Rapid 3-D Cone-Beam Reconstruction with the Simultaneous Algebraic Reconstruction Technique (SART) Using 2-D Texture Mapping Hardware

Klaus Mueller* and Roni Yagel

***Abstract*—Algebraic reconstruction methods, such as the algebraic reconstruction technique (ART) and the related simultaneous ART (SART), reconstruct a two–dimensional (2-D) or three–dimensional (3-D) object from its X-ray projections. The algebraic methods have, in certain scenarios, many advantages over the more popular Filtered Backprojection approaches and have also recently been shown to perform well for 3-D cone-beam reconstruction. However, so far the slow speed of these iterative methods have prohibited their routine use in clinical applications. In this paper, we address this shortcoming and investigate the utility of widely available 2-D texture mapping graphics hardware for the purpose of accelerating the 3-D algebraic reconstruction. We find that this hardware allows 3-D cone-beam reconstructions to be obtained at almost interactive speeds, with speed-ups of over 50 with respect to implementations that only use general-purpose CPUs. However, we also find that the reconstruction quality is rather sensitive to the resolution of the framebuffer, and to address this critical issue we propose a scheme that extends the precision of a given framebuffer by 4 bits, using the color channels. With this extension, a 12-bit framebuffer delivers useful reconstructions for 0.5% tissue contrast, while an 8-bit framebuffer requires 4%. Since graphics hardware generates an entire image for each volume projection, it is most appropriately used with an algebraic reconstruction method that performs volume correction at that granularity as well, such as SART or SIRT. We chose SART for its faster convergence properties.**

***Index Terms*—Algebraic reconstruction technique (ART), computed tomography, cone beam reconstruction, hardware acceleration, simultaneous algebraic reconstruction technique (SART), three—dimensional reconstruction.**

I. Introduction

THE ALGEBRAIC reconstruction technique (ART), first proposed by Gordon *et al.* [8], tomographically reconstructs a three–dimensional (3-D) object from its projection images. These images may be obtained from any projective imaging modality, such as X-Ray, positron emission tomography, or single photon emission computed tomography. ART is an iterative method and reconstructs the volumetric object by a sequence of alternating volume projections and correction backprojections. The volume projection measures how close the current state of the volume matches the corresponding scanner projection, while in the backprojection step a corrective image is distributed onto the volume. 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) [2] corrects the volume only after a whole projection image has been computed.[1]

In this paper, we concentrate on reconstruction from cone-beam data (parallel-beam reconstruction can be treated as a special case). Although there are no specially designed clinical cone-beam scanners as yet, the advent of 3-D reconstruction angiography using C-arm scanners (see, e.g., [1]) has recently brought cone-beam computed tomography (CT) into the arena of real clinical application. Another recent application of cone-beam CT is in radiotherapy: In an approach termed tomotheraphy [14], the MV radiation unit is not only used to administer and measure the radio-therapeutical dose, it is also employed to collect the necessary data for a 3-D reconstruction of the patient immediately before treatment. Using this 3-D reconstruction one can then register the treatment plan with the patient's position to ensure optimal tumor targeting.

Although the majority of the proposed cone-beam algorithms are based on filtered backprojection (FBP) (refer to, e.g., [22], [25] for comparisons and reviews), more recent research [16] has demonstrated that both 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°. But the iterative process is slow [17], and this lack of computational speed has so far prevented ART from being used in real-life clinical applications. This is unfortunate since there are quite a few scenarios in which ART has advantages over the more commonly used FBP. For example, the use of ART seems advantageous 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 [3], [11]. Scenarios of this sort occur in both 3-D reconstruction angiography and tomotheraphy. In the latter, one can simply not obtain a large number of MV projections due to the enormous patient dose, and recent research has shown [20]

Manuscript received May 5, 1999; revised September 6, 2000. The Associate Editor responsible for coordinating the review of this paper and recommending its publication was C. Crawford. *Asterisk indicates corresponding author.*

*K. Mueller was with the Department of Computer and Information Science, Ohio State University, Columbus, OH 43210 USA and Biomedicom, Ltd., Manachet Technology Park Malha, Jerusalem 91487, Israel. He is now with the Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400 USA (e-mail: muellerk@acm.org).

R. Yagel was with the Department of Computer and Information Science, Ohio State University, Columbus, OH 43210 USA and Biomedicom, Ltd., Manachet Technology Park Malha, Jerusalem 91487, Israel. He is now with Insight Therapeutics, Petach Tikva 49170, Israel.

Publisher Item Identifier S 0278-0062(00)10616-0.

[1]For the remainder of this section, we will use the term ART to stand for both variants of algebraic reconstruction methods: ART and SART.

that ART can produce a reconstruction with good feature delineation even with just 24 MV projections. On the other hand, in 3-D reconstruction angiography one may want to reconstruct a four–dimensional (4-D) volume of the coronary arteries in a beating heart, and the number of projections that can be obtained with a safe dose of radio-opaque dye may be limited here as well.

Thus, cone-beam ART has a number of clinical applications for which it appears useful, if only its computational speed could be improved. It was already shown in [16] that two to three iterations with 80 projections are sufficient to reconstruct a low-contrast 3-D object. However, still more than 1.5 hrs are needed on a modern workstation to reconstruct a 128^3 volume from 80 projections. As a remedy, one could build dedicated ART computer boards and incorporate those into the clinical scanners, along with the usual custom digital signal processing (DSP) 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 it 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 another option, as the graphics hardware resident in these workstations is especially designed for fast projection operations, the main ingredients of the algebraic algorithms. A 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 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/s. However, hardware texture mapping is not limited to graphics workstations only, many manufacturers offer graphics boards with texture-mapping capabilities that can be added to any modern PC.

In this paper, we will thoroughly investigate the utility and applicability of texture mapping hardware for the purpose of 3-D reconstruction with algebraic methods. Earlier, this hardware was also used by Cabral *et al.* to accelerate the FBP algorithm [5]. Our programs were written using the widely accepted OpenGL application programming interface (API) [21] and can easily be reproduced to run on any medium-range graphics workstation or PC with graphics board. Since graphics hardware generates an entire image for each volume projection, it is most appropriately used with an algebraic reconstruction method that performs volume correction at that granularity as well, such as SART.[2] This constraint does not impose a restriction in terms of the reconstruction result, since it was demonstrated in [16] that both cone-beam ART and SART deliver reconstructions of similar quality at similar convergence rates. We, hence, refer to our proposed approach as texture-mapping hardware accelerated *SART* (or *TMA-SART*).

[2]We could have chosen SIRT [7] as well, but its convergence rate is much slower than that of SART.

The outline of the paper is as follows: After a brief introduction to SART in Section II, we describe the general approach of TMA-SART in Section III. Then, in Section IV, we extend the functionality of TMA-SART to improve accuracy and also speed. Finally, Section V presents results, and Section VI discusses the prospects, impact, and future of TMA-SART in light of current graphics hardware trends.

