IPP-HURRAY! Research Group

http://www.hurray.isep.ipp.pt

Polytechnic Institute of Porto
School of Engineering (ISEP-IPP)

A Survey of Techniques and Technologies for Web-Based Real-Time Interactive Rendering

Eduardo TOVAR
Filipe PACHECO

HURRAY-TR-0108 (draft version)
March 2001

A Survey of Techniques and Technologies for Web-Based Real-Time Interactive Rendering Systems

Eduardo TOVAR, Filipe PACHECO

IPP-HURRAY! Research Group
Polytechnic Institute of Porto (ISEP-IPP)
Rua Dr. António Bernardino de Almeida, 431
4200-072 Porto
Portugal
Tel.: +351.22.8340502, Fax: +351.22.8321159
E-mail: {emt, ffp}@dei.isep.ipp.pt
http://www.hurray.isep.ipp.pt

Abstract:

When exploring a virtual environment, realism depends mainly on two factors: realistic images and real-time feedback (motions, behaviour etc.). In this context, photo realism and physical validity of computer generated images required by emerging applications, such as advanced e-commerce, still impose major challenges in the area of rendering research whereas the complexity of lighting phenomena further requires powerful and predictable computing if time constraints must be attained. In this technical report we address the state-of-the-art on rendering, trying to put the focus on approaches, techniques and technologies that might enable real-time interactive web-based client-server rendering systems. The focus is on the end-systems and not the networking technologies used to interconnect client(s) and server(s).

PROJECT PARTNERS:
Siemens SBS C-LAB, Germany (Project Co-ordinator)
Bertelsmann mediaSystems GmbH, Germany
Eptron Multimedia, Spain
Instituto Politécnico do Porto - ISEP-IPP, Portugal
Institute Laboratory for Mixed Realities at the Academy of Media Arts Cologne, LMR, Germany
Mälardalen Real-Time Research Centre (MRTC) at Mälardalen University in Västerås, Sweden
Q-Systems, Greece

1. Introduction

Visualisation provides additional insights to results which would otherwise be displayed as text or numbers [1]. It is a form of communication which is universal, and which has the ability to form an abstraction of the real world into a graphical representation which is comprehensible to a wide range of people. The society is becoming dependent on information presented in three-dimensional visual format [2], virtual reality is no longer a word only applicable to computer games.

Virtual and augmented reality (VR/AR) are increasingly being used in various business scenarios and are important driving forces in technology development. Major architectural and design decisions are based on 3D models and virtual explorations of the objects to be designed. The same techniques can also be used for virtual shopping in e-commerce applications as well as provide for exciting possibilities in the entertainment market.

When exploring a virtual environment, realism depends mainly on two factors: realistic images and real-time feedback (motions, behaviour etc.). In this context, photo realism and physical validity of computer generated images required by emerging applications, such as advanced e-commerce, still impose major challenges in the area of rendering research whereas the complexity of lighting phenomena further requires powerful and predictable computing if time constraints must be attained.

Appropriate real-time feedback imposes tough performance and predictability requirements on both the end-systems (clients and servers) and on the network. The Walkabout project addresses interactive web-based client-server rendering systems to provide realistic real-time walkthroughs.

In this technical report we address the state-of-the-art on rendering, trying to put the focus on approaches, techniques and technologies that might enable real-time interactive web-based client-server rendering systems. The focus is on the end-systems and not the networking technologies used to interconnect client(s) and server(s).

The rest of the technical report is organised as follows. In Section 2 we survey the rendering fundamentals. Polygon-based rendering is given a special relevance since the most popular way of representing virtual worlds is by means of polygons. Alternative ways to represent virtual worlds are volumes and images. Both are mentioned, with particular reference to their potential for realistic real-time rendering, in particular in what concerns image-based rendering. In Section 3 we address the parallelisation of rendering as a means of fulfilling timing requirements. Emphasis is given to clusters of personal workstations (PWS). Networking technologies such as switched-Ethernet or Myrinet are enabling are becoming available to be efficiently explored in the implementation of cost-effective high-performance cluster renderers. MPI or PVM standards are addressed as two different paradigms for the programming of parallel environments. The parallelisation of the rendering process is then addressed for both polygon and volume-based approaches. In Section 4 we discuss the general aspects of the architecture of web-based client-server rendering systems. Two different approaches, a server-based rendering and a client-based rendering, are compared and advantages of targeting a hybrid approach are also drawn. Finally, in Appendix A, we survey more deep details about parallelisation in polygon-based rendering.

Sections 2.5, 3.5 and 4 outline some research/innovation perspectives for Walkabout.

2. Rendering Fundamentals

In computer graphics, *rendering* is the process by which an abstract description of a scene is converted to an image. A scene is a collection of geometrically-defined objects in *three-dimensional (3D) object space*, with the associated lightning and viewing parameters [3]. The rendering operation illuminates the objects and projects them into *two-dimensional (2D) image space*, where color intensities of individual pixels are computed to yield a final image.

For complex scenes or high-quality images, the rendering process is computationally intensive, requiring millions or billions of floating-point and integer operations for each image. The need for interactive or real-time response in many applications places additional demands on processing power. The only practical way to obtain the needed computational power is to exploit multiple processing units to speed up the rendering task, a concept which has become known as parallel rendering.

2.1. Representation of Object Space

Traditional 3D graphics are based on surface representation. A collection of 3D points (vertices) and the interconnection of these vertices (edges) form a polygon. A set of polygons is used to form a computer graphics object. A collection of these objects forms a virtual world. There are wide ranges of techniques that can be used for generate an image of the virtual world. Some of the techniques, such as ray tracing [4, 5] and radiosity [6], are extremely computationally expensive, but they can be used to generate photo-realistic quality images.

Simpler polygon-based rendering algorithms using Phong or Gouraud shaded polygons are comparatively less expensive. In either case same steps of the rendering can be speed-up by using graphics acceleration hardware. Hence, these rendering methods may be ideal for interactive rendering. However, in this simpler version, the image generated will most of the times appear artificial to the viewer. This is due to the approximations made by these techniques. The trade-off of these approximations for the rendering speed may be acceptable [7]. *Polygon-based rendering* is the most common rendering technique for which affordable special-purpose rendering hardware has been developed in the last decade.

2.2. Details on the Polygon-Based Rendering

The rendering of a virtual world made up of objects represented as polygons can be accomplished by using a standard eighth-stage rendering pipeline [13].

1. *Selection of Visible Objects*. All the potentially visible objects in the virtual world are selected. By deselecting some visible objects, selecting some new object from the database, and changing the orientation of the visible objects, the virtual world appears to have a time component.
2. *Co-ordinate Transformation*. The second step is to transform all the selected objects to their proper orientation in the virtual environment. The transformation uses translation, rotation and scaling.
3. *Visibility Calculation*. The objects are checked to see if they will appear in the image. The image is the only window for the viewer to peek into the virtual world. Eliminating the unseen objects reduces the total amount of rendering effort. An object is visible if its volume intersects or is contained within the viewing volume of the virtual world. The volume intersection calculations can be CPU intensive, if the object has a complex shape.
4. *Lighting Calculation*. The color of the objects is computed by using the properties of the light and the surface properties of the object. The properties of a light source consist of its illuminating direction, position, and color. The type of light source, such as directional or point, determines the illumination direction. In order to determine if a surface is illuminated or not by a light, the direction of the surface normal and the illuminating direction of the light source are compared by computing the dot product of the vectors. This does not take into account the shadows cast by nearby objects (omitting shadow is one of many approximations to minimise rendering and resources and time). Several ray tracing and radiosity algorithms are described in the bibliography. Ray tracing algorithms are simpler but do not take adequately into account the entire scene and the light interactions between lights and objects.
5. *Viewing Transformation*. The objects are transformed according to the viewing specification. For example, the perspective transformation scales the objects by a factor inversely proportional to the distance between the viewpoint and the objects.
6. *Clipping*. The polygon-based objects are clipped according to the view volume. Any invisible objects, which escaped from step three, will be eliminated entirely. In addition, objects that are behind, too close or too far from the viewer are also eliminated. The clipping computation is done by using line and plane intersection calculations.
7. *Mapping to Viewpoint*. Incorporate the view requirements into the objects' co-ordinates. After the view transformation, the vertex co-ordinates of an object changes from (x, y, z, 1) to the form of (x', y', z', w). Hence, the view transformation process can be considered as mapping the vertices in subspace 1 on to other 3D subspaces in the 4D universe. The w co-ordinate of the vertices is being used to keep track of their relative distance from the viewpoint. This information is incorporated back to the object's 3D co-ordinates by dividing the x, y and z values with the w values. Then the 3D objects are projected onto 2D-screen space in the preparation of their rasterisation.
8. *Rasterisation*. Finally, the 2D objects are drawn on to the image buffer pixel by pixel, which is called rasterisation. The rasterisation of the 3D objects is the same as taking a photograph of the objects, which is a parallel projection of the objects onto a 2D space. Then the 2D space is scaled to the desired resolution. A Z-buffer records the depth information along with the color value of each pixel. Only the pixel closest to the projection plane is stored, and the pixel contributed by objects further away will be ignored. As a result, a correct image can be generated even with arbitrary rasterisation order.

