A Survey on Distributed Visualization Techniques over Clusters of Personal Computers

JOSE FERNANDO RODRIGUES JR., ANDRE R. BALAN*, LUCIANA ZAINA, AGMA J. M. TRAINA+

Universidade Federal de São Carlos at Sorocaba, Universidade Federal do ABC*,
Universidade de São Paulo at São Carlos+
[junio,lzaina]@ufscar.br, andre.balan@ufabc.edu.br, agma@icmc.usp.br

Abstract. In the last years, Distributed Visualization over Personal Computer (PC) clusters has become important for research and industrial communities. They have made large-scale visualizations practical and more accessible. In this work we survey Distributed Visualization techniques aiming at compiling last decade's literature on the use of PC clusters as suitable alternatives to high-end workstations. We review the topic by defining basic concepts, enumerating system requirements and implementation challenges, and presenting up-to-date methodologies. Our work fulfills the needs of newcomers and seasoned professionals as an introductory compilation at the same time that it can help experienced personnel by organizing ideas.

Keywords: Distributed visualization, scientific visualization, computer graphics, distributed computing

(Received July 16, 2009 / Accepted October 16, 2009)

1 Introduction

Visualizing huge datasets demands processing power that surpasses commonplace workstations. To cope with this need, high-end systems using Symmetrical Multi-Processors technology (SMP) [2] have been commercialized. Alternatively, during the last decade, a great number of works have discussed matters of design and implementation of visualization over computer clusters. These works constitute the Distributed Visualization discipline and range from complete systems to prototypes and theories to better utilize distributed computation.

According to the Top 500 Supercomputer list [9], currently more than 40 percent of the fastest computers in the world are clusters of networked computers. Among these clusters are the Quake project [8] at the Pittsburgh Supercomputing Center, which is composed of hundreds of proprietary systems in a cluster of workstations reaching top generation performance.

Concomitant to this process, the PC commodity industry has furiously evolved in the last years tending to continue at this pace, see figure 1. Advances in storage devices, processing power, memory speed, graphical buses and frame buffers have doubled every period around two years or less. These resources lead to a graphical power that was formerly infeasible. Today, a commodity U$1500 workstation has graphics capabilities that exceed those of a late 1990s U$500K supercomputer. These advances, together with network improvements, made it possible to build PC clusters to rival high-end machines.

An up-to-date benchmark [18] from the National Aeronautics and Space Administration agency (NASA) demonstrates not only that commodity computer clusters rival to SGI workstations, but also that the maintenance cost of these workstations is sufficient to build a new PC cluster every year. Thus, in this work, we compile the progresses in Distributed Visualization focusing on clusters of PCs. We build a condensed document aiming at organizing ideas and discussing unsolved issues.

In this text we review the complexities to consider when designing and implementing Distributed Visualization over clusters of PCs. At the same time, we

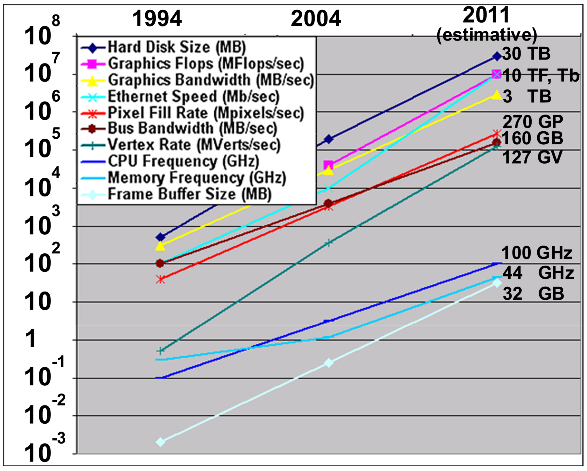

Figure 1: Various aspects of commodity hardware evolution and tendency for a single processing node equipped with a top generation Graphics Processing Unit. Except for the network speed data, all the data were extracted from Fernando's seminar [19].

present the advantages of this platform, including low costs for supercomputing, the ability to track technologies, systems that can be incrementally upgraded, open source software and vendor autonomy.

The organization of the text is as follows. Section 2 introduces the Distributed Visualization topic and section 3 presents a set of issues related to implementing clusters of PCs. Section 4 presents libraries that intend to abstract the assembling of such systems and section 5 concludes the paper.

2 The Visualization Pipeline

Visualization is organized according to the model cast by two works, Upson *et al* [48] and Haber and McNabb [25]. Their model, presented in figure 2, describes three stages to achieve a visualization. In a pipeline structure, each stage executes a distinct processing whose output feeds the next stage.

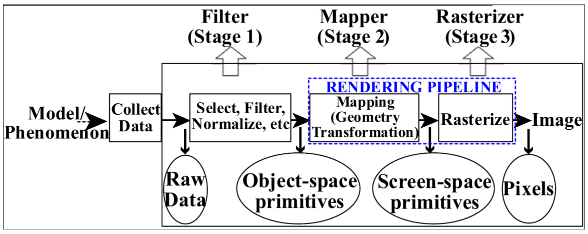

Figure 2: Three stages named filter, mapper and rasterizer compose the complete visualization pipeline. The first of them may be omitted, executed before or along the visualization process. In such case, only the mapper and the rasterizer stages compose the process. These two stages determine the core of the visualization pipeline, which is called *rendering pipeline* (dashed rectangle).

After an initial data collecting stage, the pipeline proceeds with the raw data volume that is processed according to the purposes of the intended visualization. In this stage, named filter (preprocessing or traversal), the dataset is selected, filtered, cleaned, enriched, summarized, normalized, and/or submitted to any useful processing to optimize the rendering process, e.g., culling operations. Then, as illustrated in figure 3, the prepared data (object space primitives) are submitted to a mapping procedure (geometry transformation) to determine how the data will be displayed in the form of geometrical entities (screen-space primitives), e.g., tesselation data, an isosurface or a contour map. The last stage, named rasterizer (or renderer), applies operations like projection, lightening and/or shading to the screen-space primitives in order to finally generate the image (pixels). The whole process is known as *visualization pipeline*, while the last two stages are known as *rendering pipeline*. We widely refer to the above concepts along the text.

