Spelunking: Experiences using the DIVE System on CAVE-like Platforms

Anthony Steed and Jesper Mortensen
Department of Computer Science
University College London
Gower Street, London, UK.

Emmanuel Frécon
Swedish Institute of Computer Science
Box 1263
S-164 29 Kista, Sweden.

Abstract

The Distributed Interactive Virtual Environment (DIVE) system is a mature toolkit that has been used for prototyping many collaborative virtual environment (CVE) applications [3]. Until recently, support for immersive systems has been limited to custom support for specific installations and the emphasis has been on broad support for desktop interfaces. Recently we have ported DIVE to the CAVElib™ environment† and this paper describes application programming support for immersive users and our experience in using wide-area distributed applications involving immersive projection technologies. The resulting toolkit provides a very flexible system with which to build distributed VE applications involving a variety of platforms and interfaces.

Keywords: Distributed virtual environments, virtual reality toolkits, immersive systems.

1 Introduction

One of the key requirements for the next generation of collaborative virtual environment (CVE) toolkits will be broad support of a variety of platforms from desktop through to immersive projection technologies (IPTs). Today there is a plethora of CVE toolkits boasting easy reconfiguration, broad support for devices, object-orientated programming paradigms, etc. In this paper we discuss the evolution of a mature CVE toolkit, DIVE, to support IPT displays. DIVE brings a number of useful facilities to the IPT programmer and although it does not pretend to be a replacement for low-level libraries, we suggest the framework it provides will be very useful for users wishing to prototype or explore CVE applications. In this paper we outline how DIVE has been extended to support IPTs through integration of the CAVElib™ library and we discuss some experiences of using DIVE over wide-area networks involving a variety of desktop and IPT systems.

† CAVELib is a registered trademark of the University of Illinois.

In Section 2 we give and overview of the DIVE system and in Section 3 explain how we have extended this to support IPTs. Section 4 discusses our experience in using the DIVE/CAVElib™ port in network trials and in Section 5 we discuss some distinctions between DIVE and other CVE toolkits and our plans for future work.

2 DIVE System Overview

2.1 System Services

At the conceptual and programming level, DIVE is based on a hierarchical database of objects, termed *entities*. Applications operate solely on the database abstraction and do not communicate directly with one another. This technique allows a clean separation between application and network interfaces. Thus, programming will not differ when writing single-user applications or multi-user applications running over the Internet. This model has proven to be successful; DIVE has changed its inter-process communication package three times since the first version in 1991, and existing applications did not require any redesign.

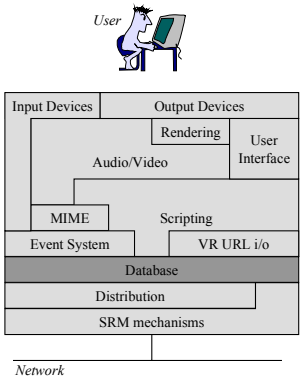

Figure 1. The different components that compose the DIVE system.

While the hierarchical model is inherited from traditional scene graphs, as used in the computer graphics community the DIVE database is semantically richer. For example, it contains structures for storing information about other users, or non-geometric data specific to a particular application. In DIVE, the database is partially replicated at all participating nodes, with a top-down approach, i.e. mechanisms are offered to control the replication of sub-branches of a given entity.

In DIVE, an event system realises the operations and modifications that occur within the database. Consequently, all operations on entities such as material modifications or transformations will generate events to which applications can react. Additionally, there are spontaneous and user-driven events such as collision between objects or user interaction with input devices. An interesting feature of the event system is its support of high-level application-specific events, enabling applications to define their content and utilisation. This enables several processes composing of the same application (or a set of applications) to exchange any kind of information using their own protocol.

