Accelerating Popular Tomographic Reconstruction Algorithms on Commodity PC Graphics Hardware

Fang Xu and Klaus Mueller

Abstract—The task of reconstructing an object from its projections via tomographic methods is a time-consuming process due to the vast complexity of the data. For this reason, manufacturers of equipment for medical computed tomography (CT) rely mostly on special application specified integrated circuits (ASICs) to obtain the fast reconstruction times required in clinical settings. Although modern CPUs have gained sufficient power in recent years to be competitive for two-dimensional (2D) reconstruction, this is not the case for three-dimensional (3D) reconstructions, especially not when iterative algorithms must be applied. The recent evolution of commodity PC computer graphics boards (GPUs) has the potential to change this picture in a very dramatic way. In this paper we will show how the new floating point GPUs can be exploited to perform both analytical and iterative reconstruction from X-ray and functional imaging data. For this purpose, we decompose three popular three-dimensional (3D) reconstruction algorithms (Feldkamp filtered backprojection, the simultaneous algebraic reconstruction technique, and expectation maximization) into a common set of base modules, which all can be executed on the GPU and their output linked internally. Visualization of the reconstructed object is easily achieved since the object already resides in the graphics hardware, allowing one to run a visualization module at any time to view the reconstruction results. Our implementation allows speedups of over an order of magnitude with respect to CPU implementations, at comparable image quality.

Index Terms—Graphics hardware, image reconstruction, tomography.

I. INTRODUCTION

VARIOUS methods for three-dimensional (3D) computed tomography (CT) reconstruction have been devised in the past three decades. While analytical approaches can be traced back to the Radon transform, iterative algorithms seek to optimize some objective function, such as maximum likelihood or minimal error. All of these algorithms have in common a series of backprojection operations which dominate the computational cost. In addition, iterative algorithms also incorporate a series of forward projections, which incur similar computational expense. Thus, to be useful in clinical practice, the backprojections (and projections) have to be made as efficient as possible. However, this goal stands in stark conflict with the complexity of these operations. Each projection/backprojection has a complexity on the order of the size of the volume dataset, which is $O(N^3)$. This complexity is always present,

unless recursive [3] or Fourier space [2] approaches are employed. These, however, have their own limiting constraints, since the need to reduce domain interpolation artifacts [3], [16] via oversampling increases the multiplicative constant in the complexity term. In this paper, we shall assume straightforward projection/backprojection in the spatial domain, at complexity $O(N^3)$. In this case, the only way to reduce the actual computational cost is to reduce the constant factor k that relates the complexity $O(N^3)$ to the computational cost $k \cdot N^3$. Unfortunately, even the most clever programming with cache-aware algorithms and fast differencing schemes can only reach a limited peak performance, when implemented on general-purpose CPUs. The general practice of precomputing the weight matrices in iterative reconstruction can yield tremendous speedups in two-dimensional (2D) reconstruction, but the memory cost involved makes the use of such precomputed matrices infeasible for 3D reconstruction. For this reason, a number of commercial custom-hardware based solutions have become available. One such approach (by TeraRecon, Inc.) uses an application specific integrated circuit (ASIC), while another (by Mercury Systems, Inc.) uses a field programmable logic array (FPGA). Both reach very impressive speeds for Feldkamp's cone-beam algorithm [8], but they do not implement any iterative algorithms, such as expectation maximization (EM) [22] or algebraic reconstruction technique (ART) [11], which are preferable for functional imaging, such as single photon emsision computed tomography (SPECT) and positron emission tomography (PET). The special-purpose proprietary boards are also quite expensive, in the range of five-digit dollar figures, and furthermore, their static custom hardware design makes them inflexible for modification and generalization. Hence, while it is economically viable to augment already expensive tomography scanners in need for stable and proven reconstruction algorithms with such hardware boards, less expensive and more flexible solutions are desirable for researchers and experimental clinicians.

When defining an appropriate platform, it helps to realize that the projection/backprojection operations, as well as the other operations involved in the grid updates and correction computations, are straightforward voxel- and pixel-based operations, which have few dependencies and are usually computed as array operations within a long loop. A very suitable platform for these kinds of calculations are vector processors or massively parallel architectures [5]. Vector processors view their input data as streams, which are combined by operators to produce an output stream. Also, while CPUs must decode every instruction in a loop, vector processors execute the entire array operation within one instruction, amortizing the cost for the single instruction decode over the entire loop. Paired with extremely high memory

bandwidth, programmable vector processors can accomplish array-based computations at impressive speeds. Unfortunately, vector processors, such as the Cray supercomputer family, are expensive machines and very few people have access to them. An exciting new development in this regard is the emergence of a mainstream computing platform that bears many features of vector processors—graphics processors (GPUs). Graphics applications fit the same instruction multiple data (SIMD) programming model of vector processors well. They typically consist of largely independent compute and data-intensive operations—the screen rasterization of large numbers of texture-mapped polygons—which expose both small-grain (per-polygon calculations) and large-grain (per-polygon list calculations) parallelism. Graphics-heavy applications, such as computer games or engineering design, require ever-increasing complex scenes to be rendered at rates of 30 frames per second, and these large consumer-driven demands have led to an unparalleled growth in the development of platforms that can satisfy these needs. The development of graphics hardware grows so fast that the chip performance doubles every six months, tripling Moore's law. These GPU boards, such as the NVidia FX 5900 or the ATI Radeon 9800, gain their speed by devoting significantly more chip real estate to the computational engine than a general-purpose CPU, such as the Intel Pentium Processor. They implement what is referred to as a stream processor, which has become a widely researched computing paradigm for high-performance computing [13]. By casting the projection/backprojection operations as well as all other CT calculations in terms of stream operations (or *fragment rasterization* operations, in graphics parlance), we can exploit these affordable mainstream architectures to achieve rapid CT.

This paper will outline how the most popular CT algorithms, such as filtered backprojection, algebraic methods, and EM methods, can be mapped onto GPU architectures for unprecedented speeds, yet without significant loss in accuracy. In Section II, we will first discuss previous work in this area, and in Section III we explain the general workings of a graphics pipeline. Sections IV and V will give the theoretical and practical implementation details, respectively, of our approach. In Section VI, we present results, while Section VII offers final conclusions as well as an outlook onto the work ahead.