2.3. Alternative Representations of Object Space

As an alternative representation of surfaces, some other authors [8,9] have been exploring volumes. A volume is a 3D array of cubic elements, each representing a unit of space. A volume is used to store 3D geometric objects and 3D pictures. The main disadvantage of volumes is their immense size. A medium resolution volume of $256 \times 256 \times 256$ requires storage for 16 million voxels. These have to be processed in order to generate an image of this 3D object onto a 2D screen. However, volume have some major advantages: they can represent the interior of objects and not only the outer shell, like surfaces do. *Volume-based rendering* and processing does not depend of the object's complexity or type, it depends only on the volume resolution. It easily supports operations such as subtraction, addition, collision detection, and deformation. For a complete comparison see [8].

In the pursuit of photo-realism in conventional polygon-based computer graphics, models have become so complex that most of the polygons are smaller than one pixel in the final image. When models are simple and the polygon (typically a triangle) is large, the ability to specify large connected regions with only three points was a considerable efficiency in storage and computation. Now that models contain nearly as many primitives as pixels in the final image, one should re-think the use of geometric primitives to describe complex environments. Some authors are investigating an alternative approach that represents 3D environments with sets of images [10,11]. The images include information describing the depth of each pixel along with the color and other properties. Algorithms have been developed for processing the depth-enhanced images to produce new images from viewpoints that are not included in the original image set (object space). Thus, using a finite set of source images, it is possible to produce new images from arbitrary viewpoints. The potential impact of using images to represent complex 3D environments is immense. The naturally "photo-realistic" rendering enables immersive 3D environments to be constructed for real places. Computation is proportional to the number of output pixels rather than to the number of geometric primitives. This should allow implementation of systems that produce high-quality, 3D imagery with much less hardware than used in current high-performance graphics systems. Some techniques like the post-rendering warping [12] allows the rendering rate and latency to be decoupled from the user's changing viewpoint. It is expected that this approach will enable immersive 3D systems to be implemented over long distance networks and broadcast media, using inexpensive image warpers to interface to the network and to increase interactivity.

2.4. Real-Time and Interactive Rendering

Numerous speed-up techniques can be applied individually to the different stages (or group of stages) in the rendering process. Recently, parallelisation started also to be a cost-effective solution for speeding up (see Section 3).

However, real-time does not mean fast.

Timely computations in rendering have been scarcely exploited by researchers. Some authors base their efforts in exploiting the use of the concept of *Levels of Detail* (LODs) of the virtual world description, which appeared first described in [24]. Basically, frames are rendered using different (in terms of complexity details) representations of the virtual world, using information from concerning the computation of previous frames. Some examples of this can be found in [25,26]. In these systems the resolution or the level of tessellation of the objects depends on the point of view of the scene. The basic idea is to simplify the far away objects and so increase the frame rate. This can be used by limiting the number of polygons [25] or using computation time from previous frames to adapt the LOD of the next frame [26] and so achieve a quasi-constant frame rate. More recent works, such as [27], use more advanced time calculation heuristics to support greater changes in computation needs from frame to frame.

Other group of solutions use priority based rendering algorithms (e.g., Priority-Layered Projection [28] or Qsplat [29]) that can produce fast results albeit with some errors. The great advantage of these solutions is that it is not so complex to limit the number of cycles of the algorithm. It is possible to dynamically increase the quality of the images by using a slower frame rate or, in the opposite direction, use faster frame rate with more errors on the final image. This prioritisation is in most cases a two-phase situation. In the setup phase the data is processed and can be rearranged or have additional information computed like priority or position hints. On the second phase the algorithm uses viewpoint information and the data gathered in the setup phase to make smart guesses on the rendering process.

2.5. Research/Innovation Perspectives for Walkabout

The optimisation of the rendering process is deeply dependent on the virtual world model. In Walkabout, investigation must take place in order to define the best rendering pipeline approach matching the overall requirements. Research will be conducted essentially at the representation level, including defining the best strategy concerning the eventual use of LODs, visibility, culling and lightning calculations. The objective will be to have time control over the overall rendering process. This research will be conducted taking into account the interactions between client(s) server, and the use of clusters of personal workstations (see Section 3), since the capabilities of the target platform influence the choice of rendering algorithms.

Image-based rendering (see Section 2.3) is quite new in the arena. In Walkabout image-based representations of the virtual world will be assessed in terms of potential use in future Walkabout systems. The enormous potential is that images provide "natural" photo-realism, while rendering computation is proportional to the number of output pixels rather than to the number of geometric primitives and complexity from viewpoints. This presents a great advantage in terms of timely computing.

3. Parallelism in Computer Graphics

Specialised hardware is not the only way to speed up computer graphics rendering. In recent years, there have been attempts to exploit the parallelism in computer graphics.

Parallel computers provide a very cost-effective ways to solve highly computational intensive problem with built-in parallelism [14]. There are four types of parallelism that can be exploited: time, large grain (task), medium grain (sub-task and/or small pocket data) and fine grain (data). Time parallelism exists when a program is run multiple times with different parameters. For example, in a flight simulation program successive images are rendered with a slight change of viewing locations. Although these images are related in time, their rendering processes are independent of each other, and therefore can be executed in parallel.

The amount of data and the complexity of calculation involved per parallel task classify the different granularity of parallelism. Let us assume that the flight simulation virtual world allows other aircraft. Depending on the aircraft model and the desired detail, it may be composed of hundreds to millions of polygons. Since the aircraft are independent of each other, they can be rendered separately onto separate intermediate image buffers. Since the amount of data and the work is considerable, this can be considered as large grain parallelism. The rendering of each polygon can be considered as the medium grain parallelism. Each polygon is a self-contained data package, so all the polygons in the virtual world can be rendered separately. When a polygon is rasterised, each pixel that is covered by the polygon is visited. However, the computation incurred by each pixel is independent of each other. Consequently, the work can be performed in parallel. Although the amount of work involved is small, the number of pixels is large, so the exploitation of the fine-grain parallelism is significant.

Two examples of parallel rendering that exploit the aforementioned parallelism are a terrain rendering program [15] and a general parallel rendering algorithm [16]. In this last, polygon rendering capitalises on the independence of polygon and pixel. All the polygons in the virtual world are evenly distributed onto a MIMD computer's nodes. All the nodes process their polygons by using a shortened (only step 2 to 7 of those described in §2.2) standard rendering pipeline. Then, the polygons are re-distributed according to their screen co-ordinates to the node that manages that part of the image buffer. Finally, the nodes concurrently and independently rasterise their collection of the polygons. Combining the distributed image buffer forms the final image.

By distributing the work among many computing nodes, the rendering process can be speeded up. However, a partial image is formed on each node after the concurrent rendering. In order to obtain the final image, all the intermediate image buffers must be combined in the correct order. This is accomplished by using the communication network that connects all the computation nodes.

3.1. Building Parallel Renderers from Personal Workstations

Two forms of parallelism can be considered: tightly coupled processors in a SMP (single computer multiple processor) configuration and a cluster of workstations connected over networks. While in a single-processor machine CPU performance is often the most important factor in determining rendering performance, parallel configurations add specific constraints to the performance of parallel rendering algorithms. For SMP workstations, the performance is affected by memory and disk bandwidth. For workstation clusters, the disk and network bandwidth are the most important parameters influencing the rendering performance.

Parallel rendering algorithms can be implemented on a variety of platforms. The capabilities of the target platform influence the choice of rendering algorithms. For instance the availability of hardware acceleration for certain rendering operations affects both performance and scalability of the rendering algorithm.

The developments in the Personal Workstations (PWS) market reflect the PWS's dual-inheritance from Unix workstations and PCs. As the NT workstation markets matures the price gap between the best performing systems and the systems with best price-performance appears to be closing. This is a known trend from the desktop PC market which has turned into a commodity market. The PWS market is assuming characteristics of a commodity market with surprising speed, i.e. most products are very similar and have to compete through pricing, marketing and support offerings. At the same time, PWSs remain different from desktop PCs – and are similar to Unix workstations – in that application performance (in contrast to serviceability and manageability) is the primary design and deployment objective. Most purchasing decisions are heavily based on the results in standard and customer-specific application benchmarks.

A particularly interesting question is whether PWSs offer inherently better price-performance than traditional Unix workstations. Over the period that both workstation types participated in the market (1996-1999), NT workstations as a whole have consistently delivered better price-performance than Unix workstations for standard benchmarks. Only recently (mid 1998) Unix workstations are beginning to reach the same price-performance levels. It is unclear whether this constitutes a reversal of the earlier trend or whether the gap will be restored when Intel delivers its next generation processors. Another explanation for the narrowing of this gap is that NT workstations are starting to include high-performance subsystems that are required for balanced systems.

