Efficient ray tracing on 3D regular grids for fast generation of digitally reconstructed radiographs in iterative tomographic reconstruction techniques

Jonas Dittmann*

*Lehrstuhl für Röntgenmikroskopie, Universität Würzburg, Josef Martin Weg 63, 97074 Würzburg, Germany

Abstract

Cone beam projection is an essential and particularly time consuming part of any iterative tomographic reconstruction algorithm. On current graphics hardware especially the amount and pattern of memory accesses is a limiting factor when read-only textures cannot be used. With the final objective of accelerating iterative reconstruction techniques, a non-oversampling Joseph-like raytracing projection algorithm for three dimensions featuring both a branchless sampling loop and a cache friendly memory access pattern is presented.

An interpretation of the employed interpolation scheme is given with respect to the effective beam and voxel models implied. The method is further compared to existing techniques, and the modifications required to implement further voxel and beam shape models are outlined.

Both memory access rates and total run time are benchmarked on a current consumer grade graphics processing unit and explicitly compared to the performance of a classic Digital Differential Analyzer (DDA) algorithm. The presented raytracer achieves memory access rates of 292 GB/s in read-and-write memory and 502 GB/s in read-only texture memory. It outperforms the DDA in terms of total run time by a factor of up to five and achieves 170 to 300 projections of a 512³ voxel volume per second.

Keywords: ray tracing; ray casting; forward projection; X-ray transform; hough transform; digitally reconstructed radiograph; volume visualization; voxel basis function; computed tomography; tomographic reconstruction; iterative reconstruction;
1. Introduction

1.1. Iterative tomographic reconstruction

X-ray Computed Tomography is an important instrument for non invasive imaging of the interior of opaque objects by means of a multitude of X-ray images at different orientations of the object in question. Classically, the reconstruction of the sought image out of the cumulative absorption images is done by Filtered Backprojection, a method which analytically derives from the relation between the Fourier transform of a tomographic slice and the Fourier transform of its X-ray projections (the Fourier slice theorem).

There are many cases though in which iterative methods as e.g. SART (Simultaneous Algebraic Reconstruction Technique, [1]) are required, i.e. methods that simulate X-ray projections of a preliminary solution and manipulate this solution until the calculated projections match the observed ones. These techniques allow for more general imaging models and can be combined with iterative regularization techniques that are required in cases of incomplete data. The major drawback of iterative methods is their high demand for computational power, as any iterative method consists of many repetitions of forward projections (i.e. simulations of the X-ray projections) and corresponding backprojections of corrections to the solution.

The task of forward projection is equivalent to evaluating the intersection of X-ray beams hitting the detector bins with the pixels or voxels of the tomogram. The problem can be stated in terms of either of two underlying approaches. First, one may regard the connecting lines between X-ray source and detector pixels and then determine intersections with image pixels or voxels (ray driven or ray tracing methods). Secondly, one can as well regard the connecting lines between source and image voxels and rather determine the affected detector pixels (voxel driven methods).

Although voxel driven methods inherently allow for optimal memory access patterns within the volume image to be projected, ray tracing methods are better suited for parallel architectures as no explicit synchronization between different threads handling independent rays is necessary. Their mostly non-contiguous memory access pattern typically is a speed limiting factor though.

1.2. Ray tracing

Ray tracing has been studied since the advent of raster graphics. Besides iterative tomographic reconstruction, its applications range from rendering lines on a raster grid over volume graphics visualization to efficient calculation of “digitally reconstructed radiographs” (DRR) for 3D image registration and radiation therapy treatment planning.

The key concept in the majority of fast ray tracing algorithms on regular grids is the notion of a “driving axis” [2–6] as already introduced in 1962 by Bresenham in the context of line drawing. The line or ray will be traced in unit steps of the designated driving axis and the intersected pixels or voxels are determined by evaluating the line equation at each step. Effective limiting of
any line’s slope to a range within $\pm 45^\circ$ by choice of the driving axis ensures that no relevant pixels or voxels are skipped.

This concept is, in the context of DRR, explicitly or implicitly used e.g. by Josephs’ algorithm [2], by shear-warp techniques [7, 8] or ray-driven splatting algorithms [9, 10] as well as by the recently popular “Distance Driven Method” [11]. It also emerges naturally from general sampling considerations, as interpolation can thus be avoided along the driving axis.

Prominent alternative techniques are the algorithm by Siddon [12] and variants thereof [13–16], which trace lines in irregular steps from intersection to intersection with any of the planes perpendicular to the coordinate axes. Their final objective of calculating the exact line-voxel intersection though can as well be achieved with driving-axis based algorithms [3, 17].

An important factor for the design of a ray tracer for DRR purposes is the assumed underlying system model. Most prevalent is the assumption that imaged objects can be exactly modeled by cubic voxels of homogeneous density and incident radiation by rectangular beam profiles of finite extent. Much effort has been put into the development of exact ray tracers in this respect [3, 17–25].

When arguing that there is no outstanding reason to assume homogeneous cubic voxels, the complexity for an “exact” ray tracer can be considerably reduced by assuming isotropic basis functions instead of box-shaped voxels [10, 26–28] as modeling of both voxel and beam profiles can then be merged into an interpolation function (footprint) acting on ray-voxel distances. Other footprint based methods replace the latter distance by algorithmically more convenient approximations [2, 11, 29]. Joseph’s projector in particular uses an implicit footprint emerging from the interpolated sampling only within the immediate neighborhood of a sampling point. It has been adopted in the past e.g. to trace X-rays through parallel stacks of textured polygons [30, 31] or for list mode PET reconstructions [32].

Beam width modeling can, with Joseph’s method, easily approximated by sufficiently matching the voxel grid size to the beam width. Both oversampling of rays or sampling an extended neighborhood of each sampling point can thus be avoided and the total amount of required memory accesses kept to a necessary minimum. Although fourier based projection methods (for parallel beam geometries) [33] or divide and conquer approaches exploiting partial overlaps of rays at close by viewing angles allow to systematically reduce the amount of memory operations even further [34], those methods only apply when many angular densely sampled projections are to be calculated simultaneously.