II. PREVIOUS WORK AND IMPACT

During the 1990s, only midrange workstations, such as the SGI Octane or Onyx, which were available at a cost of over $20,000, had the level of graphics hardware necessary for CT reconstruction. The first works that sought to exploit this hardware for the acceleration of CT was by Cabral, Cam, and Foran [6], who implemented an analytical Feldkamp-type algorithm, and Mueller and Yagel [18], who described the implementation of an iterative method—the simultaneous algebraic reconstruction technique (SART) [1]. With the emergence of low-cost PC-based graphics hardware of similar capabilities than that of the SGI, more recent work by Chidlow and Möller [7] focused on this platform. Using a NVidia GeForce 4, these authors implemented another iterative algorithm—the maximum-likelihood expectation maximization method (ML-EM) [22], and its

faster cousin, ordered-subsets EM (OS-EM) [12]. However, all of the above approaches suffered from the circumstance that the graphics hardware they employed only had integer-arithmetic at 8-bit precision (PC) or 12-bit precision (SGI). This severely limited their accuracy and performance. With integer arithmetic at this precision, one cannot perform the accumulation operations of the projection and backprojections in hardware. Also, the short precision limits the accuracy of the (sometimes small) grid corrections in iterative algorithms. For this reason, the accumulation operations had to be performed outside the GPU, on the CPU, which involved expensive data transfers between these two entities. A (virtual) 16-bit extension of the precision could be achieved by splitting high-precision calculations among two of the four color channels (red, green, blue, and alpha). A similar mechanism could also be employed to facilitate a subset of the accumulations (16 for a 4-bit virtual extension) in hardware. Although quite effective, this mechanism was only partially accurate since it dropped the lower 8 bits of the high-end channel. Fig. 7(b) shows a reconstruction of the Shepp–Logan brain phantom that was obtained using the hardware-accelerated SART algorithm. Although the result is clearly not satisfactory for the 0.5% contrast shown, reconstructions for higher contrasts (1% or 2%) were quite acceptable, and impressive speedups in the range of 35–68 could be obtained, when compared with a CPU-based method (see [18] for more detail).

A further limitation of these older generations of graphics hardware was their lack of programmability. For example, divisions are necessary for the normalization step in the iterative algorithms but were not supported on these older platforms. The major leap forward made by new generations of GPUs is the fact that they offer programmability at floating point precision at two stages in the graphics pipeline (we will elaborate on this further in Section III). A direct consequence of this added functionality is that now the entire reconstruction can be performed *within* the GPU, at CPU precision. Thus, there is no longer a need to export and import data from and to the CPU, which overcomes the severe bottlenecks inherent in these data transfers. Also, a direct consequence of the GPU-resident computation is that the generated data can be easily visualized. Since in "normal" settings the GPU's main job is the rendition of graphics images, one can simply inject a volume rendering or a volume slicing cycle into the reconstruction and then map the resulting image to the screen-visible portion of the GPU's framebuffer.

III. THE GPU GRAPHICS PIPELINE

Graphics objects are typically composed of polygon meshes, where additional surface detail can be modeled by affixing (or *mapping*) images (or *textures*) of the desired detail onto the polygons during the rendering phase. Texture mapping is an efficient way to provide intricate surface detail without increasing an object's polygon count, and graphics hardware is highly optimized to perform texture mapping very fast, even under perspective distortion [10].

There are three main stages in a graphics pipeline (see Fig. 1): the *geometry processing stage*, the *polygon rasterization stage*, and the *fragment processing* stage. In the first stage, the geometric
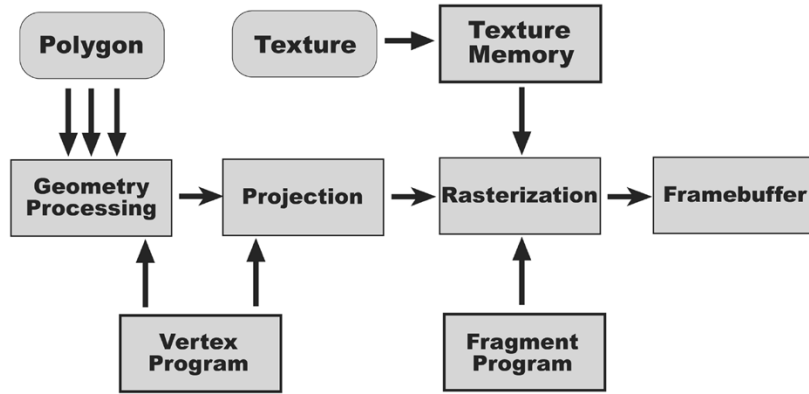
Fig. 1.   Graphics hardware rendering pipeline.

information, i.e., the vertex coordinates, are transformed to determine the screen space coordinates of the projected vertices. Then, in the second stage, these projected vertices are connected to form the (projected) polygon, whose content is filled (or rasterized), combining colors interpolated from the polygon's vertex attributes and from the mapped texture. The pixels so generated are called screen *fragments*, which can be further processed for lighting and shading effects in the third stage. In the most recent hardware, the geometry and fragment processing stages are programmable, offering a means to load small programs, called *vertex shaders* and *fragment shaders*, respectively, into the ALUs of these units at run-time. We will make use of the fact that the fragment programs can be utilized to compute any mathematical equation involving one or more input vectors. For example, one may want to compute the sum $C = A + B$ of two 2D $N \times N$ input arrays A and B. One can accomplish this by storing A and B into two textures, mapping both textures onto the same (square) polygon of size $N \times N$, and rasterizing this polygon to the framebuffer (the screen) under no magnification and in orthographic viewing. The framebuffer will then contain the array C. On the NVidia FX 5900, fragment programs can accept up to 16 different input vectors or *streams*, can be up to 1024 instructions long, and can implement most standard functions found in programming languages, such as square roots, trigonometric functions, etc. Standard application programmer interfaces (APIs) resembling high-level programming languages exist (e.g., CG from NVidia, Santa Clara, CA [17] and *Brook* from Stanford University, Stanford, CA, [4]). Apart from the large memory bandwidth that is available to access and store the data, high performance also results from the fact that both vertex and fragment processors are pipelined and that there are multiple fragment processors (eight on the NVidia FX) and multiple vertex processors (three on the NVidia FX) that all operate on different fragments and vertices, respectively, in parallel. The four available color channels (red/green/blue/alpha or RGBA) present further opportunities for parallel program execution, under the constraint that the exact same computations, e.g., projections, are performed in each.

