile sheet is composed of 4 sub-sheets (RGBA), one for each rotation group.

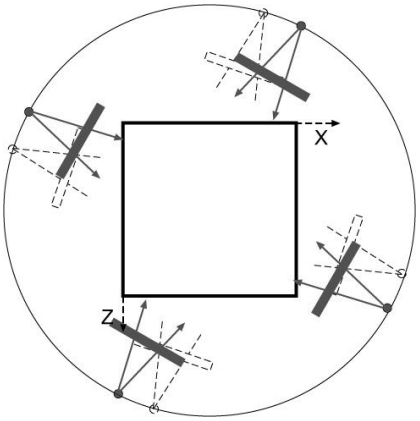


Figure 3.6: Access pattern for the orthogonal-angle backprojection in RGBA channels.

These sub-sheets are first accumulated in parallel, and the resulting four textures are then summed, after rotating them appropriately by a multiple of $90°$, in a final fragment program. Similarly we can pack adjacent incoming projections into 4 channels in the Streaming-CT framework to reduce the number of rendering passes. But there is no symmetry existed in the transformation matrices from adjacent angles which results in re-interpolation/re-sampling in every channel and yields only minor speedup thanks to the reduced number of rendering passes.

Early Fragment Kill

A source for additional speedups is a GPU facility known as early fragment kill (EFK), which can be exploited when the density-range or the spatial extent of the target object is known a-priori. In EFK a fragment is culled from the pipeline before it enters the fragment shader, thus causing near-zero computational overhead. If the spatial extent of the object is known or the reconstruction can be limited to a region of interest, then the GPU stencil buffer can be set to a bit mask, which is tested in hardware by a corresponding stencil threshold during rendering. The outcome of the test then decides if the fragment is culled. Since the stencil buffer has 32 bits, we divide the volume slice stack into 32 sub-stacks, find the

2D stencil for each, and store it in one of the 32 bits. This stencil mask is then loaded into the GPU at run-time. In addition, if the desired density range of the object is known then the current density of a reconstructed slice voxel (normalized to the number of backprojections applied so far) can be used to determine if a subsequent fragment is passed into the fragment shader or not. If a voxel's value is outside the density range of interest (plus a reasonable margin), then this slice fragment (corresponding to a voxel) is guaranteed to fall outside the structure of interest (that is, it is outside the shadow of the previously applied projections) and can be safely rejected from the pipeline using EFK. This is a conservative rejection criterion, with 100% sensitivity, and the specificity increases (that is, more fragments end up rejected) as more projections are being applied, since these lead to a better object definition. EKF can be controlled by copying the current density volume into either the depth- or stencil buffer, and then setting the appropriate depth- or stencil thresholds according to the desired density range (for more detail see e.g., [73]).

3.3 Practical Concerns

3.3.1 Precision Issues

Given a constant bandwidth processing rate on the GPU, the data format of textures will have a direct effect on its access/read speed in this streaming computational model. Fixed-point precision such as 8-bit integer data will yield faster performance but might compromise the computational accuracy. Therefore, it is important to investigate the performance of using different data format on GPU, in the sense of both computational speed and precision. Note that in the following we describe the experiments under the Slice-CT framework.

8-bit Integer

We observed that narrowing down the dynamic range of the filtered CT images from 32 bits to 8 bits and a subsequent backprojection can result in loss of detail. Figure 3.7 shows the reconstructed results at the original

(0.5%), double (1%) and quadruple (2%) contrasts, created from the projections analytically computed from Shepp-Logan phantoms of two different contrasts, filtered and narrowed down to 8-bit. We observe some streak artifacts on both images, and we also have difficulties to distinguish the three small tumors on the bottom in the lowest-contrast (0.5%) phantom. For the dataset with contrast constraint slightly weakened (1% and 2%), the tumors can be well separated and the streak artifacts are reduced. On the other hand, the speedup of using 8-bit is around 20 over the 32-bit floating framework, as shown in Table. 3.5.1. Thus, the integer precision is quite useful for object reconstruction, as long as the contrast requirements are not extremely strict. But we need do better for a diagnostic setting.

Figure 3.7: Feldkamp reconstruction from 8-bit data. Left: 0.5% contrast; Center: 1% contrast; Right: 2% contrast.

Pseudo-16bit

Since in our rendering framework, the RGBA channels are often fully occupied to obtain further speed-up, and currently 16-bit RGBA format is not yet supported by NVIDIA's GPUs, we devised a "double-precision" scheme to alleviate the narrow dynamic range presented by 8-bit integer data. For this, we first perform a compression of the dynamic range of the original floating point SLP projection data into 16-bit fixed point words. This is justifiable since data obtained from commercial scanners are usually never wider than 16 bits. Although filtering may result in a higher precision depth, we have not noticed any adverse problems due to this 16-bit compression.

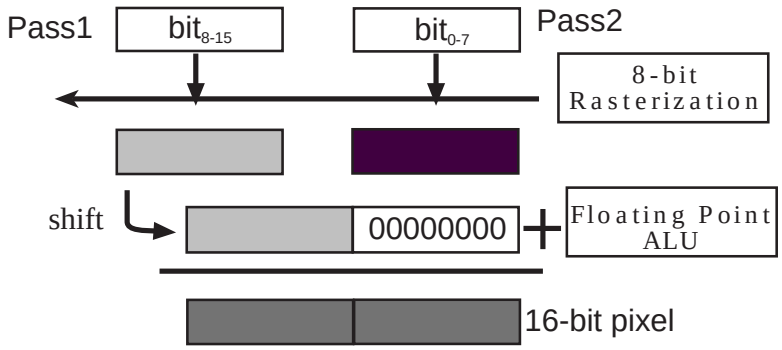


Figure 3.8: Virtual double precision (pseudo-16bit) through data splitting, individual rendering and summation

We then render these 16-bit words in two passes. First, every 16-bit word is divided into two bytes and stored in two individual 8-bit textures, containing the higher 8-bit and lower 8-bit words, respectively. Two back-projection passes are then performed on each 8-bit texture individually to obtain two tiled sheets, which are streamed into the floating point accumulation pipeline in turn. Here, we should note that the interpolation result obtained solely with the higher 8-bit texture will lose the lower 8 bits, if generated there. This is similar to the scheme presented in [70]. When assembling the two results, an additional shifting operation needs to be executed on the sheet containing the higher 8-bit information before summation (see Figure 3.8). Our technique is somewhat related to the work presented in [87] who emulated a complete set of arithmetic operations at 16-bit precision with RGBA8 textures on a NVIDIA GeForce 3, using basic arithmetic operations and dependent texture lookups. However, our scheme is much simpler, and while it is slightly less accurate, it is considerably faster.

Fig. 3.9 illustrates the reconstructed Shepp-Logan Phantom at 0.5%, 1% and 2% contrasts for the use of this technique, which shows that this scheme delivers excellent reconstruction results, despite its slightly reduced accuracy compared to a fully 16-bit approach. Also, by design, the total reconstruction time required with the extra spreading pass performed is just slightly over twice as long as that required for the single precision reconstruction (see Table 3.5.1).

Figure 3.9: Feldkamp reconstruction from pseudo-16bit data. Left: 0.5% contrast; Center: 1% contrast; Right: 2% contrast.

3.3.2 Large Datasets and Cache Performance

All popular PC graphics cards on the market have limited on-board memory. Given the potentially large magnitude of CT projection images (360 images of size 512^2 or 1024^2) and the resulting volume at a matching resolution, a trivial upscaling of the current implementation to these larger datasets will not scale linearly in performance. In our experiments, we have found that the increase in time spent for the reconstruction volumes of linearly increasing size is not linear (see Figure 3.10). In simple cases, the time required for the backprojection onto a slice of a 256^3 volume on a NVDIA 5900 is much higher than the expected four fold increase of the time incurred for the same operation for a 128^3 volume. A particularly dramatic increase can be observed near a specific dataset size (a volume size of 202^3 in Figure 3.10). This can be explained by an excessive number of cache misses, followed by texture reloads.

Texture compression strategies are often adopted to address problems due to insufficient memory space. The existing OpenGL compression extension (*ARB_texture_compression*) has been designed to boost pipeline efficiency. However, this lossy scheme does not satisfy the strict quality requirements of CT. Pre-compressing the CT images in a lossless way before texture loading, by ways of a user-defined compression scheme, is not practical either, due to the unavailability of corresponding decompression mechanism on the card, although this could be user-implemented.

We therefore adopt a scheme that alleviates the burden imposed by large images and volumes through partitioning and subsequent stitching

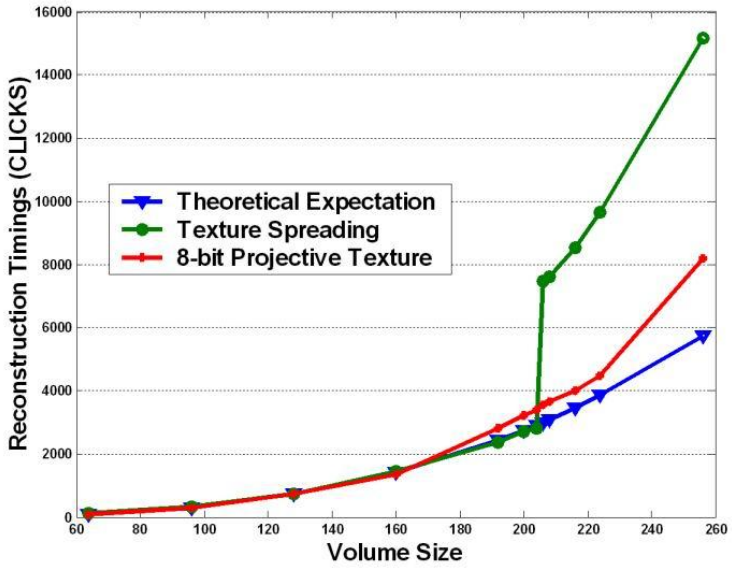


Figure 3.10: Nonlinear timing curve as a function of volume size (the size numbers are to be taken cubed).

(assembly) of the results. The idea is to control the scale of computation on the GPU in order to maximize the efficiency of the memory operations, while avoiding superfluous traffic incurred by large data. More concretely, the reconstruction task of a N^3 volume can be decomposed into 8 smaller volumes of size $(N/2)^3$ (see Figure 3.11). Reconstruction of each smaller volume still needs the participation of all projection images and will go through the same procedures with regards to projection and accumulation. But the complexity of all texture-related operations is kept at $O(N^3/8)$ and the required bandwidth is thus bounded. This scheme can be performed recursively until an appropriate size of the partial volume is found. We observed in our experiments on the current platforms that texture sizes of 128^3 exhibit good cache behavior. Thus, our decomposition with the current GPU platforms uses volume blocks of size 128^3. As a simple example, the top and bottom partial volumes shown in Figure 3.11 consist of reconstructions and assemblies from blocks 0-3 and 4-7, respectively, and are exported to the CPU in two transfers, if no subsequent visualization is needed. Else, we compress the volume section in a quick run length encoding (RLE), which compresses away the blank voxels near

the boundaries of the volume.

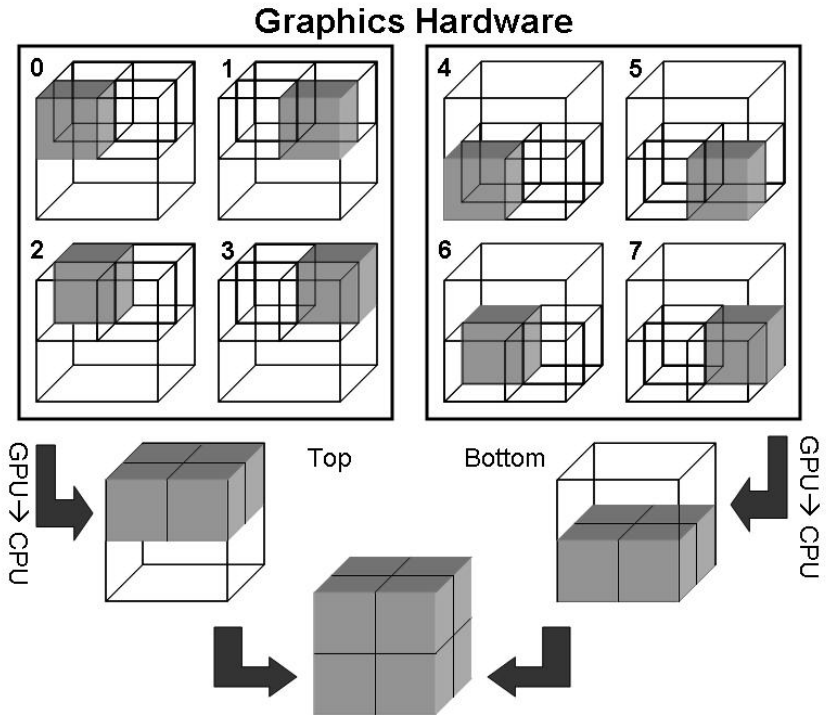


Figure 3.11: Strategy of texture partition and assembly.

For extra large projections exceeding the physical graphics memory, a straightforward compromise is to divide projection images into subsets, perform the reconstruction set by set and combine the individual results in the end. Another approach is to only reconstruct a certain range of volume slices at a time. This requires the decomposition of the projections into a corresponding set of segments. One can still use the volume block-scheme described above to keep below the cache limits, and for both strategies the projections for the next slice range can be streamed in while the previous reconstruction is still in progress, since loading and GPU calculations can occur simultaneously. Currently, we always decompose our volume into their upper and lower halves, due to the independent calculation of these volumes.

3.3.3 Load Balancing

The powerful computational capability of GPU does not come without careful planning and creative mapping of the target process onto the hardware's computational model. A correct assignment of the task to the various GPU pipeline components and an efficient management of the GPU resources are crucial to harness and maximize the available computational power. Refer to the callouts in Figure 3.12 where we indicate the locations in the pipeline at which bottlenecks usually occur. One bottleneck occurs (callouts 2, 3) when fragments, streaming out of the rasterization engine, are being pooled into the multiple pixel pipelines, waiting to be processed in the shader (where they usually request access to the texture stream). A complex shader will result in stalling the pipeline because the rasterization process outpaces the shader's processing speed and as an effect produces more fragments than the shader can digest. The situation will further deteriorate if the shader contains too many I/O requests from the texture (callout 4), which are expensive operations compared to arithmetic instructions. The latter usually occurs when sophisticated sampling (interpolation) schemes are performed, requiring access to larger patches of texture data. Another bottleneck (callout 1), resulting from these previous bottlenecks and/or from insufficient processing power, manifests itself by an overload of input data (projections and slices). It occurs when the processor cannot keep up with the projection acquisition, which is exactly what our streaming CT seeks to prevent.

For CPU-based programs, the use and scheduling of the cache as well as other lower-level memory plays a very important role in overall performance. Every cache (and memory) fault will lead to a delay within the ongoing computation, and much work has been done to optimize memory access patterns for an abundance of diverse application scenarios. This situation is considerably more complex on GPUs since they consist of several components: (1) the cache and the main memory interface, but also (2) the active units embodied by the rasterizer and the two programmable, SIMD-parallel shaders. Thus, while CPU-based programs only need to minimize wait for memory access and data delivery, GPUs must, in addition, also balance the data flow across its several active units.

Therefore, load balancing is a very important issue in our high performance streaming computation framework. A balanced load will avoid

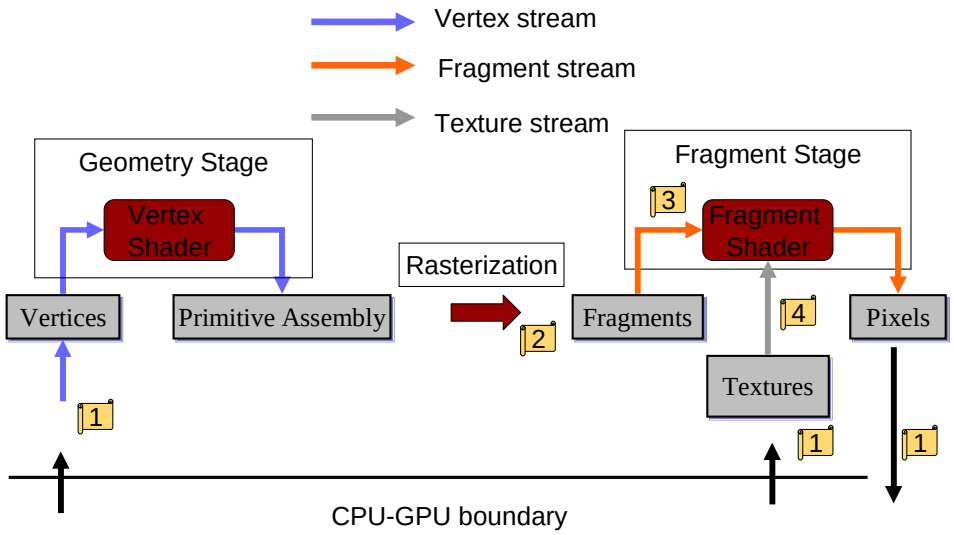


Figure 3.12: GPU load balance pipeline.

the accumulation of intermediate data within the pipeline, which is synonymous with a wait-free overall state. Consider again Figure 3.12, where the four callouts illustrate the locations where pipeline stalling may occur. This is the case when the data production rate of the previous stage does not match the consumption rate of the upcoming stage. If it is greater, then data accumulation will occur, while if it is less, then the upcoming stage stalls. For example, processing a single projection in one pass, which includes instruction execution and texture sampling (callout 3 and 4), cannot produce enough workload on each fragment generated from the rasterization engine. This results in the situation that the pipeline inside the shader may be idle, waiting for more fragment output from the rasterization. To eliminate this bottleneck, we created a projection buffer on the GPU to temporarily store the acquired and filtered but not yet backprojected projections streamed in from the CPU. Here, the number of projections held in the buffer is controlled by a window, which we can dynamically adjust via the CPU-resident control process in order to optimize the load. When using this projection buffer, the first cycle must wait until the buffer has been filled, but for the remaining cycles the filling of the buffer with the next batch occurs while the previous batch is being processed, thus there will be no wait thereafter. In practice, when passing a batch of projection

textures into the shader, we also transmit their acquisition geometry information in order to perform multiple backprojection samplings for a given slice. This multiplicity has already been indicated in Figure 3.5 (by the multiple arrows). While the MP-GPU approach (Figure 3.5(b)) can cope with arbitrary window sizes, the AG-GPU approach is limited by the number of floating point texture coordinate registers a rasterized fragment has available. These are needed to pass the interpolated texture coordinates and weights for each projection. The current hardware has a maximum of 8 such registers, which, however, did not pose a limit for our application.

