Collaborative animation over the network

François Faure, Chris Faisstnauer, Gerd Hesina, Amaury Aubel, Marc Escher, Frederic Labrosse, Jean-Christophe Nebel, Jean-Dominique Gascuel

Collaborative animation over the network

F. Faure
C. Faisstnauer
G. Hesina
Vienna University of Technology, Austria
francois, faisst, gerd @cg.tuwien.ac.at

A. Aubel
École Polytechnique Fédérale de Lausanne, Switzerland
aubel@lig.di.epfl.ch

M. Escher
MIRALab, Geneve, Switzerland
Marc.Escher@cui.unige.ch

F. Labrosse
University of Bath, United Kingdom
F.Labrosse@maths.bath.ac.uk

J.-C. Nebel
University of Glasgow, United Kingdom
jc@dcs.gla.ac.uk

J.-D. Gascuel
iMAGIS, Grenoble, France
Jean-Dominique.Gascuel@imag.fr

Abstract

The continuously increasing complexity of computer animations makes it necessary to rely on the knowledge of various experts to cover the different areas of computer graphics and animation. This fact, which can be noted in many areas of scientific working, leads to increasing effort being put into research concerning cooperative working over the internet. However, it still requires substantial effort and time to combine different animation techniques in a common virtual environment.

When trying to perform collaborative animation over a network, we often face the problem of having to combine animation systems and applications based on different software and hardware and using incompatible data structures. We present an approach, based on a client-server architecture and employing a VRML-based language as common interchange format, that allows inhomogeneous systems to be easily incorporated into a collaborative animation. The applications can be freed from employing plug-ins or libraries to link into a common animation platform; they keep a local copy of the global scene and only need the ability to export the internal data representation into the so called "PaVRML" language, the language we use use to exchange data and synchronize the clients.

This approach does not only allow a number of practitioners to share their know-how within a common animation without requiring the huge amount of work necessary to port their application to a common platform. It also makes it often possible in the first place to combine the capabilities of different animation systems into a single complex animation. Additionally, we investigate solutions to optimize the network load for real-time applications. In this paper we present preliminary results and discuss the future developments of this ongoing work.

1 Introduction

This work arisen from the desire of several research teams to gather the computer graphics software they have within a common animation platform. Unfortunately, the code has been developed on different software platforms (OpenInventor, Alias Wavefront, SoftImage, Performer, *etc.*), and using different data structures. In order to link the different pieces of code within one application, it is necessary to port them to a common platform. The question of choosing such a platform appeared intractable, mostly due to the huge work necessary to adapt the code, and also because of incompatibilities between data structures.

We present an alternative solution based on the usage of a network to let remote (and inhomogeneous) applications communicate using a given protocol. All applications process a common database they replicate locally using their

own data structures; updates are sent and received to animate the scene and maintain consistency. In contrast with a traditional animation platform, our approach allows different animation tools to run on incompatible software and hardware platforms. This makes it possible for a number of practitioners to collaborate, each of them using their favorite tools.

An animation can be concurrently or incrementally designed using a variety of tools. Concurrent scene processing allows the animation of virtual worlds. Incremental processing allows the design of complex animations for movies. Consider the animation of a walking character. A certain degree of realism can be achieved by generating skeleton motion, facial animation and cloth simulation. Each of these issues requires highly specialized software which may not be available to a single user, unless he or she owns an expensive animation platform, and he or she is sufficiently skilled to exploit it. In our approach, such an animation is performed by combining the power of different tools, which may run either separately on a single machine, or remotely over the net. We call this paradigm collaborative animation.

Running concurrent applications able to communicate over the network is hardly a new idea. In the past years, many distributed virtual environments have been created. With respect to other platforms for virtual environment (e.g. Dive [7], Simnet [10], NpsNet [15], Aviary [18]), which require to write dedicated applications for them and are explicitly encoded in the application, our approach is intended to couple already existing applications to a virtual environment, with as less effort as possible. Thus the only interface between our applications and the virtual environment consists in sending and receiving messages.

The key point of our approach is the ability of sharing data between applications. We chose to use VRML97 as a data exchange language. Within a few years, it has become the most widely accepted standard for publishing and exchanging three-dimensional data. Its geometry modeling capabilities meet our needs and its wide diffusion makes it an attractive standard. Moreover, VRML97 has been chosen as the basis of the three-dimensional capabilities of the MPEG4 standard for broadcasting sound and images over the Internet. Since our architecture allows us to address the question of efficient streaming, the use of VRML97 opens a wide range of potential applications. More precisely, we have developed a new language, PaVRML[1] that is a subset of VRML97 augmented by commands allowing the creation and modification of objects during the animation (see Appendix A).

