Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 1402 – 1411

Procedia Computer Science

ICCS 2013 - ”Computation at the Frontiers of Science”

Estimation of Volume Rendering Efficiency with GPU in a Parallel Distributed Environment

Cristian Federico Perez Monte[a,b,c], Fabiana Piccoli[b], Cristian Luciano[d], Silvio Rizzi[d], Germán Bianchini[c], Paola Caymes Scutari[c]

[a]*GridTICs-Universidad Tecnologica Nacional Regional Mendoza, Mendoza, Argentina*
[b]*LIDIC-Universidad Nacional de San Luis, San Luis, Argentina*
[c]*LICPaD-Universidad Tecnologica Nacional Regional Mendoza, Mendoza, Argentina*
[d]*Department of Mechanical and Industrial Engineering-University of Illinois at Chicago, Chicago, IL, USA*

Abstract

Visualization methods of medical imagery based on volumetric data constitute a fundamental tool for medical diagnosis, training and pre-surgical planning. Often, large volume sizes and/or the complexity of the required computations present serious obstacles for reaching higher levels of realism and real-time performance. Performance and efficiency are two critical aspects in traditional algorithms based on complex lighting models. To overcome these problems, a volume rendering algorithm, *PD-Render_intra* for individual networked nodes in a parallel distributed architecture with a single GPU per node is presented in this paper. The implemented algorithm is able to achieve photorealistic rendering as well as a high signal-to-noise ratio at interactive frame rates. Experiments show excellent results in terms of efficiency and performance for rendering medical volumes in real time.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
Keywords: Volume Rendering; Monte Carlo Method; Ray Tracing; GPU; Scientific Visualization; Cluster Computing; Parallel Processing; efficiency;

1. Introduction

Visualization methods of medical imagery based on volumetric data -obtained from magnetic resonance imaging (MRI), computed tomography (CT) scanners, and other techniques- constitute a fundamental tool for medical diagnosis, training and pre-surgical planning.

Volume rendering is a set of techniques and methods to obtain a two dimensional image from volumetric data. Ideally, these must be high-quality images, demanding algorithms that must provide sufficient level of detail to obtain photorealistic results in an adequate computational time. Often, large volume sizes and/or the complexity of the required computations present serious obstacles for reaching higher levels of realism and real-time performance.

*Cristian Federico Perez Monte. Tel.: +54-261-428-1093 ; fax: +0-000-000-0000 .
E-mail address: cristian.perez@gridtics.frm.utn.edu.ar.
E-mail address: cdlt23@gmail.com.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.307

There are different High-Performance Computing (HPC) techniques to improve the performance of these applications. One of them consists of using a parallel distributed architecture with a single Graphics Processing Unit (GPU) per node. The computational power of these systems is able to achieve photorealistic rendering as well as a high signal-to-noise ratio at interactive frame rates. Two important issues to be considered are inter-node and intra-node work. In the former, it is important to study how to efficiently divide the total work and how tasks are assigned to different nodes in the system. In the latter, the work assigned to a node must take advantage of all the available resources in each GPU. This paper presents a modification of the approach in [1] for a distributed system with GPUs. We have focused on improving the efficiency and resource usage of each node in the system. In addition, we present an analysis of the maximum theoretical efficiency of GPUs for a volume rendering process in a graphics distributed system with GPUs.

The paper is organized as follows: Sections 2.1, 2.2 and 2.3 describe related work and some fundamental concepts used in this work, Section 3 introduces the overall *PD-Render* and details *PD-Render_intra*, Section 4 analyses and establishes the performance parameters and in section 5, we show experimental results. Finally, the conclusions and future work are presented.

2. Background and Related work

In this section, we present some fundamental concepts along with related work from other researchers.

2.1. Light Propagation Models

There are many optical models of Direct Volume Rendering (DVR)[2][3]. The next list shows some of them and their main characteristics:

- *Absorption Only*: The volume is assumed to consist of cold, perfectly black material that may absorb incident light. No light is emitted or scattered.
- *Emission Only*: The volume is formed by gas that only emits light but is completely transparent. Absorption and scattering are neglected.
- *Emission-Absorption Model*: It is the most common model in volume rendering. The gas can emit light and absorb incident light, but scattering and indirect illumination are disregarded.
- *Single Scattering and Shadowing*: This model includes single scattering of light from an external light source. The shadows are modeled taking into account the light attenuation that is incident from an external light source.
- *Multiple Scattering*: The goal of this method is to evaluate the complete illumination model for volumes, including emission, absorption, and scattering.