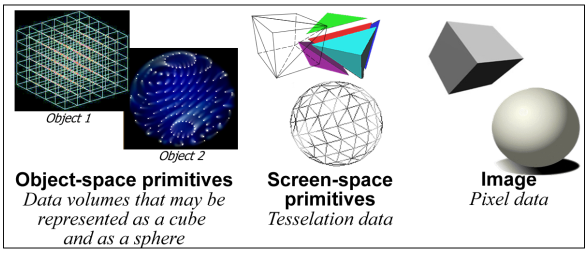

Figure 3: Illustration of data that may take part of a visualization pipeline. Two objects, a cube and a sphere, their correspondent volumes and tesselation data followed by renderization.

In the Distributed Visualization domain, the stages of the visualization pipeline are not restricted to a single machine or location. Each step may take place locally or remotely, at the client that will analyze the data or at the server that owns the processing power. Accordingly, we define Distributed Visualization as the use of distributed resources to drive visual analysis. It is expected that such systems present the possibility of disjoining data and exploration sites, the possibility of combining autonomous processing resources and the possibility of collaborative work.

Advances in commodity hardware have led to the use of PC clusters as an attractive option for Distributed Visualization. To explore these prospects, algorithms have been proposed to deal with memory and communication constraints in distributed environments. This approach increases the limits of a single PC, but it has to tackle with load balancing and inter-process communication, among other problems, that we discuss in this text.

3 Design and Implementation Issues for Distributed Parallel Rendering Systems

Distributed Visualization systems have been oriented to PC clusters (distributed memory) architectures that concern parallel rendering in distributed systems. Motivations include: the low cost of commodity PC hardware is far smaller than that of high-end visualization systems; PCs are all-purpose machinery and can also be used for non-graphical applications; it is possible to benefit from the standards of the PC market, which allows for a continuous upgrade with reduced effort; the open-source movement provides high quality software at low costs; and, it is possible to add more PCs to the system in order to bear with power increase demands.

The main consideration for Distributed Visualization over PC clusters is the algorithm that binds the distributed resources into the *visualization pipeline* defined in section 2. Such algorithms fall into one of three categories: sort-first, sort-middle and sort-last (subsection 3.1), having as their design goal concerns on load balancing and communication constrains among the nodes of the cluster (subsection 3.2). Supporting this structure lies network technology (subsection 3.3), techniques for data management (subsectoin 3.4) and techniques for parallel storage (subsection 3.5). Another concern is the operating system over which the distributed visualization ensemble will execute (subsection 3.6). Alternatively, distributed parallel rendering libraries offer high-level implementation, as reviewed in the next section.

3.1 Algorithms for parallel tiled Distributed Visualization

In reference to the foundational visualization pipeline described in seciton 2, Molnar *et al* [36] formulate the most accepted classification for distributed parallel rendering algorithms. Their analysis determines how the visualization pipeline (geometric transformation followed by rasterization) maps onto a general parallel algorithm. According to the theory, the parallelism is achieved through the assigning of transformation tasks (on object-space primitives) and rasterization tasks (on image-space primitives) to the distributed processing units.

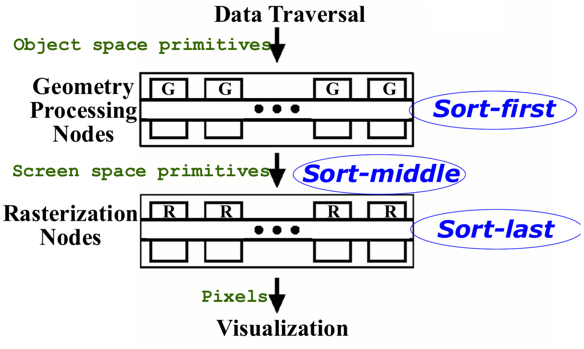

Figure 4: Algorithms for distributed rendering rely on the decision of where/when the parallel tasks will be designated for the processing units. This is considered a sorting/classification problem. There are three possibilities for this sorting operation, each of which deeply characterizes the correspondent algorithms.

The idea is that the complex modeling of visualization scenes, geometrical entities (object-space primitives and screen-space primitives) and rendered pixels can be distributed in different ways across the processing resources. The design task, thus, worries about how to assign the data portions of the pipeline so that the intended image can be achieved at the end. At the same time, processing load and network communication constraints must be satisfied. Sutherland *et al* [44] consider it a sorting (or classification) problem. This sorting can occur in one of three moments, as presented in figure 4. After the sorting is complete, the data need to be redistributed among the nodes to reflect the new-sorted arrangement.

The classes defined when considering the rendering pipeline and the sorting problem are named sort-first, sort-middle and sort-last. They differ by terms of bandwidth requirements, amount of duplicated work and load balance.

Sort-first (image (or pixel) space parallelism)

In sort-first, the screen is divided into disjoint regions (tiles) that are assigned to the processing nodes, as illustrated in figure 5. To do so, the objects primitives are submitted to a minimum geometry transformation (screen-space bounding box calculation) necessary to determine which tile of the screen they overlap. Then, the objects primitives are transmitted to the processing nodes (renderers) that correspond to the tiles of the

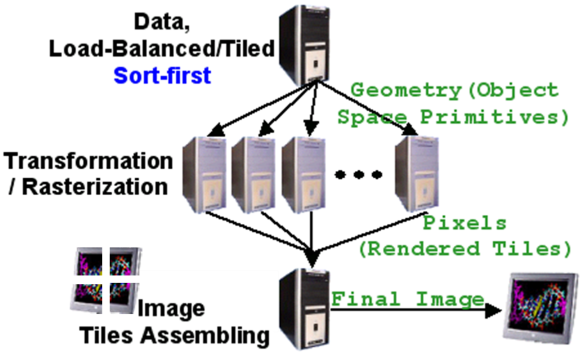

Figure 5: Classical sort-first configuration. Based on tiled partitioning information, the data is distributed among the processing nodes. After transformation and rasterization, a final step assembles the tiles to form the image.

screen, where both transformation and rasterization are performed. Finally, the rendered tiles are reassembled to form the final scene.