IV. THEORETICAL CONSIDERATIONS

In our paper, we shall use the volume representation of Lewitt [14] and others, who model a volume as a collection of point samples, positioned at the grid points. In this model, values at
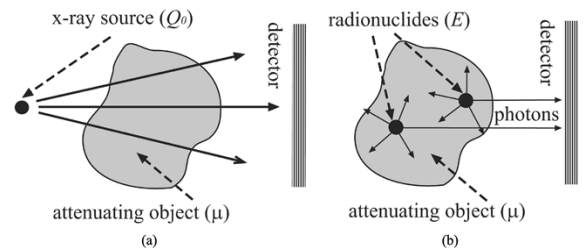

Fig. 2.   (a) Transmission imaging: an external X-ray source emits X-rays. (b) Emission imaging: internal radionuclides emit photons at sites of biochemical (metabolic) activity. Both are attenuated by the object's densities.

off-grid positions are estimated from the grid samples via interpolation with some kernel function. While [14] has proposed the use of pre-integrated Bessel functions (so-called *blobs*) for this purpose, we will employ linear functions, which have also found widespread use in backprojectors and, as we shall see later, lend themselves well for implementation in graphics hardware.

Before describing how GPUs can be exploited to perform all calculations occurring in a variety of popular CT algorithms, it is helpful to establish a common notation for these. For this purpose, let us assume a volumetric object composed of a material with attenuation function $\mu(x, y, z)$ and separately irradiated by two imaging modalities: transmission and emission X-ray. In transmission X-ray [see Fig. 2(a)], the source is located outside the object and a ray emanating with initial (source) intensity Q_0, traversing the object, and collecting in bin (u, v) of a 2D detector oriented at angle φ will be recorded with intensity

$$C_\varphi^Q(u, v) = Q_0 \cdot e^{-\int_0^L \mu(t)dt}. \tag{1}$$

Here, t is a parametric variable defined along the ray, and L is the distance between the source and the detector bin.

On the other hand, in emission X-ray [see Fig. 2(b)], the sources are the metabolic activities $E(x, y, z)$ located inside the object, each attenuated by the material between it and the detector. Integrating over all metabolic sources along the ray (shown as a dashed line) orthogonal to detector bin (u, v) gives the energy

$$C_\varphi^E(u, v) = \int_0^L E(s) \cdot e^{-\int_0^s \mu(t)dt} ds \tag{2}$$

where s is a parametric variable defined along the ray, and L and t are defined as in (1).

To illustrate the amenability of CT for vector processing, let us choose an appropriate notation. For this, we denote $C_i^Q = C_\varphi^Q(u,v)$ and $C_i^E = C_\varphi^E(u,v)$ for $0 \le i < M_\varphi$, where $M\varphi$ is the total number of pixels (rays) in the projection acquired at detector angle φ.

By further setting $q_i = -\log\left(C_i^Q/Q_0\right)$, the transmission X-ray (1) can be written as

$$q_i = \int_0^L \mu(t)dt. \tag{3}$$

Since we would like to reconstruct the values at the volume grid positions, it makes sense to rewrite (3) in the following alternative, voxel-centric form:

$$q_i = \sum_{j=0}^{N^3-1} \mu_j w_{ij}. \tag{4}$$

Here, a w_{ij} is the weight with which the object voxel j (of value μ_j) contributes to detector pixel i (with final value q_i). These weights are determined by the interpolation filter [14] and the integration rule.

On the other hand, the emission X-ray (2) indicates that the emissive quantity $E(s)$ is attenuated by the material's μ between site s and the detector. Returning to the voxel-centric representation of (4), now using the E_j as the values stored at the grid points, the projected emissive contribution $e_i(s) = C_i^E(s)$ originating at any s is

$$e_i(s) = \sum_{j=0}^{N^3-1} E_j w_{ij}(s). \tag{5}$$

Note that here the $w_{ij}(s)$ are not only given by the voxel weights, as in the transmission case, but now also incorporate the attenuation integral up to s. The equation for the total projected emissive energy is then given as

$$e_i = \int_{s=0}^L \left(\sum_{j=0}^{N^3-1} E_j w_{ij}(s) \right) ds. \tag{6}$$

Reordering the integral yields

$$e_i = \sum_{j=0}^{N^3-1} E_j \int_{s=0}^L w_{ij}(s)ds = \sum_{j=0}^{N^3-1} E_j w_{ij}(a). \tag{7}$$

Here, the $w_{ij}(a)$ combine the voxel weights of (4) and the attenuation factors. The subscript (a) is used to denote that the w_{ij} contain a factor for attenuation correction. We observe that this equation is very similar to (4). Therefore we conclude that we can project the emission volume E with methods similar to those that reconstruct the attenuation volume μ, given knowledge about the more complicated $w_{ij}(a)$ (in case we do not care about the ray attenuation, we simply use the basic w_{ij} of (4) for the projection of the emission volume). Thus, by generalizing
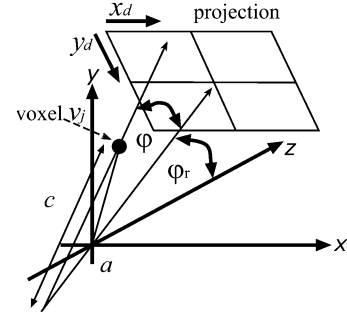

Fig. 3. Projection geometry for Feldkamp's algorithm.

(E_j, μ_j) to v_j and (e_i, q_i) to p_i, we can formulate a generalized projector and, by exchanging the roles of v_j and p_i, we obtain a generalized backprojector

$$p_i = \sum_{j=0}^{N^3-1} v_j w_{ij} \quad v_j = \sum_{t=0}^{M_\varphi-1} p_i w_{ij}. \tag{8}$$

In the following, we will denote the projection operator in the first part of (8) by $P_\varphi(V)$ and the backprojection operator in the second part of (8) as $B_\varphi(I)$. Here, V is the volume data vector (subject to reconstruction), I is an image data vector, and the projectors/backprojectors are matrices operating on them. However, in our framework the matrix elements, i.e., the w_{ij}, will not be stored explicitly but computed on the fly, using the interpolators in the rasterization hardware. We will now express the various reconstruction methods by ways of these operators.

In the Feldkamp algorithm [8], the w_{ij} are multiplied by a depth correction factor during backprojection (see Fig. 3):

$$w_{ij}(d) = w_{ij} \frac{a^2}{\left(a + \sqrt{Y(v_j) + Z(v_j)}\cos(\varphi - \varphi_r)\right)^2}. \tag{9}$$

Here, Y and Z return a voxel's y and z coordinate and φ_r is the principal orientation angle of the rth projection, with $0 \le r < S$ and S being the total number of scanner-acquired projection images. Finally, $w_{ij}(d)$ is the depth-weighted w_{ij} in (9). Using our shorthand notation, the backprojection process is written as

$$V = \sum_{\varphi \in S} B_{\varphi(d)}(I_\varphi) \tag{10}$$

where I_φ is the image obtained from the scanner at angle φ.