The Emission-Absorption model is the most widely used because it provides a good compromise between generality and performance of computation. However, its main disadvantage is that its results do not have the expected level of quality.

Single Scattering and Shadowing provide better level of quality at the expense of a much higher amount of processing. Recently, research and improvement of volume rendering techniques with illumination has grown significantly, e.g. shadows [4], ambient occlusion [5], global illumination [6], realistic scattering [7][8] and depth of field [9].

In contrast to many existing approaches, Monte Carlo rendering algorithms are capable of processing different materials, lighting, and camera configurations simulating complex light interaction with high accuracy and generating photorealistic images. However, rendered frames with a lower number of iterations show larger errors compared to fully converged rendered frames. Therefore, they also have a low signal-to-noise ratio.

Using the Monte Carlo method to render in GPU results in good interactivity and efficiency [10]. In [1], the authors implement multiple visualization improvements in a novel framework. Their work proposes a GPU solution for visualization in medical environments using Single Scattering and Shadowing combined with Hybrid Scattering.

Although [1] uses all GPU power with the most common resolutions, this is not enough to achieve a visualization with a high signal-to-noise ratio and a good interactivity for all cases. Our work extends [1] using intra-node optimizations to improve performance and visualization of the output images.

2.2. A Parallel Distributed Environment

High-performance computing (HPC) is the use of parallel or distributed processing for solving complex computational problems and improving their efficiency, reliability and the execution time. A Parallel Distributed Environment provides mechanisms for exploiting the inherent parallelism in many scientific and engineering applications. Among different parallel distributed systems, a *Graphic Distributed System* is formed by multiple independent units consisting of a CPU-GPU combination. All CPU-GPU systems are connected through a high performance network.

A CPU-GPU computing system consists of two basic components: (i) the traditional CPU (with one or more processors or cores) and (ii) one or more GPUs (Streaming Processor Array). The GPU can be considered as a manycore processor able to support fine grain parallelism, where a large number of threads run in parallel, each contributing to the solution of a given problem [11][12].

GPUs are different than other parallel architectures, providing flexibility in the local resources allocation to its threads. In general, a GPU multiprocessor consists of several stream multiprocessors with multiple processing units, registers and on-chip memory. In this work we have used the Compute Unified Device Architecture environment by Nvidia [13] to develop our GPU applications.

2.3. Distribution Techniques in Graphics Computing

In parallel distributed rendering it is important to determine how the distribution is made, specially in real time applications. The Molnar's taxonomy [14] defines sorting methods to distribute the work among the different nodes of a distributed system. Distribution can be classified as *Sort First*, *Sort Last* or *Sort Middle*.

In the *Sort First* or *2D* distribution the division is made according to the output image, which is divided in disjoint regions. A pre-transformation is done during geometry processing to determine which regions of the output are covered. Initially, the primitives are assigned arbitrarily to be later redistributed over the network to the correct renderer (processor). Each node performs the work of the entire pipeline for that primitive. Generally, *Sort First* is adequate for Image Order rendering algorithms [2]. There are many parallel implementations of *Sort First*[15][16]

On the other hand, in the *Sort Last*, or (*DB*) method, the division depends on the input volume, which is divided into subsets. Each subset is distributed to different processors (renderers) in the system. The renderers operate independently until the visibility stage, with each parallel task computing pixel values for its subset, independently of the pixel locations in the final image. These pixels are transmitted over an interconnected network to compositing processors which resolve the visibility of pixels from each renderer. It is worth mentioning that the interconnect network must handle all of the pixel data generated in every processor. Therefore, for interactive or real-time applications rendering high-quality images, *Sort Last* could require very large data communication rates. Moreover, as the image resolution grows, the compositing overhead also grows. *Sort Last* is adequate to Object Order rendering algorithms and it is also a good option for large volumes [17].

Lastly, in *Sort Middle* the primitives are redistributed in the middle of the rendering pipeline, between geometry processing and rasterization.

Another technique is Alternate Frame rendering (AFR). It is commonly used in environments consisting of one PC and multiple GPUs [18], rendering multiple frames at the same time, and then alternating the display of the frames on a single monitor, to accelerate the rendering performance. The distribution is made at the frame level. AFR has good scalability but the latency between the input and final visualization may be high. Even though [19] shows a better performance than *Sort First*/*Sort Last* method, the latency problem can be important in real time applications.