II. Preliminaries

In tomographic reconstruction with algebraic methods it is our goal to solve the following simultaneous equation system[3]:

$$P_i = \sum_{j=1}^{N} w_{ij} v_j \tag{1}$$

We would like to recover the values v_j of the $N = n^3$ voxels j in the volume, using the values p_i of the pixels i in the scanner images P_φ, where φ is the source/detector orientation at which the projection was taken by the scanner, assuming a planar source/detector orbit (without loss of generality). In (1), a w_{ij} is the weight with which a voxel j contributes its value to a pixel i. SART solves this equation by an iterative procedure, in which the correction/update for a voxel j, to be performed for each volume correction step k, is written as follows:

$$v_j^{(k)} = v_j^{(k-1)} + \lambda \frac{\displaystyle\sum_{p_i \in P_\varphi} \left(\frac{p_i - \displaystyle\sum_{l=1}^{N} w_{il} v_l^{(k-1)}}{\displaystyle\sum_{l=1}^{N} w_{il}} \right) w_{ij}}{\displaystyle\sum_{p_i \in P_\varphi} w_{ij}}. \tag{2}$$

In (2), λ is a relaxation factor, typically chosen $\ll 1.0$. The collective voxel update procedure of SART can be broken down into several steps, illustrated in Fig. 1.

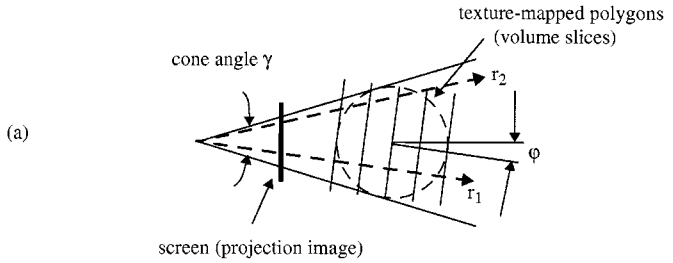
A line integral can be computed by sampling the volume at equidistant locations using some interpolation kernel (trilinear, Gaussian, or cubic spline) and forming the integral via the trapezoidal rule [2], [10]. Alternatively, one can take an approach in which one thinks of the volume as being decomposed into a field of (overlapping) 3-D interpolation kernels, with one such kernel placed at each voxel location (the grid line intersections) and attenuated by the voxel's value v_j. The weight w_{ij} that a voxel has on a ray r_i is then the line integral of the traversed voxel kernel function [12]. Using this representation, volume projection and backprojection can be performed by a procedure termed *splatting* [26], in which the kernel integrals are preintegrated into tables (so-called *footprints*) and mapped to the image plane where they accumulate into the projection image or retrieve the voxel corrections (more detail is given in [17]). The less discretized integration of the splatting methods yields more accurate weight factors and, as a consequence, more accurate projections/backprojections.

[3]We will use the following terminology: the basic elements of the reconstructed volume are the voxels ν while the bins in the projection images are referred to as pixels p.

SART algorithm

Initialize volume
Until convergence
  Select a projection P_φ
  Image projection:
    Compute line integrals for all rays r_i starting at pixels p_i of P_φ
    (the $\sum_{l=1}^{N} w_{il} v_l^{(k-1)}$ in (1) are the line integrals, the $\sum_{l=1}^{N} w_{il}$ are closely related to the ray lengths)
  Correction image computation:
    (expression in parentheses in the nominator of (1))
    For all p_i
      Subtract the calculated line integrals from the p_i's in the acquired images
      Normalize by ray length
  Image backprojection:
    Distribute normalized corrections, weighted by w_{ij}, onto voxels
    Normalize accumulated corrections by the sum of weights $\sum_{p_i \in P_\varphi} w_{ij}$
    Scale by λ

Fig. 1. The steps of the SART algorithm.



(a)

(b)

TMA-SART Projection Algorithm

Initialize software accumulation buffer
For each of the n volume slices
  Rotate associated polygon by orientation angle φ
  Texture map volume slice content onto polygon
  Project texture-mapped polygon onto the screen
  Read projection image from framebuffer
  Add projection image to software accumulation buffer

Fig. 2. Projection with TMA-SART. (a) Projection geometry [two-dimensional (2-D) case shown]. (b) Algorithm pseudocode.

Although it is possible to simulate the splatting approach in hardware, using polygon-mounted, voxel-weighted texture maps for each voxel's kernel footprint, this approach tends to be rather slow, since the granularity of this approach (on the order of voxels) is too small to be efficient [18]. We now describe an approach with higher granularity (on the order of volume slices) that offers more promise.

III. TMA-SART Components

In TMA-SART, some of the blocks in Fig. 1 can be accelerated by the graphics hardware, while others have to be performed on the CPU. We will now describe the basic TMA-SART algorithm in terms of the decomposition of Fig. 1.

A. Projection

TMA-SART decomposes the volume into n slices and treats each slice separately. In volume projection [shown in Fig. 2(a)], each slice is associated with a square polygon that has the volumetric slice content texture-mapped onto it. Rotating this texture-mapped polygon by the scanner orientation angle φ and perspectively projecting it with cone angle γ maps the voxels in this volume slice, properly weighted, onto the image pixels.[4]

Fig. 2(b) illustrates the projection algorithm, in which the slice-to-screen mapping is performed by the graphics hardware, but the accumulation of projections is done in software, due to the limited bit resolution of the framebuffer. After all n texture-mapped polygons have been accumulated in the software buffer, it contains the volume projection at projection angle φ.

Note that the hardware uses a bilinear interpolation kernel to resample the texture image into screen coordinates. Thus, the ray integrals so computed are equivalent to the ray integrals obtained in a software solution that uses a trilinear interpolation filter in conjunction with raycasting and samples the rays only within each volume slice. In this respect, the integration follows the trapezoidal rule and is similar to that obtained by Joseph's

[4]For reconstruction from parallel-beam data one just sets the viewing geometry to parallel projection.

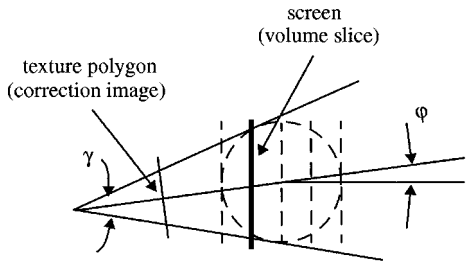


Fig. 3. Impractical backprojection with TMA-SART (2-D case shown): the main viewing axis is not perpendicular to the screen, and the viewing axis does generally not traverse the volume slices at their center.

algorithm [10]. Continuing this analogy with raycasting, note that if we sample only within the volume slices, then the distance between sample points varies depending on the orientation of a ray with respect to the volume slices. A ray that is perpendicular to the volume slices has a sample spacing Δs of 1.0 [ray r_1 in Fig. 2(b)], while for a nonslice perpendicular ray $\Delta s < 1.0$ [ray r_2 in Fig. 2(b)]. Thus we have to normalize the calculated ray integrals in the projection image for this location-dependent sample spacing. To save these calculations during reconstruction, we instead normalize the images obtained from the scanner by the inverse amount in a preprocessing step. Further, since the hardware can only produce values in the range of $[0.0 \cdots 1.0]$, the acquired images are also scaled to the range $[0 \cdots n]$.

