Fast Implementations of Algebraic Methods for Three-Dimensional Reconstruction from Cone-Beam Data

Klaus Mueller,* *Member, IEEE*, Roni Yagel, *Member, IEEE*, and John J. Wheller

***Abstract*— The prime motivation of this work is to devise techniques that make the algebraic reconstruction technique (ART) and related methods more efficient for routine clinical use, while not compromising their accuracy. Since most of the computational effort of ART is spent for projection/backprojection operations, we first seek to optimize the projection algorithm. Existing projection algorithms are surveyed and it is found that these algorithms either lack accuracy or speed, or are not suitable for cone-beam reconstruction. We hence devise a new and more accurate extension to the splatting algorithm, a well-known voxel-driven projection method. We also describe a new three-dimensional (3-D) ray-driven projector that is considerably faster than the voxel-driven projector and, at the same time, more accurate and perfectly suited for the demands of cone beam. We then devise caching schemes for both ART and simultaneous ART (SART), which minimize the number of redundant computations for projection and backprojection and, at the same time, are very memory conscious. We find that with caching, the cost for an ART projection/backprojection operation can be reduced to the equivalent cost of 1.12 projections. We also find that SART, due to its image-based volume correction scheme, is considerably harder to accelerate with caching. Implementations of the algorithms yield runtime ratios $T_{\mathrm{SART}}/T_{\mathrm{ART}}$ between 1.5 and 1.15, depending on the amount of caching used.**

***Index Terms*— Algebraic reconstruction technique (ART), computed tomography (CT), cone-beam reconstruction, three-dimensional reconstruction.**

I. Introduction

In this paper we explore several techniques that are geared toward making algebraic reconstruction methods, such as the algebraic reconstruction technique (ART) or simultaneous ART (SART), more efficient, without making any compromises in terms of accuracy. Although our discussion is mostly focussed on the three-dimensional (3-D) cone-beam case, many of the presented principles are also valid in the two-dimensional (2-D) fan-beam and in the 2-D and 3-D parallel-beam case.

Manuscript received February 6, 1998; revised February 22, 1999. This work supported in part by General Electric under CCH Grant 950925 and OSU Grant 732025. The Associate Editor responsible for coordinating the review of this paper and recommending its publication was R. Huesman. *Asterisk indicates corresponding author.*

*K. Mueller is with the Department of Computer and Information Science and the Department of Pediatrics, The Ohio State University, Columbus, OH 43210 USA.

R. Yagel is with the Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210 USA.

J. J. Wheller is with the Department of Pediatrics, The Ohio State University, Columbus, OH 43210 USA.

Publisher Item Identifier S 0278-0062(99)06614-8.

This paper is a continuation of a previous paper [13] that dealt with accuracy issues for cone-beam reconstruction. Please refer to that paper for background and references on ART and related methods. ART is an iterative method, reconstructing an object by a sequence of reconstruction grid projections and backprojections. Hence, if one wants to make ART faster and more competitive with the more common filtered backprojection (FBP) methods, one must keep the number of iterations small and, at the same time, reduce the overall cost for the projection–backprojection operations.

Several groups of researchers have worked on reducing the number of iterations for ART. An important aspect in achieving this goal is the order in which the projections are accessed in the iterative reconstruction procedure. In a recent study, Mueller [14] contrasted various previously published projection access schemes with a new scheme, the weighted distance scheme. It was found that for low-contrast objects, such as the Shepp–Logan brain phantom [17], usually three to four iterations are sufficient for good reconstruction quality. The choice of the relaxation coefficient λ is another important parameter, which was studied by Herman and Meyer [7]. The impact of λ was also examined by us in [13], along with the effects of various other ART parameters, such as grid initialization and the correction algorithm. There, it was confirmed that within three iterations, a reconstruction of a quality close to the maximum can be obtained.

Since most of the computational expense of ART is spent for projection and backprojection, we must improve the speed of ART's projection engine. It turns out that the computational cost of this projection engine is greatly affected by the perspective cone-beam projection. In the following sections, we will give a detailed description of two new highly accurate projection algorithms, one voxel-driven and one-ray-driven, and analyze their efficiency in both the parallel-beam and cone-beam setting. Although other voxel-driven projectors [19] and ray-driven projectors [9], [10] have been described, these algorithms are only efficient for the parallel-beam case or do not allow the stretched interpolation kernels prescribed in [13] as necessary for accurate cone-beam reconstruction. Furthermore, our voxel-driven perspective projection algorithm is considerably more accurate than the one described by Westover [19]. Our ray-driven algorithm, on the other hand, is a 3-D extension of the 2-D algorithm proposed by Hanson and Wecksung [6]. However, a fast projection algorithm is not enough. We must reduce the actual complexity of the overall projection–backprojection framework. Ideally, we only want

to do the computational equivalent of one projection operation per image instead of one projection and one backprojection. This can only be achieved by reusing some of the earlier computed results for later calculations, which is a technique termed caching. Our paper will give caching schemes for both ART and SART which will bring the computational cost of these popular algebraic methods closer to the theoretical cost of FBP methods.

The outline of this paper is as follows. Section II gives some background on previous ART implementations. Section III then describes a voxel-driven projection algorithm for cone beam that is more accurate for perspective projection than existing ones, but does not improve the state of the art in terms of speed. Section IV gives a new ray-driven projection algorithm for cone-beam ART that is both accurate and efficient. Section V details various caching schemes to speed ART and SART. Finally, Section VI puts everything together and presents a variety of results obtained with our ART testbed software.

II. Preliminaries and Previous Work

Please refer to our paper [13] for notations and relevant background on algebraic methods. There we have derived that the number of necessary projections for a single source rotating in a circular orbit is $S_{\mathrm{ART}} = 0.67n$ where n is length of the cubic reconstruction grid. The single-source, circular orbit configuration gives rise to a spherical reconstruction region. The twin-cone source arrangement described by Schlindwein [16] (see also the conclusions of [13]), on the other hand, reconstructs a cylindrical region of interest. Here, the necessary number of projections is

$$S_{\mathrm{ART}} = \frac{(1/4)\pi n^3}{n^2} = 0.78n \tag{1}$$

(half-) projections per detector to reconstruct the cylindrical reconstruction region sandwiched between the two circular orbits.

This number S_{ART}, just given, ensures that the ART equation system is determined. However, ART can also be applied without change if S is smaller or greater than this number. In this context, an interesting observation was made by Guan and Gordon [5] for the 2-D case. They showed that, in theory, the number of required projections in ART is about half the number of the projections required for FBP. More precisely $S_{\mathrm{FBP}} = 1.57n$. This happens because the Fourier Slice Theorem arranges the projections onto a polar grid in frequency space and, in order to provide adequate sampling in the periphery, one must oversample in the interior frequency regions. This may be part of the reason for the strength of ART in the limited projection data case.