Independent of employed model and sampling technique, researchers have tried to utilize the high processing power of dedicated graphics hardware in order to speed up CT reconstruction. Starting with SGI graphics workstations in the 1990s, the tomographic problem began to be restated in terms of graphics operations that were efficiently implemented in hardware. Detailed reviews of the developments up to now are given in [35, 36]. Current general purpose graphics processors are much more versatile and although efficient algorithms still need to be tailored to the specific strengths of the hardware, they are not limited to classic graphics operations such as texture mapping anymore.
Figure 1: Left: Illustration of the ray tracing problem for a ray \( \vec{r} \) between source \( \vec{s} \) and detector pixel \( \vec{d} \) through a voxel volume in steps of the sampling increment vector \( \tilde{r} \). Coordinates \( \vec{s} \) and \( \vec{d} \) are defined with respect to the origin of the voxel grid, and distance units are normalized to the voxel grid spacing. \( \tilde{r} \) is defined as \( \tilde{r} = \vec{r}/r_m \) (with \( m = \text{argmax}_i(|r_i|) \)) and thus \( 1 \leq \|\tilde{r}\|_2 \leq \sqrt{3} \). This choice of \( \tilde{r} \) ensures unit increments along at least one axis (the “driving axis”). Right: The driving axis concept is based on the observation that a line with slope \( \leq 1 \) always intersects boxes at most two pixels wide in orthogonal directions (colored boxes). Sampling is done in integer steps along the driving axis (square markers), i.e. in intervals of \( \|\tilde{r}\|_2 \) along the line.

The present article will contribute a particularly short and instructive formulation of a Joseph-like ray tracer for three dimensions, which will thus be referred to as “Generalized Joseph Projector” (GJP) in the following. Smooth alternatives to the linear sampling kernel as well as a suiting choice of voxel grid spacing in case of cone beam geometries will be discussed. Although the driving axis concept will be applied to derive the sampling pattern, the presented algorithm does not actually use a driving axis in the sense of dedicated code branches, i.e. handles all cases in a single code branch. It is designed with modern graphics processor architectures in mind and addresses three typical bottlenecks: kernel complexity, branching and memory access. The proposed algorithm does in particular not base its efficiency on fast read-only GPU texture memory and is thus well suited for iterative methods that require to constantly modify the voxel volume.

2. Method

The basic problem is illustrated in Fig. 1. A ray \( \vec{r} \) emanating from a source position \( \vec{s} \) traverses a voxel volume and hits a detector pixel at location \( \vec{d} \). Along its intersection with the volume, the latter will be sampled in steps of \( \tilde{r} \), also taking into account the finite width of an X-ray beam. For convenience, the coordinate system is aligned with the voxel grid, i.e. the origin is at the corner of the voxel volume and all distance units will be expressed in terms of grid spacing.

2.1. Ray tracing

Classically and particularly in 2D line drawing or ray tracing problems, the path to trace is expressed as a linear equation for which the driving axis defines the input parameter. The driving axis is chosen such that the derivative or slope of the line equation lies within \(-1\) and \(1\), and the line is evaluated in unit
steps of the driving axis (cf. Fig. 1). While this scheme is optimal in the sense that it neither regards any affected voxels twice nor skips any, the particular formulation in terms of line equations is unfavorable on SIMD architectures (single instruction, multiple data; also “vector processor”) due to the required case differentiation depending on the current driving axis.

We will thus switch to a more convenient parametric vector representation, based on the positions of source $\vec{s}$ and detector pixel $\vec{d}$ relative to the volume origin. All points $\vec{p}$ on the line are then characterized by

$$\vec{p} = \vec{s} + l\vec{r}$$

with

$$\vec{r} = \vec{d} - \vec{s}$$

and $l$ being the free parameter. The driving axis $m$ is then defined by the largest coefficient of $\vec{r}$:

$$m = \text{argmax}_i (|r_i|).$$

For convenience, we will define the indices $\bar{m}_1$ and $\bar{m}_2$ to refer to the remaining non-driving axes.

An increment vector $\vec{r}$ between successive sampling points will be chosen such that the resulting sampling points are always aligned with the driving axis, which holds for

$$\vec{r} = \frac{\vec{r}}{r_m}. $$

The first possible sampling point is defined by the intersection of the ray with the plane perpendicular to the driving axis $m$, i.e.

$$[\vec{s} + o\vec{r}]_m \perp 0$$

$$\Rightarrow o = -s_m$$

where $o$ is the distance between source and first sampling plane in units of the sampling increment $||\vec{r}||$. $o$ will thus be called “sampling offset” in the following.

The volume can now be sampled at points $\vec{p}^{(i)}$ along the defined path in unit steps of axis $m$ by evaluating

$$\vec{p}^{(i)} = (\vec{s} + o\vec{r}) + i\vec{r}$$

for integer $i \in [0, i_{\text{max}}]$, where $i_{\text{max}}$ is defined by the extent of the voxel volume along axis $m$.

The driving axis component $p_m^{(i)}$ is always integer by construction of sampling increment $\vec{r}$, sampling offset $o$ and sampling index $i$. When sampling the volume at $\vec{p}^{(i)}$, interpolation is thus only required along the non-driving axes, i.e. within a group of $2 \times 2$ voxels. Fig. 2 gives a detailed illustration of this volume traversal scheme.
Figure 2: Left: Illustration of the proposed ray tracing and sampling scheme. The volume is sampled at integer positions of the driving axis defined by the largest coefficient of $\mathbf{r}$ or equivalently in intervals of $\bar{r}$ (Eq. 4). Interpolation is thus only necessary within 4-voxel blocks along the remaining axes which are marked in changing color with increasing sampling index. Above, the intersection points $\mathbf{p}^{(i)}$ are shown in their respective sampling planes described by their nearest neighbor voxel locations $\mathbf{v}^{(i,1-4)}$. Right: The concept of a ray piercing consecutive 4-voxel planes is illustrated for all possible driving axes. The sampling planes’ orientation automatically emerges from the nearest neighbor selection according to Eq. 15 without requiring case differentiations.

Figure 3: Left: relation between sampling point $\mathbf{p}^{(i)}$ and nearest neighbor voxels $\mathbf{v}^{(i,1-4)}$. The distances $\|\mathbf{p}^{(i)} - \mathbf{p}^{(i)}_{m1}\|$ and $\|\mathbf{p}^{(i)} - \mathbf{p}^{(i)}_{m2}\|$ are used to calculate interpolation weights (with $\lfloor \rfloor$ designating the floor operation). Center and right: Examples of separable spline interpolation and bi-linear interpolation (cf. Eqs. 11, 13, 14)
2.2. Interpolation

Given the minimal set of $2 \times 2$ grid points, first order (i.e. bilinear) interpolation is indicated. Linear interpolation is e.g. used by Joseph [2], and combining the above choices of sampling points with bi- or tri-linear\textsuperscript{1} interpolation indeed results in a 3D generalization of Joseph’s method.

