Presented at the *SPIE Medical Imaging Conference 1999*, Physics of Medical Imaging, San Diego

On the Use of Graphics Hardware to Accelerate Algebraic Reconstruction Methods

Klaus Mueller[1,2] and Roni Yagel[1,2]

[1] Department of Computer Science, The Ohio State University, Columbus, OH 43212, USA
[2] Biomedicom, Ltd., Jerusalem, Israel

ABSTRACT

The Algebraic Reconstruction Technique (ART) reconstructs a 2D or 3D object from its projections. It has, in certain scenarios, many advantages over the more popular Filtered Backprojection approaches and has also recently been shown to perform well for 3D cone-beam reconstruction. However, so far, ART's slow speed has prohibited its routine use in clinical applications. Currently, a software implementation requires several hours for a 3D reconstruction, even on modest reconstruction grid sizes. Although one solution to combat these problems would be the time-consuming design of expensive custom accelerator boards, we would rather like to resort to existing and widely available hardware for our purposes. In this sense, we find that ART's main operations, i.e., volume projections and image backprojections, can be performed very rapidly on standard 2D texture mapping hardware, resident in many graphics workstations and PC graphics boards. In this paper, we discuss the use of this hardware in two volume decomposition modes: voxel and slice. Although we find that the speedups obtained in the voxel mode are respectable, the speedups obtained in the slice-mode are tremendous. Here, a quality cone-beam reconstruction on a 128^3 grid can be obtained in less than 2 minutes, which corresponds to a speedup of over 70. Since our rapid ART reconstruction algorithm can be run on the same workstations that are typically used for the viewing of clinical datasets, it is immediately available for routine parallel- and cone-beam CT.

Keywords: Algebraic Reconstruction Technique, ART, SART, Computed Tomography, CT, cone-beam CT, parallel-beam CT, reconstruction from projections, texture mapping hardware.

1. INTRODUCTION

The Algebraic Reconstruction Technique (ART), first proposed by Gordon et. al. [7], is a tomographic reconstruction method which reconstructs a 3D object from its projection images. These projection images may be acquired from any projective imaging modality, such as X-Ray, PET, or SPECT. ART is an iterative method and reconstructs a volumetric object by a sequence of alternating volume projections and correction backprojections. Here, the volume projection measures how close the current state of the volume matches one of the scanner projections, while in the backprojection step a corrective image is distributed onto the volume grid. Many such projection/backprojection operations are typically required to make the volume fit all projections in the acquired set. Different ART variants exist: While the original ART corrects the volume on a ray-basis, Simultaneous ART (SART) [1] corrects the volume only after a whole projection image has been computed.

The iterative process is slow, and this lack of computational speed has so far prevented ART to be used in real-life clinical applications. However, scenarios exist where ART has many advantages over the more commonly used Filtered Backprojection (FBP): It is superior when one does not have a large set of projections available, when the projections are not distributed uniformly in angle, or when the projections are sparse or missing at certain orientations [12]. These scenarios may occur in intra-operative CT, in cardiac CT, or when metal artifacts require the exclusion of some portions of the projection data [23]. While fan-beam CT, and more recently, helical CT [11] have established themselves as routine clinical imaging methods, cone-beam CT is still in a research state. As yet, there are no clinical cone-beam scanners, although a variety of cone-beam algorithms have been proposed in the mid-80s. Present cone-beam algorithms are mostly based on FBP [5][8][21] (see also [17][22] for comparisons and reviews). However, more recent research [14] has demonstrated that ART (with certain modifications) and SART can reconstruct general cone-beam data as well, at high accuracy and even for large cone-angles of up to 60°.

Further author information:

Klaus Mueller (correspondence): 2015 Neil Ave, 789 Dreese Lab, Columbus, OH 43210; Email: muellerk@acm.org; WWW: http://www.cis.ohio-state.edu/~mueller; Phone: (614) 292-0060; Fax: (614) 292-2911

Roni Yagel: Email: yagel@emitf.com; WWW: http://www.cis.ohio-state.edu/~yagel

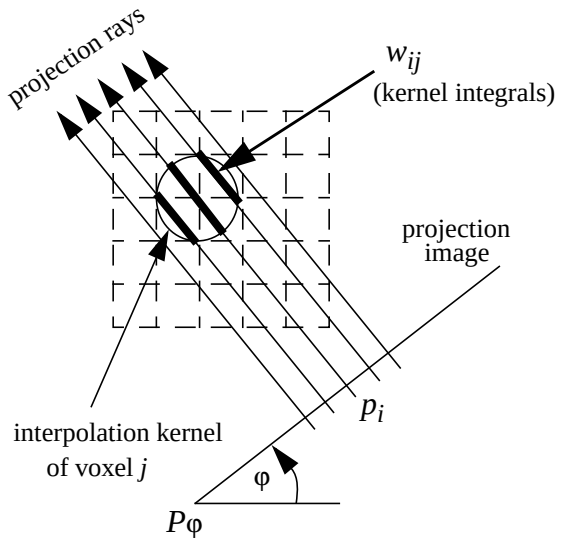
Thus ART possesses great prospects for 3D reconstruction from cone-beam data if its computational speed could be improved. It was already demonstrated in [14] that two to three iterations are sufficient to reconstruct a 3D object of low-contrast. In addition, the required number of projections in ART is typically smaller than for FPB, at least in the theoretical sense [9]. However, still more than 2.5 hours are needed on a modern workstation to reconstruct a 128^3 volume from 80 projections [15]. As a remedy, one could build dedicated ART accelerator boards and incorporate those into the clinical scanners, along with the usual custom DSP (Digital Signal Processing) chips which already run the FBP algorithm extremely fast. However, designing and configuring special chips or boards to implement our ART and SART algorithms would be a rather expensive and tedious task, and would produce narrow devices with little room for modifications and adaptations of the algorithms, hampering the evolution of technology. Fortunately, today's widely available graphics workstations provide us with a better option: The graphics hardware resident in these workstations is especially designed for fast projection operations, which are the main ingredients of the algebraic algorithms. In a different approach, this hardware was also used by Cabral et. al. to accelerate the FBP algorithm [3]. Another plus of this hardware choice is the growing availability of these machines in hospitals, where they are more and more utilized in the daily task of medical visualization, diagnosis, and surgical planning. The feature of these graphics workstations that we will rely on most is *texture mapping*, a technique that is commonly used to enhance the realism of the polygonal graphics objects by painting pictures onto them prior to display. Texture mapping is not always, but often, implemented in hardware, and runs at fill rates of over 100 Megapixels/sec. However, hardware texture mapping is not limited to graphics workstations only, many manufacturers nowadays offer dedicated graphics boards with texture-mapping capabilities that can be added to any modern PC.