B. Correction Image Computation

After a projection image has been generated, the correction image is computed in software. (A hardware acceleration is not possible, due to the extended value ranges of the images involved.) First, the calculated projection image is subtracted from the acquired projection image. Then the resulting image is divided by the weight image at that orientation. A weight image holds the sum of weights in the denominator of the nominator in (2) for each pixel. Ideally, this sum of weights is equivalent to the intersection distance of a ray with a solid sphere, which

could be calculated analytically. However, it is not exactly the same, due to the discretized ray integration and the interpolation round-off errors of the hardware. For this reason, we compute the weight images in the same fashion than the projections, by projecting a volume in which all voxels within the spherical reconstruction region have been set to 1.0 and all others have been set to zero. These weight images can be re-used for all reconstructions that have equivalent image acquisition spacings and cone angles. We simply load them from disk prior to reconstruction.

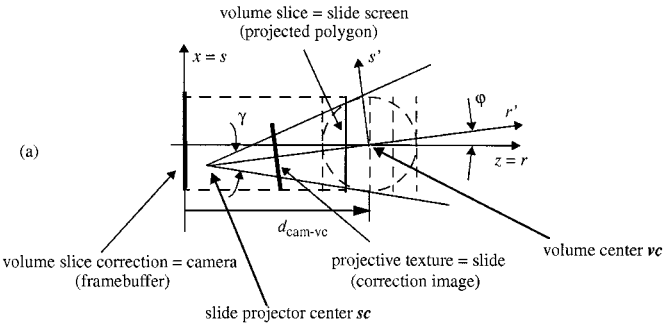
Finally, since the texture map can only hold values in the range $[0.0 \cdots 1.0]$, but the correction image may have values in the range $[-1.0 \cdots 1.0]$, we must scale and translate the values in the correction image to the $[0.0 \cdots 1.0]$ interval. Note that, in this way, the values in the volume slices are always in the range $[0.0 \cdots 1.0]$.

C. Backprojection

In backprojection, we need to distribute the correction image onto the volume slices. This is achieved by associating each volume slice, one by one, with the screen, onto which the correction image, mapped to a polygon, is rendered. Basically, this is the reverse situation of Fig. 2, with the screen now being a volume slice and the texture-mapped polygon being the correction image. Fig. 3 shows this configuration. We notice, however, that now the main viewing axis is no longer perpendicular to the screen, and neither does it always traverse the center of the screen (i.e., the volume slices). Although OpenGL does allow the viewing axis to be at an oblique angle, it is difficult, if not impossible, to set up the correct projection in presence of the second condition. Hence, a simple reversal of the forward projection to perform the backprojection is not feasible.

We shall now describe an approach that separates the screen from the volume slices, thus avoiding the problems stemming from the misalignment of the viewing axis with the screen center. Our approach uses the projective textures described by Segal *et al.* [23] and works just like a slide projector. Consider Fig. 4(a) where the method is illustrated: In contrast to the direct approach, the correction image is now first perspectively projected onto a polygon, which has been placed at the location of the volume slice that is to receive the backprojected correction. The image projected onto the polygon is then viewed by the framebuffer in parallel (orthographic) projection. In other words, the correction image is a slide (a projective texture) that is projected with a cone-beam "light" source onto a slide screen (the volume slice polygon), and the projected slide is then photographed by a parallel-beam camera (the framebuffer). The image captured in that way is the volume slice correction, properly weighted by bilinear interpolation. Note, however, that we must first scale and translate the values of the screen image back into the $[-1.0 \cdots 1.0]$ range before we can add it to the volume slice in memory. The resulting voxel values are then clamped to an interval of $[0.0 \cdots 1.0]$ and voxels outside the spherical reconstruction region are set to zero.

Let us now explain this slide-projector approach in some more detail, assuming a hardware implementation of OpenGL. In that case, when a polygon is projected onto the screen,



(b)

T_1: translate by $d_{-(\text{cam-vc})}$ **R**: rotate by φ
P: perspective mapping **S**: scale by 0.5 **T_2**: translate by 0.5

TMA-SART Backprojection algorithm

Set texture matrix to $\mathbf{TM}=\mathbf{T_1}\cdot\mathbf{R}\cdot\mathbf{P}\cdot\mathbf{S}\cdot\mathbf{T_2}$
For each of the *n* volume slices
  Associate 3D-texture coordinates with each vertex,
    set *r*-coordinate to *z*-coordinate
  Render the texture-mapped polygon onto the framebuffer
    (use **TM** to map the texture coordinates onto the projective texture)
  Read the framebuffer
  Scale and translate the values in the read image back into the interval [-1.0...1.0]
  Add the resulting image to the respective volume slice
  Clamp the summed values to the interval [0.0...1.0]

Fig. 4. Volume backprojection with TMA-SART. (a) Projection geometry (2-D case shown). (b) Algorithm pseudocode.

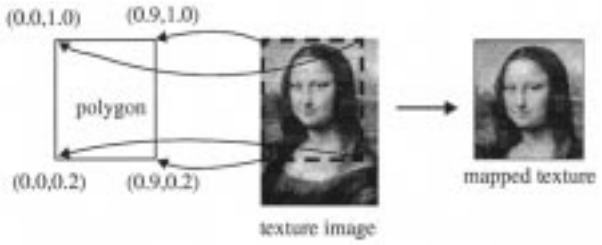


Fig. 5. Texture mapping an image onto a polygon. The texture coordinates (s, t) assigned to the polygon vertices are given in parentheses.

the coordinates of its vertices are transformed by a sequence of hardware matrix operations, which may include a perspective transform and a perspective divide. (For more detail on these fundamental issues refer to [6] and [21].) A texture is an image indexed by 2–D coordinates in the range $[0.0 \cdots 1.0, 0.0 \cdots 1.0]$. When a texture is mapped onto a polygon, the polygon's vertices are associated with texture coordinates (s, t), as shown in Fig. 5. The viewing transformation (i.e., the world-to-screen mapping) of the polygon vertices yields a closed region on the screen. In a process called *scan conversion*, all pixels inside this region are assigned (via interpolation) texture coordinates within the range assigned to the bounding vertices. These texture coordinates are used to index the texture image. Note that the transformation can lead to a stretching or shrinking of the texture on the screen.

