IOP PUBLISHING PHYSICS IN MEDICINE AND BIOLOGY

Phys. Med. Biol. **52** (2007) 3405–3419 doi:10.1088/0031-9155/52/12/006

# Real-time 3D computed tomographic reconstruction using commodity graphics hardware

**Fang Xu and Klaus Mueller**

Center for Visual Computing, Computer Science Department, Stony Brook University, Stony Brook, NY 11794-4400, USA

E-mail: fxu@cs.sunysb.edu and mueller@cs.sunysb.edu

Received 16 November 2006, in final form 1 March 2007
Published 17 May 2007
Online at stacks.iop.org/PMB/52/3405

**Abstract**

The recent emergence of various types of flat-panel x-ray detectors and C-arm gantries now enables the construction of novel imaging platforms for a wide variety of clinical applications. Many of these applications require interactive 3D image generation, which cannot be satisfied with inexpensive PC-based solutions using the CPU. We present a solution based on commodity graphics hardware (GPUs) to provide these capabilities. While GPUs have been employed for CT reconstruction before, our approach provides significant speedups by exploiting the various built-in hardwired graphics pipeline components for the most expensive CT reconstruction task, backprojection. We show that the timings so achieved are superior to those obtained when using the GPU merely as a multi-processor, without a drop in reconstruction quality. In addition, we also show how the data flow across the graphics pipeline can be optimized, by balancing the load among the pipeline components. The result is a novel *streaming CT* framework that conceptualizes the reconstruction process as a steady flow of data across a computing pipeline, updating the reconstruction result immediately after the projections have been acquired. Using a single PC equipped with a single high-end commodity graphics board (the Nvidia 8800 GTX), our system is able to process clinically-sized projection data at speeds meeting and exceeding the typical flat-panel detector data production rates, enabling throughput rates of 40–50 projections $s^{-1}$ for the reconstruction of $512^3$ volumes.

(Some figures in this article are in colour only in the electronic version)

## 1. Introduction

Recent years have seen a great commoditization of x-ray computed tomography (CT). Several companies now offer technologically advanced and extensible flat-panel 2D x-ray detectors.

  

At the same time, a variety of stable gantries with fast source–detector orbiting capabilities are also becoming commercially available. Efforts to combine these two crucial components are now well underway, in a wide spread of medical applications, such as planning or monitoring systems employed in radiotherapy (Jaffray *et al* 2002), image-guided surgery and probing (Liu *et al* 2001), trauma units with mobile scanners (Verlaan *et al* 2005), angiography (Fahrig *et al* 1997), 4D imaging for cardiac and other CT (Kondo *et al* 2005, Taguchi 2003, Mori *et al* 2004), as well as for patient positioning, instrument navigation tasks, dental applications and others. However, while this equipment offers fast acquisition of the imagery needed for CT reconstruction, the calculations required for this task have traditionally not been able to keep up with this speed, unless proprietary, inflexible and expensive special hardware was used. Fortunately, with the emergence of programmable graphics hardware (GPU), high-performance computing has also become a commodity, and this paper describes a GPU-based solution for commodity CT that pushes the envelope in this domain to a great extent.

Common to many commodity CT applications is the expectation to achieve 3D reconstructions in an expedient manner, or, when used within interventional or calibration applications, in real time. The traditional platform for flat-panel detectors is the C-arm gantry, which typically provides a spin range of 270–360° at 30–50° $s^{-1}$. However, image acquisition speed and resolution are both dictated by the flat-panel detector in use, where 30–50 projections $s^{-1}$ are typically acquired, at a matrix size of up to $1024^2$.

To enable real-time CT reconstruction, the speed of the data processing must match that of the data acquisition. More concretely, assuming an image acquisition rate of 50 2D projections $s^{-1}$, the reconstruction must proceed at the same bandwidth, that is, at a rate of 50 projection $s^{-1}$. The result will be a pipeline in which projection data are produced by the acquisition process (the detector) and are immediately consumed by the reconstruction process (the computer), giving rise to what we call *streaming CT*. In this mode of operation, the delay incurred for a full 3D reconstruction is only constrained by the speed of the gantry (assuming properly dimensioned network interconnects for data transmission).

Even highly optimized CPU-based reconstruction engines cannot achieve this degree of computational bandwidth. Typically, reconstructing a $512^3$ volume from 360 $512^2$ projections still consumes 60–100 s on a dual Pentium PC, while streaming CT would require reconstruction speeds an order of magnitude higher. The usual resort to bridge this performance gap is to employ either high-performance, specialized proprietary hardware (Application Specific Integrated Circuits (ASIC) or Field Programmable Gate Arrays (FPGA)) or a multi-node cluster, as for example the CPU/FPGA *inline reconstruction architecture* by Brasse *et al* (2005). However, all of these implementations are expensive to develop and can be difficult to modify. The GPU-based solution we outline here is able to fulfil the computational requirements of streaming CT, on a graphics board available at a price of less than $500.

A great advantage of using GPUs is that their programs are easily modified and updated, their programming model is well understood and supported by a large user base, and their existence is immensely boosted by the commercial power of interactive entertainment. These factors have resulted in a doubling of GPU performance every 6 months (triple of Moore's law governing the growth of the performance of CPUs). On the other hand, GPU on-board memory has also substantially increased (currently 768 MB to 1 GB), enabling reconstructions of realistically-sized volume datasets ($512^3$ and more) at full floating point precision on-chip. Therefore a GPU-based platform is much better prepared to scale with growing gantry speeds, projection sizes and volume dimensions.

The use of graphics hardware for CT was first attempted by Cabral *et al* (1994) for cone-beam CT with filtered backprojection (FBP) and Mueller and Yagel (2000) for iterative CT with the simultaneous algebraic reconstruction technique (SART) (Andersen and Kak 1984), both

on fixed point precision non-programmable high-end SGI workstations. Chidlow and Möller (2003) implemented emission tomography with ordered subsets expectation maximization (OS-EM) (Hudson and Larkin 1994) on a consumer-grade Nvidia GeForce 4 GPU card, which, however, had similar limitations to the SGI solutions before. Xu and Mueller (2005, 2006) were the first to use the newly emerging programmable GPUs. They were able to achieve reconstruction qualities comparable to CPU-based methods, for SART, OS-EM and FBP, also with larger datasets. Following these more fundamental works were a number of papers targeting specific CT applications, all with impressive speedup factors. Kole and Beekman (2006) accelerated the ordered subset convex reconstruction algorithm, Xue *et al* (2006) accelerated fluoro-based CT for mobile C-arm units and Schwietz *et al* (2006) accelerated the backprojection and FFT operations employed for MR k-space transforms. However, while the latter two papers describe filtered backprojection algorithms, the reported data are restricted to parallel-beam reconstructions and are therefore not directly comparable to our cone-beam results.

