159.735 Parallel computing Seminar

Parallel Rendering – Fast Graphics

Lei Sun

5/28/2009

Abstract

This seminar paper briefly reviewed and discussed the basic concepts of parallel rendering in computer graphics, the importance of parallel rendering, and the different types of rendering parallelism. The paper also explored the algorithms in parallel rendering and compared the advantages and disadvantages of the sort-first, sort-middle and sort-last algorithm.

Table of Contents

Introduction

In computer graphics, rendering can be defined as the process of converting a model into an image by means of computer programs.

The image is a digital image or raster graphics image. The model is an abstract description of a collection of geometrically-defined objects in 3D space, which would contain information such as geometry, viewpoint, texture, lighting, and shading etc.

The rendering operation projects the graphic objects in the model into a 2D image space, computes the color intensities of individual pixels of the image, and finally generates the image. (Crockett, 1997)

For complex objects or high quality images, the rendering process requires massive computational resources. For example, the rendering of medical visualization, iso-surface generation, and some CAD applications may need millions or billions of floating-point and integer operations for each image.

Some image generating techniques like ray tracing, 3D textures, etc., work very slowly in simple machines.

To obtain the required computing power, the only practical solution is to exploit multiple processing units to speed up the rendering process. This technology is known as parallel rendering. (Wikipedia)

Parallel rendering can greatly improve the performance of computer graphics applications. In parallel rendering, the work is divided, distributed to computers and performed simultaneously in parallel. Considering a ray-casting application, without parallel, rays would be sent one by one to all the pixels in the view frustum; if parallel rendering is used, the whole frustum can be divided into a number of parts; we can then run the same number of processes to send rays in parallel to different parts. A cluster of machines are usually used to fulfill such a task.

With parallelization, high-quality image of great complexity can be easily produced, and high image processing efficiency can be achieved. Parallel rendering has been widely adopted by every image generation technique in the field of computer graphics. These techniques include surface and polygon rendering, terrain rendering, volume rendering, ray-tracing, and radiosity.

The types of parallelism employed in the rendering process are various; functional parallelism, data parallelism, and temporal parallelism are the three basic types; they can further be combined into different types of hybrid system.

Sort-first, sort-middle, and sort-last are the most discussed parallel rendering algorithms. This classification is based on the point in the rendering pipeline at which the object-space to image-space mapping occurs.

There are many parallel rendering applications and frameworks, such as Chromium, Equalizer, and OpenSG etc.

1 Types of Parallelism

Several different types of parallelism may be employed in the rendering process at different levels. These include **functional parallelism**, **data parallelism**, and **temporal parallelism**. These basic types can further be combined into hybrid systems.

Functional parallelism

The rendering process can be divided into a number of distinct functions; which forms a feed forward rendering pipeline. This pipeline usually consists of two parts:

1) Geometry processing: modeling transformation, back face culling, clipping, lighting, and view transformation.
2) Rasterization: scan-conversion, shading, z-buffer compare and store, and visibility determination.

In the case of functional parallelism, these functions are applied in series to individual data items. Functions are assigned to different processing units; once a processing unit completes its work on a data item, it sends the data item to the next unit and receives a new item from its upstream neighbor.

Functional parallelism can speed critical calculations, after the rendering pipeline is filled; the degree of parallelism achieved is proportional to the number of functional units.

The functional parallelism is suitable for polygon and surface rendering applications. We feed the 3D geometric objects into the beginning of the pipeline, and retrieve the final pixel values from the end of pipeline.

There are two limitations of functional parallelism; firstly, the overall speed of the pipeline is constrained by the slowest function unit, avoiding bottlenecks is a critical design issue of the functional parallelism. Another disadvantage of functional parallelism is that the available parallelism depends on the number of stages in the pipeline; due to the number of functions divided is quite limited, functional parallelism is unable to produce very high performance. (Crockett, 1997; Girkar, 1992; Rajagopalan, 2005)

Data parallelism

In contrast to functional parallelism, data parallelism focuses on distributing the data across different processing units. (Figure 1)

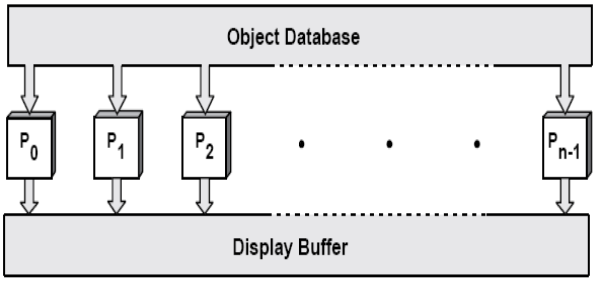


Figure 1 – Data Parallelism(Crockett, 1997)

