Creating Virtual Reality Applications on a Parallel Architecture

Shaun Bangay

Department of Computer Science
Rhodes University
Grahamstown, 6140
South Africa

Internet: cssb@cs.ru.ac.za

1. Introduction

Simulation of reality can be a computationally expensive process which can benefit from the use of parallel processing. However, no standards have yet been set in the design of a virtual reality application, let alone its parallelisation. This paper explores the design of one possible system intended to assist in the creation of virtual reality applications, and which runs on a parallel architecture. Where possible, design alternatives are pointed out and a comparison with systems which exploit the alternatives is given.

The design presented is heavily influenced by the considerations for parallel processing. The system presented here was created at the same time as a number of other systems. These other systems use varying degrees of parallelism. The characteristics and capabilites of these other systems is compared to the design presented here in [1].

The discussion is limited to the component of a virtual reality system responsible for the modelling of the virtual worlds, sometimes referred to as the kernel. It will be assumed that separate systems (device drivers) will be responsible for providing the stimuli for, and reading the responses from the users.

2. Modelling virtual worlds

The facilities which our virtual world kernel is expected to provide are summarised as follows:

- Be able to simulate multiple virtual worlds simultaneously.
- Allow multiple users to interact within each world.
- Use parallelism to reduce the load on individual processors
- Simplify implementation of a virtual reality application.
- Permit the modelling of various phenomena by allowing the properties (attributes) of the objects to interact with those of the world and the other objects.

These requirements fall into two classes. There are those that deal with the layout of the kernel and its support for multiple processors, worlds and users. The second set of requirements concerns the functionality of the kernel, and the services it will provide to support the development of virtual reality applications.

Different systems provide various levels of support for virtual reality applications. Some, like VEOS [2], are intended for prototyping parallel applications and provide only the facilities for parallel programming. Others, for example AVIARY [4][3], are dedicated to creating virtual reality applications and include mechanisms to automatically implement the laws of the various worlds and to enforce them. These levels of support are explored in detail in the next section.

3. Design of a Virtual World Kernel

The design of the kernel is shown as three stages. The first describes the manner in which worlds and objects are represented, and how they will be distributed. This is followed by a description of the functionality of the kernel. Finally the description of the communication system supporting interaction between worlds, objects and users is given. This last stage shows the manner in which the functionality of the kernel is implemented on the distributed system.

3.1. Modelling objects and worlds in parallel

Every system defines the terms ’world’, ’user’ and ’object’ differently. For the purposes of this

discussion an ’object’ will be the virtual analogue of a physical object cotaining all the attributes of the physical object being modelled. A ’world’ is a logical grouping of objects and a ’user’ is the representation of a human inhabiting a virtual world.

Each world must represent the objects it contains. Beyond that it could be either active or passive. An active world would control all its objects, implementing all the physical laws for that reality. A passive world would simply serve as a central data store, each object would be responsible for constraining itself to the laws of the reality.

The design that is chosen is required to make efficient use of multiple processors. While the active world is the closest to the manner in which physical reality works, it suffers from the disadvantage that most of the complexity is confined to a world module. Modification to the laws of the world would involve working with a large complex program. Most of the processing is done by the one process. This could be parallelised by placing computations for each object on separate processors, but then this reduces to the passive world case.

A third approach is to discard the world data structure and distribute the data over each object. This was rejected for this implementation because of the high communication overhead. One of the most common operations in a virtual reality system is to display the world. Having to gather object data whenever the world needs to be redrawn would place too high a load on the system.

The passive world approach was the one implemented since the system was to be implemented on a relatively fine-grained architecture. Each world consists of the world data structure and a simple server to supply data and update the world in response to requests from objects. The objects are active, consisting of an object data structure, together with a process. The object data structures are components of the world data structure, and so the object process must query the world server to access the object data. The object process must supervise the actions of the object to ensure that the laws of the virtual world are enforced. The world server and each of the object processes can be placed on different processors.