Several approaches to implementing parallel polygon rendering on PWSs with graphics accelerators have been investigated in [17]. It should be noted that this analysis does not consider pure software implementations of the rendering pipeline; rasterisation was assumed to be performed by a graphics adapter. This is in contrast to software-only graphics pipelines. Such approaches lead to more scaleable rendering systems, even though both absolute performance and price-performance are likely to be worse than the hardware-accelerated implementation. In [18] parallel software renderers have shown close to linear speedup up to 100 processors in a BBN Butterfly TC2000 even though the absolute performance (up to 100,000 polygons/sec) does not match the performance available from graphics workstations of equal or lower cost. However, software renderers offer more flexibility in the choice of rendering algorithms, e.g. advanced lighting models, and the option to integrate application and renderer more tightly.

Following the conclusions from [17] we will now look at the various subsystems in a PWS that may become a bottleneck for parallel rendering. In part, PWSs have inherited these bottlenecks from their desktop PC ancestors. For example, both memory and disk subsystems are less sophisticated than those of traditional workstations.

1. *Applications and Geometry Pipeline.* As pointed out above, CPU portion of the overall rendering time scales well with the number of processors. Therefore, it is desirable to parallelize rendering solutions with a large computational component. Advance rendering algorithms such as advanced lighting algorithms or ray-tracing will lead to implementations that scale to larger numbers of processors.
2. *Processor.* Contrary to initial intuition, the performance of CPU and rasteriser does not significantly influence the overall rendering performance. Therefore, parallel rendering does not benefit from enhancements to the CPU, such as by higher clock frequency, more internal pipelines or special instructions to accelerate certain portions of the geometry pipeline. However as stated earlier, faster CPUs may benefit the applications performance.
3. *Memory Subsystem.* Currently, memory bandwidth does not limit rendering performance as much as disk and network performance. Memory subsystems will keep increasing their performance over time and retain their relative performance compared to disks and networks.
4. *Disk Subsystem.* The disk subsystem offers ample opportunity for improvements over the standard IDE or SCSI found in todays PWSs. Faster disk subsystems can be used to alleviate this problem.
5. *Graphics Subsystem.* In workstation clusters the use of graphics accelerators with geometry accelerators can be beneficial. For applications with mostly static scenes, e.g. walkthroughs or assembly inspections, the use of retained data structures like display lists can reduce the bandwidth demands on system memory as geometry and lighting calculations are performed locally on the adapter. In SMP machines or for single-frame rendering faster graphics hardware will not provide large rendering speed-ups.
6. *Network.* In clusters, a slow network interconnect can become the dominant bottleneck. Increasing the network bandwidth by an order of magnitude will alleviate that problem. Current shortcomings of the protocol implementations prevent full realisation of the benefits of Gigabit-Ethernet under Windows NT. Alternative technologies, like Myrinet [19] promise higher sustained bandwidth than Ethernet. However, these technologies are either not available under Windows NT or have not yet been developed into a product. Prototype implementations under Unix (Linux) have demonstrated the advantages of such networks.

As Personal Workstations are emerging as an alternative to traditional workstations for technical applications they are frequently considered as building blocks for affordable parallel rendering. Even though PWS are used for parallel rendering in at least one commercial rendering package [20], its actual implementation is hampered by the lack of efficient networking technologies and insufficient disk performance. Improving these subsystems is possible but will result in more expensive systems, eliminating some of the perceived cost advantage of PWS over traditional workstation.

3.2. Memory Models

Many aspects of parallel programming depend on the memory architecture of a system, and many problems arise from a chosen memory architecture. The basic question is if the memory is assigned to the processor level, or if the memory is assigned on system level. This information is important for the distribution of a problem to the system. If all memory – except caches – is accessible from each part of the system – memory is assigned on system level, we are talking of a shared memory system. In case the individual processing elements can only access their own private memory – memory is assigned on processor level, we are talking of a distributed memory system. Shared memory systems are further divided into UMA (Uniform Memory Access) systems (not interchangeable with Uniform Memory Architecture), and into NUMA (Non-Uniform Memory Access) systems.

In distributed memory systems, the memory is assigned to each individual processor. At the beginning of the processing, the system is distributing the tasks and the data through the network to processing elements. These processing elements receive the data and their task and start to process the data. At some point, the processors need to communicate with other processors, in order to exchange results, to synchronise for periphery devices, and so forth. Finally, the computed results are sent back to the appropriate receiver and the processing element waits for a new task. Workstation clusters fit into this category, because each computer has its individual memory, which is (usually) not accessible from its partner workstations within the cluster. Furthermore, each workstation can distribute data via the network. Overall, it is important to note that communication in a distributed memory system is expensive. Therefore, it should be reduced to a minimum.

UMA systems contain all memory (main memory) in a more or less monolithic block. All processors of the system access this memory via the same interconnect, which can be a crossbar or a bus. In contrast, NUMA systems are combined of two or more UMA levels which are connected via another interconnect. This interconnect can be slower than the interconnect on the lower level. However, communication from one UMA sub-system to another UMA sub-system travels through more than one interconnection stage and therefore takes more time than communication within one UMA sub-system.

If UMA systems have a better communication, why should one use NUMA systems? The answer is that the possibilities to extend UMA systems are limited. At some point the complexity of the interconnect will virtually rise into infinity, or the interconnect will not be powerful enough to provide sufficient performance. Therefore, a hierarchy of UMA sub-systems was introduced. A special case of NUMA systems is cache-coherent NUMA (ccNUMA). This scheme ensures (usually in hardware) that the memory view of the execution entities (i.e., threads) is identical.

3.3. Programming Models

There are basically two different paradigms for the programming of parallel environments.

3.3.1. Message-Passing

The message-passing paradigm connects processing entities to perform a joined task. As a matter of principle, each processing entity is an individual process running on a computer. However, different processes can run on the very same computer, especially, if this computer is a multiprocessor system. The underlying interconnection topology is transparent from the user point of view. Therefore, it does not make a difference in programming, if the parallel program which communicates using a message-passing library runs on a cluster of workstations (i.e., Beowulf), on a distributed memory system (i.e., the IBM RS6000/SP), or on a shared memory system (i.e., the SGI 2x00). For the general process of using a message-passing system for concurrent programming it is essential to manually split the problem to be solved into different more or less independent sub-tasks. These sub-tasks and their data are distributed via the interconnect to the individual processes. During processing, intermediary results are sent using the explicit communication scheme of message-passing. Considering the high costs using the network, communication must be reduced to a minimum and the data must be explicitly partitioned. Finally, the terminal results of the processing entities are collected by a parent process which returns the result to the user. There are several message-passing libraries around. However, most applications are based on two standards: the PVM [21] library (Parallel Virtual Machine) and the MPI [22] standard (Message Passing Interface).

Generally, MPI was designed for message-passing on multiprocessors, while PVM was originally intended for message-passing within a heterogeneous network of workstations (NOW, clusters). Based on these different concepts, MPI has a strong emphasis on portability (a MPI-based application can be compiled on any system) and highly optimised performance, but it provides only a very limited functionality for session management (MPI 2.0 supports functions to spawn processes from a parent process). In contrast, PVM emphasises interoperability (PVM processes are supposed to communicate with processes build on completely different machines) using the concept of a virtual machine. This requires dynamic resource management – to compensate for the possible failure of system components – to build fault-tolerant applications.

Generally, a parallel application using the current PVM 3 is split into a master process and several slave processes. While the slaves perform the actual work of the task, the master distributes data and sub-tasks to the individual slave processes. Finally, the master synchronises with all slaves at a barrier, which marks the end of the parallel processing. Before starting the parallel sessions, all designated machines of the cluster need to be announced in a host-file. Furthermore, PVM demons must run on these machines. These PVM demons (virtual machines) are shut down, once the parallel sessions are completed. After the initialisation, the master starts its execution by logging on to the running parallel virtual machine (PVM demon). There-after, it determines the available hardware configuration (number of available machines (nodes), ...), allocates the name space for the slaves, and starts these slaves by assigning a sub-task (program executable). After checking if all slaves are started properly, data is distributed (and sometimes collected) to the slaves. At the end of the parallel computation, results are collected from the slaves. After a final synchronisation at a common barrier, all slaves and the master log off from the virtual machine.

3.3.2. Threads

A more recent parallel programming paradigm is the thread model. A thread is a control flow entity in a process. Typically, a sequential process consists of one thread; more than one thread enables a concurrent (parallel) control flow. While the process provides the environment for one or more threads – creating a common address space, a synchronisation and execution context – the individual threads only build a private stack and program counters. The different threads of a single process communicate via synchronisation mechanisms and via the shared memory.

