A Comparison of Virtual Reality Platforms

Shaun Bangay

Department of Computer Science
Rhodes University
Grahamstown, 6140
South Africa

Internet: cssb@cs.ru.ac.za

1. Introduction

Over the last few years the field of virtual reality developed from a subject known to a few select researchers to a household discussion topic. Research into the subject over that period has increased the available software from a few systems running on expensive machines to systems that can give reasonable results on an inexpensive PC.

Many of the newer systems are intended to allow the development of diverse applications and are not restricted to one application. These systems are mostly university research projects, not commercial products, and as a result, source code and/or design documentation for these systems is often freely available.

This paper introduces the design of one such system, developed specifically for creating virtual reality applications on a parallel architecture. This system, referred to as the VROS (Virtual Reality Operating System), is then compared with other systems which were being developed at the same time. Practical experience with the other systems is limited since many require specialised hardware, both for computation and for interaction. For this reason, the discussion will concentrate mainly on design rather than performance issues.

2. The VROS

The VROS was created as part of the development of a virtual reality system specifically for a parallel architecture. The implementation environment is a cluster of transputers, communicating via message passing. The design decisions are documented in detail in [3] but the following brief description suffices for the purpose of the comparisons presented here.

The virtual universe is divided into worlds, representing each particular scenario. Each world contains a number of objects, which have various attributes including position and orientation in space. Some of the objects may represent the humans interacting with the system and are sometimes referred to as users. A virtual reality application will normally consist of one or more worlds containing object interacting toward a particular goal. An example of an application would be a walkthrough consisting of a world containing the building object and a number of furniture objects.

The VROS contains various device drivers for supplying data to the users (output device drivers) and reading data from them (input device drivers). However the section of interest for this discussion is the virtual world simulator, usually called the kernel.

The kernel is capable for supporting multiple worlds simultaneously, and allowing multiple users to enter each world.

To utilise the parallel architecture effectively, each world is represented as a data structure consisting of the attributes of the objects in that world. The objects must query the world for all data relating to them and the other objects in the world. This approach is implemented as a number of processes which can run on any of the processors in the transputer cluster. The world processes simply act as servers, supplying data to each object and updating their databases on request. In this way these processes can keep all the data consistent. The object processes control the actions of the objects in the world and implement the laws of that world. In addition, the object processes belonging to users may invoke the device drivers for additional control information. A communication layer exists for routing messages between processes, masking the details of the physical architecture. This is represented visually in Figure 1.

