Algorithm of Ray Casting Volume Rendering Based on CUDA

Zhu Shi^{a*}, Chang Jing Yi^a Xie Cong Hua^a

Chang Shu institute of technology School of computer science and Engineering

JiangSu Chang Shu 215500 China

edward_zs@163.com

Abstract

Direct volume rendering is one of the methods for the visualization of 3D data set, It does not construct intermediate entity, directly generate 2D graphics on the screen by the 3D data set, which is better for parallel processing, but the amount of calculation is large, difficult to rendering with conventional graphics hardware. This paper mainly elaborated optimization and improvement about the algorithm of ray casting volume rendering for the visualization of 3D data set, within the framework of the CUDA, using the multi-core parallel computing ability of GPU. The vertex shader, pixel shader, calculation of the starting point of the sampling points of light, color and opacity tired and synthesis of image operations are completed by GPU. The vertex shader, pixel shader, calculation of the starting point of light, color and opacity of the sampling points accumulation and image synthesis operation are achieved by GPU. Compared with the GPU-based ray-casting algorithm, the algorithm takes full advantage of the characteristics of the CUDA parallel processing, can quickly draw a higher quality image, rendering speed has been raised about 15%.

Keywords: volume rendering ray casting CUDA.

1. Introduction

The visualization of 3D data set has experienced 20 years of development since its proposed, the algorithm can be divided into two types, surface rendering and volume rendering. Surface rendering render the geometric surface with traditional computer graphics technology which is structured by 3D data set; Volume rendering is directly generated two-dimensional graphic by the corresponding three-dimensional data set. The latter has the advantage of showing the complete picture of the object as well as the details internal, but the amount of calculation is huge. It is difficult to achieve real-time rendering requirements with traditional algorithm based on software. Acceleration technology based on software such as opacity early

termination, empty voxels Ignore, viewing transformation optimize etc, constrained by hardware, the effect of acceleration is limited, can't achieve real-time dynamic graphic display. With the development of hardware technology, direct volume rendering is implemented on the high-end graphics workstations based on programmable hardware, this improved rendering speed, however the cost of hardware is high. With the advent of CUDA, we can direct manipulate hardware resources of GPU with class C high-level language without requiring the aid of graphics application programming interface, some of the visualization work can be completed by GPU with exclusive technology of programmable GPU pipeline, the parallel processing of data greatly improved processing speed.

At present, the domestic research on CUDA technology is at the stage of development, also made a series of direct volume rendering acceleration technology based on CUDA. This paper mainly elaborated optimization and improvement about the algorithm of ray casting volume rendering for the visualization of 3D data set based on CUDA, we storing volume data into graphics memory in the form of three-dimensional texture, the vertex shader, pixel shader, calculation of the starting point of light, color and opacity of the sampling points accumulation and image synthesis operation are achieved by GPU, these all accelerate rendering.

2. Algorithm design

2.1 The architecture of CUDA and ray casting

CUDA is a hardware and software system of a GPU as a data parallel computing device, which is a three-tier structure composed by grid, block, organized by form of grid thread. Each grid is composed of a plurality of thread blocks, each block is composed of a plurality of threads.

Ray casting is along a fixed direction emitting a light (usually the viewing direction), light across the image sequence, and in this process, the image is sampled to obtain the color information of the sequence, meanwhile

according to the light absorption model the color values and opacity will be accumulated, until the light across the image sequence. we will get a pair of complete visualization by accumulating the color of all pixels.

In ray casting algorithm, the calculation process about ray along a fixed direction of emitted light is independent of each other, the calculation method is the same and has the advantages of huge parallelism. According to this, the screen is considered as a grid, each pixel considered as a thread, according to the the image characteristics, the screen is divided into several regions, which is considered as a block, then ray casting algorithm can be implemented based on cuda three-tier architecture. The algorithm includes two parts, one is CPU (Host) part; another is GPU (Device) part. CPU part reads 3D data file and stores data in memory, initialize OpenGL environment, and establish links between video memory. receives the final results from GPU and display the result of visualization. GPU part ergods ray and fnd the intersection of ray and the data set, accumulats color and opacity, returns the final result to memory

The specific steps of the algorithm are as follows:

CPU part

- (1) Read dimensional data set of raw format, store in the device memory as the form of one unsigned char array.
- (2) Initialize OpenGL environment, establish PBO (OpengGl pixel buffer objects) and create a link to GPU memory ; setting model transformation matrix, set the viewing direction.