A number of the other systems make use of a combination of the other two distribution techniques. Data concerning a particular object may be kept by the object process. Various system processes responsible for maintaining the world may signal to the object to update its

position or other attributes in response to laws of the world or the influence of other objects. This approach may be better suited to relatively coarse grained parallel systems which can support a few, computationally expensive processes.

Once the idea of an autonomous object is accepted, then the distinction between a user, where a human controls the motion of an object, and an object, which controls its own motion, becomes blurred. The only difference is that users take motion control information from an input device and display their views on an output device. Such system calls can easily be included in the control routine for an object. There is no longer a need to distinguish between objects and users.

At this stage, each object and each world server is a separate process which may occupy its own processor. The parallelisation strategy thus involves distributing all the objects and worlds across all the processors in the system.

3.2. Functionality of the kernel

The design of virtual worlds is a new field and the requirements are not completely established. Several classes of functions are proposed as a basic set for the manipulation of virtual worlds.

- Each object in the world must be able to control its own attributes. This is a necessity since it has been decided that each object must control its own response to the laws of the world. Thus a set of functions to get and set the values of these attributes is required.

- In the physical world, the attributes of other objects can usually be measured. One can generally measure position and size of other objects. This should be the case for virtual worlds as well. This requires a set of functions to get the values of the attributes of other objects.

- A set of functions is required for changing the attributes of other objects. Use of these functions can introduce contention between processes for access to the data structure of a particular object. These functions need to include some mechanism for avoiding contention, examples of which are discussed below.

- Since the intention of the system is to support implementation of virtual reality applications, functions are needed to assist with this. In particular, functions to support interactions amongst objects, and between object and worlds are necessary.

- The user objects will need to be interactive, getting data from some devices and outputting data to others. A set of functions to communicate with the devices is necessary. The kernel will thus have some characteristics of an operating system, in that it must allocate these resources in an acceptable manner.

Changing the attributes of other objects introduces a measure of complexity. In the physical world, there is usually no difficulty involved in picking up a ball and dropping it. Considerable opposition may be experienced if another person is the subject of this action. Some mechanism is required to distinguish between the two cases. This should be dynamic; there may be times when picking up another user is appropriate.

The notion of ownership is one way of avoiding contention. Objects may either be ownerless, or owned by a specific object. Ownerless objects may only be controlled by their corresponding process. Owned objects may only be controlled by the process corresponding to their owner. To control another object, ownership must first be acquired. Ownership may then be relinquished at a later stage. This technique also provides the advantage of eliminating contention for objects. Only one object can control another at any time.

Ownership can be made hereditary. If object A gains ownership of object B who happens to own object C, then A can be considered as the owner of C. This could lead to problems with contention between objects A and B to control C. The approach taken to date is to allow only single generation control, A would control B, B would then have to change C accordingly. An exception is made in the case of transfer between worlds. When an object moves from one world to another, all directly and indirectly owned objects must be transferred as well.

The other approach to solving contention problems and providing inter-object communication makes use of events. Object processes have include routines to cause the object to change its own attributes in response to a signal from another process.

3.3. Communication between worlds and objects

This section describes the implementation of a passive world system with ownership as defined in the previous two sections. Inter-object communication is not included in this system. The implementation environment for this system was a cluster of transputers. Thus the following discussion is intended for a MIMD machine using message passing for communication and synchronization. Distribution of data would probably be simpler on a shared memory architecture. The message passing design, however, allows for the future expansion of the kernel to run on separate but networked machines.

The functionality required of the system is implemented as a library which can be called by each object process. The library then sends the appropriate messages to implement the required function. Thus the messages are not generated directly by the object process, but indirectly by one of the kernel functions called by such a process. The format, and very existence, of the messages is invisible to anyone using the kernel functions. A communication system is then required to perform the routing of the messages between the various processors.

