Fast parallel algorithms for the x-ray transform and its adjoint

Hao Gao^{a)}

Department of Mathematics and Computer Science, Emory University, Atlanta, Georgia 30322
and Department of Radiology and Imaging Sciences, Emory University, Atlanta, Georgia 30322

(Received 8 July 2012; revised 29 September 2012; accepted for publication 2 October 2012;
published 5 November 2012)

Purpose: Iterative reconstruction methods often offer better imaging quality and allow for reconstructions with lower imaging dose than classical methods in computed tomography. However, the computational speed is a major concern for these iterative methods, for which the x-ray transform and its adjoint are two most time-consuming components. The speed issue becomes even notable for the 3D imaging such as cone beam scans or helical scans, since the x-ray transform and its adjoint are frequently computed as there is usually not enough computer memory to save the corresponding system matrix. The purpose of this paper is to optimize the algorithm for computing the x-ray transform and its adjoint, and their parallel computation.

Methods: The fast and highly parallelizable algorithms for the x-ray transform and its adjoint are proposed for the infinitely narrow beam in both 2D and 3D. The extension of these fast algorithms to the finite-size beam is proposed in 2D and discussed in 3D.

Results: The CPU and GPU codes are available at <https://sites.google.com/site/fastxraytransform>. The proposed algorithm is faster than Siddon's algorithm for computing the x-ray transform. In particular, the improvement for the parallel computation can be an order of magnitude.

Conclusions: The authors have proposed fast and highly parallelizable algorithms for the x-ray transform and its adjoint, which are extendable for the finite-size beam. The proposed algorithms are suitable for parallel computing in the sense that the computational cost per parallel thread is $O(1)$.
© 2012 American Association of Physicists in Medicine. [<http://dx.doi.org/10.1118/1.4761867>]

Key words: x-ray transform, discrete Radon transform, parallel algorithm, CT, GPU

I. INTRODUCTION

Computed tomography (CT) is perhaps the most widely used medical imaging modality. For health care purpose, the reduction of imaging doses is highly desirable. A promising practice for the dose reduction is through iterative image reconstruction methods, which often offer better image reconstruction quality than the conventional methods, especially for the low-dose scans³⁻⁷ and four-dimensional (4D) CT scans.⁸⁻¹² Despite their advantages, the iterative methods are not yet commonly used for clinical diagnosis. A critical obstacle is that the iterative methods need to be further accelerated to meet the clinical needs. To address this, various algorithms¹³⁻¹⁵ and GPU-based parallel solvers¹⁶⁻²⁰ were developed for computing the x-ray transform and its adjoint, which are often the most computationally expensive components of the iterative methods. The focus of this paper is to study the fast parallel algorithms for the x-ray transform and its adjoint.

We shall first specify the notations for the discrete x-ray transform. For simplicity, let us consider a 2D grid with (N_x, N_y) pixels, illuminated with the infinitely narrow beams under the fan-beam CT setting from N_v directions, with N_d detectors for each direction. For each beam indexed by (m, n) , $m \leq N_v$, $n \leq N_d$, the x-ray transform is to compute the summation of the intersection lengths of the beam with each pixel weighted by the attenuation coefficient of that pixel [Fig. 1(a)]. That is,

$$y_{mn} = \sum_{(i,j) \in D_{mn}} l_{mn,ij} x_{ij},$$
$$D_{mn} = \{(i, j) : l_{mn,ij} > 0, i \leq N_x, j \leq N_y\}, \quad (1.1)$$

where x is the attenuation coefficient, $l_{mn,ij}$ is the intersection length of the beam (m, n) with the pixel (i, j) , and D_{mn} consists of the indices of pixels that have nontrivial intersections with the beam (m, n) .

Under the same fan-beam CT setting, for each pixel indexed by (i, j) , $1 \leq i \leq N_x$, $1 \leq j \leq N_y$, the adjoint of the discrete x-ray transform is to compute the summation of the intersection lengths of all the beams that nontrivially intersect with this pixel weighted by their x-ray transform data y . That is,

$$x_{ij} = \sum_{(m,n) \in D_{ij}} l_{mn,ij} y_{mn},$$
$$D_{ij} = \{(m, n) : l_{mn,ij} > 0, m \leq N_v, n \leq N_d\}, \quad (1.2)$$

where D_{ij} consists of the indices of beams that have nontrivial intersections with the pixel (i, j) .

A popular algorithm for the x-ray transform was proposed by Siddon.² The key is the parametric line representation of the beams so that the complexity of computing the intersection lengths of each beam with a 2D or 3D domain is still with respect to a 1D line. Take a 2D domain with the isocenter at the origin and unit pixel length for example. The algorithm computes the parametric coordinates of the intersections of the beam with the line sets $\{x = i, -N_x/2 \leq i \leq N_x/2\}$ and $\{y = i, -N_y/2 \leq i \leq N_y/2\}$, sort these coordinates, and then find the nontrivial intersection pixels with the intersection length for every two consecutive parametric coordinates [Fig. 1(b)]. Therefore, the computational complexity for each beam is still with respect to 1D, i.e., $O(N_x)$, the total complexity is roughly $O(N_x \cdot N_v \cdot N_d)$, and the complexity of its ideal parallel version

FIG. 1. The x-ray transform. (a) The x-ray transform is to compute the summation of the intersection lengths of the beam with each pixel weighted by its attenuation coefficient. (b) The key of the popular Siddon's algorithm (Ref. 2) is to consider the parametric line form of the intersections.

is $O(N_x)$. However, this algorithm for infinitely narrow beams does not generalize for finite-size beams, and it does not directly apply to the parallel computation of the adjoint x-ray transform either.

In this paper, we will study the fast and highly parallelizable algorithms for the x-ray transform and its adjoint for both the infinitely narrow beam and the finite-size beam, and then present the numerical results to demonstrate the efficiency of the proposed algorithms.

