A look-up table-based ray integration framework for 2D/3D forward and back-projection in X-ray CT

Sungsoo Ha and Klaus Mueller, *Senior Member, IEEE*

***Abstract*—Iterative algorithms have become increasingly popular in Computed Tomography (CT) image reconstruction since they better deal with the adverse image artifacts arising from low radiation dose image acquisition. But iterative methods remain computationally expensive. The main cost emerges in the projection and backprojection operations where accurate CT system modeling can greatly improve the quality of the reconstructed image. We present a framework that improves upon one particular aspect – the accurate projection of the image basis functions. It differs from current methods in that it substitutes the high computational complexity associated with accurate voxel projection by a small number of memory operations. Coefficients are computed in advance and stored in look-up tables parameterized by the CT system's projection geometry. The look-up tables only require a few kilobytes of storage and can be efficiently accelerated on the GPU. We demonstrate our framework with both numerical and clinical experiments and compare its performance with the current state of the art scheme – the separable footprint method.**

***Index Terms*—iterative CT reconstruction, separable footprint, look-up table, LUT, NVIDIA GPU, CUDA**

I. Introduction

With the increasing popularity of iterative reconstruction methods in medical computed tomography (CT), modeling a realistic CT system in software is more crucial than ever. The CT system model can be represented by a matrix where each element indicates the contribution of a voxel to an X-ray path. The process of evaluating the attenuation of X-ray according to the properties of the material while traveling across a discretized object is known as *forward-projection*, while its reverse model, generally defined as the transpose of forward projection, is known as *back-projection*. Since the size of the CT system matrix is enormous, the elements are typically computed on-the-fly during forward- and back-projection. This, however, comes with high computational cost making these operations the computational bottleneck of iterative CT reconstruction methods.

The most intuitive and simplest way to model the CT system is via line integration [1], [2]. In this model, an X-ray path is depicted by a zero width line and the contribution of a voxel to the ray integral is approximated by the line intersection length. While this popular scheme has low computational complexity it suffers from under-sampling and aliasing [3], [4].

A more accurate approach is volume integration (area integration for fan-beam). While volume (or area) integration does not take into account the exponential edge gradient effect [5], it affords a much closer model of a real CT system than line integration. In the volume (area) integration model a ray path is depicted by a 3D polyhedron (2D polygon) that connects the X-ray source with the four edges (two corners) of a detector bin. The contribution of a voxel to the path is then the intersection volume (area). Yu et al. [4] demonstrated the potential of this model for improving spatial resolution and suppressing high-frequency artifacts, while Zhang et al. [6] proposed an efficient algorithm which computes the intersection area by categorizing the ray-voxel intersections into six cases that can be calculated in a simple algebraic fashion. Their scheme, however, was only described for fan-beam geometry, and it is unclear how one could support 3D geometries such as cone-beam or helical scans without losing the prescribed computational efficiency.

At the expense of accuracy there are also two popular projection-space volume integration-based approaches. The first is the distance-driven (DD) method [3] which maps the boundaries of a voxel and a detector bin to a common axis and approximates the intersection volume by the product of a rectangular shaped footprint and an amplitude term. The second approach is the separable footprint (SF) method [7] which approximates the voxel footprint by a 2D separable function. The SF has been shown to be more accurate than the DD method, while keeping a similar time performance.

Our own recent work [8] introduced various voxel subdivision schemes that approximate the voxel volume integration at arbitrary levels of precision directly in image space. These schemes leverage GPU-acceleration [9], [10] to reduce their high computational cost, but the speed is still too slow for clinical use. More recently, we introduced a method [11] for fan-beam geometries that pre-computes sampled intersection volumes and stores them in a look-up table such that unknown samples can be mapped into the table and then calculated via a bilinear interpolation scheme. This look-up table-based ray integration (LTRI) method showed very promising results both in terms of accuracy and speed, but since it was only defined for fan-beam geometry its practical use was limited.

Look-up tables are fairly popular and have been utilized in many areas to save computation time. For example, Hensley et al. [12] devised a GPU-accelerated method that quickly generates a summed-area table and employs it for real-time volume rendering. Specifically for CT reconstruction, Entezari et al. [13] utilized a lookup table to store the ray (but not volume) integrals of a box-spline basis function. Ziegler et al. [14] used lookup tables for radially-symmetric blob basis functions. While their method models a blob's volumetric contribution it approximates the locally diverging beam by a piecewise

Sungsoo Ha and Klaus Mueller are with the Visual Analytics and Imaging Lab at the Computer Science Department at Stony Brook University, Stony Brook, NY 11794-4400 USA (e-mail: sunha, mueller@cs.sunysb.edu).

parallel beam. Our scheme makes no such approximations and extends our former LTRI method defined for fan-beam to also support 3D cone-beam geometries.

Our paper is structured as follows. Section II summarizes the original fan-beam LTRI method, while Section III presents our new contribution – extending the 2D LTRI scheme to cone-beam geometries. Section IV presents our CUDA implementation [15] of the forward- and back-projection operators. [3], [7], [8]. Section V presents numerical experiments geared to (1) find the optimal table resolutions and (2) compare the 3D LTRI with the SF method [7]. Finally, Section VI concludes this paper with a discussion and future work.

II. Fan-beam projection

For simplicity of presentation we choose to describe our look-up table-based ray integration (LTRI) scheme by ways of the flat detector axial fan-beam geometry. Our method, however, readily extends to arc detectors as well as other 3D CT geometries, such as helical CT. Also, in preparation for our subsequent discussion of the 3D cone-beam case (see Section III) we shall use the term *voxel* to also refer to the 2D elements that make up the grid in the fan-beam case.

Given an X-ray point source and detector on a circular trajectory, the i-th projection at view θ (p_i^θ) corresponds to the area integral over the fan-beam segment defined by connecting the X-ray source ($\mathbf{v}_{src}$) with the two corners of the i-th detector bin. This so called source-bin triangle (SBT) can be formalized as follows [4] (see illustration in Fig. 1):

$$p_i^\theta = \frac{1}{\gamma_{\varphi,i}} \sum_{\mathbf{n}\in\Omega_{\text{SBT}}} f[\mathbf{n}] \frac{d\mathbf{n}}{||\mathbf{v_n} - \mathbf{v}_{\text{src}}||} , \qquad (1)$$

where $f[\mathbf{n}]$ is the value of a voxel $\mathbf{n} = (n_x, n_y) \in \mathbb{Z}^2$ located on a 2D grid of size $N_x \times N_y$ with spacing $\Delta_x \times \Delta_y$. The sum runs over all voxels that at least partially fall within the SBT region (Ω_{SBT}), weighted by the ratio of the ray-voxel intersection area ($d\mathbf{n}$) and the ray-arc length ($\gamma_{\varphi,i} \cdot ||\mathbf{v_n} - \mathbf{v}_{\text{src}}||$). The latter takes into account the geometric spreading effect where voxels closer to the X-ray source cast larger shadows on the detector than voxels closer to the detector. The term $\mathbf{v_n}$ is the location of the center of the $\mathbf{n}$-th voxel, and $\gamma_{\varphi,i}$ is the angle of the i-th fan-beam segment. Note that φ is used to denote the azimuthal angle. We assume $\Delta_x = \Delta_y$ which is the typical configuration in CT reconstruction. In the remainder of this discussion, our main interest focuses on the computation of the ray-voxel intersection area ($d\mathbf{n}$), which is the most compute-intensive part of (1).

