Sensors & Transducers, Vol. 21, Special Issue, May 2013, pp. 128-134

www.sensorsportal.com

Sensors & Transducers

http://www.sensorsportal.com

Optimization of Cone Beam CT Reconstruction Algorithm Based on CUDA

[1] Wang LI-Fang, [2] Zhang Shu-Hai

[1] School of Electronics and Computer Science Technology, North University of China
Taiyuan 030051, China

[2] School of Chemical Engineering and Environment, North University of China
Taiyuan 030051, China

E-mail: wanglifang2013@yeah.net

Received: 23 March 2013 /Accepted: 14 May 2013 /Published: 30 May 2013

Abstract: In CBCT, image reconstruction is difficult to meet the requirements of the user real-time reconstruction because data amount of image reconstruction is large, operation complexity is high and the time of reconstruction is long. CUDA based on GPU launched by NVIDIA is very suitable for large-scale parallel computing problem. This paper optimizes cone beam CT reconstruction algorithm by CUDA and improves the speed of weighted back-projection and filtering, and shortens the data access time by using the texture memory and constant memory in CUDA to respectively store the kernel function and the filtered data. The experimental results show that the reconstruction speed and the reconstruction quality are obviously improved compared with the reconstruction method based on GPU. *Copyright © 2013 IFSA.*

Keywords: Cone beam CT, Reconstruction algorithm, Computer unified device architecture, Graphic processing unit, Texture memory, Constant memory.

1. Introduction

Cone beam CT (Cone-Beam Computed Tomography, CBCT) compared with conventional CT system has scanning speed, low amount of radiation, better spatial resolution and other advantages. So it is widely used in medical imaging and industrial testing field [1]. The calculation amount is large and time-consuming is long of the reconstruction algorithm of three-dimensional cone beam CT. Such as Feldkamp-Davis-Kress (FDK) algorithm without any acceleration measures reconstruction the body data of 5123 needs about 90 min. The algorithm is very difficult to satisfy the application requirement. So how to improve the reconstruction speed is the urgent problem to be solved [2].

In recent years many domestic research institutions and scholars have done a lot of research on the acceleration problems of cone beam CT reconstruction algorithm. In literature [3] Fang Xu and Klaus Mueller realized the accelerate of FDK reconstruction algorithm completely in graphics and solved the transmission bottleneck between the CPU-GPU. However due to the operation of floating pipeline is about five times slower than the ordinary texture pipeline, Fang Xu and Klaus Mueller proposed the joint use of floating pipeline and normal pipeline to improve the quality and the speed of image reconstruction and achieved satisfactory results [4]. Literature [5] used multiple textures mapping to improve back-projection speed and reduced the storage amount of intermediate data and the accumulation number of floating point, and used

vertex color channel to realize the distance weighted computing, and used the extension methods to increase the texture units of parallel back-projection, so as to improve the reconstruction speed. Literature [6] used multi-threading technology to achieve the fast parallel of cone beam CT on multi-core platforms. Literature [7] proposed a voxel density modeling based on projection data and realized the fast imaging of cone beam CT. The implementations of these methods need to use API and the implementation processes are more complex.

In 2006, the United States NVIDIA company putted forward a graphics processor based on CUDA (Computer Unified Device Architecture, CUDA) and could bring reconstruction algorithm as general multiprocessor task performed in the GPU by unified programming technology of CUDA. Holger Scher1 etc proposed an accelerating image reconstruction algorithm to the unified architecture[8]. Literature [9] used stream processors in CUDA graphics processors to accelerate the filtering and back-projection calculation and realized the reconstruction acceleration of FDK. But the methods in Literature [9] has high complexity and large calculation amount. For the large computation of FDK algorithm in filtering and back-projection process, this paper rearranges the pre-filtered data by the CUFFT function library in CUDA and computes back- projection by two-dimensional thread block and speeds up the speed of the filter and back-projection. And this paper uses the texture memory and constant memory in CUDA model to respectively store the kernel function and the filtered data, shortens the data access time and realizes the acceleration of FDK algorithm.

