Fast Implementations of Algebraic Methods for 3D Reconstruction from Cone-Beam Data

Klaus Mueller[1,2], Roni Yagel[1], and John J. Wheller[2]

[1] Department of Computer and Information Science, The Ohio State University, Columbus, OH
[2] Department of Pediatrics, The Ohio State University, Columbus, OH

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. In particular, we strive to push the overall cost for an ART reconstruction as close as possible to the theoretical cost for a reconstruction obtained with Filtered Backprojection (FBP). While we focus mostly on fast implementations of ART-type methods in the context of 3D cone-beam reconstruction, different portions of the material presented here are also applicable to speed up reconstruction from fan-beam and parallel-beam data.

It was shown in previous research that three iterations are sufficient to obtain a high quality reconstruction for low-contrast cone-beam. Based on the observation that ART typically only requires only half the projections of FBP, we conclude that if the overall cost for ART's projection-backprojection operations could be cut in half, then one could obtain an ART implementation whose cost is close to the theoretical cost of FBP. To achieve this goal, we first survey existing projection algorithms and find 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, which is a well-known voxel-driven projection method. We also describe a new 3D ray-driven projector that is considerably faster than the voxel-driven projector, and at the same time more accurate and also perfectly suited for the demands of cone-beam. We then devise caching schemes for both ART and Simultaneous ART (SART). These schemes minimize the number of redundant computations for projection and backprojection and, at the same time, are very memory-conscious. We conclude that with caching and the proposed fast projection algorithm the computational cost of ART can be reduced to a level close to FBP's computational effort.

1. Introduction

The field of 2D and 3D reconstruction methods can be roughly divided into two main categories. On one side there is the domain of direct methods that capitalize on the Fourier Slice Theorem [4], while on the other side lies the domain of iterative methods that seek to solve the reconstruction problem by solving a system of simultaneous linear equations. The most prominent member of the former group is the Filtered Backprojection (FBP) algorithm. Here, the reconstruction is achieved by filtering the projection images with a ramp filter in frequency space, and then backprojecting the filtered projections onto a reconstruction grid [5]. The first and still most prevalent representative of the iterative methods is the Algebraic Reconstruction Technique (ART), attributed to Gordon et. al. [9], while another well-known ART-type method is Simultaneous ART (SART), proposed by Andersen and Kak [2]. In both methods, a reconstruction grid is iteratively updated by a projection-backprojection procedure until a convergence criterion is satisfied.

Two circumstances have led to the preference of FBP methods in today's clinical CT scanners: (i) FBP requires only one pass over the set of projection images, while ART-type methods require usually multiple passes, and (ii) for each projection image, FBP only requires a filtering operation followed by a backprojection, while ART must perform a forward projection *and* a backprojection. While the complexities of a projection and a backprojection are similar, the cost for filtering is usually somewhat less than the cost for a projection, depending on the extent of the filter used. Thus, at first glance, the preference for FBP methods is more than justified, especially if one considers that the reconstruction quality of FBP is just as good as with ART, as long as a sufficiently large number of projection images is provided (usually more than 512 images are required). If, however, one does not have a large set of projections available, or when the projections are sparse or missing at certain orientations, it was found that ART produces reconstructions of significantly better quality than FBP [1][15]. In contrast to FBP, ART also allows the application of spatial and numerical constraints, based on *a-priori* information, during the reconstruction process, which potentially leads to a better definition and contrast of the reconstructed object [25]. Finally, noisy data are also handled better with ART-type methods [15][19].

More recently, the importance of ART was reinforced by Matej et. al. [19] who found, for the reconstruction from noisy PET data, that ART produces quantitatively better results than the more popular FBP and MLE (Maximum Likelihood Estimation) methods. Finally, since the ART procedure is based more closely on the formation of the projection image, ART allows the modeling of certain X-ray artifacts such as photon scattering or beam hardening during the reconstruction procedure [15]. All these advantages make it seem well worth the while to conduct further research on ART.

From the previous discussion it becomes clear that if one wants to make ART faster and competitive with FBP for the general case, one must keep the number of iterations small and, at the same time, reduce the overall cost for the projection-backprojection operations. Particularly encouraging in this respect is the observation that ART-type methods theoretically only require about half the number of projections than the FBP methods [10]. Thus we can allows the number of iterations for ART to grow larger than one and still maintain the same complexity as FBP.

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 et. al. [21] 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 [26], 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 [12]. The impact of λ was also examined by us in [20], in conjunction with the effects of various other ART parameters, such as grid initialization and correction algorithm, on the cone-beam reconstruction of the 3D extension of the Shepp-Logan phantom. It was confirmed that within three iterations, a reconstruction of a quality close to the maximum can be obtained. In the same paper, we also described an extension to ART involving stretched interpolation kernels, which are necessary to eliminate the strong aliasing effects otherwise present in cone-beam reconstructions with ART.

In the present work, we will focus on the 3D reconstruction from cone-beam data, as this is still an active area of research. However, much of our discussion is also valid for the 2D and 3D reconstruction from parallel-beam and fan-beam data. Although a variety of general cone-beam algorithms based on FBP have been proposed [7][8][24], there are, as of today, no clinical 3D cone-beam scanners. The present literature for 3D cone-beam reconstruction with ART is mostly restricted to the reconstruction of high-contrast objects, such as encountered in computed angiography [22][25] (where we usually have a small number of projections) and reconstruction from PET [19] and SPECT [23] data (where the projections are usually noisy). Since high-contrast objects usually require a significantly smaller number of iterations than low-contrast object, it may not be fair to use the outcome of these studies for a comparison of FBP and ART in terms of their efficiency for the general case. Consequently, we will use our previous work [20][21] as an indicator that three iterations are the minimum number of iterations required for good reconstruction quality in the general reconstruction case.