The communication system is required to provide several services. Transmission of data between object process and world server is necessary for a number of purposes; to locate other objects in the world, to change position and other attributes or to display all the objects in the world. For each world there can be many objects. Objects need not remain in one world. They can, especially if they correspond to users, move from one world to another.

A limited interaction between objects is also necessary, particularly with regard to the implementation of ownership. Owned objects can be forcibly transferred from one world to another. The object process should be notified of this to allow it to keep track of where it can find the object data.

Communication with the devices also can make use of the communication layer. Devices will be identified by a single device number, all other data will be kept at a lower level. This will allow the devices to be used without the object process needing to know the address or nature of the device.

Use of the communication layer has several benefits. It keeps track of which object is in which world, and directs messages accordingly. When the object changes worlds, only a routing table need be updated; the object process is not affected. Communication with other machines can also be included in the design by modifying only the communication layer.

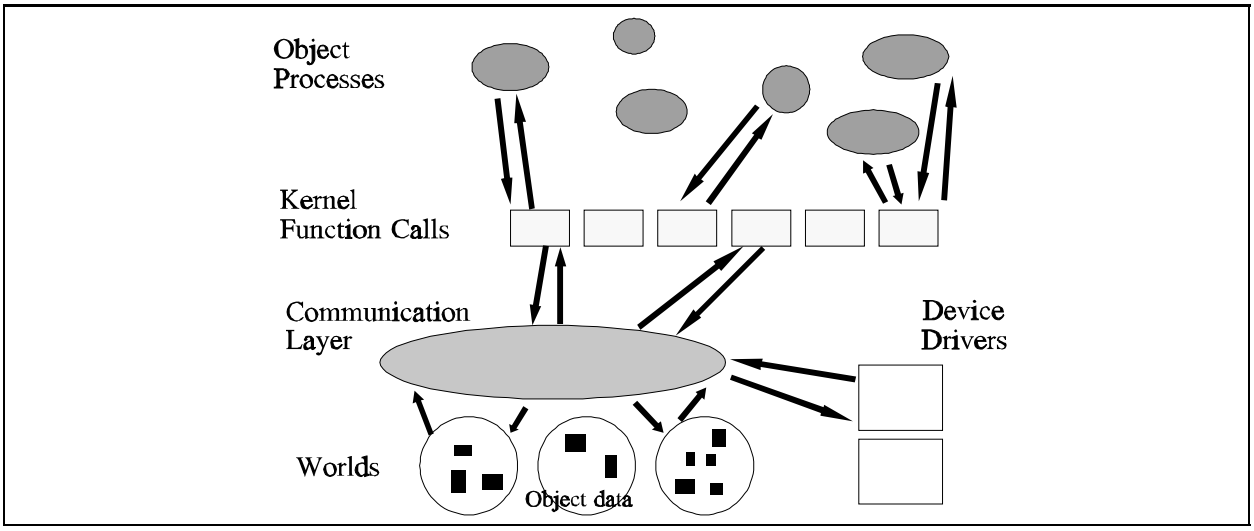


Figure 1 Summary of kernel design

A simplified view of the overall structure of the virtual world kernel is given in Figure 1.

