Volume 24 (2005), number 2 pp. 217–245 COMPUTER GRAPHICS *forum*

Visual Supercomputing: Technologies, Applications and Challenges

Ken Brodlie[2], John Brooke[3], Min Chen[4], David Chisnall[4], Ade Fewings[1], Chris Hughes[1], Nigel W. John[1], Mark W. Jones[4], Mark Riding[3] and Nicolas Roard[4]

[1]School of Informatics, University of Wales Bangor, UK
[2]School of Computing, University of Leeds, UK
[3]Manchester Computing, University of Manchester, UK
[4]Department of Computer Science, University of Wales Swansea, UK
contact email: m.chen@swansea.ac.uk

Abstract

If we were to have a Grid infrastructure for visualization, what technologies would be needed to build such an infrastructure, what kind of applications would benefit from it, and what challenges are we facing in order to accomplish this goal? In this survey paper, we make use of the term 'visual supercomputing' to encapsulate a subject domain concerning the infrastructural technology for visualization. We consider a broad range of scientific and technological advances in computer graphics and visualization, which are relevant to visual supercomputing. We identify the state-of-the-art technologies that have prepared us for building such an infrastructure. We examine a collection of applications that would benefit enormously from such an infrastructure, and discuss their technical requirements. We propose a set of challenges that may guide our strategic efforts in the coming years.

Keywords: visual supercomputing, visualization, autonomic computing, grid computing, distributed processing, mission critical visualization, computational steering.
ACM CCS: I.3.2 [Computer Graphics]; Distributed/network graphics; I.3.8 [Computer Graphics]: Applications; I.3.m [Computer Graphics]: Visualization

1. Introduction

Today there are a variety of computational resources available to visualization. While a huge number of users are content with the visualization capabilities provided through modern desktop computers and powerful 3D graphics accelerators, many are still relying on high-performance computing facilities to visualize very large datasets or to achieve real-time performance in rendering a complex visualization. In some areas, users have already demanded visualization capabilities to be provided through mobile computing systems, such as PDAs (Personal Digital Assistants), most of which are yet to benefit from powerful 3D graphics accelerators. As the size of visualization data (e.g. in visual data mining), the complexity of visualization algorithms (e.g. with volumetric scene graphs), and demand for instant availability of visualization (e.g. for virtual environments) continues to grow, it is unlikely that visualization users can be served adequately, at least in the coming years, by an infrastructure largely based on desktop computers.

Inevitably, this leads to a series of questions that we must ask ourselves:

- What would be an adequate infrastructure that is built upon modern computing and communication technologies and is designed to support visualization users?
- In what way do the computational requirements of visualization differ from other software technologies?
- Is it desirable or feasible to bring a range of technologies under one management (not necessarily under one roof)?
- If it were feasible to build such an infrastructure, what would be an appropriate virtual machine interface for the infrastructure?

Submitted June 2004
Accepted April 2005
Received May 2005

- How should users' experience be managed when they access visualization resources in the infrastructure?

In fact, the computer graphics and visualization community has been seeking answers for these questions for the past few decades. The community has invested a huge amount of effort in developing specialized graphics hardware, has always been among the first to deploy the latest technologies for high-performance computing, and has accumulated large volumes of research outputs in parallel, distributed, and web-based techniques for visualization. Recently, the community has shown equally great enthusiasm to embrace the cluster, Grid and mobile technologies. However, in general, the community has tended to address these questions mainly from the perspective of visualization technology. With the rapid expansion of the visualization user community, there is an urgent need to examine these questions from the perspective of end-users, for instance, surgeons, field workers, network managers and fraud detectives.

The authors of this survey are engaged in a collaborative project, e-Viz [1], to develop a software infrastructure for managing a variety of visualization tasks. In this survey, we trace the historic route of deploying advanced computing technologies for visualization, and survey a broad collection of scientific and technological developments, including theories, algorithms, hardware, software and services, for visualization. We utilize the term *Visual Supercomputing* to encapsulate a subject domain concerning such an infrastructure for visualization, and outline the user requirements by considering a range of applications. We present an overview of the state of the art of technologies in hardware and software for visualization, and the impacts of the Internet, Grid and mobile technologies on visualization. We highlight those latest developments that are relevant, or potentially relevant, to visualization. We propose a set of technical challenges in realizing a visual supercomputing infrastructure that manages visualization tasks in complex networked computing environments, as well as managing users' experience in accessing and interacting with visualization resources. We believe that *autonomic computing* can play an integral role in the evolutionary development of such an infrastructure.

Our survey comes at a timely moment in considering the relationship between visual supercomputing and *Grid computing*. There is now a growing body of experience in adapting applications to a Grid environment. What is emerging is a consensus that the original idea of a computational Grid that behaved like a utility Grid for computation is perhaps oversimplified. There may be several different structures for Grids depending on whether the resources aggregated in the Grid are to serve large-scale computation, large-scale data handling, complex data sources (e.g. bioinformatics databases) or perhaps to integrate business processes. In this, the visual supercomputing paradigm presents novel challenges to the Grid concept. A number of pioneering projects, described in this survey, have been testing the implications of a Grid for various visualization applications and have raised many technical issues including real-time processing, synchronicity of resource allocation and interactivity between clients and Grid services.

This survey paper is organized as follows. In Section 2, we give a more precise definition of the term *Visual Supercomputing* and outline its technical scope. In Section 3, we review major scientific and technological developments by following the arrivals of different computing technologies, and identify the state-of-the-art technologies that have prepared us for building an infrastructure for visual supercomputing. In Section 4, we examine a collection of applications that would benefit enormously from such an infrastructure, and discuss their technical requirements. In Section 5, we propose a set of challenges that may be used to guide our strategic efforts in the coming years. These are followed by a summary of our conclusions in Section 6.

2. Visual Supercomputing

In this section, we first define the term 'Visual Supercomputing'. We examine its relevance to the three semantic contexts of visualization. We then outline the technical scope of visual supercomputing from the perspectives of *applications*, *users* and *systems* respectively.

2.1. Definition

Definition. *Visual supercomputing is concerned with the infrastructural technology for supporting visual and interactive computing in general, and visualization in particular, in complex networked computing environments.*

In this survey, we are focusing only on the subject domain of visualization, though most of the discussions can be extrapolated to other subject domains involving visual and interactive computing, such as computer-aided design, computer animation, and computer vision.

As an *infrastructural technology*, visual supercomputing encompasses a large collection of hardware technologies and software systems for supporting the computation and management of visualization tasks. It focuses on generic technologies for managing the specification, execution and delivery of visualization tasks. It addresses issues such as the scheduling of visualization tasks, hardware and software configurations, parallel and distributed computation, data distribution, communication between different visualization tasks, and communication between visualization tasks and their couplings such as computation tasks or data collection tasks. In addition, it provides infrastructural support for users' interaction with visualization systems, and manages users' experience in accessing and interacting with visualization resources. Nevertheless, visual supercomputing does not concern a specific hardware, algorithm, technique and software for processing a specific type of data in order to generate visualization results.

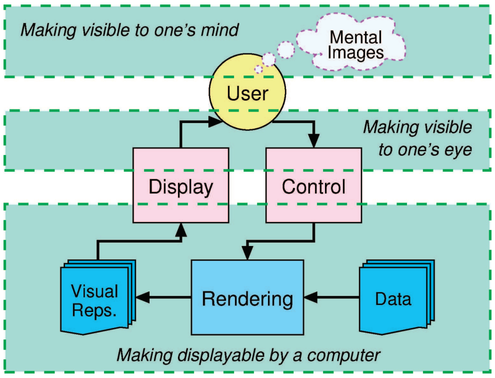

Making visible to one's mind
Mental Images
User
Making visible to one's eye
Display
Control
Visual Reps.
Rendering
Data
Making displayable by a computer

Figure 1: *Three semantic contexts of visualization.*

We put an explicit emphasis on *complex networked computing environments*, as this paper is intended not only as a survey of the technologies that have been developed so far, but also as a report on technologies that are in place as well as those that are desirable for a future infrastructure. No doubt such an infrastructure must take *web computing*, *Grid computing* and *mobile computing* into account. Hence, it has to provide comprehensive support for visualization tasks in complex networked computing environments.

The best way to capture our imagination of a visual supercomputing environment is to consider a global Grid infrastructure for visualization. The above-mentioned technical features of visual supercomputing have clearly set it apart from the traditional subject domains such as *hardware architectures for visualization*, *parallel and distributed computation for visualization*, *web-based visualization*, and *collaborative visualization*. While the advances in these traditional subject domains will have significant influence in shaping the infrastructure of visual supercomputing, we need not only to integrate these technical advances together in an environment, but also bring in, and develop new, technologies for significantly improving the quality of services (QoS) of such an infrastructure and users' experience. This will become apparent in Sections 4 and 5.

2.2. Semantic contexts

The gerund 'visualizing' refers to a process that extracts meaningful information from data, and constructs a visual representation of the information. In the field of visualization, this *process* is commonly considered in three different but interrelated semantic contexts as illustrated in Figure 1.

- *Making displayable by a computer.* This is concerned with the algorithmic and computational process of extracting information and rendering a visual representation of the information. In this semantic context, a visual supercomputing infrastructure should address issues such as allocating and scheduling computational resources for visualization tasks, managing data distribution, and providing mechanisms for inter-process, and inter-task communications within an infrastructure.
- *Making visible to one's eyes.* This is concerned with the process of specifying meaningful information, designing appropriate visual representations, and conveying visual representations to viewers. In this semantic context, a visual supercomputing infrastructure should address issues related to the interaction between users and their visualization tasks, which can be conducted in a variety of forms, including interactive virtual environments, Internet-based collaborative environments, mobile visualization environments, and so on.
- *Making visible to one's mind.* This is concerned with users' thought process and cognitive experience of interpreting received information (not necessarily in a visual form) in one's mind and converting the information to knowledge in pictorial representations. In this semantic context, it is neither desirable nor perhaps feasible for a visual supercomputing infrastructure to manage the thought process of a user. However, there may be a need for introducing gradually new capabilities to support the process of *making visible to one's mind.*

2.3. Application perspective

The demands for visualization multiply in every direction with an increasing number of new applications, which result in new, and often conflicting, requirements. For example:

- In some applications (e.g. bioinformatics), the size of datasets to be processed continues to grow, while in others (e.g. mobile visualization), a careful control of data size is absolutely necessary.
- In many applications (e.g. those involving 3D virtual environments), users still have plenty of appetite for photorealistic visualization at an interactive speed, while in others (e.g. visual data mining), schematic visual representations and non-photo-realistically rendered images are often able to convey more information.
- In many applications (e.g. virtual endoscopy), interactive visualization can now be achieved with modern personal computers, hence small integrated systems provide a high degree of independence to users who operate in various practical situations. Meanwhile, other applications (e.g. those centralized around one or more data warehouses) require a substantial amount of computation for visualization to be closely coupled with the source of data. Some applications, which have distributed or dynamic data sources, demand a more complex computational model.

From the perspective of applications, an important requirement for a visual supercomputing infrastructure is *choice*, that is, it has to provide a large collection of platforms, methods, mechanisms and tools to serve different applications, as well as offer each individual application a diverse selection of means to accomplish a visualization task.

In Section 4, we will consider several major applications, which collectively characterize the main requirements for a visual supercomputing infrastructure.

2.4. User perspective

Visualization users are no longer limited to scientists and engineers. At the same time, a visualization process often requires a high degree of domain knowledge about the application concerned. While the diversity of applications demands a visual supercomputing environment to provide a large collection of platforms, methods, mechanisms and tools, users require the service to be tailored to individual needs, and to be delivered in a seamless manner. Many users, especially those less technically oriented, would very much hope for a secretary-like visualization service, where they simply submit the data, give instructions and receive results. Although to get appropriate results may require a few feedback loops, many users certainly do not wish to get involved in choosing hardware, programming parallelism, organizing storage for input and output data, and so on. Furthermore, like a secretary, perhaps a visual supercomputing infrastructure should accumulate knowledge about various entities in the environments, profiling hardware capabilities, software usage, users' preference, etc. and gradually improving its quality of services to individual users.

Recent developments in business computing, such as electronic customer relationship management (e-CRM) [2], have shown that it is possible to provide users with better quality of services with appropriate technologies that are capable of collecting and processing users' experience. The emergence of *autonomic computing* [3] is gathering further momentum in developing self-managed services in a complex infrastructure (see also Sections 3.6.2 and 5). Therefore, a visual supercomputing infrastructure should have the responsibility for managing:

- visualization resources,
- visualization processes,
- source data and resultant data,
- users' interaction and communication,
- users' experience in accomplishing a visualization task.

2.5. System perspective

From the system perspective, a visualization task is a kind of computation task, which exhibits a specific class of characteristics. The infrastructure of visual supercomputing is built upon a range of underlying technologies, including computer hardware, operating systems, programming languages, data warehouses, communications, world wide web, Grid computing, knowledge-based systems, and standardization. It is neither sensible nor feasible for the visualization community to attempt to provide solutions in all these aspects. However, it is necessary for the construction of such an infrastructure to bring in the latest advances in other fields of computing and communications, and moreover, to influence the developments in these fields. In the following section, we thereby examine in detail the major advances and the state of the art in the relevant fields.

3. Technologies of Visual Supercomputing

The technological infrastructure for visualization has heavily depended on high-performance computing environments until recently. In this section, we examine how the advances in computing and communication technologies have shaped, and reshaped, the foundation of visual supercomputing. Obviously, it is not possible to provide a comprehensive coverage for the large number of visualization works that have impacted upon the development of visual supercomputing. We hence focus on the contributions, in connection with each major technological advance, which are particularly relevant to the state of the art of visual supercomputing. For further historic details, readers are encouraged to refer to several excellent surveys [4,5,6,7,8,9,10] and some major publications [11,12,13,14,15,16].

3.1. The era of supercomputers

Elwald and Mass's vector graphics library for Cray-1 [17] represents the earliest efforts for providing visualization capability to support scientific computation on supercomputers. Since then, there has been a huge volume of publications devoted to parallel architectures and algorithms for computer graphics and visualization. While most of these architectures are no longer in existence, and many of these algorithms have difficulties in benefiting from modern hardware, the research in the era of supercomputers has provided us with a collection of abiding concepts, which can still be entrusted to serve modern visual supercomputing environments.[1]

3.1.1. Models of parallel computation

Since the creation of the very first computer, there have been ever-increasing demands for processing power. Although Moore's Law [18], which suggests that processor power doubles every 18 months, has been satisfied for the last 40 years, today's seemingly powerful desktop computers still cannot meet the requirements of many scientists and engineers who

[1]Some of the works described in this section were developed much later than the actual 'era of supercomputers'. As they represent some fundamental concepts and methodologies, we have conveniently placed them in this section.

seek to model, compute and visualize even larger and more complex problems. Hence, there has always been, and will continue to be, a need for parallel computation. There are three paradigms for parallel computation:

- *Functional Parallelism* splits up the process of computation by dividing an algorithm into separate functional sections and distributing these among different processors organized in a graph structure. One particular case is a pipeline where functional sections are connected along a single path from beginning to end. Data is passed from one processor to another to be computed. The parallelism is achieved when different parts of data are processed concurrently by different functional sections on different processors. In many special-purpose graphics hardware systems, including commodity graphics cards, a graphics-rendering pipeline is partially realized using functional parallelism.
- *Data Parallelism* splits up the process of computation by dividing the data amongst the processors, all of which perform more or less the same algorithmic function. The parallelism is achieved when multiple streams of the data are computed in parallel. Some graphics hardware, such as the SGI InfiniteReality, makes use of data parallelism at individual stages of a graphics pipeline. A large collection of parallel visualization algorithms have been designed based on data parallelism.
- *Farm Parallelism*, which is a hybrid approach, splits up the process of computation into 'tasks', each of which is essentially a portion of data coupled with a functional operation to be performed. Typically, the tasks are kept in a queue, and are distributed to a 'worker' processor whenever one becomes available (i.e. idle). Many modern parallel visualization algorithms [19] have employed farm parallelism to optimize processor utilization.