Thus knowing that ART potentially requires only half the projection data of FBP, we have a good chance to make ART as efficient as FBP in the general case. To achieve this goal, we must improve the speed of ART's projection engine, as most of the computational expense of ART is spent for projection and backprojection. 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 [27] and ray-driven projectors [16][17] have been described, these algorithms are only efficient for the parallel-beam case or do not allow the stretched interpolation kernels prescribed in [20] as necessary for accurate cone-beam reconstruction. Furthermore, our voxel-driven perspective projection algorithm is considerably more accurate than the one described by Westover [27]. Our ray-driven algorithm, on the other hand, is a 3D extension of the 2D algorithm proposed by Hanson and Wecksung [11]. 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 done by re-using 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 general 3D reconstruction ART-type methods close to the theoretical cost of FBP methods.

Thus the outline of this paper is as follows. Section 2 gives a short recap on the workings of ART-type algorithms and describes previous work. Section 3 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 4 gives a new ray-driven projection algorithms for cone-beam ART that is both accurate and efficient. Section 5 details various caching schemes to speed ART and SART. Finally, Section 6 puts everything together and presents a variety of results obtained with our ART testbed software.

2. Preliminaries and Previous Work

In this section, we will give a brief review of those aspects of ART that are relevant to our work. While ART originally was proposed for the 2D case, the mathematical notation translates trivially to the 3D case.

As was mentioned before, ART poses the reconstruction problem as a system of linear equations:

$$
\begin{aligned}
w_{11}v_1 + w_{12}v_2 + w_{13}v_3 + \ldots + w_{1N}v_N &= p_1 \\
w_{21}v_1 + w_{22}v_2 + w_{23}v_3 + \ldots + w_{2N}v_N &= p_2 \\
&\ldots \\
w_{M1}v_1 + w_{M2}v_2 + v_{M3} + \ldots + w_{MN}v_N &= p_M
\end{aligned}
\tag{1}
$$

Here, the v_j are the values of the reconstruction grid elements (called *voxels* from now on), the p_i are the values of the pixels in the acquired projection images, and the weight factors w_{ij} represent the amount of influence a voxel j has on a ray passing from the source through image pixel i.

Usually one reconstructs on a cubic voxel grid with a sidelength of n voxel, thus the total number of voxels $N=n^3$ and the number of image pixels $M=S \cdot n^2$, where S is the number of projections. Also, for a 3D single-orbit reconstruction we generally assume a spherical reconstruction region. In this case we have $N = (1/6)\pi n^3$ unknown voxel values and $M = (1/4)\pi n^2$ relevant pixels per image. For the equation system (1) to be determined, the number of projections $S=N/M$ has then to be $0.67n$. This means that for n=128, 86 projections images are required. As was shown by Guan and Gordon [10] for the 2D case, the number of required projections in ART is half the number of the projections required for FBP. 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 is part of the reason for the strength of ART in the limited projection data case.

However, it is not always the case that S=0.67n. Sometimes (1) is overdetermined, or, more often, it is underdetermined. In either case, the large magnitude of (1) does not allow its solution by matrix inversion or least-squares methods. In addition, usually noise as well as sampling errors in the ART implementation do not provide for a consistent equation system anyhow. Thus an iterative scheme proposed by Kaczmarz [14] is used. Starting from an initial guess for the volume vector, $\boldsymbol{V}=\boldsymbol{V}^{(0)}$, we select at each iteration step k, $k>0$ one of the equations in (1), say the one for p_i. A value $p_i^{(k)}$ is measured which is the value of pixel i computed using the voxel values as provided by the present state of the vector $\boldsymbol{V}=\boldsymbol{V}^{(k)}$. A factor related to the difference of $p_i^{(k)}$ and p_i is then distributed back onto $\boldsymbol{V}^{(k)}$ which generates $\boldsymbol{V}^{(k+1)}$ such that if a $p_i^{(k+1)}$ were computed from $\boldsymbol{V}^{(k+1)}$, it would be closer to p_i than $p_i^{(k)}$. Thus, we can divide each grid update into three phases: a projection step, a correction factor computation, and a backprojection step.

The correction process for one element of $\boldsymbol{V}$, i.e. v_j, can be expressed by:

$$
v_j^{(k+1)} = v_j^{(k)} + \lambda \frac{p_i - \sum_{n=1}^{N} w_{in} v_n^k}{\sum_{n=1}^{N} w_{in}^2} w_{ij}
\tag{2}
$$

where λ is the *relaxation coefficient* typically chosen within the interval (0.0,1.0], but usually much less than 1.0 to dampen correction overshoot. This procedure is repeated in an iterative fashion for all equations in (1).

Instead of updating the volume on a ray-basis, SART [2] corrects the volume on an image-basis. This was shown to significantly reduce the noise artifacts that were observed with ray-iterative

ART. The projection step of SART performs a summed volume rendering [13] of the reconstruction grid, then subtracts the rendered image from the acquired projection image, normalizes the result, and backprojects the image in an inverse volume rendering process. More formally, the SART correction equation is as follows:

$$v_j^{(k+1)} = v_j^{(k)} + \lambda \frac{\sum_{p_i \in P_\varphi} \left(\frac{p_i - \sum_{n=1}^{N} w_{in} v_n^{(k)}}{\sum_{n=1}^{N} w_{in}} \right) w_{ij}}{\sum_{p_i \in P_\varphi} w_{ij}} \tag{3}$$

In this equation, the correction term for voxel j depends on a weighted average of all rays of a projection $P\varphi$ that traverse the voxel j. (Here, φ denotes the orientation angle at which the projection was taken.)

