

Research Article

Accelerating 3D Visualization in Reservoir Modeling System with Programmable Hardware

¹Lin Liu, ²Yumei Ning and ³Desheng Li

¹Software Institute East China Institute of Technology, Nan Chang 330013, China

²School of Computer Science and Technology, Xidian University, Xi'an 710071, Shanxi, China

³School of Science, Anhui Science and Technology University, Fengyang 233100, China

Abstract: This study presents a new method on 3D visualization in reservoir modeling system by using the computation power of modern programmable Graphics hardware (GPU). The proposed scheme is devised to achieve parallel processing of massive reservoir logging data. By taking advantage of the GPU's parallel processing capability, moreover, the performance of our scheme is discussed in comparison with that of the implementation entirely running on CPU. Experimental results clearly show that the proposed parallel processing can remarkably accelerate the data clustering task. Especially, although data-transferring from GPU to CPU is generally costly, acceleration by GPU is significant to save the total execution time of data-clustering and also significantly alleviates the computing load on CPU.

Keywords: Graphics Processing Unit (GPU), massive data, reservoir modeling, 3D visualization

INTRODUCTION

In the past years, there exist many applications of the 3D scientific visualization and acceleration in numerous areas, such as the online game, the 3D scene wander, the flight simulation, virtual operation and so forth. In particular, 3D visualization of complex reservoir modeling is proved to be a highly competitive and important task. In the reservoir modeling, the location and the shape of moving objects should be drawn in real time. The refresh of the frames should not be noticed by users when the viewpoint changed (Haldorsen and Damsleth, 1990; Omre, 1991; Manocha, 2005). Moreover, the scene also should be redrawn to adjust with the action of people. Especially, there are enormous of data should be processed to construct the complex reservoir modeling.

Unfortunately, most of the modeling software can only sustain limited logging data (Dubrule, 1993; Yorozu *et al.*, 1987). The transformation of the model would be too slow to display. At the meantime, the information could lose if we reduce the logging data increasing with the size and dimension of data sets. For the purpose of accelerating the speed for massive data processing, many approaches for parallel data clustering have been proposed (Kruger and Westermann, 2003; Li *et al.*, 2003).

This study presents an effective implementation scheme of 3D scientific visualization of the reservoir modeling system, in which each PC is equipped with a commodity programmable Graphics Processing Unit

(GPU). The proposed scheme is designed to achieve paralleling processing commoditization of modern GPUs, leading to a relatively low price per unit and rapid development of next generation processors.

Graphics Processing Unit and CUDA: GPU's amazing evolution on both computational capability and functionality extends application of GPU to the field of non-graphics computation, which is so-called General Purpose computation on GPUs (GPGPU) (Moreland and Angel, 2003; Trendall and Steward, 2000). Design and development of GPGPU are becoming significant because of the following reasons:

- **Cost-performance:** Using only commodity hardware is important to achieve high computing performance at a low cost and GPUs have become commonplace even in low-end PCs. Due to the hardware architecture designed for exploiting parallelism of graphics, even today's low-end GPU exhibits high-performance for data-parallel computing. In addition, GPU has much higher sequential memory access performance than CPU, because one of GPU's key tasks is filling regions of memory with contiguous texture data (Agarwal *et al.*, 2003). That is, GPU's dedicated memory can provide data to GPU's processing units at the high memory bandwidth.
- **Evolution speed:** GPU's performance such as the number of floating-point operations per second has

