Mingqiang YIN, Shiqi LI

Huazhong University of Science & Technology

Dynamic load balancing strategy for sort-first parallel rendering

Abstract. *Parallel rendering based on a PC cluster is an effective method to improve the performance and resolution of graphic system. In order to achieve dynamic load balance among the render nodes, we propose a new load division scheme based on the load distribution map, which is built according to the rendering time of the previous frame. The proposed load balancing algorithm is simple to be implemented and works well for sort-first parallel rendering system. Experiment results show that our method is effective. Compared with the previous works, the proposed strategies can effectively use the available graphics resources, thus improving rendering performance of parallel rendering system.*

Streszczenie: *Metodą polepszenia własności i analizy systemu graficznego jest równoległy rendering oparty o klaster PC. W celu osiągnięcia równowagi dynamicznego obciążenia odtwarzanych węzłów proponujemy nowy schemat podziału obciążenia. Schemat opiera się o mapę rozkładu budowaną w zależności od czasu renderingu poprzedniej ramki. Zastosowano prosty do implementacji algorytm, dobrze pracujący w przypadku wstępnego sortowania, w systemie renderingu równoległego. Wyniki badań wskazują, że zastosowana metoda jest skuteczna. W porównaniu do poprzednich prac, w zaproponowanej strategii można skutecznie wykorzystać dostępne graficzne zasoby, co poprawia działanie równoległego systemu renderingu.* ***Strategia równoważenia dynamicznego obciążenia do wstępnego sortowania w równoległym renderingu***

Keywords: parallel rendering, sort-first, load balancing.
Słowa kluczowe: rendering równoległy, sortowanie wstępne, równoważenie obciążenia

1. Introduction

Over the last years, the rapid growth of data information for 3D data processing places a higher demand on processing speed, data quantity, display size and frequency, etc. Traditionally, expensive specialized graphics machines are used to render extremely large data sets. However, these machines are too expensive for popular use.

On the other hand, the performance of PCs consistently improves, and in particular, the development of commercial graphics cards even exceeds Moore's Law. Thus a PC cluster that uses cheap cost-effective PCs and a high-speed network for the hardware platform is used more and more often as a substitute for an expensive specialized graphics machine. The research on PC cluster based parallel rendering has become an effective solution to these problems by decomposing the rendering task among graphic resources.

Molnar categorized the parallel rendering system into three classes [1]: sort-first, sort-middle and sort-last. However, only sort-last and sort-first are applicable in most parallel rendering context. In the sort-last situation, the data is split between the nodes, and each node renders its own portion. Then, compositing takes the depth information into account to form a final image from each node's rendering. The bottleneck of this situation is the data transmission and image composition. In the sort-first situation, primitives are distributed among the nodes at the beginning of the rendering pipeline, usually by splitting the screen into regions and associating each region to one node. In this approach, load balancing is more important to enhance the rendering performance.

Abraham etc. [2] obtain the dynamic load balance on the fact that, in an interactive application, the viewpoint changes very little from frame to frame. So the rendering time of each pixel in the current frame is estimated by the rendering time in the previous frame, also known as frame-to-frame coherence. This scheme enables a fast calculation, but at the cost of limited accuracy, as sometimes it is difficult to get exact rendering times of each pixel. Hui C. etc. in [3] proposed a deferred shading method to obtain load-balance for sort-first parallel rendering system. Their method is base on the 2-pass rendering character of deferred shading to predict the rendering load. Their algorithm can get accurate load balance but impose much overhead to the rendering process. Therefore the increased rendering efficiency is limited.

For parallel rendering system, Equalizer [4] is an open source research and development framework with respect to parallel OpenGL program. It provides scalable parallel rendering based on sort-first, sort-last and other task decomposition strategies as well as parallel image compositing. It supports a wide variety of task distribution approaches from more easily load-balanced time and view-multiplexing to more difficult sort-last or sort-first parallel rendering. Especially for the sort-first dynamic screen-partitioning is supported in the framework as well, similar in principle as in [1,2] but based on past rendering times. Our dynamic load balancing algorithm and results are demonstrated in the context of this parallel rendering framework.

In this paper, we propose a new load balancing algorithm to enhance the rendering performance. Our algorithm explores frame-to-frame coherence and estimate each node's load based on its previous frame time. Differently from previous proposals, we design a new data structure named Load Distribution Map (LDM) to record the rendering load distribution. According to the LDM, the rendering task can be subdivided precisely and equally. It is very simple to implement, and takes a negligible time to run. Experiment shows that our algorithm can greatly improve the performance of the parallel rendering system.