The texture mapping coordinates need not be 2-D. As a matter of fact, they can be up to 4-D (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 slide projector.

The algorithm proceeds as follows (see again Fig. 4). First, we create an array of n square texture coordinate polygons with vertex coordinates (s, t, r) and associate them with the volume slice polygons. We set the four (s, t) coordinates to $(n/2, n/2)$, $(-n/2, n/2)$, $(-n/2, -n/2)$, and $(n/2, -n/2)$, with n being the extent of the cubic volume. The r-coordinate is varied between $[d_{\mathrm{cam-vc}} - n/2, d_{\mathrm{cam-vc}} + n/2]$ ($d_{\mathrm{cam-vc}}$ is the distance of the camera to the *v*olume *c*enter), depending on the location of the volume slice polygon in camera space. Note, that the (s, t, r) texture coordinate system is aligned with the (x, y, z) spatial coordinate system of the camera, where it has its origin. When a volume slice polygon is parallel (orthographically) rendered to the screen (the framebuffer), the hardware generates the corresponding texture map coordinates. These are first transformed by the texture transformation matrix $\mathbf{TM}$ before indexing the texture image. To achieve the slide-projector effect, $\mathbf{TM}$ is built as a concatenation of a number of matrices [see Fig. 4(b)]. The first sequence of terms, $\mathbf{T_1} \cdot \mathbf{R}$, achieves the coordinate transform from the (s, t, r) volume coordinate system into the (s', t', r') slide-projector coordinate system, which has its origin at the volume center. The term $\mathbf{T_1} \cdot \mathbf{R}$ rotates each texture coordinate polygon about the volume center $\boldsymbol{vc}$ by the viewing angle φ. Next is the perspective mapping, achieved by the perspective matrix $\mathbf{P}$. Here, the distance between slide projector center and volume center is

$$\boldsymbol{sc} - \boldsymbol{vc} = \frac{n/2}{\sin(\gamma/2)} \tag{3}$$

and $\mathbf{P}$ is defined such that, after the perspective divide, the texture coordinates of the polygon portions that fall inside the slide projector cone assume values in the range $[-1.0 \cdots 1.0]$. This is the case in the configuration shown in Fig. 4(a). However, since we can only index the texture image within a range of $[0.0 \cdots 1.0]$, we need to scale and translate the perspective texture coordinates prior to the perspective divide and texture indexing. This is achieved by incorporating a scale and translation given by $\mathbf{S} \cdot \mathbf{T}_2$ into $\mathbf{TM}$. Thus, any texture coordinate (s, t, r) generated by viewing the volume slice polygon is transformed by $\mathbf{TM}$

$$(s', t', r') = (s, t, r) \cdot \mathbf{TM}. \tag{4}$$

The perspective divide then produces the texture index (s_t, t_t):

$$s_t = \frac{s'}{r'} \qquad t_t = \frac{t'}{r'}. \tag{5}$$

Note 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.

IV. Extensions

In this section, we shall describe some extensions to the basic TMA-SART algorithm, for the benefit of both efficiency and accuracy.

A. Projection Image Accumulation in Hardware

Let us first assume that the framebuffer has 12 bits, as is the case for the SGI Octane. We noted before that the accumulation of the projection image takes place in main memory. This is necessary since the 12 bit framebuffer does not have any extra bits beyond the resolution of the projected image, which has at least 12 bits. Besides the fact that now the CPU must be used to add the n projection images, there are also n rather expensive framebuffer reads. One way to perform accumulations in the framebuffer would be to sacrifice precision (i.e., the lower bits) for speed. This, however, would impede the accuracy of the projection images, which is clearly undesirable.

We will now discuss a scheme that we can use to virtually extend the resolution of the framebuffer. The framebuffer has three color channels, red, green, blue, and alpha. Usually, we are only reconstructing grey level data, so all we utilize is a single color channel, say red, both in texture memory and in the framebuffer. However, if we partition the 12-bit data word into two components, one 8-bit and one 4-bit, and render it into two separate color channels, red and green, then we can accumulate data into the remaining upper 4 bits of the two framebuffer channels. This is illustrated in Fig. 6.

The four extra bits allow us to accumulate up to 16 images, which decreases the number of necessary framebuffer reads by 2/16 (we now have to read two color channels). Notice, however, that bit_{8-11} of the texture word are not interpolated by the texture mapping hardware in 12 bits, but only in 8 bits. This may cause inaccuracies. To illustrate this problem, imagine the following simple case. Assume a *texel* (a texture element) has a binary value of 1 0000 0000 (only bit_8 is set) and its immediate neighbors are all 0. Thus the red texture channel contains 0, and the green texture channel contains 1 0000. Now let us assume that the texture mapping interpolation of this texel neighborhood yields a binary value of 1000. In the original approach, the framebuffer would contain that value, in the second approach (Fig. 6), however, the framebuffer would contain zero.

B. Extending the Framebuffer Accuracy

The accuracy of TMA-SART is mainly determined by the resolution of the buffer elements in the graphics hardware, i.e., the texture memory and the framebuffer. The texture memory on an SGI Octane has a resolution of 16 bits, while the framebuffer has a resolution of 12 bits. As there is no need to keep the volume data at a higher resolution than the texture memory that projects them, TMA-SART stores the volume data as 16 bit unsigned shorts. On the other hand, the images—those obtained from the scanner, the computed projections, and the weight images—are all stored as 32 bit unsigned integers, since they represent accumulations of many (i.e., n) 16 bit voxel slices. The correction image is stored in 16 bit, since it is normalized to one volume slice prior to backprojection. Thus the precision of the reconstruction process is inherently 16 bit, hampered, however, by

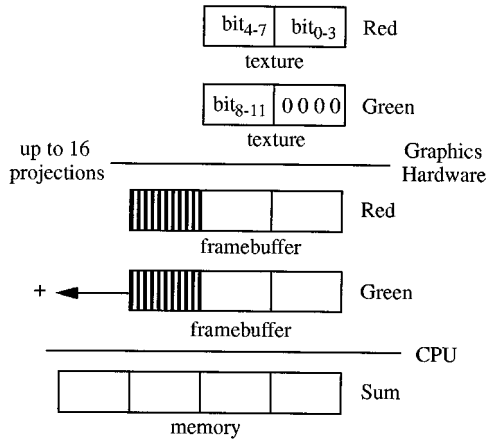


Fig. 6. Rendering a 12-bit data word using two color channels. The shaded upper four bits in the framebuffer can be used for accumulation. After the four upper bits have been filled by 16 projections, we must add the two channels in the CPU. For this purpose, the green channel must be shifted to the left by four bits.