In 1972, Flynn's taxonomy [20] redefined parallel architectures and, whilst it may be a little outdated now, it is still generally appropriate and widely used. Flynn suggested four categories of parallel machines, namely SISD (*Single Instruction stream, Single Data stream*), SIMD (*Single Instruction stream, Multiple Data stream*), MISD (*Multiple Instruction stream, Single Data stream*), and MIMD (*Multiple Instruction stream, Multiple Data stream*).

In 1978, Fortune proposed the PRAM (*Parallel Random Access Machine*) model [21], which is an idealized parallel machine of p processors sharing an unbounded global memory and a common clock. PRAM architectures are essentially synchronous shared-memory MIMD systems, which are further categorized into four subclasses according to whether a memory location can be read or written concurrently. By not considering synchronization problems and communication issues, the model focuses on the actual parallelization of a problem.

Interconnection of processors and memory is a fundamental factor in classical parallel architectures. Two distinct system architectures are UMA (*Uniform Memory Access*) and NUMA (*Non-Uniform Memory Access*) [22]. UMA systems are better known as SMP (*Symmetric Multi-Processor*) systems, where all processors can access all shared memory in the same, consistent time period. NUMA systems have differing access times for processors depending on the locality of the memory being accessed. Hence, NUMA systems can be much larger and more distributed [23].

There are two principal memory structures that can operate in both UMA and NUMA systems. Firstly, in a *distributed memory* system, each processor has private access to its own fast, local memory, but must use some form of message passing over the interconnection to access the memory of another processor. A typical example is the Cray T3D. Distributed memory systems are generally regarded as difficult to program and debug, but they can scale to many thousands of processors [24]. This is in contrast to *shared memory* systems, where all processors can access all memory directly via a shared bus (normally in a UMA system) or a complex switched interconnection network (normally in a NUMA system). Both of these require synchronization functions in order to safely handle contention for shared data. In hardware specifically designed for shared memory purposes, extra cache memories are often present along with a Cache Coherency Protocol to ensure consistency between local cache and global shared memory [25]. Volume visualization often relies on memory systems supplying conflict-free simultaneous access to multiple voxel values in a volume dataset [26].

Another major consideration in parallel computation is *granularity*, which is often used to indicate, intuitively, the size of parallel tasks in relation to the whole computation requirement. Granularity of a parallel architecture is defined as the ratio of the number of processors to the computation capacity of each processor. Granularity of a parallel algorithm is measured as the ratio of the time required for a basic communication operation to that for a basic computation. Different applications suit fine- or coarse-grain parallelization. Finer granularity brings greater potential for parallelism but increases the overhead of synchronization and communication. In graphics and visualization, researchers have developed a large collection of parallel visualization hardware and algorithms of a wide range of granularity.

3.1.2. Models of inter-process communications

Since the first multitasking systems, it has been necessary to provide means for concurrent processes to communicate. In parallel and distributed systems, inter-process communication introduces delays, which may affect the efficiency of a parallel algorithm significantly.

Shared memory architectures rely on low-latency (<1 ms) communication between processing units and memory via a *dynamic interconnection network* [27]. Several parallel architectures, such as Cray Y-MP, utilized a *crossbar switching* network to connect p processors to q memory banks. A simple but less scalable alternative is a *bus-based* network, in which p processors connect to q memory banks by sharing a common data path. Multistage interconnection networks are a class of networks that offer more scalable performance than bus networks and more scalable costs than crossbar networks. A typical configuration is the *omega network* where p processors connect to p memory banks via p stages, and each stage is an interconnection pattern connecting p inputs to p outputs.

Distributed memory architectures typically involve *static interconnection networks*, which may be of a variety connection topologies [28]. In such architectures, some kind of mechanism for *message passing* [29] or remote procedure calls (RPC) [30] is required. The former enables data communication between remote processes, and the latter facilitates server-client communication by allowing a client to activate pre-defined remote procedures at a server and exchange data in a manner similar to conventional subroutine calls. An object-oriented approach to inter-process communication enables a process to send data as well as operations to remote processes, hence significantly improve the flexibility and scalability in dynamic management of parallel computation tasks. *Common Object Request Broker Architecture* (CORBA) [31] provides such an inter-process communication in UNIX-like systems, while Microsoft Windows incorporates such features into DCOM as an operating system service. Some systems, such as Globe [32] allow a single object to be distributed across a wide area network. Recently, Bernholdt *et al.* [33] adopted a component-based approach for building parallel applications in scientific computation.

A number of modern parallel environments provide programmers with high-level programming interfaces for managing inter-process communications. These include coordination-based middleware such as Jini [34], and document-based middleware such as Globus [35]. This enables application developers to focus on the contents of the communications, and many have adopted the XML standard for defining the syntax of the contents [36]. Although there is significant overhead in parsing transferred data when compared with binary encodings, the XML standard facilitates integration of different protocols and extension of existing protocols.

3.1.3. Performance metrics for parallel systems

Many different metrics have been used to measure the performance of parallel systems and algorithms. The primary objective of using p processors in parallel to solve a problem of size n is the multiplication of the amount of processing power, commonly measured in terms of MIPS (*millions of instructions per second*) or FLOPS (*floating-point operations per second*). However, as previously outlined, it is not possible to parallelize all problems perfectly without introducing additional costs.

One widely used performance metric is the *speedup* [37], which measures the ratio of the time taken by the fastest-known sequential algorithm to that by a given parallel algorithm executed on p processors. In theory speedup can never exceed the number of processors p, though in practice *super-linear speedup* (speedup $>p$) may sometimes be observed due to the effects of a particular system architecture.

It is also important to measure a parallel system with the *efficiency* metric, which is defined as the ratio of speedup to the number of processors; and the *cost* metric, which is the product of parallel run time and the number of processors used. One design goal for a parallel algorithm is to achieve a *cost-optimal* system, the *cost* of which is proportional to the execution time of the fastest-known sequential algorithm. The main obstacle to achieving a *cost optimal* parallel system is the *overhead* resulting from parallelization, which is usually caused by inter-process communication, extra computation (e.g. initialization, distributed data management), and idle waiting (e.g. load imbalance, task synchronization).

Increasing the number of processors reduces efficiency, while increasing the size of the computation increases total speedup hence efficiency. One of the most important metrics is *scalability* [38], which measures the capability of a parallel system to maintain efficiency by increasing problem size and speedup in proportion to the number of processors.

Time-constrained scalability [39] is the core issue in some applications, such as weather forecasting, where it is necessary to fix the parallel run time, and to scale the problem size according to the number of available processors. They also examined the *memory-constrained scalability*, focusing on the largest problem that can fit the available memory in a parallel system.

3.1.4. Parallel programming paradigms

It is generally accepted that there are three primary programming paradigms for developing parallel applications, namely *message passing*, *shared-address-space* and *data parallel* paradigms. The first two are sometimes collectively referred to as the *control-parallel paradigm*.[2]

Message passing is a widely adopted programming paradigm. Although it is commonly associated with MIMD computers, it is universal enough to run on SIMD systems and uniprocessor systems as well as cluster systems and

[2]The terms *control parallelism*, *functional parallelism* and *task parallelism* are often used in an interchangeable manner, while each places different emphasis on aspects of parallel computation.

symmetric multiprocessor systems. It requires the programmer to 'manually' specify subtasks to be executed in parallel, start and stop their execution, and coordinate their interaction and synchronization.

Message Passing Interface (MPI) [29] is perhaps one of the most popular programming environments for developing parallel applications. It is hardware independent, and provides a set of library interface standards for managing process creation and message communications. The MPI-2 standard introduces dynamic process management, though only a few implementations are MPI-2 compliant at present.

The *Parallel Virtual Machine* (PVM) is another implementation of the message-passing paradigm. Using the notion of a virtual machine, PVM enables programmers to treat a set of heterogeneous computers as a single parallel computer. Although MPI is believed to be faster within a large multiple processor system, PVM still scores highly due to its fault tolerance and recovery [40]. Other vendor independent libraries for the messaging passing paradigm include EXPRESS, P4 and PICL.

The shared-address-space paradigm aims to provide programmers with a virtual shared memory machine, which can be built upon distributed as well as shared memory architectures. A programming environment for this paradigm normally includes primitives for creating processes and threads, allocating shared variables, managing mutual exclusion and facilitating synchronization. Managing mutual exclusion during concurrent memory accesses is critical to the correctness of parallel programs in this paradigm [41].

Linda is a coordination language, in the form of C and Fortran extensions, for supporting shared-address-space programming. SR is a language that supports both shared-address-space paradigm and messaging passing paradigm. X3H5 is an ANSI standard for shared-address-space programming in the context of single program and multiple data stream (SPMD). OpenMP, supported by many commercial compilers, is an API for shared-memory programming on multiprocessor architectures.

The *data parallel* paradigm provides programmers with a collection of virtual processors. Hence, it facilitates a high-level abstraction in developing parallel applications, hiding the architectural features of the underlying hardware. Data are distributed among virtual processors. It enables programmers to focus on data parallelism within a parallel algorithm. The parallelization of a computation task is usually realized by an appropriate compiler, which must map virtual processors onto physical processors [42].

Many languages were developed for supporting the data parallel paradigm in the late 1980s and early 1990s, including the CM-2 family (i.e. C*, CM-Fortran and *Lisp by Thinking Machine Co.), MP-2, Dataparallel C, DINO, PC++ and High Performance Fortran (HPF) [42].

One important strand of the data parallel paradigm is *dataflow computation* [43], in which operations are executed in an order determined by the data interdependencies and the availability of resources. The execution can be activated by the availability of input data (i.e. data-driven) or by requirements for specific output data (i.e. demand driven). The concept of dataflow computing facilitates a functional specification of a computation task and the permitted freedom as well as constraints in its parallelization.

This concept has also played a more significant role in visualization (see also 3.2). Systems, such as OpenDX, AVS, IRIS Explorer, SCIRun and DDV, are dataflow-based *modular visualization environments*. They provide a network of modules as the specification of a visualization task, which in principle can support dataflow parallelism [44]. As most networks normally define a coarse-grain dataflow, and most available modules cannot handle partial datasets, these environments offer only limited data parallelism under a centralized executive [45]. AVS, IRIS Explorer and OpenDX can all achieve control parallelism with remote modules. SCIRun provides threaded-task and data parallelism on shared-memory multiprocessors. DDV enables a pipelined-based, demand-driven execution that requires the minimum amount of input data to produce the results.

Stream-based computation, inspired by some parallel hardware architectures, represents a combination of simple control parallelism and simple data parallelism. Chromium [46] provides a collection of pluggable *stream processing units*, and allows streams of OpenGL commands (which contain mostly data) to be processed in parallel. Moreland and Thompson [47] recently described a new set of VTK parallel rendering components built on the top of Chromium for supporting 'cluster to wall' visualization.

3.1.5. Design Methods for Parallel Visualization

Parallel and distributed computation in visualization is broadly divided into two fundamental categories — *object space* and *image space* [13]. 'Object space parallel' refers to the decomposition of a visualization task by dividing input data into a collection of smaller components, each being processed by a computation node. Algorithms in this category are also known as *sort-last* [48], reflecting the need for sorting graphics primitives generated by different computation nodes at the image composition stage of a graphics pipeline. 'Image space parallel' refers to the decomposition of a visualization task into a collection of sub-tasks, each responsible for a small portion of pixels in the visualization image to be synthesized. Algorithms in this category are also known as *sort-first*, reflecting the need for organizing (or 'sorting') data according to the target sub-images prior to their entering into the graphics pipeline.

There is always a need in any parallel implementations to keep a balance between two, often conflicting, requirements, namely *data locality* and *load balance*. The former helps reduce the communication overhead, whilst the latter attempts to minimize the idle time of the processors involved.

Data partitioning is important for any visualization tasks to be computed on parallel and distributed architectures. It is particularly critical for distributed memory architectures, such as Beowulf clusters, where partitioned data components are distributed to different processing nodes. Data or spatial coherence is often harnessed by partitioning algorithms to ensure data locality while minimizing the amount of data residing on each node [49]. Further consideration includes image and frame coherence [50], and overlapping and exchange of boundary data [51]. In general, sophisticated partitioning methods are largely datatype dependent, though they can sometimes also be architecture dependent.

Data partitioning and distribution schemes may be classified according to division criteria (e.g. image-space [49], object-space [52], or hybrid methods [53]), or organization of data replication, which may be in one of the following three forms:

- *Complete Data Replication*, in which each node holds all data locally. This allows simple parallelization, classically image-space parallelization, through the same sequential algorithm on all nodes and minimizes communication overhead during processing. This technique is effective for processing read-only data (such as many graphics and visualization applications). In practice, it often achieves near linear speedup and facilitates good load balancing. However, it does not always scale well as the cost of initial data distribution is a function of both the size of data and the number of nodes. The demand for large memory in each computation node is often difficult to meet.
- *Block Replication*, in which a dataset is typically partitioned into blocks or slices based on the 'physical' organization of the data. This meets the basic needs of object-space parallelization, and replicates a small proportion of an input dataset on each processor. For example, *a regular block decomposition method* may divide a volume dataset into equally sized regular blocks. As equally sized blocks do not ensure an equal amount of workload in each block, this sometimes leads to difficulties in load balance. An *irregular block decomposition* method is often employed to produce blocks that contain similar workloads.
- *Structured or Hierarchical Partitioning*, in which one or more higher level structures are superimposed upon the raw dataset, facilitating data decomposition based on the 'logical' organization of the data. An *occupancy map* [54] is a simple form of such structures, which employs a binary flag to indicate whether or not a block of data is of any interest to the rendering algorithm. A relatively more complex approach is the *Kd-tree Partitioning* [55], which is used for partitioning k-dimensional space into sub-volumes along planes through the dataset. Another commonly used approach is *Octree subdivision* [56,57], which recursively divides the object-space (or an octant) into eight octants. Such a structure can be used to organize the data according to various attributes, including spatial occupancy and workload [58]. While most structured partitioning takes places in the object-space, many of these methods can also serve image-space parallelization as they can facilitate efficient view-dependent data fetch [59], and combined image and data coherence. Recently, scene graphs were used as a hierarchical structure for managing sort-first, distributed memory parallel visualization [60], and facilitating real-time virtual reality applications [61].