The sum terms in the nominators of (2) and (3) require us to compute the integration of a ray across the volume. The integration process can be performed by using raycasting, i.e., sampling the volume at equidistant locations with an interpolation kernel h and accumulating the interpolated values. Since accurate integration requires many sampling points, this is very time consuming. A better way that allows a more efficient and more accurate evaluation of the ray integral was proposed by a number of authors [11][16][17][27]. It consists of reordering the ray integral so that each voxel's contribution to the integral can be viewed isolated from the other voxels. In this alternative volume decomposition, the interpolation kernel is placed at the voxel locations, and the volume grid is viewed as a field of overlapping interpolation kernels which, as an ensemble, make up the continuous object representation. A voxel j's contribution is then given by $v_j \cdot \int h(s)\, ds$, where s follows the integration of the interpolation kernel in the direction of the ray. Here, $\int h(s)\, ds$ represents a voxel weight factor in (1), (2) and (2). If the viewing direction is constant for all voxels or if the interpolation kernel is radially symmetric, we can pre-integrate $\int h(s)\, ds$, often analytically, into a lookup-table (also called the *kernel footprint*). We can then map all voxel footprints to the screen, scaled by the voxel value, where they accumulate into a projection image, as is done in the splatting approach of Westover [27]. Alternatively, one can use rays to intersect the lookup tables in volume space, again scale the indexed value by the voxel value, and accumulate the density integrals ray by ray. The former approach we call voxel-driven splatting, which produces an image or an image region at a time and makes only sense in conjunction with an image-based correction algorithm, such as SART. The second approach is called ray-driven splatting and can be used with either ART or SART. Note, that the lookup-tables are also used to retrieve all other weight terms in (2) and (3). Moreover, backprojection is performed in a similar way than forward projection, only that here the voxels receive (corrective) energy, scaled by their weight factors, instead of emitting it.

The splatting approach has two advantages: (i) The ray integrals are calculated very accurately, since each footprint table entry can be integrated analytically or with good quadrature, and (ii) the complexity for interpolation is reduced from O(n^3) in volume space 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).

At this point, let us describe a peculiarity associated with perspective projection in cone-beam reconstruction. We showed in [20] that perspective projection and backprojection requires the interpolation kernels to be stretched and scaled according to the local grid sampling rate of the pixel rays. It was observed that the rays emanating from the source and traversing the volume grid form a divergent raster whose grid sampling density is a function of the distance from the X-ray source. This is shown in Fig. 1 for the 2D case. Here we see that the source rays traverse the volume in form of a curved rayfront with radius $s(z)$. The rate at which the ensemble of rays samples the grid within a rayfront is a linear function of $s(z)$. The larger $s(z)$, the smaller the ray ensemble's grid sampling rate. At $z=z_c$, the rayfront samples the grid at exactly the grid sampling rate. However, for $z<z_c$, the rayfront sampling rate is higher than the grid sampling rate, while for $z>z_c$, the rayfront sampling rate is lower than the grid sampling rate.

In order to avoid aliasing when sampling the volume grid in the ART projection phase, one must lowpass the volume grid for all $z>z_c$ according to the local ray grid sampling rate. This means that an interpolation kernel further away from the source must have a lower bandwidth, which in turn means that it must be wider in size. Changing the bandwidth of a kernel is simply achieved by stretching the kernel footprint. However, to prevent signal magnification effects, one must reduce the kernel's amplitude at the same rate as it was stretched. In 3D, the rays emanate in a 2D orthogonal sheet-like raster, which means that for a particular voxel the sampling rate in one raster dimension may be different than for the other. Thus we must be able to stretch the footprint by different amounts in the two ray raster directions. In the ART backprojection phase the situation is reversed. Now the volume grid samples the ray grid carrying the correction signal. It turns out that with the traditional constant-sized interpolation kernels the grid interpolated correction signal is magnified for $z<z_c$ and aliased for $z>z_c$. To ensure a uniform distribution of the correction signal onto the volume grid it was found (in [20]) that one must reduce the magnitude of the interpolation kernel for $z<z_c$ and, at the same time, one must increase the width of the interpolation kernel for $z>z_c$, this time without reducing its amplitude. In both cases the amount of scaling is linear in z.

Although ART must use the variable-size interpolation kernels for alias-free cone-beam reconstruction, it turns out that SART 's correction scheme also prevents the cone-beam related reconstruction artifacts [20], even without the use of variable-size kernels. However, we will see that SART is more difficult to speed up than ART.

While the concept of representing a volume by a field of interpolation kernels and pre-integrating a 3D kernel into a 2D 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 yields a circular footprint. For voxel-driven splatting, both Westover [27] and Matej and Lewitt [17] 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 3). In the case of ray-driven splatting we again assume a spherically symmetric interpolation kernel. Here the approaches are more diverse. For instance, Lewitt [16] 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

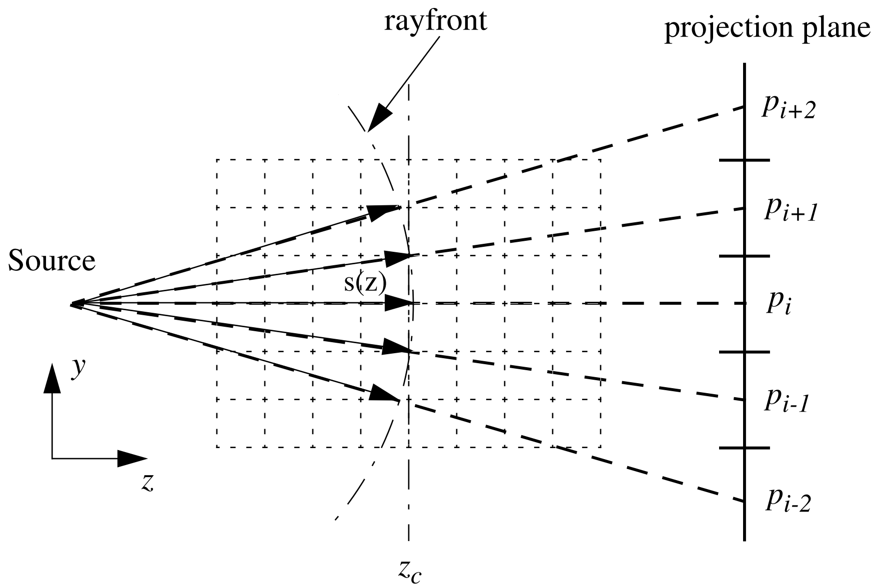

Fig. 1. Perspective projection for the 2D case.Rays emanate from the source into the volume along a rayfront of radius $s(z)$. The sampling rate of the ray grid is constant along the arc of the rayfront and decreases linearly in z.