Algebraic methods typically represent the volume grid as a collection of spherical interpolation kernels, placed at the voxel positions and scaled by the voxel values. This ensemble of scaled voxel kernels then makes up a continuous representation of the volume. Since each voxel kernel projects and backprojects in the same way (as a so-called footprint), many authors [6], [9], [10], [19] precompute this kernel projection and store it in a lookup table. (An entry in the footprint table is thus due to the integral of the ray traversing the voxel kernel at the respective table position.) Then, all one has to do is scale the footprint by the voxel value (in grid projection) or the ART correction factors (in grid backprojection). Two approaches to perform this projection have been proposed. One way is the voxel-based approach in which one maps all voxel footprints to the screen, scaled by the voxel value, where they accumulate into a projection image. This is done in the splatting approach for volume rendering, devised by Westover [19]. In backprojection, the voxel footprints are again mapped to the screen, but this time they pick up (corrective) energy, instead of emitting it. In the second approach, one can use rays to intersect the footprint tables in volume space, again scale the indexed value by the voxel value, and accumulate the density integrals (or distribute the correction factors) ray by ray. Since this is inherently also a splatting approach, we term this method ray-driven splatting, while the voxel-based method will be referred to as voxel-driven splatting. Since the voxel-driven approach produces a whole image or at least an image region at a time, it only makes sense to use it in conjunction with an image-based correction algorithm, such as SART. The ray-driven approach, on the other hand, processes one pixel at a time and can thus be used with either the pixel-based ART or the image-based SART.

The preintegrated footprint method has several advantages.

1) The ray integrals are calculated very accurately, since each footprint table entry can be integrated analytically or with good quadrature. Thus, the accumulation of the footprints on the image plane is very close to an analytic integration of the volume.
2) The complexity for interpolation is reduced from $O(n^3)$ in volume space (as required for raycasting) to $O(n^2)$ in image space. Fast incremental algorithms can then be used to index the footprint tables in volume space (in ray-driven splatting) or image space (in voxel-driven splatting).
3) Due to these fast projection algorithms we can afford interpolation kernels that are larger but have superior, *sinc*-like, frequency characteristics, such as the Bessel–Kaiser function devised by Lewitt [9].

Before we describe the existing projection algorithms in further detail, let us recall (from [13]) that for accurate cone-beam reconstruction it is necessary to stretch and scale the interpolation kernels, depending on their distance from the source to prevent aliasing in projection and backprojection. This stretching occurs along two orthogonal axes, perpendicular to the direction of the ray(s) traversing the interpolation kernel. In [13], the stretching of the 3-D interpolation function was approximated by a stretching of the 2-D footprint (the Appendix justifies this approximation). Let us now clarify these concepts in more detail. In 3-D, the rays emanate from the source along a curved rayfront of equal grid sampling rate in a raster composed of two orthogonal sets of sheets (see also Fig. 3). Each 3-D ray is part of one sheet in each set and is defined by the intersection of these two sheets. Constrained by the divergent cone-beam geometry, the distance of two adjacent sheets within a set increases with distance from the

source. This means that, depending on the location of a kernel with respect to each sheet set, its 2-D footprint must undergo different distortions in the two principal coordinate axes. Thus, our projection algorithm must be able to stretch a footprint by different amounts in the two-sheet raster directions.

While the concept of representing a volume by a field of interpolation kernels and preintegrating a 3-D kernel into a 2-D footprint is common to all existing splatting implementations, the strategy chosen to map the footprint table onto the screen (in the voxel-driven approach) or to map the rays into the footprint table (in the ray-driven approach) varies. The mapping task is facilitated since we only use spherically symmetric kernels and cubic grids which yield a circular footprint. For voxel-driven splatting, both Westover [19] and Matej [10] simply map the circular footprint to the projection screen for one voxel and use incremental shifts for the remaining voxels at that projection angle. This, however, is only correct for parallel projections, since in perspective projection the elliptical shape and size of the footprint is different for every voxel. (More detail is given in Section III.) In the case of ray-driven splatting we again assume a spherically symmetric interpolation kernel. Here, the approaches are more diverse. For instance, Lewitt [9] computes the magnitude of the crossproduct of the ray unit vector with the vector from a point on the ray to the voxel center. This yields the perpendicular distance of the ray to the voxel center which can be used to index a one-dimensional (1-D) footprint table storing the radially symmetric projection of the 3-D kernel. An efficient incremental algorithm can then be used to find all other voxel distances along the ray. This approach, however, is not appropriate for cone-beam reconstruction, as it does not allow independent footprint stretching in the two ray sheet directions, as is necessary for accurate cone-beam reconstruction. In another approach, Matej and Lewitt [11] decompose the voxel grid into a set of 2-D slices. Here, the orientation of the slices is that orientation most parallel to the image plane. Recall that a footprint is the preintegrated kernel function in the direction of a ray, thus, a footprint is not necessarily planar with the slice planes. The authors project this footprint function onto a slice plane which results in an elliptical footprint. Since in parallel projection all rays for a given projection angle have the same angle with the volume slices, this remapped elliptical footprint can be used for all slices and all rays that are spawned for a given projection orientation. Simple incremental algorithms can be designed to trace a ray across the volume slices, computing all indexes into the elliptical footprints that are intersected. However, for perspective projection, every ray has a different orientation, necessitating a footprint remapping for every ray, which is inefficient both to compute on the fly and to store. A more appropriate approach was outlined for the 2-D case by Hanson and Wecksung [6]. These authors model a 2-D ray as an implicit line equation. If one runs a line parallel to the ray across the center of a given voxel, then the offset difference of the equations of these two lines yield the perpendicular distance of the ray to the voxel center, which then can be used to index a 1-D footprint table. Our ray-driven approach is a 3-D extension of this algorithm, optimized for speed, that enables the efficient use of the same footprint table for all projection rays everywhere in the volume.

The splatting techniques are in contrast to the fast interpolative grid-traversal methods proposed by Siddon [18] and Joseph [8]. These methods employ nearest neighbor interpolation or bilinear interpolation, respectively, which, however, are functions far inferior to the ones that can efficiently be used in splatting. Matej has conducted a study that compared ART using Sidden's algorithm and ART using Bessel kernels and found that the latter produced considerably better reconstruction results [12]. In addition, the splatting approaches, due to their efficient implementation, do not need to be any costlier than the interpolative grid traversal schemes as far as the cost-per-kernel crosssection is concerned. The added expense comes from the fact that these better interpolation kernels must have a larger extent (typically a diameter of 4.0, as compared to an extent of 2.0 and 1.0 for the bilinear and nearest neighbor kernel, respectively).

III. An Accurate Voxel-Driven Splatting Algorithm for Cone-Beam ART

Let us first introduce some terminology. As suggested by Crawfis and Max [2], we can think of the interpolation kernel footprint as a polygon with a superimposed texture map that is placed in object (volume) space. Here, the texture map is given by the projected kernel function, i.e., the array of line integrals (the ones stored in the footprint table). For the remainder of our discussion we will refer to the footprint in object space as the footprint polygon, while the projection of the footprint polygon onto the image plane will be called the footprint image. Recall that splatting accumulates the same value in a pixel on the image plane as a ray would accumulate when traversing the volume. Thus, when projecting the footprint polygon to obtain the line integral for the pixel in the footprint image, we must ensure that we position the footprint polygon orthogonal to the direction of the sight ray in object space. The line integrals are retrieved from the footprint table by indexing it at the ray-footprint polygon intersection point. Thus, for splatting to be accurate, the 2-D footprint must be mapped to the pixel as if the ray emanating from the pixel had traversed it at a perpendicular angle. Only then does the looked-up preintegrated integral match the true kernel integration of the ray. Westover's perspective extension to voxel-driven splatting ([19]) violates this condition in three instances.