A few events are local to each process, e.g. rendering events. However, most events occurring within the system will generate network updates that completely describe them. Other connected peers that hold a replica of the concerned entities will be able to apply the described modification unambiguously. Network messages are propagated using the multicast mechanisms that are built in the system. DIVE uses a variation of SRM (scalable reliable multicast [2]) to control the transmission of updates and ensure the consistency of the database at all connected peers. The SRM approach requires the transport layer to be able to ask DIVE applications to regenerate updates if necessary. Update regeneration is necessary when gaps are discovered in the sequence numbers that are associated to every entity of the database. Gaps imply that network messages must have been lost along the path from a sender to one of its receivers. A thorough description of DIVE's networking mechanisms can be found in [3]. In addition to SRM, extensions to DIVE make it possible to access any document using more common Internet network protocols such as HTTP and FTP, and to integrate these documents within the environment by recognising their media types, such as VRML.

In any application, the content of the database must be initialised. DIVE uses a module that manages several three-dimensional formats and translates them into the internal data structures that best represent their content. Usually only one peer will load, and parse, a particular file and the resulting entity hierarchy will be distributed to other connected peers through a series of (multicast) updates that describe the resulting entities. This specific mechanism differs from many other systems that rely on being able to access the description files from all connected peers.

DIVE has an embedded scripting language that provides an interface to most of the services of the platform. This language is a superset of the Tool Command Language (TCL) [9] and is called DIVE/TCL. Scripts register an interest in, and are triggered by, events that occur within the system. They will usually react by modifying the state of the shared database. Moreover, these modifications can lead to other events, which will possibly trigger additional scripts. There are numerous commands; they allow the logic of the scripts to gather information from the database, decide on the correct sequence of actions and modify the state of the database accordingly. The completeness and simplicity of the scripting interface have made it the programming interface of choice for most applications developed in DIVE.

The DIVE run-time environment consists of a set of communicating processes, running on nodes distributed within both local and wide-area networks. The processes, representing either human users or autonomous applications, have access to a number of databases, which they update concurrently. As described earlier, each database contains a number of abstract descriptions of graphical objects that, together, constitute a virtual world. A typical DIVE application will, upon connection to a virtual world, introduce a set of objects to the environment that will serve as its user-interface and start listening to events and react accordingly. One essential application of the system is the 3-D browser, called *vishnu*. *Vishnu* is the application that gives its user a presence within the environment. It introduces a new entity called an actor to the shared environment, which is the virtual representation of the real user.

2.2 User-Oriented Services

The services described previously are independent of any DIVE application. This section focuses on the different modules present within the vishnu application that render a visual and aural space and provide the users with an interface that allows them to explore and interact with this space.

The primary display module is the graphical renderer. Traditionally, the rendering module traverses the database hierarchy and draws the scene from the viewpoint of the user. This module provides for geometry and material types beyond those found in common scene description languages. Additionally, some constant frame rate techniques allows for the rendering of environments that are both large in extent and deep in details, see [7].

DIVE has integrated audio and video facilities. Audio and video streams between participants are distributed using unreliable multicast communication. Audio streams are spatialised so as to build a soundscape, where the perceived output of an audio source is a function of the distance to the source, the inter-aural distance and the direction of the source. The audio module supports mono-, stereo- or quadri-phony audio rendering through speakers or headphones connected to the workstation. Input can be taken from microphones or from audio sample files referenced by a URL. Similarly, the video module takes its input from cameras connected to the workstations or video files referenced by URLs. Video streams can either be presented to remote users in separate windows or onto textures within the rendered environment.

Users are generally also presented with a two- dimensional interface that offers access to rendering, collaboration and editing facilities. The interface itself is written using the same scripting language as offered by the world database. Consequently, CVE applications can dynamically query and modify the appearance of the 2D interface.

Finally, a MIME (Multimedia Internet Mail Extensions) module is provided to better integrate with external resources. It automatically interprets external URLs. For example, an audio stream will be forwarded onto the audio module where it will be mixed into the final soundscape.

