Data Exchange in Networked Collaborative Virtual Environments

Igor Pandzic[1], Tolga Capin[2],
Nadia Magnenat-Thalmann[1], Daniel Thalmann[2]

[1] MIRALab - CUI
University of Geneva
24 rue du Général-Dufour
CH1211 Geneva 4, Switzerland
{Igor.Pandzic,Nadia.Thalmann}@cui.unige.ch
http://miralabwww.unige.ch/

[2] Computer Graphics Laboratory
Swiss Federal Institute of Technology (EPFL)
CH1015 Lausanne, Switzerland
{capin, thalmann}@lig.di.epfl.ch
http://ligwww.epfl.ch/

Abstract

One of the ultimate goals of SNHC is to provide a framework supporting the Networked Collaborative Virtual Environments (NCVE) for a wide range of applications. We briefly present an existing NCVE system called Virtual Life Network (VLNET). VLNET is a NCVE system incorporating realistic representation and animation of virtual humans for higher realism of simulation and more natural interaction with users of the system, as well as with autonomous agents. Based on our experience in VLNET we analyze the requirements of NCVE systems that should be considered by SNHC. In particular, we examine in detail various types of data traveling through the network in this type of application, and requirements for each type.

Introduction

In its ultimate extension, SNHC will provide a framework to support Networked Collaborative Virtual Environments (NCVEs) integrating multiple graphically represented users, 2D, 3D, audio and video objects for a wide range of applications ranging from shopping to health care [Doenges 96]. NCVEs represent an active area of research and several systems. Several systems have been developed [Barrus 96, Capin 95, Carlsson 93, Macedonia 94, Ohya 95, Singh 95, Thalmann 95], varying greatly in networking solutions, scalability, application scope, user embodiment solutions, efforts to integrate natural video and/or audio data. Nevertheless, whatever the network topology of the NCVE (client/server, multicast, multiple servers, combinations [Funkhouser 96]) the data that is being communicated from one participant in the simulation to the other(s) is essentially similar. Based on the example of our Virtual Life Network system, we analyze the types of data communicated in NCVE systems.

The next chapter provides an extensive introduction to VLNET. VLNET is a NCVE system exploiting highly realistic Virtual Humans to represent users as well as autonomous agents in the simulation. The bodies and faces of Virtual Humans, as well as other objects in the environment, can be controlled by exchangeable external drivers connected to a set of simple interfaces. This concept provides a simple and flexible means of controlling the otherwise complex system. To insure even more flexibility in development, and to improve performance, VLNET internal architecture is also extremely modular, with a number of processes working on precisely defined tasks.

After the introduction to VLNET, in the following chapter we explore in detail the types of data used for communication in VLNET, analyzing the scope, requirements, implementation.

Finally we give conclusions and ideas for future work.

Virtual Life Network

Virtual Life Network (VLNET) [Capin 95, Pandzic 96, Thalmann 95] is a Networked Collaborative Virtual Environment system using highly realistic Virtual Humans for the participant representation. The Virtual Humans in VLNET are completely articulated deformable bodies with articulated faces. They can faithfully reproduce body postures and facial expressions and, combined with appropriate real-time animation techniques, reproduce natural motions, actions, emotions and speech. Through a set of external interfaces, VLNET provides flexible means of controlling not only the users' embodiments but also other objects in the scene. This allows easy support of various input devices, as well as straightforward connection to external control programs simulating autonomous behaviors or animating the virtual world based on external data sources (e.g. on-line stock exchange data, weather data etc.). While providing easy access through external interfaces to all interesting functionalities, VLNET performs the basic functions like networking, database management and rendering in a manner transparent to the user.

From the networking point of view, VLNET is a based on a fairly simple client/server architecture. The server is mostly responsible for session management and message distribution. It is designed to work in pair with a standard HTTP server for database distribution. The design of the VLNET client is highly modular, with functionalities split into several processes. This allows not only performance improvements, but also the possibility to easily replace certain modules and obtain different functionalities.

Next two sections discuss in more detail the server and client architecture.

VLNET Server

A standard HTTP server and a VLNET Connection Server have to run permanently on a VLNET server site. They can serve several worlds, which can be either VLNET files or VRML 1.0 files. For each world, a World Server is spawned as necessary, i.e. when a client requests a connection to that particular world. The life of a World Server ends when all clients are disconnected.

