RGBA Packing for Fast Cone Beam Reconstruction on the GPU

Fumihiko Ino[a], Seiji Yoshida[a], and Kenichi Hagihara[a]

[a]Graduate School of Information Science and Technology, Osaka University,
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

ABSTRACT

This paper presents a fast cone beam reconstruction method accelerated on the graphics processing unit (GPU). We implement the Feldkamp, Davis, and Kress (FDK) algorithm on the OpenGL graphics pipeline, which allows us to exploit the full resources and capabilities available on the GPU. The proposed method differs from previous GPU-based methods in having an RGBA packing scheme capable of directly dealing with projections without rebinning. It also reduces the amount of computation by using a data reuse scheme, which is useful to save the memory bandwidth for this memory-intensive problem. Both schemes contribute to reduce the number of rendering passes, namely the number of kernel invocations on the GPU, realizing fast cone beam reconstruction. We show some experimental results obtained on a desktop PC with an nVIDIA GeForce 8800 GTX card. As a result, the proposed method takes 8.1 seconds to reconstruct a 512^3-voxel volume from 360 512^2-pixel projection images. This execution time is equivalent to a 15.6-fold speedup over a CPU implementation, showing 10% higher performance as compared with a previous OpenGL-based method that requires the single-slice rebinning of projections for acceleration. With respect to non-rebinned data, our method provides approximately three times higher performance than the previous method.

Keywords: RECON, CTCB, ALG, GPU, high performance computing

1. INTRODUCTION

Tomographic reconstruction[1] is the process of reconstructing three-dimensional (3-D) objects from their integral projections. In general, cone beam X-ray computed tomography (CT) scanners are required to reconstruct volume data within a short time, because they usually assist surgical procedures. However, cone beam reconstruction is a compute- and memory-intensive problem, which takes $O(IN^3)$ time, where N represents the volume size and I represents the number of projections obtained during a scan. Although optimized CPU implementations that use SSE instructions[2] can complete a reconstruction task in approximately two minutes,[3] it is not sufficient to produce a volume at an interactive rate. Therefore, we need some acceleration techniques to develop high-speed imaging systems capable of producing a volume immediately after a scan.

One acceleration technique is to use accelerators such as the graphics processing unit (GPU),[4–7] Cell Broadband Engine,[3] and field programmable gate array (FPGA).[8] To the best of our knowledge, a GPU-based method[4] demonstrates the highest performance of 8.9 seconds on an nVIDIA GeForce 8800 GTX card. The GPU[9] here is originally specialized to accelerate graphics tasks, but this commodity chip is rapidly increasing performance with more flexible programmability, allowing us to accelerate non-graphics tasks for interactive use. Although the fastest performance is demonstrated by the GPU, the previous method[4] requires the single-slice rebinning of projections. If this requirement is not satisfied, it takes 2–3 times longer time for reconstruction.[4] Furthermore, the single-slice rebinning causes artifacts if applied to real projections obtained by a cone beam CT scanner. Such a rebinning procedure can be applied to phantom data, where the cone angle can be easily set as almost zero degree during projection, but not to real data, where the cone beam penetrates multiple slices due to a wide cone angle.

Further author information: (Send correspondence to Fumihiko Ino)
Fumihiko Ino: E-mail: ino@ist.osaka-u.ac.jp

In this paper, we present a GPU-accelerated cone beam reconstruction method, aiming at developing high-speed cone beam CT scanners for intraoperative imaging. Our OpenGL-based method accelerates a widely-used filtered backprojection algorithm, namely the Feldkamp, Davis, and Kress (FDK) method,[1] using both programmable processing units and hardwired components on the GPU. Our method has two main contributions as compared with previous GPU-based methods. The first one is an RGBA packing scheme that realizes fast backprojection by packing four scalar data into an RGBA data. As compared with a previous method,[4] our scheme does not require rebinning before reconstruction. The proposed method makes the packing scheme more flexible with a 2-fold speedup over an unpacked implementation. The other one is a data reuse scheme that omits computation by sharing computational results between different voxels in the volume. This data reuse scheme reduces the computational amount to achieve further acceleration. The two schemes mentioned above reduce the number of rendering passes to 1/16, minimizing the CPU overhead such as texture switching needed at every rendering pass.