The graphics data are divided into a number of streams; multiple data items can then be processed simultaneously on different processing units, and the results are merged to create the final image.(Hillis, 1986)

There are two types of data parallelism: object parallelism and image parallelism. Object parallelism occurs in the geometry processing stage of the rendering pipeline; image parallelism occurs in the later Rasterization stage of the rendering pipeline. Data parallelism can take advantage of multiple processors, as well as lends itself to scalable implementations; most of the graphics software systems adopted this approach.

To avoid bottlenecks, usually object and image parallelism must be incorporated together; however obtaining the proper balance between them is difficult, since the workloads at each stage depends on many factors such as the scene complexity, average screen area, sampling ratio, and image resolution etc.(Schroeder, 1993)

Temporal parallelism

Temporal parallelism usually adopted by animation applications. In this kind of applications, a large number of images are generated for subsequent playback; parallelism is obtained by decomposing the overall rendering task in the time domain; and processors are assigned a number of frames to render, along with the data needed to produce those frames.

Multiple types of parallelism can be incorporated in a single system to form hybrid parallelism. For example, functional and data parallelism can be combined by replicating all or part of the rendering pipeline. Temporal parallelism may also be combined with the functional or data parallelisms. With the hybrid approach, the systems may produce higher performances.

2 Overview of Parallel Rendering Algorithms

In graphics rendering, some problems can be parallelized trivially, requiring little or no inter-processor communication, and contributing no significant computational overheads. Such algorithms are called **non-interactive parallel rendering** or embarrassingly parallel rendering.

Rendering algorithms exploiting temporal parallelism are in this category. In such applications, frames are distributed among available processors; one frame can be rendered on one processor, and multiple frames can be processed simultaneously. Rendering methods based on ray-casting may also be implemented as embarrassingly parallel due to the fact that pixel values are computed by shooting rays from each pixel into the scene. If each processor has fast access to the entire objects, then each ray can be processed independently with no inter-processor communication required.

In the category of **interactive parallel rendering**, there are different algorithms of distributing the rendering work. For example, **Sort-first** rendering decomposes the final image; each processor renders a 2D tile of the image. **Sort-last** rendering decomposes the primitives across all processors, and recombines the partially rendered frames. **Sort-middle** rendering redistributes rendered primitives in the middle of the rendering pipeline. **Pixel decompositions** divide the pixels of the final view evenly, either by dividing full pixels or sub-pixels; and no sorting of rendered primitives takes place since all rendering resources render more or less the same view. **DPlex** rendering distributes full, alternating frames to the individual processor. (Cox, 1995; Wikipedia)

The interactive parallel rendering algorithms usually introduce overheads which caused by some or all of the following reasons: 1) communication among tasks or processors; 2) delays due to uneven workloads; 3) additional or redundant computations; 4) increased storage requirements for replicated or auxiliary data structures.

In parallel rendering algorithms, the image and object data are partitioned among the available processors; and rendering can be considered as mapping from 3D object space to 2D image space. This mapping is not fixed, but depends on the modeling transformations and viewing parameters. When both the object and image data are partitioned among the processors, then at some point in the rendering pipeline, data must be communicated among the processors; each processor has to send data to, and receive data from other processors. Managing this communication is one of the main issues for parallel rendering algorithms. (Crockett, 1997)

3 Parallel Rendering - a Sorting Problem

As stated previously the rendering pipeline consists of geometry processing and rasterization. In the stage of geometry processing, parallelism is achieved by assigning each processor a subset of the objects in the scene. In the stage of rasterization, each processor is assigned a portion of the pixel calculations. The rendering can be considered as determining how each object affects each pixel. Since the modeling and viewing transformations are arbitrary, an object can fall anywhere on the screen. Therefore we can view rendering as a sorting problem, i.e. sorting objects to the screen. (Molnar, 1994; Steven Molnar 1994)

Most commonly, the sorting happens in three main locations of the rendering pipeline. The location of the sorting determines the parallel rendering algorithms; if it takes place during geometry processing, the algorithm is called sort first which redistributes raw objects before their screen-space parameters are known; if it takes place between geometry processing and rasterization, the algorithm is sort-middle which redistributes screen-space objects; if it takes place during rasterization, the algorithms is sort-last which redistribute pixels.

