A Geometry Data Independent Load Balancing Method For Graphics Parallel Rendering

Zhefan Jin
State Key Lab of CAD&CG, Zhejiang University.
310027, HangZhou, PRC.
jinzf@cad.zju.edu.cn

JiaoYing Shi
State Key Lab of CAD&CG, Zhejiang University.
310027, HangZhou, PRC.
jyshi@cad.zju.edu.cn

ABSTRACT

We describe a novel load-balancing method for sort-first parallel graphics rendering systems. It gives up geometry data which could be very large and tends to cause unacceptable cost. Instead it takes rendering time as the measurement of render nodes' work load and produces new screen decomposition using a time-to-space algorithm. Test results show that the geometry data independent method is very effective.

Keywords

sort-first, load balance, geometry data, screen division

1. INTRODUCTION

The performance of computer graphics systems is growing rapidly due to the improvement of both graphics architecture and implementing technologies. However the performance of stand-alone systems is still limited at the following aspects[Tim00a]: compute-limited, graphics-limited, interface-limited and resolution-limited. Parallelism is a crucial tool to building high performance graphics systems which can be used in high-end applications such as scientific visualization of large data set, high resolution display and photo-realistic rendering.

By implementing type, parallel rendering systems can be classified as hardware based systems such as InfiniteReality[Joh97a] and Pixel-Flow[Joh97b], large-scale parallel machine based systems such as Parallel-Mesa[Tul98a] and PGL[Tho95a], cluster based systems such as WireGL[Gre01a], AnyGL[Jia02a], Display Wall[Rud00a] and Pomegranate[Mat00a]. By the way that multiple rendering pipelines are organized, parallel rendering systems can be classified as sort-first, sort-middle and sort-last.

WSCG POSTERS proceedings
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.

In sort-first, the screen space is divided into many rectangular regions and each processor is assigned a portion of the screen to render. Each primitive is pre-transformed to determine which processor it belongs to. When the processors get all of the primitives that full into their respective portion of the screen, they work independently and each generates a sub-image which will be sent the one or more frame buffers to be displayed.

In sort-middle, there is a set of transformation processors and a set of rasterization processors. Each rasterization processor is assigned a portion of the screen. Each transformation processor completely transforms its portion of the primitives and the resulting primitive information is classified by screen location and sent to the correct set of rasterization processors. Like in sort-first, multiple sub-images are seamed together to form a full size image to be displayed.

In sort-last, each processor has a complete rendering pipeline and generates a full size image by rendering its fraction of the primitives. These image are sent across a network and depth composed to produced the final image to be displayed.

Sort-first and sort-middle systems all allocate rendering tasks by decomposing the screen into multiple regions. They both suffer from load imbalance for the primitives usually fall into screen in a random manner. There're several approaches to solve the load-balancing problem including static methods and adaptive methods. We'll present later an adaptive load-balancing method that takes rendering time as the measure of work load.

2. RELATED WORK

Static methods divide the screen into more regions than there're processors and assign them to the processors in an interlaced fashion. If the screen is divided finely enough, each processor tends to have portions of both populated and sparse areas, and thus the nearly equal loads. Static methods cause little cost, but they are passive approaches, there's still the possibility that a high concentration of the primitives will fall into one region causing load-imbalance.

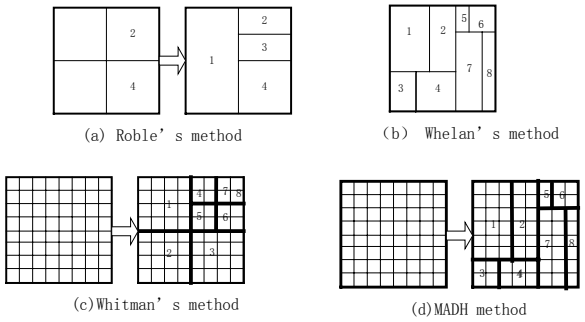

Figure 1. Adaptive methods

Adaptive methods change the screen dividing at runtime due to the distribution of work load. They come in a variety of algorithms. Roble's[Car95] method(Figure 1-a) stars with a standard rectangular decomposition. According to the number of primitives in each region, lightly loaded regions are combined and highly loaded ones are split in half and assigned to the processors freed by the combining.