The remainder of this paper is organized as follows: first we summarize previous work on distributed virtual environments. Section 3 introduces the principle of our approach, followed by an investigation on applications for real-time environments in Section 4. Applications to perform collaborative animation are examined in Section 5. The paper is concluded with a discussion on future work. The Appendix presents PaVRML, our data exchange language based on VRML97, and briefly describes input/output processing as well as our network library.

[1]The acronym PaVRML comes from the fusion of two acronyms: PAVR and VRML. PAVR (Platform for Animation and Virtual Reality) is a European project in the context of which this research is done.

2 Previous work

A lot of research has been going on to build common virtual places in which users can interact with each other and also with responsive applications. It has resulted in a number of environmental platforms such as DIVE [7], VLNET [16], AVIARY [18], NPSNET [15], or DVS [3]. Years of research and experiments with or for these platforms have led to the adoption of a few techniques as ground rules for designing efficient networked virtual environments which use the Internet as communication medium.

These platforms deal with different network topologies (e.g. unicast, broadcast, multicast) and various optimizations (e.g. dead reckoning). It is important to choose a network topology that fulfills quality of service requirements like latency, and ensures scalability. Dead-reckoning (see [15, 6]) is a powerful optimization technique that has gained wide acceptance. It aims at reducing the number of messages being sent over the network. The basic idea behind it is that some information can be inferred at each site from last known information. As an example, the position of an avatar for frame n+1 can be easily inferred by all clients from its position/velocity in the previous frames (n, n-1 etc.). Thus, the client that actually controls the avatar may omit to transmit its position as long as the extrapolated position does not differ significantly from the actual one. This technique has been applied successfully to various types of data ranging from vehicles positions [15] to joint angle values used for avatar representation [6].

The major limitation of these toolkits is that they are available on few if not just one platform. One possible way to overcome this limitation is to rely on a cross-platform graphics library and produce a specific binary for each target platform, as illustrated in DIVE [7] or more recently with the Bamboo system [19]. Another usual limitation is that some toolkits fail to enable concurrent access, which makes them inherently inadequate for a real fruitful networked collaboration.

Over the past years VRML has become widely accepted as the file format for exchanging 3D data over the Internet. Its second release added more interactive capabilities and made it more suitable for multi-user collaboration. Since then, many companies and individuals have started to use VRML97 for electronic commerce. Applications allow-

ing users to perform collaborative work through the WEB also appeared. Some of them are based on specific plug-in/programs that include a VRML browser plus a set of functionalities for dealing with the network, avatar animation or shared object management. This has led to the emergence of digital communities on the Internet [2, 1]. By means of a proprietary browser, people all around the world can gather in digital worlds, own digital spaces, build digital cities, *etc.* However, a specific installation of the proprietary browser is required to gain access to such virtual worlds. In addition, as each system has its own communication protocol, interconnection between two worlds managed by two different applications is not possible. Another way to construct a shared world on the web is to establish a link between a VRML plug-in and Java. A Java program handles the network part and manages the world, while the world updates are made inside the VRML plug-in. This solution does not require any specific installation since the needed software (Java class) is down-loaded at run-time with the VRML world. To our knowledge, only the VNET system's architecture [5] was built in this fashion. Yet, this might change soon since a standard way to define and manage such VRML-based shared world is about to be specified [4], thus facilitating connections between different worlds. In our case though, most applications we wanted to connect together are C or C++ stand-alone programs. Since porting all the existing software to Java may not be such a trivial task, especially for C programs, and the efficiency of Java programs is still questionable, we investigate another solution.

Finally, the MPEG4 standard [9] which evolves towards complete inter operability with VRML proposes a generic way to encode/decode different kinds of media (audio/video objects) for data transmission. As it also aims to integrate coding of animated 2D/3D computer graphics, it could become the new standard for data transmission within shared world applications.

3 Principle

The structure of our system is composed of clients that communicate using a server, as depicted in Figure 1; a simple interface module allows the clients to send messages to and receive messages from the server. The task of the server is not only to minimize network traffic, but also to relieve the clients as much as possible from dealing with issues concerning the communication in a distributed virtual environment. Therefore the server does not simply route the messages coming from one client to all other clients, but manages a global database, containing all information about the common scene.