II. ALGORITHMS FOR THE X-RAY TRANSFORM

II.A. 2D version

Let us first consider the intersection of a beam with two end points (x_1, y_1) and (x_2, y_2) , with a 2D domain D of (N_x, N_y) pixels and the pixel size (dx, dy) . Without loss of generality, let $N_x = N_y$, $dx = dy = 1$, and the isocenter be at the origin. Let us also define the columns of the domain to be the following, which consists of the centers of pixels,

$$C_i = \{(x, y) : x = i + 1/2\}, -N_x/2 \leq i \leq N_x/2 - 1, \quad (2.1)$$

and similarly the rows of the domain to be

$$R_j = \{(x, y) : y = j + 1/2\}, -N_y/2 \leq j \leq N_y/2 - 1. \quad (2.2)$$

The key for the proposed algorithm is the observation that when the absolute slope of the beam is smaller than 1, i.e., $|x_2 - x_1| > |y_2 - y_1|$, the beam intersects each column (2.1) for at most two pixels (Fig. 2); otherwise, the beam intersects each row (2.2) for at most two pixels (Fig. 3). (In general, the classification criterion should be $dy/dx > |y_2 - y_1|/|x_2 - x_1|$ instead. However, for the simplicity of presentation, $dx = dy = 1$ is assumed in the following discussion.)

When $|x_2 - x_1| > |y_2 - y_1|$, it is natural to divide the 2D domain D into the columns (2.1) [Fig. 2(a)]. Within each column, the beam intersects nontrivially with only at most two pixels, which is essential for the efficient parallelization. Moreover, the indices and the intersection lengths can be conveniently determined. That is, for the i th column, we first compute the y -coordinates of the intersections of the beam with two vertical boundaries of this column, i.e.,

$$y_{i-} = k_y(i - x_1) + y_1 + N_y/2 \quad \text{and}$$

$$y_{i+} = k_y(i + 1 - x_1) + y_1 + N_y/2 \quad (2.3)$$

with

$$k_y = (y_2 - y_1)/(x_2 - x_1), \quad (2.4)$$

and then take the greatest integers that are smaller than y_{i-} and y_{i+} , i.e.,

$$Y_{i-} = \lfloor y_{i-} \rfloor \quad \text{and} \quad Y_{i+} = \lfloor y_{i+} \rfloor, \quad (2.5)$$

which are exactly the y -indices of the intersecting pixel candidates. If $Y_{i+} = Y_{i-}$, there is only one intersecting pixel provided $1 \leq Y_{i-} \leq N_y$ [Fig. 2(b)], i.e.,

$$(i, Y_{i-}) : l = \sqrt{1 + k_y^2}, \quad \text{if } Y_{i+} = Y_{i-} \text{ and } 1 \leq Y_{i-} \leq N_y; \quad (2.6)$$

if $Y_{i+} \neq Y_{i-}$, there are two consecutive intersecting pixels provided $1 \leq Y_{i-}, Y_{i+} \leq N_y$ [Fig. 2(b)], i.e.,

$$\begin{cases} (i, Y_{i-}) : l = \frac{\max(Y_{i-}, Y_{i+}) - y_{i-}}{y_{i+} - y_{i-}} \sqrt{1 + k_y^2}, & \text{if } Y_{i+} \neq Y_{i-} \text{ and } 1 \leq Y_{i-} \leq N_y \\ (i, Y_{i+}) : l = \frac{y_{i+} - \max(Y_{i-}, Y_{i+})}{y_{i+} - y_{i-}} \sqrt{1 + k_y^2}, & \text{if } Y_{i+} \neq Y_{i-} \text{ and } 1 \leq Y_{i+} \leq N_y \end{cases} \quad (2.7)$$

Similarly, when $|x_2 - x_1| \leq |y_2 - y_1|$, it is natural to divide the 2D domain D into the rows (2.2) [Fig. 3(a)]. Again, within each row, the beam intersects nontrivially with only at most two pixels [Fig. 3(b)]. We will skip the algorithm details since it is similar to the case with $|x_2 - x_1| > |y_2 - y_1|$.

For the convenience of implementation, the algorithm for the 2D x-ray transform with $|x_2 - x_1| > |y_2 - y_1|$ is summarized

in Appendix C. For details of the CPU and GPU implementations in C see Ref. 1.

Here for the convenience of the presentation, $dx = dy = 1$ is assumed. In case that $dx = dy \neq 1$, the following algorithm still works after the normalization by dividing all lengths by dx . In case that $dx \neq dy$, a few corresponding changes need to be made. For example, the criterion on whether to divide the

FIG. 2. Fast algorithm for the x-ray transform. (a) When $|x_2 - x_1| > |y_2 - y_1|$, it is natural to divide the 2D domain D into the columns; (b) within each column, the beam intersects nontrivially with only at most two pixels, which allows for an efficient parallelization with the $O(1)$ computation per thread.

domain into columns or rows depends on the absolute slope $|dy/dx|$ instead of 1. Without further mentioning, the situation is similar for the following 3D algorithm.

The computational complexity of this algorithm is $O(N_x)$ for each beam. It is faster than Siddon's algorithm² mainly due to its simplicity and no required sorting, for which the numerical verification will be presented. More importantly, when considering the parallel version, the complexity of the new algorithm is $O(1)$ per parallel thread, since the intersection of the beam for each row/column can be computed in parallel. In contrast, the complexity is $O(N_x)$ per parallel thread for the parallelized Siddon's algorithm. In this aspect, this new algorithm is more suitable for parallel computing.

II.B. 3D version

The previous 2D algorithm for the x-ray transform can be extended to 3D. Let us similarly consider the intersection of a beam with two end points (x_1, y_1, z_1) and (x_2, y_2, z_2) , with a 3D domain of (N_x, N_y, N_z) voxels and the pixel size (dx, dy, dz) . Without loss of generality, let $N_x = N_y$, $dx = dy = 1$, and the isocenter be at the origin. Let us also define the x -planes of the domain

$$XP_i = \{(x, y, z) : x = i + 1/2\}, -N_x/2 \leq i \leq N_x/2 - 1, \quad (2.8)$$

FIG. 3. Fast algorithm for the x-ray transform. (a) When $|x_2 - x_1| \leq |y_2 - y_1|$, it is natural to divide the 2D domain D into the rows; (b) within each row, the beam intersects nontrivially with only at most two pixels, which allows for an efficient parallelization with the $O(1)$ computation per thread.

the y -planes of the domain

$$YP_j = \{(x, y, z) : y = j + 1/2\}, -N_y/2 \leq j \leq N_y/2 - 1, \quad (2.9)$$

and the z -planes of the domain

$$ZP_k = \{(x, y, z) : z = (k + 1/2)dz\}, -N_z/2 \leq k \leq N_z/2 - 1. \quad (2.10)$$

Again, the key for the proposed fast algorithm is the observation that when $|x_2 - x_1| > |y_2 - y_1|$ and $|x_2 - x_1| > |z_2 - z_1|$, the beam intersects each x -plane (2.8) for at most four pixels (Fig. 4); when $|y_2 - y_1| \geq |x_2 - x_1|$ and $|y_2 - y_1| > |z_2 - z_1|$, the beam intersects each y -plane (2.9) for at most four pixels; when $|z_2 - z_1| \geq |x_2 - x_1|$ and $|z_2 - z_1| \geq |y_2 - y_1|$, the beam intersects each z -plane (2.10) for at most four pixels. In the following, let us only consider the first situation, since the other two cases can be derived similarly.