In this work we use the *Sort First* technique based on considerations of low latency and the properties of the volumes used in this project.

3. Parallel Distributed Volumen Rendering System

We have developed a *Parallel Distributed Volume Rendering System* (*PD-Rend*). It works in a graphics distributed environment and converts a volume in a high resolution image. *PD-Rend* has been designed to achieve photorealistic rendering as well as to provide a high signal-to-noise ratio. In *PD-Rend*, the distribution is made

using the *Sort First* method (inter-node relation). Each node (intra-node relation) uses the *Single Scattering and Shadowing model* combined with *Hybrid Scattering*, and the Monte Carlo method to improve the signal-to-noise ratio and user interactivity. We have divided *PD-Rend* in two components: (i) *PD-Rend_inter*, which considers the inter-node relation, and (ii) *PD-Rend_intra*, which takes into account the intra-node relation. In this work, we have focused on analyzing the intra-node relation (i.e. *PD-Rend_intra*)

In a distributed system, each node renders a portion of an image according to the *Sort First* distribution model. The resolution per node is a fraction of the whole image. The Kroes' approach [1] does not show a good efficiency for low resolutions. In those cases, the frame rate increases but it is not proportional to the reduction in resolution, impacting GPU efficiency. Accomplishing maximum use of node resources was one of the design goals for *PD-Rend_intra*.

Also, there are two design constraints for *PD-Rend_intra*. Firstly, it must obtain the best performance when nodes with massively parallel architecture as GPUs are used. Secondly, as part of a distributed system, it must speed up the convergence of the Monte Carlo method to achieve high frame rate and high signal-to-noise ratio.

3.1. Rendering Process Stages

Reference [1] proposes a method that obtains excellent performance and quality in GPU. As previously mentioned, our rendering process is based on their approach. The rendering process is divided into different stages which are iteratively repeated frame by frame. Figure 1 shows the different stages in the rendering process, namely:

- *Stochastic Raycasting Stage*: This process yields a High Dynamic Range (HDR) Monte Carlo estimate of the light arriving at the vision plane.
- *Monte Carlo Integration Stage*: It is performed by computing the cumulative moving average.
- *Tone Mapping - Gamma Correction Stage*: Used to generate an image of low dynamic range for visualization from high dynamic range images.

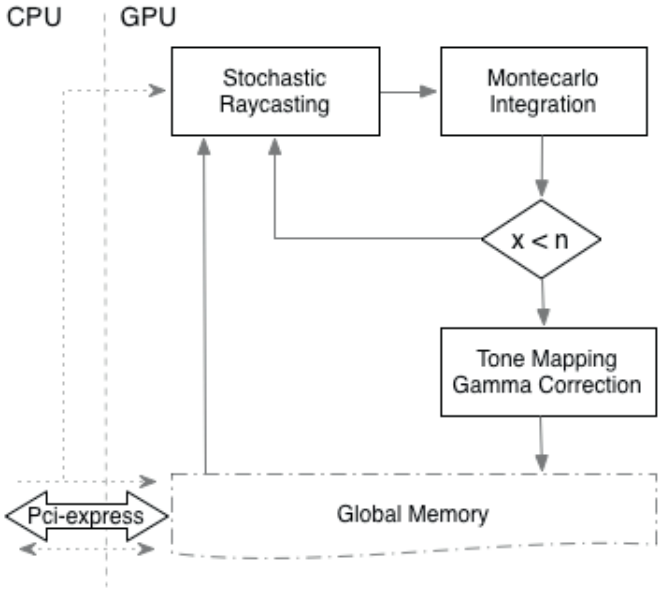


Fig. 1. Stages of Rendering Process

The CPU and GPU are connected through the PCI-express bus. There is an initial data transfer where the volume and initial rendering variables are sent from CPU memory to GPU global memory. Once frames are rendered, they are sent from GPU global memory to CPU memory. As the first rendered frames are low quality, *PD-Rend_intra* only transfers those frames with high quality, i.e., those frames that are the result of *n* iterations

of *Stochastic Raycasting Stage* and *Monte Carlo Integration Stage*, and one *Tone Mapping - Gamma Correction Stage*.