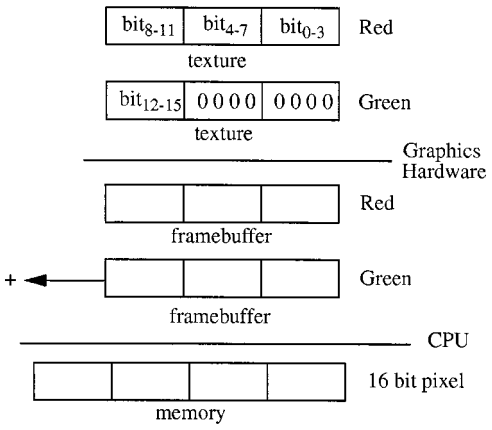


Fig. 7. Increasing the framebuffer resolution from 12 bit to 16 bit by adding up two color channels, properly shifted.

the limited 12 bit resolution of the framebuffer. Although reconstructions with moderate contrast levels should not be affected by this precision-bottleneck, it could cause low-contrast detail to be left unresolved in high-fidelity reconstructions. To alleviate these limitations, we shall now investigate a scheme that extends the framebuffer width in a virtual fashion, without hardware modification.

Consider Fig. 7 where these concepts are illustrated. The lower 12 bits of the volume data (or correction image data) are written to the Red texture channel, while the upper 4 bits of the data are written to the upper 4 bits of the Green texture channel. Rendering is performed as usual, and the Red and the Green framebuffer is read into two software buffers. The 16 bit result is constructed by adding the two software buffers, with the data in the Green buffer shifted to the left by 4 bits.

Note that, similar to the accumulation buffer, the (virtual) 16 bit data in the Green channel are not interpolated in 16 bits, but only in 12 bits. This will have effects similar to the ones outlined in the previous section. Hence, the presented implementation is not a true 16 bit extension to the framebuffer, it is only an approximation. However, it will still help produce considerably more precise results than the original 12 bit implementation.

If only an 8 bit framebuffer is available, along with 8 bit textures, then we use 12 bit volume data and render the lower 8 bit into the Red framebuffer channel and the upper 4 bit, shifted 4 bits to the left, into the Green framebuffer channel. Combining the channels is done as usual and a 12 bit data word results.

V. Results

All programs were written using the widely accepted OpenGL API (Application Programming Interface) and can easily be reproduced to run on any medium-range graphics workstation or PC with graphics board. Using both the software implementation of SART (discussed in [16] and [17]) and the new hardware-accelerated version, TMA-SART, we reconstructed a simulated brain dataset, the 3-D extension of the Shepp–Logan phantom [24] (described, e.g., in [4], a slice is shown in Fig. 9(a). Projection sets of $S = 80$ cone-beam projections ($\gamma = 40°$) of 128^2 pixels each were obtained by analytical integration of the phantom.[5] These projections were used to reconstruct a 128^3 reconstruction volume in three iterations ($\lambda = 0.1$). To evaluate the effect of the limited framebuffer resolution in TMA-SART, we acquired the brain projection sets at three different feature contrast levels for the 12-bit SGI Octane and at five different levels for the 8-bit SGI O2, a low-end Unix-PC equipped with graphics hardware.[6] 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 slices, shown in Fig. 9(a) is only 0.5%. (Note that, in all images of Fig. 9, the small dynamic range of the features was stretched into the full displayable range in order to make the features visible and to provide equivalent brightness for all contrast levels. The original range is $[0.0, 2.0]$.)

Fig. 9(b) shows a slice [from the same location than that in Fig. 9(a)] across a volume reconstructed with the software implementation of SART described in [17]. This implementation employs splatting with analytically preintegrated Bessel–Kaiser kernels [12]. Next to the reconstructed slice, we show the intensity profile along a line that cuts horizontally across the center of the three small tumor ellipsoids [see Fig. 9(a)]. We observe that very little reconstruction artifacts are present and that the brain features (e.g., the small tumors) can be well discerned.

The software implementation uses both floating point arithmetic and floating point buffers throughout the reconstruction process. TMA-SART, on the other hand, also uses floating point arithmetic but only fixed point buffers. This amounts to some inaccuracies in the reconstruction process, which is demonstrated next. In Fig. 9(c), we show three reconstructions (slices and profile plots) obtained with the basic 12-bit framebuffer TMA-SART, while in Fig. 9(d) we show three reconstructions (slices and profile plots) obtained with the enhanced, 16-bit frame-

[5]The phantom was centered on the rotation axis, no noise was added to the projections.

[6]For the purposes relevant to this paper, the O2 graphics hardwave is equivalent to the various boards available for PC's. However, computation time is likely to vary and tends to change fast in today's rapidly evolving market.

buffer TMA-SART. The contrast of the imaged phantom doubles in every column from left to right. We observe that the basic TMA-SART produces reconstructions with consistently higher levels of noise-like artifacts than the 16-bit TMA-SART. The difference is particularly striking in the first column for the original contrast. Here, the three small tumors in the lower third of the slice are only discernible in the 16-bit TMA-ART reconstruction. However, the higher the contrast, the lesser the impact of the framebuffer precision. The reconstructions obtained at twice the contrast of the Shepp–Logan phantom are already of acceptable quality. In Fig. 9(c) and (d), we also observe a number of artifacts that do not seem to be dependent on framebuffer resolution. These artifacts are more of a structured nature, as, for example, the dark ripple below the three small tumors. Other similar artifacts can be seen all over the reconstructions and do not decrease with framebuffer resolution.

Fig. 9(e) shows reconstructions obtained with an 8-bit framebuffer that uses the 12-bit enhancement described in Section IV-B. We notice that the reconstructions at twice the original contrast are not of the same quality than those of the true 12-bit framebuffer. This is due to the incomplete 12-bit precision and the use of an 8-bit texture map instead of the 12 bits available on the Octane. However, the reconstruction at four times the original contrast, although still somewhat noisy, already distinguishes the three small tumors rather well. The images at 8 and 16 times the original contrast resemble those for the enhanced 12-bit framebuffer at two and four times the original contrast, respectively, and are of acceptable quality.

Finally, Table I compares the run times for both the software and the various TMA-SART implementations. We see that by utilizing texture mapping hardware for the volume 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 2 min, down from the 1.8 hrs that were required in the software implementation on the same host CPU. This represents a speed-up of over 50.

By using the accumulation buffer enhancement, outlined in Section IV-A, we can reduce the reconstruction time even more to 1.6 min (a speedup of 68 with respect to software-ART). A reconstruction using the increased precision framebuffer, (outlined in Section IV-B, but not the accumulation buffer, takes somewhat longer (3.1 min for a speedup of 49), due to the increased number of framebuffer reads and CPU computations. The time required for reconstruction with our O2, using the 12-bit framebuffer enhancement, was 15.8 min, which amounts to a 6.8 speedup with respect to the software implementation on the Octane. However, the speedup is about eight when compared to the slower runtime of software SART on the O2 itself.