Besides these GPU-based efforts, there have also been recent works that exploited other high performance computing platforms for CT, in particular the Cell BE processor (Kachelrieß 2006). While the performance is quite good, the Cell BE does not fit (at least not currently) the profile of a commodity platform. Furthermore, it turns out that GPUs are in fact an excellent match for CT reconstruction, as the (back-) projection operations of CT have much in common with traditional graphics operations, which receive super-fast hardwired acceleration support in GPUs. We will show that exploiting this fact represents a major source of speedups, resulting in overall superior performance of our approach. Further, given the time-critical interaction of the various GPU pipeline components within our graphics-oriented framework, a careful load-balancing among these components is also needed to maximize the performance. Both of these topics represent the major contributions of this paper, which have not been reported in previous works. They also enable the desired streaming CT scenario, utilizing all-commodity hardware.

An important trend is also the recent move of GPU manufacturer Nvidia (AMD/ATI pursues a similar effort) to cast its latest line of GPUs also as a highly parallel processor for general purpose computing (GPGPU), releasing the CUDA (Compute Unified Device Architecture) programming environment for this purpose. While this is a laudable effort, it de-emphasizes the powerful graphics facilities that, as we show, work so well for the acceleration of CT, and experimental results to that effect will also be presented.

## 2. Overview of the GPU architecture

Figure 1 shows a schematic representation of the data flow in a standard graphics pipeline, all consisting of highly parallelized hardwired components. The input is typically a set of polygons, each defined by at least three vertex points (which are assigned one or more 4D vectors) plus connectivity information. The vertex transformation unit multiplies the incoming vertex coordinates by a $4 \times 4$ matrix. The rasterizer then reconnects the transformed vertices into polygons and overlays each such polygon onto a grid raster (the screen), assigning each raster point a *fragment* whose value is the result of a linear interpolation of the polygon's vertex vectors. This can be an RGBA colour, an index into a texture (a 1–3D array—in our case the 2D projection data), or any other 4D vector. This information is then used in the fragment processing stage, for shading and texture mapping, to colour the pixel the fragment is assigned to. Since this basic graphics pipeline has no loop dependencies, the objects simply *stream* across the pipeline, starting as a list of vertices, which generate fragments, which in turn form the basis for computing the visual attributes assigned to the corresponding screen pixels.

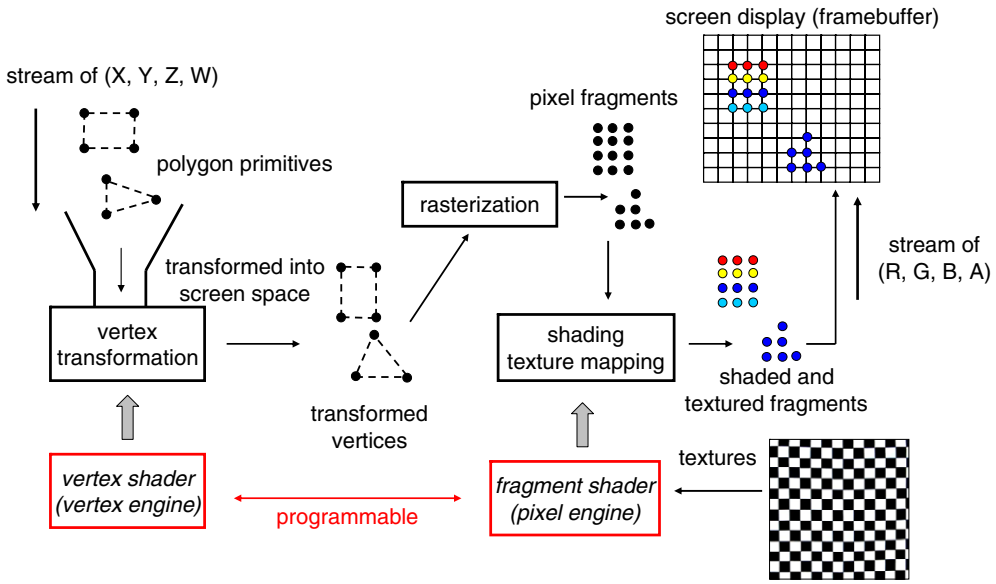


**Figure 1.** The data flow in the graphics pipeline. The streaming architecture converts graphics primitives into pixels on the display via a series of transformations. Programmable shaders are shown in red boxes in italic text.

GPUs are hence a *streaming architecture*, processing massive sets of graphics primitives, i.e., polygons and textures, in a highly parallelized fashion. In modern GPUs both the vertex units and the fragment units are programmable, via vertex and fragment *shader programs*. This greatly extends the gamut of possible operations on vertices and fragments. The rasterizer, on the other hand, remains non-programmable, but is implemented in very fast parallel hardware.

With recent GPUs, this pipeline distinction is no longer made explicit in the hardware itself. For example, the NVidia 8800 GTX features 128 uniform SIMD (Same Instruction Multiple Data) processors. These can then be viewed either as a 128-way parallel processor, in the spirit of GPGPU and in association with the CUDA (Compute Unified Device Architecture) programming interface, or as a traditional graphics pipeline, in which case the processors are dynamically assigned to vertex and fragment operations in the manner described above. We therefore have two choices: implement backprojection (i) as a multi-processing task using the 128-way parallel configuration (*MP*-GPU), or (ii) as a graphics task by ways of the Accelerated Graphics pipeline shown above (*AG*-GPU). Most previous GPU-accelerated CT reconstruction solutions, as the one of Xue *et al* (2006) mentioned before, have relied on the MP-GPU configuration, but as our comparison will show, the AP-GPU configuration is by far preferable.