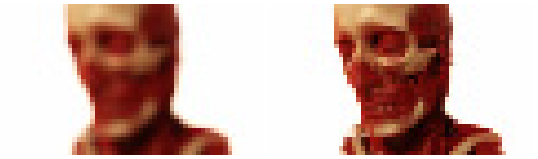
3.2. n-Iterative Process

PD-Rend_intra has an iterative part that repeats the *Stochastic Raycasting Stage* and *Monte Carlo Integration Stage n* times (See figure 1). Once the loop is finished, the *Tone Mapping - Gamma Correction Stage* is executed and the computed frame is transferred to CPU memory. This method improves the signal-to-noise ratio, which is increased according to the square root of *n* [20].

The efficiency of *PD-Rend_intra*, especially for low resolutions, is increased by reducing unnecessary operations inside the loop. This is achieved by moving the *Tone Mapping - Gamma Correction Stage* and the GPU-CPU transfer outside the loop. It is crucial to properly design the number of iterations *n*. If *n* is very small, the quality of the output frame is low. If n is large, the quality is high but the frame rate is low, reduced in a factor of *n*.

We considered other optimizations related to the GPU-CPU data transfers and the use of a filter. Data transfers from GPU to CPU demand a relatively long time where the GPU must stay idle, and consequently, efficiency is reduced. Therefore, we have implemented asynchronous buffered copies and overlapping transfers with computations. Also, rendering variables are transferred only when their values change. Regarding the use of a filter, the Kroes' approach uses a set of filters to improve the visualization when the signal-to-noise ratio is low.

However, the use of a filter reduces significantly the image quality. Figure 2(a) shows an example of an image on which the filter has been applied. Figure 2(b) shows the same image without applying the filter. As *PD-Rend_intra* has been designed to generate images with a high signal-to-noise ratio, filtering stages are eliminated to avoid their impact on image quality.



(a) With Filter (b) Without Filter

Fig. 2. Rendered Image

4. Efficiency Analysis

In this section, we derive the equation that characterizes the efficiency of *PD-Rend_intra*. We use it to analyze the Kroes' implementation and our optimizations.

4.1. Efficiency Equation

From Figure 2, we can analyze the stages of *PD-Rend_intra* as follows:

- The Monte Carlo's rendering time, stages 1 and 2, is directly related to the frame resolution and computing power of the GPU. If the rendering variables do not change, the execution times of every Monte Carlo's iteration are similar. This property allows us to make a balanced load distribution among system nodes. In consequence, the total time of Monte Carlo processing is equal to the time of an iteration multiplied by the number of iterations.
- Once the image portion has been obtained with an acceptable noise level in a high dynamic range (HDR), it is converted, for its representation, to an image with low dynamic range (LDR). This is done in stage 3 and its running time is influenced by the frame resolution and the computing power of the GPU.
- Data transfer times are also important. We distinguish three data transfer times: (i) the time it takes for GPU-CPU transfer of a rendered image portion (stage 4), (ii) the time for CPU-GPU transfering of new rendering variables, and (iii) the time required for kernel initialization. In all cases, times are influenced by the PCI-Express bus latency and speed. In addition, (i) is also affected by the frame resolution.

For all these observations, we define the following variables:

- T_p: Useful processing time of Monte Carlo rendering in once iteration
- n: Number of iterations
- m: Number of nodes in the distributed system.
- T_t: Time to transform an image from HDR to LDR.
- T_{ci}: Time to copy a rendered image portion from GPU to CPU memory.
- T_{ca}: Time of other communication and kernel initialization.
- T_c: Total time of communication for each rendering frame.

As we use a modern GPU architecture where $T_p \gg T_{ci}$ and we implement an overlap between GPU-CPU transfers and the next computing (for this, we need to use buffers), T_{ci} is ignored. In consequence, T_c is calculated for Kroes' approach as $T_c = T_{ci} + T_{ca} = T_{ca}$ and for *PD-Render_intra* as $T_c = T_{ca}$.

We define the Efficiency (E) of the rendering process as the ratio of total rendering time and total execution time, it is

$$E = \frac{n * T_p}{n * T_p + T_t + T_c} \tag{1}$$

Generally, E is smaller than the occupation percentage of GPU because the GPU power is used for the rendering process and Tone Mapping-Gamma Correction. Thus, when n is large, the GPU occupation is comparable to E.

Other interesting performance parameters of *PD-Render_intra* are the number of rendered frames per second (FPS) and the number of iterations per second (IPS). Also, IPS is determinant of image quality.