- (3) According to the size of data set, calculate the required size of block and thread.

GPU part

- (1) Initialize CUDA environment, establish 3D array and 3D texture of CUDA, unsigned char array data set of device memory read into 3D Array, set the texture parameters and bind 3D Textrue and 3D Array .

- (2) Set pre parameters, calculate corresponding screen pixel position, viewpoint coordinates according to the thread index

- (3) Calculate the the distance to the camera, the coordinate of the intersection lin about the ray enter the data set and lout about the ray leave the data set with the Ray - Box Intersection method, according to the ray direction.

- (4) Sample along the ray direction from the front to the back according to a preset sampling interval, get the color and opacity of the sampling points and accumulate them. Set the opacity threshold is equeal to 1, if the transparency value close to 1.0, stop computing, otherwise

continue to sample the next point, until the left point is reached.

- (5) The eventually accumulated value of color and opacity are stored in the PBO

Thus, the GPU work is completed, the data of PBO is directly displayed on the screen through the API of OpenGL environment.

2.3 Mapping of threads and light and designing of pre parameters

Mapping of threads and light is creating a link between any ray with corresponding thread of CUDA, which is the key of parallelization about ray casting algorithm. Two built-in CUDA variables gridDim, blockDim signify the scale of the grid and thread blocks. Two built-in CUDA variables blockIdx, threadIdx signify the thread block number and thread index. Through the combination of the two signify the parallel thread unit. blockIdx, threadIdx contains two dimensions, namely X and y. Create the index of every ray through thread index according to the formula 1 below.

$$index = (blockindex) * (blockDim) + (threadindex) \quad (1)$$

Inorder to facilitate the calculation, the value of dimensions of threadIdx is standardized to the interval [-1,1] according to the formula 2,3 below.

$$u = (index.x / imageW) * 2 - 1 \quad (2)$$

$$v = (index.y / imageH) * 2 - 1 \quad (3)$$

imageW signify the width of view plane, imageH signify the height of view plane.

Pre parameter design, first standard view plane size and eye position, as shown in figure 1.

Fig.1. Spatial relation

Eye position is on the z axis with coordinate(0,0,6), view plane is on the z axis(0,0,3) and the size is 3*3. Second, set unit Ray-Box, the imaging range constraint

in which. The lower left corner coordinates of the Box is (-1,-1,-1), the upper right corner coordinates of the Box is (1,1,1).

2.4 Intersection of ray and color and opacity accumulation

The key of ray-casting algorithm is intersection of ray , accumulating color and opacity values along the direction of the ray, and the composite image. First step is intersection operation of ray, for any light, using vector notation to describe it, set two basic parameters, origin coordinates and view direction, then any ray can be expressed as a formula 4 below, t of formula 4 represents the displacement of ray in the direction of view.

$$Ray_i = eyeRay.o + eyeRay.d * t \quad (4)$$

For seeking the intersection between rays and Ray-Box according to methods proposed by G. Scott Owen, calculate the two intersections of ray and Ray-Box, named lin and $lout$, then taking lin as the origin, moving along the line of sight, each time forward step distance. Every step forward, calculate the coordinates of a sample point on the ray in space according to the formula 5 below.

$$pos = eyeRay.o + eyeRay.d * lin + eyeRay.d * step \quad (5)$$

($eyeRay.o$ is the origin coordinates of ray, $eyeRay.d * lin$ is the coordinates of the point where ray in, $eyeRay.d * step$ is the forward distance of ray each time). Since the pos is in the Ray- Box, its coordinate values is located between $[-1, -1, -1]$ and $[1, 1, 1]$. Information of image color and opacity is stored in CUDA 3D Array in the form of 3D texture, texture coordinates is located between $[0, 1]$. Convert coordinate of pos to texture coordinates of posttexture according to formula 6 below.

$$posttexture = (pos + 1) / 2 \quad (6)$$

Then, according to the coordinates of point, calculate the corresponding texture coordinates, obtain the appropriate texture value based on texture coordinates which is the gray value of the point. Create a one-dimensional transfer function to indicate different content of image. The function has four components which is R, G, B, α , the first three parameters is the conversion between color and gray, the fourth is to control the transparency, through the function we can control the conversion between gray and color.