## 3. Our streaming CT framework

We perform 3D reconstruction using the widely popular Feldkamp (FDK) cone-beam reconstruction algorithm (Feldkamp *et al* 1984). The FDK algorithm has two stages: (2D projection-space) filtering and (3D volume-space) depth-weighted backprojection. Our framework seeks to create a computational pipeline in which all PC-resident computing facilities (CPU and GPU) are utilized in an overall balanced manner. Thus, once a projection has been acquired on the scanner, our application performs the filtering on the CPU, using

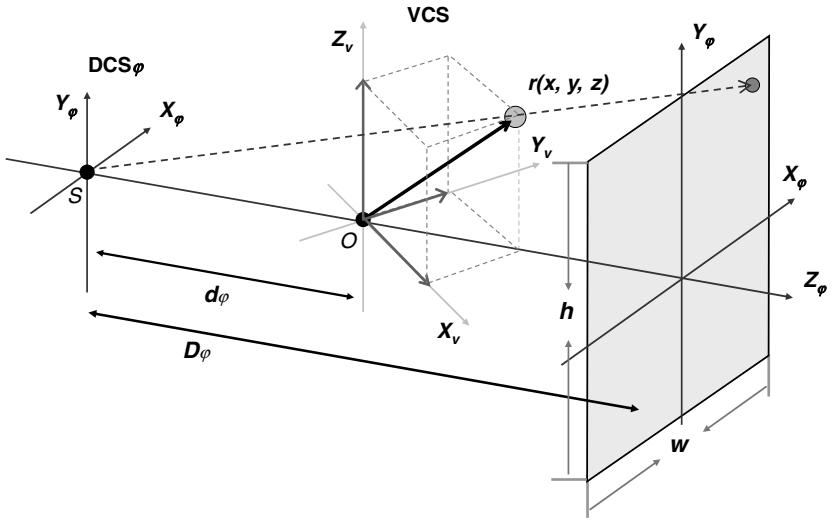


**Figure 2.** Geometry defined in the Feldkamp filtered back-projection algorithm. VCS $(x_v, y_v, z_v)$: volume coordinate system; DCS$_\phi$ $(x_\phi, y_\phi, z_\phi)$ : source–detector coordinate system; $r$: voxel to be reconstructed; $O$: rotation centre (origin); $S$: source; $w/h$: detector width/height in pixel counts; $d_\phi$: source–origin distance; $D_\phi$: source–detector distance.

FFTW (Frigo and Johnson 2005), and then streams the filtered 32-bit floating point data into the GPU to complete the backprojection. Using the Pixel Buffer Object (PBO) and Vertex Buffer Object (VBO) software interface, in conjunction with the PCI-Express memory bus, this streaming can be well overlapped with ongoing backprojections and thus does not cause significant pipeline delays. We have observed that this strategy works well in practice. It is also theoretically justifiable, considering that the complexity ratio of a projection filtering versus its backprojection (O($N^2 \log N$) versus O($N^3$)) is in good correspondence to the performance ratio of CPU versus GPU (one to two orders of magnitude). Another reason to perform the filtering on the CPU is also that 1D FFTs generally do not accelerate well on GPUs, unless $N$ is rather large (greater than 1k) (Govindaraju *et al* 2006, Sumanaweera and Liu 2005).

### *3.1. Backprojection viewing geometry*

We use the (reconstruction) volume coordinate system (VCS) described by axis vectors $(\boldsymbol{x}_v, \boldsymbol{y}_v, \boldsymbol{z}_v)$ as the reference coordinate system, with the volume centre at location (0,0,0) (see figure 2). In this VCS, a given detector image $P_\phi$ has been acquired in a source–detector pair coordinate system (DCS$_\phi$) described by axis vectors $(\boldsymbol{x}_\phi, \boldsymbol{y}_\phi, \boldsymbol{z}_\phi)$. Here $\boldsymbol{z}_\phi$ is orthogonal to the (flat) detector plane, the source is located at $\boldsymbol{s} = -d_\phi \boldsymbol{z}_\phi$ and the detector centre is located at $(D_\phi - d_\phi)\boldsymbol{z}_\phi$. A backprojection is the mapping of a voxel with VCS coordinates $\boldsymbol{r} = (r_x, r_y, r_z)$ onto the detector plane, yielding coordinates $P_\phi(X(\boldsymbol{r}), Y(\boldsymbol{r}))$. Here, $X(\boldsymbol{r})$ and $Y(\boldsymbol{r})$ are scaling functions from VCS coordinates into detector pixel coordinates. After the mapping, an interpolation operator Int( ) yields the backprojected voxel update $v_\phi(\boldsymbol{r})$, which is then depth-weighted according to the FDK equation:

$$
\begin{aligned}
v_\phi(\boldsymbol{r}) &= \frac{d_\phi^2}{(d_\phi + \boldsymbol{r}\cdot z_\phi)^2} \cdot \mathrm{Int}(P_\phi(X_\phi(\boldsymbol{r}), Y_\phi(\boldsymbol{r}))), \\
X(\boldsymbol{r}) &= \frac{\boldsymbol{r}\cdot x_\phi}{d_\phi + \boldsymbol{r}\cdot z_\phi} D_\phi, \qquad Y(\boldsymbol{r}) = \frac{\boldsymbol{r}\cdot y_\phi}{d_\phi + \boldsymbol{r}\cdot z_\phi} D_\phi.
\end{aligned}
\tag{1}
$$

This mapping can be conveniently expressed as a series of matrix operations. Since the mapping is perspective, we must use 4D (homogeneous) vectors and $4 \times 4$ matrices:

$$S \otimes T \otimes P \otimes M \otimes r = \vec{v}_h$$
$$\times \begin{bmatrix} \frac{w}{2} & 0 & 0 & 0 \\ 0 & \frac{h}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1.0 \\ 0 & 1 & 0 & 1.0 \\ 0 & 0 & 1 & 1.0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2D_\varphi}{w} & 0 & 0 & 0 \\ 0 & \frac{2D_\varphi}{h} & 0 & 0 \\ 0 & 0 & d/c & d/c \\ 0 & 0 & -1 & 0 \end{bmatrix}$$
$$\times \begin{bmatrix} x_\phi^x & x_\phi^y & x_\phi^z & -x_\phi \cdot s \\ y_\phi^x & y_\phi^y & y_\phi^z & -y_\phi \cdot s \\ z_\phi^x & z_\phi^y & z_\phi^z & -z_\phi \cdot s \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} = \begin{bmatrix} x_h \\ y_h \\ z_h \\ w_h \end{bmatrix}$$

$$D_\varphi = \frac{h}{2\tan(\theta_\varphi/2)} \qquad P_\varphi(X, Y) = \left( \frac{x_h}{w_h}, \frac{y_h}{w_h} \right). \tag{2}$$