As previously mentioned, we have designed *PD-Render* considering a *Sort First* as its distribution model. Therefore all nodes generate a stream of screen portions with the same frame-rate to each other and the same frame-rate that final stream of complete. The FPS and IPS and can be calculated respectively as

$$FPS = \frac{1}{n * T_p + T_t + T_c} \tag{2}$$

$$IPS = \frac{n}{n * T_p + T_t + T_c} \tag{3}$$

From the above, we can deduce that when n is large, the output image has less noise and E is better, but the frame-rate is reduced. The following expression relates the frame-rate and E from equations 1 and 2

$$E = 1 - FPS * (T_t + T_c) \tag{4}$$

Thus, the efficiency has two characteristics, (i) it is dependent on the FPS, the communication times, and preparation of the output image; and (ii) it is independent on the frame rendering time. Therefore, it is important to minimize T_t and T_c. T_t is dependent on GPU power and T_c is determined by the time required for kernel initialization and PCI-Express transfers.

If we increase the number of nodes for a specific output resolution and a particular signal-to-noise ratio defined by n, then the resolution is decreased for each node, then T_p and T_t are reduced and can be neglected. In this case, for a large number of nodes ($nodes \gg$), the maximum efficiency per node is expressed as:

$$E_{nodes\gg} = 1 - FPS * T_c \tag{5}$$

For small resolutions in each node, we may obtain an unnecessarily high frame-rate and a reduced node efficiency with risk to saturate the network. A good solution in this case is to increase n and consequently increase the efficiency and signal-to-noise ratio of each frame.

Besides, when there is a large number of nodes, T_c plays an important role as the limiting factor of node efficiency and the whole system.

4.2. Estimation of Times

In order to estimate efficiency, it is important to determine T_c and T_t. If we considered n large enough ($n \gg$), we can approximate T_p as ($T_p(approx)$)

$$T_p(approx) = \frac{1}{(n * FPS_{n\gg})} \quad (6)$$

Then, if $n = 1$ we get

$$T_t + T_c = \frac{1}{FPS_{n=1}} - T_p \quad (7)$$

From these values, we can obtain the maximum efficiency and frame rate for a specific resolution using equation 4. When the rendering resolutions are smaller, T_t is proportional to the resolution and it becomes negligible compared with T_c. In this case it can be depreciated. In equation 6, we express T_p and from the following equation, we determine T_c.

$$T_c = \frac{1}{FPS_{(n=1\&reduced\ resolution)}} - T_p \quad (8)$$

From T_c and using equation 5, it is possible to obtain the theoretical maximum efficiency for a particular frame rate and for any resolution and number of nodes.

From the practical measurement of FPS (for large n) and FPS (n=1), we have obtained every time involved in *PD-Render_intra* and, from them we can establish the efficiency and maximum performance for a GPU.

5. Experimental Results

In this section we present and analyze experimental results for *PD-Render_intra*. We consider two parameters: FPS and $GPULoad$. This analysis was performed in two different scenarios, each configuration is:

Table 1. Scenarios Description of Experimental Setup

Scenario	CPU	GPU
Sc I	AMD FX 8120 RAM 8GB	GTX 560 TI - SPs: 384 Shader/Clock/Memory freq: 1.76Ghz/880Mhz/1.05Ghz
Sc II	AMD Phenom II X2 545 RAM: 4 GB	GTS 250 - SPs: 128 Shader/Clock/Memory freq: 1.836Ghz/740Mhz/1.1Ghz

The testing was performed in OS Microsoft Windows 7 (64-bits version) and Nvidia Drivers v306.97. Medical Data Sets for testing are obtained from Osirix Imaging Software [21]. The model utilized is called Manix and it has been obtained from a CT scan. Its volume was resampled at 50% resulting in a volume of 256x230x256 voxels.

Each reported value is the average of the frames per second when the program runs for a relatively long time. GPU load was measured with GPU-Z v0.6.2 [22].

First, we present our performance evaluation. Table 2 show FPS-IPS and *GPU Load* parameters for Kroes' approach and *PD-Render_intra* in the two scenarios. The number of iterations is 1 as considered in [1]. The results show that the Kroes' approach increases FPS when the resolutions are reduced, but the reduction is not proportional. Furthermore, the available computing power is not fully utilized, caused by an intra-node problem.

Table 3 displays the FPS, IPS and $LoadGPU$ of *PD-Render_intra* in the two scenarios and for a different number of iterations, n.

From table 3, we can observe that the frame rate is reduced proportionally to the increase of n, the number of rendered frames is n times lesser, and there is a better signal-to-noise ratio. Moreover, we notice a higher $GPULoad$ when the resolutions are low, indicating a better usage of GPU computing power.