to index a 1D footprint table storing the radially symmetric projection of the 3D kernel. 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. In another approach, Matej and Lewitt [18] decompose the voxel grid into a set of 2D slices. Here the orientation of the slices is that orientation most parallel to the image plane. Recall that a footprint is the pre-integrated 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, giving rise to an elliptical footprint. Since in parallel projection all rays for a given projection angle have the same angle with the volume slices, this re-mapped 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 indices into the elliptical footprints that are intersected. However, for perspective projection, every ray has a different orientation, necessitating a footprint re-mapping for every ray, which is inefficient both to compute on the fly and to store. A more appropriate approach was outlined for the 2D case by Hanson and Wecksung [11]. These authors model a 2D 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 1D footprint table. Our ray-driven approach is a 3D 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.

It should be mentioned that the choice of the interpolation kernel h varies in the various ART implementations. We will be using a kernel based on the Bessel-Kaiser window, as proposed by Matej and Lewitt [18]. Multidimensional Bessel-Kaiser functions have many desirable properties, such as fast decay for frequencies past the Nyquist rate and radial symmetry. It can also be tuned so that the kernel's frequency spectrum is at a minimum at multiples of the sampling frequency, where the signal's aliases are largest.

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

Let us first introduce some terminology. As suggested by Crawfis and Max [6], 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. 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 2D 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 pre-integrated integral match the true kernel integration of the ray. Westover's perspective extension to voxel-driven splatting violates this condition at three instances: (i) He does not align the footprint polygon perpendicularly to the voxel center ray when calculating the projected screen extent. Instead he aligns it parallel to the screen and stretches it according to the perspective viewing transform. (ii) When mapping the footprint to the screen pixels he uses a linear transform instead of a perspective one. (iii) 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, the error ratio of the computed footprint table index vs. the correct index in the first two approximations can be up to 1.15 for a 30° cone half angle.

Consider now Fig. 2, where we illustrate a new and accurate solution for perspective voxel-driven splatting. For simplicity of drawing, we show the 2D 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 given by the placement of the 2D (1D in Fig. 2) 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, 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 u is shown in Fig. 2). 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 ($V_{Right}(v_{x,y})$ and $V_{Left}(v_{x,y})$ in the 2D case depicted in Fig. 2). These four vertices are perspectively projected onto the image plane. This yields the rectangular extent of the footprint image, aligned with the image axes ($Ext_{Right}(v_{x,y})$ and $Ext_{Left}(v_{x,y})$ in the 2D 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.

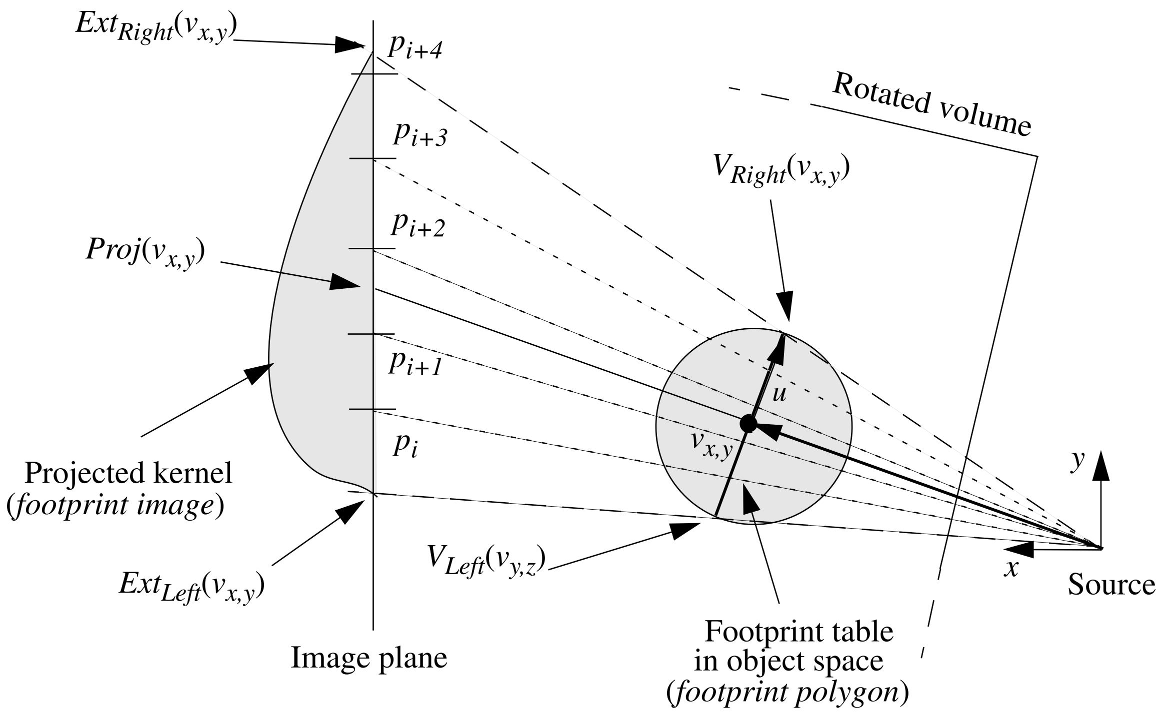

Fig. 2. Perspective voxel-driven splatting: First, the footprint polygon of voxel $v_{x,y}$ is mapped onto the image plane, then the affected image pixels p are mapped back onto the footprint table.

