International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.1 (2014), pp.199-208
http://dx.doi.org/10.14257/ijmue.2014.9.1.19

Fast 3D Graphics Rendering Technique with CUDA Parallel Processing

Ji-Hoon Kang[1], Syung-Og An[2], Shin-Jin Kang[3], Seok-Hun Kim[4] and SooKyun Kim[2]

[1]*Department of Computer Science Education, Korea University*
[2]*Department of Game Engineering, Paichai University*
[3]*School of Games, Hongik University*
[4]*Department of Mobile Media, Suwon Women's College*

Corresponding Author: SooKyun Kim (kimsk@pcu.ac.kr)

Abstract

3D Graphic Rendering has been used to express realistic, 3-dimensional, and emphasized effects in the graphics. As 3D Graphic Rendering developed and became more prevalent, the need for acceleration in data processing grew as well, leading to a development of GPU (Graphic Processing Unit) and shading language used for GPU such as GLSL (OpenGL Shading Language) and HLSL (Higher Language Shading Language). 3D Graphic Rendering based on GPU, however, clearly has its limitation in processing complicated calculations such as calculating curvatures of the surface or ray tracing method, especially as the greater magnitude of the 3D polygonal model data is being used. The following paper will discuss the new method of 3D graphic rendering that is based on faster GPU parallel processing system called CUDA (Compute Unified Device Architecture) to administer 3D polygonal model data and process calculations. In the paper, we will discuss about the characteristics of CUDA and test for graphic rendering of 3D polygonal model according to those characteristics. We will also examine whether it is possible to accelerate the graphic rendering process using CUDA for 3D graphic rendering.

Keywords: *3D Graphic, CUDA, Parallel Processing, GLSL, GPU*

1. Introduction

Developed from the 2D computer graphics that used points and lines to express images [15], 3D graphic rendering system has introduced the new concept of 3-dimensional space in the graphics and has been used to produce more realistic images. As the visual quality of the graphics improved thanks to the development of 3D graphic technology, the amount of the data used relatively increased as well, and consequently created a lot of problems with the duration of the calculation process. Thus, there was a need for a consensus and a balance between the calculation rate and the visual qualities. In order to resolve these problems with data processing speeds, shading languages such as GLSL (Open GL Shading Language)[1] and HLSL (High Level Shader Language)[2] were established. While the complications with rate of calculation are being resolved and improved, there are still problems in finding a consensus between the calculation speed and visual quality. The following thesis will propose a new method of data processing for 3D graphic rendering. We have adopted 3-dimensional polygonal model and have tested with calculation of normal vectors used for lighting of the rendering of these models.

ISSN: 1975-0080 IJMUE

The administrations and calculations of data are operated through CUDA (Compute Unified Device Architecture)[3], a type of GPGPU (General Purpose computing on Graphics Processing Units) technology developed by the hardware manufacturer company NVidia [3]. CUDA acquired parallel processing technique, C programming language, and GPU(Graphics Processing Unit), which has relatively faster calculation rate compared to CPU(Central Processing Unit), to provide a new method of administering and calculating data of 3D polygonal model. The proposed method in the paper is used to improve data administration speed through GPU. Such improvements in speed not only apply for rendering of 3D polygonal model, but also apply for processing multiple and complicated calculation. Compared to the common rendering method that uses only graphic library, this new method of rendering will possibly accelerate calculation in various areas of 3D environment such as bounding volume test and matrix multiplication.

In the following paper, we will discuss the related studies in Section 2 and characteristics of CUDA in Section 3. We will discuss how to perform 3D polygonal rendering using CUDA in Section 4, compare the result with that of currently standard rendering method that uses only graphic library in Section 5, and discuss conclusions and planned future research in section 6.

2. Related Work

Rendering process that we have discussed in the previous paragraphs is a technique used to make 3-dimensional images from a 2D model by changing geometry, location, and lighting. There are varieties of procedures that can be taken to perform this task. As discussed in the paper, there have also been a lot of studies to use GPU in rendering. As a technique often used to produce high quality rendering, ray tracing technique requires a lots of calculation, and for that that reason, there has been a lots of new studies on using GPU for rendering instead of CPU. Foley *et al.*, [4] has proposed a raytracer that implements GPU and kd-tree to search for the space that that is to be drawn by raytracing. There has also been a new method of 3D polygonal model rendering that uses GPU parallel processing based on CUDA to perform a rendering from information of points.