This global database is replicated at each client; upon login, the clients receive a model of the common scene to be stored and rendered locally. From then on, the communication between server and clients can mostly be limited to update messages. To contribute to the common scene, a client sends update messages to the server. The server updates its global database accordingly, and is then able to notify all other clients of these changes (if they are of concern to them). As this client-server architecture allows the clients to locally store and render a model of scene, they can use their own data structures and procedures; only the communication with the server must follow an agreed upon protocol. This allows to easily incorporate existing applications into the common scene; all that is required is the ability to send, receive and interpret the messages containing the scene description and the updates to that scene. As VRML97 is used to describe the geometry of the common scene, the communication with the server implies a translation from and into VRML; this translation is provided by the interface module. We also defined an additional set of commands to access the database (see Appendix A).

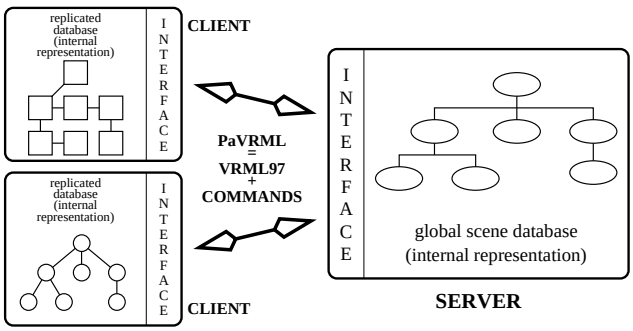


Figure 1. System architecture. A VRML scene is replicated within each client, which uses its own data structures. Conversion is achieved within the interface.

As an additional way to limit the network load we decided to give the clients the ability to control the flow of update messages coming from the server. Whenever the server receives update messages from a client to change the common scene, the other clients are not notified immediately, but only on demand: the server keeps an update log for each client and, as soon as the server receives a corresponding request, all updates ready for that client are transmitted to it. This approach has the disadvantage of making the inconsistency between the server's and the client's database also depend on the speed with which the client requests updates from the server, but on the other hand the client can choose the optimal speed for receiving the updates from the server and avoid to be overloaded with messages.

Although this project originated from the desire to achieve "collaborative animation" in a virtual environment, by having a number of research sites to cooperate on a com-

mon animation (each site contributing to that part of the animation it is specialized for), from our discussions among the different animation specialists we soon found out that we have to distinguish between two different ways of how to treat the notion of time in the animation. As an animation can be parameterized by time, it consists of an evolution of the scene over time. In our client-server approach, the clients specify the animation by determining the state of the common scene database at determined moments in time by sending update messages to the server. Depending on how the server deals with time, we distinguish between an *online* and an *offline* animation.

In an *online* animation, which is typical for virtual environments, the time (of the animation) is controlled by the server and is continuously progressing; the animation is progressing in real-time (as a virtual environment usually tries to simulate a real or imaginary world). To modify the animation, clients send scene updates to the server, which are processed upon receipt. The server notifies the other clients of these changes as soon as they request it. Usually there is no need to store a history of the environment, as only the actual state is of concern.

In an *offline* animation time is handled in a different way: time is controlled by the clients, the server does no time management. In fact, there is no progressing time at all. An offline animation is similar to a multi-track movie (*e.g.* recorded on a video recorder); the creation and "play back" of the animation are completely independent and can happen at different times. To create the animation, clients send scene updates to the server, specifying at which time they should happen. Time is relative to the first event in the animation; typically, the creation of the first objects. To view the animation, the clients ask the server to incrementally send them the state of the scene at given times. Thus, the whole animation must be stored by the server, which acts just like a common database. Offline animations are often used, *e.g.* in keyframe animations, when the animation must repeatedly be viewed and modified, played forward and backward, *etc.*, under the client's control.

4 The online server

According to the differentiation between *online* and *offline* animation, the server in our system can be operated in an *online* (or real-time) and *offline* (or VCR) mode. The online server is depicted in Figure 2. An online animation is specified while it is running, the change of the scene database and the viewing of the animation happen (almost) at the same time. When a client sends scene updates to the server, they are by default processed immediately; the clients can also specify a positive offset relative to the server's time (in the future) at which to process the updates. Negative offsets are not allowed, as the past of the animation