Load balancing is normally addressed by appropriate task assignment methods, which are typically classified by its run-time behaviour. *Static task assignment* [52,62] predetermines the workload of each processor according to the predicated workload of each sub-task and processing power of each computation node. Though it requires the preprocessing of task assignment, it demands less communication overhead and little cost in run-time monitoring and scheduling. It usually facilitates efficient data partition and distribution by taking data coherence into account in task assignment.

Dynamic task assignment (e.g. [53]) maintains a pool of tasks, which are often of small and varying workloads. Whenever a processor is free, it is assigned a new task from the pool. This procedure repeats until the pool is empty. This method is particularly effective in heterogeneous environments (where the available computation capacity of each node is difficult to predict), and image-space parallelization (where the workload of each sub-task is difficult to predict).

Image composition, which transforms parallel streams into a useful output (usually a single image), is often a bottleneck in algorithms, especially sort last algorithms. Many classical implementations use the *direct send* method, in which each processor sends its rendered pixels directly to the processor responsible for image composition. However, this simple method suffers from the problem of link contention with a large communication overhead. Lee *et al.* [63] suggested a parallel compositing algorithm to avoid link contention by routing messages along pre-defined grid paths in a mesh network. Ma *et al.* proposed to organize message paths in the form of a binary-tree (also by [52]), together with a binary swap algorithm for improving processor utilization. Recently, Stompel *et al.* [64] presented a scheduled linear image-compositing algorithm, as a highly optimized direct send method, offering better scaling on larger numbers of processors.

3.2. The arrival of graphics workstations and modular visualization environments

The arrival of graphics workstations in the late 1970s changed the face of visual computing. Up to that point, graphics was a speciality, provided in the form of a graphics terminal connected over a relatively slow communication line to a time-sharing processor. Suddenly the processor was co-located with the display, and so interaction became much more dynamic. Moreover, this development coincided with the emergence of network-based windowing systems. This was significant to visualization users, who benefited from not only the WIMP-based user interface, but also from the interactive graphics capability that allowed visualization tasks to be carried out on the desktop.

However it took some time before visualization software emerged to support these new opportunities. In the late 1980s, the performance of workstations reached a point where interactive 3D visualization was feasible, and this performance leap was accompanied by new algorithmic developments such as Marching Cubes for isosurfacing [65], and ray casting for volume rendering [66]. A number of products started to appear, first AVS and aPE, and soon followed by IRIS Explorer, Khoros and IBM Visualization Data Explorer, and more recently TGS Amira. These are known as *modular visualization environments*, since applications are composed by wiring modules together in a dataflow network, using a visual programming paradigm. They are designed to suit end-users with limited programming knowledge and enable them to interrogate interactively a dataset via its visual representation.

Often these modular visualization environments were developed in the first instance as software tools to accompany and promote particular graphics workstations. Thus, AVS was developed as a tool for use with Ardent workstations, and later Stardent; IRIS Explorer was developed to enhance the promotion of SGI workstations. The cost was typically very low, if not free. It is interesting that the software in most cases has lived rather longer than the hardware it was designed to support. For instance, responsibility for the development of AVS and IRIS Explorer was passed to software vendors in the 1990s, NAG Ltd. in the case of IRIS Explorer. Khoros was recently renamed as VisiQuest and marketed by AccuSoft. IBM Visualization Data Explorer became OpenDX as IBM decided to make it an open source product. AVS, IRIS Explorer, Khoros and OpenDX remain vibrant products today.

In the late 1990s, relatively expensive graphics workstations were gradually replaced with modern personal computers equipped with commodity graphics cards. This has certainly created new demands for visualization tools from users in all types of occupations, for instance, security officers, and stock-brokers. It has also introduced a new dilemma as to the best way to provide users with visualization capabilities, and the role of modern personal computers equipped with powerful graphics hardware in the infrastructure of visual supercomputing. Undeniably, it is a formidable argument that a future visual supercomputing infrastructure should be based on all these personal computers, either loosely or tightly connected.

3.3. From special-purpose hardware to general purpose hardware

Many graphics and visualization tasks are computationally intensive, and continuing efforts have been made to offload the tasks performed by different parts of a graphics pipeline onto special-purpose hardware. These efforts are exemplified by several often quoted developments, which include:

- *The video random-access memory* (VRAM) [67], which provides an effective solution to improve the size and access of the frame-buffer required by almost every graphics pipeline.
- *Graphics processors*, such as Intel's i860, which led to an era when graphics processing units (GPUs) facilitated firstly window-based user interfaces to the desktop computers, followed by computer games, interactive 3D graphics, and interactive visualization toolkits.
- *Multiprocessor graphics architectures*, such as Silicon Graphics' POWER IRIS, which distributed the computational costs to a number of subsystems, each serving a set of special-purpose operations, such as geometric manipulation, scan-conversion, and visibility determination.
- *Texture mapping hardware*, which has provided computer graphics and visualization with low-cost pseudo-photorealism. In addition, such hardware has played a significant role in the development of visualization algorithms, and has been effectively deployed to accelerate a range of visualization tasks, including texture-based volume rendering [68,69], flow visualization [70,71], splatting [72] and point-based rendering [73]. Both 2D and 3D texture mapping techniques benefit from hardware support, but only high-performance workstations currently offer 3D texture mapping hardware.

The latest generations of commodity graphics cards, such as the NVidia GeForce and ATI Radeon families, are allowing more and more applications to take advantage of graphics hardware. Demanding visualization techniques such as volume rendering and ray casting have already been successfully implemented [51,74,75,76]. With their generous memory capability and sophisticated numerical processing power, these cards have also been utilized for many circumstances other than graphics and visualization. Their affordability and extensive availability on almost all desktop computers, allows them to become more general purpose than ever before. There are limitations on what can be achieved today, however. For example, the size of the volume that can be manipulated is limited by the amount of dedicated graphics memory available on the card, and this can easily become a bottleneck when dealing with large datasets. Texture data must be fetched via the *accelerated graphics port* (AGP) from the main memory of the PC, and this prevents interactive performance from being achieved. Sophisticated partitioning of the data can be applied as a pre-processing stage to help overcome this limitation [77]. However, it will be the replacement of the AGP with technology based on the new PCI-express standard that will eventually overcome this bandwidth bottleneck [78].

Among all of the increasingly 'general purpose' cards, one stands out as a piece of truly special-purpose hardware; that is, the TeraRecon VolumePro, which delivers high-quality and real-time volume rendering capability [79] Built upon the results of earlier research [80], the commercial VolumePro card currently available for PCs can deliver up to 30 frames per second for a 512^3 voxel dataset.

While there has been a surge of interest in transferring more computational costs from a visualization algorithm to a commodity graphics card, there has also been effort put into building high-performance architectures that benefit from the collective power of an array of graphics cards. Several recent developments have demonstrated how graphics hardware of a PC cluster can accelerate a graphics and visualization task [51,81], implementing either image-space (sort-first) or object-space (sort-last) parallelism (see Section 3.1.5).

WireGL [82] was the first of a new breed of graphics software specifically designed to make use of such cluster systems, and it delivered general-purpose rendering capabilities through its support of sort-first rendering to tiled displays. The design of WireGL evolved into Chromium [46], which is a stream-oriented framework for processing streams of OpenGL commands on parallel architectures such as clusters. It can support sort-first, sort-last and hybrid parallelization strategies through the use of stream processing units. Integration between Chromium and visualization software such as VTK and OpenRM was recently reported [47,60]. The popularity of cluster computing has already led to a number of open-source software systems (e.g. Visapult [83,84], ParaView [85] and VisIt [86]), and commercial products, including software products such as Mod-viz and hardware products such as the Sun Fire Visual Grid system and IBM DeepView. Recent developments also include HP Sepia, ViSUS [87] and Metabuffer [88].

The latest developments in graphics hardware have suggested a modern approach to the architectural design for visual supercomputing, aiming at gaining the collective power from a large number of CPUs and GPUs simultaneously. No doubt, cluster computing is set to become a formidable technology in a visual supercomputing infrastructure.

Figure 2: *A large-scale, front projected, semi-immersive virtual environment.*

3.4. The drive for virtual reality

Immersive and semi-immersive virtual environments (Figure 2) represent a major technical drive in computer graphics and visualization, and have helped push a range of hardware and software technologies forward. Such a virtual environment enables users to be immersed inside a computer-generated world with a sense of spatial presence and often physical presence. For many visualization applications, virtual environments can provide users with realistic experiences in 'interrogating', 'navigating within', 'feeling' and 'manipulating' data via its visual representation.

3.4.1. Hardware technologies

Although conventional displays and input devices can offer the most basic means for graphical interaction, they do not provide a sense of immersion, which is highly desirable in complex visualization tasks. Such tasks may require the user to have a better spatial awareness, better physical control in direct manipulation, better interaction with other users in the same virtual world, or better association with the real world.

Several techniques were developed to enable users visually immersed in a virtual world with 3D stereoscopic views and volumetric views [15,89,90]. These include:

- *Head-mounted display*: It mounts a visual display in front of each eye. It is limited to one user at a time, and requires some form of cabled connection to the computer, which could be cumbersome.
- *Projection-based display*: It provides stereoscopic views by projecting two different series of images, one for each

eye, and allows several users to share the same visualization at a time [91]. Users typically gain stereoscopic experience using *shutter glasses* (i.e. in *active stereo*), or polarizing glasses (i.e. in *passive stereo*).

- *Autostereoscopic display*: It does not require the user to wear special glasses. One of such techniques is *volumetric display* [92], which allows users to view a 3D dataset directly. *Parallax techniques*, including *hologram*, *parallax barrier display*, *lenticular display* and *holographic stereogram* [93], facilitate stereoscopic vision with motion parallax. A special layer (e.g. for example, a horizontal array of cylindrical lenses in *lenticular display*) is placed in front of interleaved images of a 3D object from different viewing angles. When the viewer moves, a different image is picked up by the display, and the object is perceived to have rotated.

In addition to stereoscopic displays, one growing trend is building very large high-resolution displays, involving, for instance, 63 million pixels [47]. Such a display can create an unusual sensation of *presence*, and involvement, enabling a team of users to interrogate a high fidelity model in its totality.

Techniques are available for users to interact with a virtual world with 3D input devices, some of which facilitate users' experience of physical immersion [94]. These include:

- *3D mouse*: As a low-cost hand-held device, it provides a tracker sensor and a set of buttons. By changing the orientation of the mouse, the user can exert navigation control or apply direct manipulation in a virtual environment.
- *Interactive glove*: It is worn by the user and has transducers sewn into the finger joints, which can be used to tell the computer the physical characteristics of the fingers when they are bent. This allows the computer to identify when an object is being picked up, although the user would have no real sense of holding the object [15].
- *Force feedback devices*: They are able to give the user a feel of physically interacting with virtual objects, and are often referred to as *active haptic devices*. One of the available techniques is the *Phantom-like haptic device* [94], which involves a stylus fixed to a base, and can provide force resistance according to users' input actions and physical attributes defined with the object being manipulated. It can produce realistic feeling of the shape and textures of a solid object and the physical property of a deformable object.

3.4.2. Resourcing a virtual environment

The computational resources required to generate and interact with a virtual environment can be very different depending upon what is being simulated. A single desktop computer, or a cluster, with a £1,000 graphics card can be sufficient However, many high-performance applications are looking to the Grid and parallel computing to provide high-quality graphics and resource-intensive data processing.

One of the most successful implementations of a virtual environment is the CAVE (*Cave Automatic Virtual Environment*) [95]. It provides the illusion of immersion by projecting stereo images on the walls and floor of a room-sized cube. Simply by wearing lightweight stereo glasses, multiple users can enter and walk freely inside the CAVE. A head tracking system continuously adjusts the stereo projection to the current position of the main viewer. The technology of the CAVE and other large-scale visualization environments has developed greatly over the last decade. Reconfigurable environments are providing even greater flexibility today. CAVE has been deployed in numerous visualization applications around the world. Such an immersive virtual environment requires the use of a high-performance computer, for example, a SGI Onyx 3400 with 12 CPUs and 3 graphics pipes for CAVE. A special-purpose software is also needed to manage the virtual environment, such as the open source DIVERSE [96].

A related development to DIVERSE is the *Resource Aware Visualization Environment* (RAVE) [97]. It supports collaborative visualization and scales from immersive platforms, to non-immersive but network-enabled platforms, including PCs and PDAs. RAVE is 'resource-aware' so that the rendering platform and the visual representation will be determined dynamically by factors such as the client capabilities and the network bandwidth.

The Grid is becoming more and more important in visualization, particularly when computational resources required for real time interaction in a virtual environment are not locally available. Also, the popular component-based programming paradigm, which has been adopted by many visualization systems such as VTK, AVS and OpenDX, can make use of Grid resources. This allows different computation steps of a visualization pipeline to be distributed around the globe [98]. In particular, the gViz project [55] has extended IRIS Explorer to work in a Grid computing framework, with authentication to allow remote execution of modules being handled by the Globus toolkit.

3.4.3. Collaborative virtual environments

Collaborative Virtual Environments allow multiple users to interact with each other and objects in a shared virtual environment. The users are usually also represented in the virtual environment by embodying themselves in virtual actors. Many of such environments are distributed systems, providing remote users with a sense of common presence.

Examples of collaborative virtual environments include DIVE [99], MASSIVE [100], VRML-extension [101], COVEN [102], DEVRL [103], and VirtuOsi [104]. Many of them have focused on 3D virtual worlds, while others have attempted to address a wide range of issues related to collaborative virtual environments, such as avatar design, users' awareness, dynamic behaviours, system scalability, human factors and interest management. However, most of these environments were built independently on a project-based infrastructure by assembling different technologies together in an *ad hoc* manner.

Environments, such as those listed above, demand a noticeable amount of computational resources, complex distributed data management, dynamic resource allocation, as well as a variety of graphics support, it is only appropriate for the future development of such environments to be built upon a visual supercomputing infrastructure, which can facilitate computation, communication, graphics, data management, interaction management and interest management in a consistent and coherent manner.

3.4.4. Augmented reality

Augmented Reality (AR) is an extension of the traditional virtual environment technology. Instead of immersing a user inside a virtual world completely, AR allows the user to see the real world, whilst supplementing it with virtual objects superimposed within the real world [105].

Most AR technologies have been based upon the use of some form of transparent display, which is positioned between the real world and the eyes of the user [106]. The most basic method is by overlaying computer graphics onto a 2D tabletop surface. In order to align the computer graphics with the physical reality, cameras are used to track the movements of the user's vision and allow the graphics to be realigned [107]. Rekimoto *et al.* [108] developed an InfoTable, which combined a set of cameras for identifying real objects and an LCD projector for adding useful information to the known objects. The development of a collaborative AR environment has also been reported, in which several users can be tracked and see the same virtual objects from different perspectives [109].

Potential AR applications include medical visualization, maintenance and repair, annotation, robot path planning, entertainment and aircraft navigation [105]. Several AR techniques have now been shown to add value to the information available to doctors in the medical world. 3D medical datasets of a patient can be rendered in real time and overlaid onto the patient, allowing the doctor virtually to see inside the patient [110]. This technique can also be used for medical training. Some examples of deploying this technology can be found in a recent survey [111].

