

Pipelined Sort-last Rendering: Scalability, Performance and Beyond

Xavier Cavin[†] and Christophe Mion (Inria - Alice)

Abstract

We present in this paper a theoretical and practical performance analysis of pipelined sort-last rendering for both polygonal and volume rendering. Theoretical peak performance and scalability are studied, exhibiting maximum attainable framerates of 19 fps (volume rendering with back-to-front alpha blending) and 11 fps (polygonal rendering with Z-buffer compositing) for a 1280×1024 display on a Gigabit Ethernet cluster. We show that our implementation of pipelined sort-last rendering on a 17-node PC cluster can nearly sustain these theoretical figures. We finally propose possible enhancements that would allow to go beyond the maximum theoretical limits. This paper clearly shows the potential of pipelined sort-last rendering for real-time visualization of very large models on standard PC clusters.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems-Distributed/network graphics; I.3.3 [Computer Graphics]: Picture/Image Generation Viewing algorithms; C.2.4 [Computer-Communication Networks]: Distributed Systems Distributed applications

1. Introduction and related works

As polygonal and volume datasets become bigger and bigger (see Figure 1), solutions based on sort-last rendering emerge as a mean to visualize them on PC clusters in place of dedicated high-end systems. These solutions include CEI/HP Parallel Compositing API [CEI, HPP] or Sandia Ice-T [MT03, PAR].

In sort-last rendering, the dataset to be displayed is decomposed and distributed across the different nodes of a PC cluster. For each new frame, each node renders a complete image of the data it has been assigned to, using its local GPU. Then it reads back the content of the frame buffer from the GPU to main memory. A parallel image compositing step is then performed to blend all the full resolution partial images into a final image; this step intensively uses the interconnection network to exchange parts of the composed image. Finally, the master node (responsible for displaying the final image to the user) gathers the final image from the slave nodes (which involves transmitting a full resolution image to a single node), and draws it from main memory to the frame buffer of the GPU.

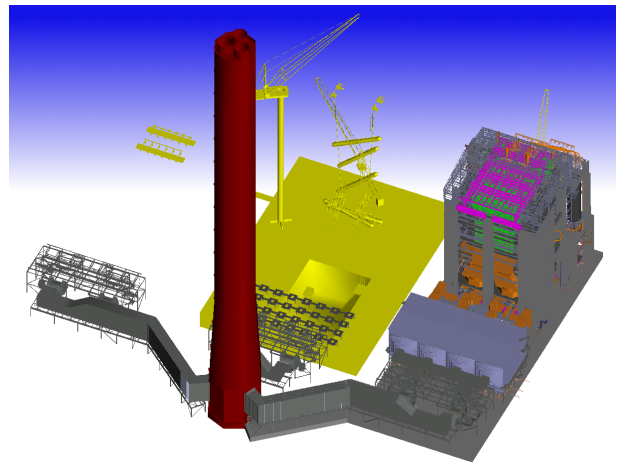


Figure 1: Real-time visualization of the Power Plant model (data courtesy of UNC-Chapel Hill): the model consists of 12,748,510 triangles and is rendered at 11 fps on a 1280×1024 display on our 17-node COTS PC cluster. No pre-processing has been applied to the initial model, and no acceleration technique has been used to speed-up the rendering (just straightly drawing the triangles).

[†] Xavier.Cavin@inria.fr

To our knowledge, the latest results for CEI/HP Parallel Compositing API report in [HP 05] 28.2 fps for a $512 \times 512 \times 512$ volume dataset and 20.6 fps for a 28 million triangle model on a 8-node PC cluster with InfiniBand 4x interconnect and a 1280×1024 screen resolution. Similarly, the latest results for Sandia Ice-T report in [HW05] 15 fps on a 264-node PC cluster with InfiniBand 4x (without unfortunately mentioning the screen resolution).

These figures on very high end PC clusters obviously serve as a reference of the obtainable performance of sort-last rendering. However, if we consider Commodity Off-The-Shelf (COTS) PC clusters, the available interconnection bandwidth is much lower compared to Infiniband. For instance, 650 MB/s are reported in [HP 05] for InfiniBand 4x, and have to be compared with the 128 MB/s bandwidth of standard Gigabit Ethernet.