The initial geometry transformation phase of sort-first is called pre-transformation. Due to this extra processing, sort-first is the most expensive design for distributed visualization, but also the least bandwidth consumer. Examples include the work by Zhu *et al* [50] and the work by Mueller [39]. The later presents a study about sort-first implementation and its advantages, mainly the frame-to-frame coherence (high for interaction sequences) and the lower bandwidth demand.

Sort-middle

It is the natural approach for distributed parallel rendering because transformation and rendering are performed at different levels of the cluster, see figure 6. Initially, the algorithm distributes object space primitives among the nodes according to some load balance method, e.g, round robin. Then, after geometry processing, the resultant screen space primitives are distributed to the rasterization nodes. Similar to sort-first, the algorithm assigns tiles of the screen to specific processing nodes but, differently, there is no pre-transformation step.

The disadvantages occur when the tessellation ratio is high. Tessellation refers to the decomposing of larger primitives into smaller ones. It determines that the system must redistribute several display primitives instead of just one object primitive. For sort-middle, high tessellation ratios imply in higher communication costs.

Another disadvantage of sort-middle is the load imbalance on the rasterization units if the primitives are unevenly distributed over the screen, what also may occur to sort-first algorithms. Also, according to Mueller [39], the loose coupling in the middle of the pipeline can limit feedback from the rasterization stage back to the transformation stage, what makes certain visibility culling algorithms either less efficient or infeasible. Montrym *et al* [37] present a custom-designed implementation of this parallelism and Ellsworth [16] makes an extensive review of sort-middle systems.

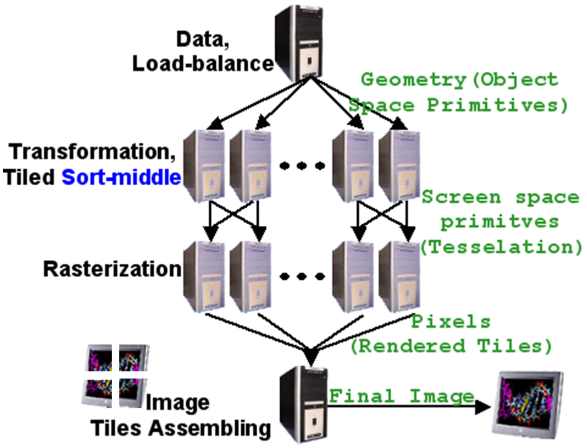

Figure 6: Sort-middle configuration with two levels of processing nodes. The screen-space data communication may also occur in between nodes at the same level, determining a one-level-only structure, where transformation nodes also act as rasterization nodes.

Sort-last (object-space parallelism)

In this case, the sorting occurs after the end of the rendering pipeline, that is, the pixels are ready to compose the image, as presented in figure 7. In a load-balanced manner, the processing nodes (renderers) receive arbitrary subsets of the object-space primitives. After transformation and rasterization, the resultant pixels are submitted to a composition procedure. At this final step, sort-last will have produced a set of full-screen images (sort-last-full) or a set of screen-space primitives (sort-last-sparse). These images are recomposed by hardware or software that compute every sample at each pixel to define the primitives' visibility. This is called (depth) sorting and, according to Foley *et al* [20], it relies on the use of Z-buffering. Thus, the processing nodes must send, along with the pixels, the correspondent Z-buffers. This need highly increases the required bandwidth to the order of gigabytes per second.

The advantages of sort-last are the better control of load balance concerned to the object-space primitives and the simplicity of the approach because the processing nodes perform the full pipeline independently. Implementation examples include the works by Lombeyda *et al* [33] and by Morel *et al* [38].

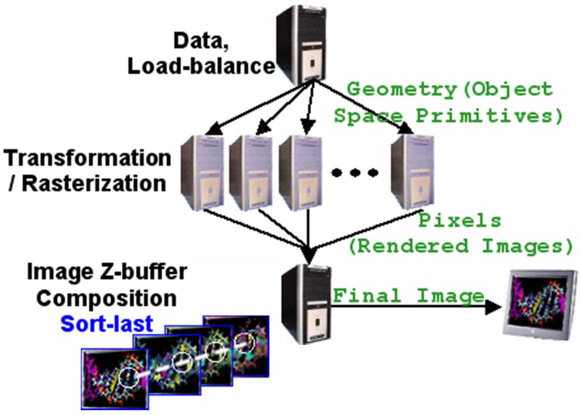
Data,
Load-balance
Geometry (Object Space Primitives)
Transformation / Rasterization
Pixels (Rendered Images)
Image Z-buffer Composition
Sort-last
Final Image

Figure 7: Sort-last-full configuration. The initial distribution of tasks considers only load-balancing directives. In a second step, each processing node is responsible for the complete rendering of a full screen image with only a subset of the objects that compose the visualization. At the final step, each screen is superimposed following depth information in order to have the correct visibility of the visual entities.

In the literature, each author advocates for her/his choice considering the most suitable features for sort-first, sort-middle or sort-last. More recent works point to sort-first and sort-last to be used with PC cluster implementations. Sort-middle is used with high-end shared-memory systems as SGI's hardware, probably because fast memory buses are less influenced by the overheads of sort-middle. For comparison, in table 1 we present an overview of the three possibilities.