To avoid aliasing artifacts caused by high frequency components of the kinked linear interpolation kernel, more bandlimited smooth kernels such as the raised cosine or Hann(ing) function have been proposed for image interpolation [26, 37]. Sunnegårdh and Danielsson motivated a similarly bell shaped kernel as strip integral model over a linearly interpolated image [29].

All of the above mentioned first order methods are described by a one dimensional interpolation kernel

$$w(d); \ d \in [0, 1)$$

acting on the fractional parts (i.e. normalized distances to grid planes) $d_{1/2} = p_{m_{1/2}}^{(i)} - \lfloor p_{m_{1/2}}^{(i)} \rfloor$ of $p_{m_1}^{(i)}$ and $p_{m_2}^{(i)}$ (cf. Fig. 3). The following properties are required for interpolation:

$$w(1 - d) = 1 - w(d)$$

$$w(0) = 1,$$

i.e. interpolation is symmetric and short distances result in high weight. A sensible interpolation kernel should also be monotonic:

$$w'(d) \leq 0 \ \forall d \in [0, 1).$$

Interpolation in higher dimensions is achieved by multiplication:

$$w(d_1, \ldots, d_n) = \prod_i w(d_i).$$

The more bandlimiting kernels have in common that they further ensure smoothness at the boundaries, which for first order methods can be guaranteed by requiring the derivative $w'(d)$ to vanish on grid points:

$$w'(d)|_{d \in \{0, 1\}} = 0.$$

In the following we will regard two examples, namely linear interpolation as well as a smooth spline kernel modeled in the style of the Hanning function (yet less computationally expensive):

$$w_{\text{lin}}(d) = 1 - d$$

$$w_{\text{spl}}(d) = 1 - 3d^2 + 2d^3,$$

---

\textsuperscript{1}As the sampling points are aligned with one axis, tri-linear interpolation is identical to bi-linear interpolation within the remaining two dimensions here.
2.3. Evaluation of sampling grid points

By construction, every sampling point \( \tilde{p}^{(i)} \) as defined by Eqs. 4–6 has (in 3D) at most two non-integer components. These non-integer coordinates necessarily lie between two integer ones along their respective axes. For each sampling point, the group of four nearest neighbor voxels (shown as \( 1 \times 2 \times 2 \) boxes in Fig. 2) can thus be identified by regarding all combinations of floor and ceiling values of the non-integer components \( p^{(i)}_{m_1} \) and \( p^{(i)}_{m_2} \):

\[
\begin{align*}
\tilde{v}^{(i,1)} &= (\lfloor p^{(i)}_{m_1} \rfloor, \lfloor p^{(i)}_{m_2} \rfloor, \lfloor p^{(i)} \rfloor) \\
\tilde{v}^{(i,2)} &= (\lfloor p^{(i)}_{m_1} \rfloor, \lceil p^{(i)}_{m_2} \rceil, \lceil p^{(i)} \rceil) \\
\tilde{v}^{(i,3)} &= (\lceil p^{(i)}_{m_1} \rceil, \lfloor p^{(i)}_{m_2} \rfloor, \lceil p^{(i)} \rceil) \\
\tilde{v}^{(i,4)} &= (\lceil p^{(i)}_{m_1} \rceil, \lceil p^{(i)}_{m_2} \rceil, \lfloor p^{(i)} \rfloor)
\end{align*}
\]

where \( \lfloor \rfloor \) and \( \lceil \rceil \) designate the floor and ceiling operators respectively. In this form, the determination of \( \tilde{v}^{(i,1–4)} \) requires explicit treatment of three different cases depending on \( m \in \{1, 2, 3\} \). This can however for algorithmic convenience be formulated in a completely driving axis (i.e. \( m \)) agnostic way:

\[
\begin{align*}
\tilde{v}^{(i,1)} &= ([p^{(i)}_{m}], [p^{(i)}], [p^{(i)}]) \\
\tilde{v}^{(i,2)} &= ([p^{(i)}_{m}], [p^{(i)}], [p^{(i)}]) \\
\tilde{v}^{(i,3)} &= ([p^{(i)}_{m}], [p^{(i)}], [p^{(i)}]) \\
\tilde{v}^{(i,4)} &= ([p^{(i)}_{m}], [p^{(i)}], [p^{(i)}])
\end{align*}
\]

(15)

by exploiting that

\[
[p^{(i)}_{m}] = [p^{(i)}] = p^{(i)}_m.
\]

Independent of \( m \in \{1, 2, 3\} \), the vectors \( \tilde{v}^{(i,1–4)} \) define a group of at most four voxels in a plane perpendicular to the driving axis. Special cases arise when either of \( p^{(i)}_{m_1} \) or \( p^{(i)}_{m_2} \) is also integer, which leads to only one or two different \( \tilde{v}^{(i,1–4)} \). Lines 13 and 14 in Algorithm 1 ensure that duplicate voxels within \( \tilde{v}^{(i,1–4)} \) will receive zero weight.

2.4. Effective beam and volume modeling

Sampling in the context of X-ray projection (equivalently DRR) implies integration within the sampled volume over a box shaped beam profile of finite width perpendicular to the ray direction. The proposed sampling kernels (Eqs. 13 and 14) themselves thus represents a convolution of a kernel describing the projection of a voxel basis function (its “footprint”) and a box shaped kernel describing the X-ray beam (cf. Fig. 4). The effective voxel footprint \( f(d) \) implicitly modeled by Eqs. 13 and 14 can thus be obtained by differentiation
Figure 4: 1D interpolation kernel $w_{\text{spl}}(|d|)$ (right) and interpretation as convolution of effective voxel footprint and beam profile (left). For $w_{\text{lin}}(|d|)$ (not shown), the voxel footprint is equally box shaped as the beam profile.