Here, $\theta$ is the cone-angle and $d/c$ indicates terms that are not needed, since we do not require $z_h$. The model-view matrix $\boldsymbol{M}$ transforms a voxel coordinate $\boldsymbol{r}$ from the VCS into the DCS. Another $4 \times 4$ matrix, $\boldsymbol{P}$, determined by $D_\phi$, and detector dimensions $w$ and $h$ implements the subsequent perspective projection. $\boldsymbol{M}$ and $\boldsymbol{P}$ map $\boldsymbol{r}$ into a *canonical viewing space*, which is essentially a volume whose Cartesian coordinates are in $[-1, 1]$. The following two transformations, translation matrix $\boldsymbol{T}$ and scaling matrix $\boldsymbol{S}$, are determined by the detector size ($w$ and $h$) in pixels. Next comes the perspective divide, using the $w_h$ term of the resulting 4D vector. This produces the (floating point) coordinates $P_\phi(X(\boldsymbol{r}), Y(\boldsymbol{r}))$ in detector pixel space. After interpolating the detector image the FDK weighting is performed, re-using the $w_h$ term. This weight is essentially computed as $|z_\varphi \cdot r - z_\varphi \cdot s|$ representing the voxel's depth with respect to the source.

We note that this matrix only accommodates the case illustrated in figure 2, that is, the source–detector pair can rotate in any (non-circular) orbit and orientation, but the centre ray must pass through the rotation centre (here the volume origin) and it must be orthogonal to the detector plane. However, generalizations of this can be easily incorporated into the $\boldsymbol{P}$ matrix by implementing a general viewing frustum (for more details of this mapping see Foley *et al* (1990) and Segal *et al* (1992)). This can accommodate any type of instable gantry situation, which often occur in practice.

### *3.2. Accelerating backprojection on the GPU*

As mentioned above, there are two alternative approaches, AG-GPU and MP-GPU, with which CT reconstruction on the GPU can be performed. For both, we represent the target volume as an axis-aligned stack of 2D textures (single 3D textures currently do not support an efficient update mechanism). We first create a series of quadrilaterals $P_i$ (called *proxy polygons*) which define the location and spatial extent of each volume slice, and associate these with a set of 2D textures $T_i$ that are used to store the voxel values to be reconstructed (initialized to zero). Then, as figure 3 illustrates, for each such slice $T_i$, we view its host polygon $P_i$ face-on in orthographic viewing mode at slice resolution, which produces the fragments that are contained inside the quadrilateral and correspond to each slice voxel to be reconstructed. These are then processed in different ways in the two available configurations, MP-GPU and AG-GPU, to produce the detector plane mappings and the subsequent backprojected values. However, before we detail these two configurations, we first rewrite figure 1 in the form of a block diagram, in figure 4 (disregard the callouts for now). Here, both geometry and fragment stages have the capability

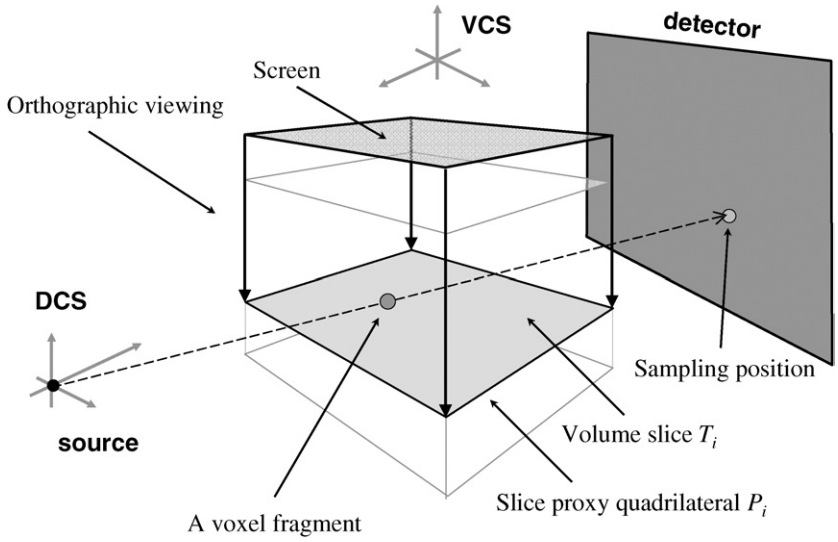


**Figure 3.** Backprojection of a volume slice on the GPU. An orthographic view (screen) onto the volume slice generates voxel fragments, each of which will then sample the detector (one fragment is shown here).

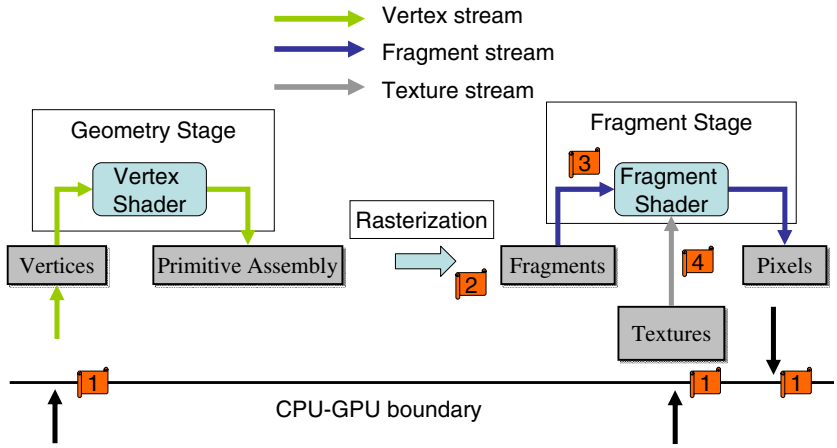


**Figure 4.** The GPU pipeline in block diagrams. Geometry, rasterization and fragment stages consist of primary engines that process three different data streams (vertex, fragment and texture). Four stages that might affect load-balancing and cause pipeline stall are marked as callouts.

of processing the graphics primitives streamed into the GPU pipeline, such as vertex, fragment and texture data. The following two sections detailing the two alternative configurations will then insert their specific shader tasks for backprojection into these blocks.