These studies of using GPU allowed acceleration of calculation tasks, as GPU has relatively more advanced calculation process compared to CPU. Additionally, there have been current studies of rendering by using CUDA, which has supplemented parallel processing technique to GPU's earlier real number calculation ability.

3. Characteristics of CUDA

This paper incorporates CUDA (Compute Unified Device Architecture), a type of GPGPU (General Purpose computing on Graphics Processing Units) technology developed by the hardware manufacturer company nVidia. In addition to the real number calculation ability that the original GPU was known for, CUDA uses thread-based parallel processing to accelerate the calculation speed, and has contributed in improving calculation rate in various areas such as cryptography and physics. As discussed in the previous paragraphs, CUDA has a parallel processing element, implemented by GPU. In other word, it can use parallel thread to accomplish iterative calculations or same calculations all at once. The number of thread block is determined by general size of processed data and number of system processor. Because all the threads within block must be on a common processor, they share limited amount of data. Hence, the number of threads in each block is restricted to 1024 [5, 6]. CUDA bases its calculation on data exchange between the main memory, called host, and GPU memory, called device. Generally, when calculations are operated through CUDA, input

data is first transferred from host to device and then parallel processing is computed using various threads according to the instructions given by CPU. When the result value is transmitted from device to host, the operation is completed. Additionally, even when each thread is being operated, data can be read or used from device memory and data can relatively be sent at a faster rate from device compared to data sent from host.

Architecture designed by CUDA has many benefits but there are also some limitations that must be aware of during programming. As explained earlier, CUDA transfers input data from host to device. Because this design does not allow the user to directly take data from device, data can only be delivered from host to device. As a result, there is a time delay during data transmission, which does not happen in usual processes that only use CPU. Considering such gap in time, time-consuming calculations must be managed by GPU. Due to this time gap, if CUDA is used for all the calculations, there is a possibility that the overall operation may be decelerated compared to when CPU is used [7, 8, 9]. Such data transmission design also indicates that clashes can occur during data delivery process due to a system bus bandwidth and time delay.

There is also a possibility of an overflow if a large quantity of data is copied all at once. When programming codes, it should be considered that operation will proceed through parallel processing carried out by threads when CUDA is used for calculation. If each concurring parallel processing design is not considered when creating codes, it is difficult to predict the data synchronization and renewal. This could lead to flawed results as threads compete against each other when reading and using memory data.

4. Fast 3D Polygonal Model Rendering using CUDA

4.1. Data Administration using CUDA

As stated previously, the first step of CUDA-based operation is transmitting input data from the host memory to the device memory. The proposed method in the following paper is structured to easily administer 3D polygonal model data. Although the proposed structure in the paper seems to be simple in that it only solves for a normal vector of 3D polygonal model, considering the actual structure where calculation is operated in device and the results are received by host, administrating all the data of 3D polygonal model in a single structure can lead to memory wastage because it will send all the structure used in CUDA as a result value. In order to effectively control organization when delivering the calculation result to host, each data must be administered separately to prevent an unnecessary copy of data. Administering data separately, as explained previously, can also prevent overflow of data, which often occur when copying an excessive amount of data all at once. Because data is copied as a whole when copying data from host to device, data must be administered separately in the host memory, just as well as in the device memory. This means that memory must be allocated in both host and device. Memory allocation function cudaMalloc [10, 11] is commonly used to allocate memory in device. In host, function malloc is used, which is a memory allocation function of main memory that is commonly used in C programming language. When it is prepared to read and use memory from both host and device, 3D polygonal model data must first be inputted in host and then be transferred to device. Data transmission uses the function cudaMemcpy [10, 11], a data copying function of CUDA. The function cudaMemcpy can be used for both when copying an input data from host to device and transmitting a result value from device to host.

```
struct Face
{
    int index[3];
};
```

```
struct Normal
{
    double normal[3];
};
```

```
struct Vertex
{
    double coord[3];
};
```

(a)Connection information data struct (b)normal vector data struct (c) peak data struct

Figure 1. Data administration structure

The Figure 1 is a data administration structure of 3D polygonal model data. By individually managing peaks, connection information, and normal vectors calculated through CUDA's parallel processing, all the data would be processed in device and only the essential data would be copied to host, which will assist in minimizing memory leak. GPU- based data administration can also allow the user to access the data relatively faster than CPU-based administration.