Whelan's[Whe85a] method(Figure 1-b), also known as media-cut algorithm, splits the screen into sub-regions based upon the distribution of the centroids of each primitive. The cuts recursively divide the longer dimension of the screen until the number of regions equals the number of processors. Whitman's[Whi94] top-down decomposition method (Figure 1-c) stats by tallying up primitives based upon how their bonding boxes overlap a fine mesh. A unit is added to each mesh cell that the bounding box overlaps. Then adjacent mesh cells are combined and summed hierarchically to form a tree structure. The tree is then traversed top-down by splitting the region with the most primitive in half each time. The subdividing goes on until the number of regions is 10 time the number of processors. Dynamic task assignment is used to even out the processor load balance.

MAHD[Car95](mesh-based adaptive hierarchical decomposition) method(Figure 1-d) also uses a fine mesh to tally primitives. The amount tallied to each cell is inversely proportional to the number of cells a primitive covers. The cells are summed into a summed area table after all primitives have been counted. Finally the screen is divided along cell boundaries using a hierarchical approach similar to that of media-cut. The summed-area table allows a binary search operation to determine the location of each cut.

All these adaptive methods need to transform primitives to screen coordinates which is somewhat questionable. For the amount of geometry primitives could be very large and transferring and computation of the data is very much time consuming. Graphics system are usually high-coupling ones, so using its media result is not practical. We present here a novel load balancing method which is independent of geometry data. It takes rendering time as the measurement of a node's work load. By a time-to-space algorithm it turns time value to space value which will be used to adjust the decomposition of screen. Empirical results show the effectiveness of this method.

3. ARCHITECTURE OF ANT FORCE 2 SYSTEM

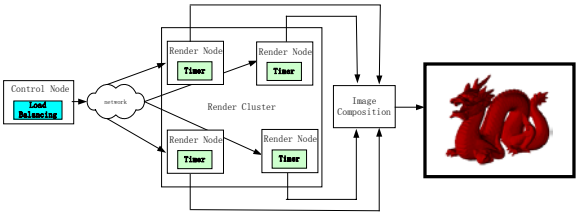

Figure 2. Ant Force 2 system

Figure 2 shows the architecture of Ant Force 2, a cluster-based sort first parallel rendering system. The control node gets user's input and distributes rendering tasks to multiple render nodes. Graphics rendering is preformed by render nodes in a parallel manner. Image composition node receives all the sub-images produced by render nodes and generates the final image to be displayed. The control node embodies a load balancing module and each render node embodies a timer.

Let the number of render nodes be n, A_1~A_n be the rectangular regions of the screen, f be the count of frames. The screen regions of the frame f is (A_{1f}, A_{2f}, ……A_{nf}) and rendering time of that frame is (t_{1f}, t_{2f}, ……t_{nf}). The working process of the system is as following:

Step 0: f=1. Euqally decompose the screen and get the regions of the 1st frame which is (A_{11}, A_{21}, ……A_{n1}).

Step 1: Control node receives user's input and command render node R_k to begin rendering region A^{kf} (k=1,2……n) and start the timer of R_k.

Step2: R_k renders A_{kf}, stop timer and send a "RENDER_OVER" message with t_{kf} to the control node.

Step3: When each R_k finished its rendering task, control node command image composition node to do the composition and output final image.

Step4: Control node calls time-space algorithm TS to get the decomposition strategy of the f+1th frame:

$(A_{1f+1}, A_{2f+1}, \ldots\ldots A_{nf+1}) = TS((A_{1f}, A_{2f}, \ldots\ldots A_{nf}), (t_{1f}, t_{2f}, \ldots\ldots t_{nf}))$

Step5: f =f+1

Step6: Goto Step 1

TS is the essential of the method which calculates the next decomposition from the current one as well as rendering time.