*3.2.1. Using the accelerated graphics pipeline (AG-GPU).* In the AG-GPU configuration both vertex and fragment shaders are used (see figure 5(a)). First the matrix $T_r = S \cdot T \cdot P \cdot M$ for the specific projection is compiled and loaded into the vertex shader. Then, for each slice, the vertices of its proxy polygon are passed into the vertex shader, and the subsequent

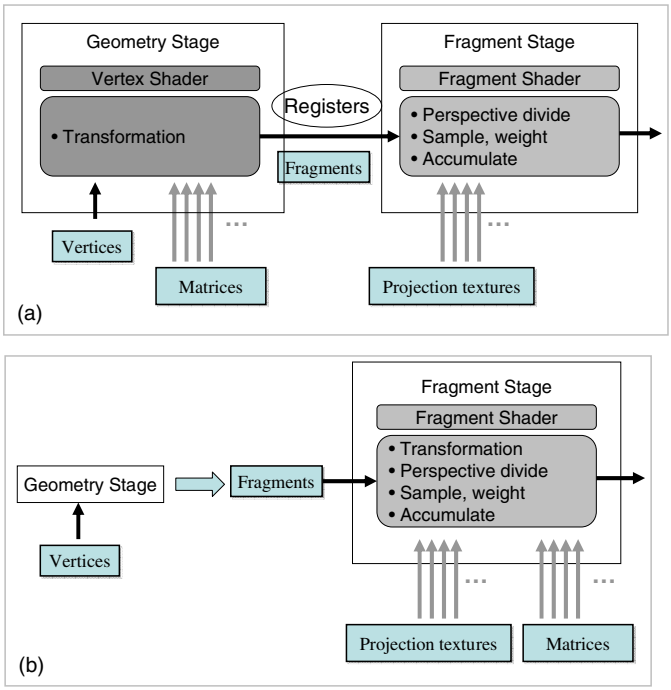


**Figure 5.** Two options for GPU-based accelerated CT reconstruction: (a) AG-GPU: accelerated graphics pipeline using both vertex and fragment engines; (b) MP-GPU: multi-processor configurations using fragment engine only.

transformation produces the mappings of the slice vertices into the detector plane (the 4D coordinate vector in equation (2)). The rasterizer, in turn, produces bilinear interpolations of these coordinates, one for each slice voxel (mapped to its fragment). These interpolated 4D coordinates are the correct mappings for the slice voxels, since the matrix $T_r$ originates in linear algebra and linear transformations. With each such fragment containing the vector $[x_h, y_h, z_h, w_h]$ for its slice voxel $\boldsymbol{r}$, the fragment program then performs the final perspective division. This produces the detector coordinates needed for the sampling of the (filtered) projection image, which is streamed in as a texture from the CPU. The sampling position usually does not coincide with the detector pixels and a sampling kernel needs to be applied to produce final values. Here, the method employed for this sampling is important. We use bilinear interpolation, which in fact runs nearly at the same speed as nearest-neighbour interpolation on the Nvidia 8800 GTX, but produces superior results, especially in cases where the projection resolution is close to the volume resolution. The last portion of the fragment program is the computation of the FDK depth-weighting factor $(d_\phi / w_h)^2$ and the result is then written to the texture accumulating the backprojections. Thus, the overall length of the fragment program is quite short: two divisions, two multiplications, one (hardwired) interpolation and one addition.

*3.2.2. Using the multi-processor configuration (MP-GPU).* In MP-GPU the generated fragments are processed in one uniform SIMD multi-processor of the GPU, most appropriately called the fragment shader (the vertex shader is not implemented). This mode is shown in figure 5(b). Here, the processors, and not the built-in hardware, must perform the matrix–vector multiplication for each slice voxel (represented by a fragment), in addition to the

other operations also performed in the AG-GPU configuration. These additional calculations amount to 12 multiplications and 9 additions, which requires nearly twice as many clock cycles than the shader program of the AG-GPU configuration.

### *3.3. Additional acceleration strategies*

A source for additional speedups is a GPU facility known as *early fragment-kill (EFK),* which can be exploited when the density range or the spatial extent of the target object is known *a priori*. In EFK a fragment is culled from the pipeline before it enters the fragment shader, thus causing near-zero computational overhead.

If the spatial extent of the object is known or the reconstruction can be limited to a region of interest, then the GPU stencil buffer can be set to a bit mask, which is tested in hardware by a corresponding stencil threshold during rendering. The outcome of the test then decides if the fragment is culled. Since the stencil buffer has 32 bits, we divide the volume slice stack into 32 sub-stacks, find the 2D stencil for each and store it in one of the 32 bits. This stencil mask is then loaded into the GPU at run-time.

In addition, if the desired density range of the object is known then the current density of a reconstructed slice voxel (normalized to the number of backprojections applied so far) can be used to determine if a subsequent fragment is passed into the fragment shader or not. If a voxel's value is outside the density range of interest (plus a reasonable margin), then this slice fragment (corresponding to a voxel) is guaranteed to fall outside the structure of interest (that is, it is outside the shadow of the previously applied projections) and can be safely rejected from the pipeline using EFK. This is a conservative rejection criterion, with 100% sensitivity, and the specificity increases (that is, more fragments end up rejected) as more projections are being applied, since these lead to a better object definition. EKF can be controlled by copying the current density volume into either the depth- or stencil buffer, and then setting the appropriate depth- or stencil thresholds according to the desired density range (for more details see, e.g., Neophytou and Mueller (2005)).

### *3.4. Load-balancing the pipeline*

For CPU-based programs, the use and scheduling of the cache as well as other lower-level memories plays a very important role in overall performance. Every cache (and memory) fault leads to a delay within the ongoing computation, and much work has been done to optimize memory access patterns for an abundance of diverse application scenarios. This situation is considerably more complex on GPUs since they consist of several components: (1) the cache and the main memory interface, but also (2) the active units embodied by the rasterizer and the two programmable, SIMD-parallel shaders. Thus, while CPU-based programs only need to minimize the waits for memory access and data delivery, GPUs must, in addition, also balance the data flow across its several active units.

In figure 4 we have marked with callout flags those GPU pipeline locations where the data flow may become compromised, that is, when the data production rate of the previous stage does not match the consumption rate of the upcoming stage. If it is greater, then data accumulation will occur, while if it is less, the upcoming stage stalls.

Our discussions so far were focused on updating one volume slice with one projection. This represents one rendering cycle (pass) on the GPU. A straightforward extension of this concept would then initialize and execute individual passes until every volume slice is updated, from every projection angle. However, we have found that instruction execution and texture sampling (callouts 3 and 4) cannot produce enough workload on each fragment generated from

the rasterization engine. Thus, fragments, streaming out of the rasterization engine, are being pooled into the multiple pixel pipelines, waiting to be processed in a shader (callouts 2, 3). Eliminating this bottleneck means we must produce more work per fragment, which can be facilitated by processing more than one projection per pass. This also reduces the number of passes which is beneficial since each pass requires time for setup, polygon rendering, fragment production, and texture writes (the slice updates).