In the following sections, we will describe two algorithms that make use of the 2D texture mapping hardware to accelerate ART's projection and backprojection operations. The two algorithms differ by the level of granularity at which the volume is decomposed for texture mapping purposes. While the first algorithm projects the volume on a voxel basis, the second approach works on a volume-slice basis. The voxel-based approach reconstructs a 128^3 volume from 80 projections in about 50 minutes, which corresponds to a speedup of 3 compared to an optimized software implementation [9]. The second version is considerably faster: It reconstructs the same dataset in about 2 minutes — a speedup of over 70 with respect to the fast software solution. The new rapid ART implementation is termed *T*exture-*M*apping hardware *A*ccelerated ART (TMA-ART), and all software was written using the widely accepted OpenGL API (Application Programming Interface). This standard interface allows easy software portability to any medium-range graphics workstation or PC with graphics board. Although we will describe our algorithm in terms of cone-beam reconstruction, it naturally also applies to parallel-beam reconstruction as a private case.

2. PRELIMINARIES

ART is inherently a pixel-based reconstruction algorithm, i.e., a grid correction is based on the projection and backprojection of a single image pixel at a time. This is usually performed via image-order projection methods, i.e., the volume is projected by casting rays into the volume, pixel by pixel. Graphics hardware, however, uses object-order projection methods, i.e., the graphics objects are projected to the screen where their contributions accumulate into the final image. This image-based projection makes it tedious and inefficient to implement a pixel-based approach, such as ART. Hence, we have opted not to implement ART, but the related method SART, which performs grid correction only after an entire projection image is available. Doing so is, in fact, to our advantage: It was shown in [14] that SART produces cone-beam reconstructions that are of similar quality than those obtained with cone-beam ART, but without requiring depth-adaptive interpolation filters.

We shall now briefly describe the individual steps of the SART algorithm, producing a decomposition that will later be emulated in the graphics pipeline. Recall that SART reconstructs a volume by a series of grid projections and grid corrections, with the latter being implemented as image backprojections. Consider Fig. 1a, that shows how a grid voxel (i.e., a volume element) contributes to a projection and a backprojection, respectively. We can think of the volume to be decomposed into a field of 3D interpolation kernels, with one such kernel placed at each grid voxel location and attenuated by the voxel's value v_j. During projection (and backprojection), this field can be thought of being traversed by projection rays r_i, connecting the X-ray source with the image pixels p_i. The influential weight w_{ij} that a voxel has on one of these rays is the integral of the traversed voxel kernel function. During projection, the rays traverse the voxel kernels, weighting all voxel values by the respective voxel kernel integrals, and accumulating the weighted voxel contributions into the ray integrals. A projection image results. The grid correction factors are calculated by subtracting the ray integrals in this projection image from the pixel values p_i^S in the scanner image $P\varphi$, obtained at the same orientation φ than the computed projection. This yields a grid correction image. During backprojection, the rays traverse the volume again, but this time the voxel kernel integrals weight the correction factors in the correction image. Each voxel adds all contributions so obtained and normalizes them by the sum of weights. The resulting voxel correction is then added to the present voxel value.



(a)

The SART algorithm

Initialize volume

Until convergence

Select a projection $P\varphi$

Image projection:

Compute line integrals through all p_i of $P\varphi$

(the inner sums in equation (1))

Correction image computation:

For all p_i, subtract line integrals from scanner p_i

(expression in parentheses in nominator of (1))

Image backprojection:

Distribute corrections onto grid voxels

(the remainder of (1))

(b)

Fig. 1. (a) The interpretation of a voxel weight factor w_{ij}, (b) the steps of the SART algorithm.

Stated more formally, the SART correction for voxel j, to be performed for each grid correction step k, is written as follows:

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

Here, λ is a relaxation factor, typically chosen «1.0. Fig. 1b illustrates the algorithm, decomposed in its constituents.

The software implementations typically perform grid projection and backprojection by a procedure termed *splatting* [24]. In splatting, the kernel integrals (i.e., the weights w_{ij}) are pre-computed and stored in a table, called the kernel *footprint*. If the kernel function is radially symmetric, such as the Gaussian function, we can use the same footprint at all viewing orientations. Two variants of the splatting algorithms are commonly used for ART: an object-order approach and an image-order approach. In the object-order approach [13][15], a footprint is associated with each grid voxel and mapped to the screen, weighted by the voxel's value. The accumulation of all weighted footprints then makes up the projection image (see Fig. 2a). In backprojection, each voxel footprint is again mapped to the screen, but this time it weights the correction image contributions and adds them to the voxel, properly normalized. In the image-order approach, the rays traverse the volume, as described above, and the footprint tables provide the pre-integrated kernel integrals [13][15].