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 $O(N)=O(n^3)$. And indeed, in our implementation, perspective projection was about four times more expensive than parallel projection.

4. 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 non-linearity 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 respect a 3D extension to the 2D algorithm sketched by Hanson and Wecksung [11] and will work both for constant-size kernels as used in cone-beam SART and variable-size kernels as required for cone-beam ART.

4.1 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 it also more suitable for ART than voxel-driven splatting. Our algorithm proceeds as follows. The volume is divided into 2D slices formed by the planes most parallel to the image plane. When a sight-ray is shot into the 3D 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. 3a for

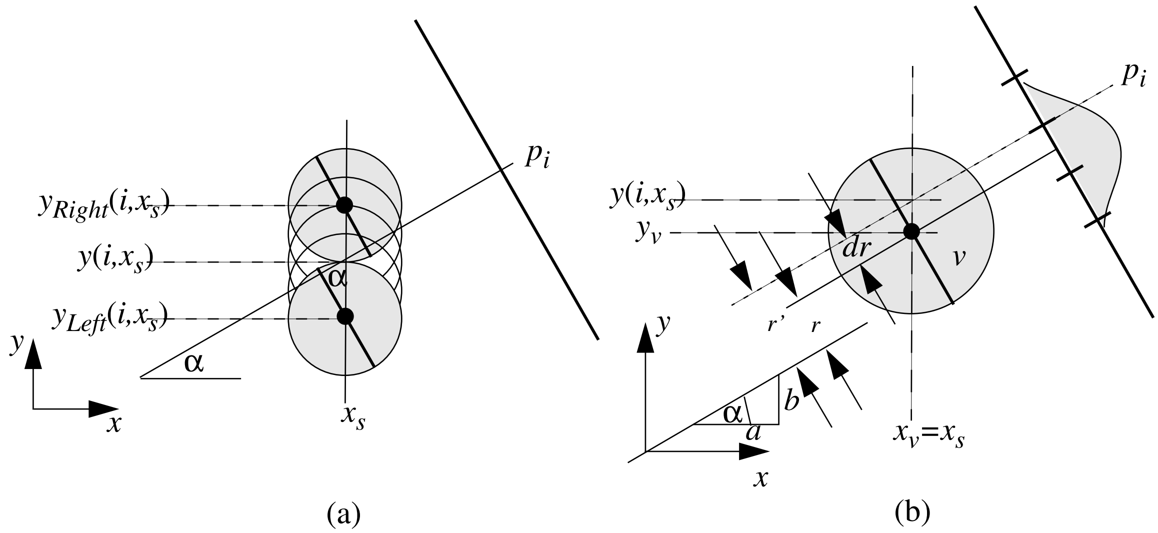

Fig. 3. Ray-driven splatting: (a) Determining the range of voxels within a given compositing plane that are traversed by a ray originating at pixel p_i. (b) Computing the index *dr* into the footprint table.

the 2D case: The ray originating at pixel p_i pierces the volume slice located at x_s at $y=y(i,x_s)$. The voxel kernels within the slice x_s that are intersected by the ray are given by the interval [Ceil($y_{Left}(i,x_s)$), Floor($y_{Right}(i,x_s)$)]. We compute $y_{Right}(i,x_s)$ as:

$$y_{Right}(i, x_s) = y(i, x_s) + \frac{extent_{kernel}}{\cos(\alpha)} \tag{4}$$

where α is the inclination of the ray. The computation for $y_{Left}(i,x_s)$ is analogous. After determining the active voxel interval [$y_{Left}(i,x_s)$, $y_{Right}(i,x_s)$] we must compute the indices 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,x_v) is given by the distance *dr* of the two parallel lines (planes in 3D) that traverse *v*'s centerpoint and the slice intersection point of the ray at $y(i,x_s)$, respectively (see Fig. 3b). One finds:

$$dr = a \cdot x_s + b \cdot y(i, x_s) - a \cdot x_s - b \cdot y_v = (b \cdot (y(i, x_s) - y_v)) \tag{5}$$

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 $y_{Left}(i,x)$, $y_{Right}(i,x)$, and *dr* along a ray can all be done using incremental additions.

For the 3D case, we need to replace the linear ray by two planes. A 3D ray is defined by the intersection of two orthogonal planes cutting through the voxel field. The normal for one plane is computed as the cross product of the ray and one of the image plane axis vectors. The normal of the second plane is computed as the cross product 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 (5) 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 indices, the value of the ray integral can be retrieved from the 2D footprint table. Note that the two orthogonal directions of the indices, dr_{col} and dr_{row}, on the footprint polygon plane allow us to implement the bi-directional 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 computed for every ray. This amounts to about 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 3D) 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, it prevents the indexing errors in the voxel-driven approach by design.

4.2 Ray-Driven Splatting with Variable-Size Interpolation Kernels

We have mentioned before that the aliasing artifacts caused by the diverging set of rays in cone-beam can be eliminated by progressively increasing the interpolation filter width (or magnitude, in backprojection for $z<z_c$) as a linear function of ray depth. This requires us to express the sampling rate ω_r of the arrangement of rays in grid coordinates (x,y,z). Once the function ω_r is known, we can determine the required interpolation filter width or magnitude at each location along a ray. We saw in Fig. 1 that ω_r is constant along spherical isocontours around the source and decreases linearly with increasing radius of these spherical isocontours. This linear dependence on ray depth means that each voxel kernel must undergo a non-uniform distortion along a ray. However, since we use identical, pre-integrated kernels in the form of 2D footprint polygons, we cannot realize this non-uniform distortion function. Hence, as an approximation, we only estimate ω_r at the location of each kernel center.