Hybrid approaches are also possible, as done by Samanta *et al* [43] and Garcia and Shen [21]. The former work tries to minimize the sort-last composition overhead by means of a dynamic sort-first partition. Their approach benefits from a view-dependent partitioning of both the 3D model and the 2D screen. The later work leverages the advantages of both sort-first and sort-last approaches with a hybrid-sorting of both image and data partitioning for load balance.

3.2 Load balancing

Load balance applies distinctly for image parallelism (sort-first, sort-middle) and object parallelism (sort-last). In object parallelism, object distribution-rules define how to reach nearly equal loads among the processing nodes. In image parallelism, load balance is based on screen partitioning methods.

Ellsworth [16] points that, for object parallelism, random or round robin approaches are used to distribute objects among the processing nodes. These techniques work fairly well for objects with homogeneous size and complexity. For objects with great differences, the time to process them may vary by a large factor. Further possibilities for load balancing consider the geometry as hierarchical structures or as sets of primitives (flat structures), this topic is reviewed by Ellsworth *et al* [17].

For image parallelism, if not equal portions of the image are assigned to the processing nodes, some of them will remain idle while waiting for others to finish their task. This problem is treated by screen dividing methods, as exemplified in figure 8, which can be static or dynamic.

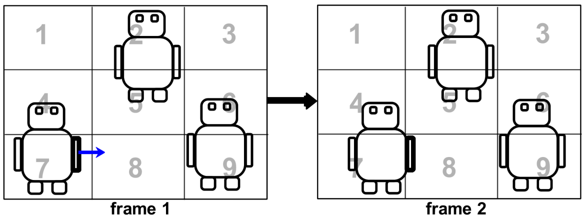

Figure 8: Screen-partitioning with 9 tiles. By means of frame coherence, only the primitive highlighted in tile 7 must be sent in order to draw tile 8 in frame 2, which was formerly empty. The other primitives remain in memory via retained mode operation.

Static screen dividing methods divide the screen into more regions than the number of processing nodes and assign the regions in an interlaced fashion to these nodes. The number of regions per node is called granularity ratio. For low granularity the workload may not be balanced. For high granularity, we may have a high overlap factor (the average number of regions overlapped by the primitives), what leads to network overload in image parallelism. If a primitive lies over three regions, the entire primitive must be transmitted three times for transformation and rasterization because, even if just a small piece overlaps a tile, its computation depends on the entire primitive. According to Molnar *et al* [36], if we assume equal sized primitives and equal probability for the positioning of these primitives on the screen, the overlap factor is given by:

$$Overlap = ((R_{weight} + P_{weight})/R_{weigth}) * ((R_{height} + P_{height})/R_{height}) \quad (1)$$

Where R_{weight}, R_{height}, P_{weight} and P_{height} are, respectively, the weight and the height of a given screen region and the weight and the height of a given primitive bounding box.

Table 1: Overview of the main algorithm possibilities for Distributed Visualization.

Feature	**Sort-first**	**Sort-middle**	**Sort-last**
Parallelism	image	object / image	object
Sorted data	object space primit.	screen space primit.	pixels (z-buffer)
Bandwidth demand	low	medium	high
Overhead factor	pre-transform./overlap	high overlap	image composition
Frame coherence	yes	no	no

Dynamic (adaptive) screen dividing methods work by computing statistics from on-screen primitives distribution in order to intelligently determine and assign the tiles. These algorithms add overhead to the visualization process first due to the gathering of statistics and decision making they implement; and second due to the more elaborated screen division. Two common approaches for adaptive screen division are: first to statically partition the screen and then dynamically allocate them to the processing nodes; another method is to first settle a constant number and assignment of regions and then dynamically vary their shape, as done by Roble [42]. Also, dynamic partitioning combined with dynamic assignment is possible, as proposed by Mueller [39]. Finally, a comparison of algorithms for space division is available in the work by Kurc *et al* [30].

3.3 Network issues

The great disadvantage of PC clusters if compared to shared bus (SMP) systems is that data sharing does not occur over a high speed direct access memory bus, but over a much slower network. Thus, network factors demand special attention in order to reach effective bandwidth and network latency performances. Bandwidth corresponds to the available data/time transmission. Network latency is the time to prepare and transmit the data between two nodes.

Bandwidth constraints are affected by the network speed, the data-network adapter speed, the bus interface and the memory. That is, the bandwidth is a function of the data traveling time, receiving time, in-node transfer time and memory storage time.

Meanwhile, the network latency varies with the network interface that sends/receives the data, the bus interface to in-node read/transfer the data, the memory architecture to access/store the data and the processing power available to decide and perform the whole process. High network latency times barely affects scarce long message communication, e.g. Internet browsing, but it is decisive for communication characterized by plentiful short messages, as required by computer clusters.

These factors must be designed to maximize the bandwidth at the same time that the latency be minimized. Together with high quality hardware and system architecture, an appropriate network must be settled. A suitable practice is to isolate the cluster in a network in which traffic is limited to the cluster communication. This setting constitutes a System Area Network (SAN). In such systems, the hubs must have minimal retransfer latency and the interconnection of different networks must be avoided due to higher latency. For optimization, a switch device, instead of a hub, can serve the network so that intelligent directional ports permit parallel communication. Following we review major factors to come up with a suitable network structure for distributed parallel rendering.

The Message-Passing Interface (MPI) standard

The MPI standard [24], is a message-passing library being developed since 1992. A message passing library is a high-level abstraction that permits intercommunication within a collection of autonomous processes each of which with its own local memory. It eases the implementation of shared memory and distributed shared memory systems, which are the foundation of computer clusters parallelism. There are several message-passing libraries, but the MPI standard is the *de facto* convention for clusters.