- He does not align the footprint polygon perpendicularly to the voxel center ray when calculating the projected screen extent. Rather, he aligns it parallel to the screen and stretches it according to the perspective viewing transform.
- When mapping the footprint to the screen pixels he uses a linear transform rather than a perspective one.
- The footprint polygon is not rotated for every mapped pixel such that the corresponding pixel ray traverses it at a perpendicular angle.

While the error for the last approximation is rather small (see [15, Sec. 5.7.1]), the former two are more significant. The first approximation computes footprint screen extents that

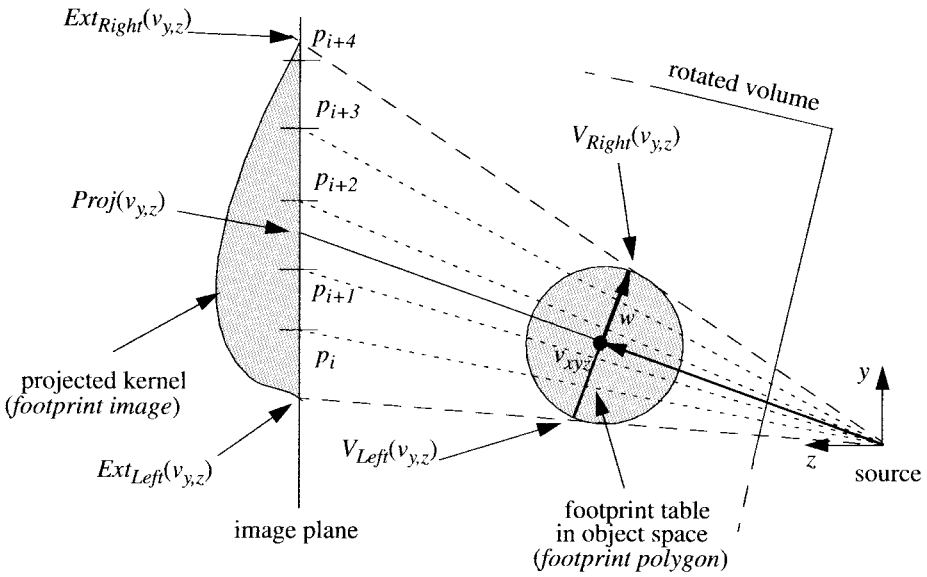


Fig. 1. Perspective voxel-driven splatting. First, the footprint polygon of voxel $v_{y,z}$ is mapped onto the image plane, then the affected image pixels $p_i \cdots p_{i+4}$ are mapped back onto the footprint table.

are smaller than the actual ones. For example, for a cone half angle of 30° and a 128^3 volume, the maximum error ratio between correct and approximate footprint extent is 1.15 and the maximum absolute difference between the two is 0.8 pixels (see [15, Sec. 5.7.2]). Here, the absolute error is again largest for those voxels that are located close to the view cone boundary. It causes the footprints of these voxels to cover less area on the projection plane than they really should. The second approximation has a similar effect. In that case, the mapping of the footprint table entries to the screen is slightly squashed. Again, voxels close to the view-cone boundary are most affected.

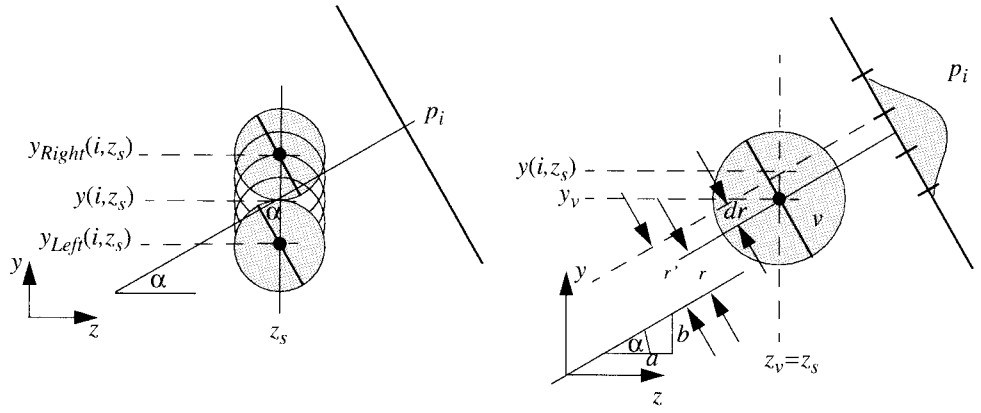
Consider now Fig. 1, where we illustrate a new and accurate solution for perspective voxel-driven splatting. For simplicity of drawing, we show the 2-D case only. Note that the coordinate system is fixed at the eye point. To splat a voxel $v_{x,y,z}$ it is first rotated about the volume center such that the volume is aligned with the projection plane. Then, the footprint polygon is placed orthogonal to the vector, starting at the eye and going through the center of $v_{x,y,z}$. Note that this yields an accurate line integral only for the center ray, all other rays traverse the voxel kernel function at a slightly different orientation than that given by the placement of the 2-D (1-D in Fig. 1) footprint polygon in object space. Thus, the first error in Westover's approximation still survives. This error, however, can be shown to be less than 0.01 pixels, even for voxels close to the source.

The coefficients of the footprint polygon's plane equation are given by the normalized center ray (the *vector source-*$v_{x,y,z}$). From this equation we compute two orthogonal vectors u and w on the plane (only w is shown in Fig. 1). Here, u and w are chosen such that they project onto the two major axes of the image. Using u and w, we can compute the spatial x, y, z positions of the four footprint polygon vertices in object space ($V_{\mathrm{Right}}(v_{y,z})$ and $V_{\mathrm{Left}}(v_{y,z})$ in the 2-D case depicted in Fig. 1). These four vertices are perspectively projected onto the image plane. This yields the rectangular extent of the footprint image, aligned with the image axes ($\mathrm{Ext}_{\mathrm{Right}}(v_{y,z})$ and $\mathrm{Ext}_{\mathrm{Left}}(v_{y,z})$ in the 2-D case). By expressing the intersections of the pixel rays with the footprint polygon in a parametric fashion, we can then set up an incremental scheme to relate the image pixels within the footprint image with the texture map entries of the footprint table.

The computational effort to map a footprint polygon onto the screen and to set up the incremental mapping of the pixels into the footprint table is quite large: Almost 100 multiplications, additions, and divisions and two square root operations are necessary. No incremental scheme can be used to accelerate the mapping of neighboring grid voxels. The high cost is amplified by the fact that the expensive mapping has to be done at $O(N) = O(n^3)$. Indeed, in our implementation, perspective projection was more than twice as expensive as parallel projection.