2. FDK Algorithm

The FDK algorithm which was proposed by Feldkamp, Davis and Kress in 1984 for the cone beam geometry circular scan trajectory is a kind of three-dimensional image reconstruction algorithm based on a circular orbit scanning filtering back-projection. Compared with the iterative reconstruction algorithm, FDK algorithm belongs to the analytic reconstruction algorithm. The main advantages of this algorithm are as follows. First, the mathematical form of the algorithm is simple and easy to realize for computer hardware system. Secondly in the case of the small rag beam cone angle, this algorithm can obtain satisfactory reconstruction effect and can satisfy the need of practical application. Finally, the mechanical structure of hardware system is simple. FDK algorithm is a general algorithm in engineering practice.

The principle diagram of FDK cone beam reconstruction algorithm is shown in Fig. 1. Ray source S is rotated around the central axis Z. R is turning radius. β is fan angle. θ is the intersection angle of the ray source and the axis X positive direction. For the convenient of computing, the detector in Fig. 1 is a virtual detector. $y'oz$ is virtual detector plane. In the rotation angle θ, the projection position corresponding point of a reconstruction point X in object is Q and the coordinates is (Y, Z). Thus the projection data of the object point can be expressed as $p_\theta(Y, Z)$.

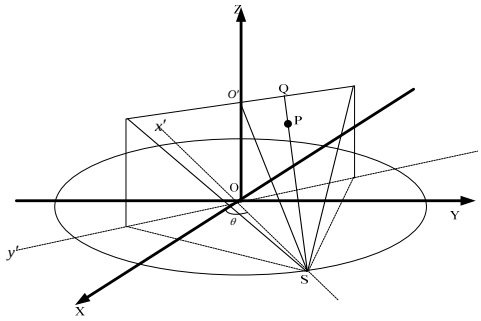

Fig. 1. Principle diagram of FDK.

FDK reconstruction algorithm mainly includes three steps: weighted, filtering and back- projection. The concrete realizations are as follows:

1) Projection data are weighted.

$$p'_\theta(Y,Z) = p_\theta(Y,Z)\frac{R}{\sqrt{R^2+Y^2+Z^2}}, \quad (1)$$

where $\frac{R}{\sqrt{R^2+Y^2+Z^2}}$ is modified weight coefficient, which is the cosine of the intersection angle between any ray and center ray. $p'_\theta(Y, Z)$ is weighted result of projection data.

2) The revised projection data are filtered.

$$p^*_\theta(Y, Z) = p'_\theta(Y, Z) * h(Y), \quad (2)$$

where $h(Y)$ is filtering kernel function and $p^*_\theta(Y, Z)$ is the result of filtered projection data.

3) Weighting and back- projection computing.

$$f(x,y,z) = \int_0^{2\pi} \frac{R^2}{V(x,y,\theta)^2} p^*_\theta(Y(x,y,\theta), Z(x,y,\theta))d\theta, \quad (3)$$

where

$$V(x, y, \theta) = Yr(x, y, \theta) + R, \quad (4)$$

$$Y(x, y, \theta) = \frac{Xr(x, y, \theta)}{V(x, y.\theta)}R \quad (5)$$

$$Z(x, y, \theta) = \frac{R \bullet z}{V(x, y, \theta)} \quad (6)$$

$$Xr(x, y, \theta) = -x\sin\theta + y\cos\theta \quad (7)$$

$$Yr(x, y, \theta) = -x\cos\theta - y\sin\theta \ , \qquad (8)$$

where V is the weighting factor and (Y, Z) is the coordinate of ray source through the rebuilt point $f(x, y, z)$ in detector plane.

In the equation (3), FDK algorithm separately deals with each angle data in the process of filtering and back- projection. The data are independent under different angles. The different reconstruction point is also independent of each other for the data under the same angle. So the parallel degree of this algorithm is high.