Consider now Fig. 4. The coordinates of an image pixel can be expressed as $p_{ij}=image\_origin+iu+jv$, where u, v are the orthogonal image plane axis vectors. The grid of diverging rays is organized into horizontal and vertical sheets that intersect the image plane spaced by u and v. The ray grid sampling rate ω_r is then a 2D vector $(\omega_{ru}, \omega_{rv})$. Fig. 4 illustrates how ω_{rv} is calculated. Here, we see the two horizontal cutting planes cp_j and cp_{j+1} defined by rays $r_{i,j}$ and $r_{i,j+1}$, respectively. To approximate $T_v=1/\omega_{rv}$ 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 $T_v=ax+by+cz+k$ where $(a\ b\ c)$ is a linear function of (x,y,z) and 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 v, respectively, are embedded in them, then we can simply stretch the footprint polygon by a scale factor of magnitude T_v to achieve the proper lowpassing in that ray grid direction. (Recall that, in foward projection, we also have to scale the kernel's amplitude by a factor $1/T_v$.) An analogous argument can be given for the vertical cutting planes and $T_u=1/\omega_{ru}$. Also recall that we only stretch the footprint polygon if T_u>1 or T_v>1. If, in forward projection, T_u≤1 or T_v≤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 magnitude of the kernel whenever T_u<1 or T_v<1. Here, the scale factor is $1/T_v$ or $1/T_u$, respectively. Note that in order to preserve 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, v lies in the vertical planes. But 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 degrees).

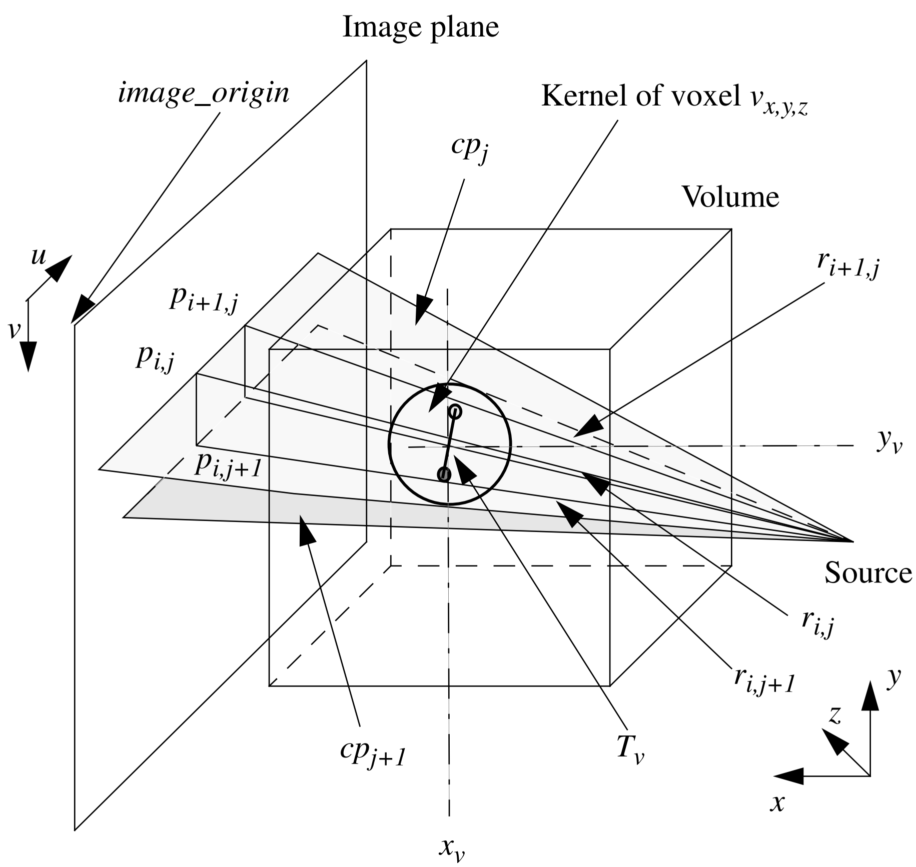

Fig. 4. Determining the local sampling rate of the arrangement of diverging rays.

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 v is aligned with the z-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 2D case in Fig. 6. 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 $T_{corr} \approx T_{Tapprox} \cdot \cos\varphi_c$. This means that the simpler method underestimates the grid sampling rate by a small amount. For example, at the view cone boundary at a cone half-angle φ_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 $v_{x,y,z}$ than required. The fact that the factor cos(φ) 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.

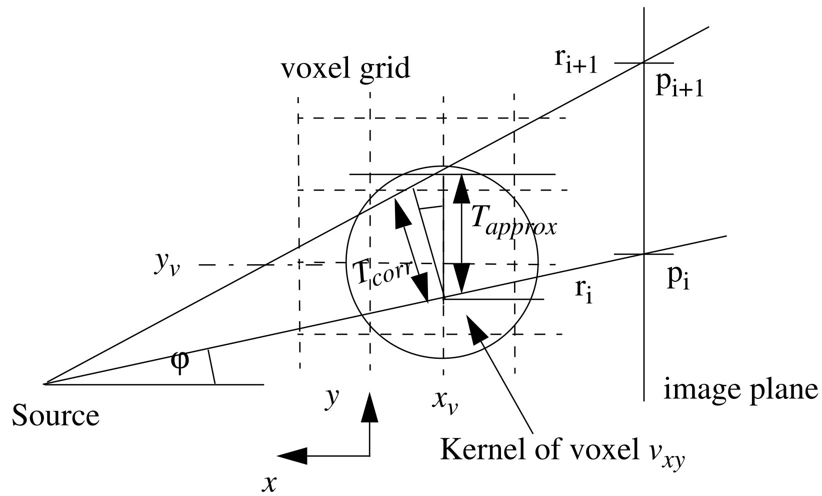

Fig. 5. Error for different schemes of measuring the ray grid sampling rate.