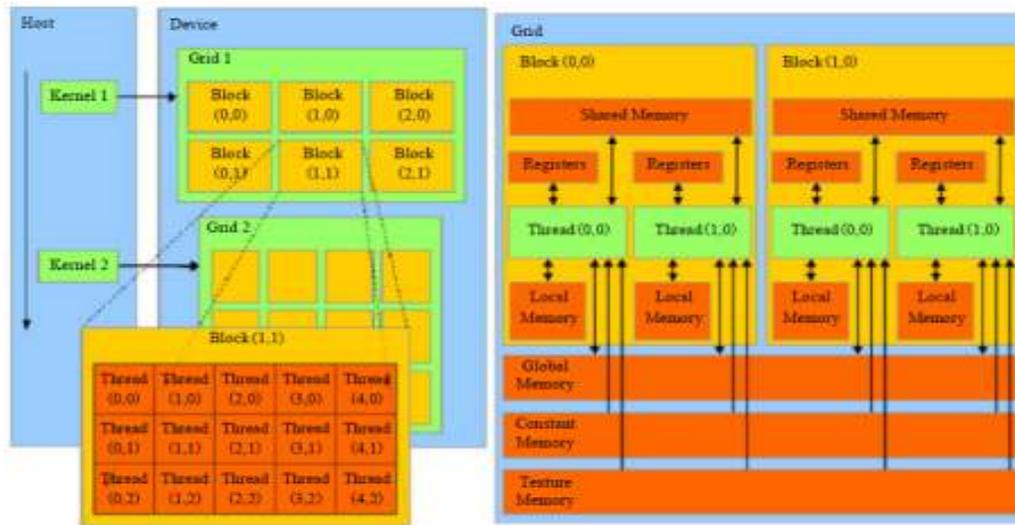


Fig. 1: The CUDA program model

been growing at a rapid pace (Owens *et al.*, 2007). Due to their highly parallel architecture, the programmable pixel pipeline of modern GPUs is capable of a theoretical peak performance that is an order of magnitude higher than CPU. An NVIDIA 7900 GTX 512 has FLOPS rating of around 200 Giga FLOPS compared to a high-end PC, which is capable of around 10 Giga FLOPS. Furthermore, GPU performance has been increasing by a factor of 2 to 2.5 per year, which is faster than the increase in CPU performance as predicted by Moore's law. The high performance-to-cost ratio, rapid increase in performance and widespread availability of GPUs, which can deliver several times the performance of a single CPU, have propelled them to the forefront of high performance computing. The utility of GPUs has expanded beyond traditional graphics rendering.

nVIDIA's CUDA programming guide (Moreland and Angel, 2003) estimates CUDA hardware to be approximately 1000% faster than a Core2Duo. However Sunlight LB's CUDA kernel achieved speed-ups of 149% running on Gee Force 8800GTS and 119% on Gee Force 8800GT in relation to Intel's Core2Duo. Hence, the port seems not to be programmed well enough to exhaust CUDA hardware's power. This is caused by non-optimized memory access patterns of Sunlight LB's core simulation functions from CUDA's point of view (Ji and Wu, 2006). To achieve high performance on CUDA, it is very important for the GPU software to access data as big sequential blocks in GPU's D-RAM memory (Owens *et al.*, 2007).

Traditional CPU software is not that dependent on block data access patterns, as the various caches of a modern CPU absorb this matter in a transparent way. If the desired algorithm is not adaptable to block access,

the software must embed cache-like routines inside the GPU code for block wise loading and unloading of input and output data between GPU's DRAM memory and share memory located on the GPU itself. Then the algorithm can access randomly to data inside the shared memory space without major performance penalty. The CUDA program model is shown as Fig. 1.

THE METHODS OF GEOLOGY STATISTICS

Among the present spatial data interpolation methods, the Kriging method is an optimal interpolation method, with an unbiased interpolated value and minimal estimation variance. Most of the 3D Visualization is using the Methods as spatial data interpolation method (Castrignan and Butta Fuoco, 2004). By using the method of Kriging interpolation in Limited area, continuous reservoir data body can be obtained. Several forms of the Kriging interpolation method exist, such as the simple Kriging method, the ordinary Kriging method, the co-Kriging method, the stratified Kriging method and the nonlinear Kriging method. However, each form has particular characteristics and it is suitable for a specific task. In this study, an ordinary Kriging interpolating approach was used to construct the data body. Supposed that there are k kinds of rock faces (s_1, s_2, \dots, s_k) in the modeling area, we can define variable:

$$I(u) = \begin{cases} 1 & Z(u) \in s_k \\ 0 & Z(u) \notin s_k \end{cases} \quad (1)$$

The probability of being the k faces for any modeling point is: $P(I_k) = 1|Z(u_\alpha) = s_\alpha. \forall \alpha.$ α could be the area which concludes the points, the probability

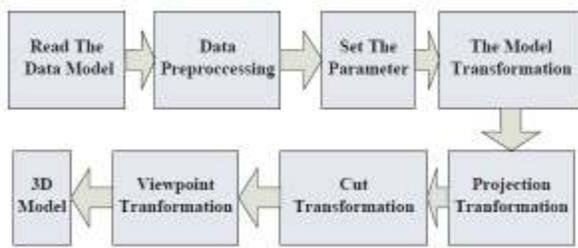


Fig. 2: The common technological process of the 3D visualization model

could be calculated by the formula below, by which the λ_α is the weight coefficient, it could be confirmed:

$$P(I_k = 1 | z(u_\alpha = s_\alpha, \forall \alpha)) = E(I_k) + \sum_{\alpha=1}^n \lambda_\alpha [1 - E(I_k)] \quad (2)$$

Before the geology model can be drawn on the computer, it should go through a series of coordinate transformation. The common technological process is shown in Fig. 2.