3 Immersion Support

Previously DIVE had custom support for a small number of immersive devices to satisfy the needs of the authors. These include drivers for Ascension Flock of Birds, Polhemus IsoTrak and Fastrak and a variety of head-mounted displays. To enable support of IPTs such as CAVE™s† we have implemented a DIVE application framework that interfaces to the CAVElib™ library. The resulting application was named *spelunk*. It differs from the standard *vishnu* application in that the renderer is written in Performer rather than OpenGL and there is an extra module for interaction which we called the spelunk vehicle.

† CAVE is a trademark of the Board of Trustees of the University of Illinois.

3.1 Performer Renderer

There are implementations of the DIVE renderer for many rendering toolkits, but primary amongst these in terms of stability and functionality was the OpenGL renderer. In the COVEN project we had done extensive work on this renderer to enable scalability to very large models [7].

The choice of porting the rendering module to Performer was motivated by two major reasons. Performer is a well-designed piece of rendering software that has existed for many years. As a result, it implements a number of optimisations that DIVE will be able to benefit from, at the expense of not being able to modify and control the rendering techniques that are used. Furthermore, each Performer viewport uses a number of pre-emptive threads, as opposed to DIVE, which relies on non-preemptive threads only. Consequently, Performer makes better use of "monster" architectures by being able to place concurrent threads on several CPUs and several rendering pipelines, whenever available and applicable.

The standard OpenGL renderer was written in an immediate mode style, with each object type within the database having a different callback. The Performer renderer is written in a very different manner; it builds a Performer scene-graph when objects are created within the local database and updates them only when necessary. Fortunately the DIVE event mechanism allows us to easily obtain notification when any change affecting the appearance or form of an object occurs. We maintain the DIVE scene graph for the purposes of efficient event recreation and to support the other modules apart from rendering, such as navigation and collision detection, which require the complete geometrical description.

Some of our scalability work on the OpenGL renderer was in optimisations such as optimising triangle strips, re-ordering objects to optimise OpenGL state changes and compiling display lists. These replicate functionality built-in to Performer so we have not duplicated the effort. Functionality that is harder to replicate in Performer and is the subject of future work, are extensions to cope with massive worlds that contain many more objects than can be rendered in real-time. At the moment the Performer renderer adaptively changes the far clip distance if a frame rate target is significantly over-run by the rendering processes.

3.2 Spelunk Vehicle

A vehicle is a collection of interaction techniques and methods that respond to input events and effect locomotion, selection and manipulation within the environment. Vehicles are also commonly responsible for manipulating the pose of the avatar to reflect input. For example they may draw rays, or animate a gait pattern. Many vehicles have been implemented covering the range from keyboard and X event input through virtual widget vehicles that lie on a heads-up display, to vehicles that rely on 3D tracker information. Multiple vehicles can be activated simultaneously, though some are mutually exclusive, and not all vehicles make sense in all contexts.

For spelunk we implemented a vehicle that interfaces to the tracker input facilities of CAVElib™. The vehicle registers a callback on a local PRE_FRAME DIVE event.

The callback reads both the current tracker positions and the state of the buttons and joystick and effects interaction.

The first responsibility of the vehicle is to effect locomotion, manipulation and selection. Locomotion is a matter of changing the world co-ordinates of the avatar. Each avatar is a direct child of the root in the world's scene graph so this process is simple. We must simultaneously alter the mapping within CAVElib™ from physical co-ordinate to world modelling co-ordinates. Selection is usually effected on a button press. The default method is selection at a distance. DIVE has built-in support for ray based intersection as well as full object-object collision detection. By default, manipulation is simple attachment of the selected object to the position of the user's hand. We implement this by dynamically re-parenting the selected object under the user's hand in the scene graph.

The second responsibility of the vehicle is to animate the avatar. In DIVE avatar representation is independent of the application and users may specify a default avatar they wish to use for every session or can specify an avatar on the command line whilst starting up. It is also possible to dynamically change the avatar at run-time. It is usual for the tracker information for head and hand positions to be reflected in the position of graphical representations of the head and hands. We also provide an optional animation that can articulate the arms of tracked hands and also body to provide a more anthropomorphic-looking avatar.