The table 4 shows the T_p using equation 6.

As previously mentioned, table 4 shows that T_p is proportional to the number of pixels of the output image.

T_c is determined by equation 8. The obtained values for each implementation (Kroes and *PD-Render_intra*) are respectively T_c=3.98 ms and T_c=T_{ca}=922 μs in ScI and T_c=1.96 ms and T_c=T_{ca}=510 μs in $ScII$. The observed differences are due to optimizations in *PD-Render_intra*: including communication, overlapping technique and use of buffers. We also notice that $ScII$, using less capable GPUs, shows a smaller T_c than ScI with more

Table 2. Kroes' Implementation vs. *PD-Render_intra* in *ScI* and *ScII*

		Kroes' Implem. - n=1		PD-Render_Intra - n=1	
Scenario	Resolution	FPS-IPS	Load GPU	FPS-IPS	Load GPU
Sc I	1920x1080	7.8	92%	9.8	98%
	960x540	28.41	86%	37.18	92%
	480x270	80.3	66%	118	81%
	240x135	151.7	40%	299.8	62%
	120x67	197.0	25%	474.3	43%
	60x34	228	18%	756.4	37%
Sc II	1920x1080	1.233	96%	1.516	99%
	960x540	4.61	96%	5.95	97%
	480x270	16.6	93%	21.23	94%
	240x135	52.88	83%	72.61	90%
	120x67	134.9	71%	211.0	80%
	60x34	301.8	64%	537.6	75%

Table 3. PD-Render_intra in two scenarios for a different number of iterations

		PD-Render_Intra - n=10			PD-Render_Intra - n=100		
Scenario	Resolution	FPS	IPS	Load GPU	FPS	IPS	Load GPU
Sc I	1920x1080	1.060	10.60	99%	0.1079	10.79	100%
	960x540	4.03	40.3	99%	0.4144	41.44	99%
	480x270	15.08	150.8	97%	1.53	153	99%
	240x135	48.05	480.5	92%	5.35	535	98%
	120x67	106.7	1067	84%	12.7	1270	97%
	60x34	192.5	1925	77%	24.98	2498	96%
Sc II	1920x1080	0.1735	1.735	100%	0.01735	1.735	100%
	960x540	0.69	6.9	99%	0.0702	7.02	100%
	480x270	2.58	25.8	99%	0.263	26.3	99%
	240x135	9.16	91.6	97%	0.945	94.5	99%
	120x67	28.75	287.5	94%	3.1	310	97%
	60x34	68.93	689.3	91%	7.36	736	95%

powerful GPUs. Modern GPU architectures have higher latencies, which increases T_c and reduces efficiency. The GPU parameter $Cp_{one-one}$ [23] illustrates this problem.

Determining the maximum efficiency and frame rate from equation 4 implies to calculate spent time in other tasks than rendering process. Table 5 shows that extra time for both implementations in our two scenarios.

For these results, we can observe that $T_c + T_t$ is influenced by the image resolution, because the time T_t uses the GPU and is proporcional to the total number of pixels (i.e. x-resolution * y-resolution) of the output image.

From the above, we can determinate the efficiency of *PD-Render_intra*. In consequence, figure 3 shows the achieved E in both scenarios and for different n. Figure 3 also shows efficiency for the Kroes' approach.

We observe that *PD-Render_intra* outperforms Kroes' approach for large n and small resolutions, e.g. IPS of *PD-Render_intra* in *ScI* for n=100 and smaller resolution is ten times higher that Kroes' approach. The same can be noted for GPU Load (see tables 2 and 3). Also, we can see that *PD-Render_intra* has a better efficiency for every value of n and for all resolutions.

We have also analyzed the quality of the output images. In this case, we compare the output image of Kroes' implementation (using filters) and of *PD-Render_intra*. The figures 4 and 5 show six images, where three of them are the output of Kroes (figure 4) and the other three correspond to *PD-Render_intra* (figure 5). In both cases, high, medium and low image resolutions are considered.

