Presented at the 1998 IEEE Medical Imaging Conference

Rapid 3D Cone-Beam Reconstruction with the Algebraic Reconstruction Technique (ART) by Utilizing Texture Mapping Graphics Hardware

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. In this paper, we devise a new hardware acceleration scheme, employing readily available texture mapping graphics hardware, that allows quality 3D cone-beam reconstructions to be obtained at almost interactive speeds.

I. INTRODUCTION

The Algebraic Reconstruction Technique (ART), first proposed by Gordon et. al. [6], is a tomographic reconstruction method which reconstructs a 3D object from its projection images, 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, 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 [10]. So far, cone-beam CT is still in the 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. These algorithms are mostly based on FBP [4][7][18] (see also [14][19] for comparisons and reviews). In more recent research [11], it was 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˚.

Thus ART possesses great prospects for 3D reconstruction from cone-beam data if its computational speed could be improved. It was already shown in [11] that two to three iterations are sufficient to reconstruct a 3D object. In addition, the required number of projections in ART is typically smaller than for FPB, at least in the theoretical sense [8]. However, still more than 2.5 hours are needed on a modern workstation to reconstruct a 128^3 volume from 80 projections [12]. As a remedy, one could build dedicated ART computer 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, as the graphics hardware resident in these workstations is especially designed for fast projection operations, 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 offer texture-mapping boards that can be added to any modern PC.

In the following sections, we will describe a *T*exture-*M*apping hardware *A*ccelerated version of ART (TMA-ART) that reconstructs a 128^3 volume from 80 projections in about 2 minutes — a speedup of over 70 with respect to the software solution outlined in [8]. The 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. Although we will describe our algorithm in terms of cone-beam reconstruction, it naturally also applies to parallel-beam reconstruction as a private case.

[1] 2015 Neil Ave, 395 Dreese Lab, Columbus, OH 43210, USA
email: {mueller, yagel}@cis.ohio-state.edu.
www: http://www.cis.ohio-state.edu/~{mueller, yagel}

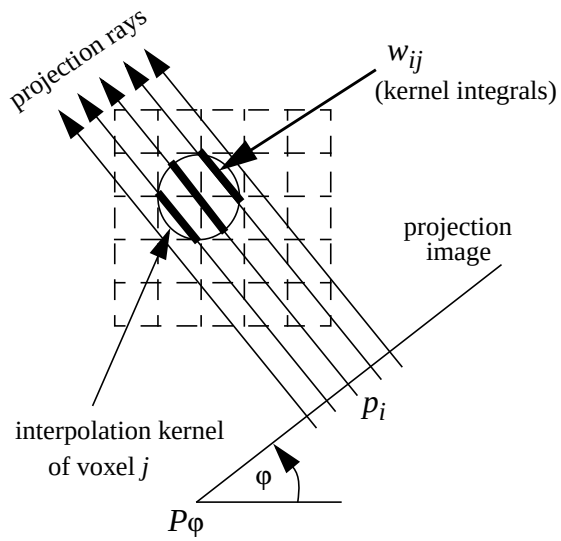


Fig. 1. The interpretation of a voxel weight factor w_{ij}.

II. 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. This decision is justified by the observation that SART produces cone-beam reconstructions of a quality equal to those obtained with cone-beam ART, but without requiring the depth-adaptive interpolation filters that were shown to be necessary for good reconstruction quality with cone-beam ART [11].

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 (implemented as backprojections). Consider Fig. 1, that shows how a grid voxel (i.e., a volume element) contributes to a projection and 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. The grid correction factors are computed by subtracting the ray integrals so obtained from the pixel values p_i in the scanner image $P\varphi$. During backprojection, the rays traverse the volume again, but this time the ray correction factors are weighted by the voxel kernel integrals and added to the present voxel values. A normalization is performed among all ray contributions that a particular voxel receives in this process.

