Distributed Virtual Reality Systems

Chris Joslin, Tom Molet and Nadia Magnenat-Thalmann

MIRALab, CUI, University of Geneva
24, rue du General Dufour
CH-1211 Geneva Switzerland
{*joslin,molet,thalmann*}*@miralab.unige.ch*

Abstract

In this paper we present our Networked Virtual Environment (NVE) System, called W-VLNET (Windows Virtual Life Network), which has been developed on the Windows NT Operating System (OS). This paper emphasizes the Real-Time aspect of this NVE system, the advanced interactivity that the system provides and its ability to transfer data across the Internet so that geographically distant users can collaborate with each other. Techniques for communication, scene management, facial and body animation, and general user interaction modules are detailed in this paper. The use of VRML97 and MPEG4 SHNC is overviewed to stress the compatibility of the system with other similar Virtual Reality systems. The software provides realistic virtual actors as well as sets of applicable high-level actions in real-time. Related issues on obtaining actor models and animating them in real-time are presented. We also introduce a case study to show an example of how the system can be used.

1 Introduction

Since the early 90's Networked Virtual Environments [1–3] have been available in some respects in many research institutions. The idea of collaborating with someone geographically remote has driven research on some of the most powerful UNIX systems, but like many other real-time systems, NVE Systems suffer from limited complexity and interactions. NVE are like any other Virtual Environments, they attempt to trade realism with speed. In the past only the aforementioned powerful UNIX systems were able to handle the complex interactions and required rendering speed of these Virtual Environments, but now powerful multiple CPU machines, based on the Windows OS, are able to take on such tasks.

A few commercial systems [4] are cropping up, although used mainly through a Web Browsers Java Interface, these systems assume a medium range computer system and are limited to enable a very low bandwidth usage, coupled with a low polygon count system. They also use simple navigation techniques based on mouse and keyboard interaction. These types of systems take advantage of the users being unable to use advanced equipment and are therefore ideally suited to the needs of such users. In situations where there is a requirement for greater and more advanced interaction, these systems are limited and therefore a more advanced system is required. This is in terms of the following items:

- More complex scenes with higher polygon counts
- Highly representative Virtual Humans
- More Intuitive Advanced Tracking Systems

Complex Scenes are required so that interaction is made easier; a faster recognition time by the user of objects, actions and the general scene provides better overall interaction with the Scene. Highly realistic Virtual Human representations are very important for perception and, coupled with an advanced Motion Tracking System, provide an extremely realistic representation of the user's movements.

In Section 2 we present the W-VLNET System and its main component parts, including the underlying architecture (including the changes made from the original). In Section 2.4 we introduce how our Motion Tracking System was directly linked with the W-VLNET system to create a completely interactive system. Section 3 concludes the design/implementation process with a Case Study of a student/teacher session of Virtual Dance, with Section 4 concluding the paper.

2 The NVE System

VPARK is a framework for distributed VE applications where human-like embodiments represent real humans. The W-VLNET system is responsible for loading and managing scenes for connecting users.

2.1 Overall System Architecture

The W-VLNET system is based on a previous architecture implementation [5] running on the UNIX OS (specifically IRIX, from Silicon Graphics). The previous system, also called VLNET, was written and optimised for UNIX and uses a multi-process/shared memory architecture. Shared Memory was used as the communication medium for processes (each process performing a different task). Although the W-VLNET system is similar, the underlying communication architecture was redesigned due to the fact that the Windows OS does not fully utilize its Shared Memory architecture. Therefore a new architecture was implemented using a special communication system and the concurrent task management was done using

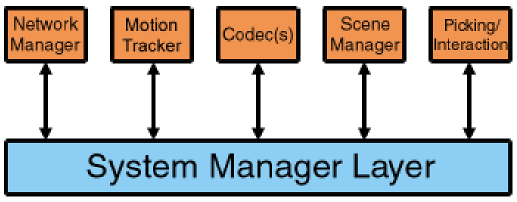

Fig. 1. System Communication

threads.

Threads are more dominant in the Windows OS and there are fewer restrictions imposed upon the design, as they do not require shared memory for inter-thread communication. The control and execution of threads was combined with the communication architecture to create a faster communication and execution mechanism. The communication is performed using a First In First Out Buffer (or FIFO Buffer), which controls the communication data flow, without limiting the data transfer. The FIFO Buffer basically allows each thread to communicate with any other thread.