Recently, Cavin *et al.* have proposed in [CMF05] a pipelined implementation of sort-last rendering on a COTS PC cluster. They reported peak performance of 19 fps for volume rendering and 11 fps for polygon rendering on a 5-node PC cluster with Gigabit Ethernet interconnect and a 1280×1024 screen resolution. If we put these figures into perspective with the latest results of the CEI/HP Parallel Compositing API using Infiniband 4x, they obtain 67% and 53% of the performance respectively for volume and polygon rendering, with only 19% of the interconnection bandwidth (not talking about the price ratio). Unfortunately, their experiments were limited to a small scale PC cluster (4 slave nodes), and the scalability of their approach still remains to be proved.

The goal of this paper is first to demonstrate the scalability of pipelined sort-last rendering, and second to suggest possible ways of acceleration. Section 2 of this paper presents a scalability and performance analysis of pipelined sort-last rendering, and applies the theory to practical cases. In Section 3, we present practical experimentations on our 17-node COTS PC cluster with a Gigabit Ethernet interconnect, and we show that the theoretical values can be attained. We conclude in Section 4 and present possible enhancements that could push pipelined sort-last rendering to new heights.

2. Pipelined sort-last rendering analysis

2.1. Theoretical performance evaluation

In this Section, we present a performance evaluation of the pipelined sort-last algorithm proposed by Cavin *et al.* in [CMF05].

According to their notations, the time needed to display a single frame using standard (non pipelined) sort-last algorithm is decomposed as:

$$time = render + read + compose + collect + draw \quad (1)$$

With their pipelined implementation, that overlaps:

- rendering (*render*) and parallel image compositing (*compose*) of the current frame with final image gathering (*collect*) and drawing (*draw*) of the previous frame;
- reading the frame buffer from GPU to main memory (*read*) with parallel image compositing (*compose*);

the time needed to display the same frame is now:

$$time = \begin{cases} compose & \text{if } render \leq compose \\ render & \text{if } render \geq compose \end{cases} \quad (2)$$

If we assume, as in [CMF05], that:

- we use an ideal COTS cluster with no latencies;
- we use one master node and n slave nodes;
- fps is the number of frames per second on a single node;
- xy is the number of pixels of the display, bpp is the number of bits per pixels of the color frame buffer and $zpth$ is the size in bits of the depth buffer;
- b_{op} is the bandwidth in bits per second of the operation op ;

and if we assume that the depth buffer is not collected at the end of each frame, then the terms of equation 2 can be rewritten as:

$$render = \frac{1}{fps} \quad (3)$$

$$compose = xy \times \left(1 - \frac{1}{n}\right) \times \left(\frac{bpp}{b_{sendandrecv}} + \frac{zpth}{b_{sendandrecvZ}}\right)$$

where $zpth$ can possibly be zero in the case of back-to-front compositing without the Z-buffer.

Note that equation 2 gives the minimum time to render a frame, in the case where overlap is completely attained. If some parts are not overlapped, then this time would be higher. This equation gives us an upper bound of the maximum obtainable frame rate.

2.2. Applying the theory

To get more concrete, we assume in the remaining of this Section that our application uses a RGBA 32 bits color buffer and a 32 bits depth buffer if required. We also assume an ideal PC cluster with a given interconnect exhibiting a point-to-point full-duplex bandwidth of b Gb/s and capable of an infinite aggregate bandwidth. We will come back later in Section 3 on this assumption.

If we consider that the rendering speed is infinite (*i.e.* $render = 0$), then *compose* gives us an upper bound of the maximal obtainable frame rate on our cluster. Figure 2 shows this upper bound for increasing resolutions, both on a Gigabit Ethernet ($b = 10^9$) and on a InfiniBand 4x ($b = 5.2 \times 10^9$) PC cluster, for polygonal (with Z-buffer compositing) and volume (with back-to-front compositing without Z-buffer) rendering.

The latest results for the CEI/HP Parallel Compositing

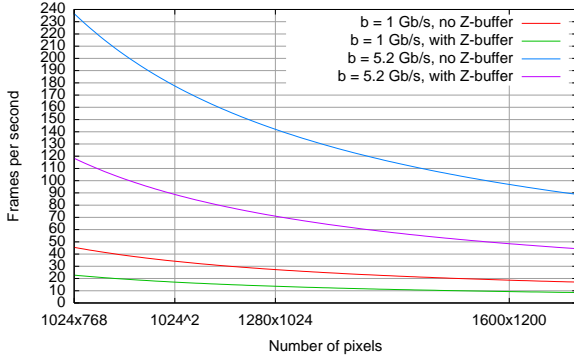


Figure 2: Theoretical optimal rendering speed of pipelined sort-last rendering in frames per second on a 17-node PC cluster, with different interconnects (Gigabit Ethernet $b = 1$ Gb/s, InfiniBand 4x $b = 5.2$ Gb/s), with zero rendering time ($render = 0$), with and without Z-compositing.