We are now ready to describe the two hardware implementation variants of SART. We will start with the voxel-based approach, and then move to the slice-based variant.

3. VOXEL-BASED TEXTURE MAPPING HARDWARE ACCELERATION OF ART

The voxel-based TMA-ART approach emulates the object-order software implementation of SART, i.e., each voxel is associated with a footprint. The texture mapping hardware is used to map these footprints onto the screen. Similar to the approach described in [4], we associate each voxel with an identical square polygon of side length 2r, where r is the radial extent of the

kernel function, and map the footprint function as a texture onto it. While the hardware projection step is similar to the software projection step, the hardware backprojection step is different. We now describe both steps.

3.1 Projection

For the voxel projection, we set the color of each polygon to the value of the corresponding voxel. Recall that a footprint is due to the integration of a kernel function in the direction of the viewing rays. Hence, for cone-beam projection, the graphics hardware must rotate each polygon to a position perpendicular to the cone-beam ray that traverses the center of the footprint polygon (see Fig. 2b). Following this rotation, the polygon is perspectively projected to the screen. The polygon color is modulated (=weighted) by the footprint texture (i.e. the footprint function) when the polygon is rasterized. The accumulation of all projected footprint polygons then yields the projection image. Note that, for parallel-beam projection, all footprint polygons are rotated by the same amount, parallel to the screen. Also note that only voxels within the spherical reconstruction region must be projected. Furthermore, we only need to project voxels that have non-zero values.

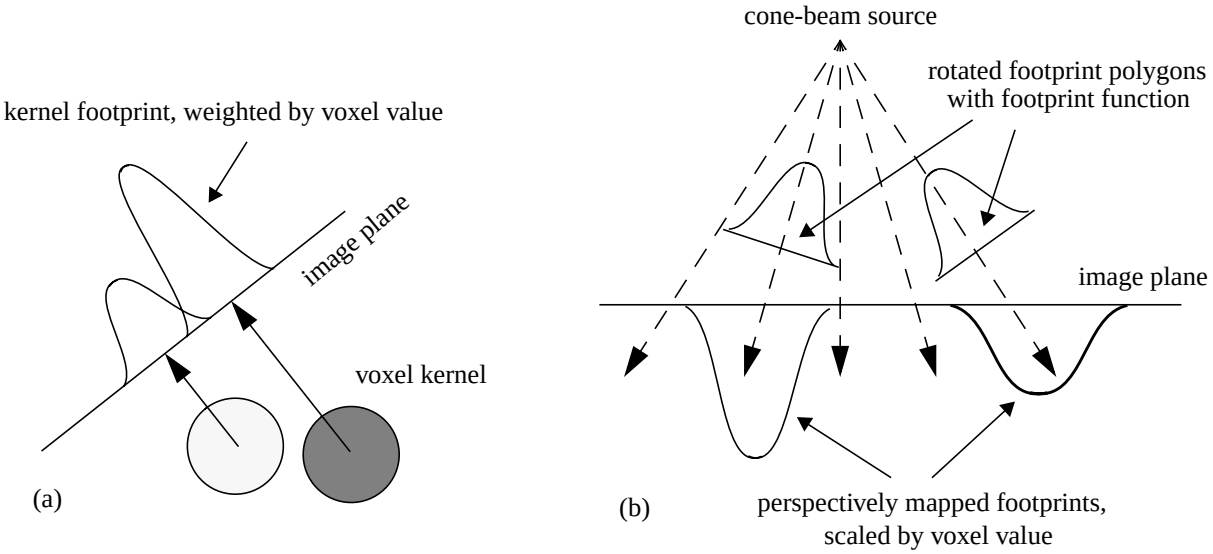


Fig. 2. (a) Screen projection of two representative kernel footprints, weighted by the value of the corresponding voxel (shown in 2D). (b) Cone-beam screen projection of two representative voxel footprint polygons in voxel-based TMA-ART (shown in 2D). Since the footprints represent the integrated kernel function in the direction of the traversing rays, the footprint polygons must be rotated perpendicular to the cone-beam ray traversing the center of the footprint polygon. This provides a good approximation to the true ray integrals. The footprint polygons are perspectively mapped onto the image plane by the texture mapping hardware, and the resulting footprints are scaled by the polygon color, set to the voxel value.

3.2 Correction image computation

To compute the correction image, we read the framebuffer, subtract the read projection image from the scanner image, scale it by the relaxation factor λ, and divide it by the corresponding weight image. The weight images are computed beforehand for each viewing angle, by projecting the footprint polygons of all relevant voxels, with colors set to unity, to the screen. None of these operations can be performed in hardware, since they either exceed the valid pixel range of the framebuffer or cannot be performed efficiently using the provided hardware. Hence, these operations are done on the CPU. Since the texture map and the framebuffer can only hold values in the range [0.0...1.0], but the correction image may have values in the range [-1.0...1.0], we must scale and translate the values in the correction image to the [0.0...1.0] interval. Note that, in this way, the values in the volume slices are always in the range [0.0...1.0].

3.3 Backprojection

Although the replication of the projection step in hardware is straightforward, the backprojection step requires an approach that is different from that of the software implementation. But let us first discuss a strategy that emulates the software algorithm

exactly. In the software implementation, we simply map the footprint polygons to the screen, using the same procedure than in the projection step, and weight the pixel values in the correction image according to the projected footprint function. The weighted correction values are then integrated and normalized by the sum of respective footprint weights to yield the additive voxel correction. A possible hardware implementation of this process is as follows: First, the hardware loads the framebuffer's alpha channel with the correction image, and the framebuffer blending mode is set to $R_{fb}=R_{poly}\cdot\alpha_{fb}$. Here, R_{fb} are the pixel values in the Red framebuffer channel after footprint projection, R_{poly} is the Red color of the polygon, set to unity, and α_{fb} are the pixel values in the framebuffer opacity channel (set to the correction factors). This blending function achieves the multiplication of the correction factors with the footprint weights. After footprint projection, the framebuffer pixels that fall within the footprint's extent are read into memory. An integration of these pixel values then yields the nominator of the voxel correction in equation (1). However, we need the denominator as well. For this purpose, we again project the footprint, but this time we use the blending function $R_{fb}=R_{poly}\cdot 1$. Integrating the resulting pixel values yields the denominator of the voxel correction equation (1). We then perform the division and update the grid voxel. Note, that we need to scale the voxel updates back from the interval [0.0...1.0] to [-1.0...1.0]. Also note, that we must clip the volume values to an interval [0.0...1.0] after the update.