The Thread Manager itself creates and controls the termination of a thread for each task that has to be executed. Depending on the task type, a priority value is assigned to each thread to enable tasks that should be executed quickly, not to be blocked by simple tasks (such as GUI Control). The Thread Manager limits the number of concurrently running threads, as too many threads would cause a system to slow down. In practical tests the upper limit is much greater than actually required.

The remaining control for each thread is left to the module to determine and manage, as it is unrealistic for the Thread Manager to be too specific to each task. These control tasks include mutual exclusion, global memory control and wait states. As threads are capable of being distributed across these multiple processors, the only requirement of the individual modules themselves is to be separated enough so that the multi-processor architecture is made use of and thus not creating dependencies. Figure 1 shows the communication between the main modules running through the System Manager (which is the collective of the Thread Manager and the FIFO Buffer)

2.2 Plugins

The entire system was designed and built around plugins; even the communication/thread managers were both designed with plugins in mind. The system was designed to be expandable, this being a key issue in the previous system (i.e. the inability to expand easily). The plugins used in this system are much the same as any other plugins; they enable the adding, changing and editing of any module without the need for recompilation. Also as all the main components are also plugins, the system can be upgraded without the user requiring major changes to the software.

Users are actively encouraged to design plugins for the system to perform a specific task they might require. An SDK is available, which is designed to aid users in understanding the plugin concept specific to this system.

2.3 Scene Management

In the same way that the System Manager (Section 2.1) controls the System, the Scene Manager controls all the aspects of the Scene. The Scene itself is quite complex and although OpenGL Optimizer [6] controls the actual Scene Graph, there are many additional interactions that need to be taken care of. As there are multiple Clients with multiple Avatars, the database used to manage an NVE Scene Graph is more complex. The system is no longer (as with normal Virtual Environments) controlling an Avatar in a one dimensional array (e.g. Avatar 1,2,3 etc), the Avatars are now multiple per Client and therefore reside in a two dimensional array (Avatar 1 on Client 1, Avatar 1 on Client 2), as each Client can own and control more than one Avatar. This causes more complications in the overall design, especially as the W-VLNET system has highly concurrent tasks being processed throughout its architecture. 2.3.1 Database Control and Object Loading

The database used has two layers: The Client Layer and the Item Layer. The Client layer contains a very simple reference to the Client. The Item (an Item being either an Object or an Avatar) Layer, which is below this, contains the references for all Objects and Avatars in the system. This includes their name, scene graph reference(s) and tasking locking switches. It is important to keep track of these objects and avatars in a very strict fashion as many complex things can happen (e.g. a Client could leave/join, crash, become disconnected etc) and this can have a very adverse effect if not handled correctly. Secondly, as mentioned in Section 2.1, concurrent tasks can be performed at once on the same Object or Avatar, and hence it is necessary to keep track of whether an Object/Avatar is being interacted with. The inability to keep track of these events will also have strange outcomes (such as strange animations, or objects/avatars ending up in different positions on different Clients). Objects are loaded into the Database and Scene Graph using the VRML97 [7] loader in OpenGL Optimizer. Therefore any VRML97 objects produced by commercial software can be loaded into the Virtual Environment and used by all other participating Clients.

2.3.1 Avatar Loading and Animation

A real sense of presence within a Virtual Environment is extremely important and the use of Virtual Human Representatives aids in this reality perception. Therefore, in line with the use of standards, HANIM/MPEG4 [8] compatible bodies are used in conjunction with MPEG4 compatible faces [9] to produce Virtual Representative Humans (or Avatars). Each Client is expected to load at least one representative avatar, which also has to be uploaded to the Server and distributed to the other connected Clients (See Section 2.5). The Avatar files themselves are compressed into zip files, making the transfer to the Server lighter in comparison with uncompressed files (normally 7-8 times larger), but even these files are between 600K and 1M and hence a caching mechanism was also implemented to reduce wait times and bandwidth utilization. The caching mechanism works in two ways; firstly it acts in the normal way, which is to check if a copy of the file exists locally (this caching mechanism also works for object files) and then just transfers the basic information (like posture/position), which is extremely small in comparison. The second caching mechanism is used if the user has a low bandwidth connection to the server; it basically uses a default avatar representation (also stored locally) for all avatars, hence reducing the requirement to download other client's representative avatar.