The paper is structured as follows. Section 2 presents related work and Section 3 summarizes the graphics pipeline in the GPU. Section 4 describes our method with the acceleration schemes. Section 5 shows experimental results obtained using a commodity graphics card. Finally, Section 6 concludes the paper with future direction.

2. RELATED WORK

To the best of our knowledge, Cabral et al.[10] firstly present a GPU-based method that implements the FDK algorithm using texture mapping hardware. Mueller et al.[4,5] also implement the FDK algorithm on the GPU. They propose two methods,[5] called MP-GPU and AG-GPU, which focus on programmable processing units of the GPU, such as vertex processors (VPs) and fragment processors (FPs). MP-GPU uses only FPs while AG-GPU uses both FPs and VPs to achieve load balancing[11] between processing units. AG-GPU improves the performance of MP-GPU from 14.5 projections per second (pps) to 40.4 pps. They also show that early fragment kill (EFK) is useful to reduce the reconstruction time if region of interest (ROI) can be specified before reconstruction.

With respect to RGBA packing, two schemes have been proposed in the past. The first scheme[12] packs four orthogonal projections into a single RGBA projection. This scheme can be easily applied to phantom data but it is not easy for real scanners to obtain such precise data due to the mechanical limitation. The other scheme[4] packs adjacent four volume slices into a single RGBA slice. However, it cannot be directly applied to real data, where the cone beam penetrates multiple slices due to a wide cone angle.

Schiwietz et al.[7] propose a reconstruction pipeline running on the GPU. Riabkov et al.[6] show that there are two different policies with respect to data assignment of the video memory. However, these methods are not faster than AG-GPU.

Li et al.[13] use compute unified device architecture (CUDA)[14] to implement the FDK algorithm on the nVIDIA GPU. Their method takes 3.6 seconds to perform backprojection (without filtering of projections). One drawback of CUDA-based methods is that they cannot be used with optimization techniques for graphics applications, such as EFK.

3. GRAPHICS PIPELINE

Figure 1 shows an overview of the graphics pipeline in the GPU.[9] This pipeline has two programmable processing units, namely VPs and FPs, which are interconnected by the rasterizer. Each unit can access data stored on the video memory. Given vertex data that represents the shape and location of objects, the pipeline outputs pixels of rendered images. Off-screen rendering techniques[15] can be used to perform general purpose computation on the GPU.

In the pipeline mechanism mentioned above, VPs are responsible for processing vertex data. Given vertex data from VPs, the rasterizer generates fragments from the region specified by the vertex data. Fragment data here corresponds to pixel data and has color and coordinate information linearly interpolated from the vertex data. That is, the rasterizer converts per-vertex data into per-fragment data. The remaining programmable units, namely FPs, take the responsibility for processing fragment data in parallel. The vertex program and the fragment program must be written as per-vertex operations and per-fragment operations, respectively.

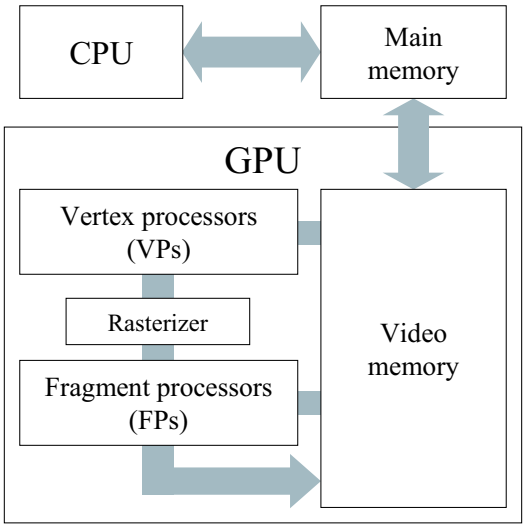


Figure 1. Graphics pipeline in the GPU.

As we mentioned earlier, fragment data has interpolated values of vertex data, so that linear operations for fragments can be replaced with operations for vertices. That is, some linear operations in the fragment program can be moved to the vertex program. This code motion[11] achieves load balancing between processing units. It also reduces the time complexity if the number of vertices is smaller than that of fragments.