MPI ranges from supercomputing to PC clusters. It is a standard and not a product, implementations of it are available for several operating systems and network technologies. Liu *et al* [32] present a performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics technologies (detailed further in this section). Gropp *et al* [23] describe the MPICH, a portable implementation of MPI ("CH" stands for "Chameleon"). MPICH is the most used free distribution of MPI and its design goal is to combine performance with high portability within a single implementation. Another popular implementation of MPI is Romio [45], with broad portability and free on-line support.

Via technology

In stacked-based communication protocols, as TCP/IP, great part of the latency is caused by the in-node operation. When a process in a cluster node wants to transmit, it prepares the data and makes a high-level call to some network API that access the network hardware. Then, when the respective interruption request (IRQ) is issued, the OS copies the data to some other memory space used as buffer. This buffer, finally, is read by the network hardware that transmits it via the physical layer. This process consumes a large amount of time per data transmission, lowering the process by up to orders of magnitude.

To lessen this problem, it was conceived the Virtual Interface Architecture (VIA) [13]. VIA, created by Intel, Compaq and Microsoft, describes an alternative interface between network hardware and user software. This interface provides direct access to the network hardware (user level communication protocol), what lowers the transmission latency by avoiding the OS intermediation (zero-copy protocol). VIA is accomplished by hardware integration on the Network Interface Card (native implementation), or by software emulation. The former achieves the best performance, the later consumes extra processing, but even though its latency performance surpass that of regular network usage, according to a IEEE report [1].

Cameron and Regnier [13] present complete information about VIA. Baker *et al* [10] describe a study on VIA performance gains over Gigabit Ethernet. In [29] it can be found information about the MVICH (MPICH for Virtual Interface Architecture), a popular implementation of MPI on top of VIA technology. Also in [29] it is described the Modular VIA (M-VIA) a high performance implementation of VIA for Linux systems. The use of VIA is a straight method to diminish the effects of network latency, which is the main limitation in computer clusters.

Network technologies

Transmitting 3D objects over the network, as for sort-middle algorithms, can compromise the available bandwidth. Thus, research is performed to devise geometry compression algorithms, as done by Touma [47]. These algorithms reduce bandwidth requirements to up to 10 bits per vertex, including connectivity. However, the required bandwidth and network latency still can overscale commodity Ethernet networks. To cope with that, high-performance interconnection technologies are used. We list them on the next paragraph.

Myrinet [12] is a packet-communication and switching technology used to interconnect clusters of workstations, it offers up to 2 Gigabit/second full duplex links and it is based on the ANSI (American National Standards Institute) Standard ANSI/VITA 26-1998. The Quadrics technology [41] reaches up to 8.5 Gigabit/second full duplex rates on top hardware systems provided with the QsNet II network. InfiniBand [5] is steered by an association of member companies involved in performance computing, data center and storage implementations. It offers up to 30 Gigabit/second channels. The Scalable Coherent Interface (SCI) network, which is an ANSI/ISO/IEEE Standard (1596-1992), has been specifically designed to computer clusters. With reduced network latency time, SCI behaves like a bus or a network using point-to-point links to achieve higher speed. It implements a cache scheme as a coherent virtual shared memory. Its Dolphin [4] release reaches up to 2.6 Gigabit/second rates.

In table 2 we present an overview of these technologies together with the Gigabit Ethernet commodity technology. The data are only for rough comparison because a number of other factors may influence the performance and costs.

All these technologies have support for VIA (native or emulated) and for implementations of MPI. The choice for one of them depends on other factors such as compatibility with the cluster equipment and operating system, performance and price. Latency is decisive for massive short message communication, thus, the ill latency performance of Ethernet makes it the worst choice. Quadrics and Infiniband present superior bandwidth coupled with very attractive latency times. The drawback is the elevated price of these options. More adequate alternatives are Myrinet and SCI networks. Myrinet has already been widely used for clustering, while SCI presents the best latency performance. In [1] it is presented a wide description of these technologies and Yeo *et al* [49] present a benchmark-oriented study about the topic.

3.4 Data management

Distributed Visualization deals with terabyte order datasets over heterogeneous platforms. The storage and utilization of this information have specific implications, specially the required physical space, the I/O tasks to be performed in suitable time and the applications' expected data format. Therefore, three efforts have emerged as leading initiatives to determine standards in scientific large volume data management: HDF [7], CDF [6] and netCDF [31].

Table 2: Network technologies overview.

Network technology	Bandwidth (MB/s)	Latency (μs)	Avg. Price/Port (U$) [40]
Gigabit Ethernet	< 125	< 100	~ 300.00
10 Gigabit Ethernet	< 1250	< 60	~ 7000.00
Myrinet [12]	< 250	< 10	~ 400.00
Quadrics QsNet II [41]	< 1064	< 3	~ 2000.00
Infiniband [5]	< 3750	< 7	~ 800.00
SCI [4]	< 326	1-2	~ 800.00

HDF, CDF and netCDF provide a platform-independent library via a high-level API. The stored data can be of any dimensionality and of several forms (numerical, string, images), it can be randomly read or written, unitarily or in blocks. HDF employs a more flexible data model (hierarchical) than netCDF and CDF (multidimensional array) and, according to Li *et al* [31], this flexibility comes at the cost of higher processing loads. The three formats are nearly equal referenced in the literature, they present equivalent features and performance. The choice for one of them should consider compatibility with the target system in hardware, software and development language.

3.5 Parallel File Systems

In Distributed Visualization, it is possible to have tens of machines simultaneously accessing the same tera byte dataset. A single disk device cannot cope with these needs. *Parallel file systems* were designed to deal with this problematic. These systems are designed on a client-server basis with multiple servers running a sort of I/O daemon. The parallel file system strips the files and store them across these servers. To retrieve information, the system reassembles a desired file and transmits it to the client. The whole process occurs automatically via calls to a user level library. Other functionalities like permission checking for file creation, open, close and removal are supported by an auxiliary manager process that handles meta data during the system operation.