Firstly, the mathematical description of the model can be preprocessed after reading it, set appropriate parameters such as length, width, etc. Then viewpoint to observe interested landscape is set. The description of how to observe the 3D model should be present after the construction. According to a series of coordinate transformation, the observation of 3D model can be observed in an appropriate position which is adapted to the viewpoint. In the observation process of 3D model, the observation way is up to the type of the projection transformation, different projection transformation get different 3D scene. The scene of transformation model is cut or zoom in the viewpoint transform which decides the whole 3D model of the image on the screen. 3D visual modeling of Reservoir can be divided into three layers on macroscopic: the data interface layer,

business logic disposal layer and human-computer interaction layer (Fig. 3).

The format of the data set (such as file, database) can be translated in the interface layer, then it would be loaded in the business logic disposal layer for processing to establish reservoir data model. Finally, the model results can be shown in human-computer interaction layer. Among them, the business logic disposal layer and divided into three main steps to complete: the tectonic modeling, sedimentary faces modeling and property modeling (usually in a phased conditions).

Element copy kernel function design: The kernel function operation principle is that the CUDA program, which is designed to be a kernel, could be executed by sending to a grid. A number of blocks are contained in a grid and several threads could be executed by every block. The element copy kernel function declared below:

```
-global-void kernel (float* d-1, float* d-2, float* d-3, float* d-4, float* d-5)
```

The function which is defined by global would be a kernel function and it would be invoked in the host computer. This part could be transformed from CPU to GPU for parallel processing. The logging parameters would be transformed to the GPU by kernel function. The kernel function could be invoked like this:

```
Dim3 grid (5, 5);
Dim3 thread (BLOCKDIM X, BLOCKDIM Y);
Kernel <<< grid, block >>> (d 1, d 2, d 3, d 4, d 5);
```

The grid.x*grid.y is the amount which would be sent, thread.x* thread.y is the amount of threads in every block.

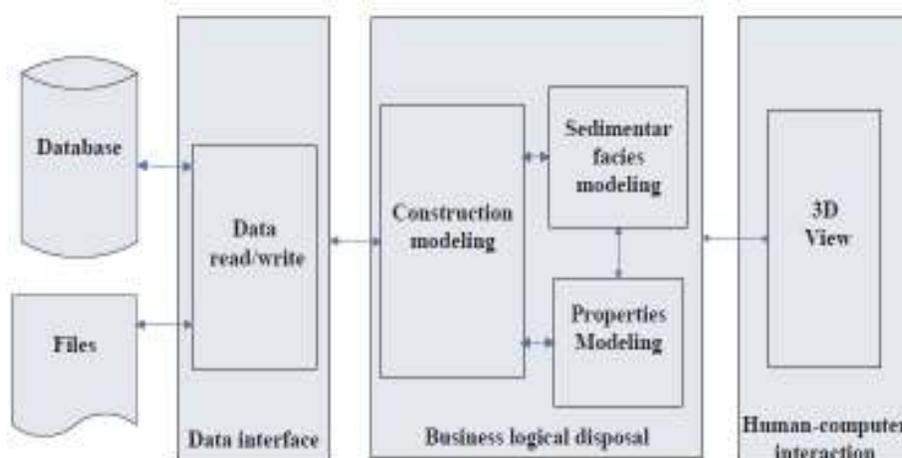


Fig. 3: The common technological process of the geology model

THE CODE DESIGN OF HOST IN 3D VISUALIZATION

The processing results of logging data are not unique. It is not very complex to compare the results and discover the suited results, but the amount of the data involved in the modeling is enormous and a wide range of data could be processed. It would be unrealistic and low efficiency relying on the CPU only. Therefore, the logging data are considered to be arranged as matrix. The data should be mapped in the textures of GPU and the parallel and floating-point calculation could be processed. In our modeling, the data type is single floating-point and the amount of the wells is 968 and the logging data would be kept in two matrixes. The logging data is included in the matrix 1 and the 3D modeling dataset is included in the matrix 2.