3. CUDA Basics

CUDA is NVIDIA's general parallel computing architecture. The C language is used to be development language in CUDA. CUDA don't need to use API in graphics. Its purpose is to make the GPU as a general parallel computing equipment, and to solve the complicated calculation problem. At present, its application has gradually beyond the computer graphics, and has been widely used in all kinds of computation intensive research field.

3.1. CUDA Programming Model

CUDA programming model makes GPU as parallel computing device to run multiple threads. And GPU is the coprocessor of main CPU. The CUDA program compiled by the compiler NVCC is divided into two parts: one part is the host code run on CPU and the other part is device code run on GUP. The host code is generally serial calculation program which controls process. The device code is parallel program for intensive computing.

The parallel computing function on the GPU is called Kernel. Kernel is organized by grid. The internal of each grid is composed by several blocks. Each block is composed by some thread. Essentially, Kernel is executed in the unit of block. Each block is executed in parallel. Blocks can't communicate with each other and also haven't the order of execution. In actual operation, the block is divided into smaller thread group which composes warp. And the warp is the real execution unit. CUDA programming model is shown in Fig. 2 [10].

3.2. CUDA Memory Model

In addition to programming model and execution model, CUDA also has specific memory model which is shown in Fig. 3.

In Fig. 3 each thread has its own registers and local memory. Each block has a piece of shared memory. When thread is executed, it can access to data in different storage spaces. All threads in grid can access global memory. And there are two kinds of read-only memory which can be accessed by all threads: constant memory and Texture Memory. The data of local storage and global memory are be stored in video memory and not stored in register or cache, so the access speed is very slow. Registers and shared memory are cache memory on the GPU and execution unit can be accessed with very low delay. Constant memory and texture memory have cache acceleration which can save bandwidth and have faster access speed.

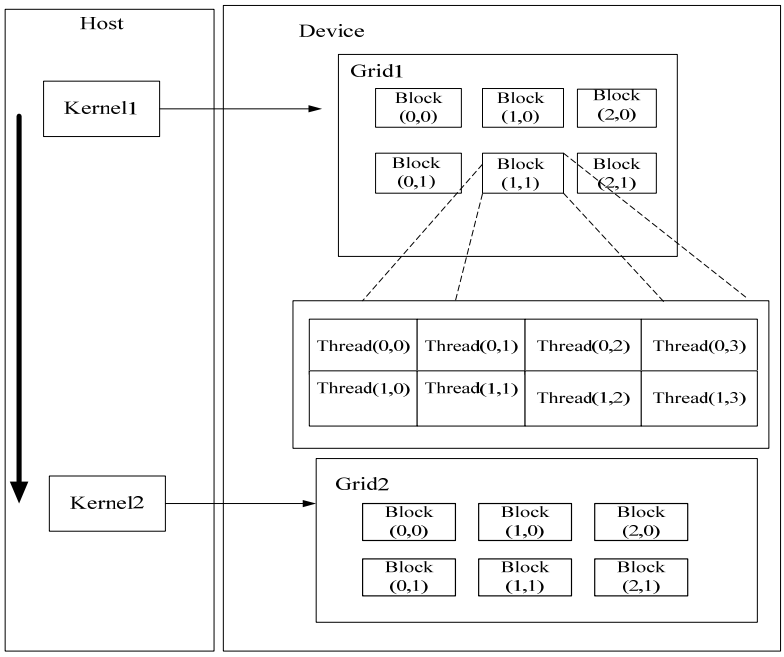

Fig. 2. Programming model of CUDA.