4. TIME TO SPACE: TS ALGORITHM

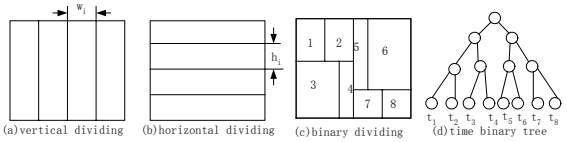

Figure 3. Vertical, horizontal and binary dividing

There're 3 types of screen dividing methods: vertical dividing, horizontal dividing and binary dividing as shown in Figure 3. When the system uses vertical dividing as Figure 3-a, $(A_{1f}, A_{2f}, \ldots\ldots A_{nf})$ is determined by $(w_1,w_2\ldots\ldots w_n)$ for all the regions have the same height. The input of real TS algorithm are:

$(w_1,w_2\ldots\ldots w_n)$ of the fth frame.

Render time of the fth frame $(t_1,t_2\ldots\ldots t_n)$

The expected output of TS is $(w_1',w_2'\ldots\ldots w_n')$ for the f+1th frame's rendering.

The process of TS is as following:

Step1. Get average value of $(t_1,t_2\ldots\ldots t_n)$: $\bar{t} = \frac{1}{n}\sum_{m=1}^{n} t_m$

Step2. Calculate $(p_0, p_1, \ldots p_n)$, in which $p_u = \sum_{v=1}^{u} t_v, p_0 = 0$

Step3. $w_0' = 0$

Step4. for (i=1 to n)

{

for (a=0 to n-1)

{

if $i \cdot t \in [p_a, p_{a+1}]$,break;

}

$$w_i' = (i \cdot t - p_a)\frac{w_{a+1}}{t_{a+1}} + \sum_{j=1}^{a} w_j - \sum_{k=1}^{i-1} w_k'$$

}

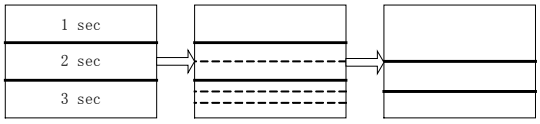

Figure 4. Principle of TS algorithm

Figure 4 shows the principle of TS algorithm. Assume that region 1,2,3 are rendered by nodes R_1, R_2, R_3 and render times are 1sec, 2sec and 3sec. It's assumed that the proportions of the work load of R_1, R_2, R_3 are 1:2:3. The regions are then decomposed according to this proportion. Finally small regions are reorganized inot 3 regions in a load-euqal manner.

To horizontal dividing, the TS algorithm is the same to that of vertical dividing. To binary dividing, TS algorithm uses a time binary tree as showed in Figure 3-d. The process of TS algorithm for binary dividing is as following:

Step 1: Fresh the time binary tree.

Step 2: Top-down traverse the tree. For each non-leaf level call TS to do horizontal or vertical cutting.

Comparing to other adaptive load balancing methods, the method presented here gives up geometry data which could be very large and to handle it tends to be time consuming. Instead it takes rendering time which reflects the work load of a render node well. TS algorithm turns time value to space value. From the processes we can see that the complexity of of TS for vertical and horizontal dividing is $O(n^2)$ and for binary dividing is O(n) where n is the count of render nodes.

5. TEST RESULTS

The load balancing method of this paper is run and tested on Ant Force 2. Figure 5 shows the render time for a series of frames. In Figure 5-a and b horizontal dividing is used. In Figure 5-c binary dividing is used. To measure the state of load balance we define a parameter $LB = \frac{T_{first}}{T_{render}}$, where Tfirst is the render time of the node that finish its job first and Trender is the total time that all nodes finish

rendering. Figure 6 shows LB values according to Figure 5.

From the result we can see that rendering speed is improved by using the load balancing method. The average rendering time with load-balancing is 60%~80% of that without load-balancing. LB is also improved 1.3~3.4 times by the method. The cost of the load balancing method is pretty low which is 2.3%~8.2% of total rendering time. It includes the cost of both TS computing and message exchanging over network.