This approach requires two framebuffer reads, two integrations, and one division per voxel. In particular, the two former operations are rather expensive, and performing them for each voxel would offset all advantages gained by the hardware acceleration. Fortunately, we can revert the direction of the footprint mapping. Instead of mapping a voxel footprint onto the correction image, we can alternatively map the correction pixel footprints onto the volume slices. In this scheme (see Fig. 3), each volume slice is updated separately. First, each (non-zero) correction image pixel is associated with a texture polygon. Again, the texture is given by the integrated kernel function, and the polygon has a side length of $2r$. The Red color of each such polygon is set to the correction factor and the Blue color is set to unity. Before projecting a polygon to the volume slice, represented by the screen, it is aligned perpendicular to the cone-beam ray that traverses its center. (For parallel-beam all polygons are aligned parallel to the view plane.) All aligned texture polygons are then perspectively projected to the screen. As a result, the Red color channel holds the nominator of equation (1) and the Blue channel holds the denominator. Following, the framebuffer is read into memory and the division Red/Blue channel image is performed. This yields the correction image for that slice. We transform and clip

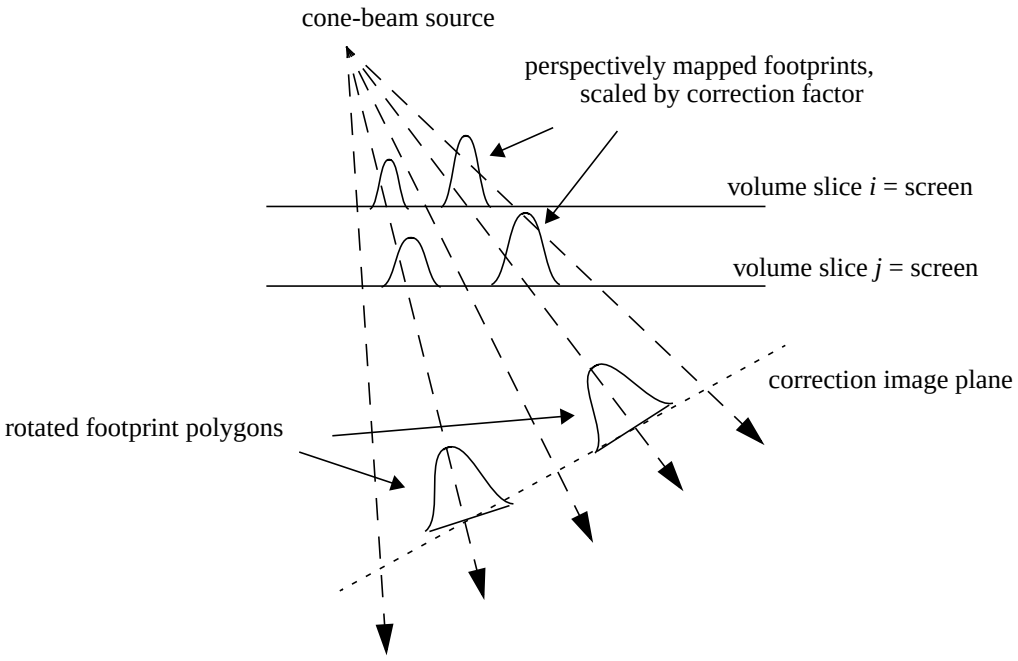


Fig. 3. Projecting the correction factors onto two representative volume slices in voxel-based TMA-ART. Each non-zero correction factors is associated with a footprint polygon. The footprint polygons are centered at the corresponding correction image pixels in the correction image plane (= the projection plane of the previous projection step). Each polygon is then rotated to align it perpendicular to the cone-beam ray that traverses its center. Then the screen is aligned with the first volume slice, and all polygons are projected. The slice correction image accumulates in the framebuffer, and after scaling and clipping the image values, this image is added to the corresponding volume slice in memory. This procedure is repeated for every volume slice, moving the screen to the corresponding volume slice location.

the correction image values, as mentioned above, and add it to the volume slice. We then proceed to the next volume slice. In this approach, only one framebuffer read and one image division is required per slice. The correction integration per voxel is done in hardware. Hence, this algorithm is considerably faster than the first, without a loss of accuracy.

Note that for both projection and backprojection, the orientation of the projected and updated volume slices is dependent on which volume axis is most perpendicular to the screen. For example, if the screen is most perpendicular to the x-axis, then the volume is decomposed into the slices that are aligned with the y-z plane.