The vehicle responds to several configuration variables, including default vehicle speed, mapping of wand buttons to actions, joystick dead-zone and rotation speed scaling. Configuration variables can be changed from module and plugin C/C++ code, and TCL code associated with the interface or objects in the current world.

Internally the spelunk vehicle relies on its own callback registration mechanism. The vehicle was designed this way to allow easy extension of the spelunk vehicle to support other interaction techniques. Thus plugins can register and over-ride callbacks on different CAVElib™ events. We have used this for example to build a plugin that over-rides the default spelunk selection metaphor of "selection at distance" to the "go-go" interaction technique [5]. Note that because plugin can register TCL functions, and since the spelunk menu system is built in TCL/TK we were able to dynamically change the spelunk menus to add options to change the behaviour of the go-go technique at run time. We used this mechanism to have the plugin optionally provide menus to configure the threshold and scaling factors associated with the go-go technique.

3.3 Desktop & Immersive Menu Interfaces

A DIVE application may or may not create a user interface when it starts up. Most DIVE applications are linked with TCL/TK so as to provide scripting of objects within worlds and thus it is natural to use TCL/TK to build a user interface. TCL/TK desktop DIVE applications such as *vishnu* will build a TCL/TK window structure and then delegate rendering of particular windows to the renderer by passing it a window context. An example of the *allinone* application's interface is shown in Figure 2

(*allinone* is a variant of the *vishnu* application). The *allinone* interface allows access to a very wide range of functionality: world and object loading, scene graph editing, avatar loading, system configuration (including audio, video, network behaviour), object editing, text and audio chat groups, subjective views and world, bookmark and viewpoint menus and participant activity summary. The main *allinone* window contains a menu and location bar at the top, rendering window below that, and viewpoint, participant and text-chat panes surrounding the rendering window.

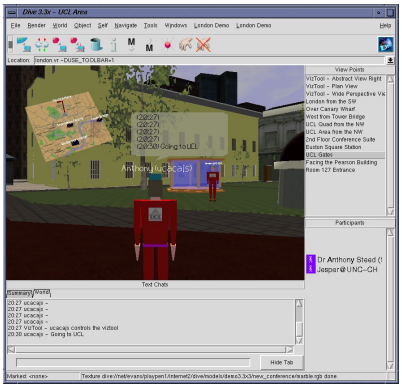

Figure 2. *allinone* desktop interface to the *vishnu* application

With the spelunk application we build a user interface on start-up and display it to a user-specified display. The spelunk interface consists simply of the menu and location bars from *allinone*, though the viewpoint, participant and text chat windows as still available. Keyboard input in still active so a person sat at the console can drive the user around independently of the joystick controls. Note also there are also other 2D widget based vehicles that can be used, the viewport menu allows the console user to teleport the immersed user about and the participant window allows teleportation to other users. This functionality is very useful when demonstrating immersive systems in network situations.

Of particular interest within the spelunk menus is the *Navigate* menu that allows access to control features of various vehicles. This allows us to enable and disable vehicles at run-time and control parameters of the vehicle by through the DIVE/TCL APIs or by setting configuration parameters. For example for the CAVElib™ vehicle the spelunk menu allows the console user to change the velocity, planar constraints, button control mapping, direction of motion and collision detection response.