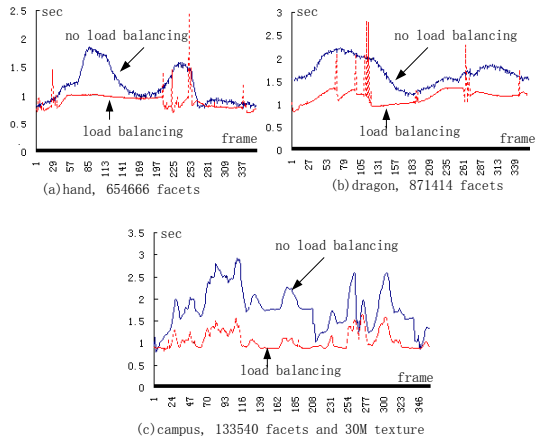

Figure 5. Test result of render time

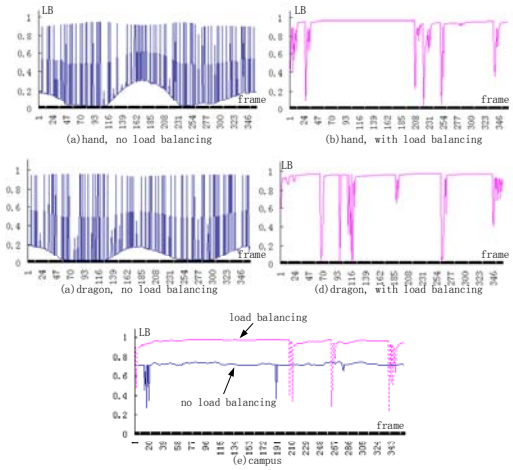

Figure 6. Test result of LB

6. CONCLUSION

Load balance severely impact the performance of sort-first and sort-middle parallel systems. The main difficulty of this problem is that the resource a load balancing method can use is limited and often little in the real world. Handling geometry tends to cause unacceptable cost. The geometry data independent method presented in this paper get render time and utilize it to perform future screen decomposition by TS algorithm. The cost of this method is pretty low and practical result show the effectiveness of this method.

REFERENCES

[Tim00a] Tim Davis, Alan Chalmers, Henrik Wann Jensen[2000]. Pra-ctical Parallel Processing for Realistic Rendering, SIGGRAPH 2000, Course 30

[Joh97a] John S. Montrym, Daniel R. Baum, David L. Dignam, Chr-istopher J. Migdal. InfiniteReality: A Real-Time Gra-phics System. Proceedings of SIGGRAPH '97, pages 293--302, Aug. 1997.

[Joh97b] John Eyles, Steven Molnar . PixelFlow Rasterizer Functional Description. Revision 7.0, November 20, 1997.

[Tul98a] Tulika Mitra, Tzicker Chiueh . Implementation and Evaluation of Parallel Mesa Library. IEEE International Conference on Parallel and Distributed Systems, December 1998.

[Tho95a] Thomas W. Crocket. Parallel Rendering. NASA Contractor Report 195080 ICASE Report No. 95-31 . http://www.icase.edu/~tom/95-31.pdf

[Gre01a] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew, Pat Hanrahan. WireGL: A Scalable Graphics System for Clusters. In Proceedings of ACM SIGGRAPH 2001

[Jia02a] Jian Yang, Jiaoying Shi, Zhefan Jin and Hui Zhang, "Design and Implementation of A Large-scale Hybrid Distributed Graphics System", Eurographics Workshop on Parallel Graphics and Visualization, Saarbruecken, Germany, 2002

[Rud00a] Rudrajit Samanta, Thomas Funkhouser, Kai Li, Jaswinder Pal Singh. Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs. Eurographics/SIGGRAPH Graphics Hardware Workshop 2000.

[Mat00a] Matthew Eldridge Homan Igehy Pat Hanrahan [2000]. Pomegranate: A Fully Scalable Graphics Architecture. In: Proceeding of ACM SIGGRAPH 2000,

[Whe85a] Whelan Daniel. Animac: A Multiprocessor Architecture for Real-Time Computer Animation, Ph.D. dissertation, California Institute of Technology, 1985.

[Whi94] S. Whitman. Dynamic Load Balancing for Parallel Polygon Rendering. IEEE Computer Graphics andApplications,14(4),1994

[Car95] Carl Mueller [1995]. The Sort-First Rendering Architecture for High-Performance Graphics. In Proceedings of the 1995 Symposium on Interactive 3D Graphics, pages 75-82, Apr. 1995.