In contrast to message-passing, threading is only possible on multiprocessor systems. Moreover, multiprocessor systems need a shared memory architecture, in order to provide the same virtual address space. Basically, there are three different kinds of implementations for threads. There is a user thread model, a kernel thread model, and a mixed model. The user thread model is usually a very early implementation of a thread package. All thread management is handled by the thread library; the UNIX kernel only knows the process, which might contain more than one thread. This results in the situation that only one thread of a process is executed at any particular time. If you are using threads on a single processor workstation, or your threads are not compute-bound, this is not a problem. However, on a multiprocessor system, we do not really get a concurrent execution of multiple threads of one process. On the other hand, this implementation model does not require a modification of the operating system kernel. Furthermore, the management of the threads does not require any kernel overhead. In Pthread terminology, this model is called all-to-one-scheduling. In contrast to user threads, each kernel thread is known to the operating system kernel. Consequently, each kernel thread is individually schedulable. This results in a real concurrent execution on a multiprocessor, which is especially important for compute-bound threads. However, allocation and management of a kernel thread can introduce significant overhead to the kernel, which eventually might lead to a bad scaling behaviour. Pthread terminology denotes this model to be one-to-one-scheduling. As usual, the best solution is probably a mixed model of user and kernel threads. The threads are first scheduled by the thread library (user thread scheduling). Thereafter, the threads scheduled by the library are scheduled as kernel threads. Threads that are not compute-bound (i.e., performing I/O) are pre-empted by the scheduling mechanism of the library, while only compute-bound threads are scheduled by the kernel, thus enabling high-performance concurrent execution. In Pthread terminology, this model is called the many-to-one or some-to-one scheduling.

To summarise, the main advantages of threads over message-passing is the fast communication and data exchange using the shared memory – no messages need to be explicitly send to other execution entities – and the cheap/fast context switch between different threads of one process. This is due to the shared address space, which is not changed during a thread switch. These features can be used to control the concurrent flow of the job at a much tighter level as which message-passing. On the other side, the use of threads is limited to (virtually) shared memory systems.

3.4. Communication Infrastructure

There are basically two networking technologies that can be considered for clusters of commodity workstations: Ethernet and Myrinet (actually it is not yet a commodity network).

3.4.1. Switched-Ethernet

Today, Ethernet is used primarily as an information network. However, there is a strong belief that some recent technological advances will enable its use in dependable applications with real-time requirements. One of the most common arguments that has been traditionally put forward against its use in real-time applications is that Ethernet has a non-deterministic access delay, leading to an unpredictable timing behaviour of the supported applications.

Recently, there have been promising technological breakthroughs [30]. Ethernet switches provide flexible and scalable solutions through the use of micro-segmentation and full-duplex operation (leading to a collision-free environment). Simultaneously, IEEE 802.1p (ratified in September 1998) gives Layer 2 switches the ability to prioritise Ethernet traffic and the IEEE Std 802.3x Flow Control provides means to control the generated traffic. All these mechanisms may be exploited to improve the timing characteristics of Ethernet networks.

3.4.2. Myrinet

Myrinet is also a recent technology of packet-communication and switching, and is already widely used to interconnect clusters of workstations, PCs, servers, or single-board computers.

Myrinet as the following characteristics [31]: full-duplex 2+2 Gigabit/second links, switch ports, and interface ports; flow control, error control, and "heartbeat" continuity monitoring on every link; low-latency, cut-through, crossbar switches, with monitoring for high-availability applications; can scale to tens of thousands of hosts, with network-bisection data rates in Terabits per second, and can also provide alternative communication paths between hosts; host interfaces that execute a control program to interact directly with host processes ("OS bypass") for low-latency communication, and directly with the network to send, receive, and buffer packets.

Therefore, Myrinet has also the potential to provide clusters with high performance and high availability.

3.5. Research/Innovation Perspectives for Walkabout

Optimal parallelisation of rendering must be addressed in Walkabout. One important topic of research must be on choosing the appropriate parallelism to be implemented for the rendering pipeline. Multiprocessor scheduling approaches need also to be considered, since the resources will be shared by to perform rendering services concerning multiple clients. The cluster-related network infrastructure will be thoroughly investigated in order to obtain tight pre-run time schedulability worst-case response times, since in workstation clusters, the disk and network bandwidth are the most important parameters influencing the rendering performance. Switched-Ethernet and Myrinet must be target communication infrastructures. Parallel programming must use state-of-the message-passing standards. Validation of MPI and/or PVM will be performed within the Walkabout project. RT-Linux will be considered.

4. Walkabout Interactive Remote Rendering

Various systems provide a visual presentation of a small part of a model that is complex and voluminous. Virtual walkthrough, flight simulation, visualisation systems are only a few common examples to this type of systems. Commonly, such systems face the task of generating images from a model containing millions of elements [23]. With technologies such as 3D scanning, graphical models are becoming increasingly complex, and the user appetite will never be satisfied with the computational power available. Models keep getting larger and more complex, and the situation is likely to worsen in the near future.

Internet-based virtual environments let users explore multiple virtual worlds with many different geometric models, while at the same time it may enable new services over the Internet.

4.1. Basic Client-Server Rendering Systems

A basic rendering system may have three building blocks: the pre-processing, the client-server program and the rendering engine, as shown in Fig. 1.

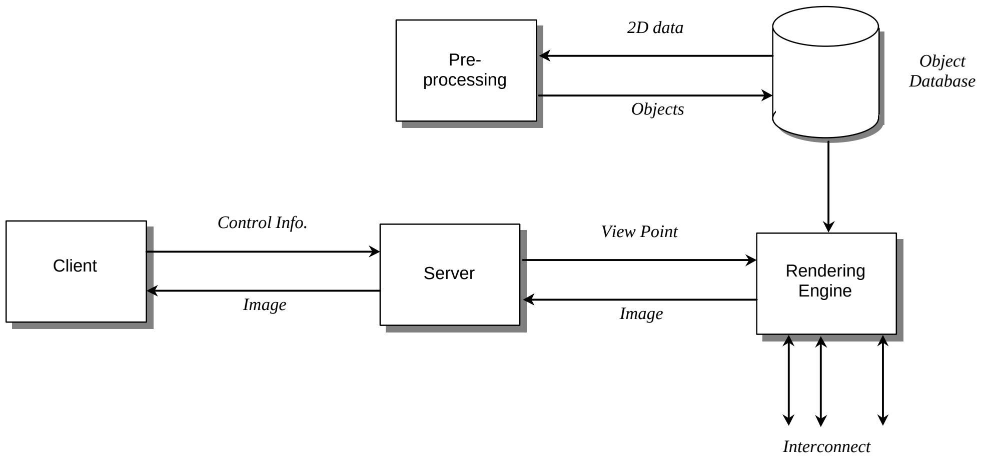

Figure 1

The client-server program provides the user interface for easy access to the object information through some controls and a display. The use of a flexible user interface building system enables the rendering system to be adapted for different applications. In addition, the interface is also responsible for maintaining the quality of the presentation of the requested information as dictated by the type of application. After the user interface has collected the user's inputs, a rendering engine is used to convert the object-space into images according to the input parameters. To accommodate the wide range of client applications, the capabilities of the rendering engine, such as rendering speed and techniques, must be maximised and efficient. In order to have speedy rendering of the data, the rendering engine must be able to access the computer graphics objects efficiently. The object database provides the storage facility for fast retrieval of the needed data in a format that minimises the start-up time of the rendering pipeline.

The pre-processing can be used not only to repackage data to make it more modular, but also to capitalise on the viewing characteristics to reduce the runtime rendering work.

In this design, the server and the rendering engine are separate processes. Hence, the inner workings of the rendering engine are isolated from the server. As a result, the implementation of the rendering engine is flexible.

A parallel computer is used not only because of the enormous size of the object data, but also for exploiting the parallelism of the model data and the rendering algorithm to speed up the computation required.

The client process can also provide an application context for the rendering engine. Possible tasks that can be handled from the client applications include, but are not limited to, walking trough. The application should also support interface to control the position and the view from the user. The client application can compute the next viewpoint by combining the inputs from the user controls with the previous co-ordinates and the application specific navigation algorithms. After all needed information is gathered, the visualisation data can be sent to the server using TCP/IP. When the rendering of the image is completed, the client receives the resulting image using a similar TCP/IP channel. Depending on the size of the data and the format of the image transferred either a session wide TCP connection or UDP datagrams can be used. The first handles session and error control automatically, the later is much faster albeit without no error correction or out-of-order compensation. When receiving the image data the client must place it in a virtual frame buffer that when filled up will replace the current frame. For an efficient implementation of this client solution a multi-threaded environment must be used to support the following tasks: 1 - computing next frame view info and send information to the server; 2 - receive and presenting the current frame. The rendering process for the rendering engine can be divided into three stages: 1 - receive viewpoint from client; 2 - render the image; 3 - send image to client.