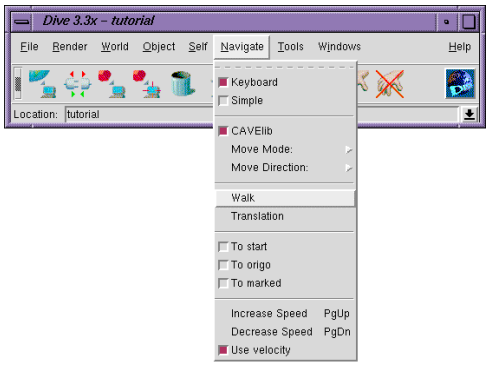

Figure 3. Spelunk menu system displayed to a desktop

These console menus therefore give the console user significant power to modify the world and the user's interaction with the world at run-time. To enable world editing work within IPT systems we can reflect the spelunk menus onto surfaces within the world as shown in Figure 4. This is achieved through a DIVE plugin that acts as a VNC† client. The plugin both receives VNC paint events and maps selection and manipulation events on the 3D surface into VNC input requests. To enable this, the spelunk menus are displayed to a VNC server rather than an X display, though it is possible to access the menus simultaneously from desktop and IPT by using a standard X or Windows VNC client as well as the DIVE plugin VNC client.

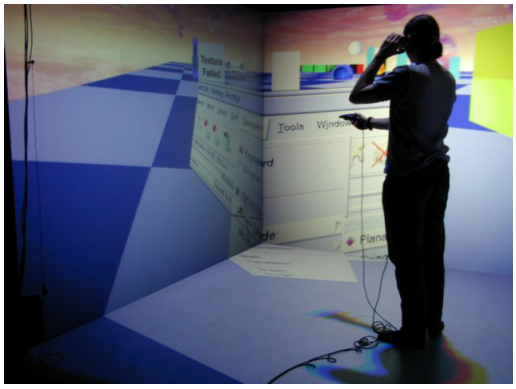

Figure 4. Using spelunk menus within the UCL ReaCTor

4 Distribution Support and Network Trials

We have used the desktop and custom immersive versions of DIVE for several years on collaborative projects. With the new CAVElib™ port we have been able to very rapidly bring our new IPT systems into these collaborative trials and perform trials

† Virtual Networking Computing, AT&T Laboratories Cambridge, available from http://www.uk.research.att.com/vnc/

and experiments using a variety of platforms. DIVE is a very useful platform for prototyping CVE applications since it takes a world-centric program approach, with individual worlds being constructed from a collection of standard sources such as VRML97 files, with behaviour being scripted in TCL/TK. The actual client applications change very little, and then usually only through initialisation time configuration through plugins. Note object behaviour scripts can check for the existence of a plugin and can alter their behaviour accordingly or warning the user that certain functionality will be disabled.

Given the interpreted nature of world programming, it is very simple for us to interactively edit a world in a distributed manner. A common editing practice is to load a world on both the IPT system and a desktop machine. The immersed user can enter the world, whilst the desktop user interactively edits the world with immediate effect on the immersed user's experience. Furthermore through DIVE/TCL programmers can remotely access interface and plugin functionality on other clients and can use this to reconfigure clients as necessary. Example uses of this is to dynamically add or delete items from the heads-up display of other users (e.g. the miniature map in Figure 2), or force them to re-align to world coordinates if they get lost.

4.1 DIVEBONE

Distribution in DIVE is based around multicast communications. For networks where multicast is not available, DIVE supports an application level bridge called the DIVEBONE [4]. A process called *proxyserver* can listen on multicast groups and transmit packets in a unicast manner. DIVE processes (including other proxyservers) can then connect directly to the unicast port on the proxyserver. Typically this might be used to connect together multicast "islands". We use this mechanism extensively in the trials described in the following two sections.

4.2 Internet2

As part of the Collaboration in Tele-Immersive Environments project funded by UKERNA we have been testing DIVE over transatlantic experimental academic networks. Our interest in these tests has been to investigate requirements for effective communication in virtual worlds. In these trials we are collaborating with laboratories at MIT (Massachussetts Institute of Technology) and UNC-CH (University of North Carolina Chapel Hill). Figure 5 illustrates the network setup in these trials.