The remaining bottleneck, callout 1, often occurs due to the slower data transfer rate on the PCI Express bus connecting CPU and GPU, but for synchronizing the GPU calculation and the data transfers from CPU to GPU, we take advantage of the new Pixel Buffer Objects (PBO) and Vertex Buffer Objects (VBO).

For multi-GPU board configurations, the projection buffer is duplicated across different nodes and each GPU performs the same scheme, as described in detail above. The CPU-based control process will then fill each GPU buffer appropriately, with as many projections as determined by the optimal window size. In order to minimize the initial buffer fill rate, projections are assigned to the individual GPU projection buffers in a round robin fashion. Dual-CPU PCs share the load for filtering. There is only little overhead associated for communication during the reconstruction, and once all GPUs have completed their volumes, they are added together (using a hierarchical algorithm when a quad-GPU is used) after which one GPU contains the entire volume.

3.4 Visual-CT

In some application settings it can be useful to monitor and track the state of the evolving object as the incoming projections are applied. A traditional choice has been to simply view selected volume slices containing the features of interest. But if it is the goal to get a quick insight into the three-dimensional shape of an object, for example, a part of the musculoskeletal system, direct 3D volume rendering (DVR) may be a better option. Further, in some cases it may also be of interest to generate real-time X-ray views from an orientation different than those generated by the

scanner, which can provide useful hints even if the object is not fully reconstructed. Since X-ray viewing, and with some additional overhead also DVR, are practically inverses of the rapid backprojection operations supported by our framework, it is relatively straightforward to inject an occasional (or frequent) X-ray or DVR pass into the on-going reconstruction. The fragments are then due to the pixels of the displayed image, while the slices of the reconstructed volume for the textures, which are interpolated by the mapping operation. We call this capability Visual CT. Visual CT has two modes of operations, depending if the desired structures are known a-priori or not. In the former case, a DVR can be easily obtained, given previously generated mappings of the Hounsfield densities to visual parameters, such as color and transparency (i.e., opacity). Since a DVR must also include shading and transparency compositing, these operations must be added to the shader program. In the latter case, the real-time reconstruction will likely prohibit the somewhat time-consuming parameter tweaking needed to establish a good DVR from unknown densities, and thus an X-ray or MIP (Maximum Intensity Projection) rendering from a novel view point is probably more desirable.

3.5 Results

3.5.1 Precision

For the precision and large dataset techniques presented in the Slice-CT framework, Figure 3.13 compares some reconstructions of the Shepp-Logan Phantom at 3 different contrasts (the original 0.5%, a more moderate 1% and 2%), obtained with both the single and the dual-pass approach. The line plots show the intensity profile of a 1D cut across the 3 small tumors near the bottom of the 3D SLP. Table 3.5.1 shows the performance obtained with the different settings discussed here (for an NVIDIA 7800 FX GPU), all using the 4-channel parallelism (RGBA) for the reconstruction of a 256^3 volume. We see that the better data caching achieved with the data partitioning (P) allows speedups of 2-3, both at floating and at fixed-point precision. We also observe that while the double-pass approach (DP) does run about 1.5 times slower than the single-pass approach (SP), it is still

Mode	Time
CPU float	180s
RGBA float	42s
RGBA float + P	20s (2)
RGBA SP	6.1s (7)
RGBA SP + P	1.9s (21)
RGBA DP	9.0s (5)
RGBA DP + P	3.0s (3)

Table 3.1: Slice-CT timings (speedups in parentheses) with various conditions discussed (**RGBA**: 4-channel packing; **P**: partitioning; **SP**: 8-bit, single pass; **DP**: pseudo 16-bit, double pass).

14 times faster than the floating point approach, while producing an image that is visually very close to the original. It is also 60 times faster than a pixel-driven CPU floating point implementation. Although there is a small amount of noise that can be detected in the line plot, the level is too small to show up in the image. We should also mention that our partitioning strategy provides the desired linear scaling in dataset size - a reconstruction into a 128^3 volume completes at 0.2s in the single-precision (SP) mode and at 0.4s in double-precision (DP). Finally, Table 3.5.1 shows the running speed for reconstructions from projection images increasing in size. Although we did not increase the size of the reconstructed volume, the larger resolution of the projections yields better interpolation results in the reconstructions, which leads to higher reconstruction fidelity, when the object requires it (the SLP can be reconstructed well with the 256^2 projections). We observe that the reconstruction runs at the expected speed as long as all projections fit into memory. Extra large images that exceed the physical onboard memory are divided into subsets small enough to fit. The reconstruction is then run individually and the results are summed together.

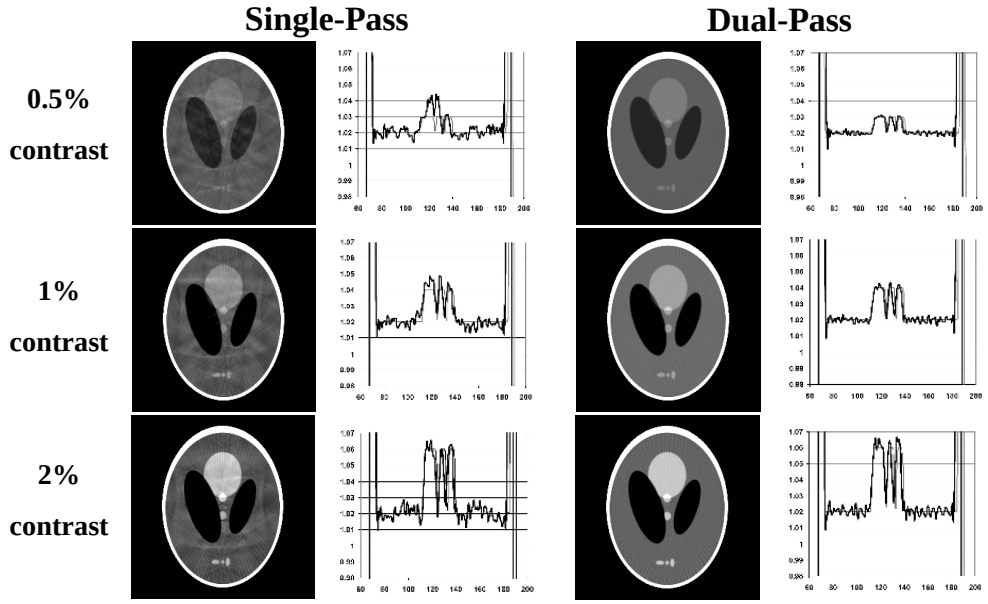


Figure 3.13: A slice of the 3D Shepp-Logan Phantom reconstructed at 256^3 resolution from 160 projections, obtained from 8-bit and pseudo 16-bit computation under various contrast settings.

3.5.2 Reconstruction Quality

To further test the performance of the Streaming-CT framework, we used a 3D version of the Shepp-Logan phantom as well as a variety of medical CT datasets (a human head, human toes, and a stented abdominal aorta) to test our framework. All experiments were performed on a 2.2GHz dual-core Athlon PC with 1GB RAM, equipped with NVIDIA Geforce 8800 GTX card (768 MB on-board memory). The phantom projections were calculated analytically, while the medical projection data were obtained by performing high-quality X-ray simulations on existing CT volume datasets. All projections were acquired on a full circular orbit at a 15° cone angle.

For the Shepp-Logan phantoms, we used 360 projections of size 512^2 each to reconstruct a 512^3 volume. Figure 3.15 shows slices from the reconstructed 3D Shepp-Logan phantom (at the original 0.5% contrast) from both parallel-beam ($0°$) and cone-beam ($15°$) projections, each obtained

Projections	Memory	Time
160×256^2	20 MB	3.0s
$160 \times 1024 \times 768$	420 MB	4.0s
$320 \times 1024 \times 768$	540 MB	6.7s

Table 3.2: Performance of Slice-CT with larger input datasets, typically used in clinical practice.

with our Streaming-CT framework as well as a traditional high-quality CPU implementation. We also compared two interpolation schemes, box (nearest neighbor) and bilinear. A Shepp-Logan filter was used for pre-filtering. The shifted profile in the cone-beam images results from the non-exact FDK reconstruction of the off-center slices [44] [94]. We found that both interpolation kernels, bilinear and box, yield excellent and, by visual inspection, nearly identical results. We observe that the box filter tends to produce somewhat sharper, but also noisier images. These differences, however, can only be discerned when comparing the values on an intensity profile, such as the one across the small tumors in the bottom. Figure 3.16 compares slices reconstructed from the simulated projection data (top row) with the corresponding slices from the original volume dataset (bottom row). We observe that the quality of the reconstruction and the original is nearly identical. The very slight blurring most likely stems from the inherent low-passing in the resampling during the projection simulation. Finally, Figure 3.17 presents a set of novel slices and volume rendered images of the toes dataset, obtained with the Visual CT reconstruction monitoring facility.

3.5.3 Reconstruction Performance

Table 3.3 compares the overall performance of our streaming-CT framework with a selection of other current high-performance FDK-based CT reconstruction solutions reported in the literature. To enable a comparison, we scaled all of these to a currently common problem size, which we have also used for our own experiments: the reconstruction of a 512^3 volume from 360 projections (the size of the projections is irrelevant, except for filtering, since the backprojection is mostly determined by the volume

size). We also use the metric projections/s to indicate the potential for real-time (streaming) reconstruction, currently requiring processing rates of 30-50 projections/s.

	Hardware Platform	Mechanism	Time (s)	Projs/s
Brasse *et al*	1 dual-core Xeon CPU 2.6GHz		3705	0.1
	12 dual-core Xeon CPU 2.6GHz		236	1.6
Goddard and Trepanier, Lesser *et al*, Li *et al*	FPGA		40.2-46.4	8-9
Kachelrieß *et al*	CPU	Hybrid	135	2.6
	Cell BE	Direct (Hybrid)	19.1 (9.6)	19 (37)
Streaming CT	GPU Nvidia 8800 GTX	MP-GPU	24.8	14.5
		AG-GPU	8.9	40.4
		AG-GPU with EFK	6.8	52.5

Table 3.3: Reconstruction speeds of various high-performance CT solutions. Timings have been normalized for a Feldkamp cone-beam CT reconstruction with 360 projections onto a volume grid of 512^3 resolution (note, not all implementations employ 32-bit floating point precision, bilinear interpolation, and generalized source-detector positioning, which are all used for our streaming CT application).

In Table 3.3, the methods labelled 'direct' employ the full 3D projection matrix (Equation 3.3) when mapping a voxel onto the detector plane, allowing practical scanning situations in which the detector - source pair need not be confined to a aligned, perfectly circular orbit. In contrast, the method labelled 'hybrid' uses data that stem from resampling the acquired projections into a virtual detector, which conforms to this ideal circular orbit. The backprojection is then performed into this arrangement, which reduces and simplifies the back projection matrix and thus allows for faster

voxel-projection mapping. All of our GPU-solutions use the general, direct projection scheme. We first observe that neither FPGA nor CPU-based solutions have reported processing rates of greater than 10 projections/s. On the other hand, our AG-GPU solution achieves the desired real-time projection throughput rate (40 projections/s) - however, the MPGPU solution does not. Further, while the Cell BE hybrid solution is quite competitive to our standard AG-GPU method, the Cell BE direct method is only comparable to the MP-GPU solution. This match is understandable since the Cell BE is a multi-processor architecture, but without specific graphics hardware support.

Next, we show results using the EFK GPU-facility. Since this is clearly data-dependent, we only show the performance for the most frequent case, that is, when the reconstruction focus falls into the maximal spherical region covered by all projections. We observe that this can achieve a further speed-up of factor 1.3, which enables data acquisition rates of over 50 projections/s or reconstructions of larger volumes at 30 projections/s. cone-beam reconstruction (512^3 volume, 360 projections, direct method, 32-bit floating point precision, bilinear sampling) using different buffer (window) sizes.

Finally, Figure 3.14 graphs the effect of window size on reconstruction performance for the AG-GPU solution. We found that a window size of 8 yields the best results for our specific experiment setting, but it can be easily adjusted to fit others. We also see that load-balancing can have a dramatic positive effect, here a speedup of 4 (comparing the no-buffer case with the case when the projection buffer size is 8).

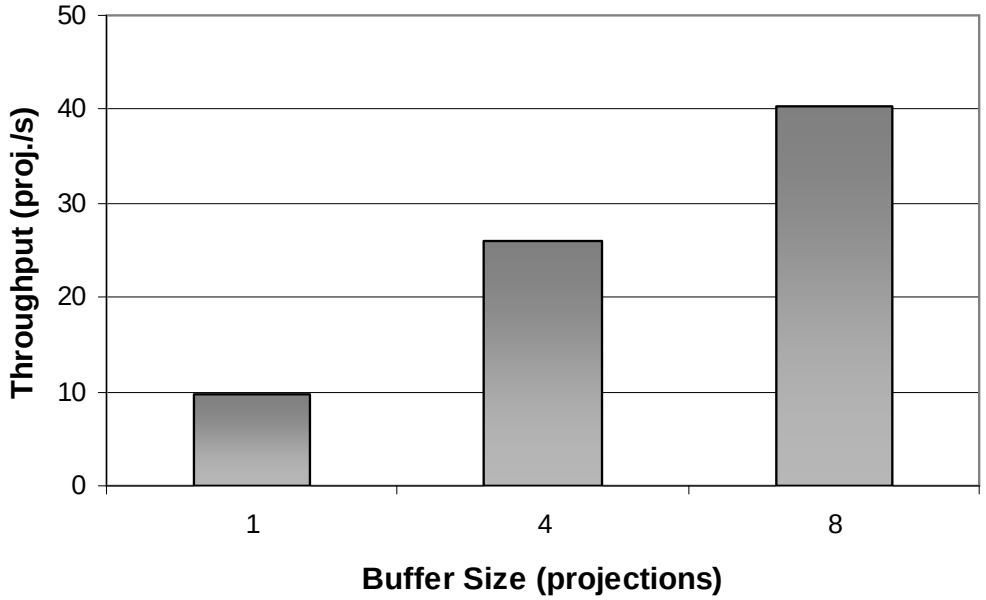


Figure 3.14: Streaming CT performance for a Feldkamp cone-beam reconstruction (512^3 volume, 360 projections, direct method, 32-bit floating point precision, bilinear sampling) using different buffer (window) sizes.

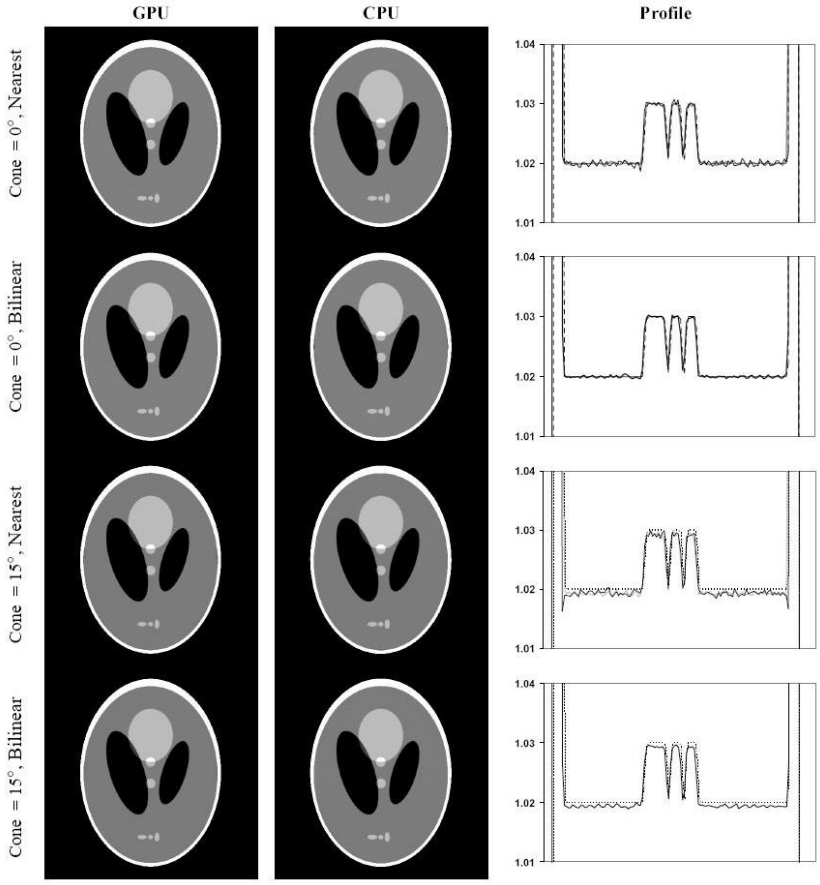


Figure 3.15: A slice of the 3D Shepp-Logan phantom, reconstructed with our streaming-CT GPU-based framework (first column) and with a traditional CPU-based implementation (middle column). A windowed density range of [1.0, 1.04] is shown. The right column shows the line profiles across the three tumors near the bottom of the phantom (dashed lines: ground truth; solid gray lines: CPU results; solid black lines: GPU results). We observe that the GPU reconstructions are essentially identical to those computed on the CPU and that they represent the original phantom well, for both parallel-beam and come-beam. The bilinear filter yields slightly smoother profiles than the box filter, but the reconstruction quality does not suffer significantly when using nearest-neighbor interpolation.

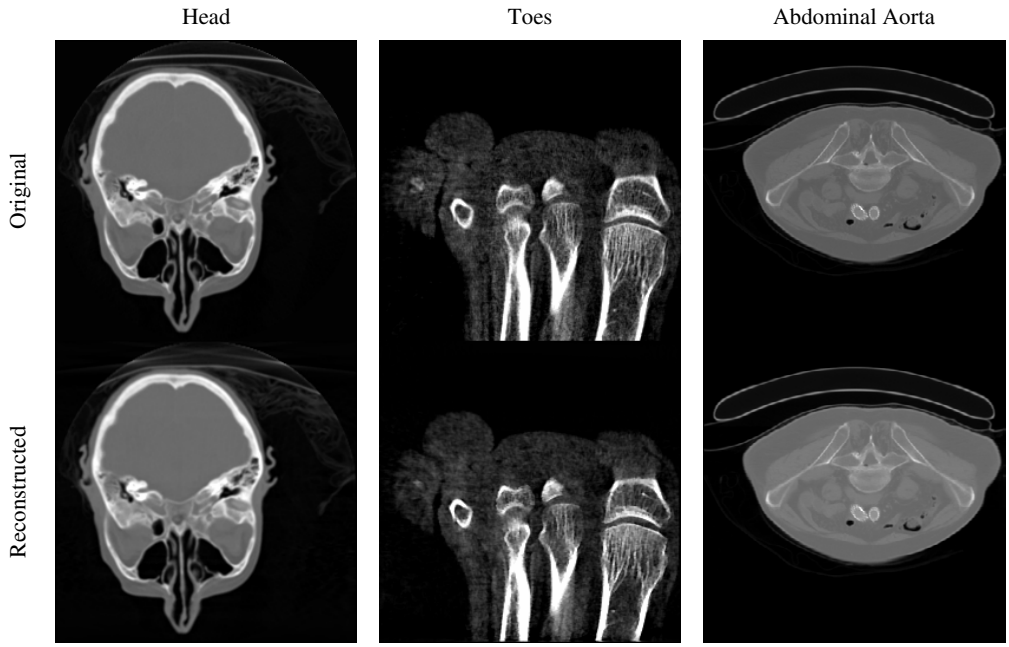


Figure 3.16: Slices of streaming-CT reconstructions from simulated projection data of three representative medical volume datasets (left to right): a human head, human toes, and a stented abdominal aorta. The slight blurring stems from the (minimal) low-passing induced by the resampling during simulation.