5. Caching Schemes for Faster Execution of ART and SART

In the previous section, we have 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 re-used, i.e. 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 pre-computed 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 1.3G floating point values, 5.3GB 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 67MB 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 67M 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 8 times the number of voxels.

Since caching on 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, the other deals with caching. While ART was easy to pair with a ray-driven projection algorithm since it itself is 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 equation (3)). Only after all backprojection rays have been traced, the correction buffer of each voxel can 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.

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 67MB), for n=256 the memory requirements would be a hefty 536MB, 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 intersected by consecutive horizontal cutting planes (recall Fig. 4). In this scheme, one active slab, $slab_p$, is composed of voxels that are currently projected, while the other, $slab_b$, is composed of currently backprojected voxels. The correction and weight buffers are kept with $slab_b$, and $slab_b$ 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 passing the data. 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 be removed from $slab_b$. The width of a slab is about four sheets. Recalling that a voxel is traversed by about four rays in each sheet, the memory complexity for the slab buffers is roughly $4\,(4+12)\,n^2 = 64n^2$. This includes the memory for the correction and accumulation buffers of $slab_b$. Thus we would require approximately 4M of memory for a 128^3 volume. Note that this scheme goes well with the variable-size voxel kernels since here the slab width is constant for $z>z_c$.

6. Results

Fig. 6a shows a slice from the 3D extension of the Shepp-Logan phantom (see [20] or [3] for details), reconstructed from 60° cone-beam data with ART using the traditional constant-size interpolation kernel. The aliasing artifacts caused by the diverging rays in cone-beam are easily noticeable. Fig. 6b shows the same slice reconstructed with ART using the variable-size interpolation kernel. Here we see that the previous aliasing artifacts are now eliminated. Finally, Fig. 6c shows the same slice reconstructed with SART using constant-size interpolation kernels. This figure shows that SART, as well, is able to prevent the cone-beam aliasing artifacts of traditional ART.

Table 2 lists the runtimes of the various ART and SART incarnations that were discussed in the

(a) (b) (c)

Fig. 6. A slice of the 3D extension of the Shepp-Logan phantom described in [20], reconstructed (a) with ART using constant-size interpolation kernels, notice the strong aliasing artifacts, (b) with ART using variable-size interpolation kernels, and (c) with SART. All reconstructions were performed for a cone angle γ=60°, the volume initialized to $\boldsymbol{V^{(0)}}$**=0**, S=80 projections, and 3 iterations. The relaxation coefficient for the ART reconstructions was λ=0.08, for SART λ=0.3.

previous sections. The runtimes were obtained on an SGI Indigo^2 workstation and refer to a reconstruction based on 80 projections with a cone angle of 40°. In this table we see that the speedup for using the ray-driven projector over the voxel-driven projector in SART is about 2.4, which justifies the sole use of ray-driven projectors for cone-beam reconstruction.

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. The timings also indicate that the use of a depth dependent kernel size incurs about a 25% time penalty for ART, 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, the more involved caching for SART achieves a speedup of 1.4.The speedups for caching in conjunction with the variable-size kernels are similar. Since the reconstruction results for SART using constant sized kernels and ART using variable-size kernels looked similar, 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 twice as fast as uncached SART. However, if SART is enhanced with elaborate caching schemes, this speed advantage shrinks to a factor of 1.15.

7. Conclusions

The prime motivation of this work was to devise techniques that make ART more efficient for routine clinical use, while not compromising its accuracy. In particular, the fact that ART has been shown to be capable of producing comparable reconstructions and, in some settings, even better reconstructions than FBP makes this effort all worthwhile. In this work, we also strived to push the overall cost for a ART reconstruction as close as possible to the cost for a FBP reconstruction.

In a back-of-the-envelope calculation we found that if computations done for forward projection could be saved for backprojection, then a maximum of three iterations would be about as time-consuming than an FBP reconstruction. This calculation took into account that ART only requires about half the projections of FBP. The reconstruction results presented in [20] indicated that three iterations are indeed as much as is required for good reconstruction quality. Since the projection

Method	cache	var. size kernel	T_{corr} (sec)	T_{iter} (hrs)	T_{3iter} (hrs)
ART	-	-	54.2	1.20	3.60
ART	-	✓	68.2	1.51	4.53
ART	✓	-	30.3	0.67	2.01
ART	✓	✓	38.1	0.85	2.55
SART (voxel-driven)	-	-	144.9	3.22	9.66
SART	-	-	60.9	1.35	4.05
SART	-	✓	73.2	1.63	4.89
SART	✓	-	43.6	0.97	2.90
SART	✓	✓	52.4	1.16	3.49

Table 2. Run-times for various ART-type algorithms with and without caching and variable-size interpolation kernels. (T_{corr}: time for one grid correction step, consisting of one projection and one back-projection, T_{iter}: time for 1 iteration (assuming 80 projection images and a cone angle of 40°), T_{3iter}:time for 3 iterations, the minimum time to obtain a reconstruction of good quality. Timings were obtained on a SGI Iris Indigo[2] with a 200MHz RS4000 CPU.) All methods use the ray-driven projection engine with linear rays, unless indicated otherwise.

algorithm represents the main source of computations, we mainly focused on this portion of the ART algorithm. First, we described a more accurate extension to Westover's voxel-driven parallel-beam splatting algorithm [27] as was previously known. 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. It was also found that most of the existing ray-driven algorithm were not applicable for the special needs of cone-beam reconstruction. We then extended an conceptually existing 2D ray-driven splatting algorithm into 3D 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.