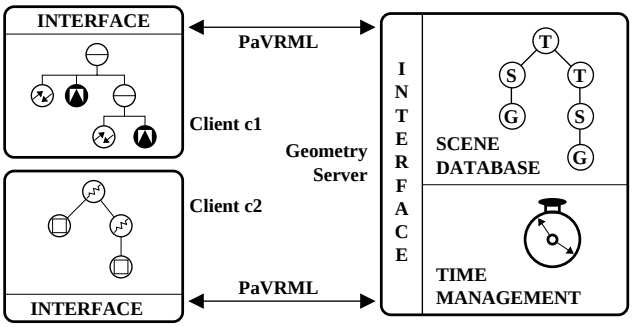


Figure 2. Structure of the online server (compare with Figure 4). The symbols T, S, and G respectively denote Transform, Shape, and Geometry nodes.

is not dealt with in a real-time animation. These changes to the scene database are distributed to all other clients after an explicit request; this allows the clients to perform a sort of flow control, although it may increase the inconsistency between the server's scene database and local databases of the clients. On the other hand, it allows us to reduce the number of messages transmitted over the network by discarding redundant updates. As the history of the scene is not of interest, the server stores only the most recent values of each object in the scene. Therefore, whenever a client requests information about the common scene, it receives only the updates describing the most recent state.

In the following example we use the messages sent from the clients to the server as trigger to get back all outstanding updates. Figure 2 shows the structure of the server operated in online mode, while a scene including a hierarchy of two shapes is edited dynamically by the clients. The server updates its graph according to the update messages received from the clients. Updates are sent on-demand back to the clients: each time the server receives a message, it replies a message containing information necessary to update the client's scene graph.

Figure 3 shows an example of graph creation and edition, following the structure of Figure 2. Starting from an empty graph, each client adds a shape. The arrows denote the direction of the messages.

Client c1 starts the session. It reads the current scene by sending a message and receiving the answer (steps 1 and 2). In this case, the scene is still empty. It then adds a sphere to the scene (step 3). When client c2 logs in, (step 5), it is notified the current state of the scene (step 6). It then adds a cylinder as a child of the sphere (step 7). In step 8, the server sends an empty message since the scene model of client c2 is up to date. However, the scene model of client c1 is now obsolete. When client c1 modifies the translation of the sphere

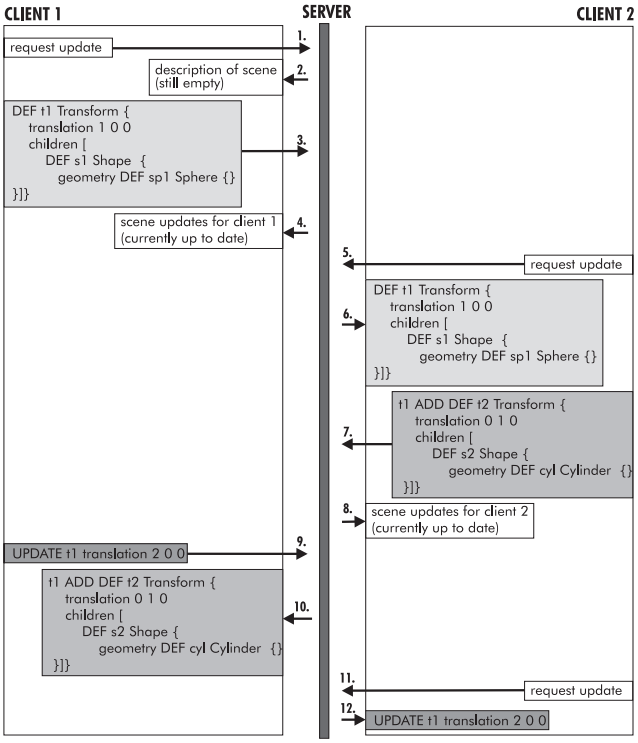


Figure 3. Example of graph creation and editing with two participating clients.

(step 9), it is notified that an object has been added (step 10). In step 11, client c2 requests updates to maintain its local database.

Note that though geometric models are described using VRML97, some commands have been added to the language. In step 7, ADD specifies which grouping node a model should be added to. By default, a model is added directly to the root of the scene such as in step 3. In step 9, UPDATE is used to modify the value of a field. A more complete description of the language is given in the Appendix. Recursively removing a subgraph can be achieved using DELETE. These basic commands allow the edition of the common graph. In this preliminary work, we have not addressed the question of permission restrictions.

In order to build up a common virtual environment, e.g. a networked community, the online server must run permanently on a specific network address for the clients to be able to connect. As such interactive applications happen in real time (at least they should), the online server is subject to heavy time constraints. Replicating the common scene locally in each client and limiting the subsequent communication between client and server to VRML updates is a first step to comply with these requirements; future research concerns the development of Levels of Detail (LODs), including viewpoint dependency and priority management [17], in order to optimize the distribution of updates to the clients.