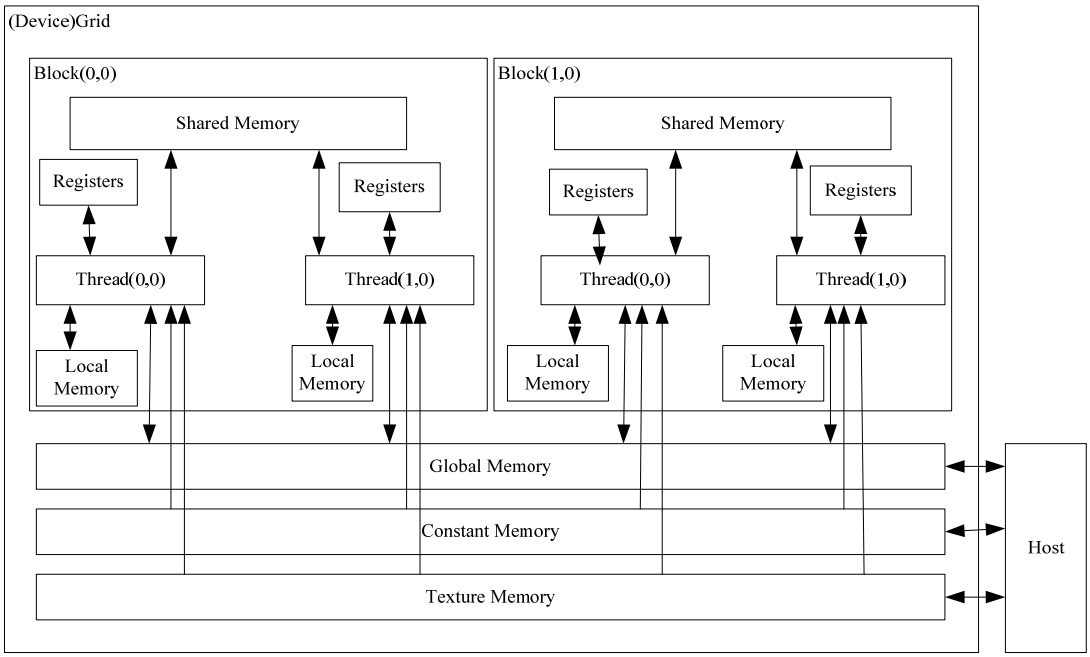

Fig. 3. Storage model of CUDA.

4. Optimizing the FDK Algorithm with CUDA

In the realization process of FDK algorithm, calculation amount of filtering and back-projection is large and the time complexity of calculation is high. Therefore optimizing FDK algorithm is mainly to accelerate the filtering and back-projection. For these problems such as large amount CT data and time-consuming to access, this paper rationally uses constant memory and texture memory in CUDA memory model to access data, and improves the access efficiency.

4.1. The Optimization of FDK Algorithm

The filtering computing on the projection data, which is equivalent to the convolution computation, can be realized by FFT algorithm. NVIDIA's CUFFT for calculating FFT provides some simple interface. Using cores in Nvidia GPU, CUFFT realizes the FFT computation of single precision floating point number in GPU. But because FFT function of CUFFT library can only be applied to complex, the projection data before filtered needs be rearranged. Thus two rows data which real part and imaginary part corresponds to can be simultaneously filtered. This row data rearranged by the two rows is calculated Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT).Then the row data is rearranged to the corresponding two rows. This method can reduce filtering operation time to half of the original. The execution process is shown in Fig. 4.

In the projection reconstruction process, the back-projection of all voxels in each angle can be simultaneously calculated, and each thread calculates the back-projection of a voxel. So a three-dimensional thread block is necessary. But the same z axis point (x, y) coordinates are the same, the three-dimensional thread block can lead to formula (5) and (8) repeatedly calculated which reduces calculation efficiency. This paper calculates back-projection by a two-dimensional thread, and each thread completes the reconstruction of series of voxels in the Z axis. In this way, the reconstruction point handled by each thread has same x, y and Y . The formula (9) can be drawn by the formula (6) and (7).

$$Z(x_0,y_0,z_1,\theta)=\frac{R}{V(x_0,y_0,\theta)}\Delta z+Z(x_0,y_0,z_0,\theta) \quad (9)$$

In formula (9), $Z_1 = Z_0+\Delta Z$. This method can make full use of the Z axis correlation of voxels, effectively reduce the complexity of the algorithm, and easy to be expanded. The way that thread is divided is shown in Fig. 5.