IV. A Fast and Accurate Ray-Driven Splatting Algorithm for Cone-Beam Art

We saw in the previous section that perspective voxel-driven splatting can be made accurate, however, the expense of perspective voxel-driven splatting seems prohibitive for use in cone-beam reconstruction. In this section, we take advantage of the fact that, in contrast to voxel-driven approaches, ray-driven methods are generally not sensitive to the nonlinearity of the perspective viewing transform. It can thus be expected that ray-driven splatting is more advantageous to use in the perspective cone-beam situation. The new ray-driven approach is, in some respects, a 3-D extension to the 2-D algorithm sketched by Hanson and Wecksung [6] and will work both for constant-size kernels, as used in cone-beam SART, and variable-size kernels, as required for cone-beam ART.



(a) (b)

Fig. 2. Ray-driven splatting. (a) Determining the range of voxels within a given volume slice plane that are traversed by a ray originating at pixel p_i. (b) Computing the index dr into the footprint table.

A. Ray-Driven Splatting with Constant-Size Interpolation Kernels

In ray-driven splatting, voxel contributions no longer accumulate on the image plane for all pixels simultaneously. In contrast, each pixel accumulates its raysums separately, which makes also makes it more suitable for ART than voxel-driven splatting. Our algorithm proceeds as follows. The volume is divided into 2-D slices formed by the planes most parallel to the image plane. When a pixel ray is shot into the 3-D field of interpolation kernels, it stops at each slice and determines the range of voxel kernels within the slice that are traversed by the ray. This is shown in Fig. 2(a) for the 2-D case. The ray originating at pixel p_i pierces the volume slice located at x_s at $y = y(i, z_s)$. The voxel kernels within the slice z_s that are intersected by the ray are given by the interval [Ceil($y_{\text{Left}}(i, z_s)$), Floor($y_{\text{Right}}(i, z_s)$)]. We compute $y_{\text{Right}}(i, z_s)$ as

$$y_{\text{Right}}(i, z_s) = y(i, z_s) + \frac{\text{extent}_{\text{kernel}}}{\cos(\alpha)} \tag{2}$$

where α is the inclination of the ray. The computation for $y_{\text{Left}}(i, z_s)$ is analogous. After determining the active voxel interval $[y_{\text{Left}}(i, z_s), y_{\text{Right}}(i, z_s)]$, we must compute the indexes into the voxel footprint table. This can be efficiently implemented by realizing that the index into the footprint table of a grid voxel v located at coordinates (y_v, z_v) is given by the distance d_r of the two parallel lines (planes in 3-D) that traverse v's centerpoint and the slice intersection point of the ray at $y(i, z_s)$, respectively [see Fig. 2(b)]. One finds

$$\begin{aligned} dr &= a \cdot z_s + b \cdot y(i, z_s) - a \cdot z_s - b \cdot y_v \\ &= (b \cdot (y(i, z_s) - y_v)) \end{aligned} \tag{3}$$

where a and b are the coefficients of the implicit line equation $a \cdot x_s + b \cdot y(i, x_s) = 0$ of the ray and are also given by the components of the (normalized) ray vector. Maintaining the variables $y_{\text{Left}}(i, z), y_{\text{Right}}(i, z)$ and dr along a ray can all be done using incremental additions.

For the 3-D case, we need to replace the linear ray by two planes. A 3-D ray is defined by the intersection of two orthogonal planes cutting through the voxel field. The normal for one plane is computed as the crossproduct of the ray and one of the image plane axis vectors. The normal of the second plane is computed as the crossproduct of the ray and the normal of the first plane. Thus, the two planes are orthogonal to each other and are also orthogonal to the voxel footprint polygons. Thus, the ray pierces the footprint polygon in a perpendicular fashion, as required. Intersecting the horizontal plane with a footprint polygon and using plane equations in the spirit of (3) results in the horizontal row index dr_{row} into the footprint table, while the intersection with the vertical plane yields the vertical column index dr_{col}. Using these two indexes, the value of the ray integral can be retrieved from the 2-D footprint table. Note that the two orthogonal directions of the indexes dr_{col} and dr_{row} on the footprint polygon plane allow us to implement the bidirectional footprint stretching required for the variable-size interpolation kernels in cone-beam ART.

There are now three nested loops. The most outer loop sets up a new ray to pierce the volume, the next inner loop advances the ray across the volume slice by slice and determines the set of voxels traversed per slice, and, finally, the most inner loop retrieves the voxel contributions from the footprint tables. For perspective projection, the plane equations have to be computed for every ray. This amounts to approximately 50 extra additions, multiplications, and divisions, and three square roots per pixel. The cost of advancing a ray across the volume and determining the footprint entries is comparable to the cost of rotating a kernel and splatting it onto the image plane in the orthographic voxel-driven approach. The ray-driven approach changes the splatting algorithm from voxel order to pixel order. Thus, the most outer loop is of $O(n^2)$. This has the advantage that the complexity of any extra work that has to be done for perspective projection (e.g., recomputing the two planes that define the ray in 3-D) is roughly one order of magnitude less than in voxel-driven splatting. Note also that ray-driven splatting does not introduce inaccuracies. As a matter of fact,

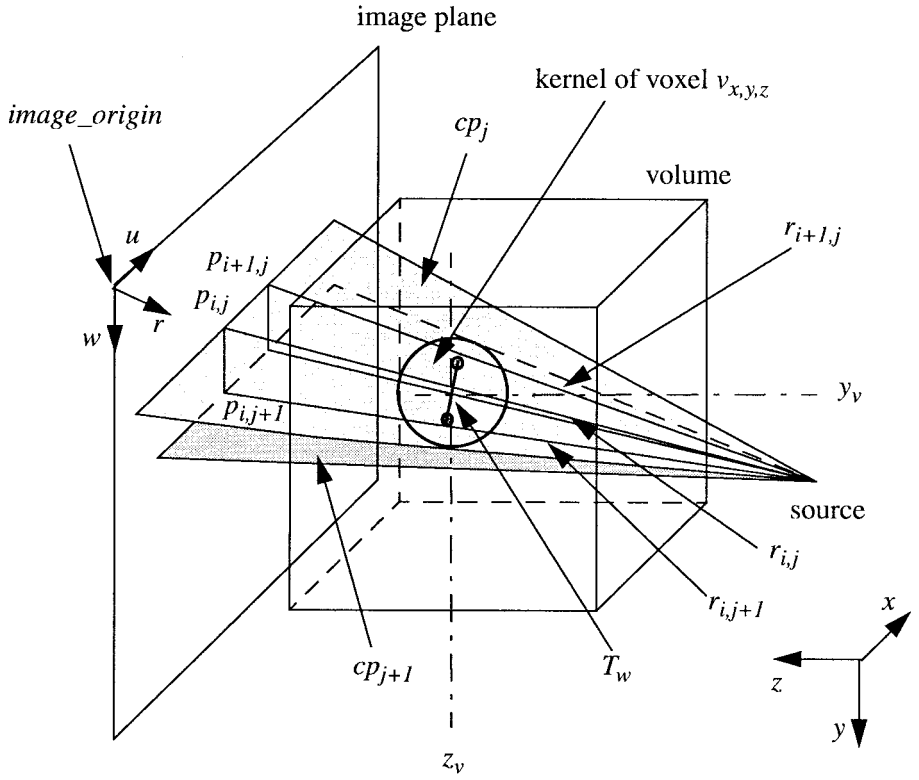


Fig. 3. Determining the local sampling rate of the arrangement of diverging rays. The arrangement of rays traverses the volume grid in two orthogonal sets of ray sheets (two horizontal sheets, i.e., the cutting planes cp_j and cp_{j+1}, are shown). Each 3-D ray is part of one sheet in each set and is defined by the intersection of these two sheets. Depending on the location of the kernel with respect to each sheet set, the 2-D kernel footprint must undergo different distortions in its two principal coordinate axes.

it prevents the indexing errors in the voxel-driven approach by design.