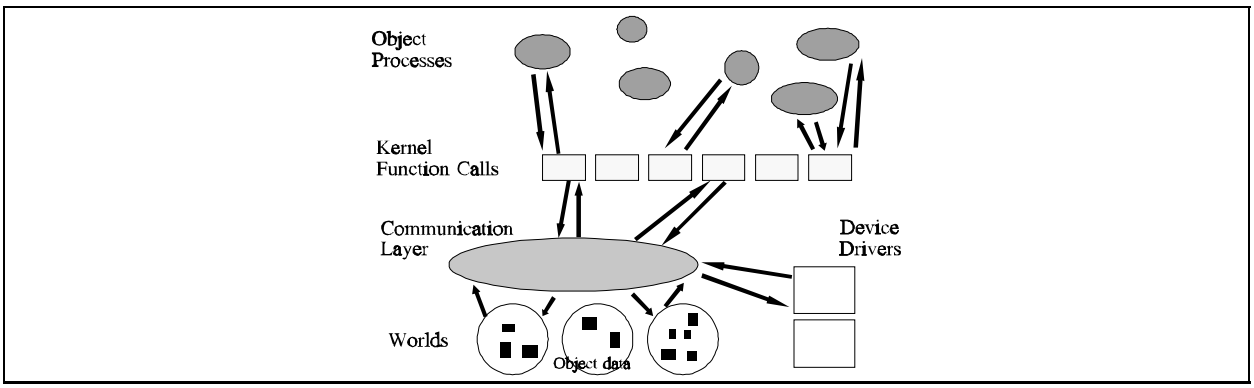

Figure 1 Summary of VROS design

Application programmers supply the control routines for each object. They are provided with a set of routines for manipulating aspects of the virtual worlds. The underlying architecture is invisible at this level. Routines exist for reading and updating the attributes of the current object. Similar routines exist for reading and updating attributes of the other objects, although more restrictions exist in this case. Objects can only control others if they are classed as owners of these objects. No direct communication between objects is currently supported by the kernel. Object processes may signal to each other by changing their attributes, or through communication routines that must be implemented by the application programmer. Various other functions exist to provide useful facilities such as transferring objects from one world to another.

3. Comparisons with other systems

The following sections describe other virtual reality systems, highlighting in particular the areas in which they differ with the virtual reality system described previously.

3.1. AVIARY

The Advanced Interfaces Group at the University of Manchester is working on the development of a general framework for advanced interfaces, which they are calling AVIARY [13][10]. This system is intended to support a broad range of Virtual Reality environments.

The AVIARY system currently runs on transputers and SUN workstations. The communications

system is the module most affected by different architectures. Versions are implemented for transputer networks and SUNs connected by ethernet. Graphics are produced by a hardware renderer.

The model of reality used in developing the AVIARY system is similar to that used in the VROS. Many of the same terms are used, but subtle differences exist in the meanings ascribed to these words. 'Worlds' are collections of attributes (eg. mass) and laws (eg. gravitation) rather than the data structure containing collections of objects and a few attributes as for the VROS. Objects are known as 'entities' and may be controlled by processes called 'demons'. Applications are distinct processes that manipulate the objects in the world. Many of the applications can exist in a single world, controlling the various objects. An application in the VROS is a more abstract concept consisting of one or more worlds, a collection of objects, and the manner in which they interact.

As with the VROS, objects are also permitted to be bound to processes which control their behaviour. An extra form of control is present, however, from users or applications. This differs from that found in the VROS, where the only interaction between objects is through the ownership concept. A user under AVIARY combines characteristics of objects, in that it also has a visible manifestation, and of applications, in that it is also subject to control beyond that of the physical laws of the world.

AVIARY is segmented into processes that can run in parallel. A communication system similar to that used in the VROS is present to allow communication between processes. The processes in the system consist of:

- Input processes (coinciding with input device drivers).
- Output processes (coinciding with output device drivers).
- A Virtual Environment Manager (where the VROS has multiple world servers).
- Environment Database that provides spatial management such as collision detection.
- Object Servers (corresponding to the object processes).
- Applications to control users or manipulate the virtual environment.

The communication and parallel strategies differ between AVIARY and the VROS. In the VROS objects communicate only with the servers, and support for communication between objects is

minimal. The world servers maintain a central data base for each world, and computational workload is limited to the object processes. With AVIARY, the various processes communicate extensively. Each object keeps the data relevant to it, and updates are transmitted when changes occur. Much of the computation is contained in the applications, and in the Environment Database which may limit the degree to which parallelism can be used.

The problem of supporting the range of features necessary to implement any reality is present in AVIARY as well. The solution implemented is to provide a basic world that may be customised to the purpose required. This results in a conflict between the need to provide assistance to the application writer while still allowing sufficient generality. The solution for the VROS is to provide library routines to handle the most common cases with the hope that few additions will be needed for more esoteric functions. The approach taken for AVIARY is far more rigid. The set of all possible worlds is structured as a hierarchy. The top of the hierarchy contains all possible worlds. Further down these laws are more refined. For example, some worlds may have gravity, while others do not. This information may also be used to restrict the types of objects that may be moved from one world to another. Consistency may be maintained by making sure that the object is capable of obeying the laws of the new world. A system of portals is used to link different worlds.