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

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

Here, λ is a relaxation factor, typically chosen «1.0. Fig. 2 illustrates the algorithm, decomposed in its constituents:

SART algorithm

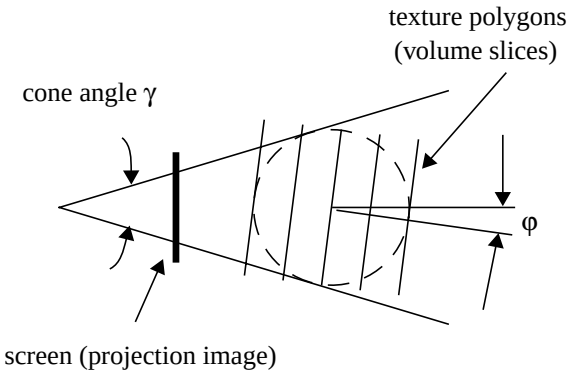
Initialize volume
Until convergence
  Select a projection $P\varphi$
  Image projection:
    Compute line integrals through all p_i of $P\varphi$
    (the inner sums in equation (1))
  Correction image computation:
    For all p_i, subtract line integrals from scanner p_i
    (expression in parentheses in nominator of (1))
  Image backprojection:
    Distribute corrections onto grid voxels

Fig. 2. The steps of the SART algorithm.

The software implementations typically perform grid projection and backprojection by a procedure termed *splatting*, in which the kernel integrals are pre-integrated into tables (so called *footprints*) and indexed by the traversing rays. Alternatively, in an object-order approach the footprints are mapped to the image plane where they accumulate into the projection image or retrieve the correction value, respectively (see e.g. [12]). We are now ready to describe the hardware implementation of SART.

III. HARDWARE ACCELERATION OF GRID PROJECTION / BACKPROJECTION

TMA-ART decomposes the volume into slices and treats each slice separately. In grid projection (shown in Fig. 3), each slice is associated with a square polygon with the volumetric slice content texture-mapped onto it. A projection image is



TMA-ART Projection algorithm

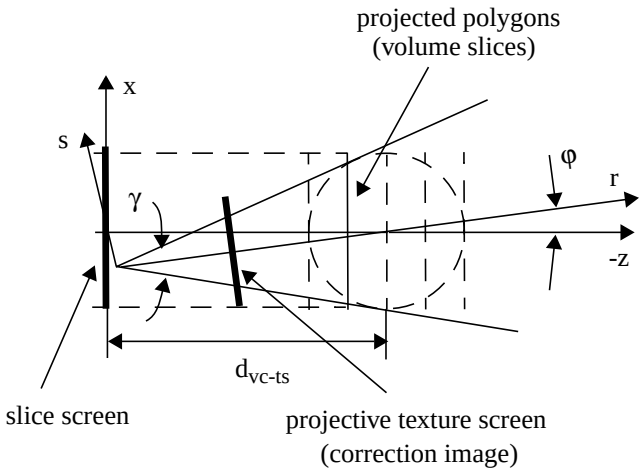
Rotate texture polygons by projection angle φ

Texture map the volume slices onto texture polygons

Project textured polygons onto the screen

Accumulate screen contributions at each pixel

Fig. 3. Grid projection with TMA-ART.



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

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. 4. Grid backprojection with TMA-ART.

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 [9]. 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.

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.) Since the texture map 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]. We may add that, at least for now, all computations with regards to the correction image are performed on the CPU.

In backprojection (shown in Fig. 3), 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. 3.) 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. [15]) 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. 3 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. 5. (For more detail on these fundamental issues refer to [5] and [13].)