One approach to facilitating interaction in an AR environment is to use *Tiles* as a reference between the virtual object and the real world [112]. Through a head mounted camera, the computer can identify the uniquely labelled *Tiles* and superimpose other graphics onto each *Tile*. ARToolKit is a software library for building AR applications [113], which has been successfully used in several applications [114]. The *Tile* approach was extended to become a *Personal Interaction Panel* (PIP), which provided a two handed 'pen and pad' interface for AR applications, allowing users to interact with virtual controls overlaid onto the panel [109].

Desktop PCs are continuing to increase in power and the latest range of GPUs are capable of meeting the requirements for many virtual environments. However, as many immersive virtual environments consume a substantial amount of computational resources, particularly when handling very large datasets, there remains a need for a visual supercomputing infrastructure.

3.5. The ever-growing world wide web

The world wide web has made navigating 3D virtual worlds a readily accessible technology, through programming environments such as VRML, X3D and Java3D. It provides a generic framework, under which it is possible to deliver visualization services to every corner of the globe. Interestingly, the web itself is becoming a focal point in information visualization as its complex infrastructure, highly dynamic traffic, and enormous amount of contents present serious challenges to the state-of-the-art visualization technology.

The initial seminal work by Ang *et al.* [115] demonstrated that the Web had a role in visualization. They associated visualization data with a MIME-type and this launched a helper application on the client side when the browser (i.e. Mosaic in 1994) downloaded the data. This data-driven approach has subsequently been rarely used, but it did show that the web can be an infrastructure for carrying out visualization, not just publishing previously created visualizations.

Two distinct approaches have emerged. In the *server-side approach*, the user submits a request from a web page, specifying the data to be visualized and the technique to be used. The request is processed on the server, and result returned as an image or a 3D VRML world. An early example, using CGI-scripting and IRIS Explorer, was developed by Wood *et al.* [116]. Engel *et al.* [117] exploit this for isosurface extraction.

In the alternative, *client-side approach*, Java applets can be used to provide simple visualizations on demand. An early example is by Michaels and Bailey [118]. This client-side approach has not gained wide popularity, perhaps because the security restrictions on Java applets prohibit the processing of local data. Thus, many applets tend to be educational demonstrations rather than real services.

We can expect the server-side approach to be the forerunner of serious attempts at visualization web services. Prototypes are being developed using SOAP/XML (for Java-based

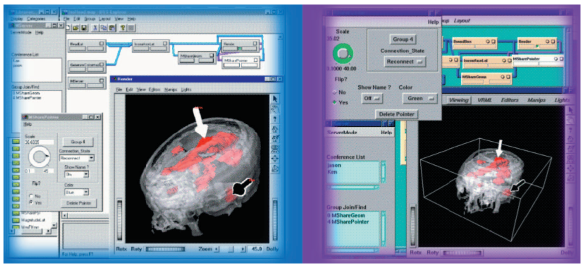


Figure 3: *A collaborative visualization environment, where Dr. Bone is collaborating with Dr. Blood at a remote site to look at CT and SPECT data together.*

services) and gSOAP (for services based on C/C++ and even Fortran). In terms of visual supercomputing, this may offer an attractive approach with its simplicity via a browser interface and power via remote server processing.

3.5.1. Collaborative visualization

The Internet has also encouraged and facilitated collaborative visualization where geographically distributed users can work together as a team. Three distinct approaches have emerged: *display sharing*, where a single application runs, but the interface is shared; *data sharing*, where data is distributed to a group of users to visualize as they wish; and *full collaboration* in which the participants are able to program the way they collaborate.

Display sharing is supported by conferencing technology such as Microsoft NetMeeting, and the non-proprietary VNC [119]. This technology uses efficient compression on the frame buffer so that screen updates can be feasibly transmitted to a group of users. *Data sharing* has been exploited in collaborative environments such as CUMULVS [120] and in pV3 [121], where data from parallel computations is made available to multiple viewers. Another example of data sharing is provided by COVISE [122] where geometry is made accessible to a group, each person in the group being able to render as they please. The most flexible approach is *full collaboration*, epitomized by the COVISA extension of IRIS Explorer developed by Wood *et al.* [123]. In this approach, each collaborator runs its own dataflow pipeline to create a visualization, but can export data and parameter settings to other users, and likewise import data and parameters. Although developed for IRIS Explorer, the idea can be exploited in any dataflow environment.

Figure 3 shows COVISA in action. It demonstrates the sort of application where collaboration can be useful: two doctors (Bone and Blood), each with his own speciality, can collaborate over the network. Bone looks at CT, and Blood at SPECT, but the two modalities can also be combined and this combined visualization viewed by both. Shared pointers allow discussion of significant features. The whole process is supported by video conferencing facilities: either desktop based using for example VRVS, or room-based using the AccessGrid [124].

There are many significant issues in the design of collaborative visualization systems: technical issues such as heterogeneity of visualization systems and of operating systems (collaboration between different visualization systems is hard because of lack of standardized data formats); and social issues such as privacy and floor control.

Both web-based and collaborative visualization have presented a visual supercomputing environment with the requirement for two essential services. As the web is likely to be the dominant information highway in the near future, it is inevitable that a visual supercomputing infrastructure will deliver a substantial amount of its services through the web. Web-based visualization and collaborative visualization will continue to challenge the underlying technologies of a visual supercomputing infrastructure.

3.6. The beginning of grid and autonomic computing

3.6.1. Grid computing

The Grid, as described by Ian Foster, is a distributed computing infrastructure for 'co-ordinated resource sharing' [125]. The Grid is composed of autonomous organizations that maintain various local policies and software for controlling their resources. This distinguishes Grid computing from cluster computing, and introduces a great deal of complexity to the software engineering needed to provide the services and resources. A great deal of experimentation is being carried out to determine the best way to provide middleware services that 'glue' differing underlying systems together. The Grid middleware sits between users' applications and remote computing resources. It is generally accepted that the following key issues and services must be addressed within the Grid middleware:

- networking quality of service (QoS),
- resource co-scheduling,
- load balancing,
- message passing,
- file transfer mechanisms,
- data security, integrity and coherence.
- authentication.

Initially Grid middleware was built as a layer on top of services and protocols common in the Unix world, e.g. `ssh`, `ftp` and `LDAP`. The Globus Project [126] developed a reference implementation of Grid protocols by providing all of the services and capabilities to construct a computational grid. This resulted in a *de facto* standard in the form of the *Globus Meta-computing Toolkit* [35], which contains a range of tools for resource allocation and process management, authentication and related security services, distributed access to structure and state information, monitoring of health and status of system components, and remote access to data via sequential and parallel interfaces.

The Grid infrastructure must be able to support a vast range of applications, allowing its services to be incorporated into the applications using a mix-and-match approach. An important aspect of the Globus Toolkit (GT) is that it separates local and global services. Local services are kept simple to allow deployment, and global services are built on top of local services. The *Metacomputing Directory Service* is provided as part of the toolkit to discover available resources and services. This allows resources to be added and removed dynamically and enables the Grid to recover if a failure was to occur.

Several other middleware developments took place in the same time frame as Globus but based on different principles. UNICORE [127] facilitated seamless access to computing resources and integration of legacy applications. This development was further extended into the European Grid infrastructure via the EUROGRID [128] and GRIP [129] projects. ICENI provided an environment for deploying software components over a federated pool of resources, with rich metadata structures for describing the characteristics of the components. Legion [130] federated computing resources as a virtual supercomputer. Condor [131], which predates the idea of a Grid, is Grid-like in that it locates and federates resources to perform application tasks mainly in numerical computation. Codine, designed for scheduling tasks across a distributed infrastructure, was developed into Sun Grid Engine [132].

Several existing Grid infrastructures have been implemented and are already being used for research. These include the UK e-Science Grid, NASA's Information Power Grid (IPG) and the European Data Grid. In addition, there is the AccessGrid [124], which is not an infrastructure for computation, but an IP-based conferencing infrastructure for supporting large-scale collaborative activities.

In 2002, an alliance was formed between the Globus Project and industrial partners to promote an Open Grid Services Architecture (OGSA) [133]. This changed the delivery of Grid middleware to a *web services framework* and all Grid resources were virtualized as Grid services accessed via web service standard interfaces written in WSDL (*Web Services Description Language*). The emphasis has been shifted from interfacing with resource control mechanisms to hosting environments for the Grid Services defined by OGSA. Thus, Apache Axis and Microsoft .NET became the tooling environments. Globus Toolkit 2 (GT2) was re-factored via tools such as Axis to create Globus Toolkit 3 (GT3), which was intended to preserve the functionality of GT2 in the new Grid Service environment.

In March 2003, a working group of the Global Grid Forum produced the first draft of the specification of Open Services Grid Infrastructure (OGSI), which involved moving key web services standards beyond what has been defined by the relevant standards groups. Because of the growing strain between the business and commerce users of web services and the scientific users of Grid computing, Globus and IBM recently announced that they were moving from OGSI to a newly proposed standard WS-RF (*Web Services Resource Framework*), which exposes resources (as in GT2) but now in the context of web services. The concept of inherent state in the former OGSI Grid Service has been abandoned, as web services are stateless entities and have difficulties in handling stateful management, such as a job queue. The beta release of the Globus Toolkit 4 (GT4), which includes support for WS-RF, has been made available in early 2005. In the meantime, WS-RF implementations are being developed based on .NET (by University of Virginia) and Perl (by University of Manchester) respectively.

The main problem with applying the Grid methodology, and any of the above implementations or proposed standards, to visual supercomputing is the need for interactivity with components running on the Grid. While users' interactive intervention is an integral part of many visualization tasks, it does not always fit naturally with the idea of virtual 'visualization' resources. Some sophisticated middleware components are therefore required. One interesting attempt is the development of an *Interactive Access* plug-in to the UNICORE client [134], which allows end-users to interact, via the UNICORE middleware, with simulation processes running at multiple locations.

The development of the Grid has laid critical foundations for a visual supercomputing infrastructure. It is highly possible that the development of visual supercomputing can be piggybacked on that of the Grid, and can learn a great deal from the evolution of the Grid technology. However, it is also important to recognize that visualization is not just another computation process and hence a visual supercomputing infrastructure is not just a subset of the Grid infrastructure for launching computational tasks.

The great emphasis on a *web services framework* in Grid computing indicates that techniques developed for web-based visualization may have a more generic use in Grid computing. The server-based visualization services (see Section 3.5) allow visualization to deliver in a coarse-grained visualization service, taking input data and sending back images. However, it poses a much more difficult challenge to deliver a sophisticated service, similar to modular visualization environments (see Section 3.2), that could access Grid resources using the web services paradigm.

3.6.2. Autonomic computing

A Grid infrastructure, or more generally, a pervasive infrastructure, will be considerably complex, and the difficulties in managing such an infrastructure raise a serious question as to whether it is adequate for it to be managed by human administrators, and whether it requires a much more system-level automation than what is currently implemented. Researchers and developers in many fields, such as distributed systems, data communications, Internet technology, Grid computing, agent technology, database systems, expert systems and business management systems, are embracing the concept of *autonomic computing* in managing large and complex infrastructures and services.

Autonomic computing [3] refers to computing systems that possess the capability of self-knowing and self-management. Such a system may feature one or more of the following attributes:

- *Self-configuring*—It can integrate new and existing components with little intervention from an administrator.
- *Self-optimizing*—It can continually try alternate configurations to determine if the current one is optimal.
- *Self-healing*—It can detect, and recover from, failure of components, hardware or software.
- *Self-protecting*—It can detect attempts to compromise it, perhaps from hackers or viruses, and react accordingly.

A noticeable amount of research effort in autonomic computing has been placed on the self-management of system infrastructure and business services. Examples of this include self-configuration in patching management [135] and Grid service composition [136], self-optimization in power management [137], business objectives management [138], and network resource management [139], and self-healing in online service management [140] and distributed software systems [141].

Efforts have also been made to broaden the scope of autonomic computing, addressing a wide range of related research issues, such as economic models [142], physiological models [143], interaction law [144], preference specification [145], ontology [146,147], human-computer interaction [148], and so forth.

Though the development of generic software environments for autonomic applications is still in its infancy, several attempts were made, which include projects such as QADPZ [149], AUTONOMIA [150] and Almaden OptimalGrid [151].

QADPZ [149] provides an open source framework for managing heterogeneous distributed computation in a network of desktop computers using autonomic principles. In QADPZ, the system complexity is hidden in the middleware layer, facilitating self-knowledge, self-configuration, self-optimization and self-healing.

AUTONOMIA [150] is a prototype software development environment that provides application developers with tools for specifying and implementing autonomic requirements in network applications and services. It features an *application management editor* for requirements specification, a *mobile agent system* as a uniform execution interface to underlying hardware and operating systems, an *autonomic middleware service* for managing autonomic services, an *application delegated manager* as a broker between components and resources in the context of Jini lookup service [34], and a *fault handler* for self-healing.

OptimalGrid is a self-configuring, self-healing and self-optimizing grid middleware, using a set of distributed whiteboards for communication between the different nodes. A computational problem is expressed using Original Problem Cells (OPCs), which describe the connectivity of the cells with their neighbours and the calculations to be performed using the neighbours' information. OPCs are aggregated in collections, which are themselves part of Variable Problem Partitions (VPP), assigned to grid nodes. The OptimalGrid system is then able to self-configure, using a list of available compute nodes with their characteristics, and can optimize the repartition of OPCs after each computation cycle. As the communication history between nodes is saved in the whiteboards, if a node is lost the system is able to recover and catch up with the computation, rather than restarting the entire problem. The use of these different autonomic features permits to deliver a grid system more robust and easier to use. Future plans include integrating support for the Open Grid Services Architecture (OGSA) [133].

By mimicking the behaviour of the human autonomic system especially in dealing with homoeostasis, autonomic computing is believed to be a solution to the increasing administrative complexity of computing infrastructures. Hence, no visual supercomputing infrastructure can afford to ignore this emerging technology.

4. Applications of Visual Supercomputing

If we were to have a Grid for visualization, what kind of applications would benefit from it, and perhaps more importantly, how would these applications necessitate specific requirements for such an infrastructure? Shalf and Bethel recently outlined a futuristic scenario depicting how a geophysics researcher and her international collaborators may benefit from grid-based computation and visualization. They concluded that the current state of visualization is not grid ready [98]. In this section, we examine several traditional and newly emerged application areas, and discuss their requirements, especially those difficult to be met by the state-of-the-art visualization environments.

4.1. Visual data mining and large-scale data visualization

Never before in history have we had such capability for generating, collecting and storing digital data. Data repositories

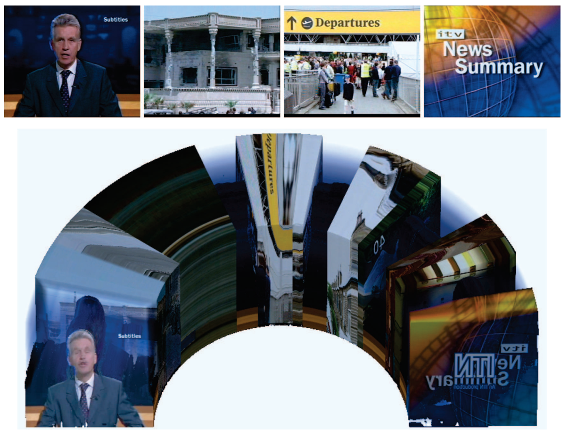


Figure 4: *Video visualization needs to deal with data streams of an arbitrarily large size. Stream-based rendering can be effectively deployed to visualize video streams.*

at terabyte level are becoming commonplace in many applications, including bioinformatics, medicine, remote sensing and nano-technology. In some applications, such as network traffic visualization [152] and video visualization [153] (Figure 4), we are encountering the scenario that dynamic data streams are almost temporally unbounded. Many visualization tasks are evolving into visual data mining processes [9].