MPEG4 Compatible Animation Parameters are used to animate the Avatar. Face Animation Parameters (or FAPs) are used to describe a movement of key feature points on the face (there being 66 key feature points in the MPEG4 specification). Body Animation Parameters (or BAPs) specify the relevant body joint degree of freedom at a given instant in time

Animation on the NVE System is done completely on a frame/frame basis. Each frame (of either BAPs and FAPs) is compressed using a simple loss-less compression technique, and using a sequential numbering system is sent directly to the Server and distributed to other clients. This means that a Client can stop its animation at any time, or adjust it accordingly; there is no set time for which an animation can last. This works equally well for both file animations (animation streams stored in files) and for Motion Tracking Units (see Section 2.4). The loss-less compression is used to reduce the overall packet size of a body animation (as the animation of all the joints can produce up to 296 values that are 4 bytes in size), combine this with other necessary data and the packet is almost 1Kbytes in size. As the normal Maximum Transmission Unit (MTU [10]) is 576 bytes, this is too large for normal Internet transmissions (where the restricted MTU size is often observed). The loss-less compression uses the upper and lower limits of each of the 296 values and reduces the size of each value to the maximum bit value required. Also, even in the worst case, conditions only a maximum 110 values are used. Hence the packet can be compressed to roughly within the MTU restriction. A Quantizer value could be used to reduce this value, with the cost of reducing the accuracy of the animations, but a better approach would be to use either Huffman or Arithmetic Coding to produce better loss-less compression. Quantizing the values produces very little reduction in the packet size, at the cost of very poor animations and therefore is not used.

2.3.2 Picking and Object Manipulation

To really interact with virtual environment, it is necessary to use object picking and manipulation. Also as the system uses a collaborative environment then the object on one Client must be seen moving on all other Clients.

Picking is done on the basis of the Clients representative Avatar. The Avatar moves towards an object and then all objects within the View Frustum and within a specific range (variable, with default of 1 metre) are then selected as being pickable objects. The database of pickable objects is dynamically changed as the Avatar moves around. The pick mode then cycles through all pickable objects stored in the picking database and once the user has selected an object, it is then picked (selecting is done either by a button press, or by moving when the object is picked). The object is then moved with the Avatar as it moves (much in the same way as an object is moved in real life). The object can be deselected to unpick the object. All object movements are sent to the other Clients so that their database is completely up-to-date.

2.3.3 Proximity, Collision Detection and Gravity

In order to provide greater interaction within the virtual environment proximity detection was implemented. The proximity function is actually a collaboration of several common functions (which can be turned on or off as necessary, according to requirements and computing resources). The functions that are coordinated together are: Proximity, Collision Detection and Response, and Gravity. Although gravity is not directly combined into the same task, it does work hand in hand with Collision Detection and Response. Gravity is applied to each and every object/avatar apart from the basic scene (as defined by the Server as the default object); each object/avatars speed is stored in the database (as specified in Section 2.3.1) and a simple gravitational equation is applied to each object/avatar. This equation is designed to be fast (real time) and to move each object/avatar a large finite distance each time the equation is applied.

Proximity and collision detection is done in the same loop. The reason for this is that the collision detection function checks for all impending collisions of objects/avatars with other objects/avatars and then implements the response mechanism to prevent the actual intersection of the two objects or avatars. Proximity performs the former part of this calculation also, although more with respect to checking whether an object or avatar is less than a set distance away. The Proximity detection is used to enable greater interaction within the Scene (i.e. allowing objects to be more dynamic). Scene file specifies a proximity sensor to be applied either to an Object or an Avatar and a set of trigger conditions, (can be triggered by Object only, Object/Avatar or the local Users representation) plus the proximity distance, see Figure 2 .

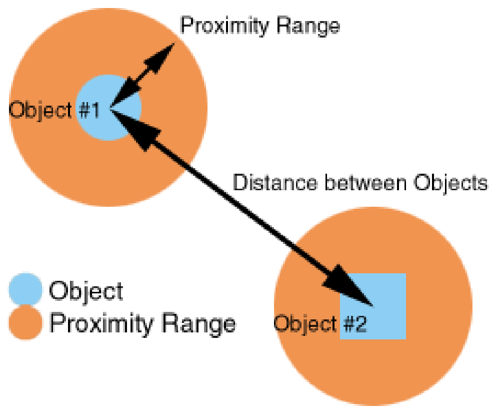

Fig. 2. Proximity Triggers

Collision Detection and Response at this time is extremely simple (to preserve the real-time aspect). To perform Collision Detection a Bound-Box is placed around each individual object and therefore simple intersections are detected. The response mechanism is currently designed only to stop an object/avatar from causing an intersection. This response mechanism and the application of gravity work in conjunction with each other, i.e. when the gravity mechanism is used, then the response mechanism must be implemented. More complex response mechanisms are expected in the future.