A strength of the AVIARY design lies in the ability to implement physical laws without excessive involvement on the part of application writer. The object oriented nature of the system with the use of inheritance to control attributes for different worlds is well suited to the design of a support environment for implementing virtual worlds.

3.2. Cyberterm

This system is intended to implement a single virtual world, a cyberspace that allows multiple users to share a common virtual area [9]. The single world is distributed over a number of workstations with each machine acting as a server for a portion of the world. The system is currently being implemented on PCs and SUNs connected by modem. Graphics are produced by public domain rendering libraries such as VOGLE and REND386.

The position of objects is kept by the server database. When an object enters a sector it makes

a local copy of this data. Velocity information is used to update the position of other objects, and updates are periodically issued when another object changes direction. This is appropriate where communication is over long distances and over limited bandwidth connections. With the VROS, it is acceptable to poll the server each time due to the fast transputer links.

Each processor runs a server and possibly a client. Movement from one 'sector' of the world to another requires the local client connecting to a different server. A similar system could be implemented in the VROS using multiple worlds to represent the various sectors. The difference, however, lies in the fact that the different processors in Cyberterm are separated by greater geographical distances.

The servers must issue permission for various actions, such as movement. Private areas of space can be created where rules decided on by the owner are enforced. This is the opposite policy to that taken with the VROS where such rules must be voluntarily obeyed by the objects. The bounding box attribute under the VROS is one way of defining a boundary, but if the object process does not implement the attribute, no further action will be taken. This relaxed attitude is reasonable for a prototype system, but may need to be more rigorously enforced in a commercial system.

3.3. Distributed Interactive Simulation (DIS)

DIS and its predecessor SIMNET are standards for distributed interactive simulations [7]. They are specifically intended for battlefield simulations. The simulations may involve thousands of objects and take place over a wide area network.

Communication occurs over a relatively low bandwidth medium, such as ethernet. Each host machine controls its own vehicle and keeps track of others by dead reckoning. Each host keeps track of its dead reckoned position and, when this differs significantly from its actual position, it transmits an update to all other hosts.

This approach is quite different to the VROS where all object data for a world is maintained by a single server process.

3.4. DIVE

DIVE (Distributed Interactive Virtual Environment) is a loosely coupled heterogeneous distributed virtual reality system based on UNIX and running over local and wide-area networks using Internet protocols [1][2]. It provides shared memory over a network and controls the sending of signals to processes.

A world consists of a set of objects and various parameters. It is a data structure, as in the VROS. Processes are capable of moving from one world to another by intersecting gateway objects. The implementation of a shared world differs from the server approach used by the VROS. Under DIVE the world is maintained as a replicated database. Each process has its own copy of the structure. Functions are provided to allow updating of entries in each copy for all the processes in the world. If all processes leave a world, the database is discarded.

An event handling system is present in DIVE allowing processes to register for certain types of event. The process can be notified when objects are created, removed, changed, or when interaction between a user and an object occurs. A timer event allows certain tasks, such as object movement, to be called periodically. Objects may be given primitive behaviour by specifying a state machine which performs certain actions on various events. A limited number of actions are possible, including moving, sending signals, and changing appearance.

The DIVE system consists of a set of processes each capable of manipulating the world and its objects. These processes consist of visualizer processes that allow users to interact with the world and application processes that operate on objects or introduce applications in the virtual world.

A number of high level tools are available for creating applications in DIVE. These functions support the selecting and grasping of objects. A vehicles module exists which uses the users actions to control the virtual environment. A gesture interpretation module in the VROS provides a similar facility.

3.5. Division

The ProVision system produced by a Bristol based company, Division, is a virtual reality server

that connects to a number of host machines [8]. The system is based on T425 and T805 transputers. Various support software is available, including the Distributed Virtual Environment System (DVS).

This system provides real time control and distributed event handling. All activities and environment handling under DVS are performed by processes called actors. Sharing of data between the actors is controlled by DVS. DVS provides more of a parallel programming platform than a system devoted exclusively to modelling of virtual worlds.