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. 6. 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 (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 $[d_{vc\text{-}ts}-n/2,\ d_{vc\text{-}ts}+n/2]$. ($d_{vc\text{-}ts}$ is the distance of the source to the volume center.) Refer now back to Fig. 5, 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

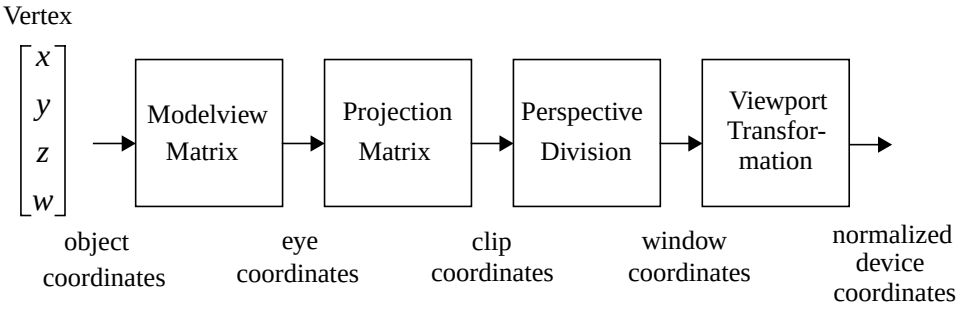


Fig. 5. Stages of vertex transformation

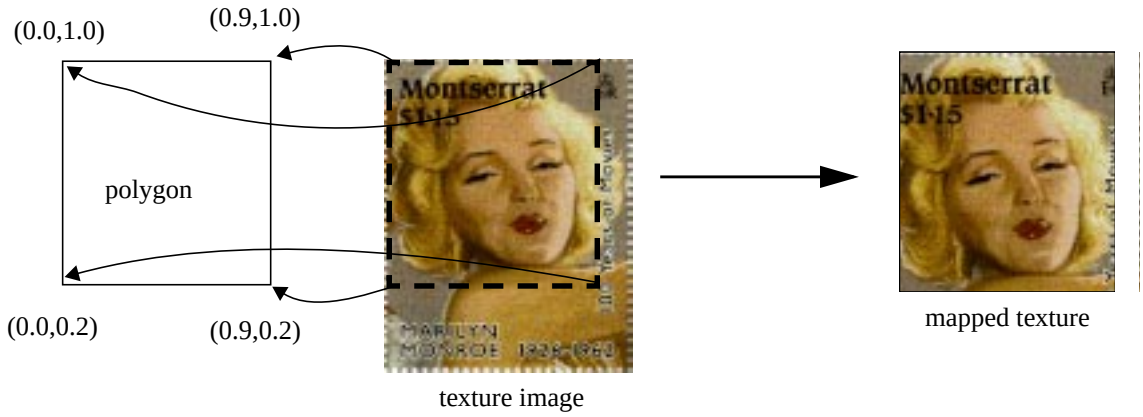


Fig. 6. Texture mapping of an image onto a polygon. The texture coordinates assigned to the polygon vertices are given in parentheses.

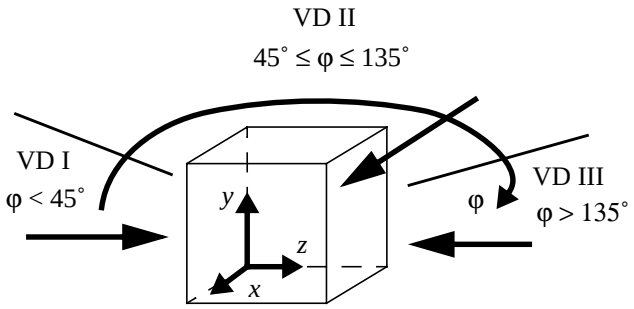


Fig. 7. Memory access order for different projection angles φ. There are three major viewing directions (VDs). For VD I and VD III the volume data are accessed one *z*-slice at a time, while for VD II, the volume data are accessed one *y*-slice at a time.

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. OPTIMAL MEMORY ACCESS

The volume data is stored as 16 bit unsigned shorts, while the scanner, projection, and weight image data are stored as 32 bit unsigned integers, since they represent accumulations of many 16 bit voxel slices. The correction image is stored in 16 bit, since it is normalized to one volume slice prior to back-projection. Thus the precision of the reconstruction process is inherently 16 bit, hampered, however, by a limited 12 bit resolution of the framebuffer. This precision bottleneck is to be kept in mind when attempting to use the graphics hardware for high-fidelity reconstruction.