In the case with $|x_2 - x_1| > |y_2 - y_1|$ and $|x_2 - x_1| > |z_2 - z_1|$, it is natural to divide the 3D domain into the x -planes (2.8) [Fig. 4(a)]. Inspired by the 2D case, we view the projection of the intersection into the x - y plane (i.e., along y -direction of the x -plane) and the x - z plane (i.e., along z -direction of the x -plane). That is, for the voxels within each x -plane, the beam intersects nontrivially with only at most two consecutive voxels along the y -direction, and at most two consecutive voxels along the z -direction as well, which is again essential for the efficient parallelization.

In fact, the intersections can be at most three consecutive voxels contained in a 2×2 "box" [Fig. 4(a)], which can be classified into five cases shown in Figs. 4(b)–4(f). The cases with one or two intersecting voxels are similar to the 2D case, i.e., one intersecting voxel along both y -direction (when viewed in the x - y projection plane) and z -direction (when viewed in the x - z projection plane) [Fig. 4(b)], one intersecting voxel along y -direction and two intersecting voxels along z -direction [Fig. 4(c)], and two intersecting voxels along y -direction and one intersecting voxel along z -direction [Fig. 4(d)]. In these cases, the indices and the weights can be determined with the similar 2D formulas as (2.6) and (2.7).

The situation is slightly more complicated when there are two intersecting voxels along both y -direction and z -direction [Figs. 4(e) and 4(f)]. To determine the intersection indices and lengths for the i th x -plane, we first compute y_{i+} , y_{i-} , Y_{i+} , Y_{i-} through (2.3)–(2.5), and the intersecting ratio of the projected beam along the y -direction

$$r_y = \frac{\max(Y_{i-}, Y_{i+}) - y_{i-}}{y_{i+} - y_{i-}}; \quad (2.11)$$

then similarly determine z_{i+} , z_{i-} , Z_{i+} , Z_{i-} and the intersecting ratio of the projected beam along the z -direction through follows:

$$z_{i-} = [k_z(i - x_1) + z_1]/dz + N_z/2 \quad \text{and} \quad z_{i+} = [k_z(i + 1 - x_1) + z_1]/dz + N_z/2 \quad (2.12)$$

with

$$k_z = (z_2 - z_1)/(x_2 - x_1), \quad (2.13)$$

FIG. 4. Fast algorithm for the x-ray transform in 3D. (a) when $|x_2 - x_1| > |y_2 - y_1|$ and $|x_2 - x_1| > |z_2 - z_1|$, it is natural to divide the 3D domain D into the x -planes, and the intersections can be up to three consecutive voxels contained in a 2×2 "box"; (b) the case with one intersecting voxel along both y -direction (when viewed in the x - y projection plane) and z -direction (when viewed in the x - z projection plane); (c) the case with one intersecting voxel along y -direction and two intersecting voxels along z -direction; (d) the case with two intersecting voxels along y -direction and one intersecting voxel along z -direction; (e) and (f) the cases with two intersecting voxels along both y -direction and z -direction (Fig. 5).

$$Z_{i-} = \lfloor z_{i-} \rfloor \quad \text{and} \quad Z_{i+} = \lfloor z_{i+} \rfloor, \quad (2.14)$$

and

$$r_z = \frac{\max(Z_{i-}, Z_{i+}) - z_{i-}}{z_{i+} - z_{i-}}. \quad (2.15)$$

When $r_y > r_z$ [Fig. 4(e)], the y -direction intersecting point is above the z -direction intersecting point when both are viewed along the z -direction in the x - z projection plane [Fig. 5(a)]. Therefore, the indices and the weights of three intersecting voxels are

$$\begin{cases} (i, Y_{i-}, Z_{i-}) : l = r_z \sqrt{1 + k_y^2 + k_z^2}, & \text{if } 1 \leq Y_{i-} \leq N_y \text{ and } 1 \leq Z_{i-} \leq N_z \\ (i, Y_{i-}, Z_{i+}) : l = (r_y - r_z) \sqrt{1 + k_y^2 + k_z^2}, & \text{if } 1 \leq Y_{i-} \leq N_y \text{ and } 1 \leq Z_{i+} \leq N_z \\ (i, Y_{i+}, Z_{i+}) : l = (1 - r_y) \sqrt{1 + k_y^2 + k_z^2}, & \text{if } 1 \leq Y_{i+} \leq N_y \text{ and } 1 \leq Z_{i+} \leq N_z \end{cases} \quad (2.16)$$

Otherwise, when $r_y \leq r_z$ [Fig. 4(f)], the y -direction intersecting point is below the z -direction intersecting point when both are viewed along the z -direction in the x - z projection plane [Fig. 5(b)]. Therefore, the indices and the weights of three intersecting voxels are

$$\begin{cases} (i, Y_{i-}, Z_{i-}) : l = r_y \sqrt{1 + k_y^2 + k_z^2}, & \text{if } 1 \leq Y_{i-} \leq N_y \text{ and } 1 \leq Z_{i-} \leq N_z \\ (i, Y_{i+}, Z_{i-}) : l = (r_z - r_y) \sqrt{1 + k_y^2 + k_z^2}, & \text{if } 1 \leq Y_{i+} \leq N_y \text{ and } 1 \leq Z_{i-} \leq N_z \\ (i, Y_{i+}, Z_{i+}) : l = (1 - r_z) \sqrt{1 + k_y^2 + k_z^2}, & \text{if } 1 \leq Y_{i+} \leq N_y \text{ and } 1 \leq Z_{i+} \leq N_z \end{cases} \quad (2.17)$$

FIG. 5. Projection view of Figs. 4(e) and 4(f) into the x - z plane. (a) When $r_y > r_z$ [Fig. 4(e)], the y -direction intersecting point is above the z -direction intersecting point when both are viewed along the z -direction in the x - z projection plane; (b) when $r_y < r_z$ [Fig. 4(f)], the y -direction intersecting point is below the z -direction intersecting point when both are viewed along the z -direction in the x - z projection plane.