Parcels of data can be shared between various actors. Each actor makes a local copy of the data. In order for one actor to update the data, it must send an update request to a special actor, the director, which will then propagate the update to other actors holding that data. Updating can be done in exclusive mode which ensures that all actor processes have consistent copies at one time. The alternative is general mode which is faster, but actors separated by low bandwidth connections may experience delay in receiving the update. This is the opposite approach to that used in the VROS, where only one copy of the data is kept by a world server.

The actors control everything from 3-D input devices to geometry databases. This approach is more general than that used in the VROS, where specialised processes with customised communication interfaces are used for each particular task. The approach taken by DVS may make creating applications more complex, with greater understanding of the system required.

In order to cope with real time constraints, each actor can maintain its own local time. When communicating, the director will compare the different times of each actor and adjust them so that they are in step. This is useful in synchronizing different hardware devices that are operating at different speeds.

Rendering is done in hardware, using Toshiba HSP polygon processors. A renderer process called Paz converts a high level scene description, similar to the world data structure used in the VROS, to the polygon equivalent. Calls to Paz can be made to alter the position, motion and illumination of the objects.

3.6. Minimal Reality (MR) Toolkit

The MR toolkit is a library of functions for supporting the development of Virtual Reality interfaces [6]. It provides support for a number of peripheral devices used for Virtual Reality. It also provides facilities for distributing the Virtual Reality over multiple workstations. The MR Toolkit assumes that different hardware will be used for the different requirements of each process, and so concentrates of parallelism. Data sharing is via simulated shared memory on a message passing architecture.

The system provides the basic services. Support for creating virtual reality applications, as found in the VROS, will be provided by high level tools still being developed.

The toolkit consists of three levels of functions. The first level consists of device support functions. These are implemented as a client-server pair, with the server continuously polling the device so the client can have access to the most recent value without delay. The server also performs the low-level processing of the data such as filtering. This approach is the same as is used in the glove device driver in the VROS.

The second level converts the data from the devices into a convenient form for the application programmer. This corresponds to the gesture recognition stage in the VROS.

The third level of functions provides services for the application programmer. These include the maintenance of distributed data structures. This level would correspond to the virtual world kernel in the VROS.

The processes in an MR application can have three roles. One must be a master to control the application and start the other processes. There can be a number of slave processes that are used to produce graphical output. There may also be a number of computational processes that receive input from the master and return results to it. Data sharing is done by keeping local copies of the data with each process. The data structures must be periodically synchronised to ensure all processes have the correct values. The application programmer is responsible for specifying when this update occurs. This contrasts with the approach taken in the VROS. Here data is not shared, and the mechanics of updating the single copy of data structure is hidden from the application

programmer.

Communication is possible between separate MR applications. The master processes of each application can send device and application-specific data to other master processes. Slave processes must communicate via the master.

3.7. Multiverse

Multiverse is a multi-user X-Windows based Virtual Reality system [5]. The system runs on a UNIX platform and is based on a client-server model. It consists of servers that model the virtual world, and clients that are used for user interfaces. Each client and each server is a separate process, and each may run on a different machine.

Multiverse models objects as a data structure with an associated control process. Multiverse models a single world containing all the objects.

The clients consist of a single program that performs roughly equivalent functions to the input and output device drivers under the VROS. The clients are generic, and independent of the world being modelled by the server. They consist of a loop which renders the world, and sends any input from the user back to the server.

A server process is the equivalent of a world server and its corresponding objects under the VROS. The main functions of managing a virtual world are taken care of transparently; the application writer is required to supply only a few functions. These are mostly trivial, the one of interest being the **animateWorld** function that defines the nature of the world. It is called from the main server loop and is usually used to move the objects in the world. Since all processes runs on a single machine, there is no need for data sharing.