A. Exact ray-voxel area

In fan-beam CT geometry, the term $d\mathbf{n}$ in (1) is the intersection *area* of a voxel and an SBT. This intersection area is a 2-D convex polygon and can be computed as follows [16] (see Fig. 2 for an illustration):

$$\text{area} = \frac{1}{2} \left| \sum_{i=1}^{K} \left(\mathbf{v}_i^x \mathbf{v}_{i+1}^y - \mathbf{v}_{i+1}^x \mathbf{v}_i^y \right) \right| , \qquad (2)$$

where $\mathbf{v}_i$ is a vertex of the polygon and there are $K(\geq 3)$ vertices. Although it is possible to compute the area precisely

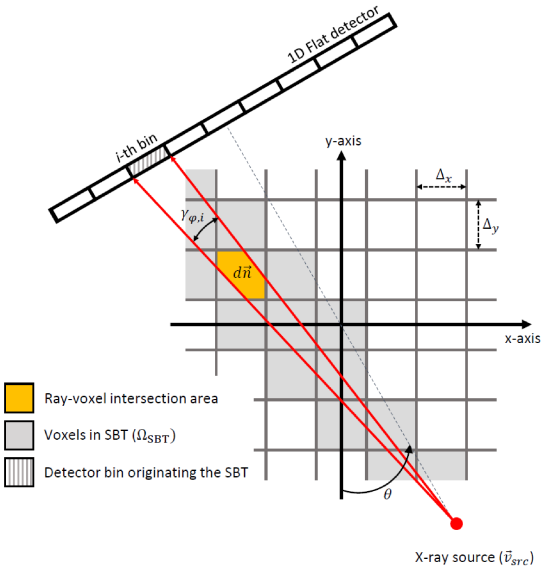

Fig. 1. Ray-voxel intersect area in fan-beam geometry.

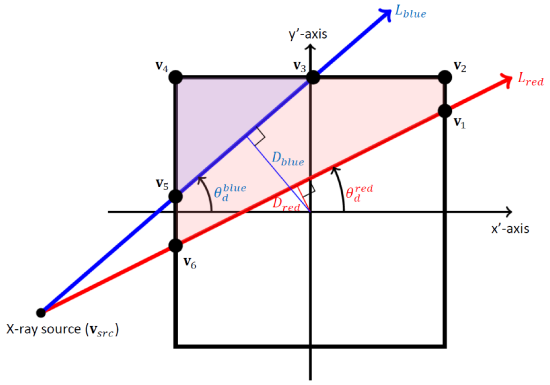

Fig. 2. Ray-voxel intersection area. The intersection area can be computed with two memory operations using the area look-up table such that $aLUT(D_{\text{red}}, \theta_d^{\text{red}}) - aLUT(D_{\text{blue}}, \theta_d^{\text{blue}})$.

using (2), this is computationally very expensive because it requires to first determine all vertices of the polygon, if any, and then sort them in a predefined direction before the summation can be applied. For example, to compute the intersection area in Fig. 2, the five vertices, $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_5, \mathbf{v}_6\}$ need to be first identified by line intersection calculations and then sorted in clock-wise or counter clock-wise order to execute (2). Sequential computations of this sort are not very amenable to parallel processing and so there is not much opportunity for GPU acceleration. In the next section, we explain how this long sequential process can be replaced by a simple memory operation, using a look-up table.

B. Area look-up table

The area look-up table ($aLUT$) we construct returns the intersection area across a voxel defined by the half plane bounded by the *directed* line that represents a ray connecting the X-ray source to one corner of a detector bin in a predefined direction. The *signed* perpendicular distance, D_{line}, from the

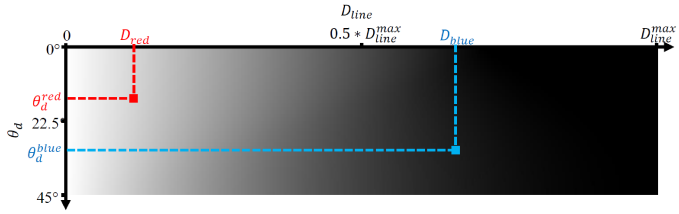

Fig. 3. The area look-up table with a 0.415 x 0.415 mm^2 sized voxel. Using symmetry properties, it covers the ranges of $\theta_d \in [0°, 45°]$ and $D_{\text{line}} \in [0, D^t_{\text{line}}]$ where $D^t_{\text{line}} \approx 0.3$ mm. The $(D_{\text{red}}, \theta^{\text{red}}_d)$ and $(D_{\text{blue}}, \theta^{\text{blue}}_d)$ correspond to the ray-voxel intersection areas shown in Fig. 2.

center of the voxel to this line is assigned to the first dimension of the $aLUT$. The second dimension is the *directed* angle, θ_d, measured from a reference axis to the head of the line. We used the x-axis as the reference axis in our work. Given a pair $(D_{\text{line}}, \theta_d)$, the corresponding intersection area is computed by following (2) and storing the result in the table. The intersection area between a voxel and a fan-beam, which is represented by using two lines, can then be efficiently computed by the difference of two such areas, fetched from the table as illustrated in Fig. 2 and Fig. 3.

With N_s being the size of a 1D flat detector, there are $(N_s + 1)$ lines that connect a corner of a detector bin with the X-ray source for a projection view, θ. For example, in Fig. 1, there are 10 lines. Each line can be represented by the directed angle (θ_d) and the normalized general form of a 1D line equation, calculated as follows:

$$\begin{aligned} \theta_d &= \arctan(A/B) \\ 0 &= (A/Z)x + (B/Z)y + (C/Z) \ , \end{aligned} \tag{3}$$

where

$$\begin{aligned} A = \mathbf{v}^y_{\text{src}} - \mathbf{v}^y_i \ , \qquad B = \mathbf{v}^x_i - \mathbf{v}^x_{\text{src}} \\ C = \mathbf{v}^x_{\text{src}}\mathbf{v}^y_i - \mathbf{v}^x_i\mathbf{v}^y_{\text{src}} \ , \quad Z = \sqrt{A^2 + B^2} \ . \end{aligned} \tag{4}$$

Here, $\mathbf{v}_{src}$ is the 2D position of the X-ray source and $\mathbf{v}_i$ is the 2D position of a detector bin corner. Since the detector and source are fixed for a given view, the coefficients A, B, C, Z, and θ_d can be pre-computed for each bin corner and stored in a temporary (for the given view only) 1D table with $O(N_s + 1)$ space complexity. In fact, we divide A, B, and C by Z and only store these normalized coefficients, along with θ_d. The table is then indexed by the line's detector bin corner index. The computational cost required for this is typically negligible in the context of the overall projection operations.

Using the A, B, C coefficients stored in this view-based table, we can compute the perpendicular distance for a given line with respect to a voxel centered at (x_0, y_0) as follows:

$$D^{(x_0,y_0)}_{line} = Ax_0 + By_0 + C \ , \tag{5}$$

which is paired with the stored θ_d coefficient to form the 2D index for the general $aLUT$ table $\left(D^{(x_0,y_0)}_{\text{line}}, \theta_d\right)$. This yields the one-sided intersection area between the voxel centered at (x_0, y_0) and a ray (or line). Bi-linear interpolation is employed for table look-ups with non-integer queries.

TABLE I
ANGULAR SYMMETRY OF $aLUT$ AND $hLUT$

θ_* [a]	θ_{sym} [b]
$45° < \theta_* \leq 135°$	$\lvert 90° - \theta_* \rvert$
$135° < \theta_* \leq 225°$	$\lvert 180° - \theta_* \rvert$
$225° < \theta_* \leq 315°$	$\lvert 270° - \theta_* \rvert$
$315° < \theta_* \leq 360°$	$\lvert 360° - \theta_* \rvert$

[a] θ_d for $aLUT$ and θ_φ for $hLUT$.
[b] angle ($\in [0°, 45°]$) that returns the same value in a LUT.

Making use of symmetry properties derived from the square shape of a voxel ($\Delta_x = \Delta_y$), the coverage of D_{line} is $[0, D^t_{\text{line}}]$ and for θ_d it is $[0°, 45°]$. For negative distances, the corresponding area is equal to $S - aLUT(|D_{\text{line}}|, \theta_d)$ where S is the square area ($\Delta_x \times \Delta_y$). For directed angles larger than 45°, we can utilize the angular symmetric properties described in TABLE I. The threshold distance, D^t_{line}, then is:

$$D^t_{\text{line}} = 0.5 \cdot \sqrt{\Delta^2_x + \Delta^2_y} \ , \tag{6}$$

This is a voxel's half diagonal since the one-sided intersection area converges either to zero or full area beyond the length.