Note here that EFK is an optimization technique that limits the drawing region to reduce the number of fragments passed from the rasterizer to FPs. Since FPs invoke the fragment program for every fragment, the effects of EFK increases with the number of eliminated fragments. This reduction contributes to less amount of computation as well as less amount of data transfer between the video memory and FPs.

The GPU is allowed to access data only on the video memory. Therefore, the data must be downloaded from the main memory to the video memory in advance of rendering. In contrast, the data must be readback to the main memory if the CPU requires results computed by the GPU.

A typical implementation strategy for GPGPU applications is to store the input/output data as two-dimensional (2-D) textures on the video memory such that texels on output textures can be computed in parallel. Since the GPU is optimized for 2-D textures, it is better to convert 3-D data into 2-D textures. RGBA textures can have four-component data as a single texel data. Each component here can have a 32-bit floating point number.

4. METHODS

The FDK algorithm[1] consists of two processing stages: the filtering stage and the backprojection stage. Our method accelerates both stages on the GPU. However, we describe here only the backprojection stage, which is the performance bottleneck of this algorithm.

Figure 2 illustrates the coordinate system used in our method. There is the volume F and a projection Q_i taken from the X-ray source at distance d' with angle θ_i, where $1 \leq i \leq I$. Notation d_i represents the distance between the X-ray source and the volume origin. Let $F(x, y, z)$ be the value of voxel (x, y, z) in the volume F. Given I filtered projections $Q_1, Q_2, \ldots, Q_I$, the backprojection stage generates the volume F according to the following equation:

$$F(x, y, z) = \frac{1}{2\pi I} \sum_{i=1}^{I} W_2(x, y, i) Q_i(u(x, y, i), v(x, y, z, i)), \tag{1}$$

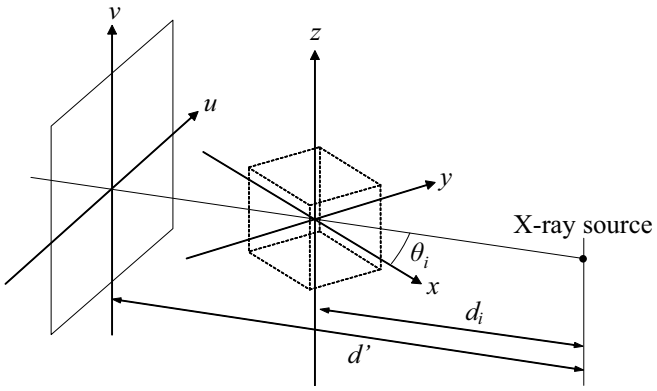
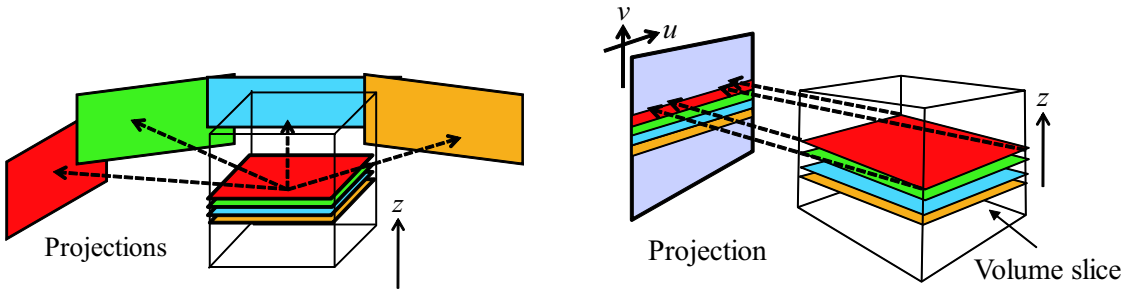


Figure 2. Coordinate system used in our method.