To enable this scheme, we created a projection buffer on the GPU to temporarily store the acquired and filtered but not yet backprojected projections streamed in from the CPU. Here, the number of projections held in the buffer is controlled by a *window*, which we can dynamically adjust via the CPU-resident control process in order to optimize the load. When using this projection buffer, the first cycle must wait until the buffer has been filled, but for the remaining cycles the filling of the buffer with the next batch occurs while the previous batch is being processed, thus there will be no wait thereafter. In practice, when passing a batch of projection textures into the shader, we also transmit their acquisition geometry information in order to perform multiple backprojection samplings for a given slice. This multiplicity has already been indicated in figure 5 (by the multiple arrows). While the MP-GPU approach (figure 5(b)) can cope with arbitrary window sizes, the AG-GPU approach is limited by the number of floating point texture coordinate registers a rasterized fragment has available. These are needed to pass the interpolated texture coordinates and weights for each projection. The current hardware has a maximum of eight such registers, which, however, did not pose a limit for our application. More detailed experimental and benchmark data with regard to load-balancing our framework is available at our website (www.rapidCT.com).

Finally, larger target volumes exceeding the on-board memory are broken into blocks and the streaming pipeline is repeated for each such block. This is possible since voxels are completely independent during reconstruction. Immediately after reconstruction a block is then read out and stitched onto the adjoining blocks to assemble the complete volume. Also note that there is no space concern for large projections, since projections are deleted from GPU memory immediately after they have been applied (consumed) for reconstruction.

## 4. Results

We employed a 3D version of the Shepp–Logan phantom as well as a variety of medical CT datasets (a human head, human toes and a stented abdominal aorta) to test our framework. All experiments were performed on a 2.2 GHz dual-core Athlon PC with 1 GB RAM, equipped with a Nvidia Geforce 8800 GTX card with 768 MB on-board memory. The phantom projections were calculated analytically, while the medical projection data were obtained by performing high-quality x-ray simulations on existing CT volume datasets. All projections were acquired on a full circular orbit at a 15° cone angle. All projections assume a flat detector. The work (and the simulations) reported here has focused only on accelerating the backprojection operations—other adverse effects, such as noise and the data blurring caused by the revolving detector gantry have not been modelled or accounted for.

### *4.1. Reconstruction quality*

For the following experiments, we used 360 projections of size $512^2$ each to reconstruct a $512^3$ volume. Figure 6 shows slices from the reconstructed 3D Shepp–Logan phantom (at the original 0.5% contrast), obtained with our GPU framework as well as a traditional high-quality CPU implementation. Both used bilinear interpolation in the projection mapping, and a Shepp–Logan filter was employed for pre-filtering. We also show the intensity profile across

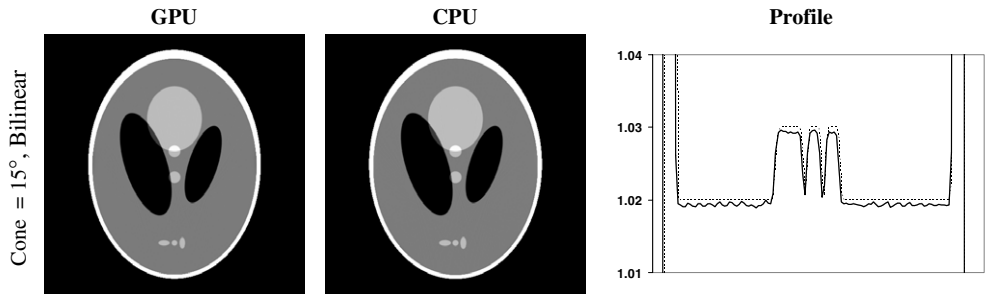


**Figure 6.** A slice of the 3D Shepp–Logan phantom, reconstructed with our streaming CT GPU-based framework (first column) and with a traditional CPU-based implementation (middle column). A windowed density range of [1.0, 1.04] is shown. The right column shows the line profiles across the three tumours near the bottom of the phantom (dashed lines: ground truth; solid grey lines: CPU results; solid black lines: GPU results). We observe that the GPU reconstructions are essentially identical to those computed on the CPU and that they represent the original phantom well.

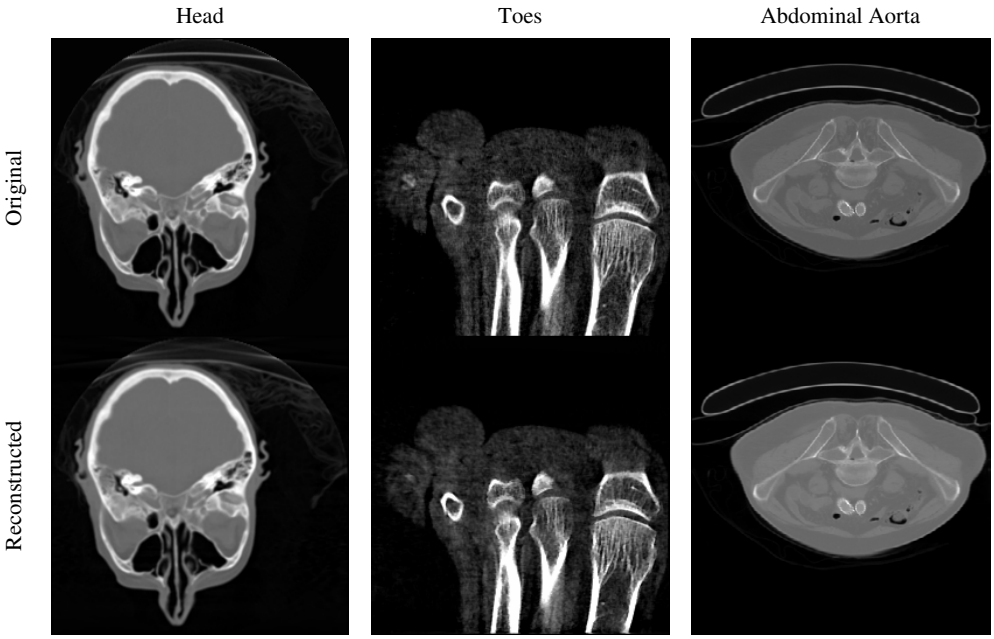


**Figure 7.** Slices of streaming CT reconstructions from simulated projection data of three representative medical volume datasets (left to right): a human head, human toes and a stented abdominal aorta. The slight blurring stems from the (minimal) low-passing induced by the resampling during simulation.

the small tumours near the bottom. Here, the slight shift comes from the non-exact FDK reconstruction of the off-centre slices (Kak and Slaney 1988, Turbell 2001). We observe that the CPU and GPU reconstructions are virtually identical (both for AP-GPU and MP-GPU).