4.2. Processing calculations using CUDA

As stated above, when calculation is processed through CUDA, its architecture will be organized to input data to host, calculate the transmitted data in device, and retransmit the result value to host. The calculation is processed in device only, and it will be performed through parallel processing. Calculation processing in CUDA applies kernel function used in GPU. Kernel is mostly analogous to other general functions of C programming language, but it differs in that the number of threads and blocks must be specified when using kernel. When the number of threads is specified, the user can activate the threads through thread ID inside each thread and approach to the input data [12].

The following paper simply tested calculating normal vector for each mesh using 3D polygonal model data. Figure 2 shows an overall flow chart of the test system performed in the paper. When 3D polygonal data is transmitted to device, kernel implements each thread and uses the essential data from 3D polygonal data for normal vector calculation.

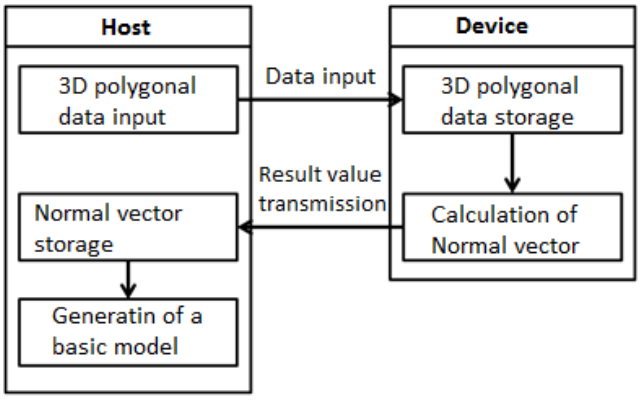


Figure 2. Flow chart

Although the system tested in the paper only tested for normal vector, this system can be used for other time-consuming tasks as well and can accelerate the calculation process through parallel processing. Figure 3 represents a display of the normal vector in the triangle mesh.

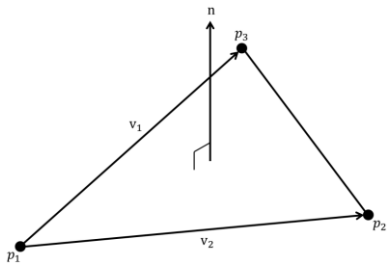


Figure 3. Normal vector of the triangle mesh

As shown in Figure 3, normal vector n is calculated by using the vector V_1 and V_2. It is commonly solved by finding cross product of the two vectors, as shown below.

$$v_1(ax, ay, az) \times v_2(bx, by, bz) = (ax * bz - az * by, az * bx - ax * bz, ax * by - ay * bx)$$

Normal vector is solved for in each triangle mesh. Also, 3D polygonal model data can be used individually. In other words, instead of using general iterative calculations, normal vectors can be calculated simultaneously by using parallel processing. Below figure represents parallel processing architecture of normal vector calculation. As shown in Figure 4, the number of each thread formed is consistent with the number of the planes in 3D polygonal model and each of them independently calculates for its normal vector. Since the normal vector calculation is a relatively fast process, the benefit of saving time does not apply much in this case but a time difference definitely exists between iterative calculation and parallel processing. Although this paper used CUDA to only process normal vector calculation, its time saving element can be maximized if it is used for other vector calculations or other processes used in 3D graphics.

Thread 1	Thread 2
Calculation of Mesh 1's two vectors	Calculation of Mesh 2's two vectors
Calculation of normal vector	Calculation of normal vector
Thread 3	**Thread n**
Calculation of Mesh 2's two vectors	Calculation of Mesh n's two vectors
Calculation of normal vector	Calculation of normal vector

Figure 4. Normal vector calculation using parallel processing

As discussed previously, CUDA processes calculation in device using parallel processing and retransmits the result value to host. The tested result values of the system in the paper represent normal vectors of each triangle mesh that compose of 3D polygonal model processed in GPU. Each normal vector of triangle mesh that was calculated through CUDA must be transmitted to host, or the main memory. CUDA only processes arithmetic function

of GPU and cannot process any operations in device data other than data processing and calculation. After transmitting the result values to host, the resident data in GPU no longer has purpose and is released. Memory release uses the function cudaFree [5, 10, 11] to cancel structure connected by each pointer. After transmitting the result value to host, all the tasks in device are completed and only host is used from that point. After transmitting normal vectors of each triangle mesh as result values, normal vectors are used for 3D polygonal model rendering. The essential data required when rendering 3D polygonal model are peak data, connection information, and normal vector. Peak data and connection information would be used to shape 3D polygonal model and normal vector would be used to add lightening to the model to display a realistic effect.