The paper is organized as follows: In the next section, we describe the dynamic load balancing algorithm in detail and how it is used in parallel rendering. Section 3 presents experiments results that illustrate the efficiency of proposed algorithm used in parallel rendering system. Finally, in section 4, some concluding and future works are drawn.

2. Dynamic load balancing strategy

In sort-first parallel rendering system, rendering task assignment is a critical technique to get optimal resource utilization, maximum throughput, and high scalability. The rendering load for each rendering slave changes according to the movement of the user's view-point or of the object while the user is browsing in the 3D interactive virtual scene. The master host has to wait for all rendering nodes to accomplish their rendering task before composing the final image. Clearly, the slowest rendering slave represents the bottleneck of the application. Although, screen space subdivision is straightforward, how to achieve accurate load balancing is still unsettled. Our algorithm takes advantage of frame-to-frame coherence and tries to balance the load based on the time each node takes to render the previous frame.

2.1. Rendering load distribution measurement

In order to divide rendering task equally, we need to measure the load distribution among screen space first. For most interactive rendering applications, position of the viewpoint could make change all the time, such that estimating each node's rendering task need to traverse all the geometry primitives with view frustum culling, which is too costly for real-time rendering and is a great challenge. In this paper, we measure each node's load by counting the rendering time of previous frame, and then we design a special data structure to store the load distribution. We call this data structure as Load Distribution Map (LDM). According to the LDM, we can get a precise load distribution among the rendering node.

By partitioning the LDM, a better load balancing for the next frame could be obtained. In this way dynamic load balancing could be achieved.

First of all, some concepts for our algorithm are introduced. Render time $(RT_{ij}, i=1,2,...,N, j=1,2,....)$ refers to the ith rendering slave to render scene of the jth frame, the resolution of the sub-image is define as $(MN_{ij}, i=1,2,...,N, j=1,2,....)$, and the number of pixels which have been shaded is define as CN_{ij}. Then we define the LDM as:

$$(1)\quad LDM[m][n]=\begin{cases}0 & (if\ \ pixel[m][n]\ not\ been\ shaded)\\ RT_{ij}/CN_{ij} & (if\ \ pixel[m][n]\ \ been\ shaded)\end{cases}$$

For example, suppose that the previous frame rendering result is a very low resolution image, as shown in figure 1, which is rendered by four rendering nodes. The resolution of the display is 14×10. The four sub-images are respectively displayed in upper-left, upper-right, lower-left and lower-right corner. If the rendering time of the upper-left rendering slave for the previous frame is 29.04ms. From the figure we can see that the number of pixels which have been shaded is 12. The rendering task per pixel can be obtained by equation (1). Then the LDM is build for the first node, as shown in the upper-left corner in figure 2. In the same way, we can build the other LDM for other render nodes respectively. After that, the overall load distribution map is build as figure 2. The next work is to divide the LDM into discrete, non-overlapping tiles.

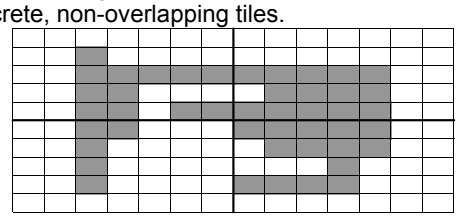


Fig. 1. A low resolution image rendered by four rendering slaves

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2.42	0	0	0	0	0	0	0	0	0	0	0
0	0	2.42	2.42	2.42	2.42	2.42	2.56	2.56	2.56	2.56	2.56	0	0
0	0	2.42	2.42	0	0	0	0	2.56	2.56	2.56	2.56	0	0
0	0	2.42	2.42	0	2.42	2.42	2.56	2.56	2.56	2.56	2.56	0	0
0	0	1.73	1.73	0	0	0	1.24	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	1.24	1.24	1.24	1.24	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 2. The LDM for the low resolution image

2.2. Rendering load assignment

In sort-first architecture, rendering task assignment should first divide the whole screen space into N tiles and with all tiles have the same amount of load, where N is the number of rendering slaves in the cluster. Mueller [1] has pointed out that natural choices to subdivide the screen include horizontal strips, vertical strips, and more rectangular shapes. Square shapes are often the preferred choice because they minimize the total region boundaries, thus minimizing the percentage of overlapping primitives. In our algorithm, the rendering load is measured by the LDM. For each rendering slave, its rendering task is measured by summing the data filled in the corresponding part of LDM. As the size of LDM is similar to the size of screen space, screen subdivision can be looked as LDM subdivision. In order to minimize tile boundaries, we choose to divide the LDM into rectangular tiles. For a perfect balancing, each tile should have a load equal to the overall frame time over the number of rendering slaves.