The iterative method SART [1] updates the grid on a projection basis. This turns out to be more convenient than the related (ray-based) ART [11] when used in conjunction with texture mapping hardware. Using our notation, SART's grid update equation is

$$V = V + \frac{B_\varphi\left(\lambda\frac{I_\varphi - P_\varphi(V)}{P_\varphi(W)}\right)}{B_\varphi(W)} \tag{11}$$

where λ is a relaxation factor. $P(W)$ and $B(W)$ denote the projection/backprojection of the weights for normalization, which can be performed using a unity I and V, respectively.

Finally, the OS-EM [12] algorithm is written as

$$V = \frac{V}{\sum\limits_{\varphi \in OS} B_{\varphi(a)}(W)} \sum\limits_{\varphi \in OS} B_{\varphi(a)} \left(\frac{I\varphi}{P_{\varphi(a)}(V)} \right) \qquad (12)$$

where OS is one of the ordered subsets of S.

We observe that, computationwise, the only real difference among these reconstruction methods is how the results of the projection/backprojection operators are combined. However, these combination operations are straightforward vector calculations. We will now discuss how (10)–(12) can be efficiently realized in GPU hardware.

V. IMPLEMENTATION

In graphics hardware, just as images, volumes can also be represented as textures. There are two choices: a stack of 2D textures or a single 3D texture. While both allow projection, there is currently no facility that would allow a backprojection into a 3D texture. We therefore store a volume as two stacks of 2D textures (see Fig. 4), one each for projections along the x and the y main viewing axes. We do not need a z-major texture stack, since we only acquire data in a circular orbit about the z-axis.

A. Projection

Perspective (cone-beam) projection is a straightforward operation with 2D textures (we shall consider parallel-beam a subset of perspective). We can approximate (3) and (4) using 2D textures as follows [see Fig. 5(a)]:

$$\hat{q}_i = \sum_{k=0}^{L/\Delta t} \mu(k\Delta t)\Delta t \rightarrow \hat{q}_i = \sum_{k=0}^{N-1} \sum_{l=0}^{N^2-1} \mu_{lk} w_{ilk} \Delta t_i. \quad (13)$$

The left part is a discretized form of (3), which is further approximated into the form of the right part by adapting (4), grouping the N^2 voxels within each of the N volume slice textures. The true voxel index j can be derived from the index lk used in (13), where k indexes the slice and l the voxels in the slice. There are two approximations here. First, the integration is now a discrete trapezoidal one, with the stepsize Δt varying across the slice (denoted as Δt_i in (13)). Since Δt is never greater than $\sqrt{3}$, this is a reasonable approximation. Second, the w_{ilk} are computed via bilinear interpolation—a square neighborhood of four slice voxels will contribute to each $\hat{q}_{ik}$ [see Fig. 5(c)]. We can compensate for the varying Δt by precomputing a sampling interval texture for each orientation angle in the set and multiplying this texture with the texture of the projection result, on the GPU.

We now look into the projection of the emission volume. If attenuation is not modeled, then the mechanism of (13) will readily apply, simply substituting (μ, q) by (E, e). However, attenuation modeling can improve reconstruction results considerably (see, e.g., [9]), and our hardware-based approach can
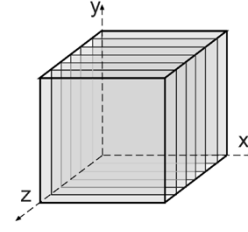

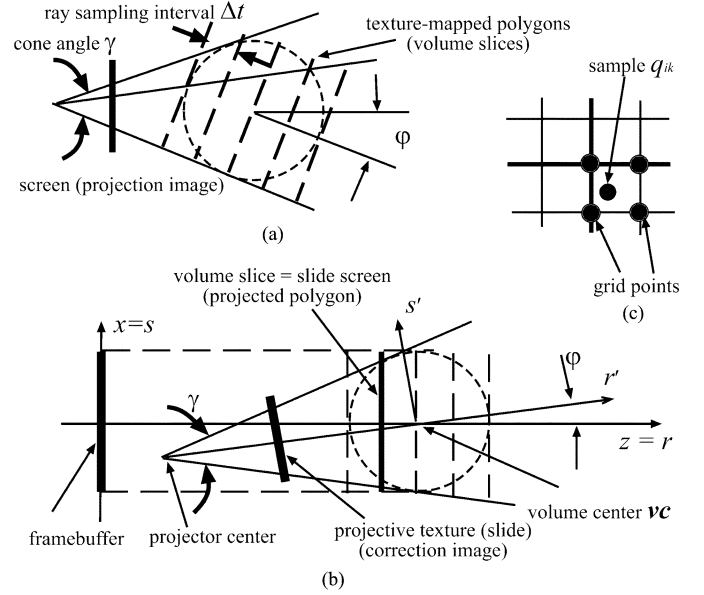Fig. 4. Volume representation as a stack of 2D textures.

Fig. 5. (a) 3D forward projection with 2D texture slices (for simplicity of illustration, only the 2D case is shown). (b) Backprojection of a correction image texture onto a volume slice. (c) Texture four-neighborhood (grid points) bilinearly interpolated by a sample.

realize this efficiently. We first discretize (2), in a fashion similar to the first part of (13) (here, we use our notational identity $C_\varphi^E(u,v) = C_i^E = e_i$):

$$C_i^E = \int_{s=0}^{L} E(S) \prod_{n=0}^{s-1} e^{-\int_{t=n}^{n+1} \mu(t)dt} ds$$

$$\approx \int_{s=0}^{L} E(s) \prod_{n=0}^{s-1} \left(1 - \int_{t=n}^{n+1} \mu(t)dt \right) ds$$

$$\approx \sum^{L/\Delta s} E(k\Delta s) \prod^{k-1} ((1 - \mu(n\Delta t))\Delta t)\Delta s. \quad (14)$$

Here, we map the μ volume to a range $[0.0, \ldots, 1.0]$. The error of the Taylor series approximation of the exponential is within reasonable bounds since the interval Δt is never greater than $\sqrt{3}$. The final expression in (14) allows us to convert (7) into the texture slice-based representation, similar to the second part of (13)

$$\hat{e}_i = \sum_{k=0}^{N-1} \sum_{l=0}^{N^2-1} E_{lk} w_{ilk(a)} \Delta s_i \qquad (15)$$

where $w_{ilk(a)}$ is the product of the interpolation weight w_{ilk} for the emissions in slice k, and the product of the slicewise interpolated attenuations up to slice $(k-1)$

$$w_{ilk(a)} = w_{ilk} \prod_{n=0} (1 - \mu(n\Delta t))\Delta t$$
$$= w_{ilk} \prod_{n=0}^{k-1} \left(1 - \sum_{m=0}^{N^2-1} (\mu_{nm} w_{imn}) \right) \Delta t_i. \quad (16)$$

Here, the w_{imn} are also determined by the interpolation filter, and n indexes the slices and m the voxels in the slices.