The runtime complexity of the SART algorithm is O($S \cdot N$), which indicates that the runtimes given in Table I should scale linearly with increasing S and N. This was verified in experiments conducted on volumes up to 512^3. The SGI Octane texture memory will currently fit 1 MB of texels, so volumes with up to 512^3 voxels (and slices with 512^2 16-bit texels) can be processed as is. For larger volumes, the slices need to be broken up into tiles, with the projection results being merged later, which adds some overhead. The O2 and many other modern graphics boards have unified memory architectures in which the texture memory is synonymous with main memory, so the texture capacity is far less constrained. Since we store the volume as voxel runs in the volume coordinate that is aligned with the rotation axis, extra cache faults during volume reads and write-backs will rarely occur [19]. Although software-ART has a somewhat more irregular data access [17], which affects its cache behavior, experiments have shown that linear relationships in terms of S and N exist here as well. Thus the speed-up ratios given in Table I remain approximately the same, even for larger S and N.

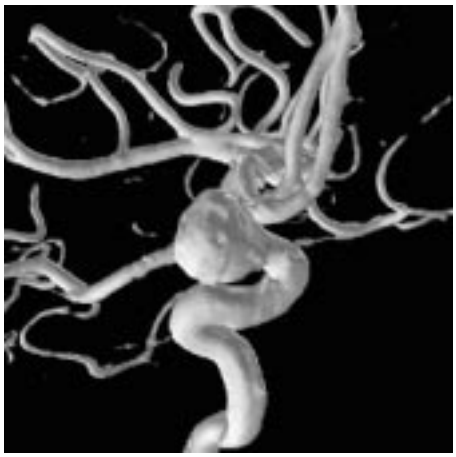


Fig. 8. Reconstructed and volume rendered blood vessel in a human brain. Projection data were obtained by 3-D rotational angiography on a Siemens C-arm scanner.

TABLE I
RUN TIMES FOR ONE ITERATION AS WELL AS FOR A COMPLETE RECONSTRUCTION (THREE ITERATIONS) OF DIFFERENT SART IMPLEMENTATIONS. [THESE TIMINGS WERE NEEDED TO PRODUCE THE IMAGES SHOWN IN FIG. 9(b)–(e).] THE OCTANE HAD A MIPS R10000/195-MHz CPU WITH 640 MB OF RAM. THE O2 HAD A MIPS R10000/175-MHz CPU AND 128 MB OF RAM. BOTH HAD 32-KB PRIMARY DATA CACHE AND 1-MB SECONDARY CACHE

SART implementation	framebuffer resolution	time / iteration	reconstruction time	speedup
software	floating point	36.0 min	1.8 h	-
TMA: basic	12 bits	42.2 sec	2.1 min	52
TMA: accumulation buffer	12 bits	33.2 sec	1.6 min	68
TMA: 16 bit enhanced framebuffer	12 bits	62.0 sec	3.1 min	35
TMA: 12 bit enhanced framebuffer	8 bits	316.0 sec	15.8 min	7- 8

VI. DISCUSSION AND CONCLUSIONS

In this paper, we have investigated the utility and applicability of widely available graphics hardware with texture mapping capabilities for the purpose of 3-D reconstruction with algebraic methods. Algebraic methods have a number of advantages in certain reconstruction scenarios, but are presently far too slow for routine clinical use. The advantage of employing graphics hardware for reconstruction acceleration is that this equipment is likely to exist in clinical imaging labs anyhow, for image-aided diagnosis and medical procedure planning, and therefore no or little capital effort has to be made to apply the proposed technology.

We first determined that of all the available algebraic methods, SART was the most appropriate one to use. We then