Just like figure 3(a). LDM can be regarded as a special tile D, which is at the origin of (x_0, y_0) and with the resolution of $W \times H (W \geq H)$. Then the tile D can be subdivided as follow. The whole process can be shown as figure 3.

1) Compute rendering load $T_j = \sum_{i=y_0}^{y_0+H} LDM[i][j]$ for each column, and the total $T_{total} = \sum_{j=x_0}^{x_0+W} T_j$, where $i \in [x_0, x_0+W], j \in [y_0, y_0+H]$;

2) Accumulate the column load from left to right as: $T_k = \sum_{i=x_0}^{x_0+k} T_i \ \ k \in [x_0, x_0+W]$;

3) If $T_k \leq T_{total}/2 < T_{k+1}$, then position k is a splitting boundary of the two sub-tiles $D_1^{'}$ and $D_2^{'}$;

4) Repeat step 1 to 3 for $D_1^{'}$ and $D_2^{'}$ until the subdivision depth reaches $log_2 N$;

The whole process can be shown in figure 3.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2.42	0	0	0	0	0	0	0	0	0	0	0
0	0	2.42	2.42	2.42	2.42	2.42	2.56	2.56	2.56	2.56	2.56	0	0
0	0	2.42	2.42	0	0	0	0	2.56	2.56	2.56	2.56	0	0
0	0	2.42	2.42	0	0	0	2.56	2.56	2.56	2.56	2.56	0	0
0	0	1.73	1.73	0	0	0	1.24	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	0	0	1.24	0	0	0
0	0	1.73	0	0	0	0	1.24	1.24	1.24	1.24	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

(a)

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2.42	0	0	0	0	0	0	0	0	0	0	0
0	0	2.42	2.42	2.42	2.42	2.42	2.56	2.56	2.56	2.56	2.56	0	0
0	0	2.42	2.42	0	0	0	0	2.56	2.56	2.56	2.56	0	0
0	0	2.42	2.42	0	0	0	2.56	2.56	2.56	2.56	2.56	0	0
0	0	1.73	1.73	0	0	0	1.24	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	0	0	1.24	0	0	0
0	0	1.73	0	0	0	0	1.24	1.24	1.24	1.24	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b)

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2.42	0	0	0	0	0	0	0	0	0	0	0
0	0	2.42	2.42	2.42	2.42	2.42	2.56	2.56	2.56	2.56	2.56	0	0
0	0	2.42	2.42	0	0	0	0	2.56	2.56	2.56	2.56	0	0
0	0	2.42	2.42	0	0	0	2.56	2.56	2.56	2.56	2.56	0	0
0	0	1.73	1.73	0	0	0	1.24	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	1.24	1.24	1.24	1.24	0	0
0	0	1.73	0	0	0	0	0	0	0	1.24	0	0	0
0	0	1.73	0	0	0	0	1.24	1.24	1.24	1.24	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

(c)

Fig. 3. Process of LDM subdivision

3. Experimental result

We implemented and tested the proposed algorithms on a cluster composed of 5 Dell precision T3400 workstations, which equipped with Intel Core 2 Duo E6550 2.33GHz (Allendale) and 2 GBytes of RAM, running Windows XP operating system. One of these workstations is worked as master host and display monitor, while the others are work as rendering nodes. All the workstations are connected by a switched Gigabit Ethernet work, running on Windows XP operation system. The packet/frustum ray tracer is used for rendering.

To test the performance of our algorithm, we compare it with two different load balance algorithm: Abraham's algorithm outlined in, and deferred shading based algorithm, which use the character of deferred shading to predict the rendering load distribution among the screen space. As we known, a good load balancing algorithm would be the one which not only make all the rendering nodes accomplish their task in the same time, but also get a higher frame rate for the rendering system. Therefore, during the experiments, we measured the load-imbalance which is defined as the ratio of the maximum rendering time over the average rendering time. Lower value of the maximum/average load ratio is considered as more reasonable load-balance. At the same time, we also record the frame rate of different algorithm used in the parallel rendering system.

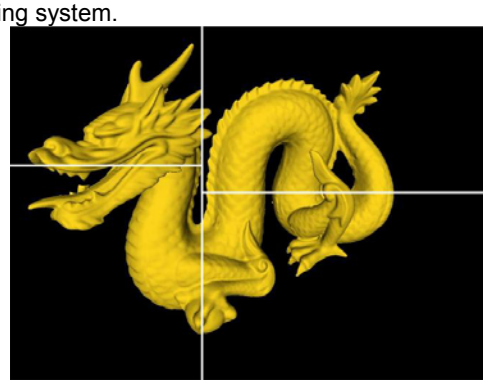
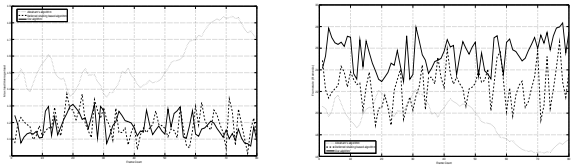


Fig. 4. Test scene: dragon

The test scenes are shown in figure 4, which contains 871470 triangles. For each algorithm, we have run the Equalizer parallel rendering system three times to make sure we have consistent frame timing statistics among the three results for each frame. During the experiment, the viewpoint circled around the virtual scene. The camera path was also the same for each run.