We compute the attenuation part of the $\hat{w}_{ilk(a)}$ in (15) recursively, via implementation of a variant of the familiar volume rendering front-to-back compositing equation [15]

$$e = e_s \cdot (1 - \mu) + e \quad e = e_s \cdot T + e$$
$$(1 - \mu) = (1 - \mu)(1 - \mu_s) \quad T = T \cdot T_s. \quad (17)$$

Here, the two columns hold equivalent expressions, with T denoting transparency. The e_s and μ_s are the newly interpolated values, while e and μ are the recursive variables. Note, that in contrast to volume rendering, e can grow past 1.0. In the end, e holds the emission volume projection, properly attenuated by μ. Equation (17) states that we must maintain a texture buffer for transparency T and one for emission e, and that we must multiply T with the newly interpolated emission. Two texture volumes are required: one for the emission volume that is being reconstructed and one for the attenuation volume, possibly obtained via a prior transmission CT.

B. Backprojection

The grid updates in (10)–(12) all have a similar backprojection term, which can be written as

$$dv_j = \sum_{i \in I_\varphi} d_i w_{ij} \quad (18)$$

where dv_j is the update to a voxel j, derived from grid update factors d_i. The w_{ij} are determined similarly as outlined for the projection case. For emission tomography, matched projector/backprojector pairs [23] that use full attenuation modeling (and other effects) only for the projection phase, but not for the backprojection phase, have been proposed and can be implemented by using w_{ij} in place of $w_{ij(a)}$. However, it is desirable to use the same w_{ij} in both projection and backprojection.

The 2D slice texture approach allows us to achieve the desired equivalent mapping by using *projective textures* [21] for the backprojection, which is illustrated in Fig. 5(b). Essentially, projective textures work similar to a slide projector. The backprojected image forms the "slide," which is perspectively projected onto the "screen" formed by a polygon that is placed at the location of the volume slice to be updated. The "slide projection" is then "viewed" in parallel projection mode on the screen. The perspective transform is given by the viewing geometry at which the projection was originally obtained from the scanner. Using this mapping, the weight with which a voxel j contributes to a projection image pixel i is identical to the weight that a correction d_i coinciding with i has on j. Projective textures can
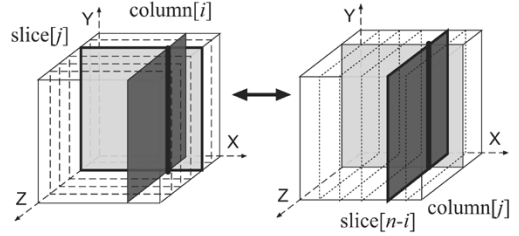

Fig. 6. Texture stack updates when the major projection direction switches from one stack to the other.

be implemented by filling the hardware texture mapping matrix with the appropriate values in a vertex program (see [18] for further detail) and performing the actual projective mapping in a fragment program.

We implemented the depth-weighting in Feldkamp's algorithm (9) using 2D lookup textures (called *dependent textures*), one for each principal projection orientation angle φ_r. This dependent texture is indexed by the voxel y- and z-coordinates (the x-coordinate is not relevant) at run-time, and the looked-up value is multiplied by the w_{ij}.

The attenuation weighting can be implemented in two ways: 1) as an interleaved projection/backprojection procedure, or 2) as a projection followed by a backprojection. The former can be formulated as follows, with D being initialized as the grid update image (computed from scanner image and projection), DV being the volume that accumulates the updates, and μ being the attenuation volume:

```
for each volume slice k = 0,...,N − 1, going
  in front-to-back order
    backproject D onto DV_k
    project μ_k onto D executing blending
      D = D · (1 − μ_k)
```

For the alternative second method, we precompute a new set of textures D_k by first rendering all projections of the μ_k slices (with the blending), saving them in texture memory, and then performing all backprojections using these D_k. This saves the somewhat expensive projection/backprojection context switches, but it consumes more storage in a texture memory. The algorithm is written as

```
initialize D_0 to D
  for each volume slice k = 1,...,N − 1,
    going in front-to-back order
      project μ_k onto D_{k−1} executing
      blending D_k = D_{k−1} · (1 − μ_k)
for each volume slice k = 0,...N − 1, going
  in front-to-back order
    backproject D_k onto DV_k
```

It needs to be mentioned that both SART and EM also require a volume that stores the w_{ij} for each updated voxel, to be used later for normalization. In practice, we have found that SART does not require normalization, due to the bilinear filter weights, while for EM we can just normalize by the number of projections in the subset (similar to [7]). However, if attenuation

TABLE I
TIMINGS FOR THE RECONSTRUCTED VOLUMES SHOWN IN FIG. 7. THE ITERATIVE ALGORITHMS USED THREE ITERATIONS OF
PROJECTION/BACKPROJECTIONS. (THE MISSING VALUES WERE NOT COLLECTED WHEN THE EXPERIMENTS WERE CONDUCTED)

Platform	Algorithm	Projections	Precision	Projection	Backprojection	Iteration	Total
SGI - hardware	SART	80	12-bit (extended)			1.1 min	3.1 min
PC - CPU	SART	80	floating point	75 s	75 s	2.5 min	7.5 min
PC - GPU	SART	80	floating point	0.4 s	9 s	12 s	36 s
PC - GPU	OS-EM	80	floating point	0.9 s	17 s	21 s	63 s
PC - GPU	Feldkamp	80	floating point	n/a	5 s	n/a	5 s
PC - GPU	Feldkamp	160	floating point	n/a	9 s	n/a	9 s

correction is applied, a weight volume W must be accumulated, which we accomplish by backprojecting a two-channel texture (D, μ) into a two-channel texture stack (DV, W).

C. Correction Texture Computation and Voxel Updates

In the iterative schemes, both the computation of the correction texture D and the new state of the voxel textures V (i.e., E or μ) are pixelwise operations, implemented as simple texture blends. Denoting an original, acquired projection as OP, the calculated projection as P, and a projection of the weights as W, the (vector or stream) calculation of D can be written as:

- $D = \mathrm{DIV}(\mathrm{SUB}(OP, P), W)$ for SART;
- $D = \mathrm{DIV}(OP, P)$ for EM.