The communication and the rendering work can proceed in parallel for rendering multiple images. An efficient rendering process needs two threads, one to communicate with the client and the other for rendering. If a rendering cluster is used then an Internet front-end server should be used to interface the protected cluster network to the public Internet. This front-end server can also be used for the cluster management. This three-tier design of the client-server solution provides the rendering engine with a programming and operating environment that is independent from the specifics of the application. As a result, the efficiency and portability of the rendering engine is increased.

In the interactive Walkabout we can take advantage of several specific constrains of the target applications in order to find the most efficient algorithm. One of the tasks of the project is to find the best compromise on flexibility vs. performance on the server side.

The Walkabout server must do its work at interactive rates and enable the integration on the final scene of not only video and audio streams from the Walkabout multimedia database, but also virtual characters with synthetic behaviour processed on the server side. To enable the use of thin clients (like DBTV receivers) the rendering and mixing of these 3D data sources should be done entirely on the server side, however more flexible clients like PWS can support more demanding tasks from the client up to the full rendering of the world (see Section 4.2.2).

In the thin-client scenario, the server will deliver a video stream (probably in standard MPEG 4 format) to the client. On the PWS client scenario in order to optimise the network utilisation the data format will have to be application dependent. However it is clear that in order to be able to efficiently render the 3D world a setup phase will be needed to download the base world or character models and other application specific data. After this phase the system will enter interactive mode and only updates will be sent to the client.

4.2. Web-Based Client-Server Rendering Systems

The basic client-server rendering system can be extended to a web-based client-server rendering systems. Two possibilities can be explored: server-based rendering (Fig. 2) and client-based rendering (Fig. 3). The first can also be denoted as the image-based server while the second can be denoted as VRML-based server [7].

Although the resulting hardware organisation of the two systems is similar, the functional organisation of the two client-server systems is different because of the different philosophy of the projected technology trends. The architecture for a server-based rendering system assumes the client has very limited rendering capability and storage capacity, but it is connected by a QoS network to a remote computer, which stores and renders the graphics objects into images on the client's behalf. Since the computing resources at the server side are finite, this set-up can only handle a fixed number of clients at a time. Additionally, the information provider retains the control of the rendering engine. Hence, the provider can improve the rendering engine by using faster hardware and a better rendering algorithm without having to redistribute the client software. Also, computer resources can easily be added by taking advantage of the scalability of the cluster.

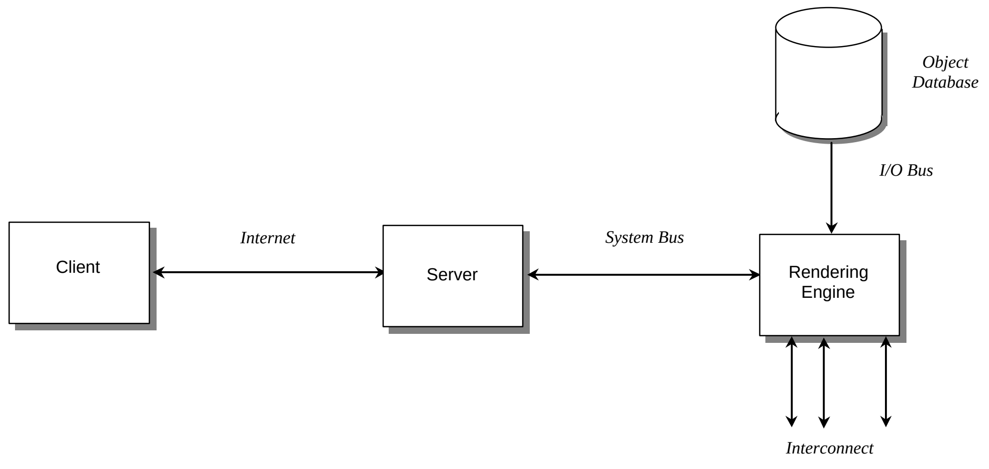

Figure 2

The client-based (VRML-based) system in Fig. 3 assumes that the client has a capable rendering engine and a high speed connection to WWW servers, where the client can access many different VRML (Virtual Reality Modelling Language) [32] worlds, including the particular Walkabout VRML. Because the rendering is done on the client side, the total amount of computing resource of the entire system increases proportional to the number of clients. The VRML-based server of this system can handle larger numbers of clients simultaneously because the computation of serving VRML is minimal, but the number of connections is limited by the required I/O and network throughput of all the users and the capability of the web server. On the other hand, the capabilities of different clients may vary, so the amount of information packaged in the VRML file is limited by an arbitrary "minimal client capability" set by the information provider. An improvement can be provided by allowing the client to dynamically select the level of detail (LOD) wanted, although this increases the needed computing resources per client at the server side to construct the VRML file dynamically at the requested resolution from a database, instead of serving a single file at a single resolution. Since both systems must use their server to augment the clients' ability to store and to manipulate the 3D data, high performance servers with lots of storage space, fast I/O and network are needed to handle the workload.

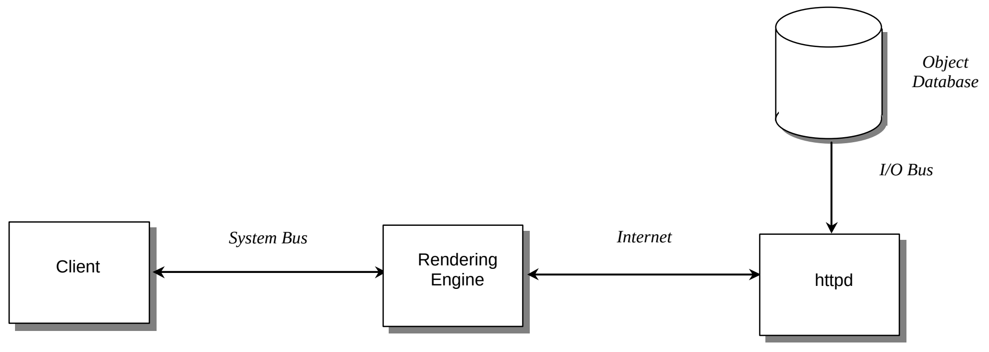

Figure 3

4.2.1. Comparison between These Two Models

Client

The clients have different requirements for their software and hardware components because of the differences in the design assumptions.

In the server-based rendering approach, the client program is responsible for acquiring the user's input and presenting the images to the user as they arrived. The client connects to the image server through a computer network using a communication protocol that can be built on top of TCP/IP. Standard MPEG 4 can be used for the server to client data coding or the more exotic Iterative Fractal Compression (IFS) [33] could be used instead. Because the client only needs to manipulate images, most client computers can easily satisfy the required computer resources, such as memory and CPU power.

The client-based rendering approach, on the other hand, can be a high performance VRML browser, which has a number of built-in user interfaces and a rendering engine. In VRML 2.0, the creator of a world can also customise the user interface. After the user interface has collected the user's inputs, the built-in rendering engine renders the output according to the input parameters. The communication between the user interface and the rendering engine is done through the system bus, which has much higher bandwidth than any existing computer network. Because VRML is a WWW standard for delivering information in 3D computer graphics, a VRML browser is part of the standard toolset for most web users. As a result, the information provider has no need to create, maintain and distribute any client and/or server software. Furthermore, the VRML encoded information is also computer architecture independent. This eliminates any need to do real-time conversion of the data to a format that the client can understand. Therefore, the server can serve more clients with the same resources as compared to an image-based system.

Before the user can see the virtual world, the client computer must receive and process the entire VRML world. The amount of memory and rendering resources needed is directly proportional to the complexity of the world. If the entire VRML world is completely defined by using predefined geometrical shapes, the browser can automatically adjust the amount of detail of the various objects to match the available rendering resource. In general, the VRML world designer must make a decision on how much detail is included in the world. Since the VRML world is intended to operate in isolation, the designer has no knowledge of, and no control over, the rendering capability of the client computer. Consequently, bounds on the amount of detail are set, such that the VRML audience will not lose interest because of slow rendering of a complex world (upper bound) or the minimal information content of simplistic world (lower bound). This limitation can be overcome by allowing the user to select the complexity of the VRML world at runtime to match the capability of available computing resource, or it could be done dynamically at run-time by the user specifying a particular frame rate. The Java capabilities of VRML 2.0 enable the user to adjust the content during the presentation. However, this increases the required server computing resource per client to interpret the inputs to select the needed data and to encode the information before transmitting to the client.

Network

Both rendering systems use the network as an integral part to acquire the world information. Because of the design differences, the two systems have distinct communication pattern. Hence, the effects of the network latency and bandwidth are different. The server-based rendering approach system maintains a continuous and regular TCP/IP data flow between the client and the server while the application is running. This network connection is used to transfer a sequence of images, which contains the information requested by the user. As a result, the user only experiences the latency of starting the terrain rendering application, which is incurred by the network protocol, once. Furthermore, these images must be displayed in the order in which they are received, that is guaranteed if the TCP is used. Another form of network delay is caused by the finite bandwidth of the computer network. In order for the user to see the images, the network has to transfer this amount of information from the server to the client. Considering an average size of 30kb for a general medium-quality compressed 640x480 JPEG image a dedicated 10Mbps Ethernet network can support more than 30 fps, a 640kb ADSL modem about 3 fps, and a more down-to-earth 64kb ISDN channel would take up almost 4 seconds to receive a single frame. Using MPEG 4 compression could have a theoretical huge impact on the network performance but not only the processing power needed for compression but also the need to bufferize several frames for efficient compression limits the usable compression in our interactive system.