B. Ray-Driven Splatting with Variable-Size Interpolation Kernels

We have proposed in [13] and mentioned in Section II that the aliasing artifacts caused by the diverging set of rays in cone beam can be eliminated by progressively increasing the interpolation-filter extent (and scaling the filter amplitude) as a linear function of ray depth. To make these concepts more clear, let us define an image coordinate system (u, w, r) with u and w being the orthogonal image vectors and r being the vector normal to the image plane (see Fig. 3). Typically, r connects the source with the center of the image plane. The amount of stretching and scaling of the voxel kernels depends on their location with respect to the image coordinate system. To determine the proper amount of kernel distortion we need to express the sampling rate ω_r of the arrangement of rays in (x, y, z) coordinates. Once the function ω_r is known, we can then determine the required interpolation filter width or magnitude at each location along a ray. It was shown in [13] that ω_r is constant along curved rayfront isocontours and decreases linearly with the increasing distance of the isocontours from the source. This linear dependency on ray depth means that each voxel kernel must undergo a nonuniform distortion along a ray. However, since we use identical, preintegrated kernels in the form of 2-D footprint polygons, we cannot realize this nonuniform distortion function. Hence, as an approximation, we only estimate ω_r at the location of each kernel center and distort the generic 2-D footprint. This approximation is justified in the Appendix.

Consider again Fig. 3. The coordinates of an image pixel can be expressed as $p_{ij} = \text{image\_}origin + iu + jw$. The grid of diverging rays is organized into horizontal and vertical sheets, or cutting planes, that intersect the image plane and are spaced by u and w. The ray grid sampling rate ω_r is then a 2-D vector $(\omega_{ru}, \omega_{rw})$ that is related to the local sheet spacings. Fig. 3 illustrates how ω_{rw} is calculated. Here, we see the two horizontal cutting planes cp_j and cp_{j+1} embedding the rays $r_{i,j}$ and $r_{i,j+1}$, respectively. To approximate $T_w = 1/\omega_{rw}$ at the location $(x_v,\ y_v,\ z_v)$ of the kernel center of voxel $v_{x,y,z}$, we measure the distance between cp_j and cp_{j+1} along the vector orthogonal to cp_j passing through $(x_v,\ y_v,\ z_v)$. This distance can be written as $T_w = ax_v + by_v + cz_v + k$ where $(a\ b\ c)$ is a linear function of the plane equations of cp_j and cp_{j+1} and can thus be updated incrementally for all intersected voxels along a ray. If we select the horizontal and vertical cutting planes such that the image plane vectors u and w, respectively, are embedded in them, then we can simply stretch the footprint polygon by a scale factor of amplitude T_w to achieve the proper lowpassing in the w direction of the ray grid. (Recall that, in foward projection, we also have to scale the kernel's amplitude by a factor $1/T_w$.) An analogous argument can be given for the vertical cutting planes and

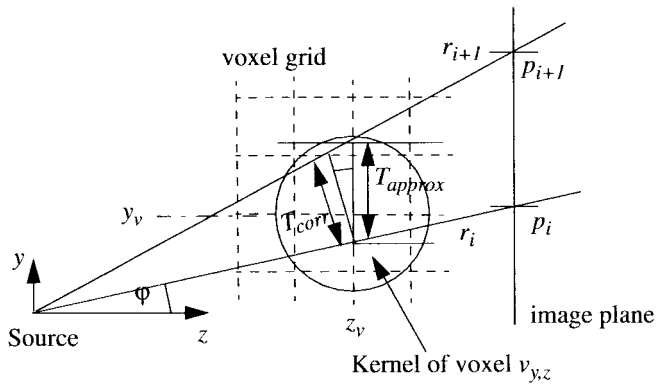


Fig. 4. Error for different schemes of measuring the ray grid sampling rate.

$T_u = 1/\omega_{ru}$. Also recall from [13] that we only stretch the footprint polygon if $T_u > 1$ or $T_w > 1$. If, in forward projection, $T_u \leq 1$ or $T_w \leq 1$, then the ray grid sampling rate in that direction is sufficient such that no aliasing can occur. However, in backprojection we do need to scale the amplitude of the kernel whenever $T_u < 1$ or $T_w < 1$. Here, the scale factor is T_u or T_w, respectively. Note that in order to preserve the orthogonality of the two cutting planes, in the general case one cannot achieve that u lies in the horizontal cutting planes and, at the same time, w lies in the vertical planes. However, since we flip the main viewing direction as soon as the angle between the ray direction and the major viewing axis exceeds 45°, the angular deviation of the true orientation of the cutting plane and the correct orientation is small (less than 5°).

The incremental scheme to compute the distance between two cutting planes requires about one addition per voxel in each slice. Since in a single-orbit reconstruction u lies in the x-z plane and w is aligned with the y-axis of the volume grid, a more efficient way is to measure the ω_r vector for a ray $r_{i,j}$ in the volume slice most parallel to the projection plane and use this ω_r vector for all footprint polygons in this plane. This is shown for the 2-D case in Fig. 4. Here, T_{corr} is the distance measured by the scheme described first, while T_{approx} is the slice-based measurement. The error is given by $T_{\text{corr}} \approx T_{\text{approx}} \cdot \cos\varphi_i$. This means that the simpler method underestimates the grid sampling rate by some amount. In the case of ω_{ru}, the maximum error occurs for voxels on the view-cone boundary. Here, for a cone half-angle $\varphi_c = 30°$, the simpler method would choose a kernel that is about $1/0.86 = 1.15$-times larger than it needs to be and thus lead to a greater amount of lowpassing of voxel $v_{x,y,z}$ in the u direction than is required. However, the fact that the factor $\cos(\varphi)$ is rather small and approaches values close to 1.0 quickly as we move away from the view-cone boundary, makes this approximation a justifiable one. In the case of ω_{ru}, which determines the kernel stretching factor in the x-z plane, the error can get a bit larger. Here, the rays can meet the volume slice plane most parallel to the viewing plane at an angle of up to 45°. Greater angles are avoided since we flip the major viewing direction as soon as an angle of 45° is exceeded, as was mentioned above. Thus, the error $T_{\text{approx}}/T_{\text{corr}}$ when determining the kernel stretch factor for the u direction can grow up to $1/\cos(45°) = 1.41$.

