SCENE DATA SYNCHRONIZATION IN SORT-FIRST RENDERING SYSTEM FOR LARGE DYNAMIC SCENES

He Bing and Wang Yangzihao
State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, Beijing, China

Keywords: Scene data synchronization, Cluster parallel rendering, Sort-first, Dynamic scenes.

Abstract: In this paper we built a cluster-based sort-first rendering system. Unlike previous sort-first rendering systems for static scenes, ours can cope with large dynamic scenes with massive data. A set of strategies are designed and implemented to give solutions for scene data synchronization in our system. The experimental results show that our system maintains favorable data consistency for dynamic scenes and is highly scalable with solid improvement of rendering performance. Using 16 computing nodes, our system can achieve interactive visualization result in the test physical simulation scene which contains 10,000 moving rigid-body models and building models with massive geometric and texture data.

1 INTRODUCTION

With the development of graphics hardware, visualization applications with massive data sets are made possible using cluster-based parallel rendering. Based on where the sort from object-space to screen space occurs, there are three parallel rendering architectures: sort-first, sort-last and sort-middle. With low communication cost and the advantage of frame-to-frame coherence, sort-first has been the most widely studied and used architecture of parallel rendering. It is highly scalable and is particularly suitable for cluster implementation.

During the past decades, several sort-first rendering systems are developed and applied to various applications(Mueller, 1995) (Samanta et al., 1999) (Samanta et al., 2000) (Humphreys et al., 2001) (Humphreys et al., 2002). However, sort-first rendering strategy with large dynamic scenes remains an unsolved problem. It is mainly due to the difficulty of keeping the consistency of the moving object's status (position, velocity, etc.) among each rendering node. One of the key issues in solving this problem is scene data synchronization.

In this paper, we build a multi-thread sort-first rendering system with physical computing module and implemented a set of methods on scene management and scene data synchronization to cope with massive dynamic data sets. Using our strategy, we achieved interactive visualization result in a scene which contains more than 10,000 moving rigid-body models and building models with massive geometric and texture data.

2 RELATED WORK

A comprehensive survey on cluster-based parallel rendering goes beyond the scope of this paper. See (Pajarola, 2008) and (Staadt et al., 2008). In this section, we will focus on sort-first architecture and scene data synchronization issue.

Molnar *et al.* have classified parallel rendering based on where the visibility sort occurs into sort-first, sort-last and sort-middle in(Molnar et al., 1994). In sort-first architecture, little intervene is done to graphic pipeline. Mueller (Mueller, 1995) has pointed out that in the sort-first architecture, the screen is partitioned into non-overlapping tiles (usually with rectangular shapes) and the rendering nodes are responsible for all the rendering computation that affects their respective screen regions. According to the frame-to-frame coherence, the network overhead is minimized.

Data synchronization is a fundamental issue in distributed systems. Several sort-first rendering systems have developed their own scene data synchronization strategies. In Samanta's retained mode rendering system for static scenes(Samanta et al., 1999), because the client and all the servers read the same 3D scene graph from disk and store it entirely in memory, there is no data synchronization requirement. In immediate mode system such as WireGL(Humphreys et al., 2001) and Molnar's first sort-first rendering system(Molnar et al., 1994), scene data synchronization is mainly pixel redistribution between each two frames during the rendering of static scenes. In the rendering of dynamic scenes though, scene data synchronization becomes more difficult due to the reason that several parameters of the moving objects (transformation matrix, velocity, acceleration *et al.*) are needed to be synchronized between each two frames. As far as we know, works on this specific topic are rare.

3 OVERVIEW

The prototype sort-first parallel rendering system we have built is composed by a single display and a cluster of computing nodes connected by 1000M bandwidth local area network. Logically, we divided the PC clusters into three groups: controlling node, computing node and image composition node.

A controlling node is in charge of the scene data synchronization, task decomposition and load balancing. It controls the running of the whole system. A computing node performs all the computing tasks. There are two kinds of computing nodes: rendering node and physical computing node. Note the classification is conceptual, in our system, every computing machine contains one pair of rendering node and physical computing node at the same time. The rendering node renders the scene within a given sub frustum assigned by the controlling node, the physical computing node performs the collision detection and response within a spatial region assigned by the controlling node. The image composition node receives all resulting images from the computing nodes and composes them into the final result.

For each frame, the controlling node first performs task decomposition for rendering nodes and physical computing nodes respectively according to load information received from the latest frame. Then it collects the collision detection/response results from each physical computing node and generates update information which will then be sent back to each computing machine along with the rendering and physical computing tasks. The physical computing nodes and the rendering nodes perform their tasks. Different rendering results are sent to the image composition node and the collision detection/response results are sent back to the controlling node along with load information such as rendering time and primitive counts.