The progress of data generating in the CPU could be simple:

```
float* a = (float*) malloc (size of (float)*W*H)
```

The realistic general purpose of GPU needs to show the data by graphics and the mapping methods from matrix to texture should be limited by special grammar and format. Hence, the transformation of the data from CPU (HOST) to GPU (DEVICE) and the malloc of memory would have special format:

```
CUDA SAFE CALL (cuda Malloc (void**) & d 1, sizeof (float)*W*H))
```

```
CUDA SAFE CALL (cuda Memcpy (d 1, a, sizeof (float)*W*H,cuda Memcpy Host to Device))
```

EXPERIMENTAL RESULTS

The software and hardware environment is shown in Table 1. Briefly, the GPU is NVIDIA Gee Force 8400M GS and the CPU is Intel Core2Duo 1.5 GHz, which run in different equipments owning same price. While the 3D visualization displays in the VS2005. The X-coordinate of reservoir modeling area is from 410km to 470km, the Y-coordinate is from 450km to 520km.

In addition, for the purpose of comparing the speed of CPU and GPU, we have designed a method for speed testing. Due to the GPU data processing ability is far higher than the CPU and the display work can be accomplished soon after the compute, we should assign calculation amount to the GPU as much as possible, the CPU only need to translate the data to the GPU. The GPU would send the data to CPU after the parallel computing is finished and calculation results is accomplished. So we just need to record the speed of CPU modeling test.

The 3D modeling results consist of the running time under CPU/GPU and the corresponding speed-up is recorded in Table 2. To explain it more intuitively, Fig. 4 draws the curves illustrating the comparison of running time between the CPU and GPU. The costing time in the CPU is from 44.786 to 72.556 ms, while the

Table 1: Experimental environment and results

Item	System environment
GUP	NVIDIA GeForce 8400 GS
CPU	Intel Core2 duo 1.5 GHz
RAM	2.0 GB
OS	Microsoft windows XP pro SP2
IDE	Microsoft visual studio 2005 visual ++
SDK	NVIDIA CUDA toolkit 2.0beta
	NVIDIA CUDA SDK 2.0beta
	NVIDIA CUDA driver 2.0beta

Table 2 Comparison of 3D modeling runtime in CPU and GPU

No.	CPU (ms)	GPU (ms)	Speed-up
1	56.245	20.542	2.738
2	67.458	29.575	2.280
3	44.786	26.455	1.692
4	60.395	35.343	1.708
5	72.556	32.546	2.229
6	70.370	33.483	2.230
7	65.735	29.476	1.951
8	58.368	38.571	2.230
9	58.465	34.280	1.513
10	57.350	31.439	1.824