The voxel update after backprojection of all projections in the set has occurred can be written as

- $V = V + DV$ for SART;
- $V = (V \cdot DV)/W$ for EM, where W is the accumulated weight volume, if attenuation correction is used.

In the iterative algorithms, our use of two stacks of 2D textures will lead to inconsistencies if one stack of textures is updated by ways of backprojection but the other is not. Therefore we must update a texture stack whenever its projection proceeds an update of the other texture stack. This is frequently the case since two subsequent projections should be close to orthogonal to maximize the rate of convergence. We can accomplish a stack update by adding an up-to-date column in the source stack texture to the corresponding out-of-date column in the destination stack texture (see Fig. 6).

D. Parallel Execution Via the RGBA Color Channels

All three CT algorithms can exploit the inherent parallelism offered by the four color channels. In Feldkamp's filtered backprojection (FBP), we may pack the data of four orthogonal projections into the RGBA channels, since they share the same projection matrices. This gives rise to a four-way (RGBA) texture stack, one each for the four 90° intervals processed in the four channels. Then, after all backprojections are completed, the volume is assembled by adding the data in the four channels, using a technique similar to the stack update described before. This speedup strategy requires that the projections were acquired at orthogonal angle increments.

In SART, we cannot project/backproject orthogonal projections in parallel due to the projectionwise volume update.

Here, we *fold* the upper and lower halves of the volume and the projections into the RG channels, while the BA channels are used to accumulate the weights in the projection phase. The two halves share the same projection matrices, just reflected about the midline. Projections, backprojections, and texture stack updates all use this decomposition and the complete volume is only assembled at the end, by merging the RG channels. The Feldkamp algorithm can also use this folded partition, which reduces the number of required texture stacks to two.

The unattenuated EM algorithm can employ a four-times parallelism, either using the folded decomposition in conjunction with two orthogonal projections or the unfolded four-way scheme. However, the latter incurs significant overhead, both for volume assembly and volume distribution from and to the four color channels each time a subset has been processed, which only amortizes when the subsets are large. We therefore chose to use the folded two-way approach. Note that this EM parallelism poses certain constraints on the composition of the subset. The parallelism in the attenuated EM algorithm is only twofold since two channels are needed for each projection/backprojection to hold (μ, e) and (DV, W), respectively.

Currently, GPU boards have 256 MB of texture memory. Let us assume a volume of size N^3 and N projections of size N^2. For any algorithm, one texture stack of a (floating point precision) volume will take up $4 \cdot N^3$ bytes, and the N 8-bit scanner images of size N^2 will take up N^3 bytes (or $2 \cdot N^3$ for 16-bit projections). The Feldkamp algorithm requires an additional N^3 bytes for the depth images, and its total memory requirement with a four-way stack is then $(4 \cdot 4 + 1 + 1 = 18) \cdot N^3$ bytes. Hence, the sidelength N of a volume is $N_{\mathrm{Feldkamp4}} = \sqrt[3]{256 \, \mathrm{MB}/18} = 242$. For the two-way, folded decomposition, the memory requirement is $(2 \cdot 4 + 1 + 1 = 10) \cdot N^3$, and $N_{\mathrm{Feldkamp2}} = 294$. SART does not need depth images; thus, its memory requirements are slightly less than Feldkamp2, i.e., $9 \cdot N^3$ bytes and $N_{\mathrm{SART}} = 305$.

Unattenuated EM in the folded configuration additionally requires two texture stacks for the temporary accumulated results, bringing the total to $(2 \cdot 4 + 2 \cdot 4 + 1 = 17) \cdot N^3$ bytes, with $N_{\mathrm{EM\_UA}} = 246$. Regular attenuated EM using the two-stage mechanism requires six volumes: 1) the two destination volumes—temporary accumulations and weights; 2) the two source volumes—attenuated corrections and weights; and 3) the two final volumes—reconstruction and attenuation. In a two-way
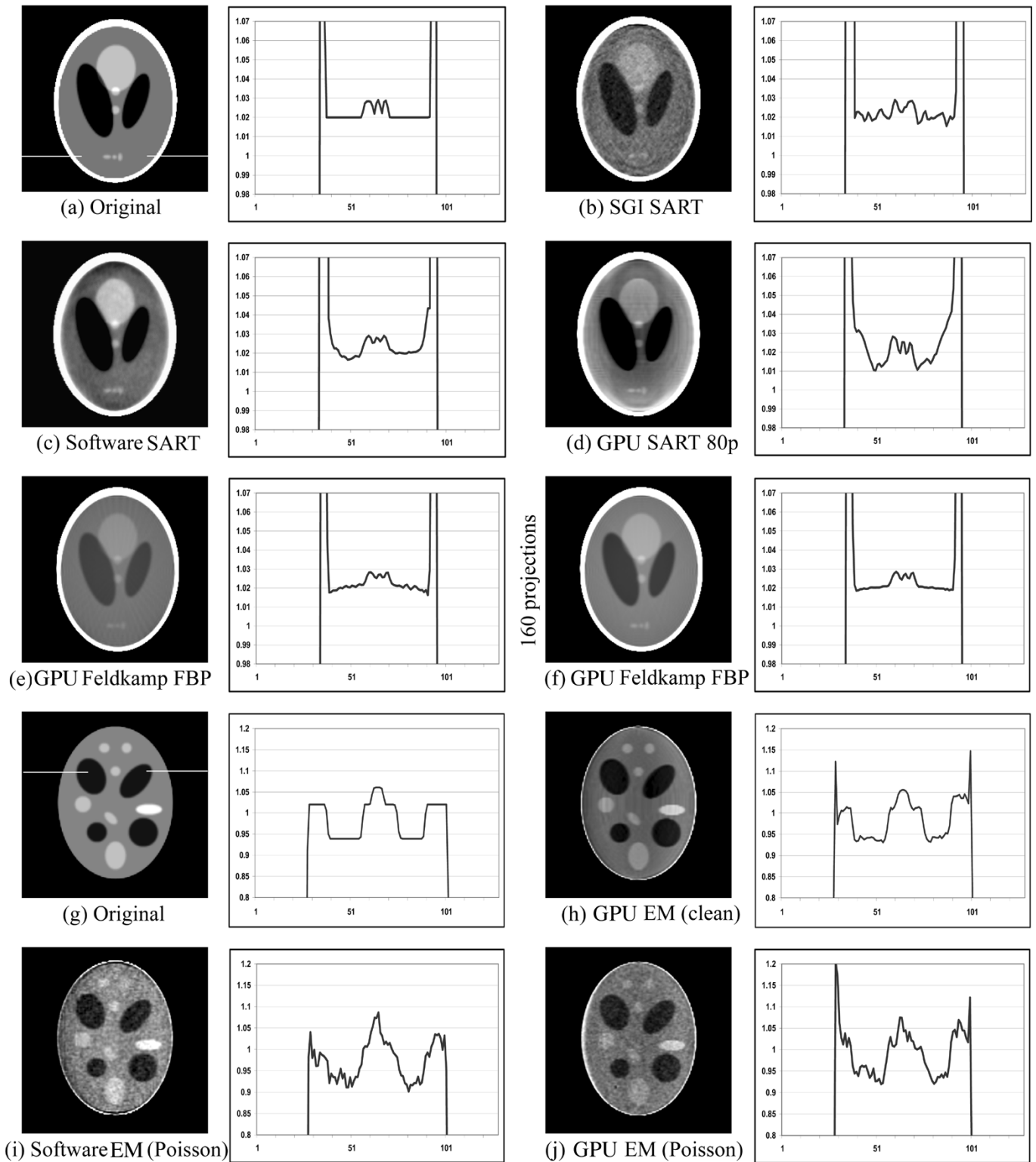
Fig. 7. Slice across the 3D (a)–(f) Shepp–Logan brain phantom and (g)–(j) ellipsoid phantom, reconstructed as a 128^3 volume from a set of 80 analytically computed projections of size 128^2 each [160 projections for (f)]. The iterative algorithms used three iterations of projection/backprojections. (i) and (j) are from the ellipsoid phantom, with random Poisson noise added to the projections. The plots show the intensity profiles across the center of three small ellipsoids near the bottom of the phantom in (a)–(f) and near the top of the phantom (g)–(j), as indicated by the white line in (a) and (g).