Figure 5: Effective range of influence resulting from interpolation among the nearest neighbor pixels at a single sampling point. The sampling point (black square) is chosen to be infinitesimally close to a grid point. The voxels considered by the sampling procedure are marked in blue. Assuming a voxel grid spacing matched to the modeled beam width, the modeling is correct when the ray is axis-aligned (a). In the worst case, when the ray runs diagonal through the grid, the effective modeled beam width and voxel diameter is $1/\sqrt{2}$ (b). If desired, the problem can be (almost) mitigated by oversampling by a factor of 2 in combination with an orientation aware interpolation function (c).

and coordinate shifting:

$$f_{\text{lin}}(d) = \begin{cases} 1 & -0.5 < d \leq 0.5 \\ 0 & \text{else} \end{cases}$$

$$f_{\text{spl}}(d) = \begin{cases} \frac{3}{2}(1 - 4d^2) & -0.5 < d \leq 0.5 \\ 0 & \text{else} \end{cases}.$$  

These results can conversely be verified by convolution with a box function of unit width and heights centered about the origin. Note that $d$ is always measured along grid axes, i.e. typically not perpendicular to the ray direction. This implies that both the footprint and the implicitly modeled beam width become narrower when the ray is oriented diagonal in the voxel grid. Figure 5 gives an illustration of this orientation dependent effective voxel and beam width modeling, which arises from the nearest neighbor evaluation based on the sampling points $\vec{p}^{(i)}$ rather than on the true geometric distance

$$d_{\text{geom}}(\vec{p}^{(i)}, \vec{v}) = \| (\vec{p}^{(i)} - \vec{v}) \times \vec{r} \| / \| \vec{r} \|$$

between voxel centers $\vec{v}$ in the vicinity of $\vec{p}^{(i)}$ and an arbitrarily oriented line $\vec{r}$. The effective implicit voxel and beam widths can be defined as the smallest
geometric distance between center ray and voxels not considered by the sampling scheme. This distance is given by

\[
\min_{\{\vec{v}\}} \left( \frac{\|\vec{v} \times \vec{r}\|}{\|\vec{r}\|} \right) \{\vec{v}|v_{m_1}, v_{m_2} \in \{-1,0,1\} \land (v_{m_1}, v_{m_2}) \neq (0,0) \land v_m = 0\},
\]

where \{\vec{v}\} represents the set of 8 possibly unconsidered nearest neighbors within the sampling plane to a sampling point infinitely close to one grid point. In 2D, this reduces to \(1/\|\vec{r}\|\), which has a minimum of \(1/\sqrt{2}\) for diagonal rays as shown in Fig. 5 (cf. Eq. 4). \(1/\sqrt{2}\) is also the minimal effective beam width in the 3D case. For a 3D diagonal ray \((\vec{r} = (1,1,1))\) the beam width increases to \(\sqrt{2}/3\).

2.5. Extension to isotropic voxel basis function models

In order to realize truly isotropic footprints and beam widths without sacrificing the branchless sampling scheme defined by Eqs. 6 and 15, oversampling can be employed in combination with evaluation of the true geometric distance (Eq. 16) between \(\vec{v}^{(i,1-4)}\) and the ray. By modeling two (or four, in 3D) beams of 1/2 width per detector pixel as also shown in Fig. 5(c), the width modeling can be made almost independent of ray orientation with only a small error remaining in the special case of beams oriented close to a 2D diagonal with sampling points very close to grid points.

2.6. Implementation

Algorithm 1 provides a commented pseudo code implementation of the proposed GJP algorithm, including explicit sampling and interpolation. The algorithm consists of four parts: the initialization phase sets up the major axis \(m\), the sampling increment \(\vec{r}\), the sampling offset \(o\) and the maximum sampling index \(i_{\text{max}}\). Based on these, the sampling points \(\vec{p}^{(i)}\) are evaluated at the beginning of the sampling loop. For each \(\vec{p}^{(i)}\) the affected voxels and and their respective interpolation weights are explicitly determined. Finally, interpolated samples are weighted by the sampling interval \(\|\vec{r}\|\) and accumulated in the reduction variable \(a\).

For presentation purposes, only lower and upper bounds for sampling offset \(o\) and sample point count \(i_{\text{max}}\) are used in combination with an in bounds test in line 8. The latter is not essential to the algorithm though and becomes obsolete when entry and exit points of the ray with respect to the voxel volume are calculated exactly. The sampling loop does in particular not require branching depending on the driving axis \(m\).

Based on the employed interpolation kernel \(w\), the method will be referred to as GJP_{spl} (GJP with spline interpolation kernel) or GJP_{lin} (GJP with linear interpolation kernel). Linear interpolation is on graphics processing hardware usually directly provided by dedicated texture units and needs not always be explicitly implemented. In these cases, the evaluation of \(\vec{v}^{(i,1-4)}\) and corresponding interpolation weights can be omitted and the volume image may be directly evaluated at the sampling points \(\vec{p}^{(i)}\). The linear interpolating GJP_{lin} utilizing hardware provided interpolation will be termed GJP_{hwlin}.
Algorithm 1 The proposed ray tracing and sampling algorithm as illustrated in Figs. 1–5. All coordinates are expressed in units of the Cartesian grid that is ought to be sampled. The in bounds test in line 8 is not essential and becomes obsolete when \( o \) and \( i_{\text{max}} \) are calculated exactly. The sampling loop iterating over indices \( i \) is thus generally branchless and further, as it needs not be executed in any particular order, is itself parallelizable.