(a)Load-imbalance (b) Rendering frame rate

Fig. 5. Rendering frame rate test with different load balancing algorithms

Figure 5 shows the experimental result we achieved. From figure 5(a), we can see that our algorithm is approximately the same ratio with deferred shading base algorithm. Obviously, they all provided most ratios below 1.3 with an average value below 1.2. While Abraham's algorithms made the load imbalance value supra 1.5 and also made the value is much wavier than our algorithm. I think the main reasons are as follow: (i) for deferred shading base algorithm, the task assignment is base on counting the geometry primitives, so it can get a precise load distribution, that will translate to more equally task decomposition. (ii) For Abraham's algorithm, it decomposes the rendering task only according to the previous rendering time. But the rendering task per pixel is non-uniform distributed in the sub-image. Therefore it could only get a roughly task distribution. This confirms that our algorithm and deferred shading based algorithm is superior over Abraham's algorithm.

At the same time, the test result for rendering performance is showed on figure 5(b). As Abraham's algorithm in load balance is worse than others, it achieves a lower frame rate than others. Comparing with deferred shading base algorithm, our method can get a higher frame rate. That's because our algorithm impose less over head to the parallel rendering system.

4. Conclusion and future works

In this paper, we have presented a new dynamic load balancing approach integrated into Equalizer. Using simple frame timing statistics from past frames, we design a data structure named LMD to store the load distribution. According to the LDM our algorithm could decompose the render task equally. Combined with the load balancing algorithm, we have also proposed an optimizing strategy. The presented optimization is based on the sequence of the parallel computation. The proposed algorithm has shown good results for rendering of 3D geometry models, while being very simple to implement. Experiments show that the presented strategy successfully enhances the frame rate of Equalizer parallel rendering system.

There are many avenues for future work, such as the large data sets stored in each rendering slave which may cause a low efficient to render special sub-scene. To solve the problem and improve the efficiency of available graphics resources, scene data management algorithm is currently under investigation. Moreover, in order to efficiency of scene rendering, the parallel occlusion culling may be worth to study.

Acknowledgements

This research work has been partially supported by National High-tech Research & Development Program of China under Grant NO.2010AA804022.

REFERENCES

[1] Molnar S., Cox M., Ellsworth D., Fuchs H.: A Sorting Classification of Parallel Rendering. IEEE Computer Graphics and Applications, vol. 14, (1994) 23-32.

[2] Abraham F., Celes W., Cerqueira R., Campos J.L.: A load-balancing strategy for sort-first distributed rendering. In Proceedings SIBGRAPI (2004) 292–299.

[3] Hui C., Xiaoyong L., Shuling D.: A dynamic load balancing algorithm for sort-first rendering clusters. In Proceedings IEEE International Conference on Computer Science and Information Technology (2009) 515–519.

[4] Eilemann S., Makhinya M., Pajarola. R.: Equalizer: A scalable parallel rendering framework. IEEE Transactions on Visualization and Computer Graphics 15, 3 (2009) 436–452.

[5] B. Moloney, D. Weiskopf, T. Mo¨ ller, and M. Strengert, "Scalable Sort-First Parallel Direct Volume Rendering with Dynamic Load Balancing," Proc. Eurographics (EG) Symp. Parallel Graphics Visualization (PGV), (2007). 45-52,

[6] Moloney B., Weiskopf D., Möller T., Strengert M.: Scalable sort-first parallel direct volume rendering with dynamic load balancing. In Proceedings Eurographics Symposium on Parallel Graphics and Visualization (2007) 45–52.

[7] Mueller C.: The Sort-First Rendering Architecture for High-Performance Graphics. In Proceedings of Symposium on Interactive 3D graphics, (1995) 75-84.

Authors: *Ph.D candidate Mingqiang Yin, 1037 Luoyu Road, Wuhan, China Huazhong University of Science & Technology, E-mail: ymqeml@yahoo.com; prof. Shiqi Li, 1037 Luoyu Road, Wuhan, China Huazhong University of Science & Technology.*