(a) (b)

Figure 3. (a) RGBA packing scheme used in our method. Our scheme packs four filtered projections into a single RGBA texture. (b) Previous scheme[4] packs four succeeding pixels along the v-axis into a single RGBA pixel. Therefore, it requires the single-slice rebinning of projections and assumes that projections have the same resolution as the volume. Both schemes also pack four succeeding volume slices perpendicular to the z-axis.

where

$$u(x,y,i) = \frac{d'(-x\sin\theta_i + y\cos\theta_i)}{d_i - x\cos\theta_i - y\sin\theta_i}, \tag{2}$$

$$v(x,y,z,i) = \frac{d'z}{d_i - x\cos\theta_i - y\sin\theta_i}, \tag{3}$$

$$W_2(x,y,i) = \left(\frac{d_i}{d_i - x\cos\theta_i - y\sin\theta_i}\right)^2. \tag{4}$$

Since the GPU architecture is optimized to 2-D data structure, a naive method decomposes the volume F into slices $F_0, F_1, \ldots, F_{N-1}$, each stored in a 2-D texture. Similarly, each of projections is stored in a 2-D texture. In this case, we have to render IN times to backproject I projections into N slices. The proposed method is designed to reduce this iteration number to $IN/16$ by packing input/output textures.

4.1 RGBA Packing Scheme

There is no data dependence between backprojection of an image and that of another image. Therefore, we can simultaneously process the backprojection of multiple images. To do this, we pack four filtered images Q_i, Q_{i+1}, Q_{i+2}, and Q_{i+3} into a single RGBA texture. Figure 3(a) shows how our RGBA packing scheme works at the backprojection stage. In our scheme, an RGBA texel (u, v) in the packed texture contains scalar texels at the same coordinates: $Q_i(u,v)$, $Q_{i+1}(u,v)$, $Q_{i+2}(u,v)$, and $Q_{i+3}(u,v)$. This allows us to utilize vector operations to accelerate computation, as shown in Fig. 4. For example, we can vectorize the computation of u, v, and W_2 for packed texels $Q_i(u,v)$, $Q_{i+1}(u,v)$, $Q_{i+2}(u,v)$, and $Q_{i+3}(u,v)$ at lines 18 and 20 in Fig. 4. That is, a vector of $u(x,y,i), \ldots, u(x,y,i+3)$ and that of $W_2(x,y,i), \ldots, W_2(x,y,i+3)$ are computed at

```
Input: Projections P_1, P_2, ..., P_I and parameters d', d_1, d_2, ..., d_I, θ_1, θ_2, ..., θ_I
Output: Volume slices F_0, F_1, ..., F_{N-1}
 1: ProposedReconstruction()
 2: begin
 3:   Download projections P_1, ..., P_I;
 4:   (Q_1, ..., Q_I) ← Filtering(P_1, ..., P_I);
 5:   for n = 0 to N/4 − 1 do begin
 6:     z = 4n;
 7:     for m = 0 to I/4 − 1 do begin
 8:       i = 4m + 1;
 9:       (F_z, F_{z+1}, F_{z+2}, F_{z+3})
            ← Backprojection((Q_i, Q_{i+1}, Q_{i+2}, Q_{i+3}), (d_i, d_{i+1}, d_{i+2}, d_{i+3}),
                 (θ_i, θ_{i+1}, θ_{i+2}, θ_{i+3}), d', z); // Rendering
10:     end;
11:     Readback volume slices F_z, ..., F_{z+3};
12:   end;
13: end
14: function Backprojection((Q_i, Q_{i+1}, Q_{i+2}, Q_{i+3}), d, θ, d', z)
15: begin
16:   // x and y is given by the GPU
17:   Compute d − x cos θ − y sin θ;
18:   Compute u and W_2; // Eqs. (2) and (4)
19:   for n = 0 to 3 do begin
20:     Compute v with z = z + n; // Eq. (3)
21:     Convert u and v to texture coordinates coord_1, ..., coord_4;
22:     for j = 1 to 4 do begin
23:       val_j ← Interpolated texel at coord_j in Q_{i+j}; // bilinear interpolation
24:     end;
25:     val ← W_2 · val;
26:     out_{n+1} ← val_1 + val_2 + val_3 + val_4;
27:   end;
28:   current ← (F_z(x, y), F_{z+1}(x, y), F_{z+2}(x, y), F_{z+3}(x, y));
29:   return out + current;
30: end
```

Figure 4. Pseoudocode of the proposed method. An RGBA value that contains four scalar values $a_1, \ldots, a_4$ is denoted as $\boldsymbol{a}$ in bold font. The number of rendering passes is reduced from IN to $IN/16$ in our method. A hardware accelerated interpolation mechanism is employed to obtain a texel at line 23.

a time. Similarly, we can apply vectorization to computation of $f_i(x, y, z), \ldots, f_{i+3}(x, y, z)$ at line 25, where $f_i(x, y, z) = W_2(x, y, i)Q_i(u(x, y, i), v(x, y, z, i))$.