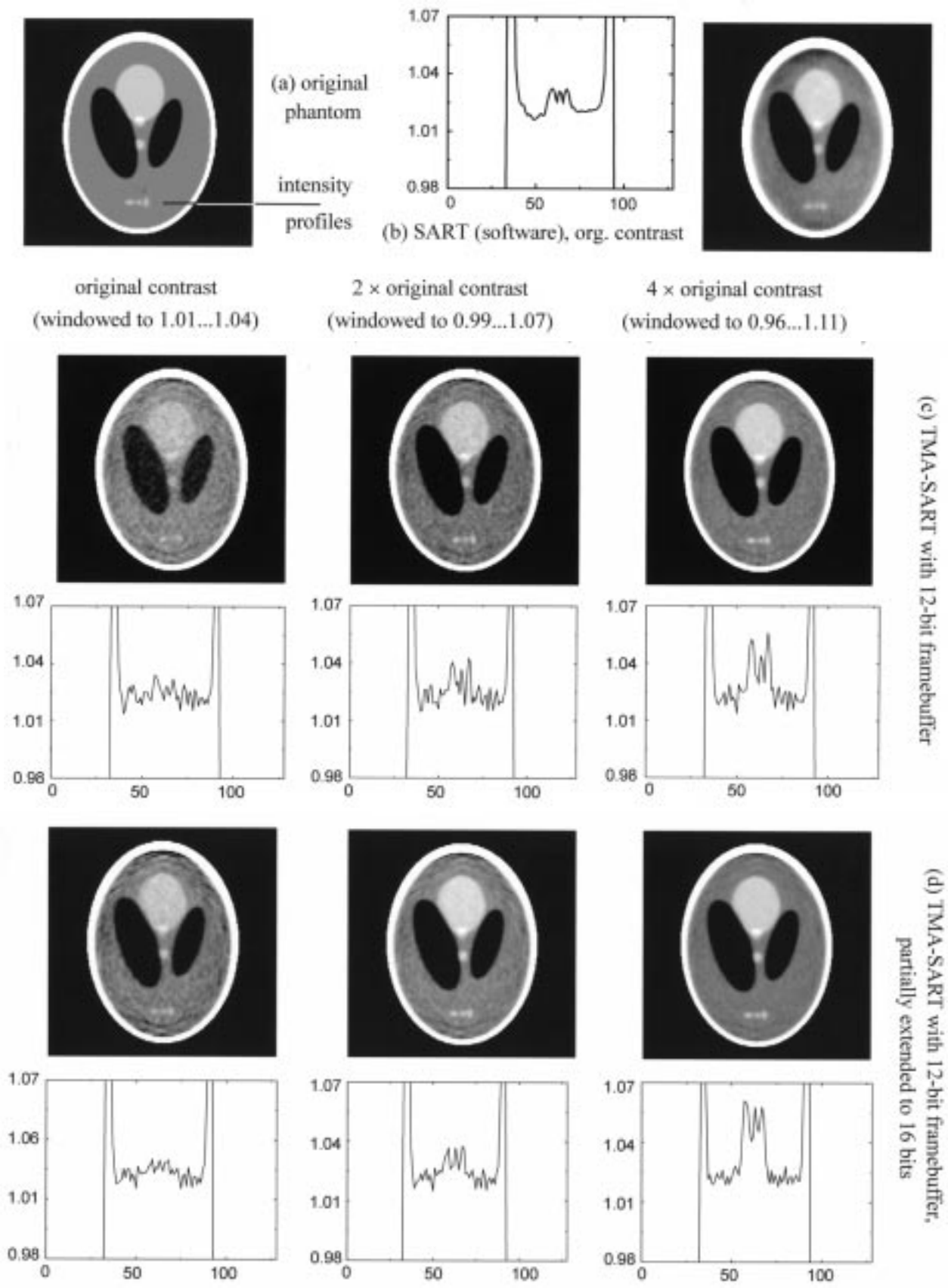


Fig. 9. Slices across reconstruction volumes obtained with different implementations of SART, software and hardware-accelerated. All reconstructions were performed using $S = 80$ 128×128 projections of the 3-D extension of the Shepp–Logan phantom (cone-angle $\gamma = 40°$, three iterations, a 128^3 reconstruction grid, $\lambda = 0.1$). The plots show the intensity profiles across the center of the three small ellipsoids (tumors) near the bottom of the phantom.

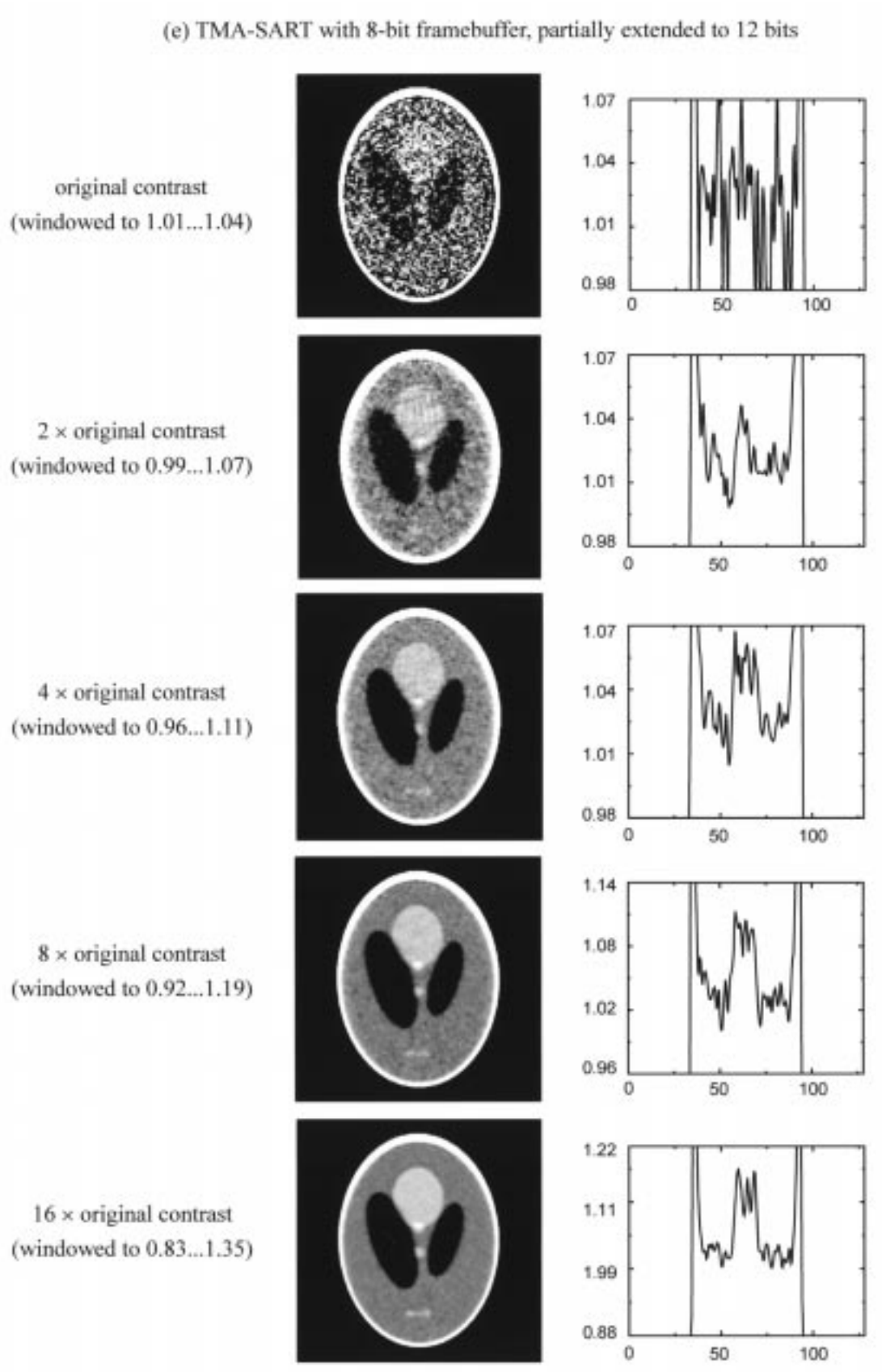


Fig. 9. (*Continued*) Slices across reconstruction volumes obtained with different implementations of SART, software and hardware-accelerated. All reconstructions were performed using $S = 80$ 128×128 projections of the 3-D extension of the Shepp–Logan phantom (cone-angle $\gamma = 40°$, three iterations, a 128^3 reconstruction grid, $\lambda = 0.1$). The plots show the intensity profiles across the center of the three small ellipsoids (tumors) near the bottom of the phantom.

partitioned the available graphics architectures into two groups: 1) mid-range graphics workstations (below $25 000), such as the SGI Octane, that have a 12-bit framebuffer, and 2) low-end graphics PCs and graphics boards that currently only have an 8-bit framebuffer. Our experiments indicate that the first group offers speedups between 35 and 68 when compared to a fairly optimized software implementation of SART, running on the same host CPU. The second group was investigated primarily to determine whether an 8-bit framebuffer can provide meaningful reconstructions at all. The timings that were obtained for that group are bound to be outdated very quickly, given the rapid performance growth of today's graphics boards, such as NVidia's GeForce series. We obtained a speedup of about eight on an SGI O2, an entry-level graphics PC with hardware comparable to a last-generation PC graphics board.

Our experiments indicate that the limiting factor in this endeavor is the limited resolution of the framebuffer. To ease this drawback we devised a scheme that partially extends a given framebuffer resolution by 4 bits. This allowed the resolvable contrast to be lowered to half the level that could be resolved without the extension. Using this extension for a 12-bit framebuffer, we found that object features of 1% contrast can already be distinguished well. We also found that the noise levels due to the limited framebuffer resolution are in the 0.5%–1% contrast range. This means that once the features exceed this range, the signal-to-(reconstruction) noise ratio becomes sufficient for a reconstruction of good quality. For an 8 bit framebuffer, partially extended to 12 bit, the noise level is slightly below 2%. Thus features of 2% contrast can be distinguished, but the noise is still well perceivable. If only features of 4% contrast are present then the noise can de-emphasized sufficiently by widening the brightness window.