1: \( m \leftarrow \text{argmax}_i(|r_i|) \) \( \triangleright m \): major (driving) axis
2: \( \tilde{r} \leftarrow \frac{r}{r_m} \) \( \triangleright \tilde{r} \): sampling increment vector
3: \( o \leftarrow -s_m \) \( \triangleright o \): sampling offset, \( s \): ray source point
4: \( i_{\text{max}} \leftarrow \text{volumeDimensions}[m] \) \( \triangleright i_{\text{max}} \): maximal sampling points
5: \( a \leftarrow 0 \) \( \triangleright a \): accumulator variable
6: for \( i = 0..i_{\text{max}} \) do \( \triangleright \) iterate over sampling points
7: \( \vec{p} \leftarrow \vec{s} + (o + i) \cdot \tilde{r} \) \( \triangleright \vec{p} \): current sampling point
8: if \( \vec{p} \) is in volume then \( \triangleright \) alternatively: precise choices for \( o \) and \( i_{\text{max}} \)
9: \( \vec{p}_l \leftarrow \text{floor}(\vec{p}) \) \( \triangleright \) find lower voxel indices
10: \( \vec{p}_u \leftarrow \text{ceil}(\vec{p}) \) \( \triangleright \) find upper voxel indices
11: assert: \( p_{l,m} = p_{u,l,m} = p_{m} = i \) \( \triangleright \) by design of \( o \) and \( \tilde{r} \), \( p_{m} = i \)
12: \( \vec{w}_l \leftarrow w(\vec{p} - \vec{p}_l) \) \( \triangleright \) elementwise mapping of distances to interpolation weights (cf. Fig. 3, Eqs. 13, 14)
13: \( \vec{w}_{l,1} \leftarrow 1 - \vec{w}_l \) \( \triangleright \) calculate complementary weights
14: \( w_{l,m} \leftarrow w_{u,m} \leftarrow 1 \) \( \triangleright \) special case: driving axis
15: \( a \leftarrow a + \text{volume}[p_{l,1}, p_{l,2}, p_{l,3}] \cdot ||\tilde{r}|| \cdot \vec{w}_{l,1} \cdot \vec{w}_{l,2} \cdot \vec{w}_{l,3} \) \( \triangleright \) (cf. Eqs. 11, 15)
16: \( a \leftarrow a + \text{volume}[p_{l,1}, p_{l,2}, p_{l,3}] \cdot ||\tilde{r}|| \cdot \vec{w}_{l,1} \cdot \vec{w}_{l,2} \cdot \vec{w}_{l,3} \)
17: \( a \leftarrow a + \text{volume}[p_{l,1}, p_{l,2}, p_{l,3}] \cdot ||\tilde{r}|| \cdot \vec{w}_{l,1} \cdot \vec{w}_{l,2} \cdot \vec{w}_{l,3} \)
18: \( a \leftarrow a + \text{volume}[p_{l,1}, p_{l,2}, p_{l,3}] \cdot ||\tilde{r}|| \cdot \vec{w}_{l,1} \cdot \vec{w}_{l,2} \cdot \vec{w}_{l,3} \)
19: end if
20: end for.
2.7. Grid matching

A fundamental assumption of the proposed ray tracing (and particular sampling) method is that the voxel grid is approximately matched to the modeled beam width. In case of parallel beams, the voxel grid spacing $\Delta v$ would consequently be chosen equal to the detector pixel spacing $\Delta d$, while it becomes successively smaller with increasing cone angle. From the sketch in Fig. 6, the trigonometric relation

$$\arcsin\left( \frac{N \Delta v}{2} \right) = \arctan\left( \frac{N \Delta d}{S + D} \right)$$

between detector size $N\Delta d$ and voxel volume diameter $N\Delta v$ can be derived, resulting in an optimal choice for $\Delta v$ dependent on source and detector distances $S$ and $D$:

$$\Delta v = \Delta d \frac{S}{\sqrt{(D + S)^2 + \left(\frac{N \Delta d}{2}\right)^2}} \quad (17)$$

which reduces to $\Delta v = \Delta d \frac{S}{(D + S)}$ (linear scaling) for $D + S \gg \frac{N \Delta d}{2}$ (small cone angles).

Although for divergent beams the projected detector pixel (or beam) width $\Delta_0'$ obviously varies within the field of view, the maximal mismatch of $\Delta_0'$ and $\Delta v$ at the source- and detector-facing sides of the volume stays within 20% even for rather large cone angles of 20° (cf. Fig. 6). Only above 38.9° (equivalently: $S < 3\frac{N \Delta v}{2}$) the maximal mismatch becomes so large that the ray tracing procedure would begin to skip voxels at the detector-facing side of the volume when the grid further is in 45° orientation to the ray direction, as the projected ray spacing $\Delta_0'$ exceeds the minimal effective grid spacing of $\cos(45^\circ)\Delta v = \frac{1}{\sqrt{2}}\Delta v$ by a factor of 2 in that case.
2.8. Memory layout

Sequentially reading subsequent memory addresses classically is the optimal memory access strategy for serial algorithms. This is due to the caching behaviour of CPUs or memory controllers, which, when physically accessing memory, will fetch a whole “cache line” covering a range of memory addresses. Any subsequent memory accesses that lie within this cache line can be served efficiently. For straightforward serial raytracing this implies that efficient cache usage is only possible for a narrow range of ray orientations aligned with the memory layout, and tracing along other directions will result in very non-contiguous memory access orders. De Man and Basu addressed this issue with their Distance Driven Method [11] which serializes memory accesses by means of interleaving the tracing of adjacent rays such that subsequent iterations of the algorithm will follow the memory layout rather than a particular ray. Instead of adapting the algorithm to the memory layout, the latter may as well be made more isotropic by mapping close by spatial coordinates to close by linear memory addresses by means of e.g. Morton coding [38]. Although this will notably reduce the average distance between subsequently accessed memory addresses for arbitrary rays, it will also prohibit perfectly serial memory accesses.

For parallel GPU algorithms, the temporal memory access order of an individual thread becomes less important. Instead, requested memory addresses by simultaneous threads should be consecutive or close by. In contrast to serial programming, optimal memory access is thus achieved when rays run perpendicular to the memory layout so that each thread will step through memory in a non-contiguous fashion when regarded isolated, yet parallel (both temporal and spatial) threads handling adjacent rays will, in each step, access consecutive memory addresses that can be efficiently served. The conebeam geometry (in contrast to 2D fanbeam geometry) is highly beneficial in this respect as it features a third dimension parallel to the rotational axis which is always perpendicular to the possible driving axes occurring for circular source trajectories. The optimal memory layout is therefore, somewhat counterintuitive, contiguous across tomosgraphic slices, i.e. along the rotational axis. Parallel GJP threads will coherently step through the voxel grid along a driving axis parallel to the axial plane and access contiguous “columns” in memory.

3. Results

Both performance with respect to system modeling as well as efficiency in terms of run time and memory bandwidth of the presented algorithm are tested on an Nvidia GeForce GTX 970 graphics processor (GPU) for a 10° conebeam configuration and compared to an implementation of the popular pencil beam model based on the 3D Digital Differential Analyzer variant by Amanatides and Woo [13].

3.1. Projection and Reconstruction Quality

The performance with respect to system modeling is demonstrated on conebeam projections of the three dimensional modified Shepp Logan phan-
tom based on the definition reproduced in [39]. $10^\circ$ conebeam projections onto a $512^2$ pixel detector are generated by averaging 64 analytic line integrals per detector pixel. In total, 803 ($\approx \frac{\pi}{2} \times 512$) projections from different orientations covering the full angular range of $360^\circ$ are computed. For comparison with numeric projections, the phantom is further rendered on a $512^3$ voxel grid using 5-fold oversampling for edge anti-aliasing.