The use of a filter improves the quality of the real-time visualization. However, its quality is severely degraded

Table 4. Calculation of T_p

Scenario \ Resolution	1920x1080	960x540	480x270	240x135	120x67	60x34
Sc I	93.02 ms	24.13 ms	6.54 ms	1.87 ms	785 μs	400μs
Sc II	576 ms	142 ms	38 ms	10.58 ms	3.22 ms	1.35 ms

Table 5. Time spent in tasks other than the Rendering Process

Scenario	Implementation	1920x1080	960x540	480x270	240x135	120x67	60x34
Sc I	Kroes	35.2 ms	9.48 ms	5.91 ms	4.72 ms	4.29 ms	3.98 ms
	PD – Render_intra	9.02 ms	2.76 ms	1.93 ms	1.46 ms	1.32 ms	922 μs
Sc II	Kroes	235 ms	74.9 ms	22.24 ms	8.33 ms	4.19 ms	1.96 ms
	PD – Render_intra	83.63 ms	26.06ms	9.1 ms	3.19 ms	1.52 ms	510 μs

when the image has a low resolution, as shown in Figures 5(c) and 4(c). These images were obtained for n=25,000.

6. Conclusion and Future Work

In this paper, we present *PD-Render_intra*, as part of *PD-Render*, a *Parallel Distributed Volume Rendering System*. *PD-Rend_intra* takes into account the intra-node relation and its efficiency is analyzed for different GPUs.

According to ours results, we observe that T_t and T_p are proportional to the total number of pixels of the output image. This is due to the nature of the applied algorithms (Image-Order algorithm). The *GPULoad* is close to its optimal value but it decreases when T_{ca} or T_c are not negligible with respect to T_t+T_p. This occurs for small image resolutions and $n = 1$. Our optimizations to the Kroes' approach allow us to reduce T_t and T_c and to improve the efficiency and performance of each node in the system. In future, we plan to integrated *PD-Render_intra* into the overall *PD-Render* followed by a thorough performance analysis. In this context, we will investigate whether there are other important performance parameters and how to compute them.

Acknowledgements

We would like to thank the UNSL, UTN (LICPaD and GridTICs) and UIC for allowing us to use their computational resources. This research has been partially supported by Project PICT2010/29 and PROICO-30310.

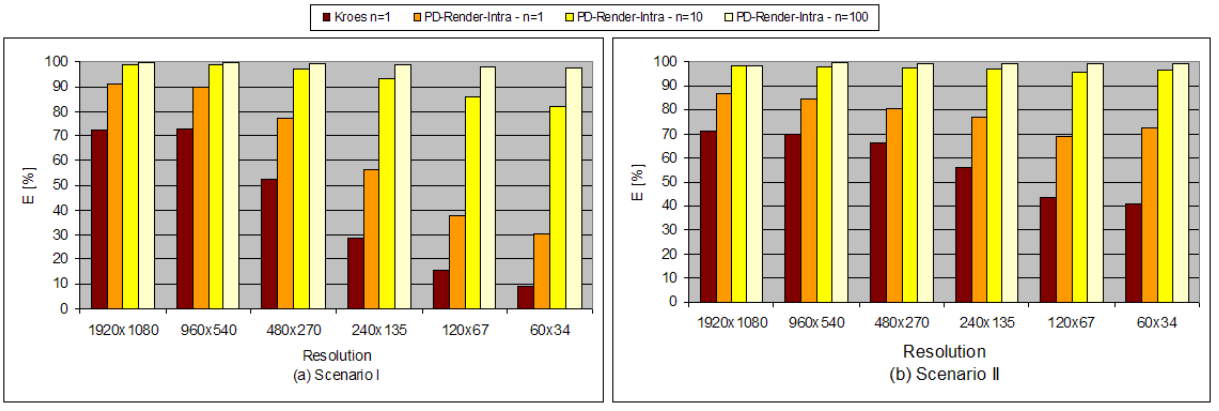
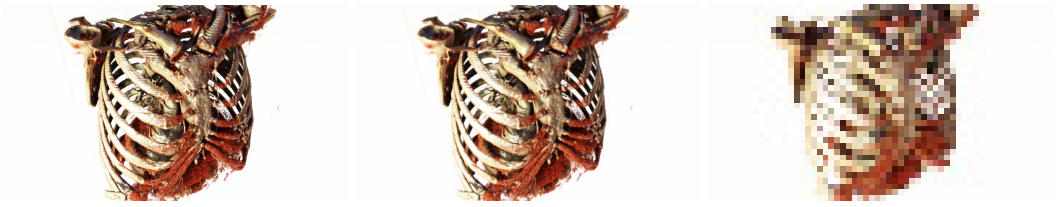