Parallel file systems have to be robust and scalable, conform to existing I/O APIs for backward compatibility, maintain addressing file semantics, provide transaction support and be easy to use and install. Among the most popular implementations of parallel file systems for commodity PCs are the Lustre system [28], from Cluster File Systems Inc., released as open-source software, and the Parallel Virtual File System (PVFS) [14], also open-source, both for the Linux platform. The later is in its second release, which presents a number of improved features and a new design.

Real parallel file systems are very complex. Maybe that is the reason why there is just a few implementations available for PC clusters. Comparing the options is not simple, as their complexity confer them a great number of features that are difficult to benchmark. Margo *et al* [35] perform an extensive analysis of PVFS, Lustre and GPFS, however no categorical conclusions are drawn being up to the analyst to decide which one to choose. With the release of Lustre as open-source and with the emergence of PVFS version 2, these systems tend to evolve providing regular new features.

3.6 Operating System

A report from Silicon Graphics observes [3] that the operating system (OS) is replicated at each machine of a PC cluster leading to costs increase for each new node. License expenses, memory and processing requirements of each operating system instance sum up to a great burden. According to Yeo *et al* [49], besides these factors, the choice for the operating system in a cluster must consider: *manageability*, management and administration of local and remote resources; *stability*, robustness against system failures with system recovery; *performance*, optimized efficiency for OS tasks; *extensibility*, ability to easily integrate cluster-specific extensions; *scalability*, scale without performance degradation; *support*, user and system administrator support; and *heterogeneity*, support to multiple architectures to define a cluster consisting of heterogeneous hardware.

Another consideration is the OS configuring likeness to enable variable configurations and customized optimizations. Choices in the market point to Unix proprietary solutions, to expensive Windows easy-to-set systems and to low-cost flexible (open-source) Linux systems. According to the worldwide top 500 hundred supercomputers list, reported by the Forbes magazine [34], the Linux platform has beaten competitors as the main choice for supercomputing.

3.7 Technologies Summarization

To link much of the information provided so far, in figure 9 we present the I/O structure of a cluster of computers with optimized data access. At the top layer is the *parallel execution* which is responsible for the data processing according to one of the parallelisms described in section 3.1. These algorithms are load-balanced according to the discussion carried out in section 3.2.

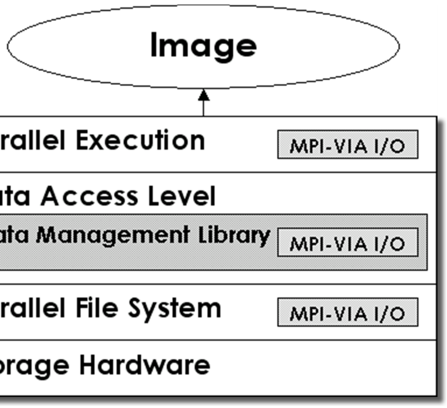

Figure 9: The layers of a Distributed Visualization system. Storage devices are at the lowest layer abstracted by parallel file system access. The MPI standard, along with VIA technology, provides easy multi-point communication for data management libraries, such as HDF and netCDF, that feed the application with the data to be processed. At the highest layer, the parallel execution is performed to produce visualization images.

This model illustrates the state-of-the-art implementation strategy for Distributed Visualization concerning large datasets. The methodologies and technologies to be used at each layer depend on several factors as discussed along the text. Currently, works in the literature deal about finding the better settings for this model and/or to simplify it with more abstracting layers. This last issue is introduced in the next section in the form of distributed parallel rendering libraries.

The layers in figure 9 are assisted by optimized network technologies, as presented in section 3.3. Among these technologies, the MPI standard, combined with VIA technology, is present at each layer of the model by providing simplified optimum access to remote data. To efficiently promote *data access*, it is necessary to use *data management* libraries, like those presented in section 3.4. These libraries access the information at the lowest layer, where lies the *storage hardware* providing voluminous data access. To abstract the storage hardware, *parallel file systems*, like the ones described in section 3.5, provide parallel high-performance transparent access.

4 Distributed parallel rendering libraries

Attempts have been carried out to abstract Distributed Visualization through graphical libraries. The goal is to allow ordinary graphical library calls and have simplified management of distributed processing units as a visualization cluster. Earlier works met this goal but are not based on commodity hardware. Later proposals address commodity PCs.

WireGL
The WireGL library [26] replaces the OpenGL driver to enable OpenGL in Distributed Visualization environments. By preserving the OpenGL API, applications can run on top of WireGL without recompilation and be provided with performance speedups, according to Humphreys *et al* [26]. WireGL supports one or multiple clients simultaneously sending commands and data to one or multiple servers in sort-first parallelism. It intercepts regular OpenGL commands and send them to servers over the network. It also implements a network protocol for geometry communication and performs final image reassembly in software or hardware.

Chromium
Chromium [27] is an advanced derivation from WireGL that similarly overlays OpenGL for compatibility. It supports the use of stream processing units, or SPUs, that perform specific rendering tasks. The SPUs can be chained to achieve a complex rendering execution. Chromium's architecture primes for its general orientation and flexibility. The SPU chain can be configured arbitrarily and both sort-first and sort-last parallelisms have been achieved, according to Humphreys *et al* [27]. The drawback of Chromium's architecture is that its performance is influenced by its stream orientation, which cannot efficiently exploit frame coherence.