configuration, this would require $(12 \cdot 4 + 1 = 49) \cdot N^3$ bytes, enabling $N_{\text{EM\_}A} = 173$, while in a folded configuration, it would require about half of that, i.e., $25 \cdot N^3$ bytes, enabling $N_{\text{EM\_}A} = 217$. It should be noted, however, that the recently announced GPU hardware will have 512 MB of texture memory, and thus volumes 1.25 times larger will be feasible, without the need to divide them into sections or blocks for separate reconstruction.
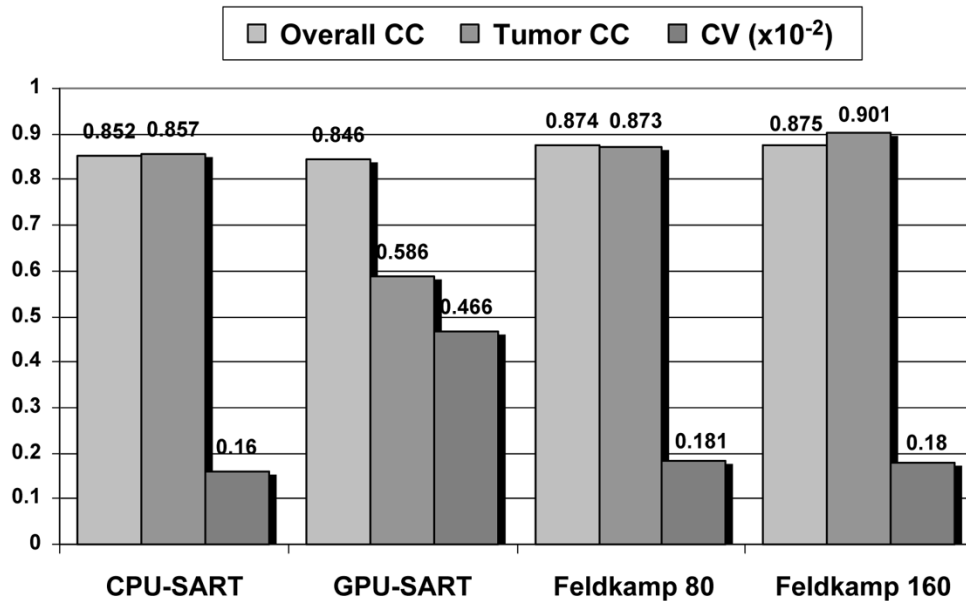
Fig. 8. CC and background CV for various reconstructions.

VI. RESULTS

Table I lists the timings obtained in our experiments. All CPU and GPU results were produced on a 2.66 GHz Pentium PC with 512 MB of memory, hosting a Nvidia FX 5900 GPU. We employ a 3D version of the Shepp–Logan brain phantom (of size 128^3) [19] at the original 0.5% contrast level to demonstrate reconstruction quality. Fig. 7 shows slices across the reconstructed phantoms, while line plots provide further insight into reconstruction fidelity and noise. These plots are obtained from the intensity profile along the line indicated in Fig. 7(a) [and Fig. 7(g) for EM]. Finally, Fig. 8 presents a formal error analysis, where we compute the correlation coefficient (CC) of the phantom with the reconstruction, both within the entire skull and within an ellipsoid just enclosing the three small "tumors" at the bottom. We also compute the coefficient of variation (CV) over four ellipsoidal regions with uniform content. Here, the CV for region i is $\mathrm{CV}_i = \delta_i/\mu_i$, where μ_i is the average and σ_i is the standard deviation of the region's voxel values [19], [20].

It is interesting to observe that a current, fairly optimized CPU implementation, using first-order (linear) interpolation filters, runs already at about half the speed of the (older) SGI texture mapping hardware implementation. We also note that the new inexpensive floating point GPUs can reconstruct a volume with SART at about 12 times the speed than this same-generation CPU, and about five times faster than the older SGI hardware. Meanwhile, the GPU reconstruction quality [Fig. 7(d)] is now nearly equivalent to that obtained with the software implementation [Fig. 7(c)], which was infeasible with the integer-based SGI hardware [Fig. 7(b)]. We suspect that the remaining artifacts for GPU SART may be due to the coarser sampling due to the fixed slice distance and the trapezoidal interpolation rule.

In Table I, the Feldkamp implementation uses the four-way mechanism without folding, while SART and EM use the folded scheme. We have observed, for Feldkamp and attenuated EM, that channel parallelism via folding provides about 90% of the

speedup obtained via the unfolded scheme, due to its smaller data granularity. We also notice that, in general, projections are much faster than backprojections on a GPU, while they cost about the same on the CPU. This is due to the considerably more expensive projective textures approach in the backprojections. The remaining cost per iteration is mostly consumed by the texture stack update operations, while the correction computations require very minimal time, due to their low data complexity.