The objects may have special code to control their movement. Objects interact with each other and with the world using an event handling mechanism. These events include MOVE_EVENT that should cause the object to move, COLLISION_NOTIFY_EVENT for when objects have collided and TERM_NOTIFY_EVENT for when an object ceases to exist. The objects are not separate processes as with the VROS, but have to be called as part of the server process. The

object control routines are generally invoked when an event occurs which affects them. This sort of inter-object communication must at present be created by the application programmer when using the VROS.

The breakdown is similar to that of the VROS. The principal difference is the degree to which parallel processing is done. Simulation of the world in Multiverse uses a single thread of execution, as opposed to the multiple processes under VROS. However, the machines that would support Multiverse typically contain a single processor, and so creating more processes would be redundant.

3.8. VR-386

VR-386 [11] is virtual reality system for the PC which is descended from Rend386, a polygon rendering library for 386 and 486 based systems with VGA displays [12]. The current version is at an intermediate stage of development, and strongly reflects the need for efficiency when rendering views of worlds.

VR-386 represents a world as a structure containing all the visible objects in that world. It is intended to be capable of supporting multiple worlds and to allow switching between these worlds.

The objects in Rend386 could have several representations corresponding to different levels of detail. Figures constructed of a hierarchy of objects can also be defined. Objects are then stored relative to the parent object in the hierarchy. For example, in a human figure the arms and legs may be made children of the torso object. VR-386 goes further by adding a degree of animation and automatic updating for parts of a figure. Objects move when the parent object moves, with additional effects from the joints linking them.

VR-386 applications run as a single process, as opposed to the multiple processes under the VROS. VR-386 provides for extensive control of input and output, and also includes many functions for manipulating virtual worlds, similar to those provided by the VROS.

3.9. The Virtual Environment Operating Shell (Veos)

Veos is an environment for creating distributed applications for Unix [4]. It is designed for prototyping distributed Virtual Reality applications.

The processes required to implement a virtual environment are known as entities and can be distributed across a number of Unix workstations. A data type known as the ’grouple’ is used as the standard data structure. The grouple is an extension to the ’tuple’ used in the Linda programming paradigm. Grouples consist of nested tuples. Lisp is used as the programming interface to Veos.

Each Veos entity consists of a distinct Unix process that controls interpretation of the task written in Lisp. Each entity has associated grouplespaces for which pattern matching facilities are provided. Asynchronous message passing of grouples between entities is supported.

The use of interpreted Lisp makes the system flexible and easy to use. It also allows evaluation of program stubs passed as messages. This however will often limit the performance of the system.

The Veos system provides support for general distributed applications. Creating a Virtual Reality application still requires a great deal of work on the part of the programmer. The pattern matching facilities for the grouplespaces can assist in the modelling of virtual worlds.

Even though the grouplespaces may suggest use of shared memory, process communication still involves message passing.

3.10. Summary

Table 1 summarises the approaches taken by each system described in this paper. The areas for comparison are as follows:

Architecture :	Hardware used by the system
Level of Support :	Support for virtual reality applications in terms of basic structures included in the system

	Architecture	Level of Support	Complexity	Parallel Decomposition	Object Implementation	Object Control and Interaction
AVIARY	Transputer Clusters and Sun networks	Object User World Application	Mutiple worlds Multiple users	Parallel processes which control objects and implement applications	Processes communicating with messages	Message passing and inheritance of behaviour
Cyberterm	PCs and Suns connected by modem	Object User World Application	One world Multiple users	Servers for storing portions of the world database, and clients	Clients and world servers hold copy of data, and respond to updates	Servers pass world events to clients for action
DIS	Large numbers of workstations connected by ethernet	None defined	None	None	Independent copies of data is kept and modified by dead reckoning and occasional updates	Events
DIVE	Networked workstations	Object User World Application	Mutiple worlds Multiple users	Application processes operating on objects and Visual User processes	Data stored in simulated shared memory. Must be locked while updating	Events
Division	Loosely coupled workstations and transputer clusters	None defined	None	Concurrent actors performing all tasks	Actors keep local copies and trasmit updates	Events
MR Toolkit	Workstations connected by ethernet	Object only	None	Master, slave and computational processes	Data kept by master and updated on slave at specifed points	Events
Multiverse	A Unix workstation running X windows	Object User World Application	One world Multiple users	Server process which simulates world and objects, and client processes	Clients share a common database on the server	Events
VEOS	Networked workstations	None defined	None	Entities executing in parallel	Communication via ports	Communication via ports
VR-386	PC	Object User World Application	Multiple worlds One user	No parallel processing	Objects share a common database	Animations
VROS	Transputer cluster	Object User World Application	Multiple worlds Multiple users	Object processes controlling objects and world servers containing database	Object processes query world servers for data	Sending through object attributes