Figure 7 compares slices reconstructed from the simulated projection data (top row) with the corresponding slices from the original volume dataset (bottom row). We observe that the quality of the reconstruction and the original is nearly identical. The very slight blurring most likely stems from the inherent low-passing in the resampling during the projection simulation.

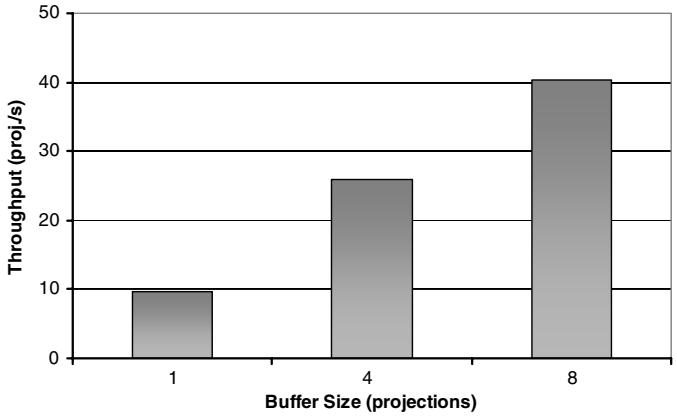


**Figure 8.** Streaming CT performance for a Feldkamp cone-beam reconstruction ($512^3$ volume, 360 projections, direct method, 32-bit floating point precision, bilinear sampling) using different buffer (window) sizes.

**Table 1.** Reconstruction speeds of various high-performance CT solutions. Timings have been normalized for a Feldkamp cone-beam CT reconstruction with 360 projections onto a volume grid of $512^3$ resolution (note not all implementations employ 32-bit floating point precision, bilinear interpolation and generalized source–detector positioning, which are all used for our streaming CT application).

| | Hardware platform | Mechanism | Time (s) | Projections $s^{-1}$ |
|---|---|---|---|---|
| Brasse *et al* | 1 dual-core Xeon CPU 2.6 GHz | | 3705 | 0.1 |
| | 12 dual-core Xeon CPU 2.6 GHz | | 236 | 1.6 |
| Goddard and Trepanier, Leeser *et al*, Li *et al* | FPGA | | 40.2–46.4 | 8–9 |
| Kachelrieß *et al* | CPU | Hybrid | 135 | 2.6 |
| | Cell BE | Direct (hybrid) | 19.1 (9.6) | 19 (37) |
| Streaming CT (this paper) | GPU Nvidia 8800 GTX | MP-GPU | 24.8 | 14.5 |
| | | AG-GPU | 8.9 | 40.4 |
| | | AG-GPU with EFK | 6.8 | 52.5 |

### *4.2. Reconstruction performance*

Table 1 compares the overall performance of our streaming CT framework with a selection of other current high-performance FDK-based CT reconstruction solutions reported in the literature. To enable a comparison, we scaled all of these to a currently common problem size, which we have also used for our own experiments: the reconstruction of a $512^3$ volume from 360 projections (the size of the projections is irrelevant, except for filtering, since the backprojection is mostly determined by the volume size). We also use the metric projections $s^{-1}$ to indicate the potential for real-time (streaming) reconstruction, currently requiring processing rates of 30–50 projections $s^{-1}$.

In table 1, the methods labelled 'direct' employ the full 3D projection matrix (equation (2)) when mapping a voxel onto the detector plane, allowing practical scanning situations in which the detector–source pair need not be confined to an aligned, perfectly circular orbit. In contrast, the method labelled 'hybrid' uses data that stem from resampling the acquired projections into

a virtual detector, which conforms to this ideal circular orbit. The backprojection is then performed into this arrangement, which reduces and simplifies the backprojection matrix and thus allows for faster voxel-projection mapping. All of our GPU solutions use the general, direct projection scheme.

We first observe that neither FPGA nor CPU-based solutions have reported processing rates of greater than 10 projections $s^{-1}$. On the other hand, our AG-GPU solution achieves the desired real-time projection throughput rate (40 projections $s^{-1}$)—however, the MP-GPU solution does not. Further, while the Cell BE hybrid solution is quite competitive to our standard AG-GPU method, the Cell BE direct method is only comparable to the MP-GPU solution. This match is understandable since the Cell BE is a multi-processor architecture, but without specific graphics hardware support.

Next, we show results using the EFK GPU-facility. Since this is clearly data-dependent, we only show the performance for the most frequent case, that is, when the reconstruction focus falls into the maximal spherical region covered by all projections. We observe that this can achieve a further speed-up of factor 1.3, which enables data acquisition rates of over 50 projections $s^{-1}$ or reconstructions of larger volumes at 30 projections $s^{-1}$.

Since it is difficult to fairly compare the (less complex) parallel-beam results of the previous GPU-accelerated implementations (mentioned in section 1) with those obtained in our direct cone-beam setting, we have not done this here. However, since all of these works have employed the less efficient MP-GPU paradigm, our results may also indicate less optimal performance for these application settings.

Finally, figure 8 graphs the effect of window size on reconstruction performance for the AG-GPU solution. We found that a window size of 8 yields the best results for our specific experiment setting, but it can be easily adjusted to fit others. We also see that load-balancing can have a dramatic positive effect, here a speedup of 4 (comparing the no-buffer case with the case when the projection buffer size is 8).

## 5. Conclusions

We have described the first continuous and buffer-free commodity-computing reconstruction pipeline for cone-beam CT. In our system, the projection data stream from the acquisition platform through a CPU-based filtering stage into a load-balanced GPU-accelerated backprojection framework. Our streaming CT can reconstruct a $512^3$ volume at a rate of 40 to over 50 $512^2$ and $1024^2$ projections $s^{-1}$, which is the current production rate of commodity flat-panel detectors, and beyond. Larger volumes could be easily accommodated by using the now available single-platform dual- and quad-GPU setups, which provide up to 4GB of memory. Our pipeline provides a throughput rate and reconstruction speed of one to two orders of magnitude higher than existing systems based on commodity (PC) hardware. It is also faster than less readily available, but more costly PC-resident high-performance platforms based on the Cell BE processor and FPGA technologies. We achieve this by (i) exploiting many of the GPU-resident graphics facilities and (ii) careful load-balancing of the various GPU pipeline components in light of the specific computing task of CT reconstruction. Our rapid real-time reconstruction pipeline enables interactive use of commodity detectors and gantries, allowing, for example, interactive monitoring of musculoskeletal systems for positioning in interventional procedures as well as applications in image-guided surgery or radiotherapy. In fact, since the projection throughput is higher than their production rate on common detector hardware, it would even be possible to interject a 3D visualization rendering cycle into the reconstruction computation.