III. CONE-BEAM PROJECTION

We shall now extend the look-up table-based ray integration (LTRI) method from the 2D to the 3D reconstruction setting. For ease of illustration, we assume a flat detector in an axial cone-beam geometry, but our framework is not restricted to that configuration. Let $f[\mathbf{n}]$ be the value of a voxel $\mathbf{n} = (n_x, n_y, n_z) \in \mathbb{Z}^3$ located in a 3D grid of size $N_x \times N_y \times N_z$ with spacing $\Delta_x \times \Delta_y \times \Delta_z$. Without loss of generality, we assume $\Delta_x = \Delta_y$ for the fan-beam case. For the added axial dimension we keep it more general, that is, $\Delta_x \neq \Delta_z$, – a voxel does not need to be a cube. This often arises in practical situations where the inter-slice distance can be different from the in-slice resolution. Further, let (s, t) denote the local coordinates on the $N_s \times N_t$ sized 2D flat detector plane where the s-axis is perpendicular to the z-axis of the volume while the t-axis is parallel to it. The size of each detector bin is $\Delta_s \times \Delta_t$. In this study, we consider CT geometries that have the rotation axis aligned with the z-axis.

The cone-beam emerging from the i-th detector bin can be drawn as a pyramid-like shape by connecting the four edges of the bin with the X-ray point source ($\mathbf{v}_{\text{src}}$). We shall denote the volume subtended by the cone-beam the source-bin pyramid (SBP) and define Ω_{SBP} to embrace all voxels intersecting with the SBP. The volume integral acquired from projection view θ is then expressed as follows (see Fig. 4 for an illustration):

$$p^\theta_i \approx \frac{1}{|\sin\alpha_i| \cdot \gamma_{\phi,i}\gamma_{\varphi,i}} \sum_{\mathbf{n}\in\Omega_{\text{SBP}}} f[\mathbf{n}] \frac{d\mathbf{n}}{||\mathbf{v}_\mathbf{n} - \mathbf{v}_{\text{src}}||^2} \ , \tag{7}$$

where the sum is over all voxels in Ω_{SBP}. The value of each voxel, $f[\mathbf{n}]$, is weighted by the ratio of the ray-voxel intersection volume ($d\mathbf{n}$) in the nominator and the squared beam divergence term in the denominator. This squared divergence term follows the inverse square law and accounts for the geometric spreading already mentioned for the 2D case. Note

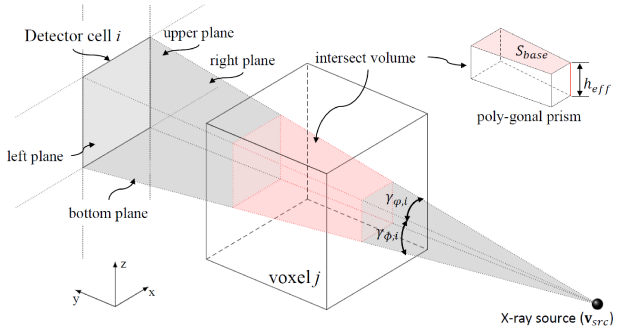

Fig. 4. Ray-voxel intersection volume in cone-beam geometry. The SBP (source-bin pyramid) is shaded gray, the intersection volume is shaded red.

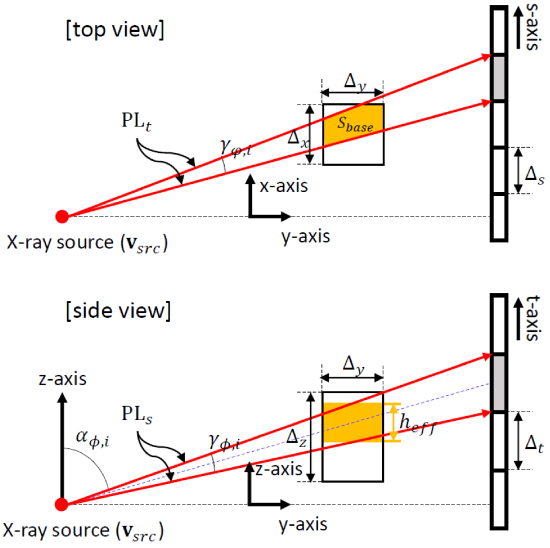

Fig. 5. Computing the intersection volume by the product of S_{base} and h_{eff}.

that $\mathbf{v_n}$ is the center position of a voxel indexed by $\mathbf{n}$. The sum is multiplied with a SBP dependent constant which includes the geometric angular dispersion of the beam ($|\sin\alpha_i|$) and the two cone angles, $\gamma_{\varphi,i}$ and $\gamma_{\phi,i}$ in which φ and ϕ represent the azimuthal and the polar angles, respectively.

The value of $|\sin\alpha_i|$ is approximated by decomposing α_i into azimuthal ($\alpha_{\varphi,i}$) and polar ($\alpha_{\phi,i}$) angles by connecting the X-ray source to the center of a detector bin assuming a narrow Ω_{SBP}. Then, $|\sin\alpha_i|$ is computed by the multiplication of the two components as:

$$|\sin\alpha_i| \approx |\sin\alpha_{\phi,i}| \cdot \max(|\cos\alpha_{\varphi,i}|, |\sin\alpha_{\varphi,i}|) \ . \quad (8)$$

Note that the angular dispersion term in (8) is view independent and only requires a one-time computation that can be used for both forward and back-projections in multiple views. The derivation of (7) is in the supplement materials.

A. Exact ray-voxel volume

In cone-beam geometry, the intersection between a cone-beam and a voxel becomes an arbitrary shaped 3D convex polyhedron as shown in Fig. 4. Unlike the 2D case, there is no direct closed form solution to compute this volume. The best we can do is decomposing the 3D polyhedron into a number of tetrahedra using a 3D Delaunay triangulation approach [17]. We can specify a tetrahedron by three edge vectors $\mathbf{a}_i$, $\mathbf{b}_i$, and $\mathbf{c}_i$ emerging from a given polyhedron vertex. The intersection volume is then computed by summing over all K tetrahedra:

$$\text{Volume} = \sum_{i=1}^{K} \frac{1}{3!} |\mathbf{a}_i \cdot (\mathbf{b}_i \times \mathbf{c}_i)| \ . \quad (9)$$

As in the fan-beam case, the complexity for an exact computation of the intersection volume of a voxel with a cone-beam, now by way of tetrahedralization, is very high and it is also difficult to parallelize. In the following subsection, we will return to the area look-up table introduced for the fan-beam case and introduce another look-up table to cover the third (height) dimension to arrive at a look-up table approach for 3D polyhedral beam-voxel intersections.

B. Height look-up table

A *prism* is a polyhedron that consists of an n-sided *convex* polygonal base, a second congruent parallel base, and n other faces orthogonally joining the corresponding sides of the two bases. The volume of a prism is easily computed by the product of the *base area* and the *height*. Our key idea is to find the prism that has a volume equivalent to the one generated by the intersection of a voxel and the SBP, as shown in Fig. 4. We call this prism V_{poly}.

The SBP consists of four planes. Two of these are drawn using the detector bin edges parallel to the t-axis, called PL_t. The others originate from the bin edges parallel to the s-axis, called PL_s. Intersected by the two PL_t, a voxel in Ω_{SBP} becomes a polygonal prism with base area S_{base} and height Δ_z (see Fig. 5). The base area of the prism is also the base of V_{poly}. More importantly, in a CT geometry in which the rotation axis is aligned with the z-axis, this area can be regarded the *shadow* that the cone-beam leaves on the xy-plane. This shadow is identical to the area left by a corresponding fan-beam intersection and is efficiently computed using the area look-up table described in Section II-B.