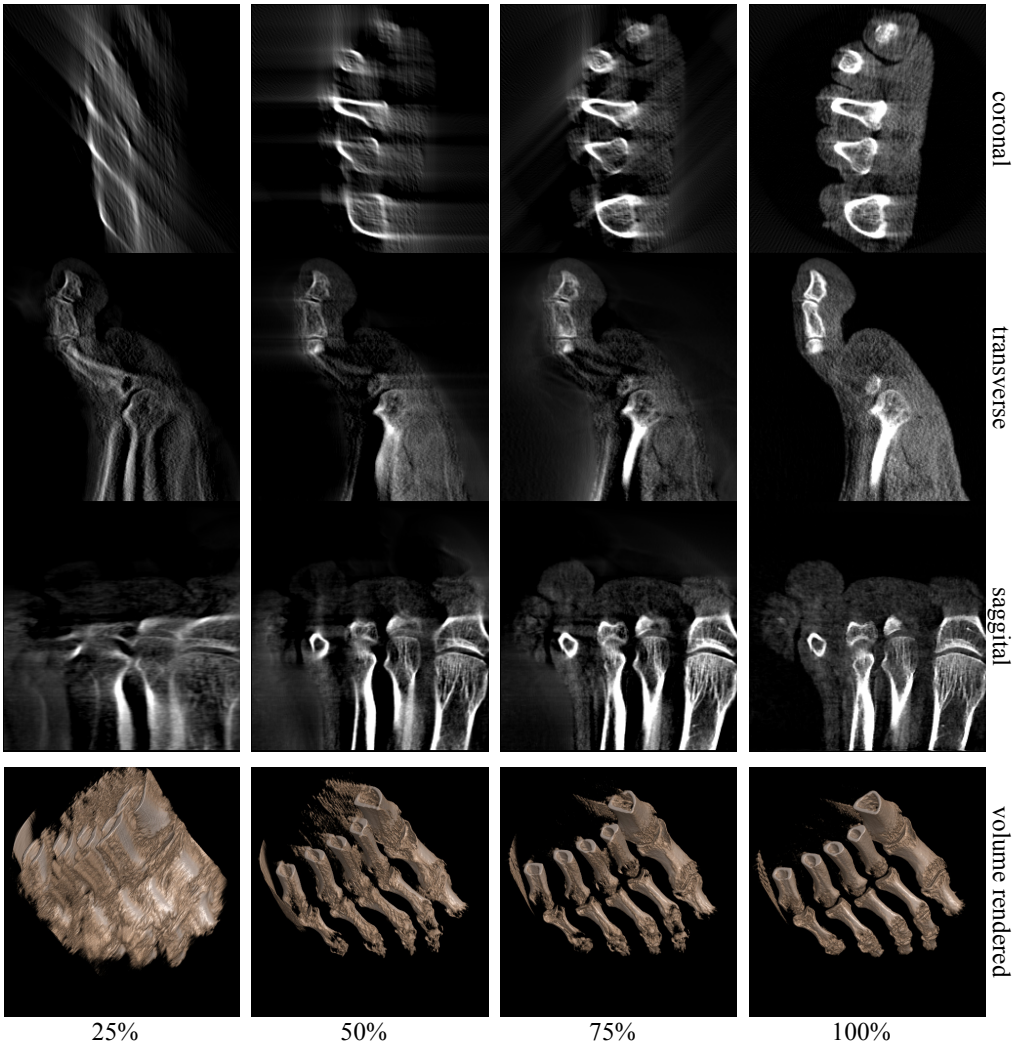


Figure 3.17: Slices and volume rendered images obtained via Visual CT.

Chapter 4

Electron Tomography Application

4.1 Introduction

Electron Tomography (ET) uniquely enables the 3D study of complex cellular structures, such as the cytoskeleton, organelles, viruses and chromosomes. It recovers the specimen's 3D structure via computerized tomographic (CT) reconstruction from a set of 2D projections obtained with Transmission Electron Microscopy (TEM) at different tilt angles. ET can be accomplished using exact analytical methods (weighted back-projection WBP) or via iterative schemes, such as the Simultaneous Algebraic Reconstruction Technique (SART) [2], the Simultaneous Iterative Reconstruction Technique (SIRT) [32], and others. The dominant use of the analytical methods is most likely due to their computational simplicity and consequently fast reconstruction speed. Iterative methods, however, have the advantage that additional constraints can be easily and intuitively incorporated into the reconstruction procedure. This, for example, can be exploited to better compensate for noise [84] and to perform alignment corrections [21] [29] [55] during the iterative updates. Additional challenges are imposed by the fact that the projection sinogram is vastly undersampled, both in terms of angular resolution (due to dose constraints) and in terms of angular range (due to limited sample access). These types

of scenarios can be handled quite well using iterative reconstruction approaches [1]. Thus, iterative approaches have great potential for ET. However, as data collection strategies [104] and electron detectors improve, the push has been to reconstruct larger and larger volumes ($2048^2 \times 512$ pixels and beyond). Although the benefits are significant, the major obstacle preventing the widespread use of iterative methods in ET so far has been the immense computational overhead associated with these, leading to reconstruction times on the order of hours to days for practical data scenarios. As in many other scientific disciplines, the typical solution to meet these high computational demands has been the use of supercomputers and large computer clusters [27] [105], but such hardware is expensive and can also be difficult to use and gain access to. Fortunately, the recently emerging commodity programmable computer graphics hardware boards (GPUs) offer an attractive alternative platform, both in terms of price and performance, leading to a new trend called General Purpose Computing on GPUs (GPGPU) [74]. GPUs are available at a price of less than $500 at any computer outlet and, driven by the ever-growing needs and tremendous market capital of computer entertainment, their performance has been increasing at a triple of Moore's law, which governs the growth of CPU processors. For example, the latest NVIDIA GPU board, the 8800GTX, has a peak performance of nearly 500 billion floating point operations per second (500G Flops), which is 1-2 orders of magnitude greater than that of a state-of-the-art CPU. This high potential of GPUs for accelerating Computed Tomography (CT) has been recognized for quite some time in the field of X-ray CT [13] [16] [50] [69] [80] [97] [98] [101] [102], and more recently also for ET [21]. The majority of GPU algorithms developed for X-ray CT have focused on 3D reconstruction from data acquired in perspective (cone- and fan-beam) viewing geometries, using flat-panel X-ray detectors in conjunction with X-ray point sources. This poses certain constraints on how computations can be managed (pipelined) given the highly parallel SIMD (Same Instruction Multiple Data) architecture of GPUs. However, data acquisition in ET is typically posed within a parallel-beam configuration, and this allows for additional degrees of freedom in the implementation, which are not available in the cone- and fan-beam configurations. Our approach exploits these opportunities to derive a novel high-performance GPU-accelerated iterative ET reconstruction framework.

The GPU method proposed by Diez et al. can be viewed as a first step

towards achieving high-performance ET. Our framework is a substantial advance of their method, speeding up their calculations by an order of 1-2 magnitudes. Such speedups are especially significant when it comes to 3D reconstructions, which are the ultimate goal of ET. The method of Diez at al. enables only 2D iterative reconstructions to be accomplished at reasonable speeds (where reasonable is defined here as being on the order of minutes). However, reconstructions at the same resolution, but in 3D, still take hours to compute. Our framework, on the other hand, obtains these 3D reconstructions in an order of minutes, on comparable hardware. Furthermore, large 2D reconstructions of 2048^2 pixels, at 50 SIRT iterations, can now be obtained at (near) interactive speeds, at an order of seconds, and not minutes. Finally, the latest generation of GPU hardware enables further considerable speed increases, which may be valued as another demonstration of the immense potential GPUs have for iterative ET.

SIRT is a commonly used reconstruction algorithm in ET and has been shown to produce good reconstruction results. On the other hand, SART has been shown to converge at considerably faster rates, but generating noisier reconstructions. This occurs since each update/correction is only based on a single projection and therefore does not enjoy the stabilizing effect of a global SIRT update (it is therefore advisable to choose a suitable relaxation factor less than 1.0 to scale each SART update). We take the stance that SIRT and SART are really just the two extreme cases of what can be called Ordered Subsets SIRT (OS-SIRT), where SART has N subsets of 1 projection each, and SIRT has 1 subset of N projections (with N being the number of projections acquired). It has been shown for the Expectation Maximization (EM) algorithm [82] that ordered subsets can lead to faster reconstruction convergence - a fact that gave rise to the Ordered Subsets EM (or OS-EM) algorithm [40]. In OS-EM, iterations are decomposed into exclusive projection sets, where the size of each set can be chosen freely. In this chapter, we examine the effects of different subset sizes in terms of the algebraic reconstruction paradigm that underlies both SIRT and SART. We study the dependence of subset size, reconstruction quality and fidelity, and wall-clock time to convergence. Here, we find that OS-SIRT allows one to balance blurring and noise artifacts, just by using different subset sizes, and that there exists an optimal subset size that delivers the best reconstruction in the smallest amount of time. Finally, an important issue in CT is the "long object" reconstruction problem. It arises in spiral CT

when the goal is to reconstruct a region-of-interest (ROI) bounded by two trans-axial slices, using a set of axially truncated cone-beam projections corresponding to a spiral segment long enough to cover the ROI, but not long enough to cover the whole axial extent of the object [54]. Essentially, in this situation some rays used for ROI reconstruction also traverse object regions not within the ROI, and these rays are sometimes called "contaminated" rays. This problem is similar to the "local tomography" problem in ET. While for ET the data acquisition trajectory is orthogonal to the one in spiral CT, ray contamination occurs whenever the object contains material is not covered by every projection (that is, only a sub-region of the object is exposed to electrons in a local view). These areas are then incompletely reconstructed in the iterative procedure, which is evidenced by vignetting - a brightness fall-off in the peripheral regions of the reconstructed object. We derive a method, within our iterative framework, which effectively compensates for this effect, correcting the contaminated rays for the missing information.

4.2 Methods

The dimension of collected tomographic projections and the target volume to be reconstructed in ET is normally huge, varying between 512 to 2048 pixel/voxel counts, which turns out to be a data intensive problem for GPU-based application. For relatively smaller dataset investigated before, we used to represent the volume into two stacks of slices (Figure 4.1a) that are orthogonal to the data collection path. The selection of stack during the forward-projection and back-projection stages depends on which stack is more parallel to the viewing rays. The past study has shown a successful implementation based on the above representation. Considering the magnitude of data encountered in ET study and its specific parallel-beam geometry, we decided to switch to the one stack representation (Figure 4.1b) with volume slices parallel to the data collection path due to the following reasons:

- During the whole reconstruction, two-stack representation requires two copies of the volume to stay in the limited GPU memory that turns out to be too small for large data processing jobs such as ET

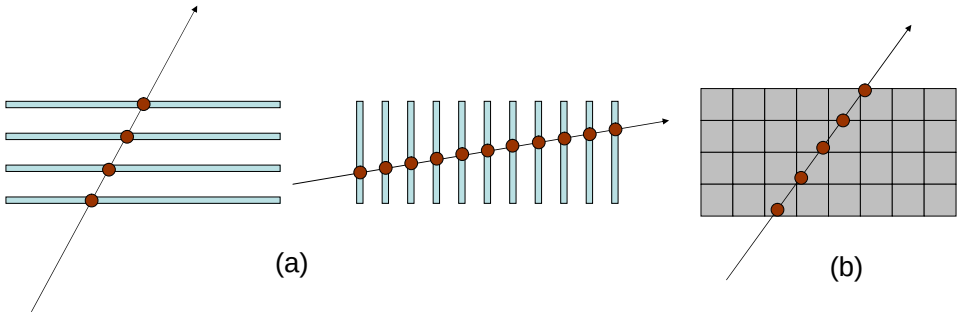


Figure 4.1: Data representation and sampling: (a)two-stack; (b) one-stack.

(on-board memory provided by current hardware is around 256/512 MB).

- In the iterative reconstruction an additional shuffling operation is required to be performed to maintain the data consistency between two slice stacks when the two-stack representation is used. Although the cost of this extra processing step can be minimized by only performing when target slice stack changes, nevertheless there is still overhead in terms of performance and data management.

- Due to the geometric limitation of the axis-aligned proxy, only on-slice nearest/bilinear interpolation can be performed on the two-stack representation, let alone the sampling steps/discretized ray segments vary from angle to angle. Despite the fact that our previous work shows it neither imposes inaccuracy nor affects the convergence speed with further optimization methods such as denser integration path or over-sampled detector, the one-stack representation is still more flexible in applying different interpolation kernels and adjusting the number of discretized integration segments.

- The one-stack representation has a high degree of data independency between slices under the parallel-beam reconstruction scenario. It is much easier to divide the target volume on a slice basis and distribute onto different computation nodes to implement a CPU/GPU cluster framework, compared to the blocking-scheme discussed in

Section 3.3.2. This high parallelism also enables an easy use of GPU's inherent RGBA color channels.

In the following sections, we first give some theory on OS-SIRT and then describe how these theoretical considerations affect and control our acceleration strategies. Then we describe an efficient projection and back-projection operator. The remaining operations, such as the correction computations, are simple vector operations of low complexity and can be implemented on the GPU by subtracting two 2D textures, the texture holding the acquired projections and the texture computed during projection.

4.2.1 OS-SIRT

We have rewritten Equation 1.5 as a generalization of the original SART and SIRT equations to support ordered subsets for our OS-SIRT:

$$v_j^{(k+1)} = v_j^{(k)} + \lambda \sum_{p_i \in OS_s} \frac{p_i - r_i}{\sum_{l=1}^{N} w_{il}} \qquad r_i = \sum_{l=1}^{N} w_{il} \cdot v_l^{(k)} \tag{4.1}$$

Here, the p_i are the pixels in the acquired images that form a specific subset OS_s, where $1 \leq s \leq S$ and S is the number of subsets. The factor λ is a relaxation factor, which will be chosen as a function of subset size (for SIRT where $S = M, \lambda = 1$, M is the total number of projections). The factor k is the iteration count, where k is incremented each time all M projections have been processed. In essence, all voxels v_j on the path of a ray r_i are updated (corrected) by the difference of the projection ray r_i and the acquired pixel p_i, where this correction factor is first normalized by the sum of weight encountered by the (back-projection) ray r_i. Since a number of back-projection rays will update a given v_j, these corrections need also be normalized by the sum of (correction) weights. Note that for SIRT, these normalization weights are trivial.

Ordered subsets are related to block-iterative methods. Since algebraic methods pose the reconstruction problem as solving the matrix $P = WV$ (where P, W, V are the projection vector, system matrix, and voxel vector, respectively), the first algebraic technique, ART [33], is also often called a row-action method since an update is performed for every ray (which

represents one row of the system matrix). Using more than one row of W per update then results in a block-iterative scheme. In that context, SART is a block-iterative scheme, where each block is composed of all rows attributed to one physical projection. Block-iterative methods have been studied early on by [15] [22], and have also been proposed recently for algebraic reconstruction from spiral CT data [14]. Ordered subsets for algebraic methods have been studied mostly in the context of parallel computing on distributed systems (clusters) [6] [8], and the overall conclusion has been that the growing communication costs associated with a larger number of subsets limit scalability in the number of processing nodes used. We look at the problem from a different perspective, that is, the speedup behavior in terms of problem size (volume resolution, number of projections, etc.). This is more important when it comes to GPUs since here the number of processing nodes (the pipelines) is fixed. In [21] it was noted that the ray-based update strategy of ART is not very amenable to GPU-acceleration as it does not exhibit much SIMD parallelism, while SART and SIRT perform better in these respects. We extend their study by widening the scope of the problem size and also introduce the intermediate levels of OS-SIRT, leading to interesting and valuable results.

Equation 4.1 described the generalization of algebraic reconstruction into an OS configuration. What is left to define is how the subsets OS_s are composed and how λ is chosen for given number of subsets S. As specified above, OS_s is the set of projections contained in each subset, to be used in a pair of simultaneous forward projections and simultaneous backward projections. In our application, each subset has the same number of projections, that is $|OS_s| = |OS|$, which is typical. Thus, the total number of projections $M = |OS| \cdot S$. The traditional way of filling a certain subset OS_s is to select projections whose indices $m (1 \leq m \leq M)$ satisfy $m \; mod \; S = s$. This is what has been adopted in OS-EM. In contrast, we use a randomized approach to fill the subsets, which we find yields better results than the regular subset population approach. For this, we simply generate a projection index list in random order and sequentially divide this list into S subsets.

The relaxation factor λ to be used for an arbitrary S is chosen by linearly interpolating the optimal λ_{SART} for SART and the typical value of

$\lambda = 1$ used for SIRT (we have found that λ_{SART} = 0.1 works well in practice):

$$\lambda = (\lambda_{SART} - 1)(\frac{S-1}{N-1}) + 1 \qquad 1 \leq S \leq N \tag{4.2}$$

This scheme balances the smoothing effect achieved by the application of a set of simultaneous projections with that obtained by using a lower relaxation factor: the lesser projections in a subset, the lower the λ.

4.2.2 Accelerating the Forward Projection

Let us begin our discussion by writing the projection procedure in form of a typical CPU implementation. Assuming S exclusive subsets and M projections in total, the pseudo-code for projection is shown in Figure 4.2 (the backprojection is interleaved for each subset, but not shown here).

```
(1) For all VS/4 volume slices
   (2) For all S ordered sets OS_s
      (3) For all projections P_st in OS_s
         (4) For all pixel rays r_sti in P_sp
             Initialize ray sum rs_sti
             Set up space traversal for r_sti
             (5) For all slice positions p_i along r_sti
                 Advance r_sti by step size Δr
                 Sum weighted contributions from the (bilinear) neighborhood of voxels v_j
                 Add interpolated values to rs_sti using the trapezoidal integration rule
                 Normalize rs_sti by the sum of weights
```

Figure 4.2: Forward projection loop of a straightforward CPU implementation.