In a general OpenGL rendering method, if the function glNormal3d [13, 14] and the function glVertex3d [13, 14] are applied to search for all the information to form a 3D polygonal model and if lightening is added to the model, it will display a realistic 3D polygonal model just as shown in Figure 5.

Table 1. Result comparison table

	Model information		Duration of calculations (unit: seconds)				
Processing method	Name of the model	Number of planes	Data input	Data transmission	Normal vector calculation	Result value transmission	Total duration
CPU	Turbine blade	1,765,388	10.7800	-	0.1240	-	11.3880
CUDA			10.6860	0.0460	0.0270	0.0470	11.1540
CPU	Buddha	1,085,634	6.5770	-	0.0810	-	6.9400
CUDA			6.5360	0.0330	0.0171	0.0330	7.1560
CPU	Igea	268,686	1.6380	-	0.0310	-	1.9660
CUDA			1.8050	0.0070	0.0066	0.0080	2.2500
CPU	Bunny	69,451	0.4050	-	0.0160	-	0.7170
CUDA			0.4550	0.0030	0.0013	0.0030	0.8880
CPU	Sphere	24,000	0.1410	-	0.0000	-	0.4520
CUDA			0.1600	0.0000	0.0000	0.0020	0.7850

5. Result

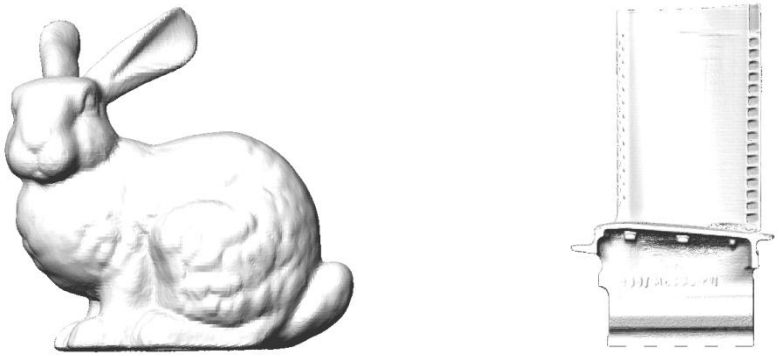
All the experiments tested in this paper were performed in Intel i7 3.2Ghz CPU main memory and Window 7(x64) computer installed with Nvidia Gforce 285 GTX graphic card. For 3D graphic rendering, we have implemented c from OpenGL library and Visual Studio 2008, and have operated CUDA 4.0 SDK for parallel processing.

Table 1 compares the result of the experiment. Most of the total duration barely show any difference and some of them were even decelerated. However, in the normal vector calculation, which was the main purpose of the experiment, it clearly showed acceleration of the operation. For the model TurbineBlade, while the total duration barely changed, the duration of normal vector calculation showed a clear decrease, from 0.1240 seconds when processed through CPU to 0.0270 seconds when processed through CUDA. For the model Buddha model, although its total duration increased, its duration of normal vector calculation dropped from 0.0810 seconds to 0.0171 seconds. For the model Igea, its total duration increased as well but its duration of normal vector calculation was shortened from 0.0160 seconds to 0.0013 seconds. The model Bunny also showed increase in its total duration but its

duration of normal vector calculation was reduced from 0.0160 seconds to 0.0013 seconds. For the last model Sphere, although its total duration was increased and its duration of normal vector calculation could not be measured due to limitations with range, the duration of normal vector calculation was kept under 0.0001 seconds both with CPU and CUDA.

Figure 5 demonstrates the rendered images of 3D polygonal models. Previously, we have illustrated that duration of data transmission exists between host and device. This can be seen in Table 1, as it shows that excluding the model Turbine Blade, rest of the model show deceleration of calculation rate. Such problem occurred because the magnitude of data that was processed through parallel processing of CUDA was not sufficient enough to show a positive result. If the total amount of data was increased to a sufficient amount and if there were more computational processes, the overall performance would be enhanced. Figure 5 shows the result images of rendering.

(a)Sphere(planes: 24,000 faces) (b)Buddha(planes: 1,085,634 faces) (c)Igea(planes: 268,686 faces)



(d)Bunny(planes: 69,451 faces) (e)Turbine blade(planes: 1,765,388 faces)

Figure 5. Result

6. Conclusion and Future Work

In the following paper, we have proposed a method of rendering by administrating data of 3D polygonal model using CUDA. By administering 3D polygonal model data using GPU instead of a standard CPU, it not only greatly improve calculation of normal vectors, but also aid tremendously in acceleration of complicated processes such as calculating curvatures of the surface of 3D polygonal model or ray tracing method, which was discussed earlier in the related studies.