2.4 *Real-Time Motion Tracking*

2.4.1 *Body Posture and Tracking*

The real-time motion capture engine is based on a set of fourteen magnetic sensors (Figure 3). These sensors measure the motion of the major human limbs (head, spine, shoulders, elbows, hips, knees and ankles). Optionally, two digital gloves are used to track the wrists and fingers movements. The sensors' raw measurements are converted into anatomical angles suited to skeleton hierarchies using an efficient technique [11]. This converter is driven by orientation measurements to remove, as much as possible, dependencies on the distorted (non-linear) position measurements of magnetic sensors.

Only one sensor position is used to recover the position of the virtual human. The key features of this engine are:

- Automatic instant sensors calibration procedure.
- Human specific optimisations such as dedicated evaluation for shoulders and hips twisting, floor and anti-skating corrections.
- Control of the whole spine using three sensors (Figure 4).

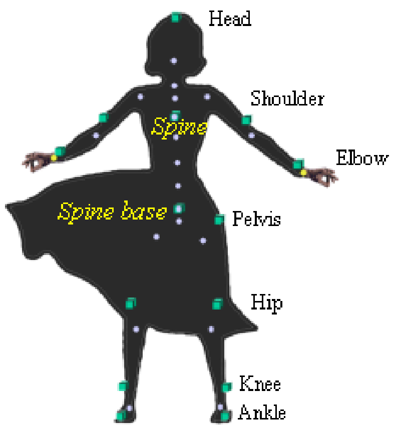

Fig. 3. Magnetic sensor locations

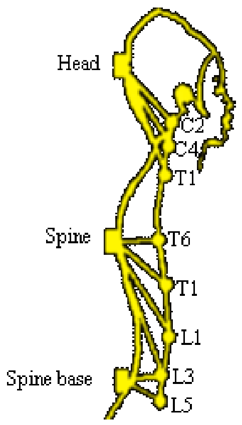

Fig. 4. Control of the spine using three sensors

The motion capture engine exists as a dedicated external application that sends BAPs to the W-VLNET core, which in turn applies the posture to the virtual human before final scene rendering. That way, the computational load is spread across separate processors. This introduces a slight lag (0.5s) between the performed movement and the rendered related posture, but we found it worthwhile in comparison to the pipelining solution where all steps are performed within the same application. In the latter solution, the lag varies between 0.3s and 0.7s depending on the rendered scene complexity.

2.4.2 MPEG-4 Body Animation Parameters

The human motion capture process is built on top of a proprietary skeleton structure [12] modelling joints using Euler angle sequence decompositions. These angles are very similar to the MPEG4 BAP. In order to animate MPEG4/HANIM hierarchies, the joint angles are translated from our internal hierarchy to the MPEG4 BAP. These computations basically consist of finding the MPEG4 counterparts (or indexes) for each joint angle, and applying simple data encoding (our internal an-

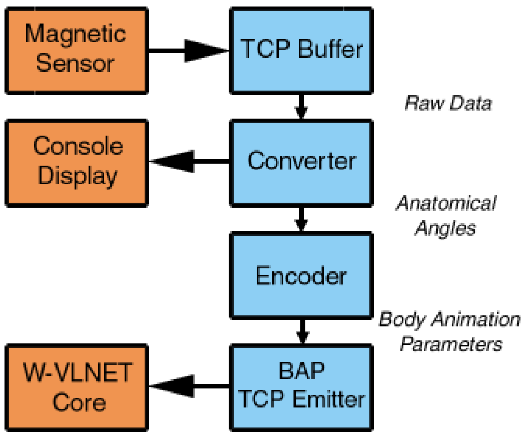

Fig. 5. External Motion Capture application processing pipeline.

gles are float values and MPEG4 parameters are encoded as 32 bit Integers). In few cases (e.g. fingers parameters), there is a slight posture difference between our internal model and the MPEG4 default postures. Consequently, we need to account for this default posture difference by adding angle offsets prior to encoding. By using key framing these offsets are identified by setting the proprietary hierarchy in the MPEG4 default posture by using key framing.