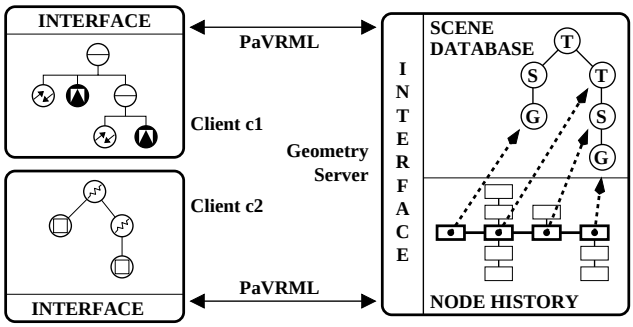


Figure 4. Structure of the offline server (compare to Figure 2).

5 The Offline server

Although the online mode of the server is suited for virtual environments and networked communities, it is not appropriate for the collaborative construction of an animation, where frequent modifications and playbacks are necessary, and thus the clients need to have full access to the server's database. These requirements led to the development of the so called "offline" mode, in which the server works comparable to a multi-track video recorder.

In offline mode, the animation is not "running"; there is no "actual" time of the animation. Thus the changes in the common scene database can be specified independently of the animation viewing; basically, the server acts as a common database, which can be modified and queried at any time. To modify the animation, clients can send scene updates to the server, specifying the point on the animation time line at which these updates should occur. To get information about the scene (in order to view the animation), a client has to explicitly query the server's scene database, specifying the time it is interested in. Then the client gets back all scene updates concerning the interval between this specified time and the time specified in its last query, allowing it to update its database.

Therefore the server has to store the complete history of the animation (see Figure 4); the offline mode of the server is more memory consuming than the online mode, but has the advantage of having less time constraints. While in online mode the server is required to make an update to the common scene available to all clients immediately and possibly contemporarily, the offline mode has an inherent inconsistency between the client's databases. As each client can independently query a specific time interval from the

server database, the local databases of the clients are not linked among each other by time constraints. Furthermore the changes to the scene and the animation viewing can happen at different times, thus not all clients must be notified immediately of a change to the scene database. The average load on a server in offline mode is usually lower than in online mode, as the number of participating clients is typically smaller; often the server is run locally for just one client.

We now present an example of incremental animation design including a server running in offline mode and two clients, where a keyframe animation and a physically-based motion combined.

The first client, produced at LIG (Lausanne), provides the model of a pirate with a feathered hat, who shakes his head as if he was saying "yes". In order to increase realism, a second client adds a physically-based motion of the feather; this is achieved by a software of TU-Vienna. In this simple example, the feather is loosely modeled as a rigid object bound to the hat using a joint including damped angular elasticity.

The "Lausanne"-client uploads the keyframe animation on the server as a VRML97 model along with a number of position updates, each of them associated with a given time. Though interpolators would be a more conventional tool, they are not yet handled by the server.

The "Vienna"-client then logs in and receives the model, which it converts into its own data structures (OpenInventor) using the parser (see Figure 5). According to an agreement between the designers, a transform node called *Feather* is located at the junction between the hat and the feather. This allows the "Vienna"-client (being responsible for the physically-based motion of the feather) to automatically find this node and replace it with an appropriate object modeling a damped elastic joint.

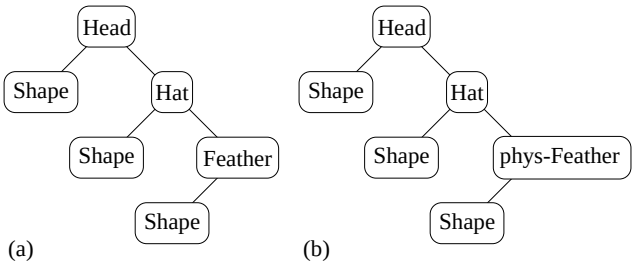


Figure 5. The graph of the pirate head, modeled by LIG (a) and by TU-Wien (b)

This "Vienna"-client then queries the state of the scene at regularly spaced times. The position of the hat is deduced from the updated graph. The motion of the feather is computed accordingly, and replaces the original motion. The velocity of the hat is not considered, though it could be numerically derived. This physical simulation adds flexibility to the feather, providing more realism as illustrated in Figures 6 and 7.