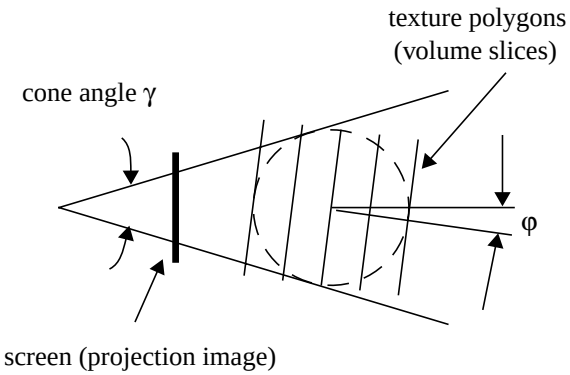
4. SLICE-BASED TEXTURE MAPPING HARDWARE ACCELERATION OF ART

The voxel-based approach is considerably faster than the software implementation. It, however, requires a separate polygon rasterization for each voxel (or correction pixel) to be texture-mapped. This produces computational overhead, since the polygons overlap on the image plane, requiring the screen pixels to be updated more than once. The footprint approach is efficient if the number of projected polygons is sparse with respect to the volume size, i.e., a large percentage of voxels have a value of zero. However, this is usually not the case in the iterative ART. Thus, we need to increase the granularity of our projected elements in order to minimize the number of pixel updates. This can be achieved by projecting an entire volume slice (or correction image), and not just a voxel (pixel) footprint. We will now describe this approach.

4.1 Projection

The slice-based TMA-ART approach decomposes the volume into slices and treats each slice separately. In grid projection (shown in Fig. 4), each slice is associated with a square polygon with the volumetric slice content texture-mapped onto it. A projection image is obtained by accumulatively rendering each such polygon into the framebuffer. Here, a bilinear interpolation kernel is used by the hardware to resample the texture image into screen coordinates.

Note that the ray integrals so computed are equivalent to the ray integrals obtained in a software solution that uses a trilinear interpolation filter and samples only within each volume slice. In this respect, the integration follows the trapezoidal rule and is similar to that obtained by Joseph's algorithm [10]. Note that since the distance between sample points is not identical for every ray (due to the perspective distortion), we have to normalize the projection image for this varied distance. This can be conveniently achieved by normalizing the scanner images by the inverse amount in a pre-processing step.



Sliced-Based TMA-ART Projection algorithm

Rotate texture polygons by projection angle φ

Texture map the volume slices onto texture polygons

Project textured polygons onto the screen

Accumulate screen contributions at each pixel

Fig. 4. Grid projection with slice-based TMA-ART.

4.2 Correction image computation

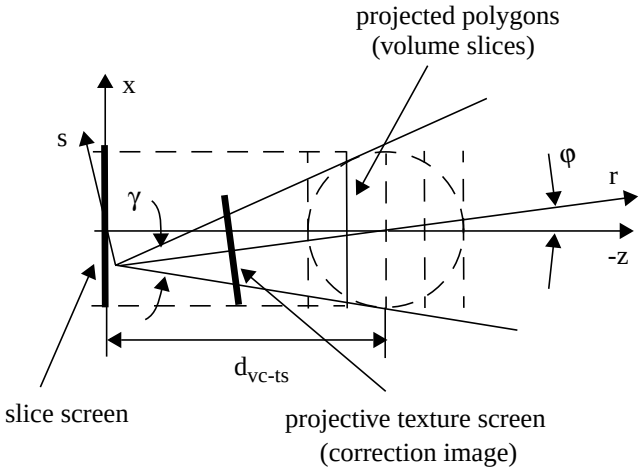
After a projection image has been generated, the correction image is computed. First, the projection image is subtracted from the scanner image. The resulting image is then divided by the weight image at that orientation. (The weight images are computed beforehand by projecting a volume in which all voxels within the spherical reconstruction region have been set to 1.0.) Similar to the voxel-based approach, we must scale and shift the values in the correction image to the range [0.0...1.0]. At least for now, all computations with regards to the correction image are performed on the CPU.

4.3 Backprojection

In backprojection (shown in Fig. 4), we need to distribute a correction image onto the volume slices. This is achieved by associating each volume slice with the framebuffer onto which the correction image, mapped to a polygon, is rendered. (This is the inverse situation of Fig. 4.) Although projection is simple, backprojection is not as straightforward, since here the main viewing direction is not always perpendicular to the screen. This, however, is required by the graphics hardware. To enable this viewing geometry, we implemented a virtual slide-projector (using the projective texture approach of Segal et. al. [18]) that shines the correction image at the oblique projection angle onto a polygon, which in turn is orthographically viewed by the framebuffer. This is shown in Fig. 4 for one representative volume slice. The correction image is perspectively mapped, according to the cone-geometry, onto the volume slice that has been placed at the appropriate position in the volume. This "slide" projection is then viewed by the screen. After the correction image has been projected onto the slice screen, we must scale and translate the values of the screen image back into the [-1.0...1.0] range. The resulting image is then added to the volume slice in memory, limiting the voxel values to an interval of [0.0...1.0] and setting voxels outside the spherical reconstruction region to zero.

Let us now explain this slide-projector approach in some more detail. In OpenGL, a polygon is represented by three or more vertices. When the polygon is projected onto the screen, the coordinates of its vertices are transformed by a sequence of matrix operations, as shown in Fig. 6 (For more detail on these fundamental issues refer to [6] and [16].)

A texture is an image indexed by 2D coordinates in the range [0.0...1.0, 0.0...1.0]. When a texture is mapped onto a polygon, the polygon's vertices are associated with texture coordinates [s,t], as shown in Fig. 7. The viewing transformation of the polygon vertices yields a closed region on the screen. In a process called *scan conversion*, all pixels inside this region are assigned



T_1: translate by d_{vc-ts} **R**: rotate by φ **T_1^{-1}**: translate by $-d_{vc-ts}$ **P**: perspective mapping
S: scale by 0.5 **T_3**: translate by 0.5

Slice-based TMA-ART Backprojection algorithm

Set texture matrix to $\mathbf{TM}=\mathbf{T_1}\cdot\mathbf{R}\cdot\mathbf{T_1^{-1}}\cdot\mathbf{P}\cdot\mathbf{S}\cdot\mathbf{T_3}$

For each volume slice

Associate 3D-texture coordinates with each vertex, set r-coordinate to z-coordinate

Render the texture-mapped polygon onto the screen
(use **TM** to map texture coordinates onto texture screen)

Scale and translate the values in the screen image back into the interval [-1.0...1.0].