These applications are placing a huge strain on the existing visualization environments, and challenging the state-of-the-art technologies in many ways. They demand a variety of infrastructural supports, such as,

- for providing sufficient run-time storage space to active visualization tasks;
- managing complex data distribution mechanisms for parallel and distributed processing;
- choosing the most efficient algorithm according to the size of the problem;
- facilitating the search through a huge parameter space for the most effective visual representation.

Data management is the very first issue in handling large datasets. Many visualization processes involve datasets that are much too large for the internal memory of a computer, and have to rely on external disk storage, usually under the virtual memory management of an operating system. The external disk access can become a serious bottleneck in terms of rendering speed. *Out-of-core algorithms* (also known as *external memory algorithms*) [7] are designed to solve a variety of batch and interactive computational problems by minimizing disk I/O overhead. Various out-of-core visualization algorithms have been proposed to handle large structured and unstructured 3D datasets, for instance, in the context of (i) isosurface extraction [154,155,156], (ii) terrain rendering [59], (iii) streamline visualization [157], (iv) mesh simplification [158], (v) rendering time-varying volume data [159], (vi) rendering unstructured volumetric grids [160], and (vii) ray tracing [161]. While some algorithms rely little on internal memory (e.g. [155,160]), others utilize preprocessed data structures, such as octree [157] and indexing [159] to optimize disk I/O operations. Kurc *et al.* [162] recently reported their experience in visualizing large volume datasets using *Active Data Repository*, which is composed of a set of modular services and a unified interface for supporting the management of, and mapping between, in-core and out-core data.

There has been a similar amount of effort, if not more, for developing techniques that synthesize a visualization image using less than the full dataset. Two commonly used approaches for determining a subset of data to be visualized are *multiresolution* and *view-dependent* data organization.

Multi-resolution data organization makes use of various hierarchical spatial structures to manage *levels-of-details* (LODs) of a graphical model or scene. Such structures facilitate real-time rendering by allowing an appropriate LOD to be selected according to the requirements of interactivity and the constraints of computational resources. In computer graphics and visualization, there exists a large collection of works based on this approach. For example, octrees and min-max indexing were used for isosurface extraction [154,163,164]. Laur and Hanrahan [165] utilized an octree for progressive refinement in splatting. Wilhelms *et al.* [166] employed a k-D tree for direct rendering irregular and multiple volumetric grids.

Viewdependent data organization makes use of the concepts and algorithms of hidden surface removal, and prioritizes geometrical primitives according to their visibility to the viewer. For example, Livnat and Hansen [167] proposed a view-dependent isosurfacing algorithm. LaMar *et al.* [168] prioritized volume data based on its proximity to the viewer. Other view-dependent works include visible set estimation [169], visibility-based prefetching [77], and view-dependent progressive rendering [170].

While it is necessary to deal with problems arising from very large datasets, it is equally important to improve our capability for managing inter-related datasets in order to generate more meaningful visualization. In computer graphics and computer aided design, *scene graphs*, built upon the concept of constructive solid geometry, have played an indispensable role in combining simple objects into a complex object and bringing many objects together into a scene. It is common for graphics systems to support scene graphs, for instance, in RenderMan, OpenGL, OpenRM, VRML, Java3D, POV-ray and Open Scene Graph. However, support for combinational modelling in visualization systems [60,61] is largely based on surface-based scene graphs, relying on image-space composition. Early research efforts for modelling complex visualizations involving multiple datasets were focused on voxelization [171]. In order to address the problems associated with voxelization [172], such as excessive data size and data degeneration, Chen and Tucker [173] outlined the concept of *constructive volume geometry* for combining volumetric datasets and procedurally defined scalar fields. vlib [174], an open source volume graphics API, offers volumetric scene graphs as its fundamental data structure, and provides a discrete ray tracer for direct rendering *volumetric scene graphs*.

In large-scale data visualization, high-performance rendering techniques, such as massively parallel rendering [175], progressive rendering [165] and stream-based rendering [46], are essential to the process of *making displayable by a computer* (Figure 1). However, facing very large datasets, *making meaningful information visible to one's eyes* is often more critical in visualization. With very large datasets, 'meaningful information' is often featured in a visualization at a sub-pixel level, in a large amount or in four or higher dimensions. This challenges us to develop visualization techniques into tools for visual data mining [9].

A popular approach to the handling of a huge amount of visual information is the use of *focus and context* techniques, which highlight a 'focus' in detail and depict its 'context' with less details to provide an overview. Focus and context techniques such as *fisheye views* [176], *perspective wall* [177], *hyperbolic space* [178] and *rubber sheets* [179], have been deployed extensively in information visualization. This approach has also been employed in scientific visualization, deformation-based volume visualization [180], distortion viewing [181], non-photorealistic rendering [182], magnification lens [183], two-level rendering [184], and digital dissection [185].

Data mining should be closely coupled with visualization [186]. Interactive visualization is an indispensable tool in many data mining activities [187,188]. Interactive visualization of large datasets not only demands sufficient computational resources, but also requires effective interactive techniques for data exploration, view navigation, data segmentation, data filtering, data fusion and direct manipulation [9].

One of the main challenges is *computer-assisted design of visual representations*. Many techniques in information visualization enable automated placement of information in a visualization, for instance, *treemap* [189] and *Sunburst* [190] in hierarchy visualization, *recursive pattern* [191] and *circle segments* [192] in time-series visualization, and *spring models* [193] and *Kohonen networks* [194] for self-organization and self-optimization in the entire information space. In volume visualization, initial attempts have been made to automate the specification of transfer functions. Marks *et al.* [195] proposed a *design galleries* approach to the problem, while Kindlmann and Durkin [196] developed a semi-automatic method for generating transfer functions.

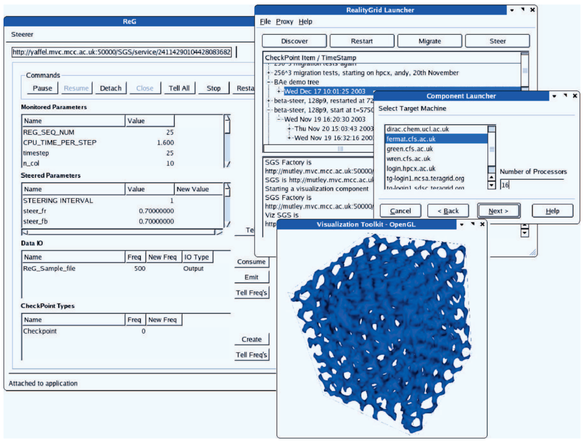
The problems surrounding large-scale data visualization are collectively becoming an infrastructure issue, as it is unlikely an individual technique can provide a satisfactory solution alone. To process a large amount of data at the speed required, it is necessary for a visual supercomputing infrastructure to provide dedicated computational resources and application software systems. It is useful for the infrastructure to select appropriate modelling, processing and rendering techniques according to the available resources and interaction requirements. It is also desirable for the infrastructure to offer a wide range of tools for visual data mining as such activities are often unplanned and the effectiveness of a particular tool cannot always be pre-determined.

4.2. Scientific computation and computational steering

Problem Solving Environments (PSEs) are 'computer systems that provide all of the computational facilities necessary to solve a target class of problems' [197]. For example, Cactus, is an open source PSE, which was originally designed to provide a framework for solving Einstein's Equations, and gradually evolved into a 'unified modular and parallel computational framework for physicists and engineers' [198]. While PSEs have been successfully deployed to model many problems in science, engineering and finance, new problems, including a number of grand challenge problems, continue to be formulated.

In scientific modelling and simulation, it is rare to get a correct model without a complex feedback loop involving specification, modelling, computation, visualization and optimization. Upson et al suggested such a computation cycle [199]. Marshall *et al.* [200] identified three modes of combining simulation and visualization, namely *post-processing*, *tracking* and *steering*.

- In *post-processing*, visualization is merely a post-processing stage of simulation and cannot directly influence (or even abort) the simulation. This asynchronous working requires the simulation to complete before visualization begins, and so there is no opportunity to effect any control on the simulation through the visualization. A benefit however is that the scientists can take as long as they want in visualizing the results, as the time scale for visualization is independent of that for the simulation.
- In *tracking*, the simulation and visualization are coupled, but there is no concept of the user altering the simulation on the basis of the visualization, other than the user hitting the abort key!
- In *steering*, the control parameters of the simulation are exposed, and can be manipulated as it runs. The model was expressed as dataflow. This concept was extended by Brodlie *et al.* [104] to allow an audit trail of check-point information to be stored in a tree structure, called *History Tree*. This generalized steering to facilitate a



http://yaffel.mvc.mcc.ac.uk:50000/SGS/service/24114290104428083682

Monitored Parameters

Name	Value
REG_SEQ_NUM	25
CPU_TIME_PER_STEP	1.600
timestep	25
n_col	10

Steered Parameters

Name	Value	New Value
STEERING INTERVAL	1	
steer_fr	0.70000000	
steer_fb	0.70000000	

Data IO

Name	Freq	New Freq	IO Type
ReG_Sample_file	500		Output

CheckPoint Types

Name	Freq	New Freq
Checkpoint	0	

Attached to application

```
CheckPoint Item / TimeStamp
+-256^3 migration tests, starting on hpcx, andy, 20th November
--BAe demo tree
  +-Wed Dec 17 10:01:25 2003
+-beta-steer, 128p9, restarted at 72
--beta-steer, 128p9, start at t=5750
  --Wed Nov 19 16:20:30 2003
    +-Thu Nov 20 15:03:43 2003
    +-Wed Nov 19 16:32:16 200
```

```
SGS Factory is
http://mutley.mvc.mcc.ac.uk:50000/
SGS is http://mutley.mvc.mcc.ac.uk
Starting a visualization component
SGS Factory is
http://mutley.mvc.mcc.ac.uk:50000/
Viz SGS is
http
```

```
dirac.chem.ucl.ac.uk
fermat.cfs.ac.uk
green.cfs.ac.uk
wren.cfs.ac.uk
login.hpcx.ac.uk
tg-login1.ncsa.teragrid.org
tg-login1.sdsc.teragrid.org
```

Number of Processors: 16

Figure 5: *A computational steering environment developed in the RealityGrid project.*

'reverse gear'. Simulation and visualization were seen as separate processes, linked through a manager. Recently Zhou *et al.* [201] proposed an approach towards automatic steering based on comparative visualization involving both experimental and computational results.

Building a generic computational steering environment is a non-trivial task. A significant development in this area was carried out by van Liere, van Wijk and Mulder [202,203,204]. The key innovation was to build steering widgets, which sat within the visualization, to enable direct manipulation of the simulation.

A major software advance was made with SCIRun, which was a dataflow environment specially designed for steering. It facilitated the interactive construction, debugging and steering of large-scale scientific computations [205]. CUMULVS was developed to provide tools for scientific programmers. It is a software framework for linking steering and visualization services with parallel simulation. It provides two libraries: one for the application, the other for the steering and visualization front-end. It is collaborative in the sense that multiple remote viewers can connect to a simulation. Recently the RealityGrid project (Figure 5) have built some impressive demonstrations of steering Lattice-Boltzmann simulations, which are massive Grid applications, involving collections of machines across the world, and are state of art in what can be achieved on a global scale [206].

On a smaller scale, the gViz e-science project [207] has studied two approaches to computational steering. One extends IRIS Explorer to run in secure distributed fashion across Grid machines, so an IRIS Explorer session spans the internet. The simulation runs inside IRIS Explorer. The other is very similar to RealityGrid in building an API for steering, and decoupling the simulation code from the visualization.

The close-coupling between computation and visualization in computational steering has highlighted the need for advanced inter-process and inter-task management in a visual supercomputing infrastructure. This challenges the underlying technologies of visual supercomputing, requires further advances in fields such as operating systems (e.g. for process management and migration), and programming environments (e.g. for component-based programming, dynamic integration management).

As scientific modelling and simulation usually involves many repetitive steps in a feedback loop, there is a great scope for a visual supercomputing infrastructure to collect performance data in such a feedback loop, and transform the data into knowledge, which can be used to offer users appropriate guidance, identify the best configuration, automate part of the process, and hence provide a higher quality of services. Such an approach has been extensively deployed in business, a conceptually similar situation, where customers are involved in a repetitive process loop, and data measuring various attributes of the process can easily be collected and analysed. There are some successful examples where the quality of services has been improved.

4.3. Mission critical visualization

This category of visualization requires the real time processing of large datasets, possibly from diverse sources, that can then be fed into an interactive visualization environment. Typically, such a system provides decision support tools to the end user. Application areas exist in defence and intelligence, law enforcement, healthcare and social services, scientific research and education, transportation and communication, and energy and the environment. A mature example of mission critical systems are training simulators such as flight simulators, which have used custom built hardware to train pilots for many years both in routine flying and critical incident handling [208].

Medical simulators are expected to be the next major application to benefit from simulator technology, but based on commodity graphics hardware (see Section 3.3). Clinicians are also using intra-operative surgical planning tools and neurosurgeons, for example, have been utilizing image guidance for the last decade [209]. The military is another large market for mission critical visualization. For example, the US Fleet Numerical Meteorology and Oceanography Centre was tasked with supplying military forces deployed in the Persian Gulf with highly accurate meteorological information critical to conducting land, sea and air operations.

A characteristic of mission critical visualization has been the requirement for specialized and often expensive equipment. Until recently, growth has been restricted to niche areas and little work has been published on the optimization and scheduling problems of the visualization task. Grid and cluster based computing, however, are now providing an

Figure 6: *Visualization-guided surgery is a typical application of mission-critical visualization.*

infrastructure for further exploitation and visualization will be a key component of future work.

For example, as shown in Figure 6, in a system for delivering interactive volume interrogation of patient data in the operating theatre [210], visualization tasks were carried out on a server over a mile away from the hospital and then delivered across the data network. Applications such as this raise many issues including: how to guarantee a minimum bandwidth required for both data communication and data processing; the use of redundancy for both communication and computation to ensure a reliable delivery of visualization; and the handling of secure information. Synchronization algorithms as well as data distribution techniques must also be considered when making use of multiple compute resources [211]. Those are exactly the issues to be addressed by a visual supercomputing infrastructure.

4.4. Mobile visualization

Ubiquitous computing is capturing our imagination of a global infrastructure that supports not only networks of desktop computers and high-performance computers, but also a huge number of wearable and mobile computing devices [212]. The prospect for integrating mobile devices into the visualization pipeline and its applications offers new opportunities for accessing, interrogating and manipulating data remotely.