Sort-first

The sort-first algorithm distributes the graphic objects to processors during the geometry processing stage of the rendering pipeline.

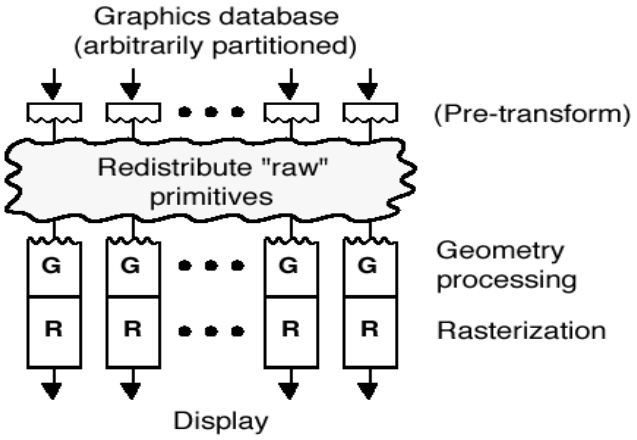


Figure 2 – Sort-first algorithm (Molnar, 1994)

Sort-first rendering decomposes the final view in screen space; and each processor renders a 2D tile of the final view. Sort – first is also called tile sort algorithm, each processor is responsible for all rendering calculations that affect the tile assigned to it.

The sort –first algorithm includes the following steps.

1) Objects are assigned to processors arbitrarily.
2) Processor computes the screen-space bounding box of the objects assigned to it and determines their respective tiles in screen space.
3) Redistribute objects to the appropriate processors.
4) Perform the remaining geometry-processing and rasterization calculations for the objects.

The main features of sort-first are: redistributing the objects at the beginning of the rendering process; and processors implement entire rendering pipeline for a portion of the screen. When the tessellation ratio and the degree of oversampling are high, it has low communication requirements.

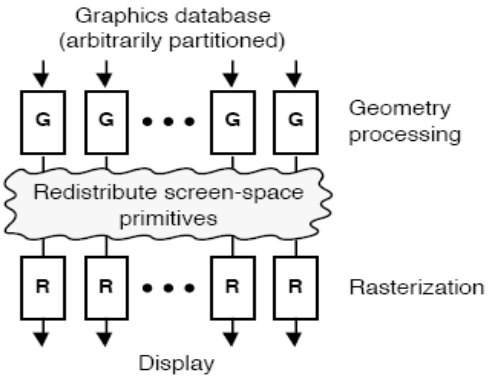
However the sort-first algorithm is susceptible to load imbalance. The objects may clump into regions, and the work load may fall onto a few processors. It has also a limited scalability due to the parallel overhead caused by objects rendered on multiple tiles. (Molnar, 1994; Mueller, 1995)

Figure 3 shows an example use of sort-first rendering on a video wall. Each computer in the video wall renders a portion of the viewing volume, and the final image is the summation of the images on the monitors that make up the video wall.

Figure 3 – Video wall.

Sort – Middle

In sort-middle algorithm, objects are redistributed in the middle of the rendering pipeline, i.e. between geometry processing and rasterization.



Before redistribution, the objects have been transformed into screen coordinates and are ready for rasterization.

Figure 4 Sort – middle algorithm (Molnar, 1994)

In the sort-middle rendering applications, the geometry processing and rasterization are usually processed on different processors. The two groups of processor can either be separate sets of processors or time-sharing processors. The objects are assigned arbitrarily to geometry processors; the tiles of the display screen are assigned to rasterization processors.

For each frame, the geometry processors perform the required calculations on the objects and classify them in terms of their screen regions. After that, geometry processors distribute the screen-space objects to their appropriate rasterization processor for further rendering.

Sort – middle algorithm is considered as general and straightforward; because the redistribution happens between geometry processing and resterization which is a natural place in the rendering pipeline.