The final loop managing the casting of a ray is sketched in Figure 4.1c. The summing of the weighted contributions is an interpolation operation, and we assume that the sum of weights used for the final normalization has been computed in an initialization phase before reconstruction is begun (that is, the first loop is entered).

From the code in Figure 4.2 we observe that (i) the projection procedure has 5 nested loops (indicated in blue), and (ii) the body of the final level is the longest in terms of operations. The implementation of Diez et al. mapped this loop structure directly to the GPU. Here, the body of loop (4) as well as loop (5) and its body are executed in the fragment shader,

while the head of loop (4) itself is parallelized by generating a raster of fragments, one for each loop instantiation. This is done by creating a polygon of size $T \times 1$ (where T is the number of pixels within a projection) and rasterizing this polygon to the screen. Note that since each volume slice is processed separately, the projection data is just a set of 1D lines drawn from the set of 2D projections. As outlined in Section 1.2, this process generates one fragment per pixel and for each pixel the fragment program is executed.

We observe that executing all instantiations of loop (4) encompasses a single pass, and therefore we have $VS \cdot M$ such passes. Furthermore, GPUs have a limit on the number of instructions in an unrolled loop (all fragment shader loops are unrolled at run time before execution in the shader). This limit is currently 65535 instructions. This becomes problematic in ET where the number of ray steps can become fairly large (assuming a sampling distance of 1.0) due to the large size of the volume slices (up to 2048 voxels along each dimension). For this reason Diez et al. were forced to break each volume slice into TL tiles and compute the rays sum for each tile in a separate pass, adding the results in the end. Thus the final number of passes becomes $VS \cdot M \cdot TL$. Assuming SIRT and $M = 85, VS = 1024$ slices, and $TL = 4$ tiles, this results in $348k$ passes, which causes significant overhead.

For an efficient GPU acceleration, we desire an implementation with as few passes as possible. One main obstacle was the size of the body in loop (5) which required a pass amplification of factor TL. We can reduce this body by pre-computing for reach ray its starting locations (r_x, r_y) at the slice boundary as well as its direction vector (r_{dx}, r_{dy}) and store these four values into a ray texture TX_{ray}. This texture is shown in Figure 4.3, with the four values mapped to RGBA. This texture has M rows, one for each projection line mapping to a specific volume slice. Since the interpolation itself is just one instruction the loop body is now minimal and can easily fit within the limits of an unrolled shader code, even for large volume slices of 2048^2 voxels.

The second measure we have taken is to group all $|OS|$ 1D projections in a subset (corresponding to a certain volume slice) into a single 2D sinogram texture TX_{proj}. Then, during projection, we create a polygon of size $T \times |OS|$, and use TX_{sim} as a rendering target. This generates rays/fragments for all angles and pixels in the currently processed subset, and eliminates loop (3).

Figure 4.3: A sample ray texture storing the initial positions and directions of the rays as RBGA values.

The third and final measure we have taken is to exploit the parallelism of the RGBA channels. Note that for this to work, all fragments in these parallel channels must exhibit the exact same mapping function - all that can be different are the data and the rendering target, with each such simultaneous pair being stored in the RBGA channels. Such a strong parallelism is readily exposed in parallel projection, and we can achieve it by storing and processing a consecutive 4-tuple of volume slices and associated projection data in the RGBA channels of their corresponding textures. This reduces the number of required passes theoretically by a factor of 4, but in practice this factor is about 3.

Thus, all put together, we can reduce the number of passes required for one iteration to $VS/4 \cdot S$. This is shown in the pseudo code in Figure 4.4 below (with more specifics on ray traversal and integration). For example, assuming classic SIRT with $S = 1$ and $VS = 1024$ slices as before, we would have 256 passes. This is less than 0.1% of the implementation of Diez et al., which has a dramatic impact on reconstruction performance.

```
(1) For all VS/4 volume slices
   (2) For all S ordered subsets OS_s
      (3) For all pixel rays in OS_s (loop parallelized into fragments)
            Fetch ray start location rayS(r_x, r_y) and direction rayV(r_z, r_w) from ray texture TX_ray
            Calculate the entry and exit points on the volume bounding box: s_1, s_2
            Set t=0, raySum=0;
            For all ray positions along rayV
                Compute the sampling location s along the ray: s = rayS + t*rayV
                Interpolate sample value and add to ray sum: raySum += Interpolate (volSlice, s)
                Increment t: t+= Δ t
            Store rendering result in TX_sim
```

Figure 4.4: Pseudo code for sinogram-based forward projection. The first two gray lines are executed on the CPU, while the remainder is GPU-resident fragment code.

4.2.3 Accelerating the Back-projection

Equivalent to the projection code, Figure 4.5 below lists the pseudo fragment code for backprojection. Similar to Diez. et al., the final two loops of the above pseudo codes are explicitly controlled and executed on the GPU and are rendered in a single pass. However, in addition, we also exploit the RGBA 4-way parallelism, reducing the total number of required passes to $VS/4 \cdot S$. Note that the major difference of backprojection vs. forward projection is that rasterized fragments are generated on the volume slices (now the rendering target), while the updates are obtained via sampling on the projection textures.

```
(1) For all VS/4 volume slices VS_s
   (2) For all S ordered subsets OS_s
      (3) For all voxels v in VS_s
          (4) For all projections P_st in OS_s
                 Transform voxel position (v_x, v_y) to sinogram position (P_x, P_y);
                 Sample the sinogram texture using coordinates (P_x, P_y);
                 Add the sample value to current voxel value
```

Figure 4.5: Pseudo code for sinogram-based back-projection. The first two gray lines are executed on CPU, while the remainder is GPU-resident fragment code.

4.2.4 Limited Detector Problem and Compensation

During the data collection stage only a small portion of the sample is imaged to obtain the tilt projections. This results in the "limited detector" or "long-object" problem as discussed in Section 4.1, and an illustration is shown in Figure 4.6. Here an off-center acquired projection image contains ray integrals across the whole sample, but the simulated projection at the same angle does not have the complete integral since the reconstruction volume must be limited (typically by a box). In other words, voxels residing in the shadow area of the original complete sample (shown shaded in grey) participate in the projection formation during imaging, but due to the restricted reconstruction area (shown in solid red), they do not contribute in the value formation of any pixels during the reconstruction, resulting in severe vignetting effects if we do not compensate for this (see Figure 4.7a).

We propose a weight correction scheme that effectively resolves this problem for iterative ET - other compensations exist for analytical algorithms [79] based on Filtered backprojection, where an extended area of around double the length of the region of interest (ROI) is used as the reconstruction target to prevent sampling artifacts. While the over-sampling approach resolves the edge problem, it introduces significant extra computation. Our approach does not require these additional computations, as we compensate for the missing target regions on the fly.

In our framework, at a particular tilt angle (see Figure 4.6), the original correction is derived as:

$$Correction = \frac{P - R}{Wsum_r} = \frac{P}{Wsum_r} - \frac{R}{Wsum_r} \tag{4.3}$$

Here we represent the acquired projection as P and the simulated projection as R. The problem of using this equation to derive a correction is that the computed sum of weights W_{sum_r} is calculated based on the bounding box that does not exist (in this closed form) in the acquired data. Therefore, this sum should not be applied towards the acquired projection P. Instead, the acquired sum of weights W_{sum_p} (shown in Figure 4.6b) is the correct value that should be used. Using these arguments, we derive an updated correction equation as follows:

$$Correction = \frac{P}{Wsum_p} - \frac{R}{Wsum_r} = \frac{P \cdot Wsum_r - R \cdot Wsum_p}{Wsum_p \cdot Wsum_r} = \frac{P \cdot \frac{Wsum_r}{Wsum_p} - R}{Wsum_r} \tag{4.4}$$

Consequently, an additional correction factor determined by dividing W_{sum_r} over W_{sum_p} should be computed to pre-weight the acquired projection P before it participates in the regular correction stage. The effect of applying the new correction scheme is shown in Figure 4.7b, where the strong vignetting artifacts present in Figure 4.7a are effectively removed.

4.3 Results

We first analyze the performance of the two interpolation kernels available as hardwired filter functions on the GPU, linear and box (nearest-neighbor), for use in an iterative reconstruction procedure (both SART and

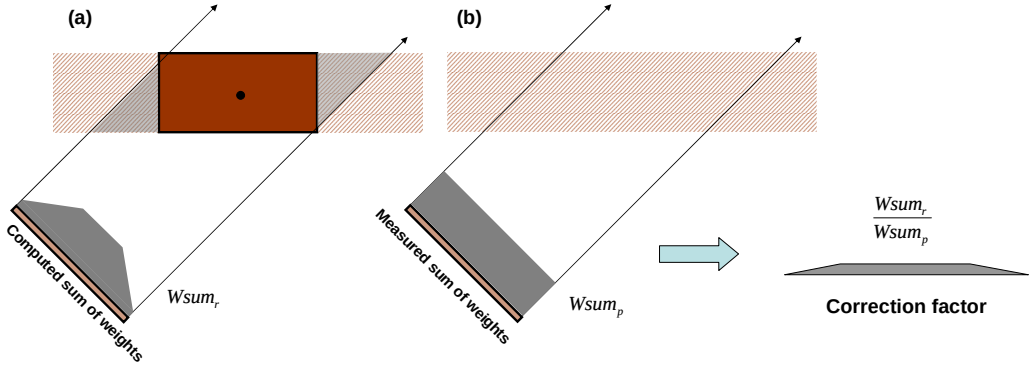


Figure 4.6: Limited detector / long object problem.

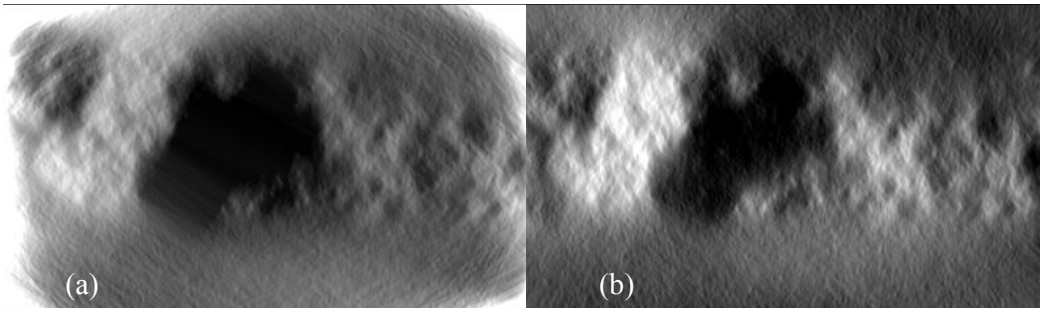


Figure 4.7: Limited detector effect. (a) uncorrected result; (b)corrected result.

SIRT). For this study we employ the 3D Shepp-Logan phantom, which is a popular CT dataset composed of a set of ellipsoids, for which analytical projections can be generated. It has features where the contrast is only 0.5%, making it highly demanding. Nearest neighbor sampling has the advantage that it only requires one data fetch and one weighting operation per sample (both in forward and backward projection), while linear interpolation requires four. It should be noted that especially in iterative reconstruction it is crucial that high-quality projections are generated since these projections form the basis for the corrections used in the grid update in the backprojection step. As Figure 4.8 demonstrates, only linear interpolation can fulfill these demands - the reconstruction obtained with nearest neighbor interpolation is of much lower quality with a significant amount

of low and high-frequency noise. As it turns out, linear interpolation on the latest GPU boards is nearly as efficient as nearest neighbor, thus the performance hit is minimal.

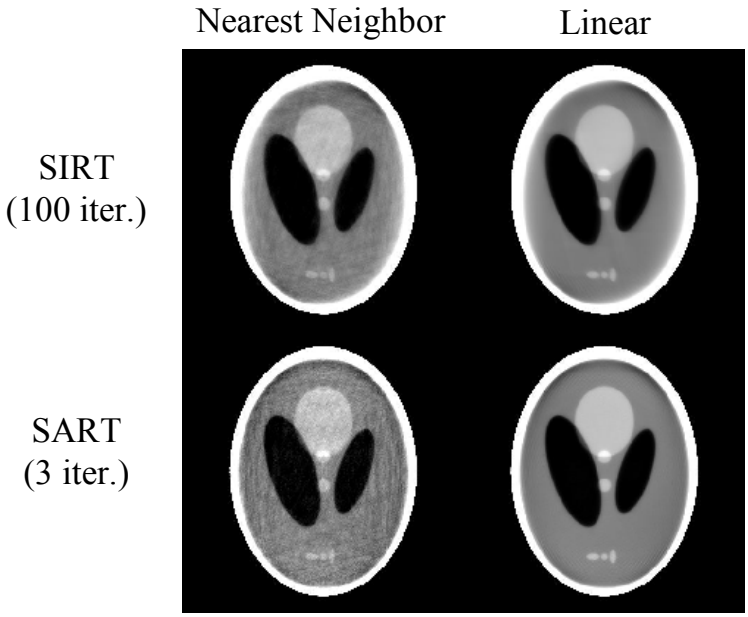


Figure 4.8: A slice from the 3D Shepp-Logan phantom reconstructed in 3D via SIRT and SART using different interpolation kernels during the projection/backprojection. All images are shown with the full intensity interval [0, 2.0] windowed to [1.01, 1.04] in order to illustrate the ability to reconstruct regions of low contrast (such as the three small tumors on the bottom).

4.3.1 OS-SIRT and Effect on Performance

For the following experiments we used the popular 2D Barbara test image also employed by [21] to evaluate the performance of the different reconstruction schemes described above. The target 2D image is created by cropping the original image to an area of 256×256 pixels resolution, and 180 projections at uniform angular spacing of $[0°, 180°]$ are obtained.

First refer to Figure 4.9a where we show the images obtained from various SIRT/SART reconstructions, using subsets of different sizes. Here, we also list reconstruction timings and cross-correlation (CC) values after 100 iterations. The CC values are measured by comparing the reconstructed image with the original input image, which is a common criterion to gauge the status of the ongoing reconstruction. We observe that both the time required to finish the computation and the CC values increase with growing numbers of subsets. Given equal iteration numbers, results from SART achieve the best score, but they need the longest time to be obtained. This originates in the fact that SART requires more rendering passes on the GPU since the projection operations (and the backprojection operations) per iterations cannot be combined into a single pass. The larger time consumed for SART reconstructions on the GPU (in contrast to SIRT) was also observed by [21], but the effect of various subset sizes (giving rise to OS-SIRT) was not studied there.

Next, we study the noise behavior for the different OS-SIRT configurations (in the following we shall refer to SART as an OS-SIRT reconstruction with M single-projection subsets). These behaviors are best revealed in homogeneous image regions, such as the area to the far left of Barbara's head on the height of her eye. Figure 4.11a(left) shows a series of tiles of this patch arranged in the same order as the images in Figure 4.9a (the first tile shows the original patch). Figure 4.11a (right) gives the Coefficient of Variation (CV) of these tiles. We observe that the smaller the subset, the noisier the reconstructions and hence the larger the CV. However, with more noise also comes better preservation of detail, as can be observed in Figure 4.9a. Finally, the tendency of SART to produce reconstructions noisier than the original and that of SIRT to produce reconstructions smoother than the original is also demonstrated in Figure 4.16 , where we show the renditions of a line profile across another area of the Barbara image (only for the original image and reconstructions with SART and SIRT).

In practical applications, researchers would like to optimize the amount of (wall clock) time required to achieve a reconstruction of given quality (gauged by the CC value of the simulated and acquired projections). We have seen that, for a fixed number of iterations, SART (OS-SIRT with M subsets) runs the slowest on the GPU, but produces the best reconstructions. In fact, both reconstruction time and quality increase with the number of subsets Therefore, it appears worthwhile to study if there is in fact an optimum in terms of the number of subsets. Such an optimum subset

size could then be used to generate the best reconstruction in the smallest amount of time. The following results produce insight with regards to this goal.

Figure 4.9b shows the reconstruction results for a fixed CC value, which means that all reconstructed images are nearly identical to each other. We observe that the smaller the number of subsets, the larger the number of iterations that are required to reach the set convergence threshold. However, due to the overhead involved in the GPU-based framework, different wall-clock times are produced. Here, using a number of 10 subsets achieves the best timing performance compared to the other subset configurations. The CV metric is also favorable in this configuration (see Figure 4.11b, comments as before). We should note that this finding is specific to GPU-based reconstructions, since here the number of passes is particularly important. A CPU-based reconstruction, on the other hand, would be much less sensitive to this relationship since the dominating cost here is dominated by the elementary operations of projection/backprojection, which are massively accelerated on GPUs.

Researchers might also be interested in knowing which configuration works best given the same amount of computation time. Figure 4.9c presents the results of such a study, where the reconstruction stops after reaching a wall clock time threshold. Again, using a number of 10 subsets achieves the best CC value. Figure 4.11c indicates that the CV metric for the background noise is also favorable for this configuration.

A similar study has also been performed from a number of 140 projections covering only 140 projection angles. This setup is more similar to the real ET configurations and reconstructed images are presented in Figure 4.10. As we can see from the result, for this specific configuration, subset size of 5 will achieve the best wall-clock time performance.

We have also measured the true CC values with respect to both number of iterations and the actual wall clock time. On Figure 4.12 we observe that the smaller the number of subsets, the slower the speed of convergence in terns of CC value. At the upper limit is SART which converges fastest. While this relationship has been known, or at least suspected, before, Figure 4.13 is more novel. It reveals that there is a certain subset number for which the lowest RMS value can be maintained, consistently at all times. In the current experiment (the Barbara image reconstructed from 180 evenly distributed projections), this number is 10. However, this

optimal number may vary in different reconstruction scenarios and applications.

Our final experiment deals with the composition of the subsets themselves Figure 4.17 shows the performance of OS-SIRT with three different numbers of subsets (here, 10, 20, and M - SART) using a regular interleaved projection selection (as it has been suggested for OS-EM) and our proposed random projection selection scheme. The results show that the random approach always performs better than the regular method, and the difference margin increases when more subsets are used, with the largest difference obtained with SART.

4.3.2 Performance of Sinogram-centric GPU-accelerated ET

We benchmarked the performance of our framework in the context of realistic ET dataset sizes. Table 4.1 compares the runtimes obtained with our framework with those reported in [21]. The timings presented here refer to a 3D reconstruction with SIRT, using projection data from 88 tilt angles, and are standardized to list the time needed for reconstructing a single slice. We have run our framework on GPU hardware comparable to the one employed by [21], that is, the NVIDIA G70 chip (this chip forms the core of both the Quadro 4500 and the GeForce 7800 GTX boards, with only minor performance differences). Since then, a new generation of NVIDIA chips has emerged, the G80 (available as the GeForce 8800 GTX board), and we also report the timings for this new hardware. We observe that the significant decrease in passes of our GPU-algorithm leads to consistent speedups of more than an order of magnitude across all resolutions and iteration numbers. The new G80 then yields another speedup of about a factor of 2.5.