API reported in [HP 05], *i.e.* 28.2 fps for volume rendering and 20.6 fps for polygonal rendering on a 8-node PC cluster with InfiniBand 4x interconnect and a 1280×1024 screen resolution, are below the theoretical optimal performance of pipelined sort-last rendering, *i.e.* 140 fps and 70 fps. This clearly shows the potential enhancements that pipeline sort-last rendering could bring to their implementation. On the other hand, the 19 and 11 fps reported by Cavin *et al.* [CMF05] on a 5-node PC cluster with Gigabit Ethernet interconnect are closer to the theoretical maximum performance, *i.e.* 27 fps and 14 fps.

It is also interesting to study how the maximal obtainable frame rate varies with different rendering speeds (*i.e.* different values of $render$), as illustrated on Figure 3: the curves compare the theoretical performance of the pipelined sort-last algorithm compared to the classical sort-last algorithm with or without taking into account the Z-buffer. They clearly show the gain of pipelined sort-last rendering as compared to classical sort-last rendering (up to 300% for the case without the Z-buffer).

3. Experimentation

3.1. Experimental setup

We have implemented the pipelined sort-last algorithm on Linux in C using OpenGL, Pthreads and TCP/IP sockets. We have run our experiments on a 16-node COTS PC cluster. Each node is equipped with:

- a bi dual core AMD Opteron (275) 2.4 GHz;
- a NVIDIA GeForce 6800 Ultra 512 MB;

and nodes are interconnected through a Gigabit Ethernet interconnect.

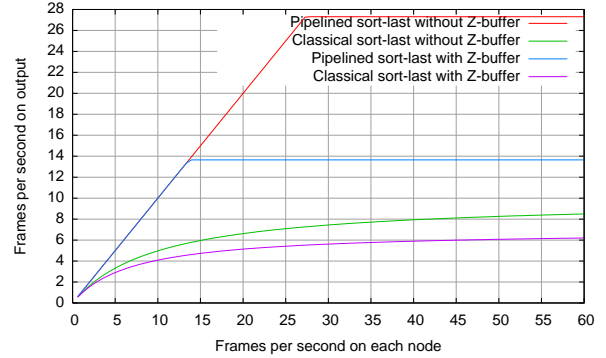


Figure 3: Evolution of the theoretical optimal rendering speed in frames per second for different nodes rendering speeds on a 17-node PC cluster with Gigabit Ethernet interconnect ($b = 1$ Gb/s), a 1280×1024 display, with and without Z-compositing.

We have performed experimentations both with polygonal rendering (with Z-buffer compositing) and volume rendering (with back-to-front compositing without Z-buffer). For polygonal rendering, we have written a very simple OpenGL application that displays polygonal models stored in PLY format. For volume rendering, we have written another OpenGL application that displays volumetric raw data.

3.2. Black screen rendering

Our first benchmarks rely on a “black screen rendering” scheme: nothing is rendered ($render = 0$) and the black images are composited together (without optimization of any kind) and displayed on the master node. This allows to test the maximum attainable framerate on our PC cluster. For any number of slave nodes (from 2 to 16), we obtain the same output rendering speed: Table 1 reports the obtained performance for different resolutions and compares it to the theoretical maximal value. The results are very similar to those reported in [CMF05] on their 5-node PC cluster. They also point out that the overlapping of the different parts of the algorithm is not completely perfect, *i.e.* some parts are serialized.

The scalability of our implementation for PC clusters bigger than our 17-node one only relies on the capability of the interconnect to handle the simultaneous $\frac{n}{2}$ point-to-point full-duplex communications occurring during the parallel image compositing stage between the n slaves of the cluster. Indeed, in our analysis, we have made the assumption that the interconnect supports an infinite aggregate bandwidth. This is clearly and unfortunately not the case. We point out that a lot of care must be taken in the choice of the interconnection switch when building a large graphics PC cluster.