The RGBA packing scheme allows us to backproject four projections per rendering pass. Therefore, we can reduce the number of rendering passes by 75%. These fewer passes contribute to reduce the CPU overhead, such as texture switching incurred before each rendering.

4.2 Data Reuse Scheme

Equations (2)–(4) indicate that some terms are independent of z: $u(x, y, i)$, $W_2(x, y, i)$, and the denominator of $v(x, y, z, i)$. Therefore, such z-independent terms must be computed for every line (x, y) rather than every voxel (x, y, z). On the other hand, our method can process every four slices of the volume at a time if the slices are packed into an RGBA texture. Therefore, as we did for the input texture, we also pack the output texture in order to reuse the z-independent terms within four volume slices.

The data reuse scheme further reduces the number of rendering passes, because we have 75% fewer output textures. Thus, we have to invoke the rendering kernel (fragment program) only $IN/16$ times, which is shown by the loops at lines 5–7 in Fig. 4. In contrast, we have more iterations at lines 19 and 22 in the invoked kernel. This means that we have more computation per rendering pass. Such a coarse-grained rendering task leads to a

Table 1. Implementations used for experiments.

Implementation	Data reuse	RGBA packing	Load balancing[5,11]
A	No	No	No
B	Yes	No	No
C	No	Yes	No
D	Yes	Yes	No
E	Yes	Yes	Yes

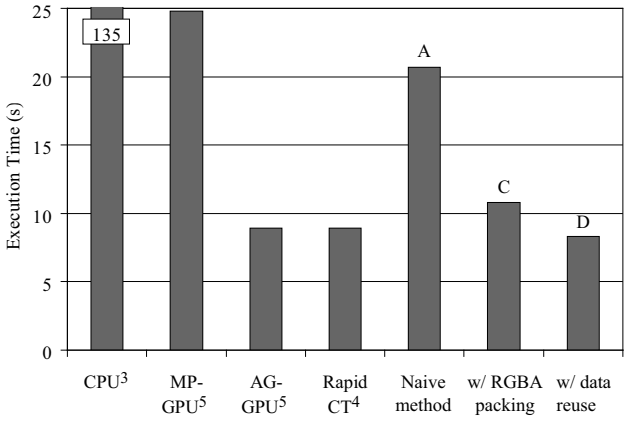


Figure 5. Execution time for reconstruction of 512^3-voxel volume from 360 512^2-pixel projections. The right three bars correspond to our methods and the remaining bars represent previous methods.[3–5] The results for previous methods are quoted from their papers, because it is measured on the same GPU.

higher efficiency because each rendering task is parallelized by the GPU. In particular, it is useful to improve the efficiency if the kernel takes a relatively shorter time than its preprocessing overhead such as texture switching.

5. RESULTS

To evaluate the proposed method in terms of performance and image quality, we have implemented the method using the C++ language, the OpenGL library, and the Cg toolkit. We also have implemented different versions of the proposed method to analyze the effect of the data reuse scheme, the RGBA packing scheme, and the load balancing scheme.[5,11] Table 1 summarizes the differences between implementations.

We perform experiments using a desktop computer equipped with a Core 2 Extreme CPU running at 2.93 GHz clock speed. This computer has an nVIDIA GeForce 8800 GTX card, running on driver version 162.01. We use Windows XP for the operation system. Reconstruction is done for the Shepp-Logan phantom and clinical brain data. The output volume is stored in frame buffer object (FBO)[15] to perform off-screen rendering.