For the convenience of implementation, the algorithm for the 3D x-ray transform with $|x_2 - x_1| > |y_2 - y_1|$ and $|x_2 - x_1| > |z_2 - z_1|$ is summarized in Appendix D. For details of the CPU and GPU implementations in C see Ref. 1.

As a result, the computational complexity of this algorithm for 3D is also $O(N_x)$ for each beam. It is again faster than Siddon's algorithm² mainly due to its simplicity and no required sorting, for which the numerical verification will be also presented next. Moreover, when considering the parallel version, the complexity of the new algorithm in 3D is again $O(1)$ per parallel thread, since the intersection of the beam for each 2D plane can be computed in parallel. In contrast, the complexity is $O(N_x)$ per parallel thread for the parallelized Siddon's algorithm. In this aspect, this new algorithm is more suitable for parallel computing in 3D.

II.C. Generalization for the finite-size beam

For improved accuracy of the x-ray transform, it is sometimes necessary to take the finite size of the beam into account, e.g., when the width of the beam is larger than the pixel size. Unlike the Siddon algorithm, the proposed new algorithms have a natural and simple extension for the finite-size beam.

For simplicity, let us take the 2D case with the same notations as in Sec. II.A for example. The key for the fast and highly parallelizable algorithm is similar as that for the infinitely narrow beam. That is, when the absolute slope of the central beam line of this finite-size beam is smaller than 1, i.e., $|x_2 - x_1| > |y_2 - y_1|$, we divide the 2D domain into the columns (2.1) and compute the intersection of the beam with each column individually. As a result, the parallelization is efficient, since the nontrivial intersection of the beam with each column is still at most a few pixels as long as the beam is still fairly focused. Even in the rare cases when the beam size is significantly larger than the pixel size, this algorithm is still suitable for the parallelization.

In this case with $|x_2 - x_1| > |y_2 - y_1|$, within each column (2.1), the indices of all intersecting pixels can be determined

by, two intersecting pixels by the boundary beam lines of the finite-size beam with the column, as the lower and upper bound. Then the intersection area of the beam with each nontrivially intersecting pixel can be computed with an efficient geometric formula given in Appendix B. For details of the CPU and GPU implementations in C see Ref. 1.

Similarly in 3D, the 3D domain can be divided into x -planes (2.8), y -planes (2.9), and z -planes (2.10) according to the slopes of the beam. The intersecting voxels within each plane can be specified by the lower and upper bounding voxels from two directions. For example, when considering x -planes, the lower and upper bounding voxels are from the y -direction and the z -direction. However, an efficient 3D formula to compute the intersection volume of the beam with each nontrivially intersecting voxel needs to be supplied.

In the following, the extension to the finite-size beam is implemented in 2D. Overall, in the case with finite-size beams, the computational complexity of the new algorithms is still $O(N_x)$ for each finite-size beam, and the complexity of their parallel version is again $O(1)$ per parallel thread.

III. ALGORITHMS FOR THE ADJOINT X-RAY TRANSFORM

III.A. 2D version

By the nature of the adjoint x-ray transform (1.2), its computation can be done in parallel at least for each pixel. In this aspect, the efficient algorithm for the adjoint x-ray transform will be quite different from that for the x-ray transform. For simplicity, let us start with the fan-beam setting [Fig. 6(a)] using the same notations as in Sec. II.A, and we shall consider the adjoint x-ray transform for a fixed pixel. For the notations, let (x_c, y_c) be the coordinate of the center of the pixel, SO be the distance from the source to the isocenter, SD be the distance from the source to the detection line, d_y be the detector width, and d_γ be the scan angle. Without loss of generality, let $dx = dy = 1$.

For a fixed pixel and a fixed scanning direction, the contribution from this scan to this pixel only comes from the detectors that lie in the shadow region that is bounded above and below by the beams passing through four corners of the pixel [Fig. 6(b)]. To put this into the algorithm, we first rotate the coordinate system back to the original position for the ease of the computation, so that the detector is perpendicular to the x -axis. As a result, the new coordinate (x_r, y_r) of the center of the pixel is

$$x_r = x_c \cos \phi + y_c \sin \phi \quad \text{and} \quad y_r = -x_c \sin \phi + y_c \cos \phi. \quad (3.1)$$

Then we determine the lower and upper bound of the contributing detectors. Due to the rotation, the pixel may not be aligned with the coordinate axis. Therefore, the computation for the beams passing through four corners is required, and then the minimum and the maximum are determined as the lower and upper bound. However for the computation efficiency, another way is to consider the circle that encloses this pixel, and then the upright rhombus that encloses this circle

FIG. 6. Fast parallelizable algorithm for the adjoint x-ray transform. (a) As the first parallelization step, the computation for the adjoint x-ray transform is done in parallel for each pixel. (b) As the second parallelization step, for a fixed pixel, the computation is done in parallel for each scan view, for which the contribution from this scan view to this pixel only comes from the detectors that lie in the shadow region that is bounded above and below by the beams passing through four corners of the pixel. Therefore, the computational complexity is $O(1)$ per parallel thread, assuming the summation of the contribution from all views is relatively negligible.

[Fig. 6(b)]. Then the diagonal of this rhombus is also the unit length. Therefore, the lower and upper index bound of the contributing detectors are

$$\begin{cases} n_{\min} = \lfloor [(y_r - 1) \cdot SD / (x_r + SO)] / d_y + N_d / 2 \rfloor \\ n_{\max} = \lfloor [(y_r + 1) \cdot SD / (x_r + SO)] / d_y + N_d / 2 \rfloor \end{cases} \quad (3.2)$$

Once the contributing detectors are found, which are often only a few, the efficient geometric formula given in Appendix A can be used to compute the intersection length of the beam from these detectors with the pixel. Please note that the offset of the detection plane with respect to the isocenter can be conveniently taken care of by adding the offset distance y_{os} to Eq. (3.2).

For the convenience of implementation, the algorithm for the 2D adjoint x-ray transform with the fan-beam setting that takes y_{os} into account is summarized in Appendix E. For details of the CPU and GPU implementations in C see Ref. 1.