Figure 1 illustrates the connection of clients to the VLNET server site. A VLNET session is initiated by a Client connecting to a particular world designated by a URL. The Client first fetches the world database from the HTTP server using the URL. After that it extracts the host name from the URL and connects to the VLNET Connection Server on the same host. The Connection Server spawns the World Server for the requested world if one is not already running and gives the Client the port address of the World Server.

The Client can provide the user data (the files describing the body the user wants to be represented with) by sending a URL. This data is distributed to other Clients in the session. The Client also has to fetch the user data from the other Clients. Once the connection is established, all communication between the clients in a particular world passes through the World Server.

The World Server manages the communication intelligently by distributing the incoming data to the clients on an as-needed basis. This means for example that a video bitstream is not transmitted to a client who is currently not looking at the video object.

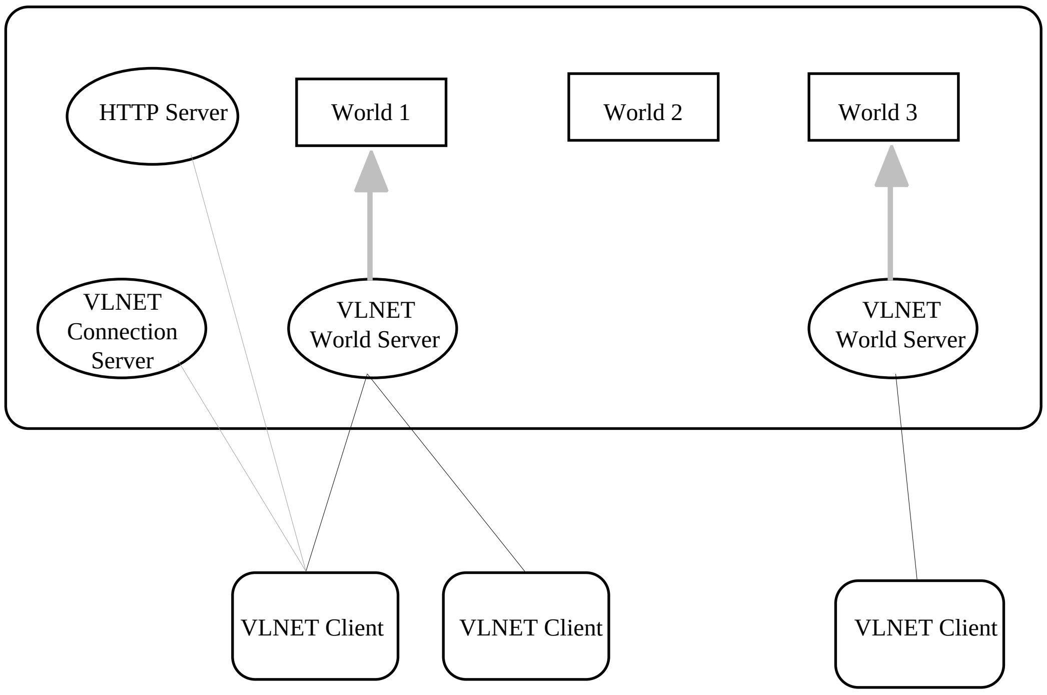


Figure 1: Connection of several clients to a VLNET server site

VLNET Client architecture

The design of the VLNET Client is highly modular, with functionalities split into a number of processes. Figure 2 presents an overview of the modules and their connections. VLNET has an open architecture, with a set of interfaces allowing a user with some programming knowledge to access the system core and change or extend the system by plugging custom-made modules, called drivers, into the VLNET interfaces. These drivers only have to use a defined API to connect to VLNET. They can run on the local host, or on a remote host. The VLNET core consists of a number of processes performing the basic functions like object updating, rendering, networking. These processes communicate through shared memory. The VLNET main process consists of four logical units, called engines, each with a particular task and an interface to external applications (drivers).

The VLNET Core

The VLNET core is a set of processes, interconnected through shared memory, that perform basic VLNET functions. The Main Process performs higher level tasks, like object manipulation, navigation, body representation, while the other processes provide services for networking (Communication Process), database loading and maintenance (Database Process) and rendering (Cull Process and Draw Process).

The **Main Process** consists of four engines covering different aspects of VLNET. It also initializes the session and spawns all other processes and drivers. Each engine is equipped with an interface for the connection of an external driver.

The **Object Behavior Engine** takes care of the predefined object behaviors, like rotation or falling, and has an interface allowing to program different behaviors using external drivers.