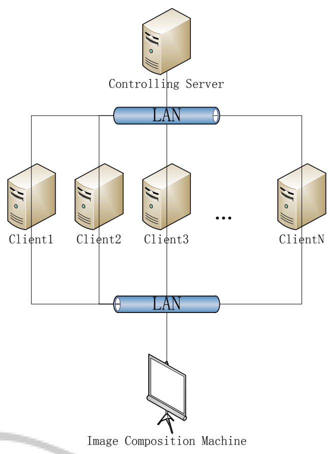

Figure 1: Topology structure of our sort-first parallel rendering system.

4 SCENE DATA SYNCHRONIZATION

Alan Chalmers and Erik Reinhard (Chalmers and Reinhard, 1998) pointed out that in a distributed system, the option to maintain sequential consistency is an expensive one. Therefore, the weak consistency technique is proposed to improve the performance. Using this technique, the local cache will stay inconsistent until the application process orders the data manager to repair the inconsistency. We built our scene data synchronization strategy on the base of weak consistency. We use a frame-rate control strategy to guarantee that each rendering node has access to the updated data of moving objects in the scene. To resolve the conflicts of octree update and scene rendering in multi-thread environment, we use a double-buffering approach. Finally, a strategy using overlapped octree region is proposed to avoid data inconsistency during the collision detection/response phase.

In parallel rendering system for dynamic scenes, real-time communication of updated objects information among each computing nodes is the key issue. It is difficult to send updated objects information to the rendering node which will render them in the next frame before the rendering node starts to acquire the information. To solve this problem, we set up two time systems in our prototype system. Rendering nodes compute object's transformation matrix using interpolation from its rendering time t_r, original position, velocity and acceleration stored in an object information list. The physical simulation time t_p is ahead of rendering time by ϕ seconds. Physical computing nodes use this period of time to process data pre-processing and objects information update. ϕ adaptively changes during the rendering process, guarantee that every rendering node can get access to the updated data of moving objects.

If t_r runs ahead of t_p, data inconsistency of objects in different neighboring rendering nodes will be caused. To prevent such situation, we propose a duplex frame-rate control method, using both time control and frame number synchronization. In our method, physical computing nodes and rendering nodes receive instructions from both the controlling node and the image composition node to maintain a smooth frame rate. As shown in figure 2, the simulation and rendering starts together when all computing nodes in the clusters have connected to both the controlling node and the image composition node. Each physical computing node sends different updated objects information to controlling node according to different task distribution. After the data processing, the controlling node sends the information to object information list on each rendering node. Each rendering node renders different part of the result image with an unique frame number according to its unique task distribution, then sends the image to the composition node, when images from all rendering nodes with the same frame number are received by the image composition node, rendering nodes can start the rendering of the next frame. A frame-rate control program runs on each rendering node turns the rendering thread to sleep when $t_p - t_r < \phi$ for a shot period of time: $(\phi - (t_p - t_r))$. This strategy minimizes the coupling between rendering nodes and physical computing nodes and gives us the convenience of generating different task distribution for rendering nodes and physical computing nodes.

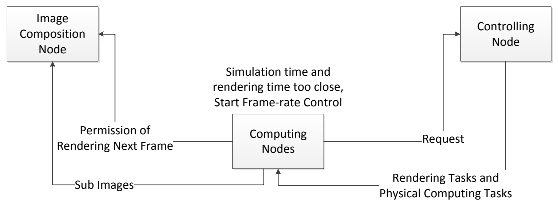
Image Composition Node
Controlling Node
Simulation time and rendering time too close, Start Frame-rate Control
Permission of Rendering Next Frame
Computing Nodes
Request
Sub Images
Rendering Tasks and Physical Computing Tasks

Figure 2: The Frame-Rate Control Strategy.

Due to the reason that our division of physical computing node and rendering node is only conceptual, one machine with multi-core CPUs can be both the physical computing node and the rendering node to take the advantage of parallel power. If we use one set of octree for both objects information update and octree traverse for rendering, a read/write conflict might be caused. We proposed a special double-buffering approach as our memory optimization strategy. We use two set of octrees pointed to a single data set, one is in charge of objects information update and the other is in charge of octree traverse for model rendering. Two octrees switch their tasks every frame. Thus we get a good balance between data delay and data inconsistency. Experiments show that by using this strategy, no read/write conflict is caused during the running of the system.

According to the physical computing task distribution, each physical computing node only processes physical simulation in a sub-region of the whole scene. Objects on the boundaries of the sub-region may collide with objects from other sub-regions, due to the missing of objects information from other regions, such collisions would be mistakenly overlooked by the physical computing node. The solution is to use an overlapped octree region for each physical computing node. Information of the objects in overlapped regions is stored in every octree node which contains these regions. We add a data preprocessing program at the controlling node to remove the redundant collision information caused by overlapped octree regions.