Table 1 Summary of differences in the various systems

Complexity : Support for interaction between more than one user and world

Parallel Decomposition : Manner in which parallelism is used within the system

Object Implementation : The way in which object share data

Object Control and Interaction : Facilities for co-ordinating object behaviour

4. Conclusion

The comparison with the other recently developed general virtual reality systems showed a number of features common to all systems. These have been implemented in different ways.

Generally, the systems identify the concept of an object, with objects grouped into worlds. The objects are generally controlled in some manner to respond in a realistic manner to the other objects and the nature of the world. Some of the system described have a means of enforcing this control on objects. A shortfall in the design of the VROS is the lack of facilities for object communication. At present, this must be manually built into each object process by the application programmer.

It is especially noticeable that almost all systems make use of some degree of parallel processing. The extent of this varies with the implementation architecture. Some form of data sharing is then necessary. The ways in which this is done is very much dependent on the bandwidth available for communication. Systems with slow links may use predication to estimate the position of other objects, while faster communication allows data to be shared whenever necessary. The strong point of the VROS is that it makes greater use of parallelism than the other systems, and may achieve a more even distribution of the computational load.

5. References

[1] Andersson, M., **Carlsson**, C., **Hagsand**, O., and **Ståhl**, O., "DIVE — The Distributed Interactive Virtual Environment Tutorials and Installation Guide", Technical Report, Swedish Institute of Computer Science.

[2] Andersson, M., **Carlsson**, C., **Hagsand**, O. and **Ståhl**, O., "DIVE — The Distributed Interactive Virtual Environment Technical Reference Manual", Technical Report, Swedish Institute of Computer Science.

[3] Bangay, S.D., "Creating Virtual Reality Applications on a Parallel Architecture", Unpublished paper.

[4] 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.

[5] Grant, R., Multiverse description and sources, available by anonymous ftp from ftp.u.washington.edu as public/virtual–worlds/multiverse–1.0.2.tar.Z.

[6] Green, M., "Minimal Reality Toolkit Version 1.2 : Programmer's Manual", Technical Report, University of Alberta, Edmonton, Alberta.

[7] Locke, J., "An Introduction to the Internet Networking Environment and SIMNET/DIS", available by anonymous ftp to sunee.uwaterloo.ca as pub/vr/documents/DISIntro.ps.

[8] Pountain, D., "ProVision: The Packaging of Virtual Reality", *Byte*, **16**(10), October 1991.

[9] Snoswell, M., "Overview of Cyberterm, a Cyberspace Protocol Implementation", available by anonymous ftp to sunsite.unc.edu as pub/academic/computer–science/virtual–reality/papers/Snoswell.Cyberterm.

[10] 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.

[11] Stampe, D., "vr_api.h", available for anonymous ftp from psych.toronto.edu as pub/vr–386/vr_api.h

[12] Stampe, D. and **Roehl**, B., "REND386 – A 3–D Polygon Rendering Package for the 386 and 486 : LIBRARY Documentation Version 4.01 – September 1992", available for anonymous ftp from sunee.uwaterloo.ca as pub/rend386/devel4.zip.

[13] 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.