To obtain the height of V_{poly}, we consider the two PL_s, call them $PL_{s,bot}$ and $PL_{s,top}$. We compute two intersection volumes, one for each of these two planes, in a pre-defined direction by following the exact approach from Section III-A, call these two intersection volumes $V_{PL_{s,bot}}$ and $V_{PL_{s,top}}$. Suppose now a rectangular prism with voxel base $\Delta_x \times \Delta_y$ and volume $V_{PL_{s,top}}$. Its height can be computed as follows:

$$h_{PL_{s,top}} = V_{PL_{s,top}} / (\Delta_x \times \Delta_y) \ . \quad (10)$$

The height for $V_{PL_{s,bot}}$ is computed similarly. The difference of the two heights derived from both of the PL_s is the height effective for the V_{poly}, called h_{eff}. The intersection volume between a voxel and a SBP can then be computed by the product of S_{base} and h_{eff}. Fig. 5 [side view] shows an illustration of this process.

To efficiently compute the h_{PL_s} in (10), we construct a look-up table that returns the height for a given plane, called

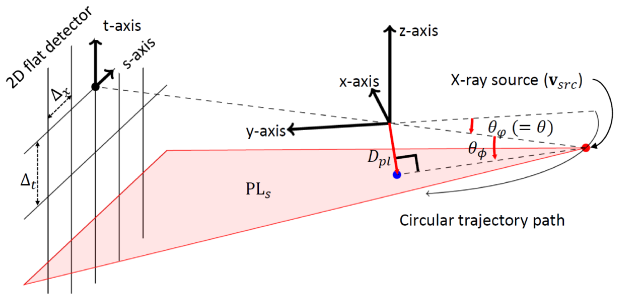

Fig. 6. Parameterization of PL_s in 2D flat detector axial cone-beam CT.

the height look-up table (*hLUT*). For this, we first represent the respective plane, V_{PL_s}, with three parameters; the first parameter is the *signed* perpendicular distance from the origin to the plane (D_{pl}), the second is the *signed* angle from the xy-plane to the plane (θ_ϕ), and the third is the rotation angle about the z-axis (θ_φ), which is equal to a projection angle, θ. Fig. 6 depicts these parameters in the context of a 2D flat detector axial cone-beam X-ray CT system. A normalized plane equation, $Ax + By + Cz + D = 0$ where $A^2 + B^2 + C^2 = 1$, can be derived using these parameters:

$$\begin{aligned}[A, B, C]^T &= R_z(\theta_\varphi)R_x(\theta_\phi)[0, 0, 1]^T \\ D &= D_{pl} \ ,\end{aligned} \tag{11}$$

where $R_{axis}(\cdot)$ is a 3×3 rotation matrix about the given $axis$ and T is the transpose operator. Then, each element of the $hLUT$ indexed by (D_{pl}, θ_ϕ, θ_φ) is computed as in (10).

In practice, similar to the fan-beam case, we pre-compute the coefficients A, B, C, D according to (11) and store them in a 1D view-based look-up table for fast implementation. This incurs a (small) space complexity of $O(N_t + 1)$ since there are $N_t + 1$ PL_s in a projection view, θ.

Any new h_{PL_s}between a plane, PL_s, and a voxel centered at (x_0, y_0, z_0) is then simply fetched from the $hLUT$ at the location of $(D^v_{pl}, \theta_\phi, \theta_\varphi)$ where

$$D^v_{pl} = Ax_0 + By_0 + Cz_0 + D \ . \tag{12}$$

This equation computes the signed perpendicular distance from the voxel center to the plane, PL_s. We obtain both $h_{PL_{s,top}}$and $h_{PL_{s,bot}}$ in this way to finally compute h_{eff}.

Using symmetry properties in the voxel-plane intersection, the range of plane parameters, (D_{pl},θ_ϕ,θ_φ), is effectively reduced. With fixed θ_ϕ and θ_φ, $hLUT(-D_{pl}, \cdot, \cdot)$ is equal to $\Delta_z - hLUT(+D_{pl}, \cdot, \cdot)$. Similarly, with fixed D_{pl} and θ_φ, $hLUT(\cdot, -\theta_\phi, \cdot)$ is equal to $\Delta_z - hLUT(\cdot, +\theta_\phi, \cdot)$. For the θ_φ, the symmetric property described in Table I is applied. In addition, with fixed θ_ϕ and θ_φ, any D_{pl} that is larger than a threshold distance converges to either zero or Δ_z regardless of θ_ϕ and θ_φ. The threshold distance is calculated as follows:

$$D^t_{pl} = \frac{1}{2} \cdot \sqrt{\Delta_x^2 + \Delta_y^2 + \Delta_z^2} \ , \tag{13}$$

which is half of the diagonal length of a voxel. Finally, the θ_{pol}, is limited by the polar angle of the given cone-beam CT geometry such that

$$\theta^{max}_\phi = \arctan((0.5 \times N_t \times \Delta_t)/SDD) \ , \tag{14}$$

where SDD is the distance from the X-ray source to the detector. Thus, the parameter space we need to cover to construct $hLUT$ with a given cone-beam CT system becomes $D_{pl} \in [0, D^t_{pl}]$, $\theta_\phi \in [0, \theta^{max}_\phi]$, and $\theta_\varphi \in [0, 45°]$.

C. Height approximation methods

While the above accurate scheme is fairly efficient, we have found that there are at least two approximations that can bring further speed-ups without significant losses in accuracy.

1) Regression method: Fig. 7 shows profiles of both the area and the height look-up tables for several fixed angular dimensions, i.e. θ_d, θ_φ, and θ_ϕ, for a $0.415 \times 0.415 \times 0.83\text{mm}^3$ voxel size. For the height look-up table in Fig. 7(b), θ^{max}_ϕ is 10°. We only present the case where $\theta_\varphi = 0$ for simplicity as we found that θ_φ does not incur much variation. We observe that especially for the height look-up table, the non-linearity along the distance dimension is not severe and hence an approximation by a piece-wise regression function ($\theta_\varphi = \theta_\phi = 0$) is reasonable:

$$\text{height} = \begin{cases} \Delta_z/2 - D_{pl}, & \text{if } 0 \le D_{pl} \le \Delta_z/2 \\ 0, & \text{otherwise} \ . \end{cases} \tag{15}$$

The constant $\Delta_z/2$ is pre-computed for fast execution.

2) Distance method: The roughly linear behavior of the height look-up table also inspires another approximation to gauge the effective height between a detector bin and a voxel. As illustrated in Fig. 8, we can use the overlapped distance on a common z-axis. The common z-axis is the axis passing through the center of the voxel onto which the two corner points of the detector bin (located on the detector t-axis) are projected. The effective height is then computed as follow:

$$h_{\text{eff}} = \begin{cases} \min(z_+, t^c_+) - \max(z_-, t^c_-), & \text{if overlap} \\ 0, & \text{otherwise} \ . \end{cases} \tag{16}$$

Here, z_+ and z_- are two corner points of a voxel on the common z-axis, computed as $z + \Delta_z/2$ and $z - \Delta_z/2$, respectively. Correspondingly, the two corner points, t_+ and t_-, of a detector bin on the t-axis are computed as $t + \Delta_t$ and $t - \Delta_t$ and are projected onto the common z-axis to yield t^c_+ and t^c_-. Although this concept is derived from a different perspective, it gives rise to a similar mathematical expression than the ones underlying the distance-driven method [3] or used for the rectangle function of the separable footprint [7].

IV. GPU Implementation using CUDA

We implemented our LTRI-based framework with NVIDIA CUDA 7.5 [15]. Algorithm 1 presents the program flow of our approach for a 3D forward projection into a single flat-panel view. Modifying it to multiple views or different viewing geometries would not conceptually change this program flow. Our parallel CUDA implementation slightly reorders and arranges the statements into three separate kernels to take advantage of certain geometric and hardware constellations.