The **Navigation and Object Manipulation Engine** takes care of the basic user input: navigation, picking and displacement of objects. It provides an interface for the navigation driver. If no navigation driver is activated, standard mouse navigation exists internally. Currently, navigation drivers exist for the SpaceBall and Flock of Birds/Cyberglove combination. New drivers can easily be programmed for any device.

The **Body Representation Engine** is responsible for the deformation of the body. In any given body posture (defined by a set of joint angles) this engine will provide a deformed body ready to be rendered. The body representation is based on the Humanoid body model [Boulic 95]. This engine provides the interface for changing the body posture. A standard Body Posture Driver is provided, that connects also to the navigation interface to get the navigation information, then uses the Walking Motor and the Arm Motor [Boulic 90; Pandzic 96] to generate the natural body movement based on the navigation. Another possibility is to replace this Body Posture Driver by a simpler one that is directly coupled to a set of Flock Of Birds sensors on the users body, providing direct posture control.

The **Facial Representation Engine** provides the synthetic faces with a possibility to change expressions or the facial texture. The Facial Expression Interface is used for this task. It can be used to animate a set of parameters defining the facial expression. The facial representation is a polygon mesh model with Free Form Deformations simulating muscle actions [Kalra 92].

All the engines in the VLNET core process are coupled to the main shared memory and to the message queue.

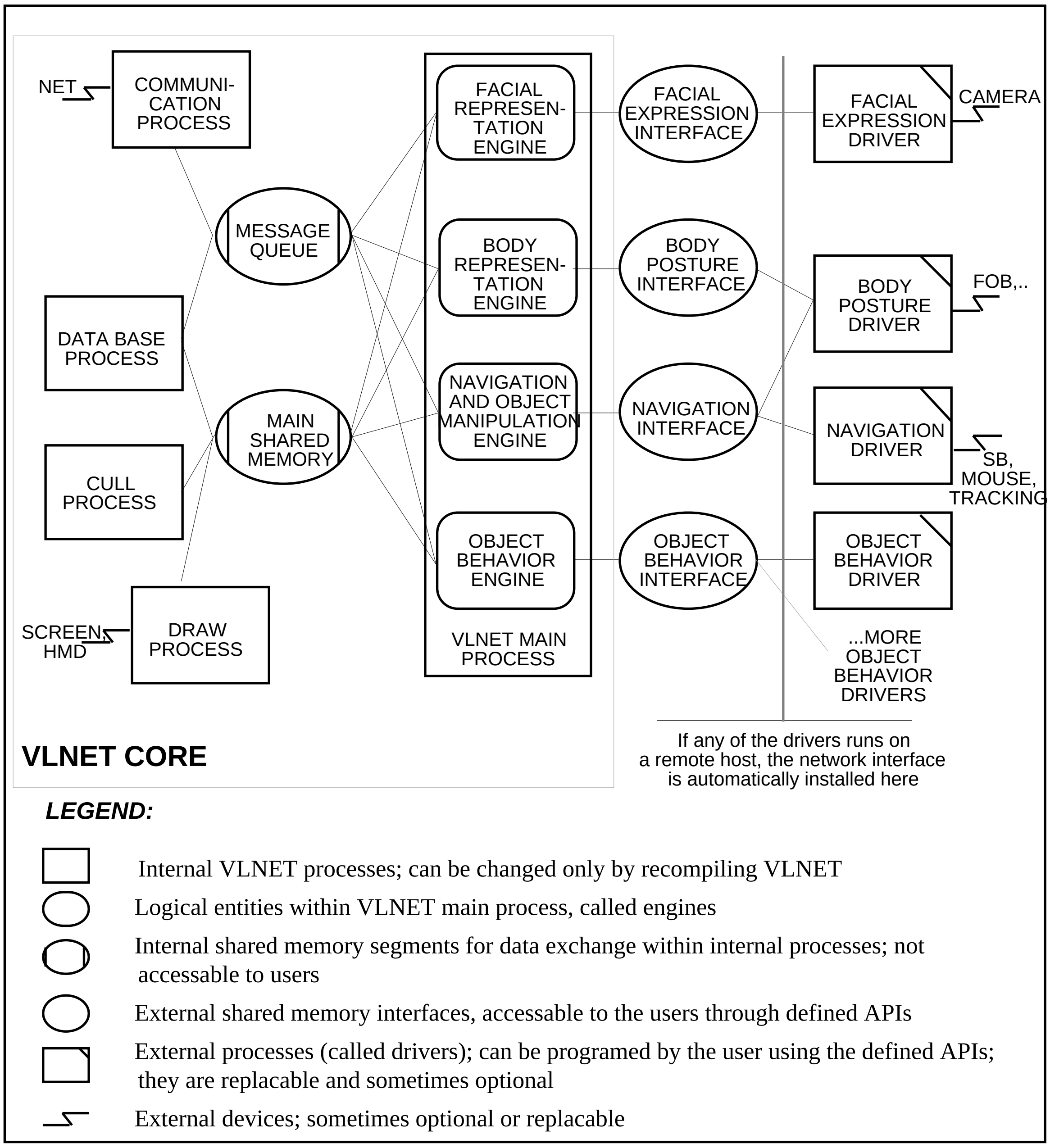
Cull and Draw Processes access the main shared memory and perform the functions of culling and drawing as their names suggest. These processes are standard SGI Performer [Rohlf 94] processes.