Finally is the accumulation of opacity and color. In here , we calculate the color and opacity according to

formula 7 and formula 8 below with method from the front to the back.

$$C_{out} \alpha_{out} = C_{in} \alpha_{in} + C_{now} \alpha_{now} (1 - \alpha_{in}) \quad (7)$$

$$\alpha_{out} = \alpha_{in} + \alpha_{now} (1 - \alpha_{in}) \quad (8)$$

In the formula 7 and 8 , C represent color value, α represent opacity. In the experiment, the early termination of opacity is adopted. The threshold is set to 1. If the opacity has been reached threshold, the voxels behind will be Ignored, this can save time. At last, the accumulated value is directly stored to the PBO.

3. Experimental results and analysis

The algorithm above is implemented in the hardware system and software system followed, cpu is Intel Core2 dual core T5500, memory size is DDRII 2GB, graphics card is NVIDIA GeForce 8400M GS with memory 512MB, operation system is Windows XP, development environment is VC++6.0, CUDA SDK3.1 and OpenGL3.1. It also Compared with ray casting algorithm based on GPU. In here two different scale data were used to test, the parameters of data set are shown in table 1. The size of original image about the experimental results is 512*512.

Table1 basic information of Data set tested

Volume data	neghip	engine
size	64*64*64	256 x 256 x 256
Storage space	256KB	16MB
sampling rate	64	256

3.1 Result of rendering

Fig 2(a) reconstruction based on CUDA

Fig2(b) reconstruction based on GPU

Fig3(b) reconstruction based on GPU

Figure 2 above is the reconstruction of neghip, which size is $64 \times 64 \times 64$. Figure a is the reconstruction based on CUDA, figure b is the reconstruction based on GPU. The resulting image of figure a is better than figure b, the internal details of neghip is richer.

Figure 4 shows the resulting image with transfer function of color, this can indicate different content of image. Here, the transfer function is one-dimensional. It's a simple mapping between the texture gray information of volume data and color, the information is stored in 3D array of CUDA as mentioned above. 11 control points are set in the transfer function, representing the different gray scale values corresponding to the corresponding color, intermediate values, using simple linear interpolation algorithm to calculate.

Fig 3(a) reconstruction based on CUDA

Figure 3 is the reconstruction of engine, which size is $256 \times 256 \times 256$. Figure a is the reconstruction based on CUDA, figure b is the reconstruction based on GPU. The resulting image of figure a is better than figure b, it shows more internal details of engine.

Fig 4 (a) reconstruction of neighp with color transfer

Fig 4 (a) reconstruction of engine with color transfer

3.2 The rendering time analysis

Table 2 lists the time required for reconstruction and comparison of the rendering speed of two sets of data set on the same hardware environment with different algorithm. The rendering time contains the data read, texture creation, the ray traversal, color and transparency of accumulation, the final display. See from the table, ray casting algorithm based on CUDA has the distinct enhancement in the rendering time. Take data set of neghip as example, rendering time of GPU-based algorithm is 0.812 seconds, rendering time of CUDA-based algorithm is 0.703 seconds, it's speed increased to 1.16 times of the GPU-based algorithm. For the engine data set, rendering speed CUDA-based also increased to 1.15 times of GPU-based.

Table2 comparison of rendering time

Volume data	Rendering time (second)		speed boost
	GPU	CUDA	
neghip	0.812	0.703	1.16
Engine	0.842	0.735	1.15

3.3 Performance analysis

From the above experimental results, rendering speed of algorithm CUDA based on has increased by about 15%. Texture generation, ray traversal and accumulation of color and opacity spend more time. Table 3 and 4 lists statistics of the two part of the reconstruction process above.