The first CUDA kernel (line 2-4) initializes the projection arrays and fills the various 1D lookup tables that hold the coefficients needed to compute the view-dependent indices for

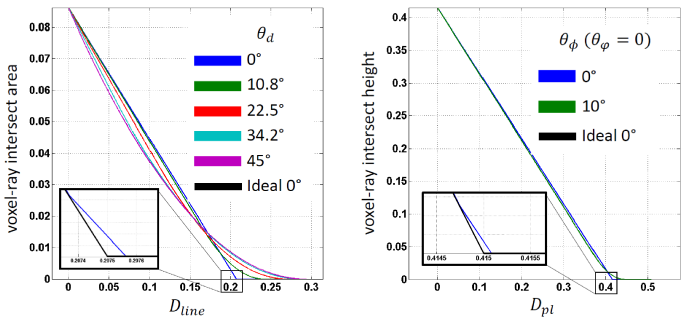

Fig. 7. Non-linearity of the (a) area and (b) height LUT.

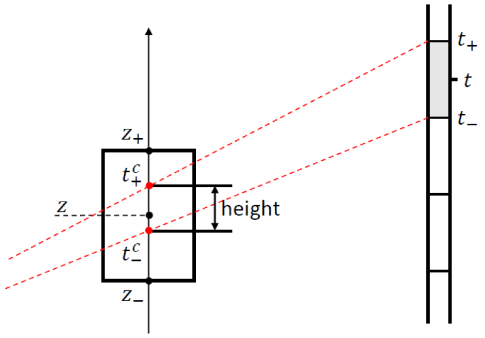

Fig. 8. Example of the overlapped distance method.

the area and height LUTs. This can be efficiently computed in parallel, assigning one thread per table and bin.

Then, per Section III-B, at a given view the base areas in the xy-plane are identical for all voxels along the z-direction. Hence, we dedicate a separate CUDA kernel (line 7-8) with $N_x \times N_y$ threads to compute for each (i,j) voxel index the set of base areas required for the detector bins (along s) the (i,j) voxel maps into.

The final CUDA kernel (line 10-16) performs the actual forward projection and has one thread per voxel (i,j,k). We configure threads into CUDA blocks that organize voxels along the z-direction [18] [19]. This enables threads within a block to efficiently share the base area information once loaded into shared memory and so gain high memory access speeds.

The thread first identifies the full maximum rectangular footprint of a voxel over (s,t). These extents form a nested loop over the covered t and s. The outer, (t), loop computes the effective height either by fetching it from the lookup table in texture memory or by computing using (15) or (16). The inner, (s), loop multiplies the height with the base area at this s-location (stored in shared memory) to compute the intersection volume of the voxel with the SBP at this (s,t) bin. This weight is then multiplied with the voxel value and the geometric divergence term. To add this product to the (s,t) bin we use *atomic* operations to avoid any race conditions incurred by other parallel threads targeting the same bin.

The corresponding CUDA kernel for back projection reverses the update step of the above forward projector. Here, the weighted CT measurements are sequentially added to a register and once fully accumulated the value is added to the 3D volume residing in global memory.

Algorithm 1: Forward projection with LUTs (one view).

1 **Procedure ForwardProjection**
2 Initialize projection array to zero;
3 $acLUT \leftarrow$ Compute area LUT coefficients A...C, θ_d;
4 $hcLUT \leftarrow$ Compute height LUT coefficients A...D, θ_ϕ,θ_φ;
5 **for** $j \in [0, N_y - 1]$ **do**
6 **for** $i \in [0, N_x - 1]$ **do**
7 $\{s_{ij}\} \leftarrow$ Determine detector bins associated with voxel (i,j) along the s-axis;
8 $\{A_{ij}\} \leftarrow$ Obtain base areas from area LUT using index (D_{line}, θ_d) computed via (3) and (5) with coefficients from $acLUT$;
9 **for** $k \in [0, N_z - 1]$ **do**
10 $d_{ijk} \leftarrow$ Compute distance from voxel (i,j) to source;
11 $\{t_{ijk}\} \leftarrow$ Determine detector bins associated with the voxel (i,j,k) along the t-axis;
12 **for** $t \in \{t_{ijk}\}$ **do**
13 $h_{ijk} \leftarrow$ Compute effective height from height LUT using index $(D_{pl}$,θ_ϕ,$\theta_\varphi)$ computed via (12) and coefficients from $hcLUT$. Or use (15) or (16) for approximated height. ;
14 **for** $s \in \{s_{ij}\}$ **do**
15 $p \leftarrow f[\mathbf{n}] \times A_{ij}[s] \times h_{ijk}/d_{ijk}$;
16 Update projection at (s, t) ;
17 Scale all projection views by $1/(|\sin\alpha_i| \cdot \gamma_{\phi,i}\gamma_{\varphi,i})$;

We store the 3D volume data in z-major order to facilitate data access of CUDA blocks. It allows a coalesced memory access pattern when reading (writing) into the array in forward (back)-projection. The storage of the CT projection data depends on the projection direction. For back-projection, we use (read-only) texture memory to take advantage of the fast L1 texture cache that is filled from our spatially local (z-major) memory access pattern. In forward projection, the CT data is stored in t-major order because the writing variation in t-direction is smaller than in the s-direction for a thread block.

The complete CUDA source code for the forward/back projection kernels is available at https://github.com/bsmind/LTRI.

V. Results

Our first goal was to find the optimal resolutions for the area and height look-up tables where we needed to balance two opposing goals – size and accuracy. We did this experimentally by assessing their accuracy while varying their resolutions. After finding these tables, we evaluated the proposed methods in terms of time performance and reconstructed image quality.

A. Optimal look-up tables

As mentioned, we devised an experimental procedure to determine the optimal size (that is, the optimal number of cells within the fixed extent) for each of the look-up tables (LUTs). We consider a LUT optimal when any further increase in size does not yield a significant improvement in bi- or tri-linear interpolation accuracy. We used a greedy method to determine these optimal sizes. For each LUT dimension, we gradually increased its size until the accuracy converged, while the other LUT dimension sizes were fixed at a reasonably high number.

This was repeated for each dimension separately to find the optimal multidimensional LUT. Our error metric is:

$$error = \frac{1}{N} \sum_{\mathbf{d} \in D} |EX(\mathbf{d}) - LUT(\mathbf{d})| \ , \qquad (17)$$

where D is a reference data set with N data point vectors, $\mathbf{d}$, which are $(D_{\text{line}}, \theta_d)$ for the area LUT ($aLUT$) and $(D_{pl}, \theta_\phi, \theta_\varphi)$ for the height LUT ($hLUT$). In this equation, $EX(\cdot)$ returns the exact solution of a voxel-ray intersection area (or volume) as described in Section II-A (or III-A), while $LUT(\cdot)$ returns the interpolated solution using the LUT under investigation. We stopped increasing a dimension size when the accuracy began to level off.

The reference set was built by uniformly sampling 4000×360 data points over the range of $D_{\text{line}} \in [0, D^t_{\text{line}}]$ and $\theta_d \in [0, 45°]$, respectively, for $aLUT$, where D^t_{line} was set to about 0.3 mm using (6). In the same way, for $hLUT$ $4000 \times 90 \times 90$ reference data points were collected over the range of $D_{pl} \in [0, D^t_{pl}]$, $\theta_\phi \in [0, \theta^{max}_\phi]$ and $\theta_\varphi \in [0, 45°]$, respectively, in which D^t_{pl} was 0.508 mm and θ^{max}_ϕ was $10°$ according to (13) and (14). Likewise, for the generation of the LUTs we also used a uniform sampling scheme. We used the same ranges as the reference data set but with different intervals according to the selected dimension size.