We noted that the existence of a fast projection algorithm is a necessary but not sufficient condition for ART to be computationally competitive with FBP in the general case. We concluded that one must provide schemes that cache weight calculations performed during projections for their re-use in subsequent backprojections. Since it proofs prohibitive in 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. Caching allows ART to reduce the time spent in backprojection to 20%, while SART has a more moderate speedup, due to the more complicated caching mechanism. Thus, at least for ART, we are close to our goal of making ART-type algorithms as efficient as FBP.

We should also mention that the projection methods outlined in this paper for cubic grids also fully extend to the dodecahedral or body-centered grids that were proposed by [18]. 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 implementation of ART-type methods in the context of 3D 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 described caching schemes can be applied for all beam configurations, while the ray-driven algorithms are also preferable in the fan-beam setting. For parallel-beam, however, the ray-driven algorithm is 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 mainly the efficiency of SART. Also, with disk-to-memory bandwidths increasing, it may be possible to load pre-computed weights faster from disk than they can be computed on-the-fly for every projection. However, as CPU speeds are also increasing, currently at greater rates than the memory speeds. today's imbalances may very well still exist in many years to come.

References

[1] A. H. Andersen, "Algebraic Reconstruction in CT from Limited Views," *IEEE Trans. Med. Img.*, vol. 8, no.1, pp. 50-55, 1989.

[2] A.H. Andersen and A.C. Kak, "Simultaneous Algebraic Reconstruction Technique (SART): a superior implementation of the ART algorithm," *Ultrason. Img.*, vol. 6, pp. 81-94, 1984.

[3] C. Axelson, "Direct Fourier methods in 3D reconstruction from cone-beam data," *Thesis*, Linkoping University, 1994.

[4] R.N. Bracewell, *The Fourier Transform and Its Applications*, 2nd ed., McGraw-Hill, Inc., 1986.

[5] Z.H. Cho, J.P. Jones, and M. Singh, *Foundations of Medical Imaging,* Wiley, 1993.

[6] R. Crawfis and N. Max, "Texture splats for 3D scalar and vector field visualization," *Visualization'93,* pp. 261-266, 1993.

[7] L.A. Feldkamp, L.C. Davis, and J.W. Kress, "Practical cone beam algorithm," *J. Opt. Soc. Am.*, pp. 612-619, 1984.

[8] P. Grangeat, "Mathematical framework of cone-beam 3D reconstraution via the first derivative of the Radon transform," *Proc. Mathematical Models in Tomography (Oberwohlfach, 1990), Springer Lecture Notes in Mathematics, 1497,* pp. 66-97.

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

[10] H. Guan and R. Gordon, "Computed tomography using algebraic reconstruction techniques (ARTs) with different projection access schemes: a comparison study under practical situations," *Phys. Med. Biol*., no. 41, pp. 1727-1743, 1996.

[11] K.M. Hanson and G.W. Wecksung, "Local basis-function approach to computed tomography," *Applied Optics*, Vol. 24, No. 23, 1985.

[12] G.T. Herman and L.B. Meyer, "Algebraic reconstruction can be made computationally efficient," *IEEE Trans. Med. Img.*, vol. 12, no. 3, pp. 600-609, 1993.

[13] A. Kaufman, Ed., *Volume Visualization*, IEEE Press 1991.

[14] S. Kaczmarz, "Angenäherte Auflösung von Systemen linearer Gleichungen," *Bull. Int. Acad. Pol. Sci. Lett*., A, vol. 35, pp. 335-357, 1937.

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

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

[17] S. Matej and R.M. Lewitt, "Practical considerations for 3-D image reconstruction using spherically symmetric volume elements," *IEEE Trans. Med. Img.*, vol. 15, no. 1, pp. 68-78, 1996.

[18] S. Matej and R.M. Lewitt, "Efficient 3D grids for image reconstruction using spherically-symmetric volume elements," *IEEE Trans. on Nucl. Sci.*, vol. 42, no 4, pp 1361-1370, 1995.

[19] 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. Bio*l, Vol. 39, pp. 355-367, 1994.

[20] K. Mueller, R. Yagel and J.J. Wheller "Accurate low-contrast 3D cone-beam reconstruction with algebraic methods," (in review), 1997.

[21] 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 Transactions on Medical Imaging*, vol. 16, no. 2, pp. 223-230, April 1997.

[22] R. Ning and S.J. Rooker, "Image intensifier-based volume angiography imaging system: work in progress," *Proc. SPIE, vol. 1896, Medical Imaging 1993: Physics of Medical Imaging*, pp. 145-155, 1993.

[23] D. Ros, C. Falcon, I Juvells, and J. Pavia, "The influence of a relaxation parameter on SPECT iterative reconstruction algorithms," *Phys. Med. Biol.*, no. 41, pp. 925-937, 1996.

[24] B. Smith, "Image reconstruction from cone-beam projections: necessary and sufficient conditions and reconstruction methods," *IEEE Trans. Med. Img.*, vol. 4, no. 1, pp. 14-25, 1985.

[25] D. Saint-Felix, Y. Trousset, C. Picard, C. Ponchut, R. Romeas, A. Rougee, "In vivo evaluation of a new system for 3D computerized angiography," *Phys. Med. Bio*l, Vol. 39, pp. 583-595, 1994.

[26] 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.

[27] L. Westover, "Footprint evaluation for volume rendering," *Computer Graphics (SIGGRAPH)* , vol. 24, no. 4, pp. 367-376, 1990.