Add the resulting image to the respective volume slice (limit the summed values to the interval [0.0...1.0]).

Fig. 5. Grid backprojection with slice-based TMA-ART.

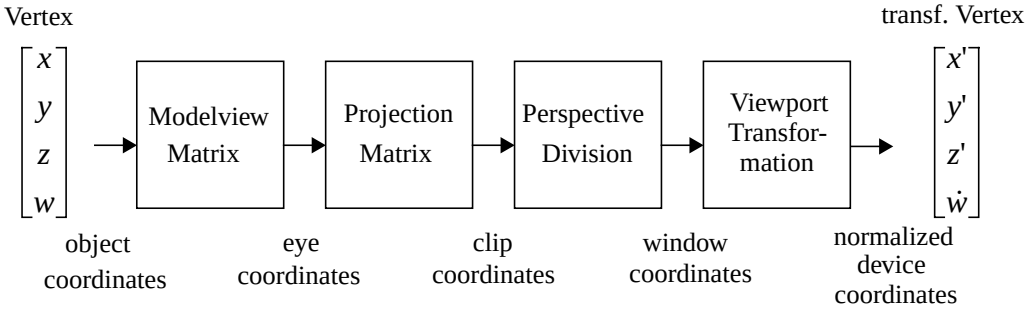


Fig. 6. Stages of vertex transformation in OpenGl.

(via interpolation) texture coordinates within the range assigned to the bounding vertices. Note that this transformation can lead to a stretching or shrinking of the texture.

The texture mapping coordinates need not be two-dimensional. As a matter of fact, they can be up to four-dimensional (involving a homogeneous coordinate), just like the vertex coordinates. In addition, OpenGL provides a transformation facility, similar to the one supplied for vertex transformation, with which the interpolated texture coordinates can be transformed prior to indexing the texture image. We can use this facility to implement our virtual slice projector.

The algorithm proceeds as follows. First, we create an array of n square texture coordinate polygons with vertex coordinates (s,t,r). Here, we set the (s,t) coordinates to (n, n), i.e., the extent of the volume slices. The r-coordinate we vary between $[d_{vc\text{-}ts}\text{-}n/2, d_{vc\text{-}ts}+n/2]$. ($d_{vc\text{-}ts}$ is the distance of the source to the volume center.) Refer now back to Fig. 6, where we show the decomposition of the texture transformation matrix. The Modelview matrix is set to the product $\mathbf{T_1}\cdot\mathbf{R}\cdot\mathbf{T_1}^{-1}$, i.e., each polygon is rotated about the volume center by the viewing angle φ. The Projection matrix is set to a perspective mapping of the texture coordinates onto the projective texture screen. After the perspective divide, the texture coordinates would be in the range [-1.0...1.0]. Since we can only index the texture image within a range of [0.0...,1.0], we need to scale and translate the texture perspective texture coordinates prior to indexing. This is achieved by incorporating a scale and translation given by $\mathbf{S}\cdot\mathbf{T_3}$ into the Projection matrix.

We can now perform the backprojection of the correction image, represented by the texture, onto the volume slices. Let us just look at one of the volume slices, represented by polygon P_s with vertex coordinates (n, n, z), which is projected orthographically on the slice screen. Depending on its z-location, the polygon is assigned one of the texture coordinate polygons. When mapping P_s onto the screen, texture coordinates are generated for each pixel within the projected polygon extent. However, these texture coordinates are not used directly to index the correction image, but are first passed through the texture transformation pipeline. The transformed coordinates then index the correction image texture as if this image had been projected onto P_s at the backprojection angle φ.

One should add that this process is not any more expensive than direct texture mapping. Once the texture transformation matrix is compounded, just one hardware vector-matrix multiplication is needed. As a matter of fact, this multiplication is always performed even if the texture transformation is unity.

5. RESULTS

Using both the software implementation of ART (discussed in [14][15]) and the two new hardware-accelerated versions, voxel-based and slice-based TMA-ART, we reconstructed a simulated brain dataset, the 3D extension of the Shepp-Logan phantom [19] (described e.g. in [2], a slice of which is shown in Fig. 8a). Note that since both versions of TMA-ART return similar reconstructions, we only show the images generated with the slice-based approach. Projection sets of 80 cone-beam projections (γ=40˚) of 128^2 pixels each were obtained by analytical integration of the brain phantom and used to reconstruct a 128^3 reconstruction volume in 3 iterations (λ=0.1). To evaluate the effect of the limited framebuffer resolution in TMA-ART, we acquired the brain projection sets at three different feature contrast levels. The original contrast of the main features in the phantom is 2% of the full dynamic range, while the background contrast of the small tumors in the bottom portion of the slice shown in Fig. 8a is only 0.5%. (Note that in all images of Fig. 8, the small dynamic range of the features was stretched into the full displayable range in order to make the features visible.)