The **Communication Process** receives messages from the network (actually from the VLNET World Server) and puts them into the Message Queue. All the engines read from the queue and react to messages that concern them (e.g. Navigation Engine would react to a Move message, but ignore a Facial Expression message which would be handled by the Facial Representation Engine). All the Engines can write into the outgoing Message Queue, and the Communication Process will send out all the messages. All messages in VLNET use the standard message packet. The packet has a standard header determining the sender and the message type, and the message body. The message body content depends on the message type but is always of the same size, satisfying all message types in VLNET.

The **Data Base Process** takes care of the off-line loading of objects and user representations. It reacts to messages from the Message Queue demanding such operations.

The Drivers

The drivers provide the simple and flexible means to access and control all the complex functionalities of VLNET. Simple, because each driver is programmed using a very small API that basically consists of exchanging crucial data with VLNET through shared memory. Flexible, because using various combinations of drivers it is possible to support all sorts of input devices ranging from the mouse to the camera with complex gesture recognition software, to control all the movements of the body and face using those devices, to control objects in the environment and to build any amount of artificial intelligence in order to produce autonomous or semi-autonomous agents in the networked virtual environment.



Internal VLNET processes; can be changed only by recompiling VLNET

Logical entities within VLNET main process, called engines

Internal shared memory segments for data exchange within internal processes; not accessable to users

External shared memory interfaces, accessable to the users through defined APIs

External processes (called drivers); can be programed by the user using the defined APIs; they are replacable and sometimes optional

External devices; sometimes optional or replacable

Figure 2: Virtual Life Network system overview

The Drivers are directly tied to the Engines in the VLNET Main Process. Each engine provides a shared memory interface to which a driver can connect. Most drivers are optional and the system will provide minimal functionality (plain navigation and manipulation of objects) without any drivers. The drivers are spawned automatically by the VLNET Main Process on the beginning of the session, based on the command line where all possible combinations of drivers can be specified. The drivers can be spawned on the local host or on a remote host, in which case the transparent networking interface processes are inserted on both hosts. In a simple case, as with most drivers shown in figure 1, a driver controls only one engine. However, it is possible to control more then one engine with a single driver, insuring synchronism and cooperation.

The Data Flow

At the bottom line, Networked Virtual Environments are about passing data from one user to another, or others, allowing this data to be input, processed and output in flexible ways with a high level of abstraction. Looking at figures 1 and 2 we can analyze the flow of data from one user to the other(s) and various steps of processing the data.

The user inputs the data through the drivers. At this level the data is classified into precise categories with defined interfaces: face, body, navigation and object behaviors. By replacing drivers, the user can use different devices or metaphors for the input.

From the drivers, the data passes to the engines, each specialized for a particular type of data. Here the data is used to perform appropriate actions if necessary (e.g. navigation).

The engines pack the data into a form appropriate for network transmission and pass it to the communication process. This level is completely transparent to the users. The communication process transmits the data to the server, which transmits it (if necessary) to the other clients' communication processes.

The communication process makes the data available to the engines. Each engine reacts to the data of its concern, unpacks it and represents it in appropriate form.

The database downloads are separated from this data flow and use the HTTP server.

The Network Traffic in VLNET

In this chapter we explore in detail the data types used for communication in VLNET. For each type of data we analyze the current and possible use, requirements, current implementation with any drawbacks and ideas for improvements based on our experience.