The problem of sort-middle algorithm is high communication requirements when tessellation ratio is high. Another drawback of this algorithm is that it's susceptible to load imbalance between rasterization processors if objects distributed unevenly over the screen space. (Molnar, 1994; Williams, 2003)

Sort – Last

The sort-last algorithm decomposes the rendered objects arbitrarily across all rendering processors; each processor computes pixel value for its own subset of objects; and then transmits these pixels over an interconnect network to the compositing processors which receive the images created by the other processors, and then composite them into the final image.(Figure 5)

In sort-last algorithm, processors operate independently until the very end of the rendering pipeline; the sorting is deferred. This algorithm scales the rendering very well and less prone to imbalance.

However the re-composition step is expensive due to the amount of pixel data processed during re-composition. For applications rendering high-quality images, this can result in very high data rates on interconnect network,

There are two typical approaches for the sort-last algorithm: 1) SL-sparse which minimizes the communication by only distributing pixels actually produced by rasterization; 2) *SL-full which* stores and transfers a full image from each processor. (Moreland, 2001)

Figure 5 - A six pipe sort-last, direct-send configuration using Equalizer(Equalizer)

The sort-first, sort-middle and sort-last algorithms are widely adopted and implemented in many applications and frameworks. For example, **Chromium** which is a system for interactive rendering on clusters of graphics workstations; sort-first and sort-last may be implemented with it.(Chromium; Paul, 2004) Another well-known open source rendering framework is **Equalizer** which is a middleware to create parallel OpenGL-based applications and support both sort-first and sort-last algorithm. (Equalizer)

Conclusion

Parallel rendering is a very important approach in computer graphics. By exploiting the power of multiple processors, many applications such as real-time simulation, animation, virtual reality, photo-realistic imaging, and scientific visualization benefit from the use of parallelism to increase rendering performance. There have been many different types of parallelism and algorithms implemented and explored. A successful parallel rendering application must take into account the cons and pros of these techniques. Although parallel rendering techniques have been successfully employed in many applications; challenges still remain to improve the scalability, load balancing, communication, and image composition of these algorithms and produce higher performance.

References

Chromium. Retrieved May 2009, from http://chromium.sourceforge.net/

Cox, M. B. (1995). *Algorithms for parallel rendering. Ph.D. dissertation, Department of Computer Science, Princeton University, 1995.*

Crockett, T. W. (1997). An Introduction to Parallel Rendering. *Parallel Computing, 23*(7), 819-843.

Equalizer. Retrieved May 28, 2009, from http://www.equalizergraphics.com

Girkar, M. B. (1992). *Functional parallelism: theoretical foundations and implementation.* University of Illinois at Urbana-Champaign.

Hillis, W. D. S., Guy L., (1986). Data Parallel Algorithms. *Communications of the ACM, December 1986*.

Molnar, S., Michael Cox , David Ellsworth , Henry Fuchs (1994). A Sorting Classification of Parallel Rendering, IEEE Computer Graphics and Applications. *IEEE Computer Graphics and Applications, 14*(4), 23-32.

Moreland, K. W., B. Pavlakos C. (2001). *Sort-last parallel rendering for viewing extremely large data sets on tile displays.* Paper presented at the Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics.

Mueller, C. (1995). The sort-first rendering architecture for high-performance graphics. *Proc. 1995 Symp. Interactive 3D Graphics (ACM SIGGRAPH, 1995) 75-84*.

Paul, B. (2004). Chromium for Cluster Rendering
Retrieved May 20, 2009, from
http://graphics.stanford.edu/~mhouston/VisWorkshop04/ChromiumVis2004pt1.pdf

Rajagopalan, R. G., D.; Mudur, S.P (2005, April 2005). *Functionality Distribution for Parallel Rendering.* Paper presented at the Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International.

Schroeder, P. (1993). Data Parallel Volume Rendering Algorithms for Interactive Visualization. *The Visual Computer, 9*, 405-416.

Steven Molnar , M. C., David Ellsworth , Henry Fuchs (1994). A Sorting Classification of Parallel Rendering, IEEE Computer Graphics and Applications. *IEEE Computer Graphics and Applications, 14*(4), 23-32.

Wikipedia. Parallel rendering Retrieved May 28, 2009, from
http://en.wikipedia.org/wiki/Parallel_rendering

Wikipedia. Rendering (computer graphics) Retrieved May 10, 2009, from
http://en.wikipedia.org/wiki/Rendering_(computer_graphics)

Williams, J. L. H., R.E. (2003). *A proposal for a sort-middle cluster rendering system.* Paper presented at the Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2003. Proceedings of the Second IEEE International Workshop on Volume , Issue , 8-10 Sept. 2003 Page(s):36 - 38.