5 IMPLEMENTATIONS AND RESULTS

We implemented our prototype system using 18 computers with Intel Core(TM)2 Quad CPU and Nvidia GTX 260 GPU. One computer serves as the controlling node, one computer serves as the image composition node, all other 16 computers serve as both the rendering node and the physical computing node.

We set up two dynamic scenes with automatic camera tracking (see figure 3). The left one is the explosion scene of 10,000 rigid objects and the left one is 10,000 moving objects flying among buildings of Tianjin Jiefang Southern Road. To test the scalability of our system, we run our system with 4, 8, and 16 computing nodes respectively. From the figure 4 we can conclude that our system has improved the rendering performance with the increasing number of computing nodes used in the system. Though with too many computing nodes, the overload from network may balance out the performance improvement.

Figure 5 shows the effectiveness of our scene data synchronization strategy. Without our scene data synchronization strategy, the right figure has an artifact in the composition result caused by data inconsistency. In the left figure, we use our scene data synchronization to completely eliminate the scene data inconsistency.

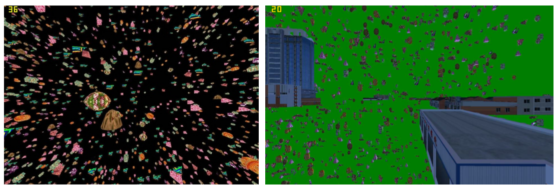

Figure 3: Left: Objects explosion scene ($36fps$, with 876,267 primitives); Right: City model scene ($20fps$, with 1,485,218 primitives).

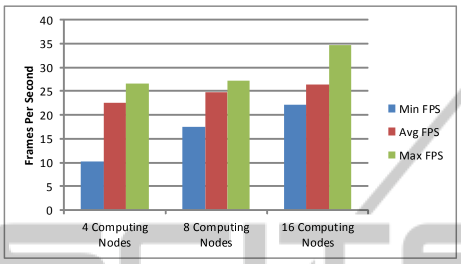

Figure 4: FPS Comparison of our experimental scene with 4, 8, and 16 computing nodes.

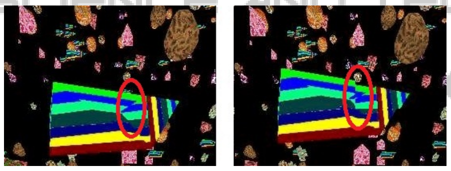

Figure 5: Two frames(partial) at the same rendering time with(left) and without(right) scene data synchronization strategy.

6 CONCLUSIONS

We designed and implemented a cluster-based sort-first parallel rendering system which is capable of rendering large dynamic scenes with massive data. We focus on scene data synchronization strategy based on the weak consistency technique. To improve the overall performance of the cluster-based parallel rendering system, we proposed a set of algorithms to acquire scene data synchronization in the rendering of dynamic scenes with massive data. Experiments show that our system has good scalability and the strategies we proposed can effectively keep the scene data consistency with interactive frame rate.

For future work we are interested in improving the performance of the parallel rendering system by transporting some of the scene data synchronization, scene management and load balancing algorithms to GPU Clusters. Future cluster-based parallel rendering systems should support both static scenes and dynamic scenes, they should also be a hybrid of cluster parallel and GPU parallel.

REFERENCES

Chalmers, A. and Reinhard, E. (1998). Parallel and distributed photo-realistic rendering. In *Philosophy of Mind: Classical and Contemporary Readings. Oxford and*, pages 608–633. University Press.

Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., and Hanrahan, P. (2001). Wiregl: a scalable graphics system for clusters. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 129–140, New York, NY, USA. ACM.

Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. (2002). Chromium: a stream-processing framework for interactive rendering on clusters. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702, New York, NY, USA. ACM.

Molnar, S., Cox, M., Ellsworth, D., and Fuchs, H. (1994). A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14:23–32.

Mueller, C. (1995). The sort-first rendering architecture for high-performance graphics. In *I3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 75–ff., New York, NY, USA. ACM.

Pajarola, R. (2008). Cluster parallel rendering. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses*, pages 1–12, New York, NY, USA. ACM.

Samanta, R., Funkhouser, T., Li, K., and Singh, J. P. (2000). Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *HWWS '00: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 97–108, New York, NY, USA. ACM.

Samanta, R., Zheng, J., Funkhouser, T., Li, K., and Singh, J. P. (1999). Load balancing for multi-projector rendering systems. In *HWWS '99: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 107–116, New York, NY, USA. ACM.

Staadt, O. G., Walker, J., Nuber, C., and Hamann, B. (2008). A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses*, pages 1–10, New York, NY, USA. ACM.