Thus, using our GPU-accelerated ET framework, one can now also reconstruct a 3D full-size volume (with a slice resolution of 1024^2) on the order of minutes (20 minutes), while [21] report 5 hours for this task (both with SIRT). A CPU-based reconstruction would take on the order of days. These performance increases readily translate to all of the presented variations of OS-SIRT (including SART).

Finally, Figure 4.14 and 4.15 present reconstructions of two real ET datasets (i) cryo data from frozen hydrated tobacco mosaic virus and (ii) chromatin, using 61 and 70 projections, respectively, obtained at uniform

spacing over a tilt angle of around 130°. The reconstructed volume was of size $680 \times 800 \times 100$ and $512 \times 512 \times 200$. Three different OS-SIRT configurations are shown, and all were stopped at the same wall clock time of 315 and 141 seconds. All reconstructions also used the limited detector compensation technique discussed in Section 4.2.4. We notice that OS-SIRT with 5 subsets yields the best quality and contrast for both datasets.

(a)

Subset number	1 (SIRT)	10	20	60	180 (SART)
Time (second)	6.77	11	35.25	66	171.3
CC values	0.952366	0.96417	0.969669	0.974911	0.976126

(b)

Subset number	SIRT	OS-SIRT 5	OS-SIRT 20	OS-SIRT 70	180 (SART)
Iteration	75	56	30	15	12
Time (second)	2.46	2.43	8.96	4.53	15.84

(c)

Subset number	SIRT	OS-SIRT 10	OS-SIRT 20	OS-SIRT 60	180 (SART)
Iteration	76	47	15	8	3
CC values	0.945931	0.949028	0.92737	0.931062	0.92663

Figure 4.9: Barbara image, reconstructed via various ordered subset methods. The image resolution is 256×256 all reconstructions were run from 180 projection angles. Linear interpolation was used in all reconstructions. (a) Reconstructions obtained for various numbers of subsets after 100 iterations (b) Reconstructions obtained for various numbers of subsets for a fixed CC = 0.949. We observe that the best result is achieved for 10 subsets. (c) Reconstructions obtained for various numbers of subsets for a fixed time = 5.13s. We observe that again the best result is achieved for 10 subsets.

(a)

Subset number	1 (SIRT)	5	20	70	140 (SART)
Time	3.28s	4.34s	29.874s	43.997	132s
CC values	0.937362	0.944249	0.95459	0.959288	0.959971

(b)

Original	SIRT	OS-SIRT 5	OS-SIRT 20	OS-SIRT 70	SART
iteration	75	56	30	15	12
time	2.46	2.43	8.96	4.53	15.84

(c)

Original	SIRT	OS-SIRT 5	OS-SIRT 20	OS-SIRT 70	SART
iteration	90	68	10	10	2
CC values	0.93472	0.93518	0.884385	0.911823	0.803966

Figure 4.10: Barbara image, reconstructed via various ordered subset methods. The image resolution is 256×256 all reconstructions were run from 140 projection angles. Linear interpolation was used in all reconstructions. (a) Reconstructions obtained for various numbers of subsets after 100 iterations (b) Reconstructions obtained for various numbers of subsets for a fixed CC = 0.93. We observe that the best result is achieved for 10 subsets. (c) Reconstructions obtained for various numbers of subsets for a fixed time = 2.95s. We observe that again the best result is achieved for 10 subsets.

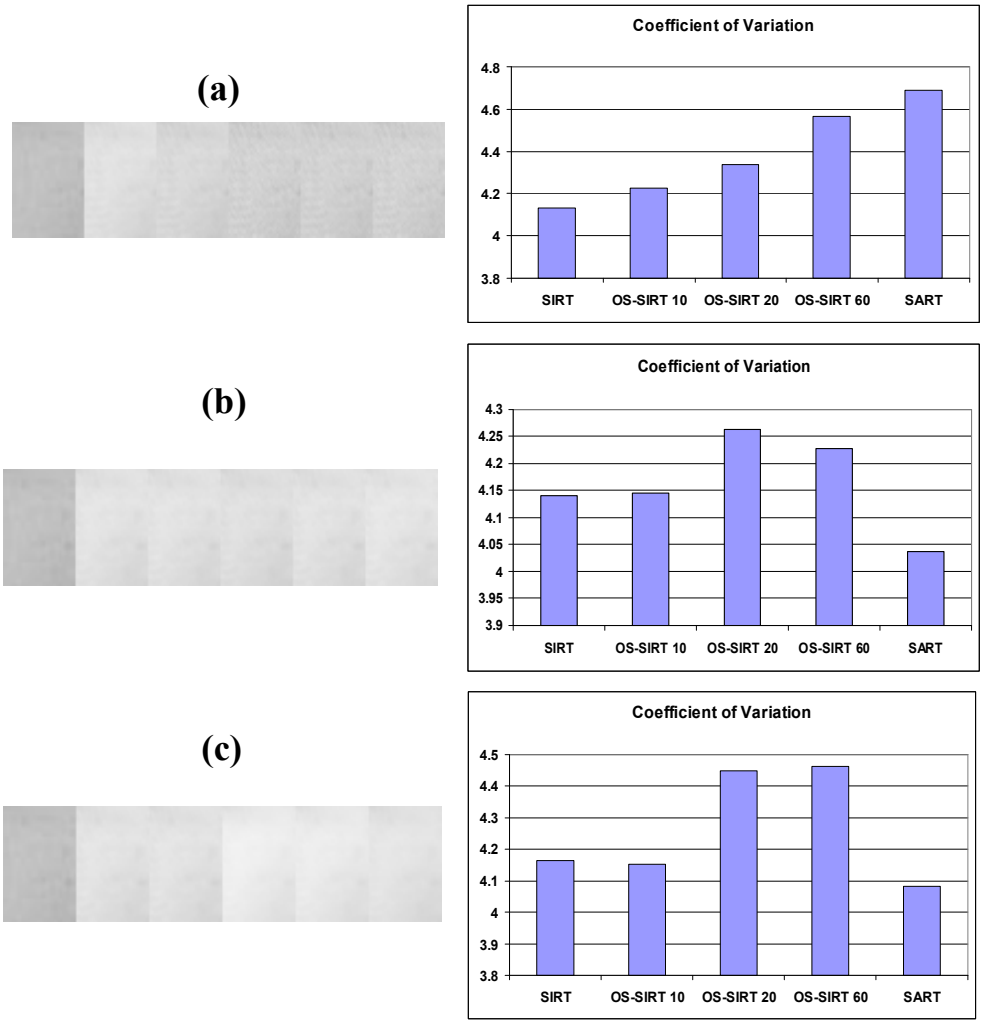


Figure 4.11: Noise study for the Barbara images shown in Figure 4.9: (a)-(c) correspond to (a)-(c) in Figure 4.9. (left column) cropped background patches illustrate the different noise levels attained by the corresponding reconstruction settings, (right column) calculated CV (Coefficient of Variation) values of these cropped background patches.

Iteration	Resolution	G70			G80
		Quadro 4500	7800 GTX	Speedup	8800 GTX
10	256 x 256	N/A	0.2	N/A	0.08
50	256 x 256	N/A	1	N/A	0.38
10	512 x 512	9	0.6	15.0	0.25
50	512 x 512	39	2.6	15.0	1.1
10	1024 x 1024	32	2.1	15.2	0.8
50	1024 x 1024	146	10.7	13.7	4.1
10	2048 x 2048	123	8.5	14.5	3.1
50	2048 x 2048	567	41.7	13.6	16.7

Table 4.1: Timings for the reconstruction of a volume slice at different resolutions using SIRT and parallel projections acquired at 88 tilt angles.

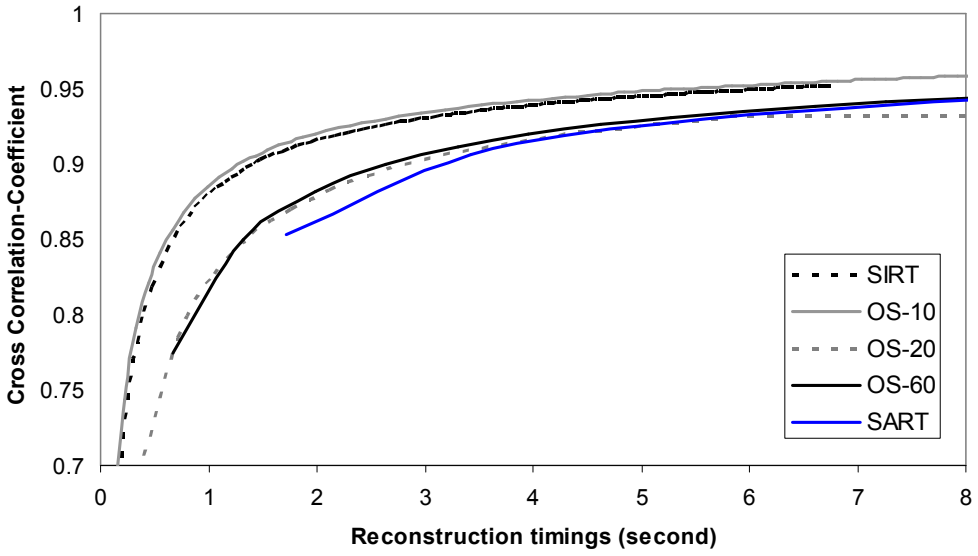


Figure 4.12: CC values vs. number of iterations for reconstructed Barbara images shown in Figure 4.9.

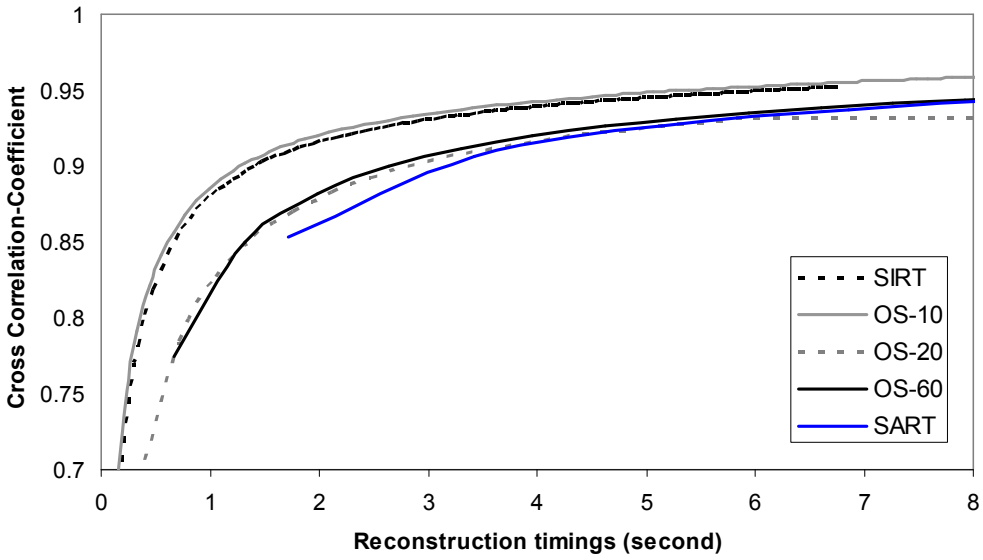


Figure 4.13: CC values vs. running time for reconstructed Barbara images shown in Figure 4.9.

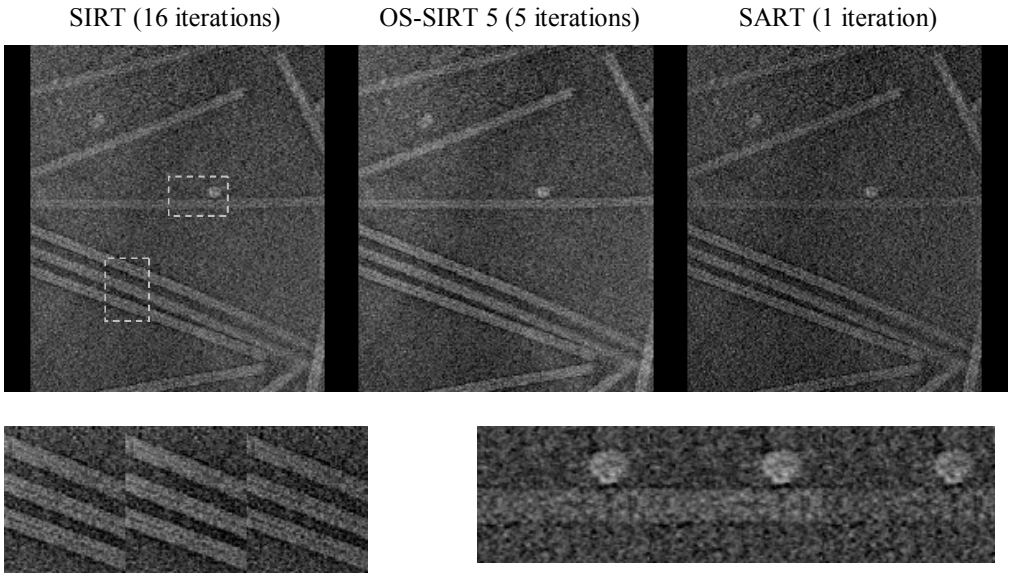


Figure 4.14: Upper row: reconstructed tobacco mosaic virus data; lower row: cropped patches from regions (outlined by boxes) in the upper row. The resolutions: volume $680 \times 800 \times 100$, projections 680×800 pixels, and 61 tilt angles were used. In all three cases the reconstruction was terminated at 315 seconds. We observe that OS-SIRT 5 achieves the best detail resolution within this given time.

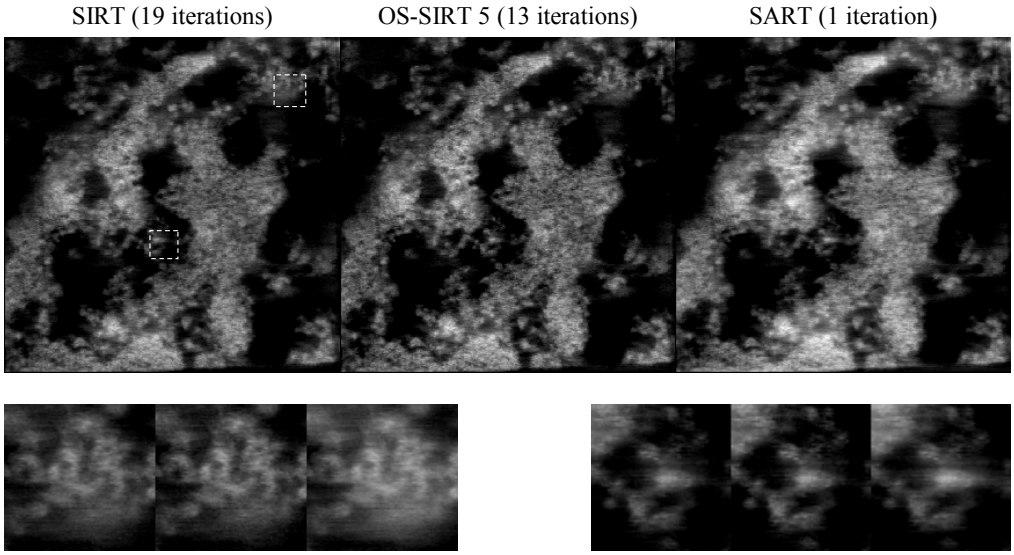


Figure 4.15: Upper row: reconstructed chromatin data; lower row: cropped patches from regions (outlined by boxes) in the upper row. The resolutions: volume $512 \times 512 \times 200$, projections 512×512 pixels, and 70 tilt angles were used. In all three cases the reconstruction was terminated at 141 seconds. We observe again that OS-SIRT 5 achieves the best detail resolution within this given time.

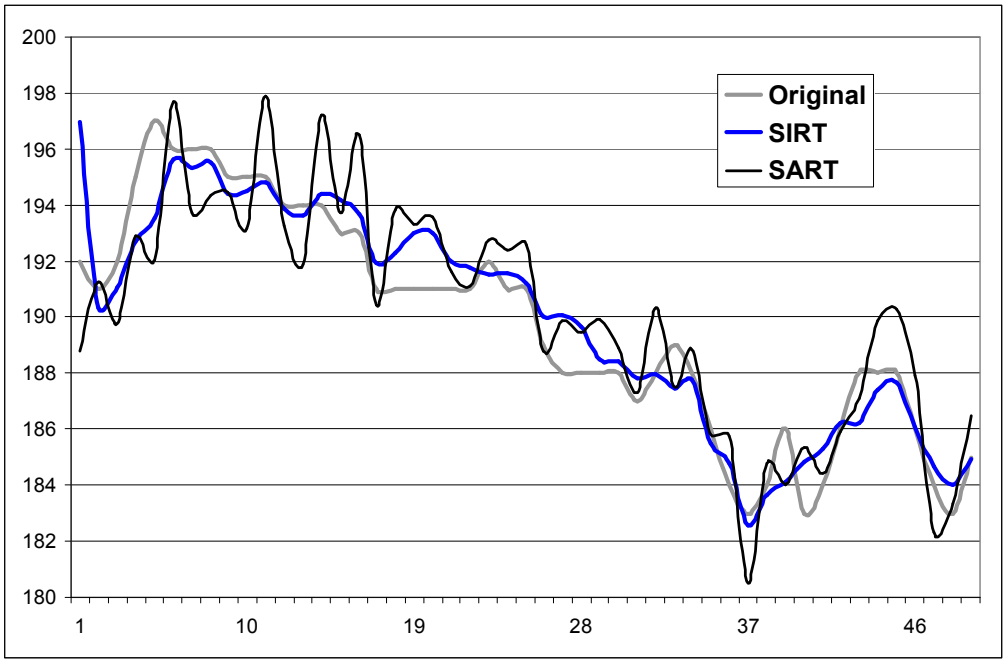


Figure 4.16: Line profiles across the reconstructed Barbara images shown in Figure 4.9.