Table3 Time of algorithm GPU-based

Volume data	Time (second)		
	texture generation	ray traversal, accumulation of color and opacity	Total
neghip	0.026	0.412	0.438
Engine	0.07	0.432	0.502

Table4 Time of algorithm CUDA-based

Volume data	Time (second)		
	texture generation	ray traversal, accumulation of color and opacity	Total
neghip	0.0012	0.097	0.0982
Engine	0.013	0.1	0.113

As can be seen from two tables above, the time CUDA-based of texture generation, ray traversal, accumulation of color and opacity has greatly improved than that GPU-based, the former time is about 1/4 of that of the latter. This embodies that the characteristics of graphics chip is single-instruction, multiple data stream processing, the advantages of CUDA architecture is parallel processing. But the use of texture in CUDA SDK3.1 has a shortcoming, texture data stored in memory can not be directly copied to 3D Array of CUDA, it must use a temporary storage space as an intermediary, texture data is first copied to temporary, then from temporary to 3D Array of CUDA. So quickly generate texture, storage need more time this affects the rendering, secondly, the algorithm CUDA-based consists of two parts, cpu parts and gpu parts, the data need to exchange between the two parts, it also has some impact on the speed. The final rendering speed increased by about 15%. Optimization of imaging effect, mainly lies in the data filtering using normalized linear filtering.

4. Copyright

The paper and researches is original and unpublished, they also have not been submitted to other conferences or journals before.

5. Conclusion

In this paper, a parallel CUDA architecture technology, programmable hardware technology, ray casting algorithm are used for visualization of 3D data set. Under the same hardware and software condition, the same 3D data

set, compared with the ray casting algorithm based on GPU, Since the CUDA architecture has advantages of parallel processing, the rendering speed is improved by about 15%. On the imaging results, filter function of cuda sdk library can help to better show details. Later work is how to improve the transfer function, According to the characteristics of drawing objects and need to use different colors to indicate the contents, and better improve the interactivity.

Acknowledgment

The author would like to thank the following people for their help and suggestions: doctor Chang jin yi, Xie Cong hua. The 3D data set is provided by O. Sadowsky and the Web site of the volume library, here the author also express his gratitude

Reference

- (1) SCHERLH, Keck B, Kowarschik M, et al: "Fast GPU-based CT reconstruction using the Common Unified Device Architecture", IEEE Press .Proceeding of Nuclear Science Symposium and Medical Imaging Conference. Washington, DC: IEEE Press, pp.4464-4466, 2007.
- (2) Lu YQ, Wang WM, Xie YM, et al: "Accelerating algebraic reconstruction using CUDA-enabled GPU". 2009 Sixth International Conference on Computer Graphics, Imaging and Visualization. Washington, DC, IEEE Computer Society, pp. 480-485, 2009.
- (3) O. Sadowsky, J.D. Cohen, and R.H. Taylor, "Rendering Tetrahedral Meshes with Higher Order Attenuation Functions for Digital Radiograph Reconstruction," Proc. Conf. IEEE Visualization, pp. 303-310, 2005.
- (4) Westenberg M A, Roerdink. "Frequency Domain Volume Rendering by TheWaVelet X-Ray Transform" . IEEE Transactions on Image Processing, 9(7), pp.1249-1261, 2002.
- (5) Xu SH, Zhang EH: "CUDA-based parallel visualization of 3D data", CT Theory and Applications, 20(1), pp.47-54, 2011.
- (6) Dong XL, Jiang GP, Zhang Y: "Fast ray casting algorithm based on CUDA", Beijing Biomedical Engineering, 29(2), pp.125-129, 2010.
- (7) Bi YW, Cheng ZHQ, Zhang L: "Real time visualization method of 3D reconstruction based on CUDA", 19(2), pp.1-8, 2010.
- (8) Zhang S, Zhu YL, "GPU high performance computing of CUDA", China hydraulic press, 2009
- (9) QiuDY, "GPGPU programming technology -- from GLSL, CUDA to OpenCL", China Machinery Press, 2011.
- (10) Shi J Y, Cai WL, "Scientific visualization algorithms and systems", Science Press, 1996.
- (11) Tang ZR, "Visualization of 3d data set", Tsinghua University press, 1999.
- (12) Cai W. "Transfer function in DRR volume rendering", Proceeding of the 13th International Congress and Exhibition of Computed Assisted Radiology and Surgery, Paris, pp. 284~289, 1999.
- (13) Sherouse W, Novins K, Chaney E L. "Computation of digitally reconstructed radiographs for use in radiotherapy treatment design". International Journal of Radiation Oncology, Biology and Physics, 18(3), pp.651~658, 1990.
- (14) Wu L Wang B: "CUDA volume rendering algorithm based on GPU" Life science instruments, 7(12), pp.26~29, 2009.