Consider now Fig. 7 where we illustrate the three different viewing directions (VDs) at which the memory is accessed during the reconstruction. Usually, computers organize their memory in a hierarchy of several levels: a small primary on-chip cache, a larger secondary on-chip cache, main memory, and some swap partition on disk. Whenever a datum requested from the CPU is not found in one level, a cache fault or a memory fault is generated and the datum is fetched from the next lower memory level, and so on. When the datum has reached the CPU, it is also stored in all memory levels that were traversed on the quest. Usually, the data are fetched in blocks, following the law of locality-of-reference which states that once a datum is requested it is likely that the neighboring datum will be needed soon. Since memory access time is generally much larger than transfer time, it is usually more efficient to transfer a whole block once the memory access is performed, than to access every consecutive datum anew. Therefore, once a block of data is loaded into cache, no more faults are generated for any data in the block. Fetching data from the caches is generally much faster than retrieving the data from main memory. In case of a secondary-cache fault, the CPU will in many cases switch to another job until the data block is in, while at a primary cache-fault the CPU will just introduce a few wait states. Hence, we want to keep the number of secondary cache faults as low as possible. (For more details on memory hierarchies please refer to [17])

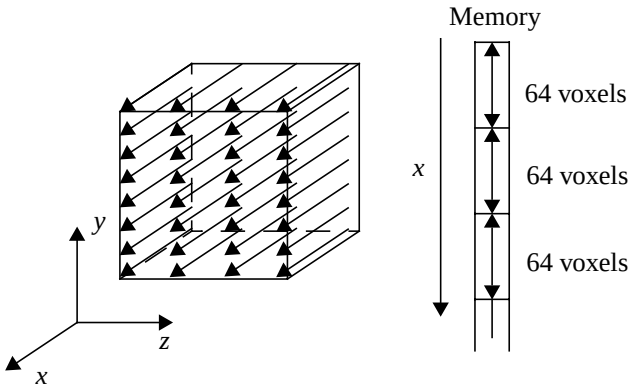


Fig. 8. Voxels are stored in *x*-*y*-*z* order. The arrows indicate consecutive data items in memory. No extra cache faults are generated for VD I and VD III. For VD II, however, a cache fault is generated for every voxel.

Our SGI Octane workstation with a Mips R10000 CPU has a secondary cache of 1MByte, with 128-byte slots and 2-way set-associativity. The secondary cache has a peak transfer rate of 3.2GByte/s to the CPU, which is magnitudes higher than the transfer rate from main memory. Since each volume voxel occupies 2 bytes, we can store 64 voxels in each cache slot.

We need to be intelligent about how we store the data in memory. The minimum number of cache faults is $N/64$ (where N is the number of voxels). Consider Fig. 8 where we show the case in which the data are stored in *x*-*y*-*z* order, i.e., *x* runs fastest. Recall that TMA-ART accesses and stores the volume data in slices. If we access the data from VD I or VD III in *x*-*y* order for each slice, we will encounter a cache fault every 64 voxels, the minimum cache-fault rate. However, if we access the data from VD II in *z*-*y* order (or *y*-*z* order) for each subsequent *x*-slice, we will have a cache fault for every voxel. This is 64 times more cache faults than for VD I and VD III, and will slow the reconstruction down considerably. Note also that these cache faults may occur not only for the loading of the volume slices, but also for the storing of the corrected volume slices, if a write-back cache-policy is used.