Figure 6. Two snapshots of the keyframe animation. The feather is rigidly bound to the hat.

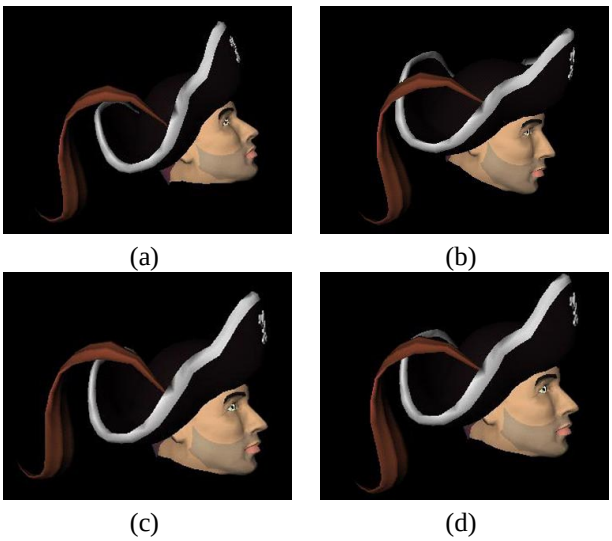


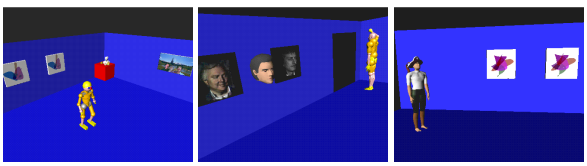
Figure 7. Four snapshots of the physical animation, while the head is moving (a,b) and after (c,d). The feather is physically swinging.

6 Examples

This section explains two usage examples for our system: the first one is about integrating in the same environment models from different sources and coming from different clients, while in the second one the system is used as a way of visualizing and sharing experimentation results over the network.

6.1 The studio

This first example shows a virtual environment containing models coming from different applications, each using its own internal representation. The clients participate in this common session, which we call *studio*, by outputting their models to the server.



(a) (b) (c)

Figure 8. Some views of the studio at a given time

We will now briefly describe the different models present in this example:

- The walls are simple VRML objects. They are decorated with two different types of paints: first we have images produced by a compositing software from the University of Bath (UK), and rendered as textures applied to VRML objects. Examples are the landscape on Figure 8(a), the left image of the pairs on Figures 8(a) and 8(c). The second type of paints is produced by the University of Bath (UK): the twin-images in Figure 8(a) and 8(c) are vectorial image representations described in VRML from the internal NURBS format of an image segmentation software.

- Our studio also includes models of actual objects created by photogrammetry at the Turing Institute, Glasgow (UK). See the objects on the red cube in Figure 8(a) and the two outer faces on the wall in Figure 8(b).

- Between the two outer faces in Figure 8(b) there is an example of facial animation produced at the University of Geneva (Switzerland).

- The two yellow bodies (Figures 8(a) and 8(b)) are produced at the University of Glasgow (UK); the internal format of the keyframe animation has been translated into VRML.

- Finally, the pirate (Figure 8(c)) is the result of a body reconstruction and animation done at École Polytechnique Fédérale de Lausanne (Switzerland).

In this example, all models are described in files containing the initial geometry as well as geometrical updates for the animations. This is particularly suited for an offline animation. Indeed, a client (or several clients) sends the files to the server running in offline mode. Then, different clients can look simultaneously at the same animation, possibly at different times on the animation time line and from different points of view. Each site can modify the part of the animation it is responsible for and send the changes to the server. These modifications can concern any animation time and can be sent at any time. This way, an animation can be built collaboratively.

It is also possible to employ the studio in online mode. However, then it is not anymore possible to modify what was previously sent to the server; everything happens at the specified time and the viewing time cannot be changed. As a consequence, new updates can only concern current or future time.

Figure 9 shows some images of an animation in the studio: the pirate says “yes”; one robot is walking, and the other is waving.

6.2 The snake in the potential

The snake algorithm [13] is used at the University of Bath to recover boundaries of homogeneous regions in blurred images. Snakes are dynamic curves that minimize an energy associated to them. This energy is made of an internal term (first and second order tension) and of an external term coming from the image, which we call the potential. This potential is made from the gradient of the image: it is minimum where a high image gradient occurs, *i.e.* where there is a boundary between to regions.