After encoding all parameters, the new posture information is sent to the client application using TCP for Communication (Figure 5). A simplified virtual human representation can be displayed within the external motion capture application to provide a diagnostic level feedback. This feature is mainly used to determine incorrect sensor positioning and other hardware related problems.

2.5 Networking

2.5.1 Overview

The network topology used for this NVE System is based on the Client/Server approach, as shown in Figure 6. This approach assumes all the Clients, requiring common interaction, connect to the same Server. Each Server hosts one or more Scenes, contains the master scene database and controls the distribution of data to all Clients.

2.5.2 Client Connection

Each Client connects to the Server via a single entry port. As soon as the connection is established the Server moves the Client to another port to keep the entry port free. The new port is established as the control port between the Server and the Client and is used as a secure data exchange for network control. The Server

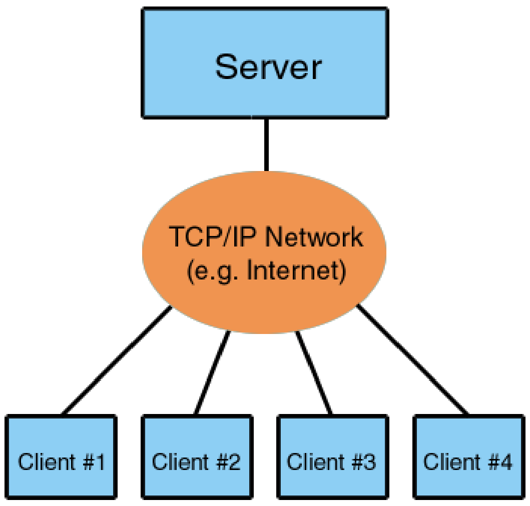

Fig. 6. Client/Server Architecture

then exchanges information with the Client to establish its identity, the channels it wishes to connect to and it also tests the data connection to determine a rough estimate of the bandwidth. The Server then sets up several channels according to the Client's request; these are as follows:

- Stream - Used for data that needs to be transmitted rapidly and at a steady rate. The channel does not retransmit data, in case of loss or error. The port is connectionless using UDP to transfer data.
- Update - This is similar to the Stream Channel, requiring only UDP connectionless port, but it has error control (using re-transmission), so data sent is treated with more care.
- File - Using a TCP connection-orientated port, File data (or very large data ¿ 1K bytes) is transferred over this port. Complete Error Control and Packet Re-send are implemented for this Channel.
- Control - Also using TCP, this channel is the one used during the Server/Client phase and stays connected until the end of the session.

The Client generally will need the Update Channel and it has no option for the Control Channel. However it can deny connection for the Stream and File Channels. This might be for a number of reasons: to preserve bandwidth, or CPU processing time, or the data transmitted is not required. The Update Channel transfers data such as Object/Avatar transforms and Avatar Animations. The Stream Channel is mainly used for Audio/Video Connections that might be required, for instance in sending real-time voice communication. Disconnection is done in reverse order, the Channels are disconnected first and then the Control Port sends a command to the Server to disconnect completely. The Server can force disconnection in the same way, which allows for a clean disconnection of ports and allows the Server to accept new connections without restarting. 2.5.3 Server Database

After the Client connects and downloads the main Scene, and once the main static Objects and Avatars (common throughout the Servers online status) have been downloaded into the Scene Graph, the Server Database is consulted to determine all the dynamic Objects/Avatars in the Scene. Each Client has the ability, at any time during connection, to add their Avatar(s), and Object(s) into the Scene to enable greater interaction.

The Scene Database contains the information on Objects and Avatars that have been uploaded by Clients, their transformation matrices, and the file reference in the Server's local Cache (which works in exactly the same way as the Client Cache). This information is distributed to each Client when a Client uploads the information. This information is also referenced when a Client connects for the first time, the Database is searched and all details are sent to the connecting Client to enable it to be completely up-to-date with the current state of the Scene. Avatars also have an extra field that stores the Avatars body posture. Both Audio and Video streams are not stored in the database as their data is dynamic.

When a Client disconnects, all uploaded Avatars are removed from the database (and corresponding messages distributed to all Clients); however, as Objects may still be in use by other Clients, the Objects move their ownership to the Server (which is Client 0) to avoid problems with later connecting Clients. 2.5.4 Communication Protocol

A common Communication Protocol is used over the Update Channel to enable simple message passing to exist. This protocol uses a generic packet that contains fields for common data types, and three generic fields provide access for other units.