Fig. 8b shows a slice (from the same location than that in Fig. 8a) across a volume reconstructed with the software implementation of ART. We observe that very little reconstruction noise is present and that the brain features can be well distinguished. The software implementation uses both floating point arithmetic and floating point buffers throughout the reconstruction process. TMA-ART, on the other hand, also uses floating point arithmetic but only fixed point buffers. The main restriction here is

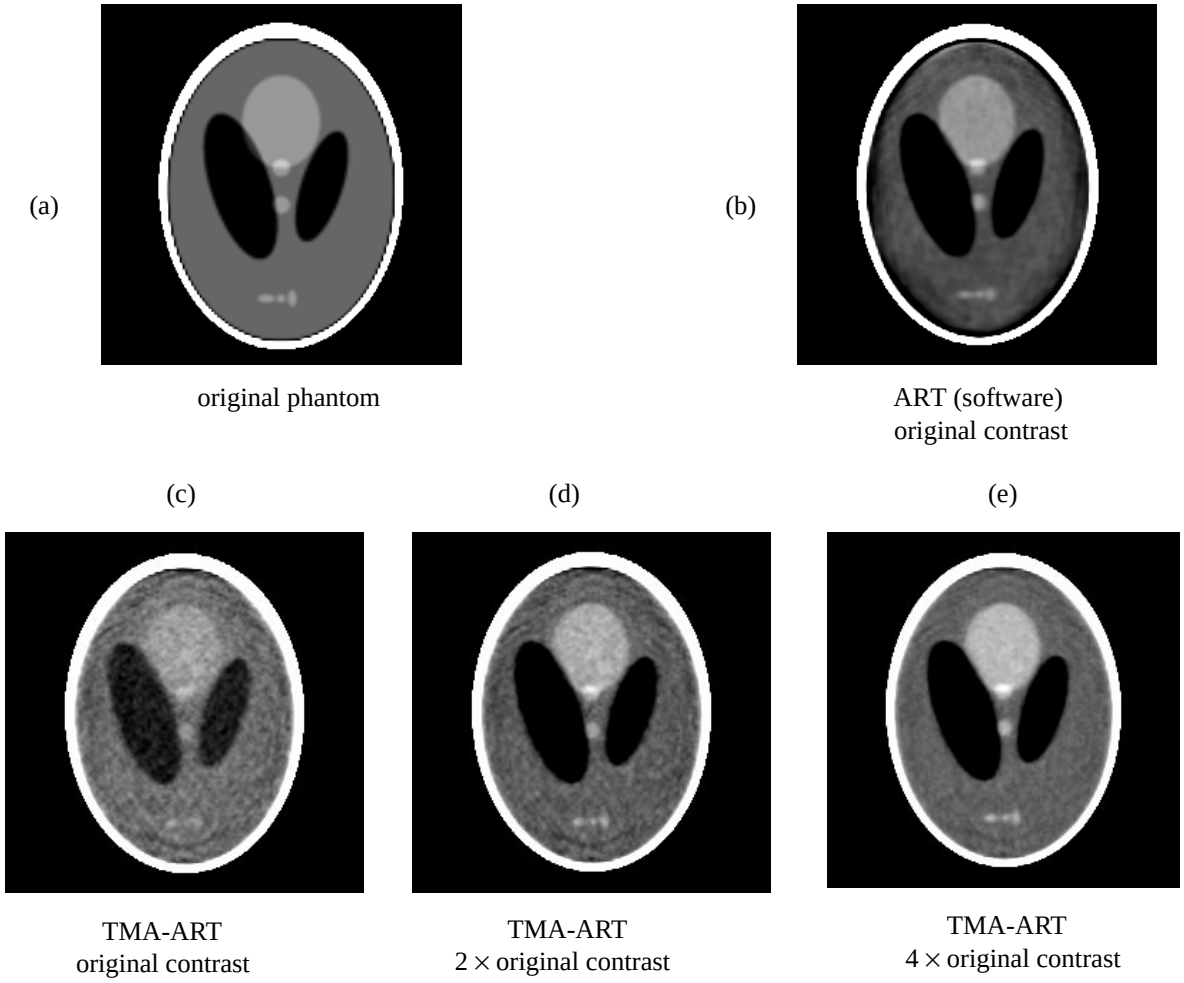


Fig. 8. Slices across reconstruction volumes obtained with different implementations of ART, software and hardware-accelerated. All reconstructions were performed using 80 128×128 projections of the 3D extension of the Shepp-Logan phantom (cone-angle γ=40˚, 3 iterations, 128^3 reconstruction grid, λ=0.1).

the limited resolution of the 12 bit framebuffer (we use an SGI Octane workstation), which can only resolve 1/4096=0.02%. As

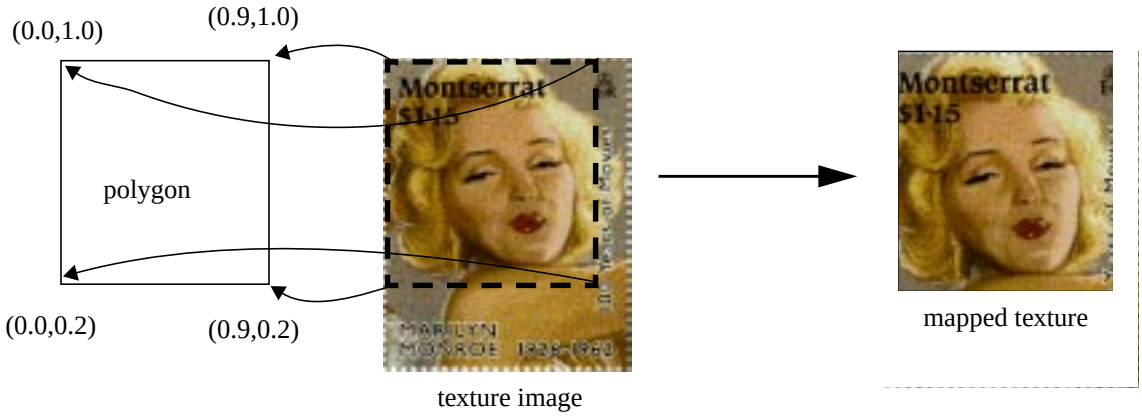


Fig. 7. Texture mapping of an image onto a polygon. The texture coordinates assigned to the polygon vertices are given in parentheses.