Fig. 3. Efficiency of Rendering Process

(a) High (b) Medium (c) Low

Fig. 4. Images Rendered using Kroes' method



(a) High (b) Medium (c) Low

Fig. 5. Images Rendered using *PD – Render_intra* method

References

[1] T. Kroes, F. Post, C. Botha, Exposure render: An interactive photo-realistic volume rendering framework, PLoS ONE 7 (2012) e38586.
[2] M. Hadwiger, J. Kniss, C. R. Salama, D. Weiskopf., K. Engel, Real-time Volume Graphics, A. K. Peters, Ltd., Natick, MA, USA, 2006.
[3] N. Max, Optical models for direct volume rendering, Visualization and Computer Graphics, IEEE Transactions on 1 (2) (1995) 99 –108.
[4] M. Hadwiger, A. Kratz, C. Sigg, K. Bühler, Gpu-accelerated deep shadow maps for direct volume rendering, in: Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, GH '06, ACM, New York, NY, USA, 2006, pp. 49–52.
[5] S. Zhukov, A. Iones, G. Kronin, An ambient light illumination model, in: Rendering Techniques, Eurographics, 1998, pp. 45–56.
[6] K. M. Beason, J. Grant, D. C. Banks, B. Futch, M. Y. Hussaini, Pre-Computed Illumination for Isosurfaces, in: VDA '94: Proceedings of the conference on Visualization and Data Analysis '06, 2006, pp. 1–11.
[7] C. R. Salama, Gpu-based monte-carlo volume raycasting, in: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications, PG '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 411–414.
[8] T. Ropinski, J. Meyer-Spradow, S. Diepenbrock, J. Mensmann, K. Hinrichs, Interactive volume rendering with dynamic ambient occlusion and color bleeding, Comput. Graph. Forum (2008) 567–576.
[9] M. Schott, P. Grosset, T. Martin, C. Hansen, V. Pegoraro, Depth of field effects for interactive direct volume rendering, Computer Graphics Forum 30 (3) (2010) 941–950.
[10] D. van Antwerpen, Improving simd efficiency for parallel monte carlo light transport on the gpu, in: Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics, HPG '11, ACM, New York, NY, USA, 2011, pp. 41–50.
[11] D. B. Kirk, W. W. Hwu, Programming Massively Parallel Processors, A Hands on Approach, Elsevier, Morgan Kaufmann, 2010.
[12] J. Sanders, E. Kandrot, CUDA by Example, An Introduction to General Purpose GPU Programming, Addison Wesley, 2010.
[13] NVIDIA, Nvidia cuda compute unified device architecture, programming guide version 4.2., in: NVIDIA, 2012, pp. 1–173.
[14] S. Molnar, M. Cox, D. Ellsworth, H. Fuchs, A sorting classification of parallel rendering, IEEE Computer Graphics and Applications 14 (1994) 23–32.
[15] R. Samanta, T. Funkhouser, K. Li, J. P. Singh, Sort-first parallel rendering with a cluster of pcs, in: In SIGGRAPH 2000 Technical sketches, 2000, pp. 26–0.
[16] N. Schwarz, J. Leigh, Distributed volume rendering for scalable high-resolution display arrays., in: P. Richard, J. Braz, A. Hilton (Eds.), GRAPP, INSTICC Press, 2010, pp. 211–218.
[17] S. Marchesin, C. Mongenet, J. M. Dischler, Multi-gpu sort-last volume visualization, in: Proceedings of the 8th Eurographics conference on Parallel Graphics and Visualization, EG PGV'08, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2008, pp. 1–8.
[18] J. R. Monfort, M. Grossman, Scaling of 3d game engine workloads on modern multi-gpu systems, in: Proceedings of the Conference on High Performance Graphics 2009, HPG '09, ACM, New York, NY, USA, 2009, pp. 37–46.
[19] C. M. Mocan, D. Gorgan, Cluster based modeling and graphical visualization of interactive large spatial data, in: MIPRO, 2010 Proceedings of the 33rd International Convention, 2010, pp. 258 –263.
[20] J. S. Liu, Monte Carlo Strategies in Scientific Computing, corrected Edition, Springer, 2008.
[21] Osirix imaging software (last rev dec 2012) url http://www.osirix-viewer.com/datasets/.
[22] Gpu-z video card gpu information(last rev dec 2012) url http://www.techpowerup.com/gpuz/.
[23] C. Perez, F. Piccoli, Towards the specification of the gpu using performance parameters, in: 40 JAIIO Cba, Argentina, 2011, pp. 117–129.