The computational complexity of this algorithm is $O(N_v)$ for each pixel, and the overall complexity is $O(N_x \cdot N_y \cdot N_v)$. In its parallel version, the contribution of the scans from different directions to the same pixel can be computed in parallel, and then summed together. Therefore, the complexity of this algorithm is $O(1)$ per parallel thread, given that the time for the summation is relatively negligible. In this aspect, this algorithm is suitable for parallel computing of the adjoint x-ray transform. Finally, although the algorithm is presented for the fan-beam scanning setting, it can be easily modified for other scanning settings.

III.B. 3D version

This algorithm can be easily extended for the 3D case. Without loss of generality, let us only consider the cone-beam scan. For the notations, let (x_c, y_c, z_c) be the coordinate of the center of the pixel, (n_a, n_b) and (d_y, d_z) be the number of detectors and the detector width along the y-direction and the z-direction (two orthonormal directions of the detection plane),

and ϕ be the scan angle. Without loss of generality, let us also assume $dx = dy = 1$.

Similar to the 2D case, for a fixed voxel and a fixed scanning direction, the contribution from this scan to this voxel only comes from the detectors that lie in the shadow region on the 2D detection plane that is bounded all around by the beams passing through the corners of the voxel. Therefore, we compute the rotated voxel coordinate through (3.1), find the detection bound along y-direction and z-direction through

$$\begin{cases} n_{a,\min} = \lfloor [(y_r - 1) \cdot SD / (x_r + SO)] / d_y + N_a / 2 \rfloor \\ n_{a,\max} = \lfloor [(y_r + 1) \cdot SD / (x_r + SO)] / d_y + N_a / 2 \rfloor \end{cases} \quad (3.3)$$

and

$$\begin{cases} n_{b,\min} = \lfloor [(z_c - \Delta) \cdot SD / (x_r + SO)] / d_z + N_b / 2 \rfloor \\ n_{b,\max} = \lfloor [(z_c + \Delta) \cdot SD / (x_r + SO)] / d_z + N_b / 2 \rfloor \end{cases}$$

$$\text{with } \Delta = \sqrt{1 + dz^2/2}, \quad (3.4)$$

and then compute the intersection length for the detectors only within this localized region. Please note that for helical scans, we simply need to subtract the incremental z-change from z_c to get the transformed coordinate z_r .

Here, the method for computing the length of the intersection line of the infinitely narrow beam with a voxel in 3D is rather preliminary. That is, we calculate the intersecting coordinate of the surfaces of the voxel with the beam, and then compute the distance between two intersecting points. The dedicated geometric formulas will be searched in the future study.

For the convenience of implementation, the algorithm for the 3D adjoint x-ray transform with the helical-scan setting that takes y_{os} into account is summarized in Appendix F. For details of the CPU and GPU implementations in C see Ref. 1.

Again, the computational complexity of this algorithm for 3D is still $O(N_v)$ for each voxel, and the overall complexity is $O(N_x \cdot N_y \cdot N_z \cdot N_v)$. When considering the parallel version, the complexity of this algorithm in 3D is also $O(1)$ per

parallel thread, provided that the time for the summation of the contributions from N_v views to a voxel is relatively negligible.

III.C. Generalization for the finite-size beam

The extension of this algorithm for the finite-size beam is again straightforward. That is, for a fixed voxel and a fixed scan direction, the detectors with nontrivial contribution to the voxel can be localized through the same method, e.g., Eqs. (3.2) and (3.4), and then the efficient geometric formulas need to be supplied.

Here, we implement the 2D version for the finite-size beam based on the geometric formula in Appendix B for computing the intersection area of the beam with each nontrivially intersecting pixel. For details of the CPU and GPU implementations in C see Ref. 1.

IV. RESULTS

The following results are based on Intel E6850 CPU (3.00 GHz), and the parallel computing results are based on NVIDIA GeForce GTX 460. The results can be reproduced with the online codes.¹

The 3D computations are performed with a typical cone-beam setting. The parameters of the cubic domain with the side length 250 are $N_x = 256$, $N_y = 256$, $N_z = 192$, $dx = 0.98$, $dy = 0.98$, $dz = 1.30$; the parameters of the detection plane are $N_a = 512$, $N_b = 384$, $d_y = 0.776$, $d_z = 0.776$; the parameters of the scan are $SO = 1000$, $SD = 1500$, $N_v = 668$. And the 2D fan-beam computations are preformed with the central slice of the detection plane, i.e., $N_x = 256$, $N_y = 256$, $dx = 0.98$, $dy = 0.98$, $N_d = 512$, $d_y = 0.776$, $SO = 1000$, $SD = 1500$, and $N_v = 668$. Here, the unit of length is millimeter.

For the x-ray transform, the comparison of computational efficiency in 2D and 3D is performed between the popular Siddon's algorithm² and the proposed algorithm, and also between their parallel versions, for the case with infinitely narrow beams. In addition, the result with the proposed algorithm for the finite-size beam in 2D is also presented. The Siddon algorithm is parallelized for each beam, and each parallel thread carries out the computation of the intersection of this beam with the 2D or 3D domain. The results are summarized in Table I. Here, the 2D parallel computing result suggests that the new algorithm is an order of magnitude faster than the Siddon algorithm, which is consistent with the difference in their computational complexity per parallel thread, i.e., $O(1)$ versus $O(N_x)$. However, the gain in speed is reduced to the two–three folds in 3D, which is possibly due to the saturated parallelizability of the current GPU card (GeForce GTX 460).

For the adjoint x-ray transform, the computation in 2D and 3D is performed with the proposed algorithm for the case with infinitely narrow beams. Please note that the current method for computing the intersection length of a voxel with a beam in 3D is rather preliminary, and a more efficient method should improve the speed considerably. In addition, the result with the proposed algorithm for the finite-size beam in 2D is also presented. The results are summarized in Table I.

TABLE I. Computational time (Unit: seconds). The first, second, and third columns represent the benchmark algorithm (Siddon), the proposed new algorithm (New), and the proposed new algorithm for the finite-size beam (New-FB), respectively, for the x-ray transform. The fourth and fifth columns represent the proposed new algorithm (Adjoint) and the proposed new algorithm for the finite-size beam (Adjoint-FB), respectively, for the adjoint x-ray transform. For rows, “2D” and “2D-p” denote the 2D computations and their parallel versions for the given fan-beam setting; “3D” and “3D-p” denote the 3D computations and their parallel versions for the given cone-beam setting.

	Siddon	New	New-FB	Adjoint	Adjoint-FB
2D	4.8	2.6	8.6	4.7	6.7
2D-p	0.61	0.05	0.18	0.07	0.09
3D	3102	1960	...	12 770	...
3D-p	83	36	...	238	...