Consider now Fig. 8 where we show a different, more favorable memory organization in which the voxels are stored in *y*-*x*-*z* order. Let us look at VD I and VD III first. If we access the data in *y*-*x* order for every *z*-slice, then we generate a cache fault every 64 voxels, just like before. On the other hand, for VD II, if we access the data in *y*-*z* order for every *x*-slice, then we will not have any extra cache faults either. Thus the *y*-*x*-*z* storage arrangement in conjunction with *y*-first memory traversal will yield the optimal cache behavior.

The following example may serve as a demonstration for the importance of optimizing cache behavior. In our initial implementation, we used the most natural storage scheme, the *x*-*y*-*z* order, with which a reconstruction could be performed in 15

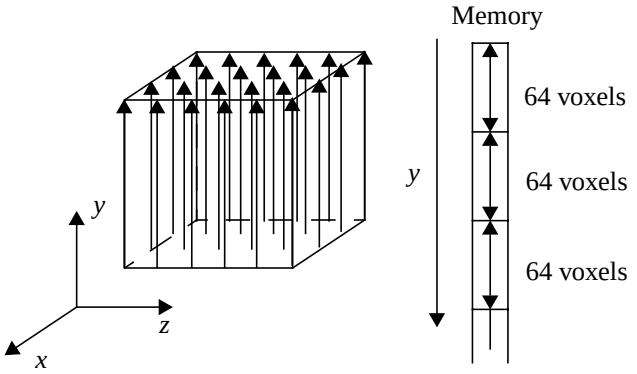


Fig. 9. Voxels are stored in *y*-*x*-*z* order. No extra cache faults are generated for VD I, VD II and VD III.

minutes. After observing the large number of cache faults (using SGI's *perfex* utility), we switched to the *y*-*x*-*z* storage order and the *y*-first data traversal. By using this access scheme, a reconstruction could be otained in 2 minutes.

V. RESULTS

Using both the software implementation of ART (discussed in [11][12]) and the new hardware-accelerated version, TMA-ART, we reconstructed a simulated brain dataset, the 3D extension of the Shepp-Logan phantom [16] (described e.g. in [2], a slice is shown in Fig. 10a). Projection sets of 80 cone-beam projections (γ=40˝) of 128^2 pixels each were obtained by analytical integration of the 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 slices, shown in Fig. 10, is only 0.5%. (Note that, in the images of Fig. 10, the small dynamic range of the features was stretched into the full displayable range in order to make the features visible.)