Such as formula (9), this new division way makes that every single thread only uses formula (5) to calculate the Z of each voxel corresponding, and then uses formula (9) to simply the projection index calculation of other voxels. And this division way also optimizes the access order of the two-dimensional texture, makes full use of the texture cache, and can reduce the register consumption.

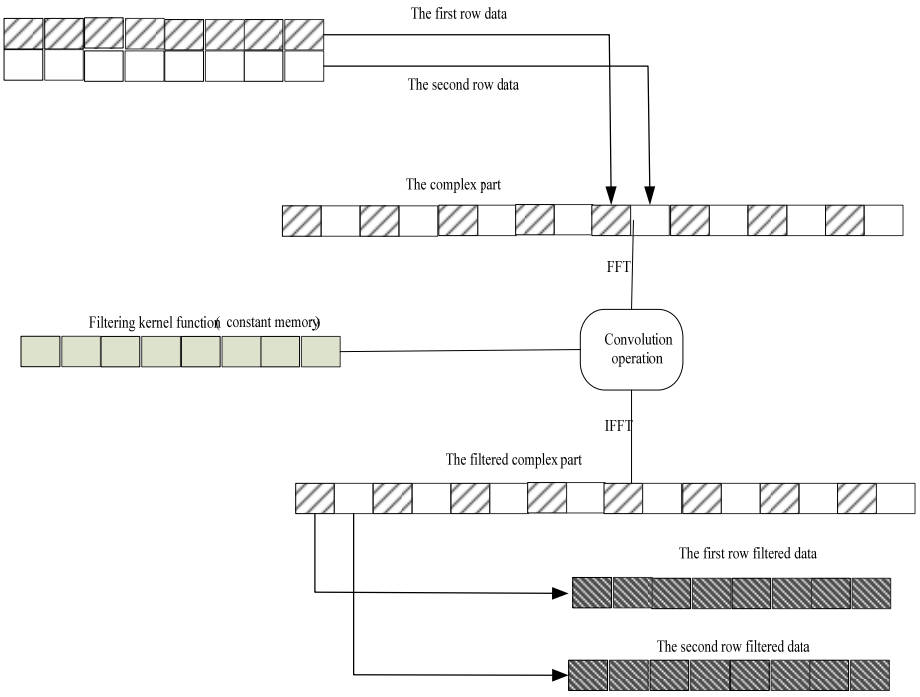

Fig. 4. Implementation process of filtering.

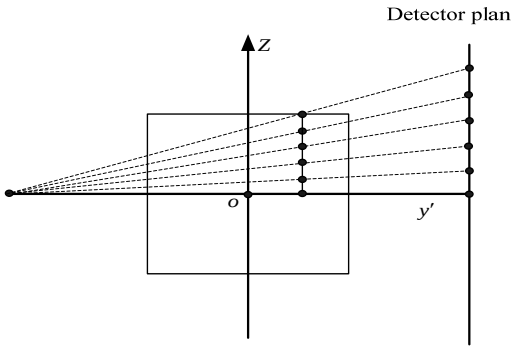

Fig. 5. Thread division of reverse projection.

4.2. Memory Type Optimization

Filtering is that the back-projection data in reconstruction process are made one dimensional filtering. The filtering kernel functions are the same in each row. If the filtering kernel function is stored in the global memory, each row filtering data needs to be read from the global memory. But the global memory in GPU is common memory and furthest away from the arithmetic execution unit, and has no cache. The global memory access requires 400-600 clock cycles and access speed is relatively slow. Due to the constant memory in GPU has cache which access speed is fast, the fixed filtering kernel function can be stored in the constant memory in GPU to improve the access speed. The back-projection process of the FDK algorithm is the part which has the largest calculation amount and most time-consuming. In using CUDA technology, each thread calculates the back-projection of one pixel in the slice image and adds the calculation results to the corresponding pixel. First each thread calculates its projection coordinates (*Y*, *Z*), and then interpolates its back- projection. Due to projection data is random accessed in the interpolation process, direct picking pretreated data in video can cause an access violation and increase memory access latency and reduce memory access efficiency. The back-projection only read-only accesses the pretreated data, and in view of this the filtered data can be bound to the texture memory which has good acceleration for large amount random access to increase the efficiency of memory access. At the same time the texture memory also has linear filtering processing functions which doesn't occupy the programmable computing unit and directly calls the hardware resources, and eliminates the interpolation time to improve the computing efficiency.