Figure 5 shows the timing results. With respect to our method, the execution time contains times for RGBA packing, download, filtering, backprojection, readback, and RGBA unpacking, as shown in Table 2. As a result, the proposed method takes 8.1 seconds to reconstruct a 512^3-voxel volume from 360 512^2-pixel projections. This execution time is equivalent to a 15.6-fold speedup over an SSE-optimized CPU implementation.[3] It also achieves 10% higher performance as compared with the fastest previous method.[4] Thus, the proposed method is useful to produce volume data immediately after a scan of real objects.

We also analyze the effectiveness of our method in terms of the memory bandwidth and the floating point operations per second (FLOPS). Table 3 shows that the effective bandwidth reaches 70.8 GB/s out of 86.4 GB/s. With respect to the floating point performance, it provides 231.6 GFLOPS out of 518.4 GFLOPS. Thus, the performance is limited by the memory bandwidth. Therefore, we have to reduce the amount of data being fetched from video memory in order to achieve further acceleration. Such a reduction can be realized by EFK,[5] which is available in the graphics pipeline but not available in CUDA. Actually, we observe that EFK further reduces the reconstruction time to 7 seconds if 75% of volume region is skipped during backprojection.

Table 2. Breakdown of execution time (s). Total time here includes the synchronization overhead between the CPU and the GPU. Such an overhead incurs only when obtaining breakdown information.

Breakdown	Implementation				
	A	B	C	D	E
RGBA packing	0.3	0.3	0.6	0.6	0.6
Download	0.4	0.4	0.5	0.5	0.5
Filtering	5.6	5.6	2.5	2.4	2.4
Backprojection	16.3	7.4	6.8	3.7	4.0
Readback	0.5	0.4	0.5	0.4	0.4
RGBA unpacking	1.0	0.9	0.9	0.6	0.6
Others	3.2	4.0	1.3	1.4	1.9
Total time	27.2	18.9	13.0	9.6	10.4
Total time w/o synchronization	20.7	17.0	10.8	8.3	8.1

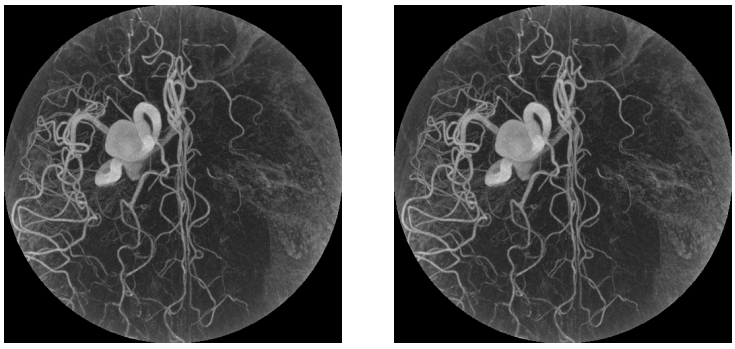
Table 3. Effective performance and bandwidth of each implementation. Note that implementation E reduces the time complexity by load balancing. This slightly reduces the reconstruction time but decreases the efficiency in terms of floating point performance.

Metrics		Effective performance					Theoretical performance
		A	B	C	D	E	
Computation (GFLOPS)	Processing unit	117.0	112.0	62.6	99.3	61.3	345.6
	Texture unit	33.9	69.4	70.6	132.3	128.3	172.8
	Total	150.9	181.5	133.1	231.6	189.6	518.4
Memory bandwidth (GB/s)		28.7	37.1	59.8	70.8	68.6	86.4

With respect to the image quality, GPU-based methods cannot reconstruct volumes identical to those generated by CPU-based methods. This is due to the fact that the GPU has a different architecture that does not provide the same instruction set as the CPU. For example, a multiply-add operation is executed by a single instruction on the GPU while it is executed by two instructions on the CPU. In addition, the GPU has larger errors than the CPU. Despite of these differences, we find that this problem is not critical in practical situations. Actually, there is no visible difference in visualization results, as shown in Figs. 6(a) and (b). In addition, peak signal-to-noise ratio (PSNR) values of the raw volume are more than 100 dB, which indicates that it is hard to see image degradation.