V. CONCLUSION AND FUTURE WORK

In this paper, a faster algorithm than the popular Siddon's algorithm for computing the x-ray transform has been proposed, particularly for the parallel computing of the x-ray transform. And this fast algorithm can be extended for the finite-size beam. On the other hand, a fast and highly parallelizable algorithm has been proposed for computing the adjoint x-ray transform, which can also be extended for the finite-size beam. The proposed algorithms are suitable for parallel computing in the sense that their computational complexity is $O(1)$ per parallel thread.

The proposed fast algorithms will serve as the critical building blocks to develop the iterative reconstruction methods for the low-dose CT imaging, such as breast imaging,^{21,22} cardiac imaging,²³ four-dimensional CT,²⁴ multienergy CT,²⁵ and radiation therapy.^{26–29}

The future work will also include the search of an efficient geometric formula for computing the intersecting volume of a finite-size beam with a voxel in order to extend the proposed algorithm for the 3D adjoint x-ray transform with the finite-size beam. In addition, more efficient geometric formulas for computing the intersecting length and area will be researched in order to further accelerate the proposed algorithm for computing the x-ray transform with the finite-size beam or the adjoint x-ray transform, especially in 3D.

ACKNOWLEDGMENT

This work is partially supported by National Institutes of Health (NIH)/NIBIB Grant No. EB013387.

APPENDIX A: INTERSECTING LINE FORMULA IN 2D

Here, we present an efficient computation of the length of the intersection line of the infinitely narrow beam with a pixel, given the line formula $ax+by=c$ with normalized coefficients (i.e., $a^2+b^2=1$), and (x_0, y_0) as the coordinate of the pixel center. Without loss of generality, we still assume $dx = dy = 1$.

FIG. 7. An efficient geometric formula for computing the length of the intersection line of the infinitely narrow beam with a pixel. The computation can be conveniently divided into three cases according to the distance from the pixel center to the beam.

The distance from the pixel center to the beam is utilized to divide the computation into three cases (Fig. 7). Let $a_2 = |a|$ and $b_2 = |b|$. The distance from the pixel center to the beam $d = |ax_0 + by_0 - c|$, $d_+ = (a_2 + b_2)/2$, and $d_- = |a_2 - b_2|/2$. Then the geometric formula is

$$l = \begin{cases} 1/b_2 & d < d_-, a_2 < b_2 \\ 1/a_2 & d < d_-, a_2 \geq b_2 \\ (d_+ - d)/(a_2 b_2) & d_- \leq d < d_+ \\ 0 & d_+ \leq d \end{cases} \quad (A1)$$

APPENDIX B: INTERSECTING AREA FORMULA IN 2D

Here, we present an efficient computation of the area of the intersection region of the finite-size beam with a pixel, given two boundary lines of the beam prescribed by $a_1 x + b_1 y = c_1$ and $a_2 x + b_2 y = c_2$ with normalized coefficients (i.e., $a_1^2 + b_1^2 = a_2^2 + b_2^2 = 1$), and (x_0, y_0) as the coordinate of the pixel center (Figs. 8 and 9). Without loss of generality, we still assume $dx = dy = 1$.

FIG. 8. When two sides of the beam are at the different side with respect to the pixel center, the area of the intersection of the beam with the pixel [the colored region in (a)] is equal to the area sum of two parallelograms [two colored regions in (b)], which can be individually computed based on the method in Fig. 7.

FIG. 9. When two sides of the beam are at the same side with respect to the pixel center, the area of the intersection of the beam with the pixel [the colored region in (a)] is equal to the absolute area difference of two parallelograms [two colored regions in (b)], which can be individually computed based on the method in Fig. 7.

Based on the method in Appendix A, we first compute the length of each boundary line of the beam l , and then each area can be individually computed by

$$A = \begin{cases} ld & d < d_- \\ 1/2 - l(d_+ - d)/2, & d_- \leq d < d_+ \\ 1/2 & d_+ \leq d \end{cases} \quad (B1)$$

Finally, the total area of the intersection region is given by

$$A = |A_1 - \text{sgn}((a_1 x_0 + b_1 y_0 - c_1)(a_2 x_0 + b_2 y_0 - c_2))A_2|. \quad (B2)$$

APPENDIX C: ALGORITHM FOR THE 2D X-RAY TRANSFORM

For the convenience of implementation, the proposed algorithm for the 2D x-ray transform is summarized as follows for the case with $|x_2 - x_1| > |y_2 - y_1|$, where the C language is used with the convention that the indices start from 0 instead of 1. Please note that $dx = dy = 1$ is assumed. For details of the CPU and GPU implementations in C see Ref. 1

```

y = 0; nx2 = nx/2; ny2 = ny/2;
if (ABS(x1-x2)>ABS(y1-y2))
{
  ky = (y2-y1)/(x2-x1);
  for (ix = 0; ix<nx; ix++)
  {
    xx1 = ix-nx2; xx2 = xx1+1;
    if (slope>= 0)
    {
      yy1 = y1+ky*(xx1-x1)+ny2; yy2 = y1+ky*(xx2-x1)+ny2;
    }
    else
    {
      yy1 = y1+ky*(xx2-x1)+ny2; yy2 = y1+ky*(xx1-x1)+ny2;
    }
    c1 = floor(yy1); c2 = floor(yy2);

    if (c2 == c1)
    {
      if (c1>= 0&&c1<= ny-1)
      {
        iy = c1; y+ = sqrt(1+ky*ky)*x[ix*nx+ix];
      }
    }
  }
}

```



```

else
{
  if (c1 >= -1 && c1 <= ny-1)
  {
    if (c1 >= 0)
    {
      iy = c1; y+ = ((c2-yy1)/(yy2-yy1))*sqrt(1+ky*ky)*
      x[iy*nx+ix];}
      if (c2 <= ny-1)
      {
        iy = c2; y+ = ((yy2-c2)/(yy2-yy1))*sqrt(1+ky*ky)*
        x[iy*nx+ix];}
      }
    }
  }
}