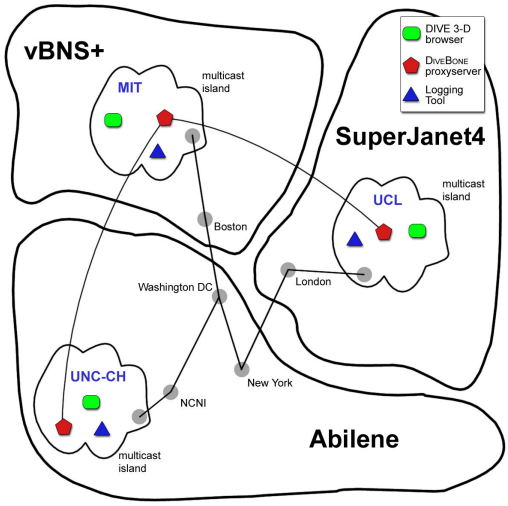

Figure 5. Network topology

Approximately eight afternoon trials have been held so far using a variety of models with aural communication. We use the DIVEBONE to connect together three multicast pools, one at each site. We have used a variety of machines, including Windows NT3.5 & 2000 and SGI O2 in desktop mode, SGI Onyx2 at UCL driving the ReaCTor and SGI Onyx2 at UNC-CH driving HMD's. Over this test-bed we get very stable round-trip times of ~130ms on average and ~90ms at best from UCL to UNC-CH (and similar RTT's to MIT). In our situation the gross bandwidth is not an issue except at model load time. Network performance analyses on the 14 hop path between UCL and UNC-CH (using pchar†) have shown that the bottleneck in this setup is the 10Mbps LAN link between the Onyx2 driving the ReaCTor at UCL and the external Cisco router at UCL and the sustained bandwidth achieved in tests is ~7.4Mbps between these two sites (similar performance between UCL and MIT). We have experimented using the London Traveller application [7] between these sites without any difficulty (see Figure 2). This model is ~107MB and is loaded across the transatlantic link in ~2 mins.

The low latency and absence of significant jitter means that collaboration between the sites is very natural. We plan to continue these trials looking at requirements for latency and jitter for visual, audio and haptic interaction.

4.3 ReaCTor to VR-CUBE

As part of a collaboration between UCL and Chalmers Technical University, Gothenburg, Sweden, we have been running a series of trials between two IPT systems. The IPT system at Chalmers is a TAN VR-CUBE consisting of four walls and a floor driven from a 14 processor Onyx with 3 InfiniteReality2 pipes. Tracking is

† pchar is written by Bruce A. Mah of Cisco Systems, it is based on the algorithms of the pathchar utility written by Van Jacobson, formerly of Lawrence Berkeley Laboratories.

done with Polhemus Fastrak and there is a tracked Wanda controller.

We installed spelunk on the Chalmers VR-CUBE in early October 2000. The only configuration required beyond normal DIVE installation was an adjustment of the dead-zone for the joystick. We tested several simple applications between the two sites and between mid-October and early-December we ran 7 afternoons of experiments using spelunk.

The network situation between UCL and Chalmers was very different to the Internet2 trials. Again the DIVEBONE was used to connect the sites together. This time UCL ran a proxyserver and Chalmers connected directly to this over a unicast connection. This allowed UCL to monitor activity by adding a desktop user as a 3rd client. The desktop user had an invisible avatar so the participants were not aware of their presence. Network latency varied between ping times of 160 and 380ms. Users did not notice any lag in the animation of the avatars, though when jitter was high occasional drop outs in audio occurred.

The experimental situation was similar to that reported in [8] with two users collaborating in the solution of a cube assembly puzzle. Our initial results indicate that collaboration between users of IPT systems is much more fluid and natural than either collaboration between IPT system and desktop or between two desktop users. Setting up and controlling the experiment was much more simple than our experience with running distributed experiments with other CVE toolkits [6]. The experimental scenario was ported from a different CVE toolkit in two ½ day sessions, one of which was a collaborative review to investigate different lighting effects and avatar representations.

5 Discussion

5.1 Comparison to Other Toolkits

DIVE provides two principle levels at which to program: custom C/C++ applications using the DIVE core libraries and C/C++ plugin based extension of one of the standard processes in combination with world files and interpreted DIVE/TCL scripts.