Fig. 9(a,b) shows our experimental process to determine the optimal $aLUT$. First, the D_{line} dimension length was varied with the θ_d dimension size fixed to 360. The optimal size of the D_{line} dimension was selected as 1500 because the accuracy improvement was not significant after that. With the optimal dimension size for D_{line} determined, the θ_d dimension size was explored. We found that here the optimal size converged at a value of 50. This yielded an optimal $aLUT$ of size 1500×50 points which takes about 0.3 MB of memory space in single precision. The optimal $hLUT$ was determined in a similar fashion (see Fig. 9(c,d,e)), resulting in a size of $1500 \times 25 \times 7$ with 1.05 MB space complexity. For a smoother appearance, we fit a non-linear regression curve to the error data points in Fig. 9. This eliminated a few outlier data points that had much smaller errors than the majority of their neighbors. These outliers occurred because the uniform sampling scheme caused some sampling points to match misleadingly well with the reference data. Fitting the regression curve helped in the visual assessment of the optimal dimension size.

The experimental results presented in Fig. 9 show that the LUTs are more sensitive in the distance dimensions (D_{line} and D_{pl}) than in the angular ones. This means that the functions for the distance dimensions have a higher degree of non-linearity, requiring a higher table resolution to keep the bi- or tri-linear interpolation error low. Fig. 7 shows this non-linear behavior for both $aLUT$ and $hLUT$ using a fixed angular dimension. Especially, for the $hLUT$, the non-linearity is not as severe as for $aLUT$ and this validates the approximate methods of III-C for the CT reconstructions presented in the following Section.

B. CT reconstruction with look-up tables

To competitively assess our LTRI method we compared it with the separable footprint (SF) method [7]. The SF has been shown to be more accurate than the (older) distance-driven (DD) method [3] while keeping a comparable time performance. Since the SF method was already compared exhaustively with the DD method in [7] we have not conducted a formal comparison with the DD method (see also [20])..
We begin by examining the accuracy of the two methods, LTRI and SF, for forward projection only and then move to comparing them within a complete iterative CT reconstruction framework using both phantom and clinical CT data.

1) Accuracy of forward projector: Since the volume integration model can be used for both forward- and back-projections, we focus on the forward projection to evaluate the accuracy of our LTRI method. Our first test object is a simple cube with side length 2 mm, uniform density. We placed the center of this object at four different locations – at $(0, 0, 0)$ mm and $(100, 150, 0)$ mm in the in-center plane, and at $(0, 0, -100)$ mm and $(100, 150, -100)$ mm in an off-center plane. Forward projection was simulated under an axial cone-beam X-ray CT system with a flat detector. The source-detector distance is 949 mm and the source-axis distance is 541 mm. We generated 360 true projection data uniformly distributed over $360°$ by linearly averaging 1000×1000 analytical line integrals of rays sampled over each detector bin where $\Delta_s = \Delta_t = 1$ mm.

For the LTRI, there are three approaches. The first one used a 1500×50 size $aLUT$ and a $1500 \times 25 \times 7$ $hLUT$ as we discovered in Section V-A, called LL. The other methods replaced the $hLUT$ with either the regression model in (15) or the distance model in (16) while keeping the same $aLUT$. We call these methods LL, LR and LD, respectively. For the separable footprint (SF) methods, we used either two trapezoid functions (TT) or one trapezoid and one rectangle function (TR) with the A1 amplitude method [7].

To evaluate the accuracy of each tested forward projector we define the maximum absolute error as

$$e_\theta = \max \left| \mathbf{P}^{\text{an}}_\theta - \mathbf{P}^{\text{ap}}_\theta \right| \ , \qquad (18)$$

where $\mathbf{P}^{\text{an}}_\theta$ is the projection obtained with the analytical method and $\mathbf{P}^{\text{ap}}_\theta$ is the same projection obtained with one of the tested approximate methods – LL, LR, LD, TT, and TR.

Fig. 10 plots the per-view e_θ on a logarithmic scale over all 360 views, and Table II summarizes its average (maximum) over this range of views, for all tested methods. We observe that when the cube is located in the center slice (cases a and b), for both LTRI and SF, the variants that approximate the height term are just as accurate as their accurate counterparts. This is because the height term is not relevant here. On the other hand, when the cube is the center (z-) column (a, c), the LTRI methods show about 1.6-2 times better accuracy than the SF methods. Lastly, when the cube is in an off-center column (b, d), then the LTRI methods are only slightly better or equivalent. We believe this occurs because the approximation of the geometric angular dispersion that all methods use have a dominating effect. In practice, however, the off-center slice, off-center column cubes (d) will be the most frequent. This is where the accurate versions, LL and TT, as well as the approximations LR, LD, and TR are about equivalent (within 5%). Finally, it is also interesting to see that the view-based

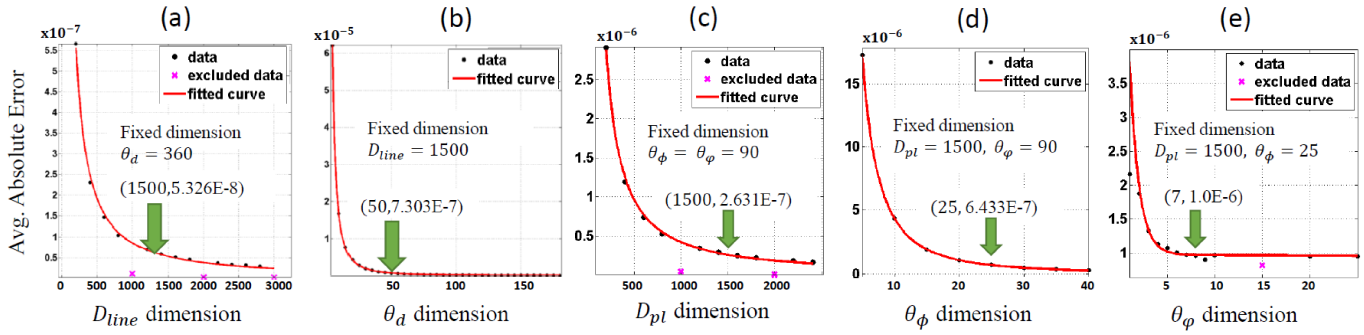

Fig. 9. Optimal LUT. Varying (a) D_{line}, (b) θ_d of area LUT, and (c) D_{pl}, (d) θ_ϕ, (e) θ_φ of height LUT.

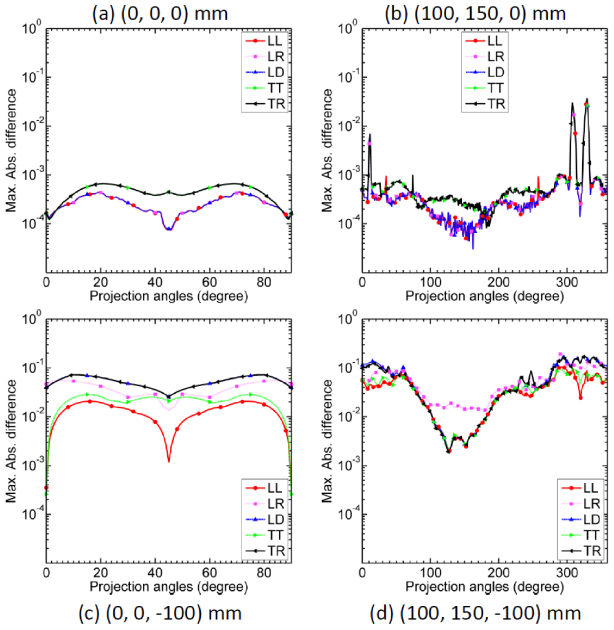

Fig. 10. Max. abs. error for the cube placed at different locations

TABLE II
AVERAGE (MAXIMUM) OF THE MAXIMUM ABSOLUTE ERROR (X10^{-2})

Loc [a]	LL	LR	LD	TT	TR
(a)	**0.02** (**0.04**)	0.02 (0.04)	0.02 (0.04)	**0.04** (**0.06**)	0.04 (0.06)
(b)	**0.11** (**3.70**)	0.11 (3.70)	0.11 (3.70)	**0.12** (**3.73**)	0.12 (3.73)
(c)	**1.33** (**2.06**)	3.82 (5.56)	5.31 (7.28)	**2.17** (**2.83**)	5.31 (7.23)
(d)	**3.75** (**10.4**)	5.95 (20.3)	6.52 (18.0)	**3.95** (**10.1**)	6.17 (17.2)

[a] Use index of Fig. 10 for the cube location.

error curves of Fig. 10 show that the difference between the two method families is especially pronounced around 120°.