- Message Type - Identifier for Message Packet, declares contents
- Animation Stream - 400 Bytes used for different types of animation and data (FAP, BAP, Text etc).
- Message String - 32 Byte Text Identifier (e.g. Filename)
- Message Value 1,2,3 - Used for general values and references.
- Transformation Matrix - 4 by 4 float value. Used because most objects/avatars will require transformations in nearly every packet.

The Stream Channel uses a simplified version of this, with a Message Identifier and a Transformation Matrix for each packet, then 500 Bytes of compressed data. The File Channel splits all data into manageable packets and then sends it directly over the channel. Waiting for an acknowledgement from the receiving end that all data was received correctly, otherwise a packet-by-packet re-send message is transmitted to the sending end. The Control Channel receives undefined messages regarding the state of the Server (connections, load, Client status etc). The Client then has a rough database of the connected clients (to reduce network load, the updates of the Client database are done on very large time steps).

2.6 Multimedia Objects

As can be seen from the Channel distribution, different types of data can be exchanged. The list of currently added data types/streams is as follows:

- Audio Stream - The basic stream of audio is transferred at 16Kbits/s and compressed using the G.728 [13] Audio Compression Codec. However for larger bandwidth systems, or systems with less bandwidth but greater CPU power, the G.711 (64Kbits/s) and G.723.1 (5.3Kbits/s and 6.4Kbits/s) audio codecs both use the same audio channel. Each audio stream is given a reference object in order that 3D Audio can be created.
- Speech - Speech communication over this type of system is useful for Clients connected over very low bandwidth connections. The Speech itself is transmitted as plain ASCII text and this is passed to a Text-to-Speech Engine [14]. This converts the text into phonemes and visemes, visemes being the corresponding lip movement of phonemes.

3 VPARK Case Study

The case study is a scenario for a Teacher (using the motion tracking system at one location) to teach dance to a student (also using one motion tracking system in another geographically remote location). Both teacher and student were cloned by our in-house cloning system, to obtain a virtual copy of both humans, and attached to a motion tracking system, as shown in Figure 7 (one at University of Geneva and the other at EPFL, Switzerland). An overlaid musical sequence is used to enable the teacher and student to synchronize with each other, both teacher and student can see each other (virtually) on a screen (as shown in Figure 8) and therefore the teacher is able to see what the student is doing wrong and the student can watch the teacher to see what should be done.

The system is fully interactive, allowing each participant the ability to see not only the exact movements of their counterparts, but also to talk with each other. This type of scenario is typical of an NVE System being used to its maximum benefit and certainly is difficult to replace with other conventional systems (such as Video Conferencing). The scenario is not limited to two participants; more users could join to provide a teacher with a class of students, providing the motion tracking equipment is available. To increase the teacher sense of submersion and also to enable a clearer perception of the situation a lightweight head- mounted display could be used, although as dance typically uses very jerky movements, hence the display should be rugged and should secure to the teacher so that the movement is not restricted.