Figure 7 shows error measures based on the differences of analytic and numeric projections. The DDA (Digital Differential Analyzer) or equivalently the Siddon or pencil beam projector model exhibit most artifacts, particularly in cases where rays run roughly parallel to grid axes. In these situations the pencil beam model is equivalent to nearest neighbor sampling. When oversampling the DDA by a factor of two in each dimension, i.e. tracing four rays per detector pixel, the resulting projection quality becomes comparable to that of the non oversampled $\text{GJP}_\text{spl}$ using spline interpolation. The apparently best results are achieved by the linear interpolating $\text{GJP}_\text{lin}$ despite the presumed high frequency artifacts of the linear interpolation kernel.

As iterative reconstruction techniques as SART subsequently enforce consistency of the solution with each provided projection based on a given forward model, model errors of the discrete forward projectors will directly translate to artifacts in reconstruction results. SART is known to sufficiently converge within less than 10 iterations and Figure 8 shows respective reconstructions from the analytic projections using different forward models on a $512^3$ voxel grid. In accordance with the visual quality of the projections as shown in Fig. 7, the reconstruction quality is worst for the non-oversampling DDA, comparable for 2-fold oversampled DDA and $\text{GJP}_\text{spl}$ and best for $\text{GJP}_\text{lin}$.

### 3.2. Projection Speed

Run time performance is evaluated for projections of a cylindric volume within a cubic bounding box of $512^3$ voxels onto a $512^2$ pixel detector (cf. Fig. 9). The volume is stored in 32bit floating point format in either main- or texture memory of the graphics processing unit. As typical for computed tomography setups, projections are performed for a multitude of source and detector orientations over the full angular range of $360^\circ$ on a circular trajectory around the volume center. The rotational axis is aligned parallel to the fastest index of the memory layout. For each individual configuration of source and detector, the run time is optimized over a wide range of possible thread block or work group size parameters (CUDA and OpenCL terminology respectively). This eliminates the influence of technicalities introduced by the parallelization schemes of graphics processors. Measured execution times further exhibit a variance of up to 10% when running the same code multiple times. We report the fastest measured times for each algorithm.

Table 1 lists the so evaluated run times as averages over 360 equidistant projection angles. As a measure for GPU occupancy it further lists average memory access rates based on the total runtime and the amount of accessed voxels by each raytracing algorithm respectively. Although the latter is not
Figure 7: Approximation errors of different projection algorithms for a 10° conebeam geometry. Numeric projections of the rasterized modified Shepp Logan phantom (on a 512³ grid) onto a 512³ detector are compared to respective analytic projections. The top row shows exemplary difference images for a frontal view. Below, the difference $\ell_1$ norm normalized to the $\ell_1$ norm of the reference projection is plotted for all projection angles.

Figure 8: Axial and sagittal central slices of iterative SART reconstructions (10 iterations) from analytic projections of the modified Shepp Logan phantom using different numeric projection methods within the iterative process. The grayscale window is $[0.16, 0.32]$. Limitations of the discrete forward models (cf. Fig. 7) manifest themselves in the final reconstruction result.
Figure 9: Sketch of the benchmark configuration. A cylindrical volume of both 512 voxel diameter and height is projected onto a 512\(^2\) pixel detector. The projection cone has an opening angle of 10°. Source and detector are rotated about the center axis to acquire projections from different orientations.

| Memory    | DDA       | DDA 2×    | GJP\(_{\text{lin}}\) | GJP\(_{\text{hwlin}}\) |
|-----------|-----------|-----------|-----------------------|------------------------|
| Texture   | 4.59 ms   | 15.2 ms   | 4.97 ms               | 3.28 ms                |
|           | 118 GB/s  | 143 GB/s  | 334 GB/s              | 502 GB/s               |
| GPU RAM   | 4.39 ms   | 15.7 ms   | 5.69 ms               | —                      |
|           | 123 GB/s  | 139 GB/s  | 292 GB/s              | —                      |

Table 1: Average execution times in milliseconds and memory access rates in gigabytes per second for 2D projections of a cylindrical volume within a 512\(^3\) bounding box onto a 512\(^2\) detector in a 10° conebeam setup (cf. Fig. 9) measured on an Nvidia GTX 970 GPU. The voxel data is stored in 32bit floating point format either in texture or main GPU memory. Timings are measured for the Digital Differential Analyzer (DDA, equivalent to Siddon’s method), 2-fold oversampled DDA (i.e. \(2 \times 2\) rays per detector pixel) and the proposed Generalized Joseph Projector (GJP) using explicit (GJP\(_{\text{lin}}\)) or implicit (GJP\(_{\text{hwlin}}\)) linear interpolation.

strictly known in the case of GJP\(_{\text{hwlin}}\) due to unknown implementation details within the GPU, it is reasonably assumed to be the same as for GJP\(_{\text{lin}}\).

4. Discussion

4.1. Beam Modeling

Finite beam width modeling, as required for realistic modeling of X-ray projections, is already implied by interpolated sampling from a discrete voxel grid. Computationally costly explicit beam shape modeling by oversampling, i.e. tracing of multiple rays per detector pixel, can thus be avoided by matching the voxel grid spacing to the projected beam width within the voxel volume. Although the exactness of the implicit beam width modeling in the non-oversampling case is limited as it inherits the anisotropic nature of the Cartesian voxel grid, the quality of forward projections is visibly improved both with respect to the widespread pencil beam model and its oversampled variant. Although the intrinsic beam width modeling further does not account for the divergent beam width within the field of view, this error remains within \(\pm 10\%\) for typical cone angles of 10°. Absolutely exact ray modeling is in fact mostly not required as Hofmann et al. report [40] in agreement with our own observations.
If desired though, the presented voxel volume traversal scheme may as well be used to implement a Siddon like pencil beam model by replacing the interpolation kernel with a function evaluating the line-box intersection length (cf. [3, 17]). It can further be interpreted as an approximate radial voxel basis function and finite beam width model [26–29] with simplified nearest neighbors and footprint evaluation. The required modifications in order to implement a truly isotropic voxel basis function and beam shape model at the expense of higher computational cost have been outlined. Such an extended model may also consider the finite beam divergence, i.e. the varying beam width along the path through the voxel volume.