Izadi *et al.* [213] proposed the FUSE system as a development tool for collaborative systems across multiple platforms. Lamberti *et al.* [214] demonstrated a mobile graphical interactive rendering task running on a PDA, which is provided by a remote graphics workstation. Wolf *et al.* [215] proposed the Smart Pointer as a role for PDA devices, where it either presents a subset of the visualization when part of a larger visualization environment (such as a CAVE) or it aims to provide the same overall image as other (desktop) clients, both approaches using a remote visualization server. Hartling *et al.* [216] presented a middleware system, Tweek, which displays a 2D GUI to a virtual environment using a PDA. The user may interact with the virtual environment via the PDA. D'Amora and Bernardini [217] developed a PDA 3D viewer that can access a remote database of CAD models. Apart from the technical aspect, human factor issues in using PDAs for visualization need to be addressed [218].

We categorize the demands upon both the mobile device and the visualization service into the following classes ordered according to their communication requirements:

- *Remote scheduling*: A device, such as a PDA, can be used to monitor the account status of the user on a visualization server. The users should be able to consult their account, see the current state of any job, and perform basic management tasks, such as *start*, *stop*, *hold* and *remove*. This requires a low bandwidth duplex channel for textual communications.
- *Remote monitoring*: Higher level monitoring functions can take advantage of the colour displays on the device. Users may query their account to retrieve still images which are visualizations of their data. They may (pre)select parameters for rendering (such as rendering method and transfer function), and be presented with the image. Such parameters may be used to assist with scheduling decisions. This class requires a duplex channel with a higher bandwidth downstreaming traffic.
- *Remote steering*: A remote user can be notified on job (or intermediate result) completion, and may view a visualization of the result. Some limited interaction with the visual representation is possible as the user's feedback can be used to generate modifications to the current job. This is most useful for checking intermediate results during batch mode without having to be tied down to one location. Some steering of the simulation is possible as jobs can be stopped and restarted from a recent state with new parameters. The bandwidth requirement is higher as the wait time for several images may be undesirable. The computational demands on the PDA are higher due to the need to zoom, pan, and interact with the data. At this stage, transmission and interaction with small 3D models may be desirable and possible.
- *Remote visualization*: The users interact freely with the simulation, using the visualization to explore all aspects of their data. This places a high demand on the PDA as well as the server. The visualization could be in the form of a sequence of images generated by the server and transmitted compressed to the PDA, or the server could send a stream, which could be processed by the limited graphics hardware available on the PDA. User interface widgets could be overlaid over the data, and the user will send interaction data back to the server in order to

Figure 7: *Mobile technology has offered an exciting scope for developing new visualization applications.*

steer the simulation. Some frame loss, and some pauses in results are inconvenient but not critical.

Mobile visualization (Figure 7) introduces an interesting design problem for a visual supercomputing environment. It reminds us of the desktop technology two decades ago when low resolution displays and limited computation resources were supported by mainframe computers. However, it also exhibits a completely new scenario where the requests for visualization or task management, can come from anywhere with often unreliable communication channels in terms of bandwidth and security. The infrastructural support to mobile visualization may significantly broaden the application scope of visualization, and transform this largely laboratory-based technology to a pervasive technology.

5. Challenges in Visual Supercomputing

The above discussions have clearly indicated the need for encompassing a large collection of infrastructural issues related to the management of visualization tasks in a common framework, for which we have introduced the subject domain of *visual supercomputing*. The requirements from applications, such as visual data mining, computational steering, mission-critical visualization and mobile visualization, have indicated a high research priority to the infrastructure of visual supercomputing. While such an infrastructure can benefit from the state-of-the-art technologies in visualization, we are still facing many new challenges in order to realize a well-designed, serviceable and cost-effective infrastructure for visual supercomputing.

Hoare outlined a set of criteria for a grand challenge in computer science [219]. According to these criteria, building a visual supercomputing infrastructure can be considered as a grand challenge in the field of visualization. It raises a series of scientific questions such as:

- *Architectural design*: Would it be desirable or feasible to build an infrastructure for visual supercomputing based on that of the Grid? How would it accommodate the different needs for centralized, distributed or independent services from various applications? How would such an infrastructure provide generic support to the management of visualization data, distributed visual data mining, very large-scale data visualization, mission-critical visualization and mobile visualization?
- *Technology deployment*: Should special-purpose graphics hardware form the central core of a visual supercomputing environment? If so, what would be the relationship between such central hardware and graphics hardware available on personal computers? How would different hardware attributes impact upon visualization algorithms, and how would visualization tasks be managed to take such attributes into account?
- *Quality of service*: How would a visual supercomputing infrastructure provide seamless services to many users and for many applications, instead of just another 'remote login' service? What would be the role of the infrastructure in managing interaction, data and knowledge about users' experience? In what way could users benefit from a knowledge-based infrastructure?

One emerging strategy for developing complex computing infrastructure is *autonomic computing* [3] (see also 3.6.2), which seeks inspiration in self-adaptive biological systems and self-governing social and economic systems.

Adapting the deployment model, proposed by IBM [220], for the gradual evolution of complex system-wide self-managing environments, one can envisage a similar five-level deployment model for visual supercomputing, which can be developed evolutionarily.

- *Level 1: Basic*—At this level, a visual supercomputing infrastructure is an integrated *system* platform that provides visualization applications with necessary computation and communication resources. Typically, users are fully involved in identifying appropriate tools, locating computation resources, and managing data distributions. It is often necessary for users to navigate themselves through complicated technical obstacles, such as networking, security, parallelization, data replication, and so forth.
- *Level 2: Managed*—At this level, a visual supercomputing infrastructure will have a managed *service* layer between the user interface and the system platform. The service layer is aware of the availability and ontology of data and resources, and can provide services to various visualization applications according to dynamic requirements of users and applications as well as dynamic states of the system platform. To a large extent, the development of the Grid technology is aiming at the delivery of a general-purpose infrastructure. To manage visualization applications effectively, it is necessary to incorporate more advanced service features into the Grid technology

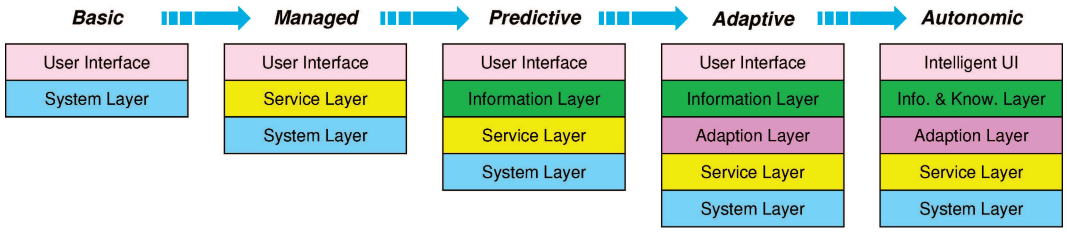


Figure 8: *The deployment model for developing a visual supercomputing infrastructure.*

for supporting a variety of visualization needs such as interactive, distributed, mobile, and mission-critical applications in a more transparent manner.

- *Level 3: Predictive*—At this level, a visual supercomputing infrastructure will have an *information* layer between the user interface and the service layer, which collects, monitors and correlates various user interaction data and system performance data. It provides users with analytical data, which may indicate the quality of visualization results, effectiveness of visualization tools, and so on. In addition, this layer can enable faster and better task specification by reporting potential problems and recommending suitable tools and visual representations. It is at this level, the infrastructure starts to manage users' experience in carrying out visualization tasks.
- *Level 4: Adaptive*—At this level, a visual supercomputing infrastructure will have an *adaptation* layer between the information layer and the service layer. Based on the information collected, the adaptation layer has the functionality for self-configuring and self-optimizing the computational requirements of a visualization task, as well as the functionality for self-managing the system platform and various visualization services dynamically. It is at this level, visualization users can be largely freed from software management, and are able to focus on their core business, that is, visualization.
- *Level 5: Autonomic*—At this level, the traditional user interface in a visual supercomputing infrastructure will be replaced by an intelligent user interface, for instance 'a virtual secretary', which is capable of transforming information to knowledge and provides users with a wide range assistance. Such assistance may include specifying visualization tasks, scheduling inter-dependent jobs, organizing raw data and visualization results, managing security, checking the quality of the service and results, and arranging the sharing of the data with other users.

Figure 8 illustrates evolutionary advance of the infrastructure through the five levels. In this deployment model, each layer is merely a conceptual placeholder for a collection of functional components (e.g. services, tools, agents, databases, knowledge-bases, and so on). It is not necessary for the development and deployment of each level to follow a temporal order. Nor is it desirable to make each layer a centralized bottleneck in the process of visualization. It is most likely that the infrastructure will be realized with a large number of autonomous, interacting, self-governing functional components.

Building a visual supercomputing infrastructure is no doubt an ambitious grand challenge. However, we have already had a solid foothold at Level 1, and are rapidly approaching Level 2. A noticeable amount of research effort is being made to develop system-level autonomic computing techniques in many fields, including distributed systems, data communications, Internet technology, Grid computing, agent technology, database systems and business management systems. Some of such effort can be viewed as 'horizontal' deployment of autonomic computing at the *system layer* and *service layer* of a visual supercomputing infrastructure (Figure 8), while others can provide new concepts, methods and tools for the development of the *intelligent user interface*, *information and knowledge layer* and *adaptation layer*. Hence, we believe that having such a visual supercomputing infrastructure is a realistic challenge.

6. Conclusions

In this survey paper, we have outlined an agenda for *visual supercomputing*, which defines a subject domain concerning the infrastructural technology for visualization. We have considered a broad range of scientific and technological advances in computer graphics and visualization, which are relevant to visual supercomputing. We have identified the state-of-the-art technologies that have prepared us for building such an infrastructure. We have examined a collection of applications that would benefit enormously from such an infrastructure, and discussed their technical requirements. We have proposed a set of challenges that may guide our strategic efforts in the coming years. In particular, we have highlighted the integral role of *autonomic computing* in the gradual evolution of an infrastructure for visual supercomputing.

Acknowledgments

This work is conducted under the framework of the e-Viz project, which is jointly funded by EPSRC ICT programme and e-Science Programme with grants GR/S46567, GR/S46574 and GR/S46581. Special acknowledgements to Chris Hughes whose initial review motivated this survey, Nigel John who suggested making this as a group exercise for the e-Viz project, and Min Chen who coordinated the preparation of this paper. We are thankful to John Sharp, University of Wales Swansea, for his valuable advice on data flow computing.

References

1. e-Viz.www.eviz.org.
2. S. L. Pan and J.-N. Lee. Using e-CRM for a unified view of the customer. *Communications of the ACM* 46(4): 95–99, 2003.
3. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, pp. 41–50, 2003.
4. S. R. Whitman. A survey of parallel algorithms for graphics and visualization. In *Proc. Int. Workshop on High Performance Computing for Computer Graphics and Visualisation*, pp. 3–22, 1996.
5. C. Hansen. Known and potential high performance computing applications in computer graphics and visualization. In *Proc. Int. Workshop on High Performance Computing for Computer Graphics and Visualisation*, pp. 23–29, 1996.
6. D. Bartz and C. Silva. Rendering and visualization in parallel environments.In *Eurographics Tutorials*, 2001.
7. J. S. Vitter. External memory algorithms and data structures: dealing with massive data. *ACM Computer Survey* 33(2): 209–271, 2001.
8. K. Engel and T. Ertl. Interactive high-quality volume rendering with flexible consumer graphics hardware. In *Eurographics State of the Art Reports*, 2002.
9. D. A. Keim, W. Muller and H. Schumann. Visual data mining. In *Eurographics State of the Art Reports*, 2002.
10. K. Brodlie, J. Wood, D. Duce, J. Gallop, J. Walton. Distributed and collaborative visualization. In *Eurographics 2003 State of the Art Report*, 2003.
11. P. M. Dew, R. A. Earnshaw and T. R. Heywood. (Eds.) *Parallel Processing for Computer Vision and Display*. Addison-Wesley, 1989.
12. T. Theogaris. *Algorithms for Parallel Polygon Rendering*. Springer-Verlag, 1989.
13. S. Green. *Parallel Processing for Computer Graphics*. MIT Press, 1991.
14. S. Whitman. *Multiprocessor Methods for Computer Graphics Rendering*. AK Peters, 1992.
15. M. Slater, A. Steed and Y. Chrysanthou. *Computer Graphics and Virtual Environments*. Addison Wesley, 2002.
16. M. Koutek. *Scientific Visualisation in Virtual Reality: Interaction Techniques and Application Development*. Delft University of Technology, 2003.
17. R. Elwald and L. Mass. A high performance graphics system for the Cray-1. *ACM SIGGRAPH Computer Graphics*, pp. 82–86, 1978.
18. G. E. Moore. Cramming more components onto integrated circuits. *Electronics* 38(8): 1965.
19. S. Parker, W. Martin, P.-P. J. Sloan, P. Shirley, B. Smits and C. Hansen. Interactive ray tracing. In *Proc. Symp. Interactive 3D Graphics*, pp. 119–126, 1999.
20. M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Computers* C-21(9): 948–960, 1972.
21. S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. ACM Symp. Theory of Computing*, pp. 114–118, 1978.
22. D. B. Skillicorn. A taxonomy for computer architectures. *IEEE Computer* 21(11): 46–57, 1988.
23. X. Zhang and X. Qin. Performance prediction and evaluation of parallel processing on a NUMA multiprocessor. *IEEE Trans. Software Engineering* 17(10): 1059, 1991.
24. K. Davis, A. Hoisie, G. Johnson, D. J. Kerbyson, M. Lang and S. Pakin, F. Petrini. A performance and scalability analysis of the BlueGene/L architecture. In *Proc. ACM/IEEE Conf. on Supercomputing*, pp. 41–ff, 2004.
25. T. Terasawa, S. Ogura, K. Inoue and H. Amano. A cache coherency protocol for multiprocessor chip. In *Proc. 7th IEEE Int. Conf. on Wafer Scale Integration*, pp. 238–247, 1995.
26. H. Ray, H. Pfister, D. Silver and T. A. Cook. Ray casting architectures for volume visualization. *IEEE Trans. Visualization and Computer Graphics* 5(3): 210–223, 1999.
27. A. S. Tanenbaum. *Modern Operating SystemsM*, 2nd Ed. Prentice Hall, 2001.
28. T. Y. Feng. A survey of interconnection networks. *IEEE Computers*, pp. 12–27, 1981.
29. R. Buyya. (Ed.) *High Performance Cluster Computing—Volume 2: Programming and Applications*. Prentice Hall, 1999.

30. A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Trans. Computer Systems*, 39–59, 1984.

31. S. Baker. *CORBA Distributed Objects*. Addison Wesley, 1997.

32. M. van Steen, P. Homburg and A. S. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency* 7: 70–78, 1999.

33. D. E. Bernholdt, B. A. Allan and R. Armstrong *et al.* A component architecture for high-performance scientific computing. *Int. J High-Performance Computing Applications*. to appear, 2005.

34. S. Oaks and H. Wong. *Jini in a Nutshell*. O'Reilly & Associates, 2000.

35. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. J. Supercomputer Applications* 11(2): 115–128, 1997.