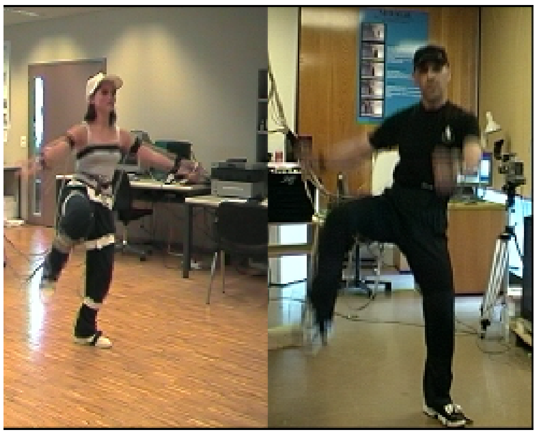
Fig. 7. Real Teacher and Student

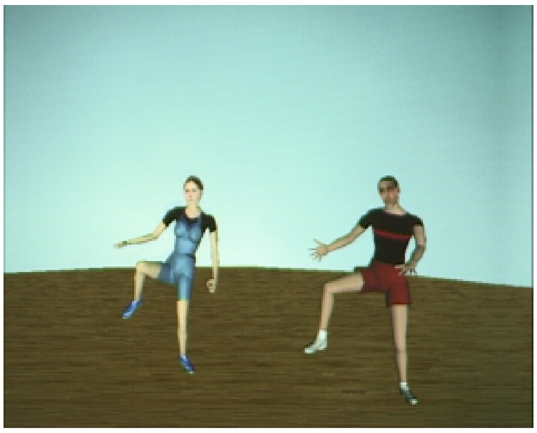
Fig. 8. Virtual Teacher and Student

4 Conclusion and Future work

In our work, we have presented the W-VLNET System, a powerful multithreaded system that is capable of running not only the Virtual Environment, but connecting two or more users together in a Networked Virtual Environment. The environment itself is made more real by the integration of an animation system that completely animates the user's virtual representation, and simple collision detection and response. This was coupled with our real-time body animation capturing system and full audio support for both sound and music, that adds to the realism of the experience. The W-VLNET System was designed and executed on Windows OS machine.

Finally, both systems actively use the latest standards for scene and virtual avatar representations (VRML97 and MPEG4) to enable greater interoperability between the two systems presented here and other commercial products.

To complete the work, we have designed, created and tested a complex dance scenario that allowed a distance teaching session to be implemented. This allowed us to visualize problems, prove the work in a real situation and finally to self-regulate ourselves and focus our research.

In future work we aim to augment the overall experience of the virtual environment by improving the collision models and improving the depth of the multimedia inputs (including video) and improving the audio perception in the environment. We also aim to improve the transmission rates for the system in real time using better compression methods and Server filtering.

5 Acknowledgements

The VPARK project is funded by the European Community (ACTS project Number AC353). We would like to thank Uwe Berner, Mireille Clavien, Nabil Sidi-Yacoub and Marlene Arevalo for their help during the case study demonstration and especially to Gabby Reider and Mahmood Davari for their dance performance.

References

[1] M. J. Zyda, D. R. Pratt, J.S. Falby, P. Barsham, K.M. Kelleher, ”NPSNET and the Naval Postgraduate School Graphics and Video Laboratory”, Presence, Vol.2, No.3, pp. 224 -258 (1993)

[2] C. Carlsson, O. Hagsand, ”DIVE - A Multi-User Virtual Reality System”, Proceedings of IEEE VRAIS’93, (1993)

[3] M.R. Macedonia, M.J. Zyda, D.R. Pratt, P.T. Barnham, ”NPSNET: A Network Software Architecture for Large-Scale Virtual Environments”, Presence; Teleoperators and Virtual Environments, Vol.3, No.4, (1994).

[4] Blaxxun Interactive - http://www.blaxxun.de/

[5] I. S. Pandzic, T. K. Capin, N. Magnenat-Thalmann, D. Thalmann, ”VLNET: A Networked Multimedia 3D Environment with Virtual Humans”, Proc. Multi-Media Modeling MMM ‘95 (ISBN 981-02-2502-4), (1995).

[6] Silicon Graphics - OpenGL Optimizer, http://www.sgi.com/software/optimizer/

[7] Virtual Reality Modeling Language (VRML), http://www.web3d.org/vrml/vrml.htm

[8] H-ANIM Humanoid Animation Working Group, Specification for a Standard Humanoid Version1.1, http://ece.uwaterloo.ca/ h-anim/spec1.1/

[9] W.S. Lee, M. Escher, G. Sannier, N. Magnenat-Thalmann, ”MPEG-4 Compatible Faces from Orthogonal Photos”, Proc. Computer Animation 99, pp.186-194, (1999).

[10] W. Richard Stevens, TCP/IP Illustrated - Volume 1, 1994, pp. 29.

[11] T. Molet, R. Boulic and D. Thalmann, ”Human Motion Capture Driven by Orientation Measurements”, Presence, MIT, Vol.8, No.2, 1999, pp. 101-115.

[12] R. Boulic, T. Capin, Z. Huang, L. Moccozet, T. Molet, P. Kalra, N. Magnenat-Thalmann, I. Pandzic, K. Saar, A. Schmitt, J.Shen and D. Thalmann, ”The HUMANOID Environment for Interactive Animation of Multiple Deformable Human Characters”, Proc. Eurographics’95, Maastricht, 1995, pp. 337-348.

[13] http://www.itu.ch

[14] S. Kshirsagar, M. Escher, G. Sannier, N. Magnenat-Thalmann, ”Multimodal Animation System Based on the MPEG4 Standard”, Multimedia Modelling 99, pp. 215-232, (1999)