2) Time performance of forward and back projections: The time performance for the forward and back projectors was measured by simulating an axial cone-beam CT system with 949.075 mm source to detector distance, 647.7 mm source to axis distance, and a flat detector with 512×512 bins and 1.0279×1.0964 mm^2 bin size. We used a uniform object with a resolution of $512 \times 512 \times 512$ and $0.4883 \times 0.4883 \times 0.6250$ mm^3 voxel size so that all voxels in the object would reside within the field-of-view in both forward and back-projections. Note that for the back-projection uniform measurements in all views were used. In addition to the average time performance over 360 views (which were uniformly distributed over 360°), we also measured the Giga-Updates Per Second (GUPS) rate which is independent of the problem size in the first order approximation [20]:

$$\text{GUPS} = \frac{512^3 \times 360}{1024^3}/\text{time}\ . \tag{19}$$

Here, 512^3 is the number of voxels involved in the forward and back projections, and 360 is the number of views.

Table III lists the results in ms for the average projection time and the GUPS rate is given in parentheses. All measurements were obtained using a NVIDIA GTX 1070 GPU. For the time measurements and analyses, we used the NVIDIA Nsight tool [21]. Recall that in our CUDA implementation both the intersecting base area for LTRI and the transaxial footprint for SF are pre-computed. In addition, the A1 amplitude term for SF and the angular dispersion term for LTRI are post-multiplied (to the projections) in the forward projection operation and pre-multiplied in the back-projection operation. In both SF and LTRI, these two operations take less than 0.1 ms which is negligible compared to the execution time for the main projection operation. Thus, the time performance listed in Table III is solely differentiated by the utilization of GPU memory and the number of arithmetic operations in computing the height term of the ray-voxel intersection for LTRI or the axial footprint in the z-direction for SF.

A first observation we make is that in all methods back-projection (BP) is generally faster than forward projection (FP). This is due to the serialized memory write operations.

For LL, both the height look-up table and the CT measurement data reside in texture memory. During projection operations, the memory access pattern to the height look-up table is random which causes L1 texture cache bank conflicts – this is the main reason for the slow time performance. Likewise, in back-projection, the bank conflicts in L1 texture cache reduce the L1 texture cache hit rate from 80% to 20% for reading the CT measurement data from the texture memory. Storing the height LUT in other memories did not help matters either since these facilities are either not suitable, slower, or incur access conflicts with other data elements.

TABLE III
TIME PERFORMANCE COMPARISONS [MSEC, (GUPS)]

	LL	LR	LD	TT	TR
FP	44.3, (2.8)	19.0, (6.5)	18.7, (6.6)	19.9, (6.2)	18.2, (6.8)
BP	42.0, (2.9)	10.4, (12.0)	9.6, (13.0)	13.4, (9.3)	9.4, (13.1)

LR, LD, and TR which approximate the axial contribution of a voxel to a detector bin with a simple model show better time performance than either LL and TT due to the reduced arithmetic/memory overhead. LD is slightly better than LR because of its smaller amount of arithmetic operations – LR requires three multiplications and three additions to calculate the signed distance (D_{pl}) from a voxel to a plane which is not required for LD. All LTRI methods need to compute the distance from a voxel to the X-ray source for the geometric spreading term – this adds an additional three additions and one division operation which are not required by TR. Hence, TR shows a slightly (2%) better time performance than LD.

3) Within iterative CT reconstruction: Next we plugged the LTRI method into a simultaneous algebraic reconstruction technique (SART) framework [22]. To investigate the performance within SART, an axial cone-beam CT with 949.075 mm for the source to detector distance and 647.7 mm for the source to axis distance was used. A flat detector was assumed with 888×640 detector bins and 1.0279×1.0964 mm^2 bin size. Noiseless 360 CT projections uniformly distributed over $360°$ of a modified Shepp-Logan phantom (SLP) were simulated by linearly averaging a set of 8×8 analytical line integrals for each detector bin [23]. We run SART for 500 iterations with a 0.0025 relaxation factor.

Fig. 11 presents a qualitative comparison of the reconstructed images for the Shepp-Logan phantom, captured after 500 SART iterations, for slices taken transversal, coronal, and sagital for all variations of the LTRI and SF methods. The SLP slices on the left indicate the direction of two profile plots (red and yellow lines) and these plots are shown in the last row of Fig. 11. We observe that the slices look virtually identical for all methods and so do the profiles. Finally, for a quantitative comparison of the two methods we measured the RMS error after each SART iteration had completed:

$$e_{\text{RMS}} = \sqrt{\frac{1}{M}\sum_{i=1}^{M} ||f(i) - f_{SART}(i)||}\ , \qquad (20)$$

where f is the original Shepp-Logan phantom used to generate the projection data, f_{SART} is the image reconstructed with either SF (TT and TR) or LTRI (LL, LR and LD), and M is the number of voxels. We found that the RMS error measured after each SART iteration was virtually identical, and so was the convergence rate for all methods.

The findings obtained for the phantom scenario fully extend into the clinical setting. We obtained a set of 360 clinical CT projections of a cervical region with 20 cm of field-of-view using a Medtronic O-arm O2 surgical imaging system with 1147.7 mm source to detector distance and 647.7 mm source to axis distance and a flat detector with 1024×386 bins of size 0.384×0.755 mm^2. The object size for the reconstruction was $512 \times 512 \times 196$ with a $0.415 \times 0.415 \times 0.83$ mm^3 voxel size. As Fig. 12 shows, the reconstructions and profile plots obtained with the clinical data set after 200 SART iterations are nearly indistinguishable. For brevity,we only present reconstruction results for LTRI-LL and SF-TT but confirm that all variations show a similar visual appearance and profiles.

VI. CONCLUSIONS

Modeling a CT system as accurately as possible has become an important goal in CT reconstruction research. It is particularly crucial in iterative reconstruction where systematic errors can hamper convergence. In this work we have focused on performing the volumetric integration of the box-shaped voxel basis function at high fidelity. In contrast to previous efforts that have used projection space approaches to model the basic function's projective footprint, we have developed a scheme that performs the voxel integration directly in image space. Intersecting the box-shaped voxel with the pyramidal X-ray beam segment defined by the boundaries of the respective detector bin gives rise to an irregular polyhedron which is difficult to integrate. To make this integration computationally feasible we have devised an efficient lookup table-based approach which is amenable to GPU acceleration. We throughly compared our image-space method with the state-of-the-art projection space method – the separable footprint. For this study we fully optimized GPU-implementations of both methods and also derived two mindful approximations for ours. We find that our method and its variations have similar performance both in time and accuracy than the respective variations of the separable footprint method and by these measures it is equivalent. Nevertheless, we believe that the extensive study we performed sheds new light on understanding the accurate modeling of voxel-based basis functions. Furthermore, all our program code is open source and freely available on github.

ACKNOWLEDGMENT

This research was partially supported by NSF grant IIS 1527200 and the MISP (Ministry of Science, ICT & Future Planning), Korea, under the "ICT Consilience Creative Program" (IITP-2015-R0346-15-1007) supervised by the IITP. We also thank Medtronic, Inc. for their support.

REFERENCES

[1] R. L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional ct array," *Medical physics*, vol. 12, no. 2, pp. 252–255, 1985.

[2] P. M. Joseph, "An improved algorithm for reprojecting rays through pixel images," *IEEE transactions on medical imaging*, vol. 1, no. 3, pp. 192–196, 1982.

[3] B. De Man and S. Basu, "Distance-driven projection and backprojection in three dimensions," *Physics in medicine and biology*, vol. 49, no. 11, p. 2463, 2004.