Our results indicate that for the reconstruction settings tested here, a window size of 8 produced the best speedups, and along with it, the most optimal memory bandwidth and instruction execution patterns. This may change with different reconstruction scenarios, and this could be easily corrected for by adjusting the window size dynamically in an automatic binary-search optimization scheme, taking into account reconstructions just acquired. For the EFK scheme the optimal window size may also exhibit a more dynamic behaviour. Here one could start with smaller windows when all fragments still get rendered and then increase the window size for later (sparser) active projection fragments sets.

## Acknowledgments

This work was partially funded by NIH grant R21 EB004099-01 and the Keck Advanced Microscopy Laboratory, University of California at San Francisco.

## References

Andersen A and Kak A 1984 Simultaneous algebraic reconstruction technique (SART): a superior implementation of the ART algorithm *Ultrason. Imaging* **6** 81–94

Brasse D, Humbert B, Mathelin C, Rio M and Guyonnet J 2005 Towards an inline reconstruction architecture for micro-CT systems *Phys. Med. Biol.* **50** 5799–811

Cabral B, Cam N and Foran J 1994 Accelerated volume rendering and tomographic reconstruction using texture mapping hardware *VVS '94: Proc. 1994 Symp. on Volume Visualization (Tysons Corner, VA)* (New York: ACM Press) pp 91–8

Chidlow K and Möller T 2003 Rapid emission tomography reconstruction *Vol. Graph.* pp 15–26

Fahrig R, Fox A J, Lownie S and Holdsworth D W 1997 Use of a C-arm system to generate true three-dimensional computed rotational angiograms: preliminary in vitro and in vivo results *Am. J. Neuroradiol.* **18** 1507–14

Feldkamp L, Davis L and Kress J 1984 Practical cone beam algorithm *J. Opt. Soc. Am.* A **1** 612–9

Foley J, van Dam A, Feiner S and Hughes J 1990 *Computer Graphics: Principles and Practice* (New York: Addison-Wesley)

Frigo M and Johnson S G 2005 The design and implementation of FFTW3 *Proc. IEEE* **93** 216–31

Goddard I and Trepanier M 2002 High-speed cone-beam reconstruction: an embedded systems approach *SPIE Med. Imaging Conf.* **4681** 483–91

Govindaraju N, Larsen S, Gray J and Manocha D 2006 A memory model for scientific algorithms on graphics processors *UNC Technical Report* available at http://gamma.cs.unc.edu/GPUFFTW

Hudson H and Larkin R 1994 Accelerated image reconstruction using ordered subsets of projection data *IEEE Trans. Med. Imaging* **13** 601–9

Jaffray D A, Siewerdsen J H, Wong J W and Martinez A A 2002 Flat-panel cone-beam computed tomography for image-guided radiation therapy *Int. J. Radiat. Oncol. Biol. Phys.* **53** 1337–49

Kachelrieß M, Knaup M and Bockenbach O 2006 Hyperfast perspective cone-beam backprojection *IEEE Med. Imaging Conf. Rec.*

Kak A C and Slaney M 1988 *Principles of Computerized Tomographic Imaging* (Piscataway, NJ: IEEE)

Kole J S and Beekman F J 2006 Evaluation of accelerated iterative x-ray CT image reconstruction using floating point graphics hardware *Phys. Med. Biol.* **51** 875–89

Kondo C, Mori S, Endo M, Kusakabe K, Suzuki N, Hattori A and Kusakabe M 2005 Real-time volumetric imaging of human heart without electrocardiographic gating by 256-detector row computed tomography: initial experience *J. Comput. Assist. Tomogr.* **29** 694–8

Leeser M, Coric S, Miller E, Yu H and Trepanier M 2002 Parallel-beam backprojection: an FPGA implementation optimized for medical imaging *Proc. 10th Int. Symp. on FPGA* pp 217–26

Li J, Papachristou C and Shekhar R 2005 An FPGA-based computing platform for real-time 3D medical imaging and its application to cone-beam CT reconstruction *J. Imaging Sci. Technol.* **49** 237–45

Liu X, Defrise M, Desbar L and Fleute M 2001 Cone-beam reconstruction for a C-arm CT system *IEEE Nucl. Sci. Symp.* **3** 1489–93

Mori S, Endo M, Tsunoo T, Murase K and Fujiwara H 2004 Evaluation of weighted FDK algorithms applied to four-dimensional CT (4D-CT) *IEEE Nucl. Sci. Symp.* **5** 3243–5

Mueller K and Xu F 2006 Practical considerations for GPU-accelerated CT *IEEE Int. Symp. Biomed. Imaging* 1184–7

Mueller K and Yagel R 2000 Rapid 3-D cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2-D texture mapping hardware *IEEE Trans. Med. Imaging* **19** 1227–37

Neophytou N and Mueller K 2005 GPU accelerated image aligned splatting *Vol. Graph. 2005* pp 197–205

Schiwietz T, Chang T, Speier P and Westermann R 2006 MR image reconstruction using the GPU *Proc. SPIE* **6142** 1279–90

Segal M, Korobkin C, van Widenfelt R, Foran J and Haeberli P 1992 Fast shadows and lighting effects using texture mapping *Proc. Siggraph*. 249–52

Sumanaweera T and Liu D 2005 Medical image reconstruction with the FFT *GPU Gems II* (Reading, MA: Addison-Wesley) pp 765–84

Taguchi K 2003 Temporal resolution and the evaluation of candidate algorithms for four-dimensional CT *Med. Phys.* **30** 640–50

Turbell H 2001 Cone-beam reconstruction using filtered backprojection *PhD Dissertation* Linköping University, Sweden

Verlaan J-J, van de Kraats E B, Dhert W J A and Oner F C 2005 The role of 3-D rotational x-ray imaging in spinal trauma *Injury* **36** S98–103

Xu F and Mueller K 2005 Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware *IEEE Trans. Nucl. Sci.* **52** 654–63

Xue X, Cheryauka A and Tubbs D 2006 Acceleration of fluoro-CT reconstruction for a mobile C-Arm on GPU and FPGA hardware: a simulation study *Proc. SPIE* **6142** 1494–501