OpenSG
OpenSG [46] is a scene graph multi-threaded API, specially designed for Virtual Reality applications. The scene graph metaphor (or hierarchical graphics database) organizes a graphical model as a graph that can manage visual entities hierarchically. By maintaining a copy of the scene graph for each thread, the threading system copes with distributed rendering because various servers can simultaneously respond to interaction (graph changes). To do so, OpenSG bears a client-server setup to replicate data on multi-

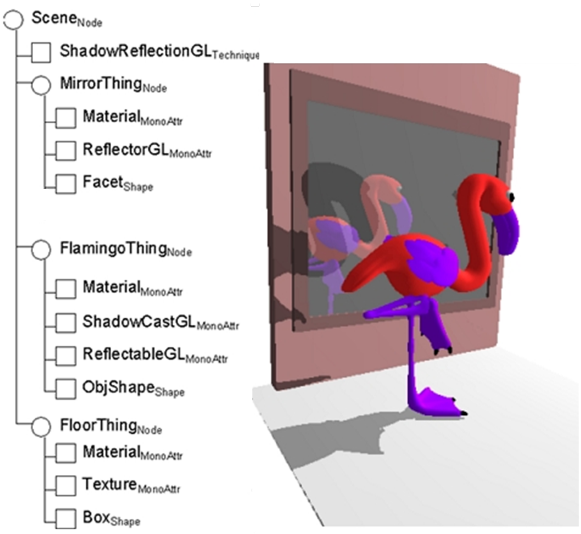

Figure 10: Example of a scenegraph defined with the *Generalized Scene Graph API* [15]. The hierarchical structure of the graph provides eased handling of complex scenes via propagated operations along the paths of the graph. Reproduced with permission granted by Jürgen Döllner and Klaus Hinrichs.

ple machines that receive broadcasts informing of graph changes every frame. The library is flexible and can be used for implementing sort-first, sort-middle and sort-last algorithms. A similar library, also scene graph oriented, is the OpenRM project [11].

The general orientation and high-level style of these libraries cause them to be less scalable than specific optimized implementations. This limitation is clearly demonstrated by Gribble *et al* [22] who ported their Simian project both to a customized cluster implementation and to the top of the Chromium framework for performance comparison. Other issues are flexibility and compatibility.

5 Conclusions

We have surveyed basic concepts on Distributed Visualization. The presented content aims at elucidating what a Distributed Visualization system is, how it is characterized and what issues involve its design and implementation. We provide to beginners an introductory direction both for research and development and, for more experienced readers, we provide an analytical view of such systems. We have focused on distributed parallel rendering architectures, a cluster-based systematization that has popped up in the literature as works that explore flexible commodity low-cost PCs. These implementations have reached great performance levels and scalable architectures that evolve to the pace of market innovations.

Many challenges still have to be bypassed in Distributed Visualization. Although the higher performance of PC clusters, their power is far from workstations as Silicon Graphics's InfiniteReality4 enabled systems, which scales up to 20.6 Gpixel textured antialiased pixels filled per second, or further. Robust real-time rendering for dynamic datasets is also an open challenge. Of-the-shelf Distributed Visualization software to amplify collaborative analytical work has not been accomplished either. We expect that this work can stimulate the quest for these goals by providing a source of information about the Distributed Visualization expertise.

Aknowledgements Thanks to Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Capes).

References

[1] IEEE computer society task force on cluster computing - cluster computing white paper - version 2.0. 2000. 119 pages.

[2] Silicon graphics's white paper 3353 - infinitereality4 graphics - the ultimate in visual realism, 2002.

[3] Silicon graphics's white paper 3661 - choosing a visualization system for your organization, 2004.

[4] Dolphin interconnect solutions inc., 2005. http://www.dolphinics.com/support/; accessed April, 2009.

[5] Infiniband trade association, 2005. http://www.infinibandta.org; accessed April, 2009.

[6] Nasa - cdf home page, 2005. http://cdf.gsfc.nasa.gov; accessed April, 2009.

[7] Ncsa hdf home page, 2005. http://hdf.ncsa.uiuc.edu/; accessed April, 2009.

[8] The quake project, carnegie mellon university and san diego state university, 2005. http://www.cs.cmu.edu/˜quake/; accessed April, 2009.

[9] Top 500 supercomputer sites, top 500 statistics, 2009. http://www.top500.org; accessed April, 2009.

[10] Baker, M., Farrell, P. A., Ong, H., and Scott, S. L. Via communication performance on a gigabit ethernet cluster. *Proceedings of the 7 th International Euro-Par Conference - Lecture Notes in Computer Science, vol. 2150/2001, Springer Verlag*, 2001.

[11] Bethel, E. W. White paper: Sort-first distributed memory parallel visualization and rendering with openrm scene graph and chromium. 2003. R3vis Corporation - 19 pages.

[12] Boden, N., Cohen, D., Felderman, R., Kulawik, A., Seitz, C. L., Seizovic, J., and Su, W. Myrinet: A gigabit-persecond local area network. *IEEE Micro*, 15(1):29–36, February 1995.

[13] Cameron, D. and Regnier, G. *The Virtual Interface Architecture - 1st edition*. Intel Press, 2002. 210 pages.

[14] Carns, P. H., III, W. B. L., Ross, R. B., and Thakur, R. Pvfs: A parallel file system for linux clusters. *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, 2000.

[15] Döllner, J. and Hinrichs, K. A generalized scene graph api. *Vision, Modeling, Visualization 2000 (VMV 2000) - Saarbrücken. Akademische Verlagsgesellschaft.*, pages 247–254, 2000.

[16] Ellsworth, D. *Polygon Rendering for Interactive Visualization on Multicomputers*. Phd thesis, University of North Carolina, 1996.

[17] Ellsworth, D., Good, H., , and Tebbs, B. Distributing display lists on a multicomputer. *Computer Graphics*, 24(2):147–154, 1990.

[18] Farley, D. L. Performance comparison of mainframe, workstations, clusters, and desktop computers. Technical Report NASA/TM-2005-213505, National Aeronautics and Space Administration (NASA), 2005. 26 pages.

[19] Fernando, R. Trends in gpu evolution. *Annual Conference of the European Association for Computer Graphics - Industrial Seminars 2 Session*, 2004. http://eg04.inrialpes.fr/Programme/IndustrialSeminar/PPT/Trends_in_GPU_Evolution.pdf, Accessed April, 2009.