[4] H. Yu and G. Wang, "Finite detector based projection model for high spatial resolution," *Journal of X-ray Science and Technology*, vol. 20, no. 2, pp. 229–238, 2012.

[5] P. M. Joseph and R. D. Spital, "The exponential edge-gradient effect in x-ray computed tomography," *Physics in medicine and biology*, vol. 26, no. 3, p. 473, 1981.

[6] S. Zhang, D. Zhang, H. Gong, O. Ghasemalizadeh, G. Wang, and G. Cao, "Fast and accurate computation of system matrix for area integral model-based algebraic reconstruction technique," *Optical Engineering*, vol. 53, no. 11, pp. 113 101–113 101, 2014.

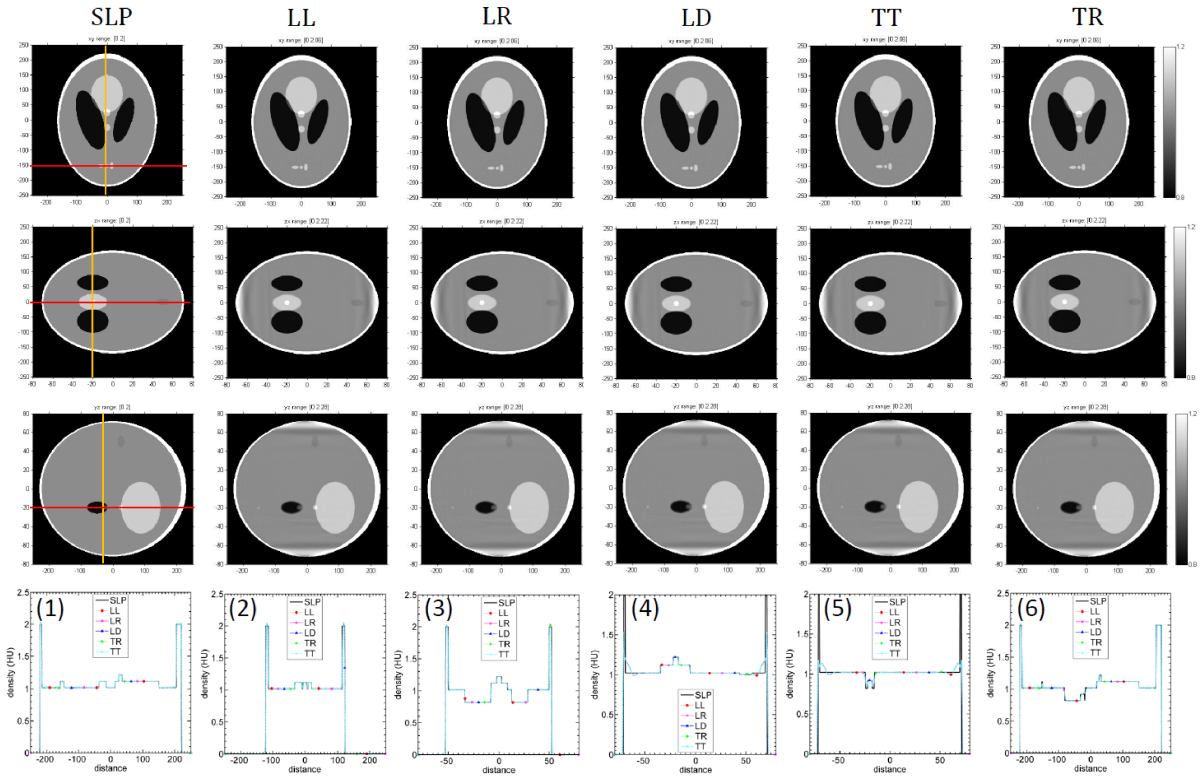

Fig. 11. Shepp-Logan phantom (SLP) study. Top to bottom: transverse, sagittal, coronal slices of the reconstructions and the profile plots across the lines indicated in the SLP slices . Left to right: the original SLP phantom and the various methods we discussed. For the profiles (last row), (1,2) corresponds to the vertical (yellow) and horizontal (red) lines in the transverse view. Likewise, plots (3,4) are for the sagittal view and plots (5,6) is for the coronal view.

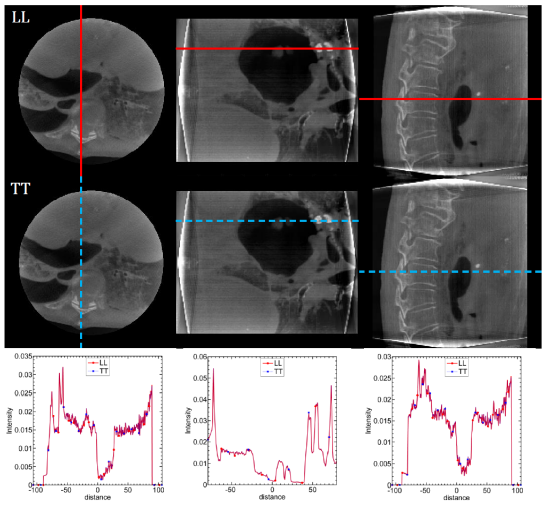

Fig. 12. Clinical study. [top] LTRI (LL), [middle] SF (TT), [bottom] profile analysis. From left to right: transverse, sagittal, and coronal views.

[7] Y. Long, J. A. Fessler, and J. M. Balter, "3D forward and back-projection for X-ray CT using separable footprints," *IEEE Transactions on Medical Imaging*, vol. 29, no. 11, pp. 1839–1850, 2010.

[8] S. Ha, A. Kumar, and K. Mueller, "A study of volume integration models for iterative cone-beam computed tomography," *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, 2015.

[9] 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.

[10] ——, "Real-time 3d computed tomographic reconstruction using commodity graphics hardware," *Physics in medicine and biology*, vol. 52, no. 12, p. 3405, 2007.

[11] S. Ha, H. Li, and K. Mueller, "Efficient area-based ray integration using summed area tables and regression models," *CT Meeting*, 2016.

[12] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra, "Fast summed-area table generation and its applications," in *Computer Graphics Forum*, vol. 24, no. 3. Wiley Online Library, 2005, pp. 547–555.

[13] A. Entezari, M. Nilchian, and M. Unser, "A box spline calculus for the discretization of computed tomography reconstruction problems," *IEEE transactions on medical imaging*, vol. 31, no. 8, pp. 1532–1541, 2012.

[14] A. Ziegler, T. Köhler, T. Nielsen, and R. Proksa, "Efficient projection and backprojection scheme for spherically symmetric basis functions in divergent beam geometry," *Medical physics*, vol. 33, no. 12, pp. 4653–4663, 2006.

[15] "Nvidia cuda c programming guide," https://docs.nvidia.com/cuda/cuda-c-programming-guide.

[16] E. W. Weisstein, "Polygon area," From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/PolygonArea.html.

[17] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[18] E. Papenhausen, Z. Zheng, and K. Mueller, "Gpu-accelerated back-projection revisited: squeezing performance by careful tuning," in *Workshop on High Performance Image Reconstruction (HPIR)*, 2011, pp. 19–22.

[19] M. Wu and J. A. Fessler, "Gpu acceleration of 3d forward and backward projection using separable footprints for x-ray ct image reconstruction," in *Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med*, vol. 6. Citeseer, 2011, p. 021911.

[20] R. Liu, L. Fu, B. De Man, and H. Yu, "Gpu-based branchless distance-driven projection and backprojection," *IEEE Transactions on Computational Imaging*, 2017.

[21] "Nvidia nsight," http://www.nvidia.com/object/nsight.html.

[22] A. H. Andersen and A. C. Kak, "Simultaneous algebraic reconstruction technique (sart): a superior implementation of the art algorithm," *Ultrasonic imaging*, vol. 6, no. 1, pp. 81–94, 1984.

[23] A. C. Kak and M. Slaney, *Principles of computerized tomographic imaging*. SIAM, 2001.