Figure 3 presents a classification of various types of network traffic happening in VLNET. The principal types of data are the VLNET message packets carrying the essential simulation data, download data, video, audio and text.

VLNET Message Packets

Use

The VLNET Message packets carry all the essential simulation data: session establishment messages, client/server negotiation, state updates, events. According to their use (and it will be shown later that it has an impact also on the requirements) we divide them into state, event and system messages.

System messages are used for session management and client/server negotiation. These are all the messages necessary for the client to establish the connection with the server, receive a user ID, send his personal information, receive information about other users, quit a session.

Event messages communicate events generated by the user, e.g. picking up and object or grouping objects.

The state messages update the state of the user or the objects. The MOVE message updates the transformation matrix of the object or user, thus changing the position, orientation and scaling. For a user, the change can concern the whole body, or separately head or hand positions. The JOINTS message updates the joint angles of the user's body, defining the body posture. The HAND_JOINTS message does the same for left or right hand. The FACE_EXP message updates the facial expression of the user.

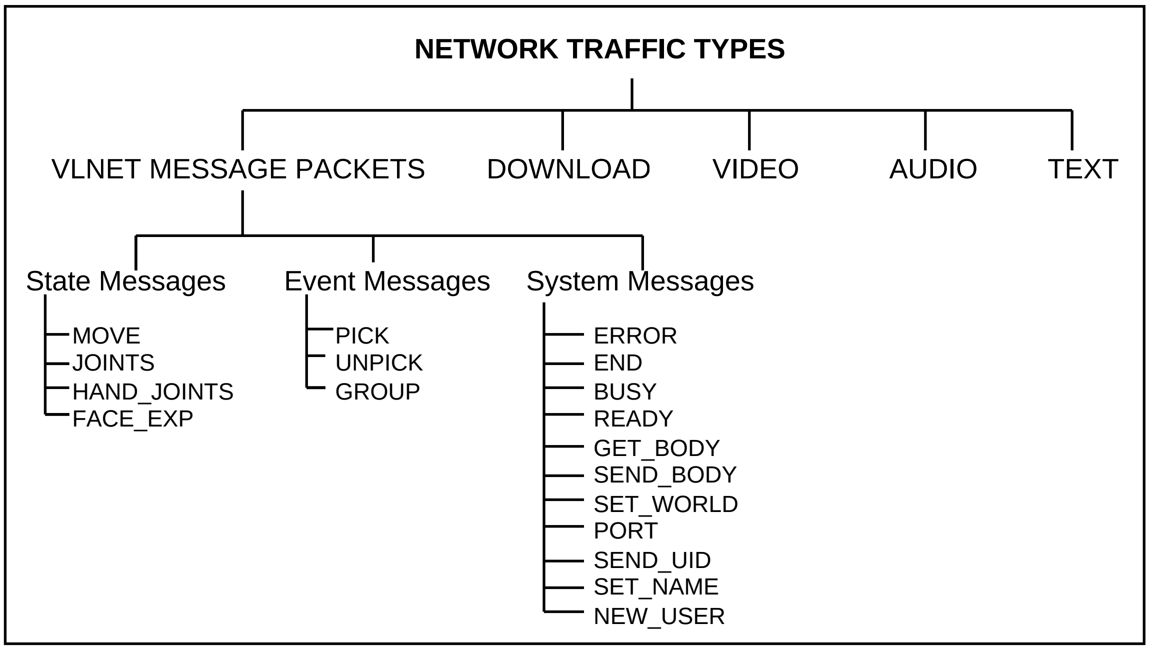


Figure 3: Network Traffic Types in VLNET

Requirements

The VLNET Message Packets (and similar messages in other systems) are small, typically under 100 bytes. Some of the event and system messages are even smaller, carrying just a flag (e.g. BUSY and READY messages indicating the client state to the server). Thus the bitrate generated by event and system messages is typically extremely low, aided also by the fact that these messages happen relatively rarely. However, the error resilience of both system and event messages must be maximal, in particular the system messages whose loss can cause complete malfunction of the system.

For the State messages the situation is slightly different. Though they are also small, they do carry more information then event and system messages. Also, they happen much more often. As a result, they generate a higher bitrate, though still in the order of tens of kilobits maximally, in case of very high client activity. Each message outdates and replaces the previous message of the same type: for example, once the new face expression is received, the last one is obsolete. Thus a loss of one message is not crucial, and will cause only a very temporary function flaw - up to the moment when the next message is received correctly. Therefore the error resilience doesn't represent such a big issue for these messages as for the previous types.