```

```

*****

```

```

*****

```

```

y = 0; nx2 = nx/2; ny2 = ny/2; nz2 = nz/2;
if (ABS(x1-x2) > ABS(y1-y2))
{
  ky = (y2-y1)/(x2-x1);
  kz = (z2-z1)/(x2-x1);
  for (ix = 0; ix < nx; ix++)
  {
    xx1 = ix-nx2; xx2 = xx1+1;
    if (ky >= 0)
    {
      yy1 = y1+ky*(xx1-x1)+ny2; yy2 = y1+ky*(xx2-x1)+ny2;
    }
    else
    {
      yy1 = y1+ky*(xx2-x1)+ny2; yy2 = y1+ky*(xx1-x1)+ny2;
    }
    cy1 = floor(yy1); cy2 = floor(yy2);
    if (kz >= 0)
    {
      zz1 = (z1+kz*(xx1-x1))/dz+nz2; zz2 = (z1+kz*(xx2-x1))/dz+nz2;
    }
    else
    {
      zz1 = (z1+kz*(xx2-x1))/dz+nz2; zz2 = (z1+kz*(xx1-x1))/dz+nz2;
    }
    cz1 = floor(zz1); cz2 = floor(zz2);

    if (cy2 == cy1)
    {
      if (cy1 >= 0 && cy1 <= ny-1)
      {
        if (cz2 == cz1)
        {
          if (cz1 >= 0 && cz1 <= nz-1)
          {
            iy = cy1; iz = cz1; y+ = sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];
          }
          else
          {
            if (cz1 >= -1 && cz1 <= nz-1)
            {
              rz = (cz2-zz1)/(zz2-zz1);
              if (cz1 >= 0)
              {
                iy = cy1; iz = cz1; y+ = rz*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];
              }
              if (cz2 <= nz-1)
              {
                iy = cy1; iz = cz2; y+ = (1-rz)*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];
              }
            }
          }
        }
      }
    }
    else
    {
      if (cy1 >= -1 && cy1 <= ny-1)
      {
        if (cz2 == cz1)
        {
          if (cz1 >= 0 && cz1 <= nz-1)
          {
            ry = (cy2-yy1)/(yy2-yy1);
            if (cy1 >= 0)
            {
              iy = cy1; iz = cz1; y+ = ry*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];
            }
            if (cy2 <= ny-1)
            {
              iy = cy2; iz = cz1; y+ = (1-ry)*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];
            }
          }
        }
      }
    }
  }
}

```

APPENDIX D: ALGORITHM FOR THE 3D X-RAY TRANSFORM

For the convenience of implementation, the proposed algorithm for the 3D x-ray transform is summarized as follows for the case with $|x_2 - x_1| > |y_2 - y_1|$ and $|x_2 - x_1| > |z_2 - z_1|$, where the C language is used with the convention that the indices start from 0 instead of 1. Please note that $dx = dy = 1$ is assumed. For details of the CPU and GPU implementations in C see Ref. 1.

```

else
{ if(cz1> = -1&&cz1< = nz-1)
    { ry = (cy2-yy1)/(yy2-yy1);rz = (cz2-zz1)/(zz2-zz1);
      if(ry>rz)
        { if(cy1> = 0&&cz1> = 0)
            { iy = cy1;ziz = cz1;y+ = rz*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];}
          if(cy1> = 0&&cz2< = nz-1)
            { iy = cy1;ziz = cz2;y+ = (ry-rz)*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];}
          if(cy2< = ny-1&&cz2< = nz-1)
            { iy = cy2;ziz = cz2;y+ = (1-ry)*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];}
        }
      else
        { if(cy1> = 0&&cz1> = 0)
            { iy = cy1;ziz = cz1;y+ = ry*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];}
          if(cy2< = ny-1&&cz1> = 0)
            { iy = cy2;ziz = cz1;y+ = (rz-ry)*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];}
          if(cy2< = ny-1&&cz2< = nz-1)
            { iy = cy2;ziz = cz2;y+ = (1-rz)*sqrt(1+ky*ky+kz*kz)*x[iz*ny*nx+iy*nx+ix];}
        }
      }
    }
  }
}

```

APPENDIX E: ALGORITHM FOR THE 2D ADJOINT X-RAY TRANSFORM

For the convenience of implementation, the proposed algorithm for the 2D adjoint x-ray transform with the fan-beam setting is summarized as follows, where the C language is used with the convention that the indices start from 0 instead of 1. Here, the detector offset distance y_{os} is taken into account. For details of the CPU and GPU implementations in C see Ref. [1].

```

x = 0;nd2 = nd/2;
for(iv = 0;iv<nv;iv++)
{
  xr = cos_phi[iv]*xc+sin_phi[iv]*yc;
  yr = -sin_phi[iv]*xc+cos_phi[iv]*yc;
  tmp = SD/((xr+SO)*d_y);
  nd_max = floor((yr+1)*tmp-y_os+nd2);
  nd_min = floor((yr-1)*tmp-y_os+nd2);
  for(id = MAX(0,nd_min);id<= MIN(nd_max,nd-1);id++)
  {if(l>0){x+=1*y[iv*nd+id];}}
}

```

APPENDIX F: ALGORITHM FOR THE 3D ADJOINT X-RAY TRANSFORM

For the convenience of implementation, the proposed algorithm for the 3D adjoint x-ray transform with the helical-scan setting is summarized as follows, where the C language is used with the convention that the indices start from 0 instead of 1. Here, the detector offset distance y_{os} is taken into

account. For details of the CPU and GPU implementations in C see Ref. 1.

```

x = 0;na2 = na/2;nb2 = nb/2;nd = na*nb;
d = sqrt((1+dz*dz)/2);
for(iv = 0;iv<nv;iv++)
{
  xr = cos_phi[iv]*xc+sin_phi[iv]*yc;
  yr = -sin_phi[iv]*xc+cos_phi[iv]*yc;
  zr = zc-z[iv];
  tmp = SD/((xr+SO)*d_y);
  na_max = floor((yr+1)*tmp-y_os+na2);
  na_min = floor((yr-1)*tmp-y_os+na2);
  tmp = SD/((xr+SO)*d_z);
  nb_max = floor((zc+d)*tmp+nb2);
  nb_min = floor((zc-d)*tmp+nb2);
  for(ib = MAX(0,nb_min);ib<= MIN(nb_max,nb-1);ib++)
  {
    for(ia = MAX(0,na_min);ia<= MIN(na_max,na-1);ia++)
    {
      if(l>0){x+=1*y[iv*nd+ib*na+ia];}
    }
  }
}

```

a) Electronic mail: hao.gao@emory.edu

¹<https://sites.google.com/site/fastxraytransform>.

²R. L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional CT array," *Med. Phys.* **12**, 252–255 (1985).