4.2. Projection and Reconstruction Quality

The projection quality as empirically assessed by means of comparing numeric projections of a modified Shepp Logan phantom with analytic ones indicates that GJP lin performs best among the tested techniques despite its non-bandlimited linear interpolation kernel. This is also confirmed by iterative reconstruction results. The theoretical advantage of reduced high frequency aliasing by the Hanning-like kernel used in GJP spl is apparently outweighed by its inferior ability to model gradients. Both GJP methods are however considerably better than the classic Digital Differential Analyzer or Siddon method. These findings seem to stand in contrast to [31] who conclude from both projection and reconstruction root mean square errors that slice-interpolating (Joseph-like) schemes perform similar to line-box integration (Siddon-like) schemes. To further improve over linear interpolation, true higher order interpolation methods or equivalently overlapping voxel footprints are required, in accord with [26, 31]. Either option contradicts the restriction of the sampling scheme to a 4-neighborhood though.

4.3. Memory Efficiency and Related Work

The choice of sampling points and sampling distances respectively based on the well known driving axis concept ensures that no relevant voxels will be skipped nor sampled twice. The problem is concisely formulated as parametric vector equation and a commented pseudo code implementation is given. In contrast to previous literature [2–4, 6, 15, 17, 22, 32], it does not require branching dependent on the driving axis. While the similar 3D Wu approach by Schretter for list mode backprojection [32] profited from fixed point arithmetic and caching of a precomputed weights table within the CPU, the present GJP algorithm particularly profits from the matching of voxel grid and ray spacing in combination with the alignment of memory layout and rotational axis. This results in coherent memory accesses by parallel GPU threads tracing adjacent rays and further allows to notably exceed the nominal memory bandwidth (224 GB/s) of the GPU owing to a high cache hit rate. The GJP is thus able to consider more than three times as many voxels as the DDA within the same time frame and outperforms the oversampled DDA of comparable ray modeling capability by a factor of three to five.

In contrast to the 3D generalization of Joseph’s projector discussed by [30, 31] for stacks of textured 2D planes, the present approach does not require
resampling when the driving axis changes and is also highly efficient using read-and-write memory (RAM) of the GPU. This is particularly advantageous for iterative reconstruction methods that inherently require constant read-and-write operations and do otherwise need to maintain copies of the reconstruction volume.

Other driving axis oriented (also “slice-based”) approaches \cite{9, 21, 22, 29, 34} as well as general ray-driven splatting methods \cite{10} applicable to arbitrary footprint models \cite{19, 26–28} focus on more elaborate ray modeling at the expense of an additional inner loop dynamically identifying all intersected (by some definition) voxels within each sampling plane. This is also true for the Distance Driven method \cite{11} which further interleaves looping over voxels and detector pixels and thus complicates parallelization.

Most similar to Joseph-like approaches in terms of sampling is the shear-warp technique presented in the early 1990s \cite{7, 8}, which however requires the storage of a temporary copy of the sheared volume as well as an additional resampling step (“warp”).

5. Conclusion and outlook

A both simple and efficient non-oversampling ray tracing forward projector for three dimensions based on a parametric vector formulation of the driving axis concept has been presented. Matching of detector and volume sampling ensures implicit modeling of finite beam widths while optimizing the amount of required memory accesses in the ray tracing process. The method is a generalization of Joseph’s 2D projector \cite{2}, and different interpolation kernels were discussed. An interpretation of the interpolation schemes with respect to the effective modeled voxel footprint and beam profile has been given and required modifications for the implementation of other voxel and beam shape models have been outlined. The ray modeling capabilities were demonstrated on a Shepp Logan phantom example.

The presented ray tracing and sampling algorithm is, due to its branchless design and predictable memory access pattern, particularly well suited for the SIMD architectures of modern general purpose GPUs. The scheme easily parallelizes both in a single as well as in a multiple threads per ray configuration. Compared to the classic Digital Differential Analyzer (DDA) algorithm, a three to five times higher memory access rate is achieved on an Nvidia GTX 970 GPU, allowing to outperform the DDA also in terms of total run time despite the higher total amount of memory accesses.

In the context of computed tomography, improved ray tracing performance directly translates to shorter image reconstruction times for most iterative tomographic reconstruction techniques as the forward modeling of X-ray projections often constitutes the largest time factor of such algorithms. Its high efficiency also when operating on read-and-write memory eliminates the need to constantly hold and update copies of the reconstruction volume.
6. Acknowledgements

The author is supported by Fraunhofer IIS and further thanks the anonymous reviewers for their valuable feedback.

References

References

[1] A. H. Andersen, A. C. Kak: Simultaneous Algebraic Reconstruction Technique (SART): A Superior Implementation of the ART Algorithm. Ultrasonic Imaging 6, 81–94 (1984)

[2] P. M. Joseph: An Improved Algorithms for Reprojecting Rays Through Pixel Images. IEEE Transactions on Medical Imaging MI-1 (3) 192–196 (1982)

[3] S. B. Lo: Strip and Line Path Integrals with a Square Pixel Matrix: A Unified Theory for Computational CT Projections. IEEE Transactions on Medical Imaging 7 (4) 355–363 (1988)

[4] A. Fujimoto, T. Tanaka and K. Iwata: ARTS: Accelerated Ray-Tracing System. Computer Graphics and Applications, IEEE 6 (4) 16–26 (1986)

[5] R. Endl, M. Sommer: Classification of Ray-Generators in Uniform Subdivisions and Octrees for Ray Tracing. Computer Graphics Forum 13 (1) 3–19 (1994)

[6] Y. K. Liu, B. Žalik: A General Multi-step Algorithm for Voxel Traversing Along a Line. Computer Graphics Forum 27 (1) 73–80 (2008)

[7] G. G. Cameron, P. E. Undrill: Rendering volumetric medical image data on a SIMD-architecture computer. Proceedings of the Third Eurographcis Workshop on Rendering, 135–145, Bristol, UK, May 1992

[8] P. Lacroute, M. Levoy: Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. SIGGRAPH 1994, Orlando, Florida, USA

[9] S. Matej, R. M. Lewitt: Practical Considerations for 3-D Image Reconstruction Using Spherically Symmetric Volume Elements. IEEE Transactions on Medical Imaging 15 (1) 68–78 (1996)