Latency is not a critical issue, but should be small for event and update messages.

Implementation

The VLNET Message Packets are uniformly sized (80 bytes). They have a header containing the sender ID and the message type. The body of the message is interpreted according to the type (this is implemented as a union in C). The uniform message size and header provide great benefits for the internal message handling, especially when messages are passed between processes through shared memory, because all the messages can be treated in the same way. Once the messages reach the communication process (see figure 2), the benefit of uniformity is spoiled by the efficiency issue. Clearly it is very inefficient to use a big message packet just to send a one-bit flag. This is currently tolerated because of the ease of implementation - all the messages can be treated in the same way for sending and receiving. Nevertheless, by processing the messages more intelligently at this level lower bitrate and higher error resilience can be achieved.

For some of the state messages, we have successfully used the dead-reckoning technique with a big save in bitrate (HOW MUCH TOLGA?).

For the event and system messages, higher security can be achieved by treating them with a better networking protocol.

In the current implementation, the actual parameters used to define body postures and facial expressions correspond to those proposed in the response to the SNHC Call for Proposals [MPEG-N1195] and described in detail in our submissions to the CfP [MPEG-M1211, MPEG-M1165]. Therefore they are very similar to the SNHC draft for those parameters [MPEG-N1365] which was based on these and other proposals.

Download

Use

Download is used for all database transfer to the client, as well as for the transfer of user's body representation data to other users. The data consists typically of 3D geometry and textures.

Requirements

The size of files to be downloaded depends on the complexity of a particular virtual environment or body representation. Potentially it can be very large, easily it reaches megabytes. Errors can not be tolerated, because even a small error in a downloaded file can cause a bad read. It is obviously desirable that the download time should be as short as possible, within limits of the requirement on error-free download.

Implementation

All downloads in VLNET are performed using the HTTP protocol. This allows also to use the familiar HTTP servers for data exchange. Various data formats are supported for 3D object data, including most of the popular formats.

In this domain there is a great potential for improvement using 3D data compression techniques for 3D objects, and image compression for textures. When using the 3D compression techniques, the notion of error tolerance also changes, because it moves into the 3D object domain. It is possible to define error tolerance in terms of coordinate errors for vertices, normals etc. However, compression techniques are usually tied to a particular data format, implying that a conversion is necessary before compression if multiple data formats are to be supported.

Video

Use

In VLNET the use of video is currently limited to facial communication [Pandzic 96-1]. In our approach the video sequence of the user's face is continuously texture mapped on the face of the virtual human. The user must be in front of the camera, in such a position that the camera captures his head and shoulders. A simple and fast image analysis algorithm is used to find the bounding box of the user's face within the image. As illustrated in figure 4, this provides effective means of facial communication in real time, transmitting facial expressions and lip movements and displaying them on the face of the Virtual Human representing the user.

Requirements

Using small images and compression, the bitrate is reduced to approx. 150 Kbit/sec, but this is still huge in comparison with the rest of the data. This data is also less critical in terms of error tolerance. A very important requirement on the implementation is that most of the other data should be treated preferentially to video, i.e. it is not acceptable for a system message to be retarded because of the video.