We use the server in online mode to visualize the temporal evolution of the snake in the potential. The potential is represented as a triangulated mesh while the snake is represented with diamonds for each of its control points. The potential as well as the snake in its initial state are sent to the server. Clients connected to the server can then ask for the initial geometry. At each iteration, after computing the new snake state, updates containing the new diamonds’ positions are sent to the server. Clients can thus display the animation in real time by regularly querying the server for the updates.

Figure 10 shows the time evolution of the snake during the extraction of the boundary of a blurred shape in front of a background. In the first column, the snake is at its initial position. The last column shows the snake at equilibrium. The two middle columns show intermediate positions.

In this example, we have used the server in online mode but all messages sent to the server could have been saved into a file and visualized later in offline mode. In both modes, our system is used to visualize experiments and to share the results over the network.

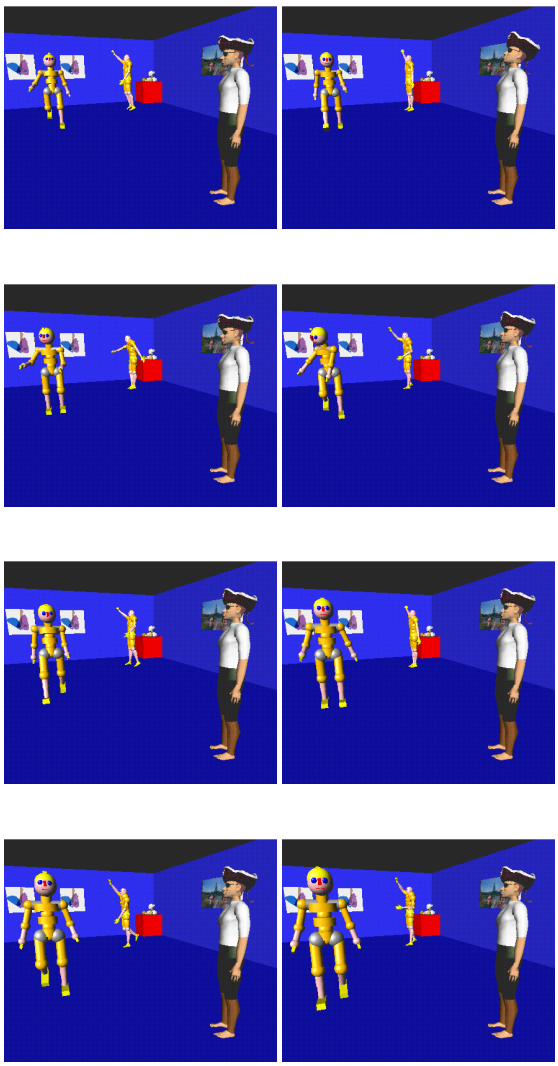


Figure 9. Some images of an animation in the studio

7 Conclusion and future work

We have presented a new application to VRML-based distributed environments. An external interface allows different, previously incompatible software to contribute to the animation of a common scene. This flexibility allows a group of researchers or practitioners to easily share their competence. The only necessary modification of existing code is the addition of a communication layer. The applications can run remotely over the net. This avoids all problems of local installation and guarantees code privacy. Preliminary results in two application domains, real-time distributed environments and collaborative animation, have been shown.

The centralized database included in the server allows us to address the question of smart streaming in real-time environments. Currently, on-demand data transmission allows the server to send only the data which is relevant to a given client at the time it queries the database. In the future, further reduction of the network load should be obtained by exploiting the knowledge of the server about the scene and the viewers. Knowing the camera position of a client will allow the server to send only data relevant to what the client actually sees, and to use levels of detail for low priority items.

Collaborative animation has been performed and appears a valuable approach. The ability of building on preexisting animations generated by specialized software and practitioners allows different users to incrementally design complex animations, independently of what software and hardware each uses. Further work should introduce physical data such as mass and force to allow physical objects managed by different clients to interact physically. Velocity should also be modeled. The prototyping capabilities of VRML97 allow the creation of the desired new nodes.

Finally, we plan to increase compatibility with existing or future standards. Using interpolators and switching nodes will allow the server to export an animation in true VRML97 language for web publishing. Conforming our additional commands to the MPEG4 specifications will highly increase the possible range of collaboration over the net.

Acknowledgements

We gratefully acknowledge the support of the European Union's Training and Mobility of Researchers (TMR) programme in funding this work.

A The language

A subset of VRML97 is used to model geometry. The following nodes are currently supported: Appearance, Box, Color, Cone, Coordinate, Cylinder, DirectionalLight,

Group, ImageTexture, IndexedFaceSet, Material, Normal, PixelTexture, Shape, Sphere, TextureCoordinate, TextureTransform, Transform, Viewpoint.