36. K. W. Brodlie, J. Wood, D. Duce, J. R. Gallop, D. Gavaghan, M. Giles, S. Hague, J. Walton, M. Rudgyard, B. Collins, J. Ibbotson and A. Knox. XML for visualization. In *Euroweb*, 2002.

37. P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *IEEE Trans. Parallel and Distributed Systems* 11(6): 589–603, 2000.

38. E. V. Krishnamurthy. *Parallel Processing: Principles and Practice*. Addison-Wesley, 1989.

39. J. L. Gustafson, G. R. Montry and R. E. Benner. Development of parallel methods for a 1024 hypercubes. *SIAM J. Scientific and Statistical Computing* 9(4): 609–638, 1988.

40. G. A. Geist, J. A. Kohla and P. M. Papadopoulos. PVM and MPI: A comparison of features. *Calculateurs Paralleles* 8(2): 137–150, 1996.

41. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam. *PVM Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

42. V. Kumar, A. Grama, A. Gupta and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin, 1993.

43. J. A. Sharp (Ed.): *Data Flow Computing: Theory and Practice*. Ablex Publishing, 1992.

44. D. Song and E. Golin. Fine-grain visualization algorithms in dataflow environments. In *Proc. IEEE Visualization*, pp. 126–133, 1993.

45. J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. C. Law and M. Papka. Large-scale data visualization using parallel data streaming. *IEEE CG&A* 21(4): 34–41, 2001.

46. G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner and J. T. Klosowski. Chormium: a stream-processing framework for interactive rendering on clusters. In *Proc. ACM SIGGRAPH*, 2002.

47. K. Moreland and D. Thompson. From cluster to wall with VTK. In *Proc. IEEE Symp. Parallel and Large-Data Visualization and Graphics*, pp. 25–32, 2003.

48. S. Molnar, C. M, D. Ellsworth and H. Fuchs. A sorting classification of parallel rendering. *IEEE CG&A* 14(4): 23–32, 1994.

49. P. Mackerras and B. Corrie. Exploiting data coherence to improve parallel volume rendering. *IEEE Parallel and Distributed Technology: Systems and Applications* 2(2): 8–16, 1994.

50. S. Goil and S. Ranka. Dynamic load balancing for ray-traced volume rendering on distributed memory machines.In *Proc. Int. Conf. on High Performance Computing*, 1995.

51. S. Muraki, E. B. Lum, K.-L. Ma, M. Ogata and X. Liu. A PC cluster system for simultaneous interactive volumetric modeling and visualization. In *Proc. IEEE Symp. Parallel and Large-Data Visualization and Graphics*, pp. 95–102, 2003.

52. C. M. Wittenbrink and M. Harrington. A scalable MIMD volume rendering algorithm. In *Proc. 8th Int. Parallel Processing Symp.*, pp. 916–922, 1994.

53. P. Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorisation. *IEEE Trans. Visualization and Computer Graphics* 2(3): 218–231, 1996.

54. M. Meissner, M. Doggett, U. Kamus and J. Hirche. Accelerating volume rendering using an on-chip SRAM occupancy map. *In Proc. IEEE Int. Symp. Circuits and Systems*, vol. 2, pp. 757–760, 2001.

55. K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton and J. Wood. Visualization in grid computing environments. In *Proc. IEEE Visualization*, pp. 155–162, 2004.

56. L. J. Doctor and J. G. Torborg. Display techniques for octree-encoded objects. *IEEE CG&A* 3(1): 29–38, 1981.

57. C. H. Chien and J. K. Aggarwal. Volume/surface octrees for the representation of three-dimensional

objects. *Computer Vision, Graphics and Image Processing* 36(1): 100–113, 1986.

58. J. Veenstra and N. Ahuja. Line drawings of octree-represented objects. *ACM Trans. Graphics* 7(1): 61–75, 1988.

59. P. Lindstrom and V. Pascucci. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE Trans. Visualization and Computer Graphics* 8(3): 239–254, 2002.

60. E. W. Bethel, G. Humphreys, B. Paul and J. D. Brederson. Sort-first, distributed memory parallel visualization and rendering. In *Proc. IEEE Symp. Parallel and Large Data Visualization and Graphics*, pp. 41–50, 2003.

61. M. Naef, E. Lamboray, O. Staadt and M. Gross. The blue-c distributed scene graph. In *Proc. IEEE Virtual Reality*, pp. 275–276, 2003.

62. K. Ma, J. S. Painter and M. F. Krogh. Parallel volume rendering using binary swap composition. *IEEE CG&A* 14(4): 59–67, 1994.

63. T.-Y. Lee, C. S. Raghavendra and J. B. Nicholas. Image composition schemes for sort-last polygon rendering on 2D mesh multicomputers. *IEEE Trans. Visualization and Graphics* 2(3): 202–217, 1996.

64. A. Stompel, K.-L. Ma, E. B. Lum, J. Ahrens and J. Patchett. SLIC: Scheduled linear image compositing for parallel volume rendering. In *Proc. IEEE Symp. Parallel and Large-Data Visualization and Graphics*, 2003.

65. W. Lorensen and H. Cline. Marching cubes: a high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* 21(4): 163–169, 1987.

66. M. Levoy. Display of surfaces from volume data. *IEEE CG&A* 8(5): 29–37, 1988.

67. R. Pinkham, M. Novak and K. Guttag. Video RAM excels at fast graphics. *Electronic Design* 31(17): 161–182, 1983.

68. B. Cabral, N. Cam and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proc. ACM/IEEE Symp. Volume Visualization*, pp. 91–98, 1995.

69. R. Westermann and T. Ertl. Efficiently using graphics hardware in volume rendering applications. In *Proc. ACM SIGGRAPH*, pp. 169–177, 1998.

70. C. Rezk-Salama, P. Hastreiter, C. Teitzel and T. Ertl. Interactive exploration of volume line integral convolution based on 3d-texture mapping. In *Proc. Visualization*, pp. 233–240, 1999.

71. A. Telea and J. J. van Wijk. 3D IBFV: hardware-accelerated 3D flow visualization. In *Proc. IEEE Visualization*, pp. 233–240, 2003.

72. J. E. II Swan, K. Mueller, T. Müller, N. Shareef, R. Crawfis and R. Yagel. An anti-aliasing technique for splatting. In *Proc. IEEE Visualization*, pp. 197–206, 1997.

73. H. Pfister, M. Zwicker, J. van Baar and M. Gross. Surfels: surface elements as rendering primitives. In *Proc. ACM SIGGRAPH*, pp. 335–342, 2000.

74. M. Meissner, U. Hoffmann and W. Strasser. Enabling classification and shading for 3d texture mapping based volume rendering using OpenGL and extensions. In *Proc. IEEE Visualizationn*, 1999.

75. C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner and T. Ertl. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage-rasterization. In *Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp. 109–118, 147, 2000.

76. S. Roettger, S. Guthe, D. Weiskopf, T. Ertl and W. Strasser. Smart hardware accelerated volume rendering. In *Proc. EUROGRAPHICS/IEEE-TCVG Symp. Visualization*, 2003.

77. W. T. Corrêa, J. T. Klosowski and C. T. Silva. Visibility-based prefetching for interactive out-of-core rendering. In *Proc. IEEE Symp. Parallel and Large-Data Visualization and Graphics*, pp. 2–8, 2003.

78. A. Wilen, J. P. Schade and R. Thornburg. *Introduction to PCI Express: A Hardware and Software Developer's Guide*. Intel Press, 2003.

79. H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer and L. Seiler. The VolumePro real-time ray-casting system. In *Proc. ACM SIGGRAPH*, pp. 251–260, 1999.

80. H. Pfister and A. Kaufman. Cube-4—a scalable architecture for real-time volume rendering. In *Proc. ACM/IEEE Symp. Volume Rendering*, pp. 47–54, 1996.

81. B. Wylie, C. Pavlakos, V. Lewis and K. Moreland. Scalable rendering on PC clusters. *IEEE CG&A* 21(4): 62–69, 2001.

82. G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett and P. Hanrahan. WireGL: a scalable graphics system for clusters. In *Proc. ACM SIGGRAPH*, 2001.

83. Visapult 2.0. http://www-vis.lbl.gov/Research/visapult2/.

84. W. Bethel and J. Shalf. Consuming network bandwidth with visapult. In *The Visualization Handbook*,C. Hansen, C. Johnson, (Eds.) Academic Press, 2003.

85. ParaView. http://www.paraview.org/.

86. VisIt. http://www.llnl.gov/visit/.

87. VisSUS. http://pascucci.org/visus/.

88. The Metabuffer Project. http://www.ices.utexas.edu/ccv/projects/DiDi/Metabuffer.htm.

89. G. Burdea and P. Coiffet. *Virtual Reality Technology*, 2nd ed. Wiley, 2003.

90. W. R. Sherman and A. B. Craig. *Understanding Virtual Reality: Interface, Application and Design*. Morgan Kaufmann, 2002.

91. S. Smith, T. Marsh, D. Duke and P. Wright. Drowning in immersion. In *Proc. UK-VRSIG*, 1998.

92. B. G. Blundell and A. J. Schwarz. The classification of volumetric display systems: characteristics and predictability of the image space. *IEEE Trans. Visualization and Computer Graphics* 8(1): 66–75, 2002.

93. M. Halle. Autostereoscopic displays and computer graphics. *ACM SIGGRAPH Computer Graphics* 31(2): 58–62, 1997.

94. R. J. Stone. Haptic feedback: A potted history from telepresence to virtual reality. In *Proc. 1st Int. Workshop on Haptic Human-Computer Interaction*, vol. LNCS 2058, pp. 1–7, 2000.

95. C. Cruz-Neira, D. J. Sandin and T. A. DeFanti *et al.* The CAVE: Audio visual experience automatic virtual environment. *Communications of the ACM* 35(6): 65–72, 1992.

96. J. Kelso, L. E. Arsenault, S. G. Satterfield and R. D. Kriz. DIVERSE: a framework for building extensible and reconfigurable device independent virtual environments. In *Proc. IEEE Virtual Reality*, pp. 183–190. 2002.

97. RAVE. http://www.wesc.ac.uk/projects/rave.

98. J. Shalf and E. W. Bethel. The Grid and future visualization systems architectures. *IEEE CG&A* 23(2): 6–9, 2003.

99. C. Carlson and O. Hagsand. DIVE: a platform for multiuser virtual environments. *Computers and Graphics* 17(6): 1993.

100. C. M. Greenhalgh and S. D. Benford. MASSIVE: a virtual reality system for teleconferencing. *ACM Trans. Computer Human Interfaces* 2(3): 239–261, 1995.

101. W. Broll. Interacting in distributed collaborative VE. In *Proc. VRAIS*, 1995.

102. V. Normand and J. Tromp. Collaborative virtual environments: the COVEN project.In *Proc. the Framework for Immersive Virtual Environments Conf.*, 1996.

103. M. Slater, M. Usoh, S. Benford, D. Snowdon, C. Brown, T. Rodden, G. Smith and S. Wilbur. Distributed extensible virtual reality laboratory (DEVRL). In *Proc. 3rd Eurographics Workshop on Virtual Environments*, 1996.

104. S. D. Benford, J. Bowers, S. Gray, T. R. Rodden, M. Rygol and V. Stanger. The VirtuOsi project. In *Proc. London Virtual Reality Expo*, 1994.

105. R. T. Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4): 355–85, 1997.

106. C. Pinhanez. Augmenting reality with projected interactive displays. In *Proc. Int. Symp. Virtual and Augmented Architecture*, 2001.

107. D. LaRose. *A Fast, Affordable System for Augmented Reality*. Master's thesis, Robotics Institute, Carnegie Mellon University, April 1998.

108. J. Rekimoto and M. Saitoh. Augmented surfaces: A spatially continuous work space for hybrid computing environments. In *Proc. ACM CHI*, pp. 378–385, 1999.

109. D. Schmalstieg, A. Fuhrmann, G. Hesina, Z. Ari, L. Encarnac, A. Gervautz and W. Purgathofer. *The Studierstube Augmented Reality Project*. Tech. rep., TU Wien, 2000.

110. W. E. L. Grimson, T. Lozano-Pérez, W. M. Wells III, G. J. Ettinger, S. J. White and R. Kikinis. An automatic registration method for frameless stereotaxy, image guided surgery, and enhanced reality visualization. *IEEE Trans. Medical Imaging* 15(2): 129–140, 1996.

111. F. P. Vidal, F. Bello, K. Brodlie, N. W. John, D. Gould, R. Phillips and N. Avis. Medical visualization and virtual environments. In *Eurographics State of the Art Reports*, 2004.

112. I. Poupyrev, D. S. Tan, M. Billinghurst, H. Kato, H. Regenbrecht and N. Tetsutani. A survey of augmented reality. *IEEE Computer*, 2–9, 2002.

113. M. Billinghurst, H. Kato and I. Poupyrev. The MagicBook: A transitional AR interface. *Computers and Graphics*, 745–753, November 2001.

114. The High Performance Computing Center in Stuttgart: Visualization — augmented reality. http://www.hlrs.de/organization/vis/ar/.

115. C. S. Ang, D. C. Martin and M. D. Doyle. Integrated control of distributed volume visualization through the world wide web. In *Proc. IEEE Visualization*, pp. 13–20, 1994.

116. J. D. Wood, K. W. Brodlie and H. Wright. Visualization over the world wide web and its application to environmental data. In *Proc. IEEE Visualization*, pp. 81–86, 1996.

117. K. Engel, R. Westermann and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In *Proc. IEEE Visualization*, pp. 139–146, 1999.

118. C. K. Michaels and M. J. Bailey. VizWiz: A Java applet for interactive 3D scientific visualization over the web. In *Proc. IEEE Visualization*, 1997.

119. VNC. http://www.realvnc.com/.

120. NERSC:. http://acts.nersc.gov/cumulvs/.

121. pV3. http://raphael.mit.edu/pv3/pv3.html.

122. COVISE. http://www.vircinity.com/.

123. J. Wood, H. Wright and K. Brodlie. Collaborative visualization. In *Proc. IEEE Visualization*, pp. 253–259, 1997.

124. Access Grid documentation. http://www.accessgrid.org.

125. I. Foster and C. Kesselman. (Eds.) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

126. I. Foster and C. Kesselman. The Globus project: a status report. In *Proc. Heterogeneous Computing Workshop*, pp. 4–18, 1998.

127. D. W. Erwin and D. F. Snelling. UNICORE: a Grid computing environment. *Lecture Notes in Computer Science 2150*, 825–834, 2001.

128. EUROGRID. http://www.eurogrid.org.

129. GRIP. http://www.grid-interoperability.org/.

130. S. J. Chapin, D. Katramatos, J. Karpovich and A. S. Grimshaw. The Legion resource management system. In *Job Scheduling Strategies for Parallel Processing*,D. G. Feitelson, L. Rudolph, (Eds.) Springer Verlag, pp. 162–178, 1999.

131. Condor. http://www.cs.wisc.edu/condor.

132. Sun Microsystems, Inc. http://www.sun.com/solutions/infrastructure/grid.

133. I. Foster, C. J. N. Kesselman and S. Tuecke. The physiology of the Grid: an open Grid services architecture for distributed system integration. In *Open Grid Services Infrastructure WG*, 2002.