Resolution	Volume	Polygon
1024 × 768 (theory)	46 fps	22 fps
1024 × 768 (observed)	31 fps	17 fps
1280 × 1024 (theory)	27 fps	14 fps
1280 × 1024 (observed)	19 fps	11 fps

Table 1: Observed and theoretical output performance of our pipelined sort-last implementation for “black screen rendering” ($render = 0$) on our 17-node PC cluster in the volume (without Z-buffer) and the polygonal (with Z-buffer) rendering case.

3.3. Case study: the Power Plant model

As a final experiment, we have used our polygonal rendering application to visualize the famous Power Plant model (see Figure 1). This model is composed of 12,748,510 triangles distributed among several sections. In our application, we do not use any acceleration technique of any kind (no Level of Details, no culling, no impostors, no display list, no vertex array, ...): we simply draw each triangle one after the other by sending its three vertices to the GPU. Moreover, we have performed absolutely no pre-processing on the model. This allows the interactive manipulation of the triangles of the model at constant frame rate, as compared to more sophisticated rendering approaches relying on a static model.

The decomposition of the model across the different nodes is very simple: every single triangle that is read from a PLY file is distributed in a round-robin manner to the given nodes. This ensures an excellent load balancing, since for any point of view, the visible triangles are equally distributed among the nodes. On our 17-node PC cluster, each sub-model (composed of less than a million of triangles) can be rendered at 45 fps on each node. Not surprisingly, the whole Power Plant model can be displayed at the framerates reported in Table 1.

4. Conclusion and future works

In this paper, we have presented an analysis of the performance and of the scalability of the pipeline sort-last rendering algorithm introduced in [CMF05]. Our theoretical analysis allows the authors of related works to compare their observed performance with the theoretical maximum value, with respect to their cluster characteristics. It also shows that pipeline sort-last rendering is a very scalable approach, if one take care in the choice of the interconnection switch, which is a potential bottleneck when the number of nodes in the cluster increases. Finally, we have presented experimentations of our implementation of pipeline sort-last rendering on a 17-node COTS PC cluster. These experimentations have proved the scalability of this approach for a cluster of this size; our theoretical analysis makes us confident about the scalability for larger clusters.

However, the results we have obtained are below the theoretical maximum values, as shown by Table 1. This is due to the fact that some parts of the algorithm are not completely overlapped. In particular, we suspect that the exchanges occurring during the parallel image compositing and the send operations occurring during the final collect operation conflict in some way. We are currently investigating these parts to optimize the overlap. This should help us getting closer to the theoretical limit.

Then, the only way to increase the performance will be to increase the available bandwidth for the communications. For instance, Figure 2 shows the theoretical frame rate that could be achieved using an InfiniBand 4x interconnect. The drawback of InfiniBand is obviously its price, as compared to classical Gigabit Ethernet. We are currently investigating another (cheaper) way of increasing the available bandwidth: each node of our cluster is actually equipped with four Gigabit Ethernet attachments, which gives us a potential bandwidth of 512 MB/s, to be compared with the 650 MB/s of InfiniBand 4x. Our future works include finding solutions to make a full usage of this potential bandwidth.

A success in both directions (maximizing the overlapping and increasing the bandwidth) will give a scalable sort-last rendering solution capable of very high framerates with high resolutions.

References

- [CEI] CEI Insight Parallel Compositor. <http://www.insight.com/>.
- [CMF05] CAVIN X., MION C., FILBOIS A.: COTS cluster-based sort-last rendering: Performance evaluation and pipelined implementation. In *Proceedings of IEEE Visualization 2005* (2005).
- [HP 05] HP VISUALIZATION TEAM: Compositing using COTS components, 2005. <http://www.hp.com/techservers/hpccn/downloads/HP-Compositing-public.pdf>.
- [HPP] Advanced visualization collaboration. http://www.hp.com/techservers/hpccn/sci_vis/.
- [HW05] HIGHAM D., WYLIE B.: Sandia National Labs achieves breakthrough performance using NVIDIA technology for scientific visualization, 2005. NVIDIA press release, http://www.nvidia.com/object/IO_19962.html.
- [MT03] MORELAND K., THOMPSON D.: From cluster to wall with VTK. In *Proc. of IEEE 2003 Symp. on Parallel and Large-Data Visualization and Graphics* (2003).
- [PAR] Paraview - parallel visualization application. <http://www.paraview.org/>.