Most CVE toolkits that support IPT displays such as VRJuggler [1] support the former paradigm. Thus a CVE is set up between custom applications, and the CVE toolkit provides a framework for building such applications. Although DIVE does support many of the same library facilities as recent toolkits, it was not primarily designed to be used this way. However the plugin interface goes a long way to providing the types of initialisation-time extension that is required by most programmers, and the scripting interface encourages programmers to think about writing plugins that provide services that can be exposed through the DIVE/TCL scripting interface rather than build monolithic applications.

DIVE's origins were in workstation based 3D applications for distribution over wide area networks. Thus two areas are well developed: desktop interfaces and multicast networking. As has been discussed in previous section support for desktop interfaces is very wide, with run-time reconfiguration of interfaces being exploited in many

applications. The multicast support is also mature, with support for real-time audio and video transmission as well sharing of the world-database.

5.2 Future Work

The current version of DIVE including CAVElib™ support will be available shortly from http://www.sics.se/dive/ as version 3.4x of DIVE. This release will be a maintenance release. A new branch of work, that will probably lead to version 4.0, has now started at SICS. This will focus on general improvements to the internals of DIVE such as an entire rewrite of the database and event system and moving many of the current modules in to plugins that will be dynamically loaded as required at run-time.

DIVE's focus on multicast networking does mean that currently it is not well suited to large-volume data visualisation problems. Currently the database is geometry centric, with bulk transfer of data taking place over ftp or http. This is being addressed in a number of manners. First the internal database and renderer are being re-engineered for more generic object type support. Second we are investigating model transfer protocols that take into account pre-computed visibility within the models. We will also revisit the issue of scalability support in scenes where there are many more objects in the scene than can be rendered at a reasonable frame-rate at any level of detail.

References

1. Bierbaum, A., "VR Juggler: A Virtual Platform for Virtual Reality Application Development". MS Thesis, Iowa State University, 2000.
2. Floyd, S., Jacobson, V., Liu, C., McCanne, S. & Zhang, L. (1997). "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *IEEE/ACM Transactions on Networking*, 5(6), 784-803.
3. Frécon E., Stenius M., "DIVE: A Scaleable network architecture for distributed virtual environments", Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments), Vol. 5, No. 3, Sept. 1998, pp. 91-100. See also *http://www.sics.se/dive*
4. Frécon, E., Greenhalgh, C. & Stenius, M. (1999), "The DIVEBONE - An Application-Level Network Architecture for Internet-Based CVEs", *Proceedings of the ACM Symposium on Virtual Reality Software and Technology'99* (pp. 58-65), London, UK: ACM Press.
5. Poupyrev, I., Billinghurst, M., Weghorst, S. and Ichikawa, T. (1996). "The Go-Go Interaction Technique: Non-Linear Mapping for Direct Manipulation in VR". In Proceedings of ACM Symposium on User Interface Software and Technology (UIST '96), pp. 79-80.
6. Slater, M., Sadagic, A., Usoh, M. and Schroeder, R. "Small Group Behaviour in a Virtual and Real Environment: A Comparative Study". Presence: Teleoperators and Virtual Environments. 9(1), 37-51, 2000
7. Steed, A., Frécon, E., Avatare-Nöu, A., Pemberton, D. & Smith G. (1999). "The London Travel Demonstrator", *Proceedings of the ACM Symposium on Virtual Reality Software and Technology'99* (pp. 58-65), London, UK: ACM Press.
8. Wideström, J. Abelin, Å. Axelsson, A-S, Nilsson, A., Schroeder, R., "The Collaborative Cube Puzzle: A Comparison of Virtual and Real Environments". CVE 2000, the Third International Conference on Collaborative Virtual Environments. 2000.
9. Ousterhout J. (1994), "TCL and the TK toolkit", Addison-Wesley, ISBN 0-201-63337-X.