[20] Foley, J. D., Dam, A. v., Feiner, S. K., and Hughes., J. F. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2 edition, 1997.

[21] Garcia, A. and Shen, H.-W. An interleaved parallel volume renderer with pc-cluster. In *Eurographics Workshop on Parallel Graphics and Visualization*, pages 51–59, Blaubeuren, Germany, 2002.

[22] Gribble, C., Cavin, X., Hartner, M., and Hansen, C. Cluster-based interactive volume rendering with simian. Technical Report TR UUCS-03-017, University of Utah, 2003. 7 pages.

[23] Gropp, W., Lusk, E., Doss, N., and Skjellum, A. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[24] Gropp, W., Lusk, E., and Thakur, R. Using mpi-2: Advanced features of the message passing interface. Technical report, MIT Press, 1999. ISBN 0-262-57132-3, 382 pages.

[25] Haber, R. B. and McNabb, D. A. Visualization idioms: A conceptual model for scientific visualization systems. In Shriver, B., Neilson, G. M., and Rosenblum, L., editors, *Visualization in Scientific Computing*, pages 74–93. IEEE, 1990.

[26] Humphreys, G. and al., P. H. e. Wiregl: A scalable graphics system for clusters. In *Proceeding of SIGGRAPH*, pages 129–140, Los Angeles, CA, USA, 2001.

[27] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. Chromium: A stream-processing framework for interactive rendering on clusters. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 693–702, San Antonio, Texas, USA, 2002.

[28] Inc., C. F. S. White paper - lustre: A scalable, high-performance file system. 2002. 13 pages.

[29] Kim, J.-S., Kim, K., and Jung, S.-I. Building a high-performance communication layer over virtual interface architecture on linux clusters. In *ICS '01: Proceedings of the 15th international conference on Supercomputing*, pages 335–347. ACM Press, 2001.

[30] Kurc, T. M., Aykanat, C., and Ozguc, B. A comparison of spatial subdivision algorithms for sort-first rendering. *Lecture Notes in Computer Science*, 1225:137–146, 1997.

[31] Li, J., Liao, W.-K., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. Parallel netcdf: A high-performance scientific i/o interface. In *Proceedings of Supercomputing 2003 Conference*, 2003.

[32] Liu, J., Chandrasekaran, B., Wu, J., Jiang, W., Kini, S., Yu, W., and Buntinas, D. Performance comparison of mpi implementations over infiniband, myrinet and quadrics. In *Proceedings of the ACM/IEEE SC2003 Conference*, Phoenix, Arizona, USA, 2003. 58 pages.

[33] Lombeyda, S., Moll, L., Shand, M., Breen, D., and Heirich, A. Scalable interactive volume rendering using off-the-shelf components. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 115–121, San Diego, CA, USA, 2001.

[34] Lyons, D. Linux rules supercomputers. *Forbes Magazine*, 2005. http://www.forbes.com/2005/03/15/cz_dl_0315linux_print.html; accessed April, 2009.

[35] Margo, M. W., Kovatch, P. A., Andrews, P., and Banister, B. An analysis of state-of-the-art parallel file systems for linux. Technical report, San Diego Supercomputer Center - University of California, 2004. 28 pages.

[36] Molnar, S., Cox, M., Ellsworth, D., and Fuchs, H. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, 14(4):23–32, July 1994.

[37] Montrym, J., Baum, D., Dignam, D., and Migdal, C. Infinitereality: A real-time graphics system. In *Proceedings of SIGGRAPH*, pages 293–302, 1997.

[38] Morel, K., Wylie, B., and Pavlakos, C. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *Proc. IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 85–92, San Diego, CA, USA, 2001.

[39] Mueller, C. A. *The Sort-First Architecture for Real-Time Image Generation*. Phd thesis, The University of North Carolina at Chapel Hill, 2000.

[40] Networkworld. Technology to tie together servers swells, 2004. http://www.networkworld.com/news/2004/0412specialfocus.html?page=1; accessed April, 2009.

[41] Petrini, F., Feng, W., Hoisie, A., Coll, S., and Frachtenberg, E. The quadrics network: High-performance clustering technology. *IEEE Micro*, 22(1):46–57, 2002.

[42] Roble, D. R. A load balanced parallel scanline z-buffer algorithm for the ipsc hypercube. In *Proceedings of Pixim*, pages 177–192, Paris, France, 1988.

[43] Samanta, R., Funkhouser, T., and Li, K. Parallel rendering with k-way replication. In *Proceedings of the IEEE 2001 Symposium on Parallel and Large-data Visualization and Graphics*, pages 75–84, 2001.

[44] Sutherland, I., Sproull, R., and Schumacker, R. A characterization of ten hidden surface algorithms. *ACM Computing Surveys*, 6(1):1–55, March 1974.

[45] Thakur, R., Lusk, E., , and Gropp, W. Users guide for romio: A high-performance, portable mpi-io implementation. Technical Report Technical Memorandum ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory, 2004. 15 pages.

[46] The-OpenSG-Staff. Opensg starter guide. http://www.opensg.org/downloads/OpenSG-1.2.0-UserStarter.pdf; accessed April, 2009.

[47] Touma, C. and Gotsman, C. Triangle mesh compression. *Graphics Interface*, pages 26–34, 1998.

[48] Upson, C. and al, e. The application visualization system: A computational environment for scientific visualization. In *IEEE Computer Graphics and Applications*, volume 9, pages 30–42, 1989.

[49] Yeo, C. S., Buyya, R., Pourreza, H., Eskicioglu, R., Graham, P., and Sommers, F. Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. pages 521–551, 2005.

[50] Zhu, H., Chan, K. Y., Wang, L., Cai, W., and See, S. Dpbp: A sort-first parallel rendering algorithm for distributed rendering environments. In *IEEE International Conference on Cyberworlds*, pages 214–220, Marina Mandarin Hotel, Singapore, 2003.