4.3. Instruction Stream Optimization

Due to Various instructions in the program have different execution time, those instructions which occupy less execution time can be used as far as possible to improve the execution speed. Instruction-level optimization of program code that needs to run multiple times can effectively improve the throughput of the instruction stream which refers to the performed operation number of each multi-processor in one clock cycle.

Each pixel of each angle in the back-projection process is required to calculate the formula (4), (5), (7) and (8), and these formulas include the division and

the trigonometric calculation. In CUDA, the instruction stream throughput of single-precision floating-point addition is 0.88 operating per clock cycle. The throughput of the built-in function $\_fdivide(x)$ in CUDA is 1.6 operating per clock cycle. Therefore, using the built-in function $\_fdivide(x)$ can increase the throughput of the instruction stream. For the instruction operation computation amount to $\sin x$ and $\cos x$ corresponding, this paper uses the built-in functions sin(x) and cos(x) to improve the throughput of the instruction stream in the high-speed version of CUDA.

5. The Experimental Results of Accelerating Reconstruction

The experimental platform configuration is as follows: AMD5200+ CPU, 2G memory of Kingston, GTX550 graphics card, XP operation system, Visual Studio 2005+CUDA 3.0 development environment. The data model is Shepp – Logan, scanning mode is circular locus, the effective detector size is 512×512, the rotation angle is 360 degrees, scanning interval is 1 degree, the distance between the ray source and the rotation center is 850 mm, the distance between the object center and the probe is 150 mm, the FDK algorithm is respectively rebuilt by GPU and CPU and the reconstruction size is 512×512×512. The reconstruction results by GPU and CPU are respectively shown in Fig. 6 and Fig. 7.

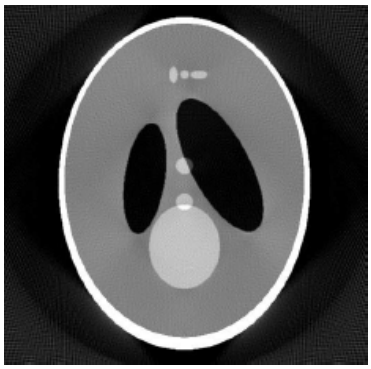

Fig. 6. GPU Reconstruction.

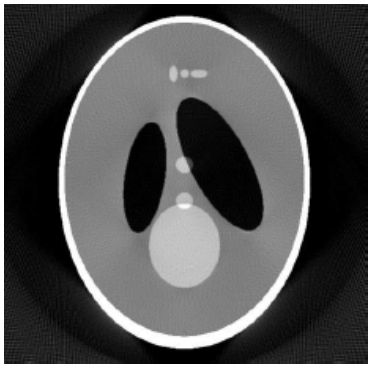

Fig. 7. CPU Reconstruction.

The comparison of reconstruction time of the optimized FDK algorithm and the original CUDA method is shown in Table 1.

It can be seen from Fig. 6 and 7 that the reconstruction experimental results respectively by GPU and CPU are basically the same and within the scope of the allowable error. From Table 1 it can be seen that after the memory optimized, the reconstruction speed compared with the original CUDA method can be increased nearly 3.6 times. This is because that global memory access latency is long and the speed of accessing data from global memory is slow. After optimized by the algorithm and directive, the reconstruction speed can be increased by about 10 times. From the reconstruction schedule of Table 2, it can be seen that the GPU reconstruction time compared with the CPU can be increased 84 times and the back-projection process can be increased nearly 89 times, and the reconstruction speed has been significantly improved.