V. Caching Schemes for Faster Execution of ART and SART

In the previous section, we discussed a ray-driven projection algorithm that minimizes the number of necessary calculations per projection by utilizing fast incremental ray traversal schemes. In this section we will investigate to what extent previously computed results can be reused, i.e., stored or cached, so the number of redundant calculations can be minimized.

Caching can be performed at various scales, with the largest scale being iteration based, in which all weights are precomputed and stored on disk. The number of weights to be stored can be estimated as follows. If we only consider voxels in the spherical reconstruction region, then the total number of relevant voxels $N \approx 0.5n^3$. With a square footprint extent of 4.0, the average number of rays traversing a footprint polygon is 16. Thus, the number of relevant weights per projection is $8n^3$. For a number of projections $S = 80$ and a voxel grid of 128^3 voxels, the total number of relevant weight factors is then about 1.3 trillion floating point values and 5.3 GB of actual data. This is clearly too much to hold in RAM. On the other hand, if we just held the coefficients for one projection at a time, we would need 67 MB of RAM. This is in addition to the volume data and other data structures, but is feasible with today's workstations. However, then we would have to load a 67-MB file for every new projection that we work on. It is likely that the disk transfer time exceeds the savings in that case. In addition, the memory demands grow dramatically for larger volumes, since the number of weights to store is eight times the number of voxels.

Since caching on both the iteration and the image level is not practical, one may exploit caching on the ray level. ART is an easy candidate for this form of caching since a pixel projection is immediately followed by a pixel backprojection. So one can just cache the weight factors calculated in the projection step for the backprojection step and speedups of close to 2.0 can be realistically expected, with only little memory overhead.

For SART, two special problems need to be addressed. One has to do with the use of a ray-driven projection algorithm, while the other deals with caching. While ART was easy to pair with a ray-driven projection algorithm since it is itself ray-driven, the backprojection step of SART is inherently voxel based and requires some adaption in order to limit memory requirements. In a brute-force implementation, a backprojection would require two additional volumes, one to store the weight accumulation and one to store the correction accumulation per voxel (see [13, eq. (4)]). Only after all backprojection rays have been traced can the correction buffer of each voxel be divided by the weight buffer to form the voxel correction value. Thus, we need extra memory to hold $2n^3$ floating point values. We can reduce this amount by an order of magnitude to $2n^2$ by tracing all rays simultaneously in form of a ray front. Since the projection algorithm is slice-based, i.e., it considers all voxels in one volume slice before it moves to the next, we can step the entire ray front from one slice to the next, buffering and updating only the voxels within the active volume slice.

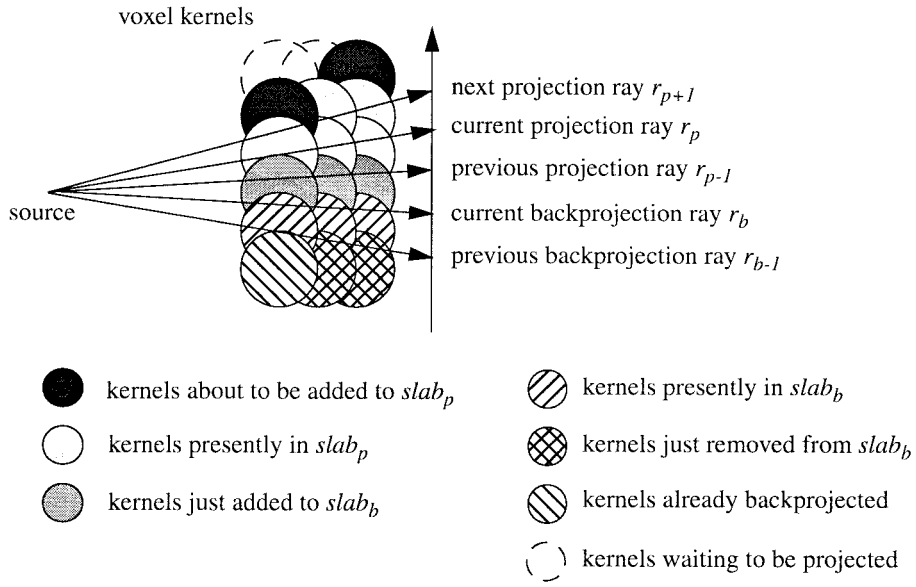


Fig. 5. Caching in SART (illustrated for the 2-D case). A partial kernel set of the volume is shown. The ray r_p is the current projection ray. Kernels on its path that are not currently in slab_p are added to slab_p. Once kernels have been entered into slab_p, their computed weights are cached. After the ray has been cast, all kernels in slab_p that have just been fully projected are transferred to slab_b, along with their cached weight factors. Ray r_b is the current backprojection ray. All kernels within its path are present in slab_b and are backprojected along ray r_b. The voxel weight and correction accumulation buffers are updated in this process. All voxels that have been fully backprojected are then updated by their accumulated correction value and removed from slab_b. In 3-D, the rays become plane sheets (compare Fig. 3).

In SART, the caching of weights computed during projection is also more difficult, since first an entire image must be projected before the grid corrections can be backprojected. Thus, at first glance we may only be able to use caching at an image level. This would require us to allocate memory space for $8n^3$ floating point weights, e.g., $32n^3$ bytes, which is in addition to other memory requirements. While for $n = 128$ this may be feasible for an average workstation (the required memory is then 67 MB), for $n = 256$ the memory requirements would be a hefty 536 MB, which may not be readily available in most workstations. Thus, in real world applications, caching on the image level is not feasible, at least with today's workstations, and one must design a caching scheme at a finer granularity.

For this purpose, we designed a scheme that keeps two active slabs, composed of sheets of voxel cross sections. These voxel cross sections are formed by intersecting the voxel kernels by consecutive horizontal cutting planes (recall Fig. 3). In this scheme, illustrated in Fig. 5, one active slab, slab_p, is composed of voxels that are currently projected, while the other, slab_b, is composed of currently backprojected voxels. Here, a slab voxel is a data structure with a weight array and an accumulation buffer for weight and correction sums, to be used during backprojection. Slab slab_b is always trailing slab_p. At first, slab_p caches the weights computed in the projection step. Then, as slab_p moves upward in the volume, voxels on the bottom of slab_p have eventually been completely projected and can be removed from slab_p and added to slab_b (along with all cached weights). A linked list can be used to facilitate the passing of the data. Once all voxels that a ray can traverse have been transferred to slab_b, the ray updates the accumulation buffers of the affected slab_b voxels, using the cached weight factors and its correction value. As slab_b moves upward as well, voxels at the bottom of slab_b can eventually be updated by the accumulated correction buffer term and removed from slab_b.

Let us now compute the memory requirements. The width of a slab is about four sheets, and a voxel kernel with an extent of 4.0 is traversed by about 16 rays. Thus, the data structure of a slab voxel consists of an array of 16 weight factors in addition to two accumulation buffers. Thus, the memory complexity for the two slab buffers is roughly $4(18 + 18)n^2 = 144n^2$. This includes the memory for the correction and accumulation buffers of slab_b. Thus we would require approximately 10 M of memory for a 128^3 volume. Note that this caching scheme goes well with the variable-size voxel kernels, since here the slab width is constant for all voxels with $T_u > 1$ and $T_w > 1$.