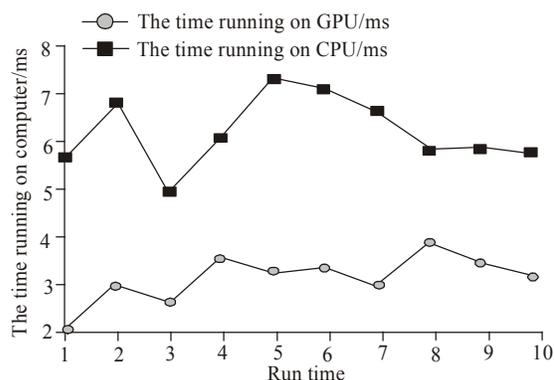


Fig. 4: Comparison of 3D modeling runtime in CPU and GPU

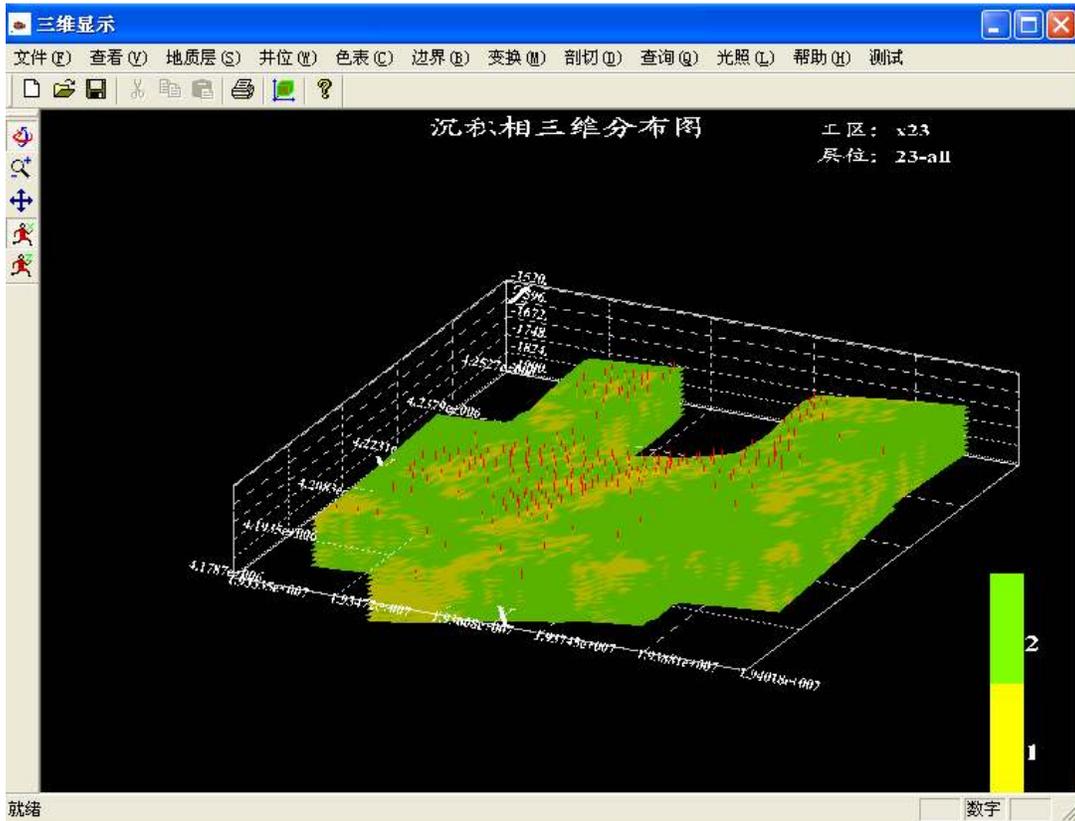
costing time in the CPU is from 20.542 to 38.57ms, respectively. From these data, the acceleration using our approach is obvious and significant.

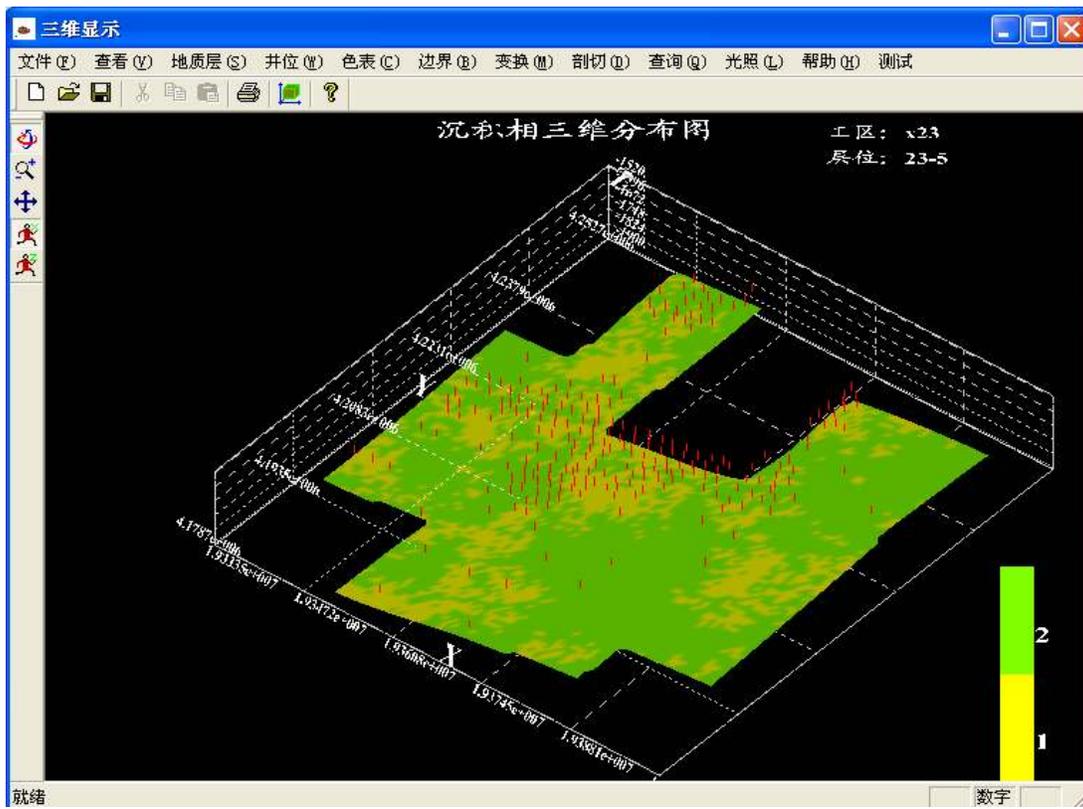
Furthermore, the 3D modeling of restraint area, that with boundary and 1st section are shown in Fig. 5a to c respectively. The results show that the general purpose is suited for the processing of mass data. The logging data has a huge amount in our model and most of them are floating point data, where the display speed would be slow if only relying on CPU. We presented the methods of parallel processing by GPU with the same price to share the calculated amount and then the speed is accelerated. Hence, the general purpose of GPU could be an important and meaningful research.