To maximise the number of web clients that a web server can serve simultaneously, VRML browsers are designed to contact the server only when they need information. Before retrieving the wanted information, the VRML-based system establishes a temporary connection to the server. Because the VRML browser processes the entire VRML file before rendering its content, the user must endure in addition to the processing delay, the full network latency of setting-up the transient connection and transferring the VRML file. Eliminating unwanted details and compressing the data file before transfer can minimise these delays

Server

The differences of the design philosophy of the two systems have influence in the functionality of the servers. For the server-based rendering system, the server provides the client a gateway to the parallel rendering engine by converting the viewing information into a view transformation matrix and dispatching the resulting images to the client through the dedicated communication channel. The parallel rendering engine, which can be composed of a number of sequential rendering pipelines connected by a high-speed network, use the view transformation matrix to render 3D model data concurrently onto image buffers. The high-speed network, which connects all the computation nodes, can be used not only to broadcast the viewing transformation matrix, but to assemble the final image. The 3D-model database coexists on the same computation nodes as the sequential rendering pipelines. The overhead of accessing the 3D model data involves finding all visible objects on the computing node and passing their resident memory location to the local sequential rendering engine for rendering.

The job of a VRML-like model server is to furnish the client with requested geometric data. This is accomplished by processing the user request, performing view specific model optimisation like LOD and encoding the optimised 3D-model data. To enable dynamic adjustment of the 3D-model detail, the model information is stored in a server side database. Finally, the reconstructed 3D-model is encoded in VRML before shipping to the client. This VRML-based model rendering system can serve a large number of clients, and the server must have fast access to the model data. This requires the server machine to have excellent I/O and network bandwidth. However, the I/O and network bandwidth of a site is finite.

4.3. Hybrid Web-Based Client-Server Rendering Systems

As the popularity of Internet grows, a popular virtual street VRML web site can easily be overwhelmed by enthusiastic responses from the web users. To relieve this congestion, computing resources of the server must increase at the same rate as the number of clients increases. Furthermore, many of the new web clients will be network computers (NCs) or PDAs, with minimal hardware capabilities. As a result, a new type of server is needed to serve computing capabilities to the low-end clients. Using a hybrid system (Fig. 4) with a mix of web and rendering servers not only allows low-end web clients, such as web appliance and network computer, to access the virtual world, but also it also provides the additional computing resources to increase the I/O and network bandwidth of the system. This system leverages the maturity of web server software and operating model by storing the Walkabout 3D models in a database on a web server with large storage capacity, high I/O and network performance. High performance clients and rendering servers are able to connect to the Walkabout 3D data web server to retrieve the VRML files. The VRML world is rendered on screen by a high performance client. For the low-end web clients, the VRML world can be rendered by a rendering server. The resulting images are compressed and shipped to the clients. The rendering server must have high performance multimedia support and 3D rendering capabilities to render the 3D models data quickly according to the viewing parameter, the requested rendering quality and the rendering instruction in the VRML file. If a high performance client takes delivery from a rendering server, then the rendering server will act as the mirror site of the Walkabout web site making the data stream much more stable.

The difference between a web server and a rendering server is that the rendering server knows more about its client than a web server because it is dedicated to serve a group of clients. As a result, the requested data is automatically adjusted to the need of the presentation. This arrangement enables more efficient use of network and computing resources at organisations (Universities, Corporations, ISP, etc.).

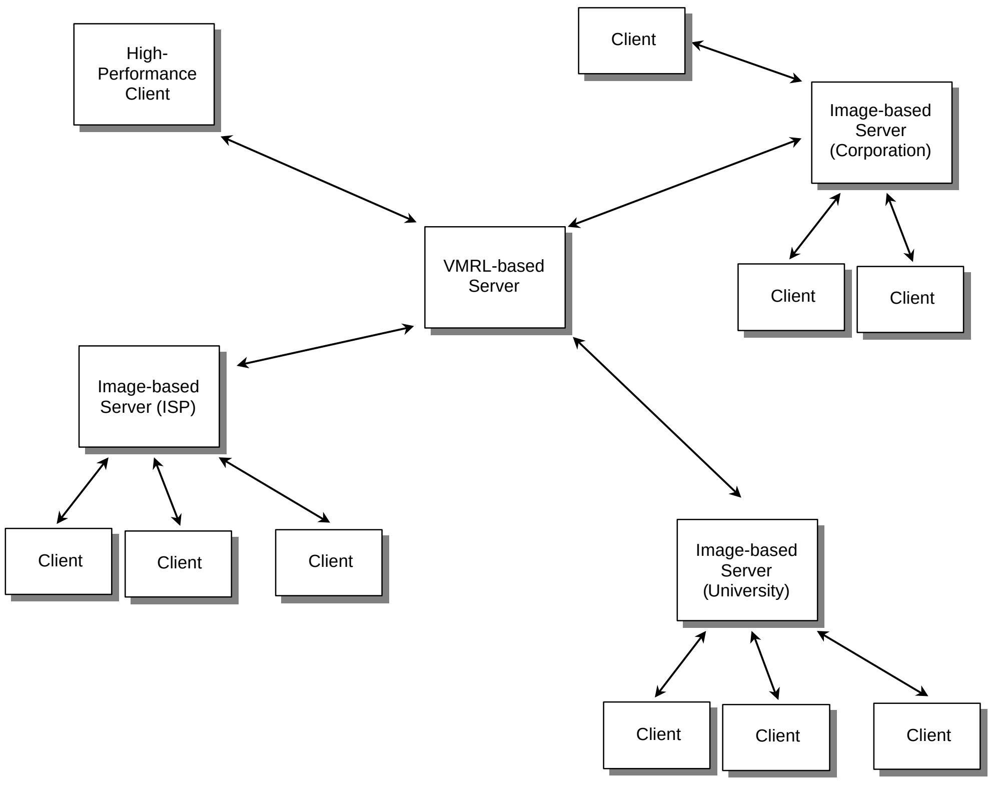

Figure 4

The image-based and VRML-based systems each provide unique advantages and disadvantages. Because the two systems have a complementary set of design compromises, a hybrid of the image-based and VRML-based system can offer the best features of the two systems.

Walkabout must also integrate properly the Media server and Behaviour server.

5. References

[1] Loh, D., Holtfrerich, D., Choo, Y., and Power, J. (1992). *Techniques for Incorporating Visualisation in Environmental Assessment: and object oriented perspective*. Landscape and Urban Planning, 21, pp. 796-804.

[2] Faust, N. (1995). *The Virtual Reality of GIS*. Environmnet and Planning B: Planning and Design, 22, pp. 257-268.

[3] Crockett, T. (1995). *Parallel Rendering*. Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, ICASE Report No. 95-31.

[4] Enzmann, A., Kretzschzmar, L. and Young, C. (1994). *Ray Tracing Worlds with POV-RAY*. Waite Group Press.

[5] Wilt, N. (1994). *Object-Oriented Ray Tracing in C++*. Wiley Professional Computing.

[6] Greenberg, D. (1991). *Computer and Architecture*. Scientific American, February, pp. 104-109.

[7] Leung, A. (1997). *Real-Time Interactive Client-Server Terrain Rendering*. PhD Thesis, Syracuse University, New York, USA.

[8] Kaufman, A., Cohen, D. and Yagel, R. (1993). *Volumetric Graphics*. *IEEE Computer*, 26 (7), pp. 51-64.

[9] Yagel, R. (1996). *Towards Real-Time Volume Rendering*. Proceedings of GRAPHICON’96, Vol. 1, pp. 230-241.

[10] Oliveira, M. and Bishop, G. (1999). *Image-Based Objects*. Proceedings of 1999 ACM Symposium on Interactive 3D Graphics, pp. 191-198.

[11] Chang, C., Bishop, G. and Lastra, A. (1999). *LDI Tree: a Hierarchical Representation for Image-Based Rendering*. Proceedings of SIGGRAPH’99.

[12] Mark, W. and Bishop, G. (1998). *Efficient Reconstruction Techniques for Post-Rendering 3D Image Warping*. UNC Computer Science Technical Report TR98-011, University of North Carolina, USA.

[13] Foley, J., Dam, A., Feiner, S. and Hughes, J. (1992). *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd Edition.

[14] Fox, G., Williams, R. and Messina, P. (1994). *Parallel Computing Works!*. Morgan Kaufmann.

[15] Li, P., William, D. and Curkendall, D. (1996). *RIVA: A Versatile Parallel Rendering System for Interactive Scientific Visualisation*. IEEE Transactions on Visualisation and Computer Graphics, 2(3), pp. 186-201.