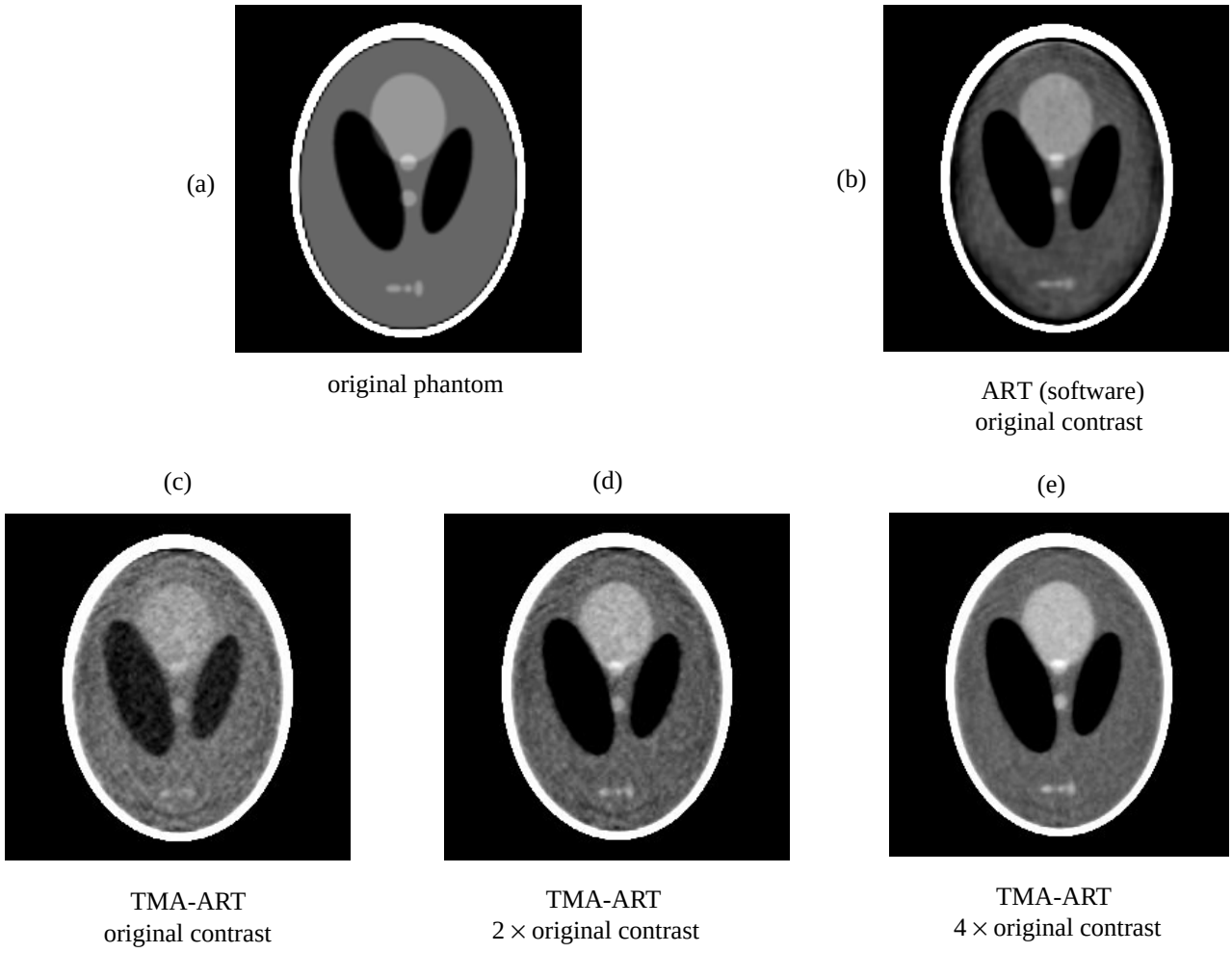


Fig. 10. Slices across reconstruction volumes obtained with different implementations of ART, software and hardware-accelerated. All reconstructions were performed using 80 128×128 projections of the 3D extension of the Shepp-Logan phantom (cone-angle γ=40˚, 3 iterations, 128^3 reconstruction grid, λ=0.1).

Fig. 10b shows a slice (from the same location than that in Fig. 10a) through 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 the limited resolution of the 12 bit framebuffer, which can only resolve 1/4096=0.02%. As Fig. 10c 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. 10d, 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. 10e, where the tumor contrast was 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 software and the two different TMA-ART implementations (i.e., x-y-z data access order and y-x-z data access order). We see 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 2 minutes, down from the 2.5 hours that were required in the software implementation. We also see that the effects of caching on the volume data retrieval are very significant. When accessing the volume data in x-y-z order a reconstruction still takes almost 15 minutes (a speedup of 10.5 with respect to the software implementation). However, when accessing the data in the optimal y-x-z order, the reconstruction time shrinks to a mere 2 minutes (a speedup of 73 with respect to software-ART and a speedup of 7 with respect to the sub-optimal x-y-z data access order).

implementation	time / iteration	reconstr. time	speedup
ART (software)	0.85 min	2.55 h	-
TMA-ART: x-y-z	289.0 sec	14.45 min	10.5
TMA-ART: y-x-z	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.

VI. 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.

VII. 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 multiprocessor PCs and workstations is currently being developed. It is expected that this version will achieve a reconstruction in less than 30 sec.

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

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

[6] 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.

[7] 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.

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

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

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

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

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

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

[14] 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.

[15] 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.

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

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

[18] 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.

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