Table 1. The reconstruction contrast of original CUDA and optimized FDK (unit: second).

Algorithm	Algorithm execution process			
	Optimization mode	Filtering time	Back -projection time	Total time
Original CUDA	Null	2.5	112.5	115.0
Optimized FDK	Memory optimization	1.5	30.4	31.9
	Algorithm optimization	0.7	10.7	11.4
	Instruction optimization	0.8	10.5	11.3

6. Conclusion

This paper puts forward a method which can accelerate FDK algorithm by CUDA. This method improves the FDK respectively from the memory optimization, algorithm optimization and instruction optimization. Experimental results show that the speed can reach 11.4 s with the projection data of 360 degrees 512 × 512, and the reconstruction speed compared with CPU improves 84 times. After the circle of projection data is collected, the reconstructed voxel data can be obtained. Due to the limitations of instruction and memory bandwidth, the computing performance of filtering and back-projection needs to be further improved. How to take advantage of asynchronous processing and stream processing mechanism to improve the speed of the reconstruction algorithm is the content of the future research.

Acknowledgements

The Work is supported by Natural Science Foundation of Shanxi Province of China (N0.20110011053)

References

[1]. A. V. Narasimhadhan, Kasi Rajgopal, FDK-Type Algorithms with No Backprojection Weight for Circular and Helical Scan CT, *International Journal of Biomedical Imaging*, Vol. 2012, Issue 11, 2012, pp. 1-12.

[2]. M. Abella, J. J. Vaquero, A. Sisiega, J. Pascau, A. Udias, V. Garcia, etc, Software architecture for multi-bed FDK-based reconstruction in X-ray CT scanners, *Computer Methods and Programs in Biomedicine*, Vol. 107, Issue 2, 2012, pp. 218-232.

[3]. Nitin Jain, M. S. Kalra, Characteristic Signature of Specimen Using an Approximate Formula for 3D Circular Cone-Beam Tomography, *Research in Nondestructive Evalation*, Vol. 22, Issue 3, 2011, pp. 169-195.

[4]. Dai Zhisheng, Chen Zhiqiang, Xing Yuxiang, Accelerated 3-D T-FDK reconstruction Algorithm using the commodity graphics hardware, *Journal of Tsinghua Univ. (Sci & Tech)*, Vol. 46, Issue 9, 2006, pp. 1589-1592.

[5]. Ma Cheping, Zeng-Li, Fast 3D CT reconstruction with multitexture of GPU, C*omputer Engineering and Applications*, Vol. 44, Issue 7, 2008, pp. 227-228.

[6]. Zhang Shunli, Zhang Dinghua, Li Xiaolin, Fast Calculation of Simulation Projection for Cone-beam CT Based on Shepp-Logan Head Phantom, *Computer Science*, Vol. 39, Issue 5, 2012, pp. 257-260.

[7]. Qian Ying, Yang Wenfeng, Cone-beam CT Perfusion Imaging Method on Image-guided Radiation Therapy, *Journal of Computer Applications*, Vol. 31, No. 5, 2011, pp. 1242-1248.

[8]. Scherl H., Keck B., Kowarschik M., Fast huGPU-Based CT reconstruction using the common unified device architecture(CUDA), *Nuclear Science Symposium and Medical Imaging Conference*, Paris, France, 18-20 August 2009, pp. 4464 -4466.

[9]. Wang Jue, Cao Siyuan, Zhou Yongning, Fast Cone-beam CT Reconstruction with CUDA, *Nuclear Electronics & Detection Technology*, Vol. 30, Issue 3, 2012, pp. 315-318.

[10]. Wang Cianchao, Hu Guoen, Yan Bin, etc, Fast low-dose reconstruction from truncated data in dental CT, *IEEE Transactions on Nuclear Science*, Vol. 60, Issue 1, 2013, pp. 174-181.