[16] Crockett, T. (1996). *Beyond the Renderer: Software Architecture for Parallel Graphics and Visualisation*. Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, ICASE Report No. 96-75.

[17] Schneider, B. (1998). *Parallel Rendering on PC Workstations*. Proceedings of 1998 International Conference on Parallel and Distributed Processing Techniques and Applications.

[18] Whitman, S. (1994). *Dynamic Load Balancing for Parallel Polygon Rendering*. IEEE Computer Graphics & Applications, pp. 41–48.

[19] Boden, N. et al.(1995). *Myrinet: A gigabit-per-Second Local-Area Network*. IEEE Micro, pp. 29–36.

[20] Hubbell, J. (1996). *Network rendering*. Autodesk University Sourcebook Vol. 2, pp. 443–453. Miller Freeman.

[21] Geist, A., Beguelin, A., Dongarra, J., Jian, W., Machek, R. and Sunderam, V. (1994). *PVM: Parallel Virtual Machine*. MIT Press.

[22] MPI Forum (1997). *MPI-2: Extensions to the Message-Passing Interface*. Technical Report MPI 7/18/97, Message-Passing Interface Forum.

[23] Yagel, R. and Ray, W. (1996). *Visibility Computation of Efficient Walkthrough of Complex Environments*. Presence, Vol. 5, No. 1, pp. 1-16.

[24] Clark, J. (1976). *Hierarchical geometric models for visible surface algorithms*. Communications of the ACM, 19(10), pp. 547–554.

[25] Hesina, G. and Schmalstieg, D. (1998). *A Network Architecture for Remote Rendering*. Proceedings of Second International Workshop on Distributed Interactive Simulation and Real-Time Applications, pp. 88-91.

[26] Funkhouser, T. and S´ equin, C. (1993). *Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments*. In SIGGRAPH ’93, pp. 247–254.

[27] Gobbetti, E. and Bouvier, E. (1999). *Time-critical Multiresolution Scene Rendering*. IEEE Visualization '99.

[28] Klosowski, J. and Silva, C. (1999). Rendering on a Budget: a Framework for Time-Critical Rendering. IEEE Visualisation'99, pp. 115-122.

[29] Rusinkiewicz, S. and Levoy, M. (2000). *QSplat: A Multiresolution Point Rendering System for Large Meshes*. Proceedings of SIGGRAPH'2000.

[30] The CIDER Project: http://www.hurray.isep.ipp.pt/cider

[31] The Myrinet Homepage: http://www.myri.com/myrinet

[32] Pesce, M. (1995). *VRML: Browsing and Building Cyberspace*. New Riders.

[33] Fractal Image Encoding: http://inls.ucsd.edu/y/Fractals/

[34] G. Abram and H. Fuchs. Vlsi architectures for computer graphics. In G. Enderle, editor, Advances in Computer Graphics I, pages 6–21, Berlin, Heidelberg, New York, Tokyo, 1986. Springer-Verlag.

[35] J. Montrym. Infinite reality: A real-time graphics system. In Computer Graphics (Proc. Siggraph), pages 293–302, August 1997.

[36] B.-O. Schneider. A processor for an object-oriented renderin system. Computer Graphics Forum, 7:301–310, 1988.

[37] M. Deering et al. The triangle processor and normal vector shader: A vlsi system for high-performance graphics. *Computer Graphics (Proc. Siggraph),* 12(2):21–30, August 1988.

[38] S. Molnar et al. A sorting classification of parallel rendering. *IEEE Computer Graphics & Applications*, pages 23–32, July 1994.

[39] M. Deering and S.R. Nelson. Leo: A system for cost effective 3d shaded graphics. In *Computer Graphics (Proc. Siggraph),* pages 101–108, August 1993.

[40] K. Akeley. Realityengine graphics. In *Computer Graphics (Proc. Siggraph)*, pages 109–116, August 1993.

[41] D. Ellsworth. A new algorithm for interactive graphics on multicomputers. *IEEE Computer Graphics & Applications*, pages 33–40, July 1994.

[42] J. Eyles et al. Pixelflow: The realization. In *Proc. 1997 Siggraph/Eurographic Workshop on Graphics Hardware*, pages 57–68, New York, 1997. ACM Press.

[43] C. Mueller. The sort-first rendering architecture for high-performance graphics. In *Proc. 1995 Symposium on Interactive 3D Graphics*, pages 75–84, New York, 1995. ACM Press.

[44] A. Barkans et al. Guardband clipping method and apparatus for 3d graphics display system. U.S. Patent 4,888,712. Issued Dec 19, 1989.

[45] M. Cox and P. Hanrahan. Depth complexity in object-parallel graphics architectures. In *Proc. 7th Eurographics Workshop on Graphics Hardware*, pages 204–222, Cambridge (UK), 1992.

[46] H. Fuchs et al. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics (Proc. Siggraph)*, 23(3):79–88, July 1989.

[47] M. Hu and J. Foley. Parallel processing approaches to hidden-surface removal in image space. *Computers & Graphics*, 9(3):303–317, 1985.

[48] S. Whitman. Dynamic load balancing for parallel polygon rendering. *IEEE Computer Graphics & Applications*, pages 41–48, July 1994.

Appendix A - Details on Parallel Polygon-Based Rendering

A.1. Single-frame vs. Multi-frame Rendering

Rendering polygonal models can be driven by several needs. If the model is only used once for the generation of a still image, the entire rendering process outlined in Section 2 has to be performed. The creation of animation sequences requires rendering of the same model for different values of time and consequently varying values for time-dependent rendering parameters, e.g. view position, object location, or light source intensities. Even though multi-frame rendering could be handled as repeated single-frame rendering, it offers the opportunity to exploit inter-frame coherence. For example, access to the scene database can be amortised over several frames and only the actual rendering steps (geometry processing and rasterisation) must be performed for every frame. Other ways to take advantage of inter-frame coherence will be discussed below.

A.2. Available Algorithm Classes

For many years the classification of parallel rendering algorithms and architectures has proven to be an elusive goal. We will discuss several such classifications to gain some insight into the design space and possible solutions.

Pipelining vs. Parallelism

Irrespective of the problem domain, parallelisation strategies can be distinguished by *how* the problem is mapped onto the parallel processors.

For pipelining the problem is decomposed into individual steps that are mapped onto processors. This method fits naturally in the rendering pipeline. However, the existence of a limited number of pipeline function blocks restricts the amount of parallelism. And since the different functions in the pipeline vary in computational complexity the load-balancing can be difficult. To overcome such constraints pipelining is often improved by replicating some or all pipeline stages. Data is distributed amongst those processors and can be handled in parallel.

In interactive systems pipelining is not an interesting solution since the time it takes from the pipeline object space input to an image being produced is equal (or even greater due to parallel overhead) than a standard non-parallel rendering pipeline.

Object Partitioning vs. Image Partitioning

One of the earliest attempts at classifying partitioning strategies for parallel rendering algorithms took into consideration whether the data objects distributed amongst parallel processors belonged into object space, e.g. polygons, edges, or vertices, or into image space, i.e. collections of pixels such as portions of the screen, scan lines or individual pixels [34]. Object-space partitioning is commonly used for the geometry processing portion of the rendering pipeline, as its operation is intrinsically based on objects.

Most parallelisation strategies for rasterisers employ image-space partitioning [35]. A few architectures apply object-space partitioning in the rasteriser [36, 37].

Sorting Classification

Based on the observation that rendering can be viewed as a sorting process of objects into pixels, different parallel rendering algorithms can be distinguished by where in the rendering pipeline the sorting occurs. Considering the two main steps in rendering, i.e. geometry processing and rasterisation, there are three principal locations for the sorting step: Early during geometry processing (sort-first), between geometry processing and rasterisation (sort-middle), and after rasterisation (sort-last). Sort-middle architectures form the most natural implementation of the rendering pipeline. Many parallel rendering systems, both software and hardware, use this approach, e.g. [39,40]. They assign primitives to geometry processors that implement the entire geometry pipeline. The transformed primitives are then sent to rasterisers that are each serving a portion of the entire screen. One drawback is the poor load-balancing among the rasterisers due to the, unfortunately common, uneven distribution of objects across the screen. Another problem of this approach, in parallel implementations, is the redistribution of primitives after the geometry stage that requires a many-to-many communication between the processors. A hierarchical multi-step method to reduce the complexity of this global sort is described in [41].

Sort-last assigns primitives to renderers that generate a full-screen image of all assigned primitives. After all primitives have been processed, the resulting images are merged/composited into the final image. Since all processors handle all pixels this approach offers good load-balancing properties. However compositing the pixels of the partial images consumes large amounts of bandwidth and requires support by dedicated hardware, e.g. [42]. Further, with sort-last implementations it is difficult to support anti-aliasing, as objects covering the same pixel may be handled by different processors and will only meet during the final compositing step.