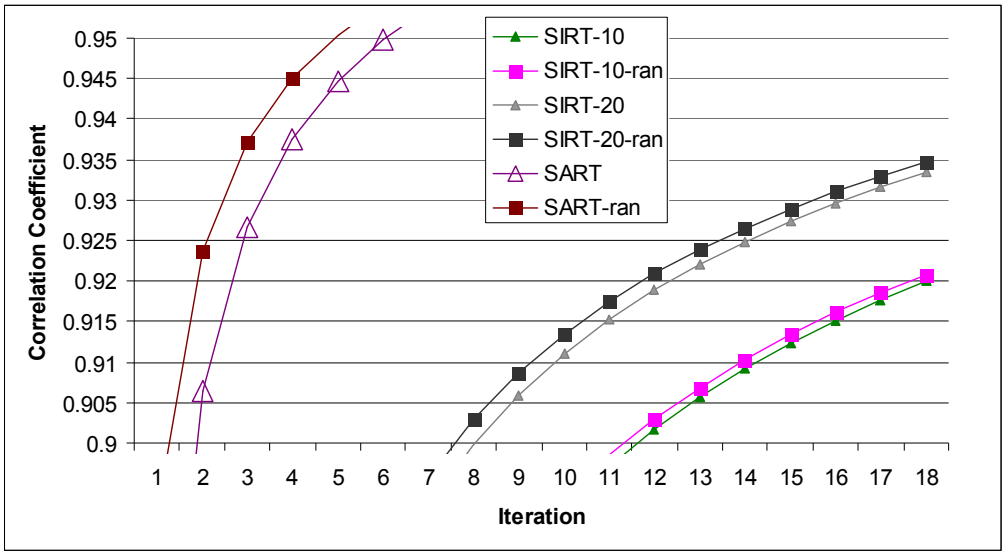


Figure 4.17: CC values for regular OS-SIRT and randomized OS-SIRT.

Chapter 5

Projection-based Volume Rendering

5.1 Introduction

Volumetric datasets can have many origins. They can be the output of numerical simulations, such as CFD, finite elements, finite difference, and other calculations of this nature. They may also be generated by inverse Fourier transform, which is the case for MRI imaging. And they can be the product of the voxelization of analytical functions or polygonal objects. Finally, they can result from tomographic reconstruction (CT), which is a process invoked whenever an object is scanned with a transmissive radiation, such as X-ray, ultrasound, or infrared light. In CT, an object is irradiated with a transmissive source on one side and a projection is acquired on the other side. This process is repeated at a sufficient range of viewing angles, and the resulting projection set is processed in the CT reconstruction procedure. Major applications are in medical imaging and also in industrial CT and security. The well-known engine dataset, for example, was obtained via industrial CT. Medical imaging with CT is

ubiquitous. It is a relatively inexpensive scanning technology, when compared to MRI, and has many diagnostic applications in medicine. Almost all parts of the human body can be imaged and diagnosed with CT imaging. This technique is dedicated to the large body of datasets obtained with CT reconstruction methods.

There are a number of different scanning geometries: parallel-beam, fan-beam, cone-beam, and spiral (helical) CT. Reconstruction algorithms for the former two geometries are referred to as slice-based CT, while the latter are referred to as volumetric CT. In fact, spiral CT has become a multi-slice acquisition method, that is, a stacked (multislice) array of detector arrays rotates about the patient, with a cone-beam source irradiating the patient. Nowadays, spiral CT scanners with up to 64-slices are employed in clinical practice. For fan and spiral beam geometries, there exists the ability to rebin the projection data into parallel-beam data. If a sufficient amount of projection (ray) data have been collected, then one may sort these rays into equivalent bins of parallel beam rays, which can then be used in conjunction with conventional parallel-beam CT reconstruction methods. However, there are also reconstruction methods for fan-beam, cone-beam, and spiral CT which do not use rebinning. This is the domain of the exact reconstruction methods, as described in [44] [46] [54] [90] [93] [36] and others. Finally, there are also approximate methods that work well in practice and are very popular, under certain conditions. For example, Feldkamp algorithm [26] is often used in cone-beam reconstruction and produces good results for sufficiently small coneangles $< 20^{\circ}$.

The original projection-based volume rendering framework (called as D^2VR) [76] assumes that the CT reconstruction resulting in the volume dataset is (or better, would have been) obtained with parallel beam data and algorithms. However, direct fan and cone-beam reconstruction geometries would also be feasible with D^2VR. Finally, the more complicated reconstruction techniques used in advanced spiral-CT algorithms could be considered as well, but these would produce a larger overhead.

CT reconstruction is a data conversion process, and due to associated sampling with imperfect filters, it is a lossy data conversion process. CT reconstruction is needed to convert the data into the format used by traditional volume renderers. D^2VR, on the other hand, is a non-traditional volume renderer, which does not require the data conversion and instead produces volume renderings directly from the raw projection data, eliminating the errors incurred in the conversion process.

D²VR traverses the volume space as usual, but instead of performing the sampling there, it maps the sample positions into each projection image and interpolates the data in those. A sample value is then composed by adding all contributions so obtained. By finding the derivatives in each projection image, one can reconstruct the sample gradient in a similar fashion. Once the sample and gradient values have been obtained, the usual transfer function lookup, shading, and compositing can take place. In essence, the 3D array of sample values that ensue from this process are those that would be generated with a parallel-beam CT reconstruction, had the volume grid been placed at this orientation. Therefore, one may say that the result using an axis-aligned volume rendering with traditional techniques and D²VR are identical (assuming the gradients were calculated in volume space in both cases)

One should note, however, that the CT-based data conversion process does not only produce data in a format that is more convenient to render, it also reduces the overall data complexity. Assuming fan-beam geometry and rendering at no magnification, alias-free reconstruction requires $\pi/2 \cdot N$ projections to reconstruct a slice with N^2 voxels [44]. Thus, with D²VR, each sample requires $\pi/2 \cdot N$ bilinear interpolations, which is substantially more effort than the K^3 neighbors needed for an interpolation in volume space (with K being the 1D extent of the interpolation filter). Even with acceleration techniques, such as empty space skipping and early ray-termination, which reduce the number of samples to be computed (at complexity $\pi/2 \cdot N$), GPU-assistance in this task still seems necessary to overcome this great computational burden.

5.2 Related Work

The combination of volume rendering and CT on graphics hardware is not new. Already in 1995, Cabral et al. [13] utilized the inverse relationship of these two procedures to derive a common mathematical theory for both plus a common framework that would accelerate them on SGI texture mapping hardware. The capabilities of this hardware were quite limited, and in this early work the hardware was mainly used to accelerate the rasterization effort. Much of the work, such as accumulation and filtering in CT, had to be performed on the CPU. There was also no

direct connection of CT and volume rendering, such as the one that was formulated for D^2VR. The same hardware was later used by Mueller and Yagel [70] to accelerate iterative CT algorithms, such as SART (Simultaneous Algebraic Reconstruction Technique). This approach showed how two color channels with 8-bit precision could be combined to reach higher precision for integer-arithmetic. With the evolution of GPUs more sophisticated CT implementations were possible. Xu and Mueller [98] described a general framework for GPU-accelerated CT, spanning iterative and analytical algorithms for transmission (CT) as well as emission tomography (PET, SPECT), fully accelerated on the GPU. They achieved speedups of 1-2 orders of magnitude, compared with CPU implementations of the same accuracy. Chidlow and Möller [16] described a GPU accelerated implementation for SPECT imaging, but the accumulation stage was exported to the CPU.

The GPU has been the source of many acceleration efforts for volume rendering as well. As mentioned before, the common ancestor for both domains is Cabral et al. [13], but the work that followed on the volume rendering track was much more prolific. While the implementations using the SGI rendering hardware were constrained by the limited set of operations, the revolution of the more recent PC-based graphics hardware took away most of these restrictions. In the following, we shall just name a few of the most prominent advancements, for regular grids. First, there is the work by Rezk-Salama et al. [77], which used multi-texturing to enable fully-hardware based volume rendering, and there is the work by Engel et al. [23], which introduced pre-integrated volume rendering to eliminate the stair-stepping artifacts caused by the common slice-based rendering paradigm. A more recent work is that by Krüger and Westermann [52] who describe a ray-casting implementation, fully GPU-accelerated. Their implementation also includes mechanisms for early ray-termination and empty-space skipping, the latter by using a low-resolution occupancy octree. Neophytou and Mueller [73] showed how the z-buffers early fragment-kill capabilities can be exploited to skip over empty space and voxels that would project into already opaque image regions. The latest development is the system proposed by Stegmeier et al. [85], which completely eliminates the use of slice rasterization and runs the entire ray advancement in a single fragment shader loop. This bears some advantages for the implementation of non-linear ray effects, such as refractions.

The approach presented here builds on our GPU-accelerated CT framework for filtered backprojection [69], which is detailed in Chapter 3 but generalizes it substantially. First, it allows the reconstruction of arbitrary oriented volumes, which is not needed for CT, but is necessary for D^{2}VR since we must generate a matrix of samples exactly aligned with the image plane, whose orientation is completely arbitrary. Second, we incorporate various acceleration techniques to limit the reconstruction effort to only the visible and non-occluded (the relevant) matrix samples. Our system enables framerates of 2 frames/s and more for realistic dataset sizes.

5.3 GPU Implementation

The GPU implementation is based on the GPU-accelerated filtered backprojection framework, which includes both Streaming-CT and Slice-CT methods. Various strategies of selecting appropriate data precision and optimizing the pipeline discussed in Chapter 3 can be also applied here. In addition to the basic GPU-based D^{2}VR framework, we also describe techniques that can be used to accelerate it further in the following sections.

5.3.1 Basic Framework

Let us assume an arrangement in which all projections are distributed around a circular orbit. In order to achieve a maximal volume resolution N^3, which we would like to reconstruct without aliasing, we need to have $M = \pi/2 \cdot N$ projections distributed around a half circle, assuming parallel beam data. It is our goal to be able to reconstruct image-aligned volume slices in front-to-back order, since this will enable us to perform occlusion culling. We first create the volume proxy polygons, rotate and place them in an orientation that is orthogonal to the viewing vector. By doing so, the reconstructed voxels on slices defined by these polygons will be aligned accurately with the viewing ray samples, as D^{2}VR requires. Then the slice stack is passed into the Streaming-CT/Slice-CT pipeline. The view-aligned slices will be accumulated/reconstructed in a front-to-back

fashion, and conventional volume rendering procedures such as classification, shading and composition are performed slice by slice (see Figure 5.1).

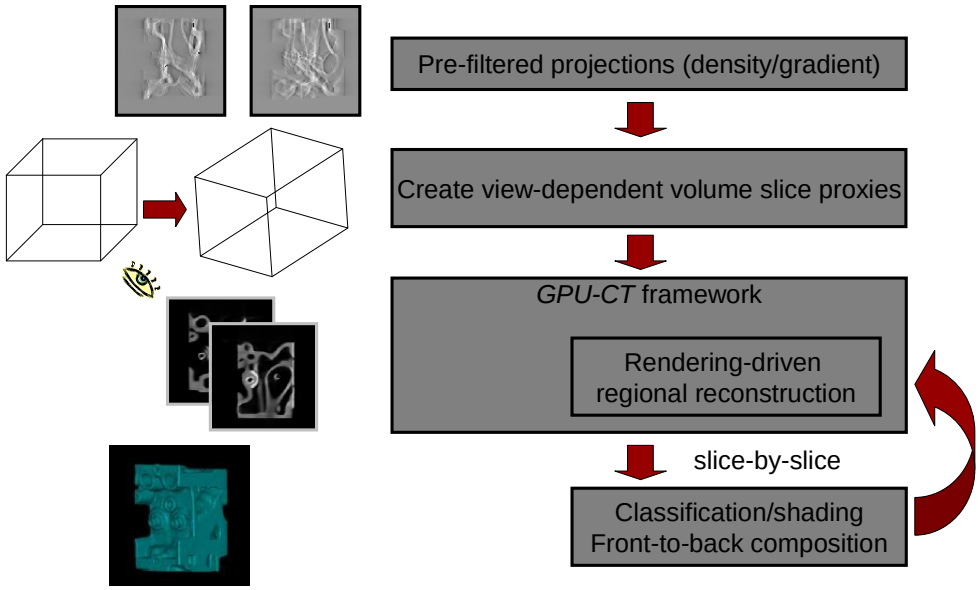


Figure 5.1: GPU-based D^{2}VR pipeline

5.3.2 Gradient Estimation

In D^{2}VR, we have two choices. We can either reconstruct only the densities and then compute the gradients in volume space directly by central-differencing adjacent samples in all three orthogonal directions, or we can estimate the gradients in projection space and reconstruct the gradients as well. Note that when computed in projection space, the two gradient components more parallel to the viewing direction should be scaled with $\cos\theta$ and $\sin\theta$, where θ is the projection angle with respect to the image. We call this approach *gradient-from-projections* (*GFP*). The projection space gradient is stored into the RGB color channels with the alpha channel carrying the filtered density values and both properties are reconstructed at the same time. The other approach, called *gradient-from-samples* (*GFS*), reconstructs only the density values and performs the gradient estimation

from these samples in volume space. This reduces the memory bandwidth in the accumulator. It benefits the reconstruction stage but slows down the rendering procedure, but since memory bandwidth is usually the GPU-bottleneck, this tradeoff is advantageous. The computation of gradients in this approach will require 6 neighborhood samples when the central-difference operator is used, and each such sample should be a direct neighbor.

The *GFS* also tends to produce images of higher perceived definition (or visual sharpness and acuity) than the *GFP*. As a justification, consider the following. Let us first compare the D^{2}VR with regular DVR in terms of their filter pipelines:

$$\textbf{CT+DVR} \rightarrow R\{p \otimes h_p\}^N \otimes h_v \tag{5.1}$$

$$\textbf{D}^2\textbf{VR} \rightarrow R\{p \otimes h_p\}^N \tag{5.2}$$

In this equation, p are the projection data, h_p and h_v are the interpolation filters in projection and volume space, respectively, and R is the reconstruction operator with N being the number of projections. This is a formal way to show that D^{2}VR's filter pipeline only involves one interpolation filter, and thus produces lesser artifacts. Most gradient filters, in particular the central difference filter, fall off towards the highest frequencies and mostly accentuate the midrange frequencies (see e.g. [7]). This is a good feature in some respect since it reduces the effect of noise, which typically resides in the higher bands. But on the other hand, the sharpness of the gradients also suffers, since the desirable (signal) portions of the higher bands are now missing. The difference of *GFP* and *GFS* are most prominent when upsampling during the D^{2}VR, that is, when the viewport has a higher resolution than the reconstructed volume slices. We shall explain this now. Consider the following pair of filter pipelines for gradient computation, assuming D^{2}VR is used for both:

$$GFP \rightarrow R\{(p \otimes g_p) \otimes h_p\}^N \tag{5.3}$$

$$GFS \rightarrow R\{p \otimes h_p\}^N \otimes g_v \tag{5.4}$$

We observe that with the GFP method the gradient filter g_p is applied first, in projection space (followed by filtering with the ramp-filter), and then, within the reconstruction R, the upsampling of the above-mentioned band-passed frequency spectrum is performed, using interpolation filter h_p. On the other hand, for *GFS* the unlimited (ramp-filtered) frequency spectrum is interpolated first, using h_p in reconstruction R, which then undergoes the band-passing of the gradient filter, in volume space. Here, the upsampling plays an important role. It stretches the most-active (mid-range) frequency window of the gradient filter into the higher frequency bands of the reconstructed slices (that is, it better approximates the ideal ramp filter), and thus accentuates these higher frequencies more than *GFP*. This leads to stronger gradients, and therefore sharper object features, but also possibly to enhanced noise.

For D^{2}VR there is a special advantage that comes with the texture-spreading method. If projections are spread onto the slice stack orthogonal to the view point, we can immediately start shading and compositing the reconstructed slice as soon as the accumulation process from all projections is finished. This assumes that the gradients are reconstructed. Just like in the projective texture method, when only the density values are reconstructed, we need to wait until the slice behind it is generated, in order to compute the gradients using the central difference operator.

5.3.3 Viewport vs. Volume Resolution

When determining the resolution of the volume slices to be reconstructed from the projection data to obtain an image at a certain viewport resolution, one should realize that CT projection data of a certain resolution, say N^2, will not be able to yield a volume of higher resolution than N^3 (assuming there are a sufficient number of them, theoretically $\pi/2 \cdot N$, as mentioned before). This is due to the frequency spectrum as derived from the *Fourier Projection Slice Theorem*. Therefore, reconstructing density volume slices at the resolution of the projection data, with subsequent upsampling of these to the viewport resolution will produce similar results, provided a decent interpolation filter is used to sufficiently suppress aliasing. An obvious consequence is faster rendering speed, since in-slice density upsampling is less expensive than density reconstruction on a finer grid. Classification and shading is performed on the high-resolution grid

in both such pipelines.

5.3.4 Acceleration Methods

According to the theory of computed tomography, reconstructed voxel values are valid only for those voxels that fall into the "effective" reconstruction area, which is a (truncated) circle for a 2D (rectangle) square volume slice, or a (truncated) cylinder for a 3D cubic (rectangular solid) volume, assuming parallel beam reconstruction. Hence we can calculate the initial bounding volume and slice it according to the viewing direction. We then use these bounding volume slices as our depth buffer to guide the reconstruction, which essentially restricts the computation within the effective area throughout the whole reconstruction pipeline. A more sophisticated scheme would, as a preprocess, reconstruct a volume mask that would label all voxels that are in the shadow of all non-zero projection data. This occupancy mask could then be sliced, for each visualization frame, with the present slice configuration to drive the reconstruction more accurately than the bounding sphere. This is equivalent to empty space skipping, with the former method being a good approximation for many cases.

We shall now turn to early ray termination acceleration, a.k.a. occlusion culling (the mechanism for GPU-D^{2}VR is illustrated in Fig. 5.2). For this, we must reconstruct the slice stack in front-to-back order. When a new slice is computed from all projections, we composite it with the current frame buffer. We then examine every fragment's opacity value and compare it to the preset threshold, which usually varies between 0.0 to 1.0, depending on the rendering mode (isosurface or full volume rendering). All those fragments whose opacity values exceed the threshold will be recorded to update the depth buffer. The updated depth buffer will then be used for the reconstruction of the next slice to prevent the GPU from generating fragments at those marked positions (which is the early z-buffer kill mechanism). The technique effectively eliminates the need for reconstructing voxels that do not contribute to rendering, hence it greatly reduces the effort consumed on the computational intensive component. For iso-surface rendering, where emitted rays are generally terminated early, this rendering-driven technique can achieve good speed-ups. Implementation-wise, the effect of early z-culling largely relies on how

long the rendering of a fragment would take. Low computational effort ratio with respect to a single fragment could offset the advantage brought by early z-cull mechanism. Therefore, the spreading method that incorporates a longer fragment shader tends to benefit more from the above acceleration strategies, compared to the projective texture method, where texture mapping imposes relatively light rendering efforts on individual fragments.