The software implementation of SART [17] uses high-quality Bessel–Kaiser interpolation kernels that were analytically preintegrated. This provides near-analytical line integration and very accurate weight factors. TMA-SART, on the other hand, must use bilinear interpolation within the volume slices and the trapezoidal rule for integration. Furthermore, the sampling interval is greater than unity for the majority of rays, i.e., all those that are nonperpendicular to the volume slices most parallel to the image plane. The lower quality filter and integrator approximation can cause aliasing artifacts that are independent of framebuffer resolution, and we have observed them as the minor structured artifacts in Fig. 9(c) and (d). One possible way to increase the sampling rate is to interpolate additional, intermittent volume slices prior to a projection step. This interpolation can be done in hardware by blending two adjacent volume slices according to their distance relative to the new slice. Projection would then be based on all volume slices—the original ones and the interpolated ones. Backprojection would occur in the reverse fashion: One would backproject onto all slices, and the contributions of the intermittent slices would be distributed onto the original slices by a separate blending step in hardware.

Although TMA-SART, with present hardware, does not deliver reconstructions that can resolve the low contrasts (0.5%) of a software implementation, it can resolve 1% with a 12-bit framebuffer and 2%–4% with an 8-bit framebuffer, which is not too far from the desired levels. There are in fact a number of CT applications that do not require contrast levels beyond the capabilities of TMA-SART. One example is 3-D reconstruction angiography [1] (see Fig. 8), where the task is to reconstruct an opacified blood vessel from its projections. Another application is the reconstruction of bone structures. Many industrial CT applications may also benefit from TMA-SART. Finally, algebraic methods have also been quite successful in PET and SPECT reconstruction, where contrasts are typically rather high [13].

But nevertheless, an improvement of the framebuffer resolution of inexpensive graphics boards would be ideal. One incentive for board developers to provide wider framebuffers is the circumstance that they enable fast, high-quality motion-blur, a highly desirable effect in graphics and game applications. To achieve motion-blur, a moving object is rendered into the framebuffer a number of times, once for each time step. This calls for an extended framebuffer to allow for the frame accumulations without loss of precision. After a certain number of frames are rendered, the framebuffer is averaged by bit-shifting it to the right, and the motion-blurred scene or object is displayed. These extended framebuffers, called accumulation buffers [9], can also be used to generate soft shadows, depth-of-field effects, as well as high-quality anti-aliasing of polygonal objects [9]. On the other hand, there are also a number of new graphics and volume rendering boards currently being prepared for market introduction (e.g., the VIZARD volume rendering board [15]). One may be able to work with the designers to add mechanisms that would allow a framebuffer word to be changed, via software switch, from three 8-bit RGB slots to a single 24-bit Luminance slot. The added arithmetic would only require a small number of extra logical gates and would enable the TMA-SART accumulator as well.

REFERENCES

[1] "3D reconstruction angiography at General Electric,", Available: http://www.ge.com/medical/xray/msxclnv1.htm.

[2] A. H. Andersen and A. C. Kak, "Simultaneous algebraic reconstruction technique (SART)," *Ultrason. Img.*, vol. 6, pp. 81–94, 1984.

[3] A. H. Andersen, "Algebraic reconstruction in CT from limited views," *IEEE Trans. Med. Imag.*, vol. 8, pp. 50–55, Feb. 1989.

[4] C. Axelson, "Direct Fourier methods in 3-D reconstruction from cone-beam data," master's thesis, Linkoping Univ., Linkoping, Sweden, 1994.

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

[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] P. Gilbert, "Iterative methods for the three-dimensional reconstruction of an object from projections," *J. Theo. Biol.*, vol. 36, pp. 105–117, 1972.

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

[9] P. Haeberli and K. Akeley, "The accumulation buffer: hardware support for high-quality rendering," *Computer Graphics (Proceedings of SIGGRAPH'90)*, vol. 24, no. 4, pp. 309–318, 1990.

[10] P. M. Joseph, "An improved algorithm for reprojecting rays through pixel images," *IEEE Trans. Med. Imag.*, vol. 1, June 1982.

[11] A. C. Kak and M. Slaney, "Principles of computerized tomographic imaging," . Piscataway, NJ: IEEE Press, 1988.

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

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

[14] T. R. Mackie, T. Holmes, S. Swerloff, P. Reckwerdt, J. O. Deasy, J. Yang, B. Paliwal, and T. Kinsella, "Tomotherapy: a new concept for the delivery of dynamic conformal radiotherapy," *Med. Phys.*, vol. 20, no. 6, pp. 1709–1719, 1993.

[15] M. Meißner, U. Kanus, and W. Straßer, "VIZARD II: a PCI-card for real-time volume rendering," in *Proc. 1998 Eurographics Workshop Graphics Hardware*, pp. 61–67.

[16] K. Mueller, R. Yagel, and J. J. Wheller, "Anti-aliased 3-D cone-beam reconstruction of low-contrast objects with algebraic methods," *IEEE Trans. Med. Imag.*, vol. 18, pp. 519–537, June 1999.

[17] ——, "Fast implementations of algebraic methods for the 3-D reconstruction from cone-beam data," *IEEE Trans. Med. Imag.*, vol. 18, pp. 538–547, June 1999.

[18] K. Mueller and R. Yagel, "On the use of graphics hardware to accelerate algebraic reconstruction methods," in *Proc. 1999 SPIE Medical Imaging Conf.*, Feb. 1999, Paper 3659–62.

[19] ——, Rapid 3-D cone-beam reconstruction with ART utilizing texture mapping graphics hardware. presented at IEEE Medical Imaging Conf. 1998

[20] K. Mueller, J. Chang, H. Amols, and C. C. Ling, "Cone-beam computed tomography (CT) for a megavoltage linear accelerator (LINAC) using an electronic portal imaging device (EPID) and the algebraic reconstruction technique (ART)," in World Congress on Medical Physics and Biomedical Engineering, July 23–28, 2000, to be published.

[21] J. Neider, T. Davis, and M. Woo, *OpenGL Programming Guide*. Reading, MA: Addison-Wesley, 1993.

[22] 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. Amer. A*, vol. 8, no. 10, pp. 1639–1648, 1991.

[23] 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.

[24] 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.

[25] B. Smith, "Cone-beam tomography: recent advances and a tutorial review," *Opt. Eng.*, vol. 29, no. 5, pp. 524–534, 1990.

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