CONCLUSION AND RECOMMENDATIONS

In this study, the GPU implementation of 3D Visualization in Reservoir Modeling System is discussed compared with the CPU implementation to clarify the performance gain of GPU co-processing.

The results have shown that the approach is efficient and the computing load of CPU is also





(c) The 3D modeling of restraint area with 1st section

Fig. 5: Screenshots of 3D modeling of restraint area

alleviated. Using modern graphics processing units for no-graphics high performance computing is motivated by their enhanced programmability, attractive cost/performance ratio and incredible growth in speed. Although the pipeline of a modern Graphics Processing Unit (GPU) permits high throughput and more concurrency, they bring more complexities in analyzing the performance of GPU-based applications.

In this study, we did not compare the performances of CPU which has more than double cores. Instead, we only restricted our comparison to dual-core. The use of a specific application programming interface might prove the calculation time through the use of a multi-core CPU rather than a dual-core CPU (Spoerk *et al.*, 2007). The multi-threading implementation with effective load balancing between CPU and GPU will be investigated in our future work.

ACKNOWLEDGMENT

This study was supported by the support of Natural Science Fund from Jiangxi Province (No. 20114BAB211026), the Schoolmaster Fund in East China Institute of Technology (No. DHXK0934), the Talent Introduction Special Fund of Anhui Science and

Technology University (No. ZRC2011304) and Science and Technology Plan Project of Chuzhou City (No. 201236).

REFERENCES

- Agarwal, P., S. Krishnan and N. Mustafa, 2003. Streaming geometric optimization using graphics hardware. Proceeding of 11th European Symposium on Algorithms, pp: 544-555.
- Castrignan, A. and G. Butta Fuoco, 2004. Geostatistical stochastic simulation of soil water content in a forested area of south Italy. Biosyst. Eng., 87(2): 257-266.
- Dubrulle, O., 1993. Introducing more geology in stochastic reservoir modeling. Quant. Geo. G., 5: 351-369.
- Haldorsen, H. and E. Damsleth, 1990. Stochastic modeling. J. Petrol. Technol., 42(4): 404-412.
- Ji, J. and E. Wu, 2006. View-dependent re_nement of multiresolution meshes using programmable graphics hardware. Visual Comput., 22: 424-433.
- Kruger, J. and R. Westermann, 2003. Linear algebra operators for gpu implementation of numerical algorithms. ACM Trans. Graph., 22: 908-916.

- Li, W., X. Wei and A. Kaufman, 2003. Implementing lattice boltzmann computation on graphics hardware. *Vis. Comput.*, 19(7-8): 444-456.
- Manocha, D., 2005. General-purpose computations using graphics processors. *Computer*, 38(8): 85-88.
- Moreland, K. and E. Angel, 2003. The FFT on a GPU. *Proceedings of the SIGGRAPH/ Eurographics Workshop on Graphics Hardware*. San Diego, pp: 112-119.
- Omre, H., 1991. Stochastic models for reservoir characterization. In: Kleppe, J. and S.M. Skjaeveland (Eds.), *Recent Advances in Improved Oil Re-covery Methods for North Sea Sandstone Reservoirs*. Norwegian Petroleum Directorate, Stavanger, Norway.
- Owens, J.D., D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn and T.J. Purcell, 2007. A survey of general-purpose computation on graphics hardware. *Comput. Graph. Forum*, 26(1): 80-113.
- Spoerk, J., H. Bergmann, F. Wanschitz, S. Dong and W. Birkfellner, 2007. Fast DRR splat rendering using common consumer graphics hardware. *Med. Phys.*, 34: 4302-4308.
- Trendall, C. and A.J. Steward, 2000. General calculations using graphics hardware, with applications to interactive caustics. *Proceedings of Eurographics Workshop on Rendering*. Springer, pp: 287-298.
- Yorozu, Y., M. Hirano, K. Oka and Y. Tagawa, 1987. Electron spectroscopy studies on magneto-optical media and plastic substrate interface. *IEEE T. J. Magn. Japan*, 2: 740-741.