VI. Results

Table I lists the run times of the various ART and SART incarnations that were discussed in the previous sections. The run times were obtained on an SGI Indigo^2 workstation and refer to a reconstruction based on 80 projections with a cone angle of 40°.

Let us first look at the SART correction algorithm. We observe in Table I that for parallel-beam reconstruction with SART the voxel-driven approach is about 33% faster than the ray-driven approach. Hence, it is more advantageous in the parallel-beam case to perform the grid projection in object order (i.e., to map the footprint polygons onto the screen) than to perform the projection in image order (i.e., traverse the array of footprint polygons by the pixel rays). The computational savings in the voxel-driven algorithm for parallel-beam projection come from the fact that here the

TABLE I
RUN-TIMES FOR SART, USING BOTH VOXEL-DRIVEN AND RAY-DRIVEN SPLATTING, AND FOR ART USING RAY-DRIVEN SPLATTING (VOXEL-DRIVEN SPLATTING IS NOT APPLICABLE FOR ART). THE EFFECT OF CACHING AND VARIABLE-SIZE INTERPOLATION KERNELS ON THE RUN TIME IS ALSO SHOWN. (T_{corr}: TIME FOR ONE GRID CORRECTION STEP, CONSISTING OF ONE PROJECTION AND ONE BACKPROJECTION, T_{iter}: TIME FOR ONE ITERATION (ASSUMING 80 PROJECTION IMAGES AND A CONE ANGLE OF 40°) $T_{3\mathrm{iter}}$: TIME FOR THREE ITERATIONS, THE MINIMUM TIME TO OBTAIN A RECONSTRUCTION OF GOOD QUALITY [13]. TIMINGS WERE OBTAINED ON A SGI IRIS INDIGO 2 WITH A 200-MHz RS4000 CPU)

Method	beam	splatting method	cache	var-size kernel	T_{corr} (sec)	T_{iter} (hrs)	$T_{3\mathrm{iter}}$ (hrs)
SART	parallel	voxel	-	-	35.3	0.78	2.35
SART	parallel	ray	-	-	47.1	1.04	3.14
SART	cone	voxel	-	-	144.9	3.22	9.66
SART	cone	ray	-	-	60.9	1.35	4.05
SART	cone	ray	-	✓	73.2	1.63	4.89
SART	cone	ray	✓	-	43.6	0.97	2.90
SART	cone	ray	✓	✓	52.4	1.16	3.49
ART	cone	ray	-	-	54.2	1.20	3.60
ART	cone	ray	-	✓	68.2	1.51	4.53
ART	cone	ray	✓	-	30.3	0.67	2.01
ART	cone	ray	✓	✓	38.1	0.85	2.55

footprint-screen mapping is much simpler than the mapping described in Section III, since the perspective distortion does not have to be incorporated. In cone-beam reconstruction, on the other hand, the situation is reversed in favor of the ray-driven projector. Here, the speedup for using the ray-driven projector over the voxel-driven projector in SART is about 2.4. Thus, since the use of the image-based voxel-driven splatting algorithm is not practical for ART, we conclude that cone-beam reconstruction should always be performed with ray-driven projectors.

Now let us investigate the computational differences of ART and SART and the effects of caching and variable-size splatting kernels on run time. Comparing the costs for SART and ART, we notice that uncached SART is about 12% slower than uncached ART. This is due to the extra computations required for weighting the corrections before a voxel update and due to the overhead for managing the additional data structures. The timings also indicate that the use of a depth-dependent kernel size incurs about a 25% time penalty for ART and 20% for SART. In terms of the benefits of caching, we notice that the straightforward caching for ART speeds reconstruction by a factor of 1.78. Stated in another way, cached ART reduces the time for a projection/backprojection operation to the time-equivalent of 1.12 projections. The more involved caching for SART, on the other hand, achieves a speedup of 1.4. Caching in conjunction with the variable-size kernels produces similar improvements. Since the reconstruction results for SART, using constant-sized kernels, and ART, using variable-size kernels, are similar [13], it makes sense to compare these two methods as well. In this respect, ART, with variable-size kernels and easy-to-implement caching, is about 1.5 as fast as uncached SART. However, if SART is enhanced with elaborate caching schemes, this speed advantage shrinks to a factor of 1.15.

VII. CONCLUSIONS

The prime motivation of the work presented here was to devise techniques that make algebraic methods more efficient for routine clinical use, while not compromising their accuracy. In particular, the fact that algebraic methods have been shown to be capable of producing comparable reconstructions and, in some settings, even better reconstructions than FBP makes this effort all worthwhile.

Since the projection algorithm represents the main source of ART computations, we first focused on this portion of the ART algorithm. We started by describing a new cone-beam extension to Westover's voxel-driven parallel-beam splatting algorithm [19]. This new extension removes almost all inaccuracies of previously outlined extensions of that sort. Then we analyzed existing ray-driven projectors in terms of their suitability for perspective cone-beam reconstruction. It was found that, generally, a ray-driven algorithm is far more suited for the perspective cone-beam projection case than a voxel-driven splatting algorithm. However, it was also found that for parallel-beam reconstruction with SART, the voxel-driven splatting algorithm is faster. In the quest for an efficient ray-driven algorithm for perspective 3-D projection we observed that most of the existing ray-driven algorithms were not applicable for the special needs of cone-beam reconstruction. We hence extended a conceptually existing 2-D ray-driven splatting algorithm into 3-D and optimized it for speed and accuracy. We also described how this algorithm is best used in conjunction with the depth-dependent interpolation kernels necessary for cone-beam ART.

However, a fast projection algorithm is not enough. We must also reduce the actual complexity of the overall projection–backprojection framework. This can be achieved by designing schemes that memorize or cache weight calculations, performed during projections, for their reuse in subsequent backprojections. Since it proves prohibitive with regard to today's memory cost to perform caching on an iteration-level or image-level, we devised an easy-to-implement ray-based caching scheme for ART and a more elaborate ray sheet-based caching scheme for SART. The latter is more involved since in SART a voxel must first be fully projected before a correction/backprojection can be performed on it. Using these concepts, experiments revealed that caching allows ART to reduce the cost for a projection/backprojection operation to the time equivalent of 1.12 projections, while SART has a more moderate speedup, as was to be expected, due to the more complicated caching mechanism.

The projection methods outlined in this paper for cubic grids also fully extend to the dodecahedral or body-centered grids that were proposed by Matej and Lewitt [11]. (These grids were shown to reduce the number of voxels to be processed by about 30%.) Since the dodecahedral grids are really just a stack of interleaved square grids, the incremental grid traversal algorithms have to be modified only slightly.