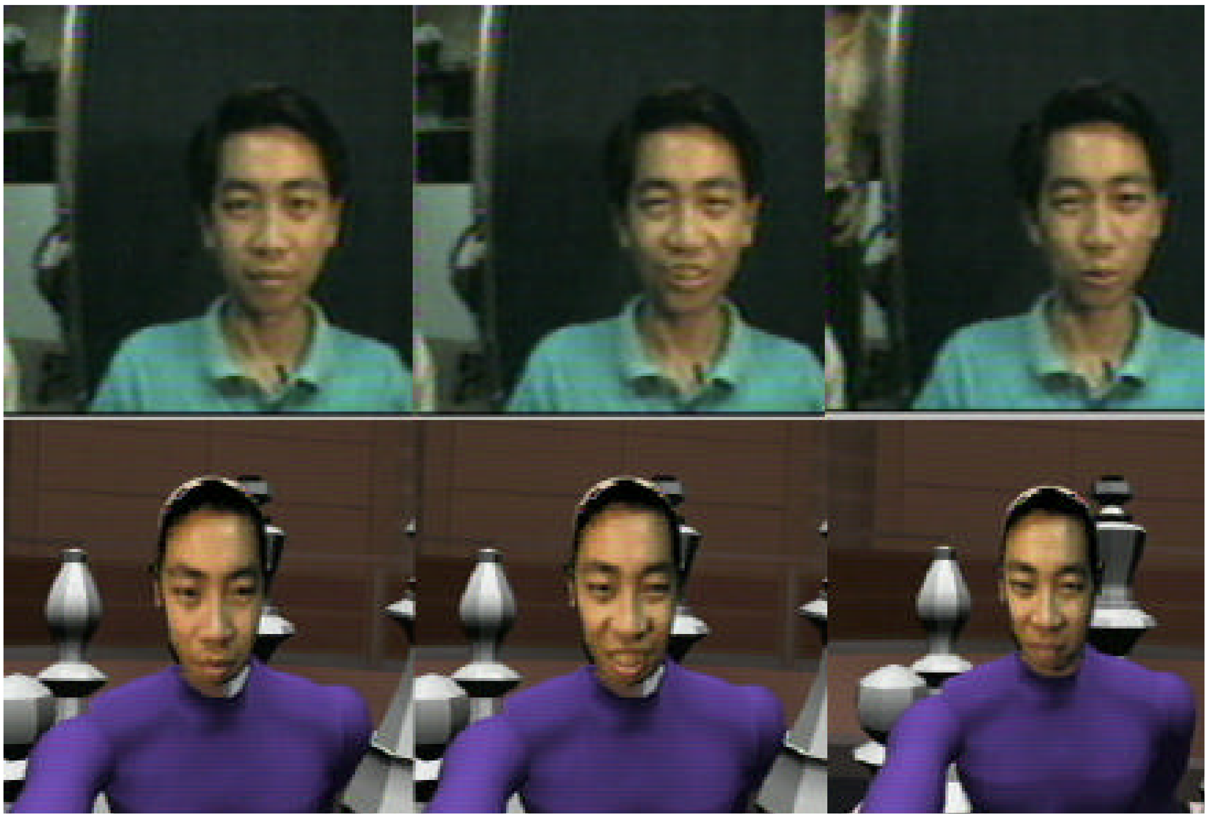


Figure 4: Video texturing of the face

Implementation

Each facial image in the video sequence is compressed using SGI Compression Library and the compressed images are passed to the Facial Representation Engine of VLNET, then redirected to the Communication Process. Obviously, the color images of 120 x 80 pixels, even compressed, do not fit in the standard VLNET Message Packets used by the Communication process. To avoid multiplexing of the VLNET Message packets and the video data we use a separate channel (socket) for the video. This allows preferential treatment of the VLNET packets, without a more complex multiplexing. Nevertheless, opening separate communication channels complicates the system.

Audio

Use

Audio communication is currently used only in the very classical sense: talking to each other.

Requirements

Audio produces a medium, steady bitrate when active. In the conversation setup, typically there will be pauses in speech of each speaker which can be detected and no bitrate generated during the pauses.

Implementation

Currently we use an external public domain software for the audio communication. We are working on an implementation using a separate audio channel, similar to the implementation of video.

Text

Use

The obvious use of text communication is the chat using text messages. However, in NCVEs text can be used for more then that. Combining the facial animation based on text with a Text To Speech system simple text can be transformed into speech accompanied with appropriate lip movement. Text is also used to communicate with autonomous agents.

Requirements

Bandwidth needed for text transfer is low. Occasional errors can be tolerated. Considering the typing speed, latency is not such a critical issue.

Implementation

Since the text has low requirements, it is easy to squeeze it in the same channel with the video.

Conclusions and future work

We have presented the Virtual Life Network system and used it as an example to analyze various types of network traffic appearing in the Network Collaborative Virtual Environment systems. For each type of data we have discussed the use, networking requirements and implementation. Obviously, NCVE systems generate a broad variety of network traffic types with widely varying requirements in terms of bitrate, error resilience and latency. Currently we solve this problem by using different data channels, possibly with different transmission protocols or even driven by different applications, for major data categories. A powerful multiplexing algorithm with a possibility of specifying requirements for each incoming data type would be a more effective solution. Another interesting domain for future work is the integration of 3D geometry compression techniques within NCVE systems, which would greatly reduce session establishment times.

References

[Barrus96] Barrus J. W., Waters R. C., Anderson D. B., "Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments", *Proceedings of IEEE VRAIS*, 1996.