Fig. 8c indicates, this is apparently not sufficient to resolve the three small tumors in the bottom portion of the phantom. The limited resolution also gives rise to a somewhat noisy appearance of the reconstructed slice. However, as can be observed in Fig. 8d, TMA-ART manages to resolve slightly higher tumor contrasts of 1% rather well. In addition, only little noise is apparent in the slice of Fig. 8e, where the tumor contrast is 2%. It appears that the limited resolution of the framebuffer and, probably to a lesser extent, the limited resolution of the texture memory, causes reconstruction noise levels equivalent to the 0.5-1% contrast range.

Finally, Table 1 compares the run times for both the optimized software and the two different TMA-ART implementations (i.e., the voxel-based and the slice-based approach). We observe, that by utilizing texture mapping hardware for the grid projection and backprojection operations, dramatic speedups can be achieved: A cone-beam reconstruction of a 128^3 volume from 80 projections can now be performed in about 51 minutes with the voxel-based approach (speedup=3), and in 2 minutes with the slice-based approach (speedup=73), down from the 2.5 hours that were required in the optimized software implementation.

implementation	time / iteration	reconstr. time	speedup
ART (software)	51 min	2.55 h	-
TMA-ART: voxel-based	17 min	51 min	3
TMA-ART: slice-based	42.2 sec	2.1 min	73

Table 1. Run times for one iteration as well as for a complete reconstruction (3 iterations) of different ART implementations. (These timings are needed to produce the images shown in Fig. 8.)

6. CONCLUSIONS

In this paper, we have shown that ART can be accelerated to almost interactive speeds, without building any expensive custom hardware. All that is needed is a standard graphics workstation with 2D texture mapping capabilities or one of many inexpensive texture mapping boards that are readily available for almost any desktop PC. Previously, ART's many qualitative advantages could not be utilized in clinical applications since the computational effort was too high compared to other 2D and 3D reconstruction methods. Our texture-mapping accelerated algorithm closes this performance gap and makes ART available for any clinical setting, with the added benefit that it runs on a platform that can also be used to visualize the reconstructed data.

7. FUTURE WORK

The quality of the reconstructions is currently limited by the resolution of the framebuffer (and, to a lesser extent, that of the texture memory). We find that objects of 1% contrast can be resolved well even with a 12 bit framebuffer. We also find that the reconstruction noise levels are in the 0.5-1% contrast range. This means that once the features exceed this range, the signal-to-noise ratio becomes sufficient for a reconstruction of good quality. It is hoped that hardware manufacturers will supply machines with higher-resolution framebuffers, yielding better signal-to-noise ratios. To parallel this effort, we are currently working on schemes that extend the machine's framebuffer resolution (independently of what it is) by utilizing all color channels and combining them to yield a virtual, higher resolution data word. These schemes extend a 12 bit framebuffer into 16 bits, and initial results look promising.

We are also working to implement other portions of the ART algorithm in hardware, such as the computation of the correction image, the accumulation of the projection image, and the voxel update. In addition, a TMA-ART version for multi-processor PCs and workstations is currently being developed. It is expected that this version will achieve a reconstruction in less than 30 sec.

8. REFERENCES

[1] A.H. Andersen, A.C. Kak, "Simultaneous Algebraic Reconstruction Technique (SART)," *Ultrason. Img.*, vol. 6, pp. 81-94, 1984.

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

[3] B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," *1994 Symposium on Volume Visualization*, pp. 91-98, 1994.

[4] R. Crawfis and N. Max, “Texture splats for 3D scalar and vector field visualization,” *Visualization’93,* pp. 261-266, 1993.

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

[6] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice.* New York: Addison-Wesley, 1990.

[7] 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.

[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] H. Guan and R. Gordon, “Computed tomography using ART with different projection access schemes,” *Phys. Med. Biol*., no. 41, pp. 1727-1743, 1996.

[10] P.M. Joseph, “An improved algorithm for reprojecting rays through pixel images,” *IEEE Trans. Med. Imag.*, vol. 1, no. 3, 1982.

[11] W. Kallender, W. Seissler, and P. Vock, “Single-breath-hold spiral volumetric CT by continuous pateint translation and scanner rotation,” *Radiology* 173P, pp. 414, 1998.

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

[13] 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.

[14] K. Mueller, R. Yagel, and J.J. Wheller, “Accurate 3D cone-beam reconstruction with algebraic methods,” in review, 1998.

[15] K. Mueller, R. Yagel and J.J. Wheller, “Fast implementations of algebraic methods for the 3D reconstruction from cone-beam data,” in review, 1998.

[16] J. Neider, T. Davis, M. Woo, *OpenGL Programming Guide*, Addison-Wesley, 1993.

[17] P. Rizo, P. Grangeat, P. Sire, P. Lemasson, and P. Melennec, “Comparison of three-dimensional x-ray cone-beam reconstruction algorithms with circular source trajectories,” *J. Opt. Soc. Am. A*, vol. 8, no. 10, pp.1639-1648.

[18] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P.E. Haeberli, “Fast shadows and lighting effects using texture mapping, ” *Computer Graphics (Proceedings of SIGGRAPH’92)*, vol. 26, pp. 249-252, 1992.

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

[20] A. Silberschatz, J.L. Peterson, P.B. Galvin, *Operating Systems*, Addison-Wesley Publishing Company, 1991.

[21] 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.

[22] B. Smith, “Cone-beam tomography: recent advances and a tutorial review,” *Optical Engineering*, vol. 29, no. 5, pp. 524-534, 1990.

[23] G. Wang, D.L.Snyder, J.A. O’Sullivan and M.W. Vannier, “Iterative Deblurring for CT Metal Artifact Reduction,” *IEEE Trans. Med. Img.*, vol. 15, pp. 657-664, 1996.

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