³G. H. Chen, J. Tang, and S. H. Leng, “Prior image constrained compressed sensing (PICCS): A method to accurately reconstruct dynamic CT images from highly undersampled projection data sets,” *Med. Phys.* **35**, 660–663 (2008).

⁴X. Jia, Y. Lou, R. Li, W. Y. Song, and S. B. Jiang, "Cone beam CT reconstruction from undersampled and noisy projection data via total variation," *Med. Phys.* **37**, 1757–1760 (2010).

- ⁵E. Y. Sidky and X. C. Pan, "Image reconstruction in circular cone-beam computed tomography by constrained, total-variation minimization," *Phys. Med. Biol.* **53**, 4777–4807 (2008).
- ⁶J. Wang, T. F. Li, and L. Xing, "Iterative image reconstruction for CBCT using edge-preserving prior," *Med. Phys.* **36**, 252–260 (2009).
- ⁷H. Y. Yu and G. Wang, "Compressed sensing based interior tomography," *Phys. Med. Biol.* **54**, 2791–2805 (2009).
- ⁸S. S. Vedam, P. J. Keall, V. R. Kini, H. Mostafavi, H. P. Shukla, and R. Mohan, "Acquiring a four-dimensional computed tomography dataset using an external respiratory signal," *Phys. Med. Biol.* **48**, 45–62 (2003).
- ⁹D. Low, M. Nystrom, E. Kalinin, P. Parikh, J. Dempsey, J. Bradley, S. Mutic, S. Wahab, T. Islam, G. Christensen, D. Polite, and B. Whiting, "A method for the reconstruction of four-dimensional synchronized CT scans acquired during free breathing," *Med. Phys.* **30**, 1254–1263 (2003).
- ¹⁰P. J. Keall, G. Starkschall, H. Shukla, K. M. Forster, V. Ortiz, C. W. Stevens, S. S. Vedam, R. George, T. Guerrero, and R. Mohan, "Acquiring 4D thoracic CT scans using a multislice helical method," *Phys. Med. Biol.* **49**, 2053–2067 (2004).
- ¹¹E. Rietzel, T. Pan, and G. T. Chen, "Four-dimensional computed tomography: Image formation and clinical protocol," *Med. Phys.* **32**, 874–889 (2005).
- ¹²T. Li, E. Schreiber, B. Thorndyke, G. Tillman, A. Boyer, A. Koong, K. Goodman, and L. Xing, "Radiation dose reduction in four-dimensional computed tomography," *Med. Phys.* **32**, 3650–3660 (2005).
- ¹³H. Zhao and A. J. Reader, "Fast ray-tracing technique to calculate line integral paths in voxel arrays," in *Proceedings of the IEEE Nuclear Science Symposium and Medical Imaging Conference* (2003), pp. 2808–2812.
- ¹⁴B. De Man and S. Basu, "Distance-driven projection and backprojection in three dimensions," *Phys. Med. Biol.* **49**, 2463–2475 (2004).
- ¹⁵Y. Long, J. A. Fessler, and J. M. Balter, "3-D forward and back-projection for x-ray CT using separable footprints," *IEEE Trans. Med. Imaging* **29**, 1839–1850 (2010).
- ¹⁶F. Xu and K. Mueller, "Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware," *IEEE Trans. Nucl. Sci.* **52**, 654–663 (2005).
- ¹⁷X. Zhao, J. Bian, E. Y. Sidky, S. Cho, P. Zhang, and X. Pan, "GPU-based 3D cone-beam CT image reconstruction: Application to micro CT," in *Proceedings of the IEEE Nuclear Science Symposium and Medical Imaging Conference* (2007), pp. 3922–3925.
- ¹⁸M. Knaup, S. Steckmann, and M. Kachelrieß, "GPU-based parallel-beam and cone beam forward- and backprojection using CUDA," in *Proceedings of the IEEE Nuclear Science Symposium and Medical Imaging Conference* (2008), pp. 5153–5157.
- ¹⁹W. Xu and K. Mueller, "A performance-driven study of regularization methods for GPU-accelerated iterative CT," in *Proceedings of the High Performance Image Reconstruction Workshop* (Beijing, China, 2009).
- ²⁰X. Jia, B. Dong, Y. Lou, and S. B. Jiang, "GPU-based iterative cone-beam CT reconstruction using tight frame regularization," *Phys. Med. Biol.* **56**, 3787–3807 (2011).
- ²¹J. L. Ducote and S. Molloy, "Quantification of breast density with dual energy mammography: An experimental feasibility study," *Med. Phys.* **37**, 793–801 (2010).
- ²²I. Sechopoulos, "X-ray scatter correction method for dedicated breast computed tomography," *Med. Phys.* **39**, 2896–2903 (2012).
- ²³J. Hsieh, J. Londt, M. Vass, J. Li, X. Tang, and D. Okerlund, "Step-and-shoot data acquisition and reconstruction for cardiac x-ray computed tomography," *Med. Phys.* **33**, 4236–4248 (2006).
- ²⁴H. Gao, J. F. Cai, Z. Shen, and H. Zhao, "Robust principal component analysis-based four-dimensional computed tomography," *Phys. Med. Biol.* **56**, 3181–3198 (2011).
- ²⁵H. Gao, H. Yu, S. Osher, and G. Wang, "Multi-energy CT based on a prior rank, intensity and sparsity model (PRISM)," *Inverse Probl.* **27**, 115012 (2011).
- ²⁶S. B. Jiang, J. Wolfgang, and G. S. Mageras, "Quality assurance challenges for motion-adaptive radiation therapy: Gating, breath holding, and four dimensional computed tomography," *Int. J. Radiat. Oncol., Biol., Phys.* **71**, S103–S107 (2008).
- ²⁷T. Niu, A. Al-Basheer, and L. Zhu, "Quantitative cone-beam CT imaging in radiation therapy using planning CT as a prior: First patient studies," *Med. Phys.* **39**, 1991–2000 (2012).
- ²⁸J. Wang, T. Li, Z. Liang, and L. Xing, "Dose reduction for kilovoltage cone-beam computed tomography in radiation therapy," *Phys. Med. Biol.* **53**, 2897–2909 (2008).
- ²⁹L. Xing, B. Thorndyke, E. Schreiber, Y. Yang, T. F. Li, G. Y. Kim, G. Luxton, and A. Koong, "Overview of image-guided radiation therapy," *Med. Dosim.* **31**, 91–112 (2006).