6. CONCLUSION

We have presented a GPU-accelerated reconstruction method for cone beam CT scanners. Our method makes it possible to pack projections into RGBA data without rebinning. It also reuses data to reduce the amount of computation and to save the memory bandwidth for this memory-intensive problem. Both RGBA packing



(a) (b)

Figure 6. Maximum intensity projection (MIP) of the brain data generated (a) on the GPU and (b) on the CPU.

and data reuse schemes contribute to minimize the CPU overhead, such as texture switching needed at every rendering pass.

As a result, the proposed method takes 8.1 seconds to reconstruct 512^3-voxel volume, which is 10% faster than a previous method. In addition, our method does not assume rebinned data. With respect to the efficiency, the effective bandwidth reaches a peak of 70.8 GB/s, which is 82% of the theoretical bandwidth. Therefore, we think that further acceleration will be achieved if reducing the data amount being fetched from the video memory.

Future work includes the support of large volume such as 1024^3 voxel data. Some data decomposition techniques are needed to deal with such large size on current graphics cards.

ACKNOWLEDGMENTS

This work was partly supported by JSPS Grant-in-Aid for Scientific Research (A)(2)(20240002), Young Researchers (B)(19700061), and the Global COE Program "in silico medicine" at Osaka University. We are also grateful to Dr. Kazuyoshi Nishino, Dr. Yoshihiro Mukuta, and Dr. Yukio Mishina from Shimadzu for their valuable discussions as well as clinical data.

REFERENCES

[1] Feldkamp, L. A., Davis, L. C., and Kress, J. W., "Practical cone-beam algorithm," *J. Optical Society of America* **1**, 612–619 (June 1984).

[2] Klimovitski, A., "Using SSE and SSE2: Misconceptions and reality," in [*Intel Developer Update Magazine*], (Mar. 2001).

[3] Kachelrieß, M., Knaup, M., and Bockenbach, O., "Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware," *Medical Physics* **34**, 1474–1486 (Apr. 2007).

[4] Mueller, K., Xu, F., and Nephytou, N., "Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography?," in [*Proc. SPIE Medical Imaging 2007*], (Feb. 2007).

[5] Xu, F. and Mueller, K., "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Physics in Medicine and Biology* **52**, 3405–3419 (June 2007).

[6] Riabkov, D., Xue, X., Tubbs, D., and Cheryauka, A., "Accelerated cone-beam backprojection using GPU-CPU hardware," in [*Proc. 9th Int'l Meeting Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine (Fully 3D '07)*], 68–71 (July 2007).

[7] Schiwietz, T., Bose, S., Maltz, J., and Westermann, R., "A fast and high-quality cone beam reconstruction pipeline using the GPU," in [*Proc. SPIE Medical Imaging 2007*], 1279–1290 (Feb. 2007).

[8] Gac, N., Mancini, S., and Desvignes, M., "Hardware/software 2D-3D backprojection on a SoPC platform," in [*Proc. 21st ACM Symp. Applied Computing (SAC'06)*], 222–228 (Apr. 2006).

[9] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. J., "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum* **26**, 80–113 (Mar. 2007).

[10] Cabral, B., Cam, N., and Foran, J., "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware," in [*Proc. 1st Symp. Volume Visualization (VVS'94)*], 91–98 (Oct. 1994).

[11] Ikeda, T., Ino, F., and Hagihara, K., "A code motion technique for accelerating general-purpose computation on the GPU," in [*Proc. 20th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS'06)*], (Apr. 2006). 10 pages (CD-ROM).

[12] Xu, F. and Mueller, K., "Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware," *IEEE Trans. Nuclear Science* **52**, 654–663 (June 2005).

[13] Li, M., Yang, H., Koizumi, K., and Kudo, H., "Fast cone-beam CT reconstruction using CUDA architecture," *Medical Imaging Technology* **25**, 243–250 (Sept. 2007). (In Japanese).

[14] nVIDIA Corporation, "CUDA Programming Guide Version 2.0," (July 2008).

[15] OpenGL Extension Registry, "Gl_ext_framebuffer_object," (2006). `http://oss.sgi.com/projects/ogl-sample/registry/EXT/framebuffer_object.txt`.