Although not shown here, similar speedups are also obtained with the GPU implementations of the other algorithms studied, due to their similar projectors/backprojectors. The GPU implementation of Feldkamp FBP produces nearly perfect results. As indicated in Fig. 7(e) and (f), doubling the number of projections from 80 to 160 can eliminate the residual streak artifacts that are common for FBP when less than N projections are used.

In order to test the EM implementation, we designed a volume more suited for emission studies. We also added Poisson noise to the analytically computed projections. The phantom consists of ellipsoids with four times the original Shepp–Logan contrast [Fig. 7(g)]. Fig. 7(h) shows an EM reconstruction without noise, and Fig. 7(i) and (j) show a CPU and GPU reconstruction, respectively, from noisy data. The GPU-based EM implementation yields fairly good results from both clean and noisy data. We notice some faint ringing and some elevated level of noise in the both GPU-reconstructed datasets. We attribute this again to the coarser sampling rate and the trapezoidal integration rule. The EM projector and backprojector are more costly than those for SART since they incur the extra cost for attenuation correction. However, we have found that the (four-way parallel, folded) unattenuated EM runs more than twice as fast as SART, since the number of texture stack updates is lower.

The line plots of Fig. 7 and the error metrics of Fig. 8 indicate that the GPU reconstructions have generally greater noise and structural artifacts, but only at moderate levels, but distinguish the phantom features quite well. The remaining artifacts are greater for the iterative algorithms than for Feldkamp, which

we believe is due to the fact that projection errors accumulate through the iterative process.

It is impressive to see that GPUs allow a Feldkamp FBP reconstruction for a 128^3 volume to be conducted at good quality in a mere 5 s. Extrapolating from these results, we expect that reconstructing a 256^3 volume from 160 projections will then require just over a minute. This near-interactive speed of a volume reconstruction will allow accelerated full CT or region-of-interest CT in image-guided surgery.

VII. CONCLUSION AND FUTURE WORK

The new techniques introduced in this paper demonstrate that the recently escalating revolution in PC graphics board technology has enormous potential for computed tomography. For the first time, the quality that can be achieved rivals that obtained with software algorithms. Yet, speedups of over an order of magnitude for both analytical and iterative reconstruction methods are possible, on easily programmable and mass-produced hardware available for less than $500 at any local computer outlet. The results are especially encouraging since GPUs have increased in performance at a triple of Moore's law in the past few years.

We found that the Feldkamp algorithm performed well with the bilinear filter that was used for interpolation (many researchers have used this filter as well). On the other hand, the iterative methods can possibly benefit from better filters (compare [14]) and/or a more narrow sampling interval, at least when the number of projections is low. Both can be programmed on the GPU, but at additional cost.

Future work will focus on the modeling the collimator/detector response and photon scattering for emission tomography, which is crucial for obtaining high-quality images from SPECT and PET, the exploitation of the faster integer arithmetic in the GPUs, as well as devising techniques for handling large datasets.

REFERENCES

[1] A. H. Andersen and A. C. Kak, "Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm," *Ultrason. Imaging*, vol. 6, pp. 81–94, 1984.

[2] C. Axelsson and P.-E. Danielsson, "Three-dimensional reconstruction from cone-beam data in $O(n^3 \log n)$ time," *Phys. Med. Biol.*, vol. 39, pp. 477–491, 1994.

[3] S. Basu and Y. Bresler, "An $O(n^3 \log n)$ backprojection algorithm for the 3D Radon transform," *IEEE Trans. Med. Imag.*, vol. 21, no. 2, pp. 76–88, Feb. 2002.

[4] I. Buck, T. Foley, D. Horn, J. Sugerman, and P. Hanrahan, "Brook for GPUs: Stream computing on graphics hardware," *ACM Trans. Comput. Graph. (Siggraph '04)*, vol. 23, no. 3, pp. 777–786, 2004.

[5] S. Butler and M. I. Miller, "Maximum a posteriori estimation for SPECT using regularization techniques on massively parallel computers," *IEEE Trans. Med. Imag.*, vol. 12, no. 1, pp. 84–89, Mar. 1993.

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

[7] K. Chidlow and T. Möller, "Rapid emission volume reconstruction," in *Volume Graphics Workshop*, 2003, pp. 15–26.

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

[9] M. King, B. Tsui, and T. Pan, "Attenuation compensation for cardiac single-photon emission computed tomographic imaging: Part 1, Impact of attenuation and methods of estimating attenuation maps," *J. Nucl. Cardiol.*, vol. 2, pp. 513–524, 1995.

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

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

[12] H. Hudson and R. Larkin, "Accelerated image reconstruction using ordered subsets of projection data," *IEEE Trans. Med. Imag.*, vol. 13, no. 4, pp. 601–609, Dec. 1994.

[13] U. Kapasi, W. Dally, B. Khailany, J. Ahn, P. Mattson, and J. Owens, "Programmable stream processors," *IEEE Computer*, vol. 36, no. 8, pp. 54–62, Aug. 2003.

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

[15] B. Lichtenbelt, R. Crane, and S. Naqvi, *Introduction to Volume Rendering*. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[16] T. Malzbender, "Fourier volume rendering," *ACM Trans. Graph.*, vol. 12, no. 3, pp. 233–250, 1993.

[17] W. Mark, S. Glanville, and K. Akeley, "CG: A system for programming graphics hardware in a C-like language," in *Proc. SIGGRAPH'03*, 2003, pp. 896–907.

[18] K. Mueller and R. Yagel, "Rapid 3D cone-beam reconstruction with SART using texture mapping hardware," *IEEE Trans. Med. Imag.*, vol. 19, no. 12, pp. 1227–1237, Dec. 2000.

[19] K. Mueller, R. Yagel, and J. J. Wheller, "Anti-aliased 3D cone-beam reconstruction of low-contrast objects with algebraic methods," *IEEE Trans. Med. Imag.*, vol. 18, no. 6, pp. 519–537, Jun. 1999.

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

[21] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. E. Haeberli, "Fast shadows and lighting effects using texture mapping," in *Proc. SIGGRAPH'92*, vol. 26, 1992, pp. 249–252.

[22] L. Shepp and Y. Vardi, "Maximum likelihood reconstruction for emission tomography," *IEEE Trans. Med. Imag.*, vol. 1, no. 2, pp. 113–122, 1982.

[23] G. Zeng and G. Gullberg, "Unmatched projector/backprojector pairs in an iterative reconstruction algorithm," *IEEE Trans. Med. Imag.*, vol. 19, no. 5, pp. 548–555, May 2000.