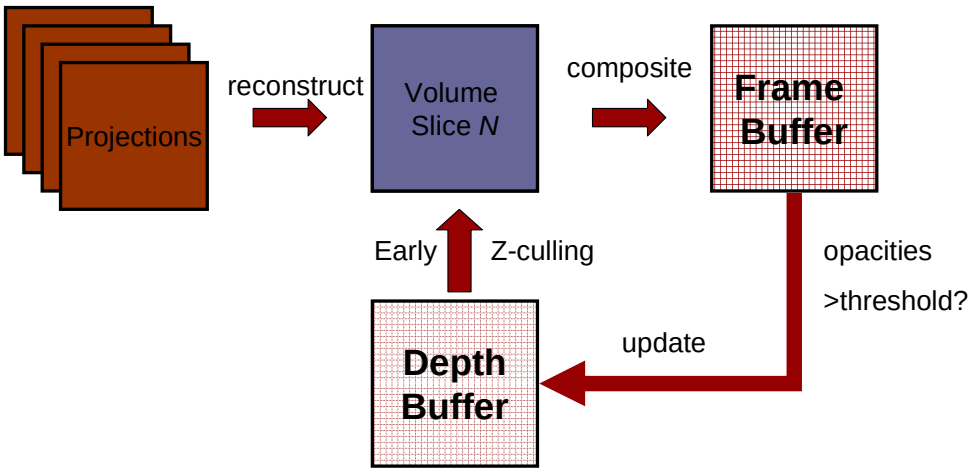


Figure 5.2: Rendering-driven occlusion culling.

5.4 Results

We experimented with the GPU-D^{2}VR framework on an AMD Athlon 2.2GHz dual core PC with 1GB RAM and a GeForce 6800 GT. Shaded images were rendered into either a 128^2 or a 256^2 viewport.

Timings for the different strategies described, using various CT datasets, are presented in Table 5.4. We observe a speedup of a factor 1.5 for density-only D^{2}VR over density+gradient D^{2}VR. The 256^3 size of volume could not be reconstructed with both density and gradients since it exceeds the current maximum size of the GPU memory. The acceleration techniques

Dataset	Proj.	Vol.	Viewport	D^2VR	D^2VR-G	D^2VR+B	D^2VR+B+OC	Ras. Voxels(%)	in Fig. 5.3
Foot (iso1)	128×128^2	128^3	128^2	0.62	0.85	0.54	0.41 (2.4 fps)	37%	N/A
Foot (iso1)	128×128^2	128^3	256^2	0.8	0.95	0.72	0.59 (1.7 fps)	37%	(e)(f)
Foot (iso2)	128×128^2	128^3	256^2	0.8	1.0	0.72	0.66 (1.5 fps)	36.1%	(k)
Foot (iso2)	128×128^2	256^3	256^2	2.3	N/A	2.0	1.67 (0.6 fps)	36.0%	(j)
CT Head	128×128^2	128^3	256^2	0.8	1.1	0.94	0.7 (1.4 fps)	24.2%	(c)(d)
Toes	256×256^2	256^3	256^2	4.93	N/A	4.26	3.5 (0.3 fps)	31.6%	(l)

Table 5.1: Rendering performance in seconds for various datasets under different strategies: **D^2VR** is projection-based volume rendering with reconstruction of density only (*GFS*); **D^2VR-G** is the projection-based volume rendering with reconstruction of both density and gradient properties (*GFP*); **B** uses the bounding volume empty-space culling strategy, and **OC** uses occlusion culling. All of the above timings are measured with shading.

mentioned in Section 5.3.4 also yielded a factor of 1.5 speedup, as compared to their respective basic GPU D^{2}VR counterparts.

A near interactive performance (2.5 fps) is obtained for volumes of size 128^3 when rendered into a 128^2 viewport. Rendering into a larger viewport of 256^2 with in-slice interpolation of 128^3 reconstructions (as described above) only decreases performance by a small amount (1.7 fps). On the other hand, rendering from a 256^3 reconstruction volume takes 4 times as much (0.6 fps).

The timings reported in [75] always match the reconstruction volume resolution with the viewport resolution. Their CPU and GPU implementations take about 1453 s and 0.25 fps, respectively, for $128\ 128^2$ projections and a 256^2 viewport. Our framework offers a fairly large speedup over these. The timings presented in Table 5.4 use the floating point pipeline. If we use the dual-channel 16-bit pipeline, as mentioned in Section 3.3.1, we can obtain another speedup of 2.

Rendering results are presented in Figure 5.3. We produced images with the GPU D^{2}VR projection-based volume rendering with density only as well as with density+gradient renderings. The images are similar in quality than those reported in [76]. We show images rendered with all methods discussed before, including one rendered with the 16-bit pipeline. We observe that the images rendered with D^{2}VR reconstructing the gradients in volume space seem to have higher feature definition on zooms than those where the gradients were backprojected. A theoretical justification for this was presented in Section 5.3.2. At the same time, the method is also more computationally efficient.

We also observe that reconstruction into a larger viewport indeed does not require a reconstruction into a volume grid of identical resolution. Figure 5.3(i)(k) show that an upsampling on a slice-basis, followed by classification and shading produces very similar results (compare with Figure 5.3(h)(j)). Finally, we also observe (in Figure 5.3(g)) that the 16-bit pipeline produces high-quality images as well.

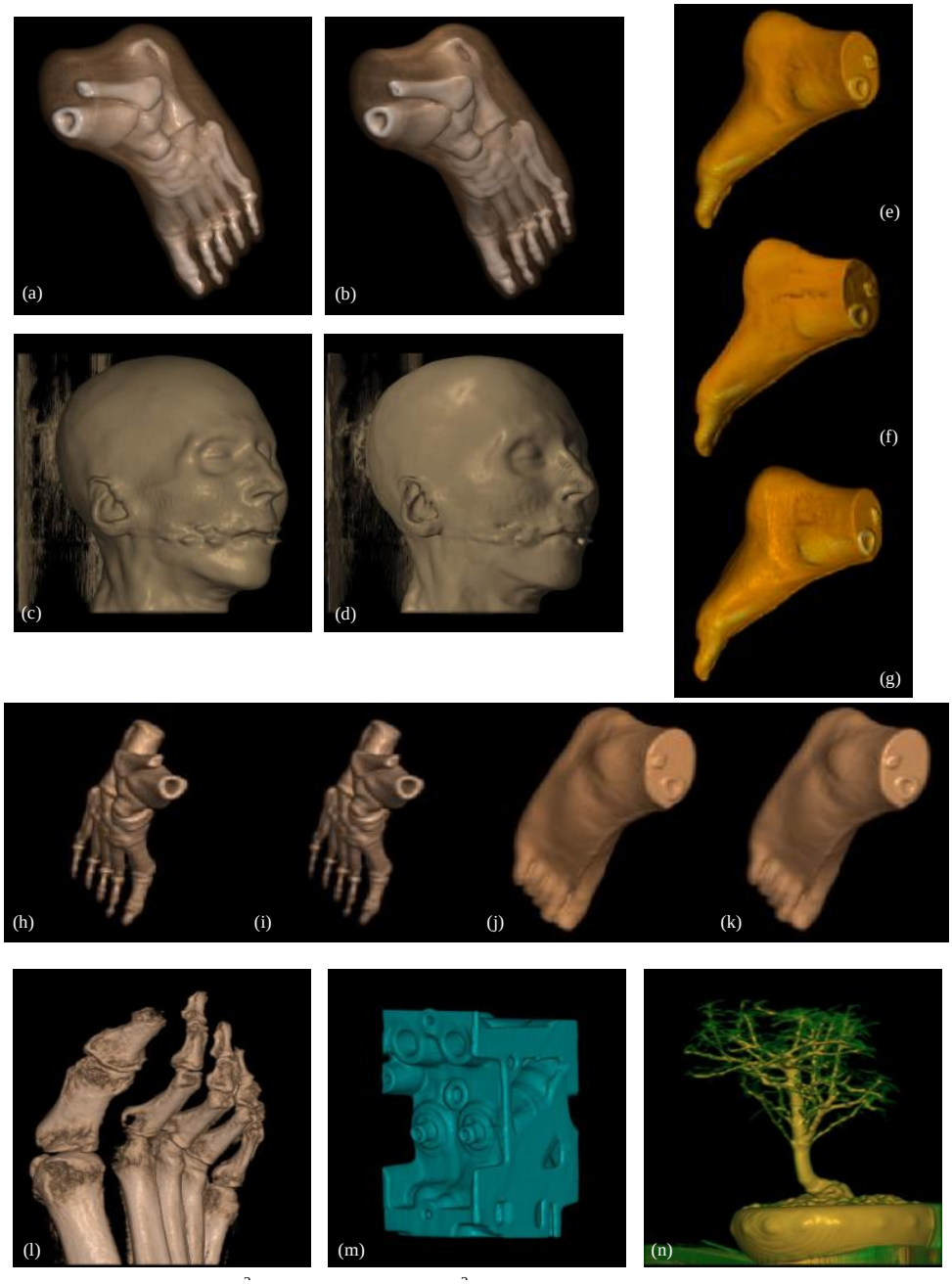


Figure 5.3: Rendering results: D^2VR+GFS (a, c, e, g, h-n); D^2VR+GFP (b, d, f); (g) is rendered from 16-bit pipeline; (h, j) are rendered from matched volume and viewport resolution, and (i, k) are rendered by upsampling on reconstructed volume slices.

Chapter 6

Conclusions

6.1 Summary

The new GPU framework and techniques in the thesis demonstrate that the recently escalating revolution in PC graphics board technology has enormous potential for computed tomography. For the first time, the quality that can be achieved rivals that obtained with software algorithms. Yet, speedups of over an order of magnitude for both analytical and iterative reconstruction methods are possible, on easily programmable and mass-produced hardware available for less than $500 at any local computer outlet. The results are especially encouraging since GPUs have increased in performance at a triple of Moore's law in the past few years. In addition to the basic emission tomography model, attenuation and scattering effects in iterative emission CT have been shown to be efficiently performed with GPUs. Unmatched operators are significantly more efficient and already seem effective to improve reconstruction quality in the presence of these effects.

Quality-wise, our evaluation study demonstrates that the slice interpolated schemes employed in the GPU framework have projection and reconstruction performances comparable to those of the grid-interpolated and box-line-integrated schemes, despite the reduced and non-uniform sampling rates. The optimal hexagonal-2$\times$-oversampled scheme we have

proposed in fact performs better than the cartesian-2$\times$-oversampled scheme, yet costs less computational effort, which enables the detector elements to be made larger while obtaining more accurate projections. By extending the sampling pattern from 2D to 3D for the purpose of reconstruction, we also demonstrated that 3D BCC lattices increases the recovery and detectability of small features, for example, small tumors in the brain, which is an important aspect in CT practice.

Targeting the widely-used Feldkamp filtered backprojection algorithm, we have described the first continuous and buffer-free commodity computing reconstruction pipeline for cone-beam CT. In our system, the projection data stream from the acquisition platform through a CPU-based filtering stage into a load-balanced GPU-accelerated backprojection framework. Our streaming CT can reconstruct a 512^3 volume at a rate of 40 to 50 512^2 and 1024^2 projections s^{-1}, which is the current production rate of commodity flat-panel detectors, and beyond. Larger volumes could be easily accommodated by using the now available single-platform dual- and quad-GPU setups, which provide up to 4GB of memory. Our pipeline provides a throughput rate and reconstruction speed of one to two orders of magnitude higher than existing systems based on commodity (PC) hardware. It is also faster than less readily available, but more costly PC-resident high-performance platforms based on the Cell BE processor and FPGA technologies. We achieve this by (i) exploiting many of the GPU-resident graphics facilities and (ii) careful load-balancing of the various GPU pipeline components in light of the specific computing task of CT reconstruction. Our rapid real-time reconstruction pipeline enables interactive use of commodity detectors and gantries, allowing, for example, interactive monitoring of musculoskeletal systems for positioning in interventional procedures as well as applications in image-guided surgery or radiotherapy. In fact, since the projection throughput is higher than their production rate on common detector hardware, it would even be possible to interject a 3D visualization rendering cycle into the reconstruction computation. Our results indicate that for the reconstruction settings tested here, a window size of 8 produced the best speedups, and along with it, the most optimal memory bandwidth and instruction execution patterns. This may change with different reconstruction scenarios, and this could be easily corrected for by adjusting the window size dynamically in an automatic binary-search optimization scheme, taking into account reconstructions just acquired.

George et al. [31] proposed a shear-based hierarchical backprojection method that achieved a complexity of $O(N^2 logP)$ when backprojecting an $N \times N$ pixel image from P projections. In practice, a speedup of an order of magnitude has been obtained due to the reduction of the operations counts without significant loss on the image quality. Compared to other popular hierarchical and Fourier-based methods, the shear-based approach does not need all the projections before the reconstruction begins. In addition, the algorithm scales with the size of the region of interest (ROI). Most importantly, the shear operation used in the approach is amenable to commodity graphics hardware, therefore an efficient implementation can be easily obtained. Due to above advantages over conventional backprojection methods and the fact that our GPU-based CT framework has obtained a performance of over two orders of magnitude faster than regular CPU methods, we would expect a further degree of speedup when both hardware and such optimized designed algorithms are coupled.

For the 3D Electron Tomography application, we have described new contributions and presented confirming results within three major areas: (i) an iterative reconstruction framework using algebraic methods, (ii) the compensation for the artifacts stemming from the limited-angle range at which projections can be obtained (iii) the acceleration of ET via commodity graphics hardware (GPUs). For the last, we have presented a novel data decomposition scheme that minimizes the number of GPU passes required, yielding speedups of 1-2 orders of magnitude with respect to present GPU-acceleration efforts. Our GPU-accelerated framework allows full-size 3D ET reconstructions to be performed at an order of minutes. We have also generalized the popular Simultaneous Iterative Reconstruction Technique (SIRT) to OS-SIRT, which allows researchers to optimize the wall clock time required for a GPU-accelerated reconstruction, enabling high-quality reconstructions to be obtained faster, by taking full advantage of the particularities associated with the GPU architecture and programming model. Our OS-SIRT optimizes the reconstruction performance by choosing the optimal number of subsets into which the projections are distributed (in random order). Here, it is likely that this optimal number of subsets will vary depending on the domain application and the general reconstruction scenario. Thus, in order to identify the optimal subset number for a new application setting, to be used later for repeated reconstructions within this application setting, one may simply run a series of experiments with different numbers of subsets and use the setting

with the shortest wall clock time required for the desired reconstruction quality. In fact, such strategies are typical for GPU-accelerated general-purpose computing applications (GPGPU). For example, the GPU bench (http://graphics.stanford.edu/projects/gpubench/) was designed to run a vast benchmark suite to determine the capabilities of the tested hardware. The presented GPU-accelerated ET platform allows ET researchers to achieve a major task which has so far been infeasible without expensive and extensive hardware: the iterative reconstruction of full-size 3D volumes.

Finally, our GPU framework provides an efficient solution to accelerate the D^2VR method which is able to deliver higher image quality than conventional volume rendering methods. For this we have built on our GPU-accelerated CT framework and extended it in the following ways. First, we allowed the reconstruction of arbitrary oriented volumes, which is not needed for CT, but is necessary for D^2VR since we must generate a matrix of samples exactly aligned with the image plane, whose orientation is completely arbitrary. Second, we incorporated various acceleration techniques to limit the reconstruction effort to only the visible and non-occluded (the relevant) matrix samples. Our system enables framerates of up to 2 frames/s for realistic dataset sizes, which is 1-2 orders of magnitudes faster than the software solution.

6.2 Future Work

Future work lies in the following aspects. While the axis-aligned data representation in our framework appeared sufficient, the 3D texture approach should be investigated once it is better supported by the hardware.

We will further study the "smart" multi-resolution grids using optimal lattices that adapt their resolution to the projection images provided. This, in some ways, is a continuation of earlier work on D^2VR [75] [76], where volume rendered images are generated directly from the projections and no volume lattice is ever generated.

Future work on hexagonal detector lattices should focus on further explorations of the tradeoff in lattice resolution and detector element size. More experiments will help to define the optimal configurations in that respect, given the task of the application. This goes hand in hand with

more research on better interpolation filters for the subsequent hexagonal-Cartesian regridding. Meanwhile, backprojection from $45\circ$ usually produces results with downgrade quality. We exepect the special layout of the hexagonal grid and its property of holding more frequency than the regular Cartesian grid can help alleviate the problem.

For the ET application, the impact of the GPU-based framework enables demanding iterative schemes crucial for the improvement of image resolution and contrast, such as iterative projection alignment and registration. On the more fundamental side, we are also planning to research the issue if even better convergence speeds can be obtained by varying the number of subsets as iterations continue. For instance, one might start with a small number of subsets to reconstruct the low detail aspects of the volume and then switch to a large number of subsets or SART to reconstruct the small detail. Multi-resolution lattices may also be a very good way to significantly accelerate convergence, that is, first do cycles on 4×4 or 2×2 binned (averaged) data, then interpolate and continue at higher resolution.

Bibliography

[1] A. H. Andersen, "Algebraic reconstruction in ct from limited views," *IEEE Transactions on Medical Imaging*, vol. 8, pp. 50–55, 1989.

[2] A. H. Andersen and A. C. Kak, "Simultaneous algebraic reconstruction technique (SART): a superior implementation of the ART algorithm," *Ultrasonic Imaging*, vol. 6, pp. 81–94, 1984.

[3] C. Axelsson and P.-E. Danielsson, "Three-dimensional reconstruction from cone-beam data in $O(N^3logN)$ time," *Physics in Medicine Biology*, vol. 39, pp. 477–491, 1994.

[4] C. Bai, G. Zeng, and G. Gullberg, "A slice-by-slice blurring model and kernel evaluation using the Klein-Nishina formula for 3D scatter compensation in parallel and converging beam SPECT," *Physics in Medicine and Biology*, vol. 45, pp. 1275–1307, 2000.

[5] S. Basu and Y. Bresler, "An $O(N^3logN)$ backprojection algorithm for the 3D Radon transform," *IEEE Transactions on Medical Imaing*, vol. 32, no. 2, pp. 76–88, 2002.

[6] T. Benson and J. Gregor, "Modified simultaneous iterative reconstruction technique for faster parallel computation," *IEEE Nuclear Science and Medical Imaging Symposium*, pp. 2715–2718, 2005.

[7] M. Bentum, B. Lichtenbelt, and T. Malzbender, "Frequency analysis of gradient estimators in volume rendering," *IEEE Transactions on Visualization and Computer Grpahics*, vol. 2, no. 3, pp. 242–254, 1996.

[8] J.-R. Bilbao-Castro, J. Carazo, J. Fernandez, and I. Garcia, "Parallelization and comparison of 3D iterative reconstruction algorithms," *Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 96–102, 2004.