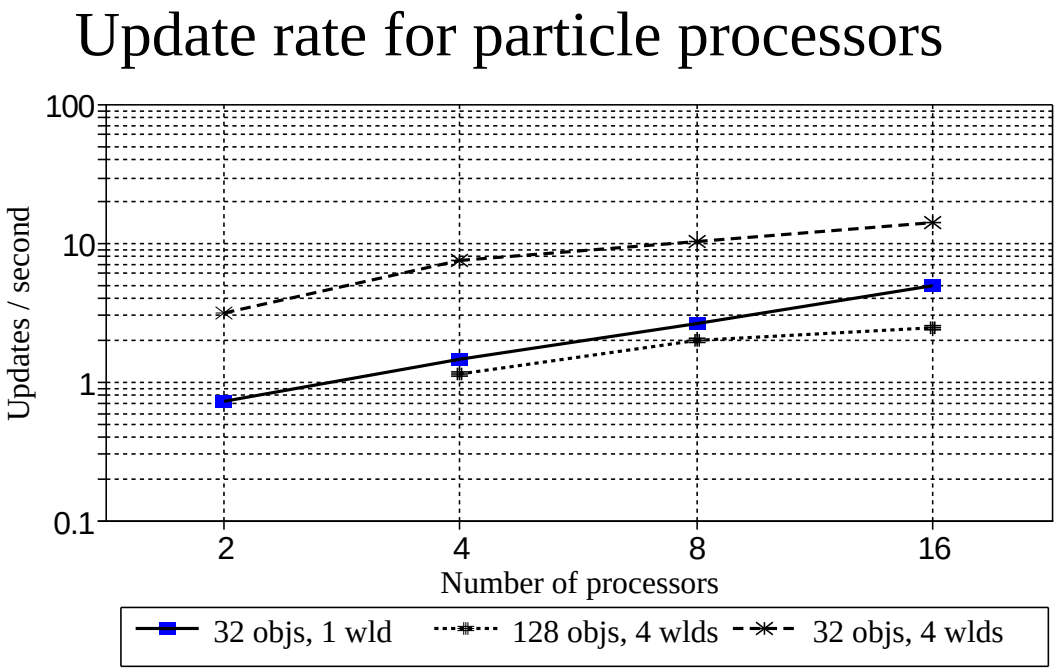
4. Kernel Performance

The effect of the parallel implementation of the virtual world kernel was measured for a simple simulation. This simulation modelled gas particles in a sealed container. Each particle could collide with the container walls and with all other particles.

The main loop of the object process controlling each particle is relatively simple. It first looks for the closest object to itself. If that object is closer than a critical distance, then it changes direction to indicate a collision. The particle process then checks the world attributes to locate any boundaries. If collision with a boundary is imminent, then a direction change is made. Finally, the object calculates its new position and stores this in the world database.

Three versions of the simulation were tested. The first used 32 particles and each particle process