In this paper, we have performed a simple experiment to display a presentation of processing 3D polygonal model using CUDA. Basing this system as a fundamental, we aim to

maximize the efficiency of the calculation rate with various types of rendering processes in the future.

References

[1] R. J. Rost and B. Licea-kane, "OpenGL Shading Language 3/E", Addison-Wesley, **(2009)**.
[2] J. Zink, M. Pettineo and J. Hoxley, "Practical rendering and computation with Direct3D 11", CRC Press, **(2011)**.
[3] https://developer.nvidia.com/category/zone/cuda-zone.
[4] T. Foley and J. Sugerman, "Kd-tree Acceleration Structures for a GPU Ray tracer", In HWWS '05: Proc. of the ACM SIGGRAPH/EUROGRAPHICS Conf. on Graphics Hardware, ACM Press, New York, NY, USA, **(2005)**, pp.15-22.
[5] J. Sanders and E. Kandrot, "CUDA by Example", Addison-Wesley, **(2010)**.
[6] W. -M. Hwu and D. Kirk, "Programming Massively Parallel Processors: A Hands-on Approach", Morgan Kaufmann, **(2010)**.
[7] R. Farber, "CUDA Application Design and Development", Elsevier, **(2011)**.
[8] W. -M. Hwu, "Gpu Computing Gems", Morgan Kaufmann Publ Inc., **(2011)**.
[9] H. Nguyen, "Gpu Gems 3", Addison-Wesley Professional, **(2007)**.
[10] nVidia, "NVIDIA CUDA C Programming Guide", nVidia, **(2011)**.
[11] nVidia, "CUDA API REFERENCE MANUAL", nVidia, **(2011)**.
[12] nVidia, "Getting Started with CUDA", nVidia, **(2008)**.
[13] R. S. Wright jr, "OPENGL SUPERBIBLE 3/E", Sams, **(2005)**.
[14] Hearn, "Computer Graphics with OpenGl 4/E", PearsonEducationAsia, **(2011)**.
[15] W. O. Odoyo, J. -H. Choi, I. -K. Moon and B. -J. Cho,"Silhouette-Edge-Based Descriptor for Human Action Representation and Recognition", Journal of Communication convergence and engineering, vol. 11, no. 2, **(2013)**, pp. 124-131.

Authors

Jihun Kang is a Ph.D. student in Department of Computer Science Education at Korea University, Seoul, Korea. His research interests include GPGPU, parallel processing, computer graphics and distribute data processing.

Syung-Og An is a professor in the Department of Game Engineering at Paichai University, Korea. She received M.S. and Ph.D. in Computer Science & Engineering Department of Korea University, Seoul, Korea, in 1984 and 1989, respectively. She was a visiting professor at the University of Pennsylvania State University from 1993 to 1994. Her research interests include multimedia system, computer graphics and database.

Shin Jin Kang received Ph.D. degree in Computer Science from Korea University in 2011. Since 2003, he has worked at Sony Computer Entertainment Korea and NCsoft as a lead game designer. He is now a professor at the School of Games at Hongik University. He is also the technical advisor of NCsoft.

Seok-Hun Kim is an assistant professor in the Mobile Media at Suwon Women's College. He received the M.S and Ph.D. degree in Computer Engineering from Hannam University in 2003 and 2006, respectively. His teaching and research specialties are in the fields Mobile computing, Web-App programming, information security.

SooKyun Kim (Corresponding Author) received Ph.D. in Computer Science & Engineering Department of Korea University, Seoul, Korea, in 2006. He joined Telecommunication R&D center at Samsung Electronics Co., Ltd., from 2006 and 2008. He is now a professor at Department of Game Engineering at Paichai University, Korea. Dr. Kim has published many research papers in international journals and conferences. Dr. Kim has been served as Chairs, program committee or organizing committee chair for many international conferences and workshops; Chair of ICCCT'11, ITCS'10, HumanCom'10, EMC'10, ICA3PP'10, FutureTech'10, ACSA'09, Em-Com'09, CSA'09, CGMS'09, ISA'09, SIP'08,FGCN'08 and so on. Also Dr. Kim is guest editor of the International Journal of "IET Image Processing" and "Multimedia Tools and Applications". His research interests include multimedia, pattern recognition, image processing, mobile graphics, geometric modeling, and interactive computer graphics. He is a member of ACM, IEEE, IEEE CS, KACE, KMMS, KKITS and KIIT