134. D. Snelling. UNICORE: A grid computing environment. In *Proc. EUROPAR* August, 2001.

135. J. Dunagan, R. Roussev, B. Daniels, A. J. C. Verbowski and Y. Wang. Towards self-managing software patching process using black-box persistent-state manifestes. In *Proc. 1st Int. Conf. on Autonomic Computing*, pp. 106–113, 2004.

136. M. Agarwal and M. Parashar. Enabling autonomic compositions in Grid environments. In *Proc. 4th Int. Workshop on Grid Computing*, pp. 34–41, 2003.

137. N. Kandasamy, S. Abdelwahed and J. Hayes. Self-optimization in computer systems via on-line control: applications to power management. In *Proc. 1st Int. Conf. on Autonomic Computing*, pp. 54–61, 2004.

138. S. Aiber, D. Gilat, A. Landau, N. Razinkov, A. Sela and S. Wasserkrug. Autonomic self-optimization according to business objectives. In *Proc. 1st Int. Conf. on Autonomic Computing*, pp. 206–213, 2004.

139. J. Norris, K. Coleman, A. Fox and G. Candea. OnCall: defeating spikes with a free-market application cluster. In *Proc. 1st Int. Conf. on Autonomic Computing*, pp. 1198–205, 2004.

140. M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan and E. Brewer. Failure diagnosis using decision trees. In *Proc. 1st Int. Conf. on Autonomic Computing*, pp. 36–43, 2004.

141. N. H. Minsky. On conditions for self-healing in distributed software systems. In *Autonomic Computing Workshop—Proc. 5th Int. Workshop on Active Middleware Services*, 2003.

142. T. Eymann, M. Reinicke, O. Ardaiz, P. Artigas, F. Freitag and L. Navarro. Self-organizing resource allocation for autonomic network. In *Proc. 14th Int. Workshop on Database and Expert Systems Applications*, pp. 656–660, 2003.

143. A. Lee and M. Ibrahim. Emotional attributes in autonomic computing systems. In *Proc. 14th Int. Workshop on Database and Expert Systems Applications*, pp. 681–685, 2003.

144. N. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. In *ACM Trans. Software Engineering and Methodology*, 2000.

145. W. E. Walsh, G. Tesauro, J. O. Kephart and R. Das. Utilities functions in autonomic systems. In *Proc. 1st Int. Conf. on Autonomic Computing*, pp. 70–77, 2004.

146. C. Linn. Semantic reliability in distributed systems: ontology issues and system engineering. In *Proc. IEEE/WIC Int. Conf. on Web Intelligence*, pp. 292–300, 2003.

147. G. Tziallas and B. Theodoulidis. Building autonomic computing systems based on ontological component models. In *Proc. 14th Int. Workshop on Database and Expert Systems Applications*, pp. 674–680, 2003.

148. S. Anderson, M. Hartswood, R. Procter, M. Rouncefield, R. Slack, J. Soutter and A. Voss. Making autonomic computing systems accountable: the problem of human computer interaction. In *Proc. 14th Int. Workshop on Database and Expert Systems Applications*, pp. 718–724, 2003.

149. Z. Constantinescu. Towards an autonomic distributed computing environment. In *Proc. 14th Int. Workshop on Database and Expert Systems Applications*, pp. 699–703, 2003.

150. X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri and S. Rao. Autonomia: an autonomic computing environment. In *Proc. IEEE Int. Conf. on Performance, Computing, and Communications*, pp. 61–68, 2003.

151. G. Deen, T. Lehman and J. Jaufman. The almaden optimalgrid project. In *Autonomic Computing Workshop—Proc. 5th Int. Workshop on Active Middleware Services*, 2003.

152. E. E. Koutsofio. Visualizing large-scale telecommunication networks and services. In *Proc. IEEE Visualization*, 1999.

153. G. W. Daniel and M. Chen. Video visualization. In *Proc. IEEE Visualization*, pp. 409–416, 2003.

154. P. Cignoni, C. Montani, E. Puppo and R. Scopigno. Optimal isosurface extraction from irregular volume data. In *Proc. IEEE Symp. Volume Visualization*, pp. 31–38, 1996.

155. Y.-J. Chiang and C. T. Silva. I/O optimal isosurface extraction. In *Proc. IEEE Visualization*, pp. 293–300, 1997.

156. P. M. Sutton and C. D. Hansen. Accelerated isosurface extraction in time-varying fields. *IEEE Trans. Visualization and Computer Graphics* 6(2): 98–107, 2000.

157. S.-K. Ueng, C. Sikorski and K.-L. Ma. Out-of-core streamline visualization on large unstructured meshes. *IEEE Trans. Visualization and Computer Graphics* 3(4): 370–380, 1997.

158. P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proc. ACM SIGGRAPH*, pp. 259–262, 2000.

159. H.-W. Shen, L.-J. Chiang and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proc. IEEE Visualization*, pp. 371–378, 1999.

160. R. Farias and C. T. Silva. Out-of-core rendering of large, unstructured grids. *IEEE CG&A* 21(4): 42–50, 2001.

161. M. Pharr, C. Kolb, R. Gershbein and P. Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *Proc. ACM SIGGRAPH*, pp. 101–108, 1997.

162. T. Kurc, Ü. Çatalyürek, C. Chang, A. Sussman and J. Saltz. Visualization of large data sets with the active data repository. *IEEE CG&A* 21(4): 24–33, 2001.

163. J. Wilhelms and A. van Gelder. Octrees for faster isosurface generation. *ACM Trans. Graphics* 11(3): 201–227, 1992.

164. M. W. Jones. *The Visualisation of Regular Three Dimensional Data*. PhD thesis, University of Wales, 1995.

165. D. Laur and P. Hanrahan. Hierarchical splatting: a progressive refinement algorithm for volume rendering. *ACM SIGGRAPH Computer Graphics* 25(4): 285–288, 1991.

166. J. Wilhelms, A. van Gelder, P. Tarantino and J. Gibbs. Hierarchical and parallelizable direct volume rendering for irregular and multiple grids. In *Proc. IEEE Visualization*, 1996.

167. Y. Livnat and C. Hansen. View dependent isosurface extraction. In *Proc. IEEE Visualization*, pp. 175–180, 1998.

168. E. LaMar, B. Hamann and K. Joy. Multiresolution techniques for interactive texture-based volume visualization. In *Proc. IEEE Visualization*, pp. 355–362, 1999.

169. J. T. Klosowski and C. T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Trans. Visualization and Computer Graphics* 6(2): 108–123, 2000.

170. A. Norton and A. Rockwood. Enabling view-dependant progressive volume visualization on the grid. *IEEE*

CG&A 23(2): 22–31, 2003.

171. S. Wang and A. Kaufman. Volume sampled voxelization of geometric primitives. In *Proc. IEEE Visualization*, pp. 78–84, 1993.

172. M. W. Jones. The production of volume data from triangular meshes using voxelisation. *Computer Graphics Forum* 15(5): 311–318, 1996.

173. M. Chen and J. V. Tucker. Constructive volume geometry. *Computer Graphics Forum* 19(4): 281–293, 2000.

174. A. S. Winter and M. Chen. vlib: a volume graphics API. In *Proc. Volume Graphics*, pp. 133–147, 2001.

175. K.-L. Ma and S. Parker. Massively parallel software rendering for visualizing large-scale data sets. *IEEE CG&A* 21(4): 72–83, 2001.

176. M. Sarkar and M. Brown. Graphical FishEye views of graphs. In *Proc. ACM CHI*, pp. 83–91, 1992.

177. J. D. Mackinlay, G. G. robertson and S. K. Card. The perspective wall: detail and context smoothly integrated. In *Proc. ACM CHI*, pp. 172–180, 1991.

178. T. Munzner. Exploring large graphs in 3d hyperbolic space. *IEEE CG&A* 18(4): 18–23, 1998.

179. M. Sarkar, S. S. Snibbe, O. Tversky and S. P. Reiss. Stretching the rubber sheet. In *Proc. ACM Symp. User Interface Software and Technology*, 1993.

180. Y. Kurzion and R. Yagel. Space deformation using ray deflectors. In *Proc. 6th Eurographics Workshop on Rendering*, pp. 21–32, 1995.

181. M. S. T. Carpendale, D. J. Cowperthwaite and F. D. Fracchia. Extending distortion viewing from 2D to 3D. *IEEE CG&A* 17(4): 42–51, 1997.

182. S. M. F. Treavett and M. Chen. Pen-and-ink rendering in volume visualization. In *Proc. IEEE Visualization 2000*, pp. 203–210, 2000.

183. E. LaMar, B. Hamann and K. I. Joy. A magnification lens for interactive volume visualization. In *Proc. IEEE Pacific Conf. on Computer Graphics and Applications*, pp. 223–231, 2001.

184. M. Hadwiger, C. Berger and H. Hauser. High-quality two level volume rendering of segmented data sets on consumer graphics hardware. In *Proc. IEEE Visualization*, pp. 301–308, 2003.

185. M. Chen, J. V. Tucker, R. H. Clayton and A. V. Holden. Constructive volume geometry applied to visualization of cardiac anatomy and electrophysiology. *Int. J. Bifurcation and Chaos* 13(12): 3591–3604, 2003.

186. P. Wong. Visual data mining. *IEEE CG&A* 19(5): 20–21, 1999.

187. J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth and P. J. Haas. Interactive data analysis: The control project. *Computer* 32(8): 51–58, 1999.

188. A. Hinneburg, D. A. Keim and M. Wawryniuk. Hd-eye: Visual mining of high-dimensional data. *IEEE CG&A* 19(5): 22–31, 1999.

189. B. Shneidermann. Tree visualization with treemaps: a 2D space filling approach. *ACM Trans. Graphics* 11(1): 92–99, 1992.

190. J. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial space-filling hierarchy visualization. In *Proc. IEEE Information Visualization*, pp. 57–65, 2000.

191. D. A. Keim, H. P. Kriegel and M. Ankerst. Recursive pattern: a technique for visualizing very large amounts of data. In *Proc. IEEE Visualization*, pp. 279–286, 1995.

192. M. Ankerst, D. A. Keim and H. P. Kriegel. Circle segments: a technique for visual exploring large multidimensional data sets. In *Proc. IEEE Visualization*, p. Hot Topic Session, 1996.

193. H. Theisel and M. Kreuseler. An enhanced spring model for information visualization. *Computer Graphics Forum* 17(3): 335–343, 1998.

194. T. Kohonen. *Self-Organizing Maps*, 2nd ed. Springer, 1997.

195. J. Marks, B. Andalman and P. A. Beardsley *et al*. Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proc. ACM SIGGRAPH*, pp. 389–400, 1997.

196. G. Kindlmann and J. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proc. IEEE Symp. Volume Visualization*, pp. 79–86, 1998.

197. E. Gallopoulos, E. N. Houstis and J. R. Rice. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computional Science and Engineering* 1(2): 11–23, 1994.

198. G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel and J. Shalf. Cactus tools for Grid applications. *Cluster Computing* 4(3): 179–188, 2001.

199. C. Upson, T. Faulhaber Jr., D. Kamins, D. H. Laidlaw,

D. Schlegel, J. Vroom, R. Gurwitz and A. van Dam. The application visualization system: A computational environment for scientific visualization. *IEEE CG&A* 9(4): 30–42, 1989.

200. R. Marshall, J. Kempf, S. Dyer and C. Yen. Visualization methods and simulation steering for a 3D turbulence model for Lake Erie. *ACM SIGGRAPH Computer Graphics* 24(2): 89–97, 1990.

201. H. Zhou, M. Chen and M. Webster. Comparative evaluation of visualization and experimental results using image comparison metrics. In *Proc. IEEE Visualization*, pp. 315–322, 2002.

202. J. J. van Wijk and R. van Liere. An environment for computational steering. In *Scientific Visualization: Overviews, Methodologies, and Techniques*, G. M. Nielson, H. Müller, H. Hagen, (Eds.), 1997.

203. R. van Liere, J. Mulder and J. van Wijk. Computational steering. *Future Generation Computer Systems* 12(5): 1997.

204. J. Mulder, J. van Wijk and R. van Liere. A survey for computational steering environments. *Future Generation Computer Systems* 15(2): 1999.

205. M. S. G. Parker, C. D. H. Miller and C. R. Johnson. An integrated problem solving environment: the SCIRun computational steering system. In *Proc. 31st Hawaii Int. Conf. on System Sciences*, 1998.

206. J. M. Brooke, P. V. Coveney, J. Harting, S. Jha, S. M. Pickles, R. L. Pinning and A. R. Porter. Computational steering in RealityGrid. In *Proc. UK e-Science All Hands Meeting*, 2003.

207. J. Wood, K. Brodlie and J. Walton. gViz—visualization and steering for the Grid. In *Proc. e-Science All Hands Meeting*, 2003.

208. F. Sogandares. Stone axes and warhammers: a decade of distributed simulation in aviation research. In *Proc. 16th Workshop on Parallel and Distributed Simulation*, pp. 125–132, 2002.

209. T. Peters and B. Davey *et al*. Three dimensional multimodal image guidance for neurosurgery. *IEEE Trans. Medical Imaging* 15: 121–128, 1996.

210. R. F. McCloy and N. W. John. Remote visualization of patient data in the operating theatre during helpatopancreatic surgery. In *Computer Assisted Radiology and Surgery*. Elsevier, pp. 53–58, 2003.

211. R. M. Fujimoto. Advanced tutorials: Parallel and distributed simulation systems. In *Proc. 33nd Conf. on Winter Simulation*, pp. 147–157, 2001.

212. M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM* 36(7): 75–84, 1993.

213. S. Izadi, P. Coutinho, T. Rodden and G. Smith. The FUSE platform: supporting ubiquitous collaboration within diverse mobile environments. *Automated Software Engineering* 9(2): 167–186, 2002.

214. F. Lamberti, C. Zunino, A. Sanna, A. Fiume and M. Maniezzo. An accelerated remote graphics architecture for PDAS. In *Proc. 8th Int. Conf. on 3D Web Technology*, pp. 55–ff, 2003.

215. M. Wolf, Z. Cai, W. Huang and K. Schwan. Smartpointers: personalized scientific data portals in your hand. In *Proc. ACM/IEEE Conf. on Supercomputing*, pp. 1–16, 2002.

216. P. Hartling, A. Bierbaum and C. Cruz-Neira. Virtual reality interfaces using Tweek. In *SIGGRAPH 2002 Sketches*, 2002.

217. B. D'Amora and F. Bernardini. Pervasive 3D viewing for product data management. *IEEE CG&A* 23(2): 14–19, 2003.

218. J. Pascoe, N. Ryan and D. Morse. Using while moving: HCI issues in fieldwork environments. *ACM Trans. Computer-Human Interaction* 7(3): 417–437, 2000.

219. T. Hoare, R. Milner, M. Thomas and A. Bundy. Criteria for a grand challenge. http://www.cra.org/Activities/grand.challenges/hoare.pdf, 2002.

220. IBM: Autonomic deployment model. http://www-306.ibm.com/autonomic/levels.shtml, 2004.