While this paper focused mostly on fast implementations of ART-type methods in the context of 3-D cone-beam reconstruction, it should be noted that different parts of the material presented are also relevant for parallel-beam and fan-beam reconstruction. For instance, the caching schemes that were described in Section V can be applied for all beam configurations, while the ray-driven algorithms are also preferable in the fan-beam setting. However, as was mentioned before, for parallel-beam projection the ray-driven algorithms are generally slower (by about 20%), due to the extra overhead for setting up the rays.

With memory costs decreasing at a fast rate, it may be possible to use image-based caching in the near future, even for large reconstruction volumes. However, at the present time it seems infeasible to do so, which hampers the efficiency of SART. In addition, with disk-to-memory bandwidths increasing, it may be possible to load precomputed weights faster from disk than they can be computed on the fly for every projection. However, as CPU speeds are also increasing at greater rates than the memory speeds, today's imbalances may very well still exist for many years to come.

APPENDIX
STRETCHING OF THE 3-D INTERPOLATION KERNEL VERSUS STRETCHING THE 2-D FOOTPRINT

We have shown in [13] that in cone-beam reconstruction the 3-D interpolation kernel of voxels beyond a certain distance z from the source must be stretched above its normal size. To be specific, the stretching must occur according to the function z/z_c, where z_c is the distance at which the sampling rate of the volume grid just equals the sampling rate of the ensemble of rays. This means that we must stretch the interpolation kernel function according to a ramp function of slope $1/z_c$ [see Fig. 6a].

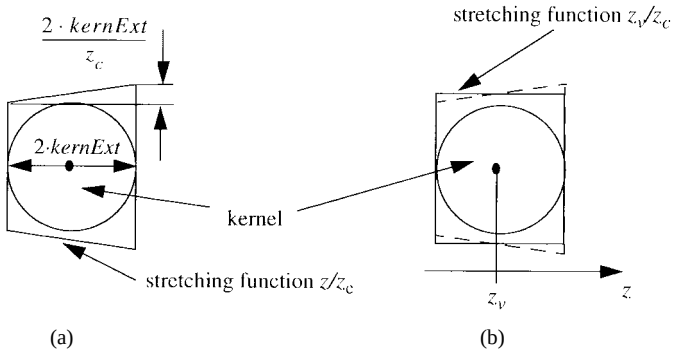


Fig. 6. (a) Stretching the interpolation kernel at $z = z_v$ according to the perspective stretching function z/z_c which has the form of a trapezoid. (b) Stretching the kernel by the function z_v/z_c, which is a box.

But how is this done in practice? The relative kernel distortion varies depending on the kernel's z coordinate. Simply preintegrating the distorted kernel for one kernel center location $z = z_1$ and then scaling this footprint for another kernel at $z = z_2, z_2 \neq z_1$, based on the ratio z_2/z_1, does not yield the correct footprint of a distorted kernel at that location. If we wanted to use preintegrated footprints that bear the correct kernel distortion, we would have to use a different footprint for each kernel center z location. This is obviously impractical. However, we realize that in most cases $z_c \gg 2 \cdot kernExt$. For instance, for a 128^3 volume, a 128^2 projection image, and a 60° cone angle, the factor $2 \cdot \mathrm{kernExt}/\mathrm{z_c}$ is about 0.03 volume units. Thus, we may stretch the kernels by a box instead of a trapezoid [see Fig. 6(b)] without committing much of an error (i.e., the error is 0.015 volume units). This minimal error enables us to use the generic footprint polygons, scaled up in the x- and y direction, depending on their location along z.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their careful reading of the manuscripts and their suggestions.

REFERENCES

[1] A. H. Andersen and A. C. Kak, "Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm," *Ultrason. Imag.* vol. 6, pp. 81–94, 1984.
[2] R. Crawfis and N. Max, "Texture splats for 3D scalar and vector field visualization," *Visualization'93*, 1993, pp. 261–266.
[3] L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone beam algorithm," *J. Opt. Soc. Amer.*, 1984, pp. 612–619.
[4] R. Gordon, R. Bender, and G. T. Herman, "Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography," *J. Theoretical Biol.*, vol. 29, pp. 471–481, 1970.
[5] H. Guan and R. Gordon, "Computed tomography using algebraic reconstruction techniques (ART's) with different projection access schemes: A comparison study under practical situations," *Phys. Med. Biol.*, no. 41, pp. 1727–1743, 1996.
[6] K. M. Hanson and G. W. Wecksung, "Local basis-function approach to computed tomography," *Appl. Opt.*, vol. 24, no. 23, 1985.
[7] G. T. Herman and L. B. Meyer, "Algebraic reconstruction can be made computationally efficient," *IEEE Trans. Med. Imag.*, vol. 12, pp. 600–609, June, 1993.
[8] P. M. Joseph, "An improved algorithm for reprojecting rays through pixel images," *IEEE Trans. Med. Imag.*, vol. 1, pp. 193–196, June, 1982.

[9] R. M. Lewitt, "Alternatives to voxels for image representation in iterative reconstruction algorithms," *Phys. Med. Biol.*, vol. 37, no. 3, pp. 705–715, 1992.

[10] S. Matej and R. M. Lewitt, "Practical considerations for 3-D image reconstruction using spherically symmetric volume elements," *IEEE Trans. Med. Img.*, vol. 15, pp. 68–78, Jan. 1996.

[11] _____, "Efficient 3D grids for image reconstruction using spherically-symmetric volume elements," *IEEE Trans. Nucl. Sci.*, vol. 42, pp 1361–1370, Apr. 1995.

[12] S. Matej, G. T. Herman, T. K. Narayan, S. S. Furuie, R. M. Lewitt, and P. E. Kinahan, "Evaluation of task-oriented performance of several fully 3D PET reconstruction algorithms," *Phys. Med. Biol*, vol. 39, pp. 355–367, 1994.

[13] K. Mueller, R. Yagel, and J. J. Wheller "Anti-aliased three-dimensional cone-beam reconstruction of low-contrast objects with algebraic methods," *IEEE Trans. Med. Imag.*, vol. 18, pp. 519–537, June 1999.

[14] K. Mueller, R. Yagel, and J. F. Cornhill, "The weighted distance scheme: A globally optimizing projection ordering method for the Algebraic Reconstruction Technique (ART)," *IEEE Trans. Med. Imag.*, vol. 16, pp. 223–230, Apr. 1997.

[15] K. Mueller, "Fast and accurate three-dimensional reconstruction from cone-beam projection data using algebraic methods," Ph.D. dissertation, Ohio State Univ., Columbus, OH, 1998.

[16] M. Schlindwein, "Iterative three-dimensional reconstruction from twin-cone beam projections," *IEEE Trans. Nucl. Sci.*, vol. 25, pp. 1135–1143, May, 1978.

[17] L. A. Shepp and B. F. Logan, "The Fourier reconstruction of a head section," *IEEE Trans. Nucl. Sci.*, vol. NS-21, pp. 21–43, 1974.

[18] R. L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional CT array," *Med. Phys.*, vol. 12, no. 2, pp. 252–255, 1985.

[19] L. Westover, "Footprint evaluation for volume rendering," *Comput. Grap. (SIGGRAPH)*, vol. 24, no. 4, pp. 367–376, 1990.