[Boulic 90] Boulic R., Magnenat-Thalmann N. M.,Thalmann D. "A Global Human Walking Model with Real Time Kinematic Personification", *The Visual Computer,* Vol.6(6),1990.

[Boulic 95] Boulic R., Capin T., Huang Z., Kalra P., Lintermann B., Magnenat-Thalmann N., Moccozet L., Molet T., Pandzic I., Saar K., Schmitt A., Shen J., Thalmann D., "The Humanoid Environment for Interactive Animation of Multiple Deformable Human Characters", *Proceedings of Eurographics '95*, 1995.

[Capin 95] Capin T.K., Pandzic I.S., Magnenat-Thalmann N., Thalmann, D., "Virtual Humans for Representing Participants in Immersive Virtual Environments", *Proceedings of FIVE '95,* London, 1995.

[Carlsson93] Carlsson C., Hagsand O., "DIVE - a Multi-User Virtual Reality System", *Proceedings of IEEE VRAIS '93,* Seattle, Washington, 1993.

[Doenges 96] Doenges, P.K., Capin, T.K., Lavagetto, F., Ostermann, J., Pandzic, I.S., Petajan, E.D., "MPEG-4: Audio/Video & Synthetic Graphics/Audio for Mixed Media", Image Communication Journal (to appear)

[Funkhouser 96] Funkhouser T.A., "Network Topologies for Scalable Multi-User Virtual Environments, *Proceedings of IEEE VRAIS '96,* 1996

[Kalra92] Kalra P., Mangili A., Magnenat Thalmann N., Thalmann D., "Simulation of Facial Muscle Actions Based on Rational Free Form Deformations", *Proc. Eurographics '92*, pp.59-69., 1992.

[Macedonia 94] Macedonia M.R., Zyda M.J., Pratt D.R., Barham P.T., Zestwitz, "NPSNET: A Network Software Architecture for Large-Scale Virtual Environments", *Presence: Teleoperators and Virtual Environments,* Vol. 3, No. 4, 1994.

[MPEG-N1195] MPEG4-SNHC Call for Proposals, ISO/IEC JTC1/SC29/WG11 N1195, MPEG96/March 1996.

[MPEG-N1365] Face and body definition and animation parameters, Eric Petajan, Igor Pandzic, Tolga Capin, Pei-Hwa Ho, Roberto Pockaj, Hai Tao, Homer Chen, Janice Shen, Pierre-Emmanuel Chaut, Joern Osterman, ISO/IEC JTC1/SC29/WG11 N1365, MPEG96/October 1996.

[MPEG-M1165] Vidas submission to SNHC CfP on facial animation, ISO/IEC JTC1/SC29/WG11 M1165, MPEG 1996.

[MPEG-M1211] EPFL/University of Geneva: Body Representation Proposal, Ronan Boulic, Tom Molet, Tolga Capin, Igor Pandzic, Nadia Magnenat Thalmann, Daniel Thalmann, ISO/IEC JTC1/SC29/WG11 M1211, MPEG 1996.

[Ohya95] Ohya J., Kitamura Y., Kishino F., Terashima N., "Virtual Space Teleconferencing: Real-Time Reproduction of 3D Human Images", *Journal of Visual Communication and Image Representation,* Vol. 6, No. 1, pp. 1-25, 1995.

[Pandzic96] Pandzic I.S., Capin T.K., Magnenat Thalman N., Thalmann D., "Motor functions in the VLNET Body-Centered Networked Virtual Environment", *Proc. of 3rd Eurographics Workshop on Virtual Environments,* Monte Carlo, 1996.

[Pandzic96-1] Pandzic I.S., Capin T.K., Magnenat Thalman N., Thalmann D., "Towards Natural Communication in Networked Collaborative Virtual Environments", *Proceedings of FIVE '96,* Pisa, 1996. (to appear)

[Rohlf94] Rohlf J., Helman J., "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics", *Proc. SIGGRAPH'94*, 1994.

[Singh95] Singh G., Serra L., Png W., Wong A., Ng H., "BrickNet: Sharing Object Behaviors on the Net", *Proceedings of IEEE VRAIS '95*, 1995.

[Thalmann95] D. Thalmann, T. K. Capin, N. Magnenat Thalmann, I. S. Pandzic, “Participant, User-Guided, Autonomous Actors in the Virtual Life Network VLNET”, *Proc. ICAT/VRST ’95*, pp. 3-11.