[9] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: conjugate gradients and multigrid," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 917–924, 2003.

[10] S. Bonnet, A. Koenig, S. Roux, P. Hugonnard, R. Guillemaud, and P. Grangeat, "Dynamic X-ray computed tomography," in *Proceedings of the IEEE*, vol. 91, 2003, pp. 1574–1587.

[11] D. Brasse, B. Humbert, C. M. M. Rio, and J. Guyonnet, "Towards an inline reconstruction architecture for micro-CT systems," in *Physics in Medicine and Biology*, vol. 50, no. 24, 2005, pp. 5799–5811.

[12] S. Butler and M. I. Miller, "Maximum a Posteriori estimation for SPECT using regularization techniques on massively parallel computers," *IEEE Transactions on Medical Imaging*, vol. 12, no. 1, pp. 84–89, 1993.

[13] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," in *Proceedings of the symposium on Volume visualization*. ACM Press, 1994, pp. 91–98.

[14] B. Carvalho, G. Herman, and S. Matej, "Art for helical cone-beam ct reconstruction," *Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 249–252, 2001.

[15] Y. Censor, "On block-iterative maximization," *Journal of Information and Optimization Science*, vol. 8, pp. 275–291, 1987.

[16] K. Chidlow and T. Möller, "Rapid emission tomography reconstruction," in *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*. ACM Press, 2003, pp. 15–26.

[17] J. Conway and N. Sloane, Eds., *Sphere Packings, Lattices and Groups*, 2nd ed. Springer Verlag, 1993.

[18] F. Crow, "Summed-area tables for texture mapping," in *Proceedings of the ACM SIGGRAPH'84*, 1984, pp. 207–212.

[19] B. Csebfalvi, "Prefiltered gaussian reconstruction for high-quality rendering of volumetric data sampled on a body-centered cubic grid," in *Proceedings of IEEE Visualization*, 2005, pp. 40–48.

[20] B. De Man and S. Basu, "Distance-driven projection and backprojectionin three dimension," *Physics in Medicine and Biology*, vol. 49, pp. 2463–2475, 2004.

[21] D. C. Diez, H. Mueller, and A. S. Frangakis, "Implementation and performance evaluation of reconstruction algorithms on graphics processors," *Journal of Structural Biology*, vol. 157, no. 1, pp. 288–295, 2007.

[22] P. Eggermont, G. Herman, and A. Lent, "Iterative algorithms for large partitioned linear systems, with applications to image reconstruction," *Linear Algebra and Its Applications*, vol. 40, pp. 37–67, 1981.

[23] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2001, pp. 9–16.

[24] A. Entezari, R. Dyer, and T. Möller, "Linear and cubic box splines for the body centered cubic lattice," in *Proceedings of IEEE Visualization*, 2004, pp. 11–18.

[25] R. Fahrig, A. J. Fox, S. Lownie, and D. W. Holdsworth, "Use of a c-arm system to generate true three-dimensional computed rotational angiograms: preliminary in vitro and in vivo results," *American Journal of Neuroradiology*, vol. 18, no. 8, pp. 1507–1514, 1997.

[26] L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone beam algorithm," *Journal of the Optical Society of America A: Optics, Image Science, and Vision*, pp. 612–619, 1984.

[27] J.-J. Fernández, J.-M. Carazo, and I. García, "Three-dimensional reconstruction of cellular structures by electron microscope tomography and parallel computing," *Journal of Parallel and Distributed Computing*, vol. 64, pp. 285–300, 2004.

[28] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*. Addison-Wesley, 1990.

[29] J. Frank and B. F. McEwen, "Alignment by cross-correlation," in *Electron Tomography*, J. Frank, Ed. Plenum Press, 1992, pp. 205–214.

[30] M. Frigo and S. G. Johnson, "The design and implementation of fftw3," in *Proceedings of the IEEE*, vol. 93, no. 2, 2005, pp. 216–231.

[31] A. K. George and Y. Bresler, "Shear-based fast hierarchical backprojection for parallel-beam tomography," *IEEE Transactions on Medical Imaging*, vol. 26, no. 3, pp. 317–334, 2007.

[32] P. Gilbert, "Iterative methods for the 3D reconstruction of an object from projections." *Journal of Theoretical Biology*, vol. 76, pp. 105–117, 1972.

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

[34] N. Govindaraju, S. L. J. Gray, and D. Manocha, "A memory model for scientific algorithms on graphics processors," *UNC Technical Report*, 2006.

[35] P. Grangeat, "Analyse d'un Systeme D'Imagerie 3D par reconstruction a partir de radiographies X en geometrie. conique," Ph.D. dissertation, Ecole Nationale Superieure des Telecommunications, 1987.

[36] ——, "Mathematical framework of cone beam 3D reconstruction via the first derivative of the Radon transform," in *Mathematical Methods in Tomography (Lecture notes in Mathematics)*, G. Herman, A. Louis, and F. Natterer, Eds. Springer, 1991, pp. 66–97.

[37] M. J. Harris, W. V. Baxter, T. Scheuermann, and A. Lastra, "Simulation of cloud dynamics on graphics hardware," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, 2003, pp. 92–101.

[38] M. J. Harris, G. Coombe, T. Scheuermann, and A. Lastra, "Physically-based visual simulation on graphics hardware," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, 2002, pp. 109–118.

[39] G. Herman, Ed., *Image reconstruction from projections*. Academic Press, 1980.

[40] H. Hudson and R. Larkin, "Accelerated image reconstruction using ordered subsets of projection data," *IEEE Transactions of Medical Imaging*, vol. 13, pp. 601–609, 1994.

[41] D. Jaffray, J. Siewerdsen, J. Wong, and A. Martinez, "Flat-panel cone-beam computed tomography for image-guided radiation therapy," *International Journal of Radiation Oncology, Biology, Physics*, vol. 53, no. 5, pp. 1337–1349, 2002.

[42] P. Joseph, "An improved algorithm for reprojection rays through pixel images," *IEEE Transactions on Medical Imaging*, vol. 1, pp. 192–196, 1983.

[43] M. Kachelrieß, M. Knaup, and O. Bockenbach, "Hyperfast perspective conecbeam backprojection," in *IEEE Medical Imaging Conference*, 2006.

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

[45] U. Kapasi, W. Dally, B. Khailany, J. Ahn, P. Mattson, and J. Owens, "Programmable stream processors," *IEEE Computer*, vol. 36, no. 8, pp. 54–62, 2003.

[46] A. I. Katsevich, "An improved exact filtered backprojection algorithm for spiral computed tomography," *Advances in Applied Mathematics*, vol. 32, no. 4, pp. 681–697, 2004.

[47] M. King, B. Tsui, and T. Pan, "Attenuation compensation for cardiac single-photon emission computed tomographic imaging: Part 1, Impact of attenuation and methods of estimating attenuation maps," *Journal of Nuclear Cardiology*, vol. 2, pp. 513–524, 1995.

[48] J. Kniss, S. Premoze, C. Hansen, P. Shirley, and A. McPherson, "A model for volume lighting and modeling," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 150–162, 2003.

[49] J. Kniss, S. Premoze, C. Hansen, and D. Ebert, "Interactive translucent volume rendering and procedural modeling," in *Proceedings of IEEE Visualization*, 2002, pp. 109–116.

[50] J. Kole and F. J. Beekman, "Evaluation of accelerated iterative X-ray CT image reconstruction using floating point graphics hardware," *Physics in Medcine and Biology*, vol. 5, pp. 875–889, 2006.

[51] C. Kondo, S. Mori, M. Endo, K. Kusakabe, N. Suzuki, A. Hattori, and M. Kusakabe, "Real-time volumetric imaging of human heart without electrocardiographic gating by 256-detector row computed tomography: initial experience," *Journal of Computer Assisted Tomography*, vol. 29, no. 5, pp. 694–698, 2005.

[52] J. Krüeger and R. Westermann, "Acceleration techniques for GPU-based volume rendering," in *Proceedings of IEEE Visualization*, 2003.

[53] J. Krüeger and R. Westermann, "Linear algebra operators for gpu implementation of numerical algorithms," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 908–916, 2003.

[54] H. Kudo, F. Noo, and M. Defrise, "Quasi-exact filtered backprojection algorithm for long-object problem in helical cone-beam tomography," *IEEE Transactions on Medical Imaging*, vol. 19, no. 9, pp. 902–921, 2000.

[55] M. C. Lawrence, "Least-squares method of alignment using markers," in *Electron Tomography*, J. Frank, Ed. Plenum Press, 1992, pp. 197–204.

[56] S. Lee, H. Kim, I. Chun, M. Cho, S. Lee, and M. Cho, "A flat-panel detector based micro-CT system: performance evaluation for small-animal imaging," *Physics in Medicine and Biology*, vol. 48, pp. 4173–4185, 2003.

[57] R. M. Lewitt, "Alternatives to voxels for image representation in iterative reconstruction algorithms," *Physics in Medicine and Biology*, vol. 37, no. 3, pp. 705–716, 1992.

[58] W. Li, X. Wei, and A. Kaufman, "Implemented lattice boltzmann computation on graphics hardware," in *The Visual Computer*, 2003.

[59] Z. Liang, T. Turkington, D. Gilland, R. Jaszczak, and R. Coleman, "Simultaneous compensation for attenuation, scatter and detector response for spect reconstruction in three dimensions," *Physics in Medicine and Biology*, vol. 37, pp. 587–6 031 274–1280, 1992.

[60] B. Lichtenbelt, R. Crane, and S. Naqvi, Eds., *Introduction to Volume Rendering*. Prentice-Hall, 1998.

[61] X. Liu, M. Defrise, L. Desbat, and M. Fleute, "Cone-beam reconstruction for a C-arm CT system," in *IEEE Nuclear Science Symposium Conference*, vol. 3, 2001, pp. 1489–1493.

[62] T. Malzbender, "Fourier volume rendering," *ACM Transactions on Graphics*, vol. 12, no. 3, pp. 233–250, 1993.

[63] S. Marschner and R. Lobb, "An evaluation of reconstruction filters for volume rendering," in *Proceedings of IEEE Visualization*, 1994, pp. 100–107.

[64] S. Matej and R. Lewitt, "Efficient 3D grids for image reconstruction using spherically-symmetrical volume elements," in *IEEE Transactions on Nuclear Science*, vol. 42, 1995, pp. 1361–1370.

[65] T. Meng, B. Smith, A. Entezari, A. E. Kirkpatrick, D.Weiskopf, L. Kalantari, and T. Möller, "On visual quality of optimal 3d sampling and reconstruction," in *Graphics Interface*, 2007, pp. 265–272.

[66] T. Möller, R. Machiraju, K. Mueller, and R. Yagel, "Evaluation and design of filters using a taylor series expansion," *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 2, pp. 184–199, 1997.

[67] S. Mori, M. Endo, T. Tsunoo, K. Murase, and H. Fujiwara, "Evaluation of weighted FDK algorithms applied to four-dimensional CT

(4D-CT)," in *IEEE Nuclear Science Symposium Conference*, vol. 5, 2004, pp. 3243–3245.

[68] K. Mueller and R. Yagel, "The use of dodecahedral grids to improve the efficiency of the algebraic reconstruction technique (ART)," in *Annals of Biomedical Engineering, Special issue, 1996 Annual Conference of the Biomedical Engineering Society*, 1996, pp. S–66.

[69] K. Mueller and F. Xu, "Practical considerations for GPU-accelerated CT," in *IEEE International Symposium on Biomedical Imaging*, 2006.

[70] K. Mueller and R. Yagel, "Rapid 3D cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2D texture mapping hardware," *IEEE Transactions on Medical Imaging*, vol. 19, no. 12, pp. 1227–1237, 2000.

[71] K. Mueller, R. Yagel, and J. J. Wheller, "Anti-aliased 3D cone-beam reconstruction of low-contrast objects with algebraic methods," *IEEE Transactions on Medical Imaging*, vol. 18, no. 6, pp. 519–537, 1999.

[72] N. Neophytou and K. Mueller, "Space-time points: 4d splatting on effcient grids," in *Symposium of Volume Visualization and Graphics*, 2002, pp. 97–106.

[73] ——, "GPU accelerated image aligned splatting," in *Volume Graphics*, 2005, pp. 197–205.

[74] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. Purcel, "A survey of general-purpose computation on graphics hardware," in *Eurographics*. Eurographics Association, 2005, pp. 21–51.

[75] P. Rautek, "D^2VR high-quality volume rendering of projection-based volumetric data," Master's thesis, Vienna University of Technology, 2005.

[76] P. Rautek, B. Csebfalvi, S. Grimm, S. Bruckner, and M. E. Gröller, "D^2VR: high-quality volume rendering of projection-based volumetric data," in *Joint Eurographics - IEEE TCVG Symposium on Visualization*, 2006, pp. 211–218.

[77] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, "Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization," in *Proceedings of the Eurographics/SIGGRAPH Workshop on Graphics Hardware*, 2000, pp. 109–118.

[78] D. Ros, C. Falcon, I. Juvells, and J. Pavia, "The influence of a relaxation parameter on SPECT iterative reconstruction algorithms," *Physics in Medicine and Biology*, vol. 39, no. 41, pp. 583–595, 1996.

[79] K. Sandberg, D. Mastronarde, and G. Beylkina, "A fast reconstruction algorithm for electron microscope tomography," *Journal of Structural Biology*, vol. 144, pp. 61–72, 2003.

[80] T. Schiwietz, T. Chang, P. Speier, and R. Westermann, "MR image reconstruction using the GPU," in *Proceedings of the SPIE*, vol. 6142, 2006, pp. 1279–1290.

[81] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli, "Fast shadows and lighting effects using texture mapping," in *ACM SIGGRAPH Computer Graphics*, 1992, pp. 249–252.

[82] L. Shepp and Y. Vardi, "Maximum likelihood reconstruction for emission tomography," *IEEE Transactions of Medical Imaging*, vol. 1, pp. 113–122, 1982.

[83] R. Siddon, "Fast calculation of the exact radiological path length for a three-dimensional CT array," *Medical Physics*, vol. 12, pp. 252–255, 1985.

[84] U. Skoglund, L. Ofverstedt, R. Burnett, and G. Bricogne, "Maximum-entropy three-dimensional reconstruction with deconvolution of the contrast transfer function: a test application with adenovirus," *Journal of Structural Biology*, vol. 117, pp. 173–188, 1996.

[85] S. Stegmaier, M. Strengert, T. Klein, and T. Ertl, "Simple and flexible volume rendering framework for graphics-hardware-based raycasting," in *Volume Graphics Workshop*, 2005, pp. 187–195.

[86] M. Stijnman, R. Bisseling, and G. Barkema, "Partitioning 3D space for parallel many-particle simulations," *Computer Physics Communications*, vol. 149, no. 3, pp. 121–134, 2003.

[87] R. Strzodka, "Virtual 16 bit precise operations on rgba8 textures," in *Vision Modeling and Visualisation*, 2002, pp. 171–178.

[88] T. Sumanaweera and D. Liu, "Medical image reconstruction with the fft," in *GPU Gems II*, M. Pharr and R. Fernando, Eds. Addison Wesley, 2005, pp. 765–784.

[89] K. Taguchi, "Temporal resolution and the evaluation of candidate algorithms for four-dimensional CT," *Medical Physics*, vol. 30, no. 4, pp. 640–650, 2003.

[90] K. C. Tam, S. Samarasekera, and F. Sauer, "Exact cone beam CT with a spiral scan," *Physics in Medicine and Biology*, vol. 43, pp. 1015–1024, 1998.

[91] T. Theußl, T. Möller, and E. Gröller, "Optimal regular volume sampling," *Proceedings of IEEE Visualization*, pp. 91–98, 2001.

[92] P. Thévenaz, T. Blu, and M. Unser, "Interpolation revisited," *IEEE Transactions on Medical Imaging*, vol. 19, no. 7, pp. 739–758, 2000.

[93] H. Turbell and P.-E. Danielsson, "Helical cone beam tomography," *International Journal of Imaging System and Technology*, vol. 11, pp. 91–100, 2000.

[94] H. Turbell, "Cone-beam reconstruction using filtered backprojection," Ph.D. dissertation, Linköping University, Sweden, SE-581 83 Linköping, Sweden, February 2001, dissertation No. 672, ISBN 91-7219-919-9.

[95] J.-J. Verlaan, E. B. van de Kraats, W. Dhert, and F. C. Oner, "The role of 3-d rotational X-ray imaging in spinal trauma," *Injury*, vol. 36, pp. B98–B103, 2005.

[96] D. V. D. Ville, T. Blu, M. Unser, W. Philips, I.Lemahieu, and R. V. de Walle., "Hex-splines: a novel spline family for hexagonal lattices," in *IEEE Transactions on Image Processing*, vol. 13, no. 6, 2004, pp. 758–772.

[97] Z. Wang, G. Han, T. Li, and Z. Liang, "Speedup OS-EM image reconstruction by PC graphics card technologies for quantitative SPECT with varying focal-length fan-beam collimation," *IEEE Transactions on Nuclear Science*, vol. 52, no. 5, pp. 1274–1280, 2005.

[98] F. Xu and K. Mueller, "Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware," *IEEE Transactions on Nuclear Science*, vol. 52, no. 3, pp. 654–663, 2005.

[99] ——, "GPU-accelerated D^2VR," in *Volume Graphics*, 2006, pp. 23–30.

[100] ——, "GPU-acceleration of attenuation and scattering compensation in emission computed tomography," in *9th Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine (High Performance Image Reconstruction Workshop)*, 2007.

[101] ——, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Physics in Medicine and Biology*, vol. 52, pp. 3405–3419, 2007.

[102] X. Xue, A. Cheryauka, and D. Tubbs, "Acceleration of fluoro-CT reconstruction for a mobile C-arm on GPU and FPGA hardware: a simulation study," in *Proceedings of the SPIE*, vol. 6142, 2006, pp. 1494–1501.

[103] G. Zeng and G. Gullberg, "Unmatched projector/backprojector pairs in an iterative reconstruction algorithm," *IEEE Transactions on Medical Imaging*, vol. 19, no. 5, pp. 548–555, 2000.

[104] S. Q. Zheng, M. B. Braunfeld, J. W. Sedat, and D. A. Agard, "An improved strategy for automated electron microscopic tomography," *Journal of Structural Biology*, vol. 147, no. 2, pp. 91–101, 2004.

[105] S. Q. Zheng, B. Keszthelyi, E. Branlund, J. M. Lyle, M. B. Braunfeld, J. W. Sedat, and D. A. Agard, "UCSF tomography: An integrated software suite for real-time electron microscopic tomographic data collection, alignment, and reconstruction," *Journal of Structural Biology*, vol. 157, no. 1, pp. 138–147, 2006.