Sort-first architectures quickly determine for each primitive to which screen region(s) it will contribute. The primitive is then assigned to those renderers that are responsible for those screen regions. Currently, no actual rendering systems are based on this approach even though there are some indications that it may prove advantageous for large models and high-resolution images [43]. [43] claims that sort-first has to redistribute fewer objects between frames than sort-middle. Similar to sort-middle, it is prone to suffer from load-imbalances unless the workload is levelled using an adaptive scheme that resizes the screen regions each processor is responsible for.

A.3. Load Balancing Definitions

As with the parallel implementation of any algorithm the performance depends critically on balancing the load between the parallel processors. There are workload-related and design-related factors affecting the load balancing. Before the discussion of load balancing strategies, we will define terms used throughout the rest of the discussion.

Tasks are the basic units of work that can be assigned to a processor, e.g. objects, primitives, scan lines, pixels etc.

Granularity quantifies the minimum number of tasks that are assigned to a processor, e.g. 10 scan lines per processor or 128x128 pixel regions.

Coherence describes the similarity between neighbouring elements like consecutive frames or neighbouring scan lines. Coherence is exploited frequently in incremental calculations, e.g. during scan conversion.

Parallelization may destroy coherence, if neighbouring elements are distributed to different processors.

Load balance describes how well tasks are distributed across different processor with respect to keeping all processors busy for all (or most) of the time. Surprisingly, there is no commonly agreed upon definition of load balance in the literature.

Workload Characterisation

Several properties of the model are important for analysing performance and load-balancing of a given parallel rendering architecture. Clipping and object tessellation affect load-balancing during geometry processing, while spatial object distribution and primitive size mostly affect the balance amongst parallel rasterisers.

Clipping. Objects clipped by the 2D image boundaries incur more work than objects that are trivially accepted or rejected. It is difficult to predict whether an object will be clipped and load-imbalances can result as a consequence of one processor receiving a disproportionate number of objects requiring clipping. There are techniques that can reduce the number of objects that require clipping by enabling rasterisers to deal with objects outside of the view frustum [44]. This reduces the adverse affects of clipping on load-balancing to negligible amounts.

Tessellation. Some rendering APIs use higher order primitives, like NURBS, that are tessellated by the rendering subsystem. The degree of tessellation, i.e. the number of triangles per object, determines the amount of data expansion occurring during the rendering process. The degree of tessellation is often view-dependent and hence hard to predict a priori. The variable degree of tessellation leads to load imbalances as one processor’s objects may expand into more primitives than objects handled by another processor. Tessellation also affects how many objects need to be considered during the sorting step. In sort-first architectures, primitives are sorted before the tessellation, thus saving communication bandwidth compared to sort-middle architectures.

Primitive distribution. In systems using image-space partitioning, the spatial distribution of objects across the screen decides how many objects each processor must process. Usually, objects are not distributed uniformly, e.g. more objects may be located in the centre of the screen than along the periphery. This creates potential imbalances in the amount of work assigned to each processor.

Primitive size. The performance of most rasterisation algorithms increases for smaller primitives. The mix of large and small primitives therefore determines the workload for the rasteriser. Several experiments have shown (see e.g. [45]) that many scenes contain a large number of small objects and a few large objects. The primitive size also affects the overlap factor, i.e. the number of screen regions affected by an object. The overlap factor affects the performance of image-space partitioning schemes like sort-first and sort-middle algorithms.

A.4. Load Balancing Solutions

Several design techniques are used to compensate for load-imbalances incurred by different workloads. They can be distinguished as *static*, *dynamic* and *adaptive*.

Static load balancing uses a fixed assignment of tasks to processors. Although, a low (i.e. no) overhead is incurred for determining this assignment, load imbalances can occur if the duration of tasks is variable. Dynamic load balancing techniques determine the on the fly which processor will receive the next task. Adaptive load balancing determines an assignment of tasks to processors based on estimated cost for each task, thereby trying to assign equal workload to each processor.

On-demand assignment

This is a dynamic method that relies on the fact that there are many more tasks (objects or pixels) than there are processors. New work is assigned to the first available, idle processor. Except during initialisation and for the last few tasks, every processor will be busy all the time. The ratio of the number of tasks and the number of processors is called the *granularity ratio*. Selecting the granularity ratio requires a compromise between good load balancing (high granularity ratio) and overhead for instance due to large overlap factor (low granularity ratio). The optimal granularity ratio depends on the model, typical values range from about 4 to 32. An example for dynamic load-balancing through the use of on-demand assignment of tasks is the Pixel-planes 5 system [46]. In Pixel-planes 5, the tasks are 80 128x128 pixel regions that are assigned to the next available rasteriser module.

Interleaving

Interleaving is a static technique that is frequently used in rasterisers to decrease the sensitivity to uneven spatial object distributions. In general, the screen is subdivided into regions, e.g. pixels, scan lines, sets of scan lines, sets of pixel columns, or rectangular blocks. Among *n* processors, each processor is responsible for every *n*-th screen region. The value *n* is known as the interleave factor. Since clustering of objects usually occurs in larger screen regions and since every object typically covers several pixels, this technique will eliminate most load-imbalances stemming from non-uniform distribution of objects. Interleaving makes it harder to exploit spatial coherence as neighbouring pixels (or scan lines) are assigned to different processors. Therefore, the interleave factor, i.e. the distance between pixels/scan lines assigned to the same processor, must be chosen carefully. Several groups have explored various aspects of interleaving for parallel rasterisation, e.g. [47].

Adaptive scheduling

This technique tries to achieve balanced loading of all processors by assigning different number of tasks depending on task size. For geometry processing this might mean to assign fewer objects to processors that are receiving objects that will be tessellated very finely. In image-space schemes this means that processors are assigned smaller pixel sets in regions with many objects, thus equalising the number of objects assigned to each processor.

Adaptive scheduling can be performed either dynamically or statically. Dynamic adaptation is achieved by monitoring the load-balance and if necessary splitting tasks to off-load busy processors. Such a scheme is described in [48]: Screen regions are initially assigned statically to processors. If the system becomes unbalanced, idle processors grab a share of the tasks of the busy processors.

Statically adaptive schemes attempt to statically assign rendering tasks such that the resulting work is distributed evenly amongst all processors. Such schemes are either predictive or reactive. Predictive schemes estimate the actual workload for the current frame based on certain model properties. Reactive schemes exploit inter-frame coherence and determine the partitioning of the next frame based on the workload for the current frame, e.g. [41].

Numerous rendering algorithms using adaptive load-balancing have been described. Most these methods operate in two steps: First, the workload is estimated by counting primitives per screen regions. Then, either the screen is subdivided to create regions with approximately equal workload or different number of fixed-sized regions (tasks) are assigned to the processors.

Frame-parallel rendering

This is a straightforward method to use parallel processors for rendering. Each processor works independently on one frame of an animation sequence. If there is little variation between consecutive frames, frames can be assigned statically to processors as all processor tend complete their respective frame(s) in approximately the same time. If processing time varies between frames, it is also possible to assign frames dynamically (on-demand assignment). In either case, the processors are working on independent frames and no communication between processors is required after the initial distribution of the model. Unfortunately, this approach is only viable for rendering of animation sequence. It is not suitable for interactive rendering as it typically introduces large latencies between the time a frame is issued by the application and when it appears on the screen.

The *application programming interface (API)* impacts how efficiently the strategies outlined above can be implemented. Immediate-mode APIs like OpenGL or Direct3D do not have access to the entire model and hence do not allow global optimizations. Retained-mode APIs like Phigs, Performer, OpenGL Optimizer, Java3D and Fahrenheit maintain an internal representation of the entire model which supports partitioning of the model for load-balancing.

A.5. Data Distribution and Scheduling

In distributed memory architectures, e.g. clusters of workstations or message-passing computers, object data must be sent explicitly to the processors. For small data sets, one can simply send the full data set to every processor and each processor is then instructed which objects to use. This approach fails however for large models either because there is not enough storage to replicate the model at every processor and/or the time to transfer the model is prohibitive due to the bandwidth limitations of the network.

Therefore, most implementations replicate only small data structures like graphics state, e.g. current transformation matrices, light source data, etc., and distribute the storage for large data structures, primarily the object descriptions and the frame buffer.

For system using static assignment of rendering tasks object data have to be distributed only during the initialisation phase of the algorithm. This makes it easy to partition the algorithm into separate phases that can be scheduled consecutively.

For dynamic schemes data must be distributed during the entire process. Therefore processors cannot continuously work rendering objects but must instead divide their available cycles between rendering and communicating with other processors. Such an implementation is described in [49]: The system implements a sort-middle architecture where each processor works concurrently on geometry processing and rasterisation, i.e. producing and consuming polygons. The advantage is that only a small amount of memory must be allocated for polygons to be transferred between processors. Determining the balance between polygon transformation (generation) and polygon rasterisation (consuming) is not obvious. However, [49] states that the overall system performance is fairly insensitive to that choice.