Table I Update rate for particle processes

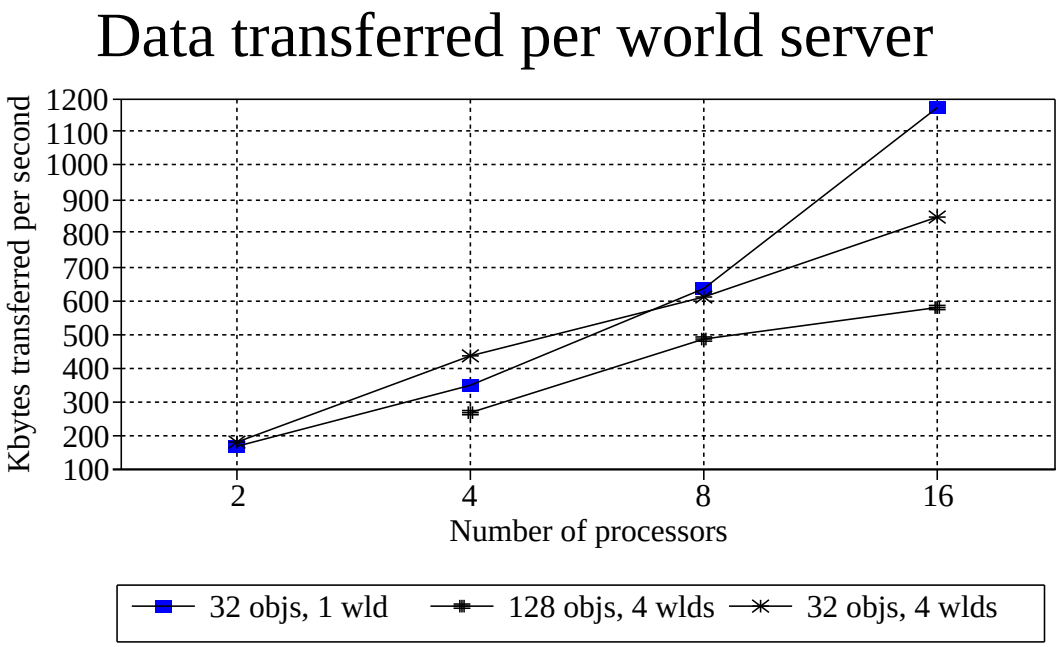


32 objects, 1 world	0.72	1.45	2.65	4.89
128 objects, 4 worlds		1.12	2.04	2.41
32 objects, 4 worlds	3.15	7.53	10.21	14.15

was given extra work to do so as to just prevent saturation of the communication links in the 16 processor case. The second used 4 worlds, each identical and each containing 32 particles. Extra work was given to each particle process but much less than for the first test case. This simulation was too complex to run on only two nodes. The last simulation again had 4 worlds, but with only 8 particles in each. No extra work was added. Timing figures may be seen in Table I.

The processors were configured as a hypercube of order $\log_2 N$ where N is the number of processors used. The communications time limits the speedup particularly in the last two test cases where much of the time used by each object process was spent waiting to messages to arrive. The collision detection is a particularly expensive operation with regard to communication, since it requires transfer of about 7 Kbytes per call. An estimate of the amount of data transferred per world server is shown in Table II, given that each particle process transfers about 7.5 Kbytes per iteration.

Table II Data transferred per world server in Kbytes/second



32 objects, 1 world	173	349	635	1172
128 objects, 4 worlds		268	490	579
32 objects, 4 worlds	182	436	612	848

For the first test case, increase in speed is due to the increase in the number of processors available to do the work. For the other simulations, bandwidth seems to be the dominant feature in determining the speedup. In these cases the speedup is closer to $\log_2 N$, which is also the number of links used per transputer in constructing the N processor hypercube.

The greater than linear speedup when going from 2 to 4 processors in the final simulation can be attributed to the presence of multiple world servers on a single node in the 2 processor case. Each server is a high priority process and would normally act immediately to an incoming message if alone on the processor. When sharing, it may have to wait for the other server to finish processing before it is able to run. Thus delays are removed when increasing the number of processors in this case.

5. Conclusion

The system described above simplifies the creation of a variety of virtual worlds. All objects are controlled by their own processes. Data for all objects corresponding to a particular world are stored in a single data structure from where they may be accessed. The system attempts to make maximum use of parallelism, by distributing each world and the objects in them onto a network of processors.

The centralised data structure simplifies the problem of collision detection, and simplifies the accessing of the data for one object by another objects process. Objects that are owned by another can be easily manipulated. At the moment, the design lacks a method for communication between objects of equal status. This must currently be implemented at a higher level by the application programmer.

A communication system exists to route data between object process and world database. This system is used to make the underlying parallel architecture transparent to the application programmer. It would also assist in extending the system to include other different processors. The load balancing with this approach is quite satisfactory, most object processes require delays.

The drawback to this approach is the amount of communication required. Greater bandwidth between processors is required for this design to be truly effective.

6. References

[1] Bangay, S.D., "Comparing virtual reality kernels", *Unpublished paper*.

[2] Coco, G.P., "The VEOS project : VEOS 2.0 Tool Builders Manual", available for anonymous ftp from milton.u.washington.edu as public/veos/veos.tar.Z.

[3] Snowdon, D.N., **West**, A.J., **Howard**, T.L.J., "Towards the next generation of Human—Computer Interface", *Informatique '93: Interface to Real and Virtual Worlds*, March 24 – 26 1993, 399 – 408.

[4] West, A.J., **Howard**, T.L.J., **Hubbold**, R.J., **Murta**, A.D., **Snowdon**, D.N., **Butler**, D.A., "AVIARY – A Generic Virtual Reality Interface for Real Applications", An invited paper for "Virtual Reality Systems" May 1992 sponsored by the British Computer Society.