[10] K. Mueller, R. Yagel: Fast Perspective Volume Rendering with Splatting by Utilizing a Ray-Driven Approach. Proceedings of the 7th Conference on Visualization ’96, 65–72 (1996)

[11] B. De Man, S. Basu: Distance-driven projection and backprojection in three dimensions. Physics in Medicine and Biology 49 (11) 2463–2475 (2004)
[12] R. Siddon: *Fast calculation of the exact radiological path for a three-dimensional CT array*. Medical Physics 12 (2) 252–255 (1985)

[13] J. Amanatides, A. Woo: *A Fast Voxel Traversal Algorithm for Ray Tracing*. Eurographics 87 (3) 10 (1987)

[14] F. Jacobs, E. Sundermann, B. De Sutter, M. Christiaens, I. Lemahieu: *A fast algorithm to calculate the exact radiological path through a pixel or voxel space*. Journal of computing and information technology 6 (1) 89–94 (1998)

[15] H. Zhao, A. J. Reader: *Fast Ray-Tracing Technique to Calculate Line Integral Paths in Voxel Arrays*. Nuclear Science Symposium Conference Record, 2003 IEEE 4 2808–2812 (2003)

[16] M. de Greef, J. Crezee, J. C. van Eijk, R. Pool, A. Bel: *Accelerated ray tracing for radiotherapy dose calculations on a GPU*. Medical Physics 36 (9) 4095–4102 (2009)

[17] H. Gao: *Fast parallel algorithms for the x-ray transform and its adjoint*. Medical Physics 39 (11) 7110–7120 (2012)

[18] W. Yao, K.: *Analytically derived weighting factors for transmission tomography cone beam projections*. Physics in Medicine and Biology 54 (3) 513–533 (2009)

[19] Y. Long, J. A. Fessler, J. M. Balter: *3D forward and back-projection for x-ray CT using separable footprints*. IEEE Transactions on Medical Imaging 29 (11) 1839–1850 (2010)

[20] M. Wu, J. Fessler: *GPU acceleration of 3D forward and backward projection using separable footprints for X-ray CT image reconstruction*. Proceedings of the International Meeting on Fully 3D Image Reconstruction in Radiology and Nuclear Medicine 6, 021911 (2011)

[21] V. Ngyen, S. Lee: *Graphics processing unit-accelerated iterative tomographic reconstruction with strip-integral system model*. Optical Engineering 51 (9) 093203–1–11 (2012)

[22] S. Zhang, D. Zhang, H. Gong, O. Ghahsinalizadeh, G. Wang, G. Cao: *Fast and accurate computation of system matrix for area integral model-based algebraic reconstruction technique*. Optical Engineering 53 (11) 113101–1–9 (2014)

[23] S. Ha, A. Kumar, K. Mueller: *A Study of Volume Integration Models for Iterative Cone-Beam Computed Tomography*. Proceedings of the 13th Meeting on Fully 3D Image Reconstruction (2015)

[24] S. Ha, H. Li, K. Mueller: *Efficient area-based ray integration using summed area tables and regression models*. Proceedings of the 4th International Meeting on image formation in X-ray CT, 507–510 (2016)
[25] R. Sampson, M. G. McGaffin, T. F. Wenisch, J. A. Fessler: *Investigating multi-threaded SIMD for helical CT reconstruction on a CPU*. Proceedings of the 4th International Meeting on image formation in X-ray CT, 275–278 (2016)

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

[27] R. M. Lewitt: *Alternatives to voxels for image representation in iterative reconstruction algorithms*. Physics in Medicine and Biology 37 (3) 705–716 (1992)

[28] A. Ziegler, T. Köhler, T. Nielsen, R. Proksa: *Efficient projection and backprojection scheme for spherically symmetric basis functions in divergent beam geometry*. Medical Physics 33 (12) 4653–4663 (2006)

[29] J. Sunnegårdh, P. Danielsson: *A New Anti-Aliased Projection Operator for Iterative CT Reconstruction*. Proceedings of the Ninth International Meeting on Fully Three-dimensional Image Reconstruction in Radiology and Nuclear Medicine, Lindau, Germany, July 9–13, 2007

[30] F. Xu, K. Mueller: *Accelerating Popular Tomographic Reconstruction Algorithms on Commodity PC Graphics Hardware*. IEEE Transactions on Nuclear Science 52 (3) 654–663 (2005)

[31] F. Xu, K. Mueller: *A Comparative Study of Popular Interpolation and Integration Methods for use in Computed Tomography*. 3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro, 1252–1255 (2006)

[32] C. Schretter: *A Fast Tube-of-Response Raytracer*. Medical Physics 33 (12) 4744–4748 (2006)

[33] S. Matej, J. A. Fessler, I. G. Kazantsev: *Iterative Tomographic Image Reconstruction Using Fourier-Based Forward and Back-Projectors*. IEEE Transactions on Medical Imaging 23 (4) 401–412 (2004)

[34] J. Brokish, D. B. Keesing, Y. Bresler: *Iterative circular conebeam CT reconstruction using fast hierarchical backprojection/reprojection operators*. SPIE Medical Imaging. International Society for Optics and Photonics, 2010.

[35] K. Mueller, F. Xu, N. Neophytou: *Why do Commodity Graphics Hardware Boards (GPUs) work so well for acceleration of Computed Tomography?*. SPIE Electronic Imaging 2007

[36] G. Pratx, L. Xing: *GPU computing in medical physics: A review*. Medical Physics 38 (5) 2685–2697 (2011)

[37] H. B. Kekre, S. C. Sahasrabudhe, N. C. Goyal: *Raised Cosine Function for Image Data Interpolation*. Computers & Electrical Engineering 9 (3-4) 131–152 (1982)
[38] G. M. Morton: *A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing*. IBM Technical Report, https://domino.research.ibm.com/library/cyberdig.nsf/0/0dabf9473b9c86d48525779800566a39 (1966)

[39] M. Schabel: *3D Shepp-Logan phantom*. MATLAB script, http://de.mathworks.com/matlabcentral/fileexchange/9416-3d-shepp-logan-phantom/content/phantom3d.m (2005–2006)

[40] C. Hofmann, M. Knaup, M. Kachelrieß: *Effects of ray profile modeling on resolution recovery in clinical CT*. Medical Physics 41 (2) 021907-1–14 (2014)