Additionally, a set of commands to access the database are defined (see Table 1).

Table 1. The commands added to VRML to form PaVRML

command	syntax	description
ADD	*name* ADD *graph*	Adds a VRML node or graph to the specified node
GETTIME	GETTIME *float*	Asks for the scene description at the time specified (used in the offline mode only)
INLINE	INLINE "*url*"	Includes a remote scene or animation
REMOVE	*parent* REMOVE *child*	remove the specified child and subnodes
SETTIME	SETTIME *float*	Specifies the time updates should occur at
UPDATE	*name* UPDATE *field value*	specifies numerical values

B The network library

Our system design is based on a client-server approach similar to the approaches presented by Funkhouser [11] and Das et al. [8]. The network implementation was intended to function as a "middle-ware" layer that mediates between the application and the underlying network infrastructure [12].

The network layer offers two types of basic message delivery services:

- Lightweight transport intended for fast, unreliable delivery of small messages of idempotent semantics. Such messages are automatically discarded if newer information becomes available as proposed by Kessler [14].
- Heavyweight transport for reliable, stream-oriented data delivery with the option of repeated acknowledgement of the application layer when a user specified amount of data has been received.

The selection of the appropriate mode of transport can either be made by the system based on the message's size, or it can be chosen by the user.

Further functions of the network layer include intelligent maintenance of the network connections, which is much harder from the application layer. As an example, if no data is received from a client for a specific period, it is assumed dead. The client's network connection is shut down and the network layer notified the application layer and other clients.

C The interface

The only requirement for each partner of the network is to be able to send and receive messages using the PaVRML syntax:

- The production of VRML97 nodes from our own data structures is quite an easy task: only a conversion program of a few hundred lines has to be written by each site involved in the project.

  At the beginning of the animation a full description of the objects added to the global scene has to be sent. Then, after the generation of each new frame, only modified values are sent.

- The task of receiving messages may be divided into two subtasks:

  – Parsing the received messages,

  – Building a local copy of the modeled scene using its own data structures.

  Each partner does need a PaVRML parser. We use a generic customizable parser. This parser is a C++ library which has been written using LEX for the lexical analysis and YACC for the grammar analysis. Each time a VRML97 node is read by the parser, it calls a virtual method from an abstract class. The task of each partner is to derive this class and to write the methods which will allow the building of the local copy of the global scene. Similarly, our additional commands are interpreted using procedure calls.

References

[1] Active worlds. *http://www.digitalspace.com/avatars/aworld.html*.

[2] Blaxxun. *http://www.blaxxun.com.*

[3] Building and interacting with universal virtual products. *http://support.division.com/5.tec/a_papers/uvp.htm*.

[4] Distributed interactive simulation dis-java-vrml working group. *http://www.vrml.org/*.

[5] Vnet. *http://ariadne.iz.net/ jeffs/vnet/FAQ.html.*

[6] T. Capin, J. Esmerado, and D. Thalmann. Dead-reckoning algorithms for streaming virtual human data. *to appear in IEEE Transactions on Circuits and Systems for Video Technology*, 1998.

[7] C. Carlsson and O. Hagsand. Dive - a multi-user virtual reality system. In *Proceedings of IEEE VRAIS '93, Seattle, Washington*, 1993.

[8] T. K. Das, G. Singh, A. Mitchell, P. S. Kumar, and K. McGhee. NetEffect: A Network Architecture for Large-scale Multi-user Virtual Worlds. In D. Thalmann, editor, *ACM Symposium on Virtual Reality Software and Technology*, pages 157–163, New York, NY, Sept. 1997. ACM, ACM Press.

[9] P. Doenges, T. Capin, F. Lavagetto, J. Ostermann, I. Pandzic, and E. Petajan. Mpeg-4: Audio/video and synthetic graphics/audio for real-time, interactive media delivery. *Image Communications Journal*, 5(4):433–463, 1997.

[10] J. C. et al. The simnet virtual world architecture. *Proceedings of VRAIS'93*, pages 450–455, 1993.

[11] T. Funkhouser. Network topologies for scalable multi-user virtual environments. *Proceedings of VRAIS'96, Santa Clara CA*, pages 222–229, 1996.

[12] G. Hesina. A network library for multi-user virtual environments. Master's thesis, Technical University of Vienna, 1997.

[13] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[14] G. Kessler and L. Hodges. A network communication protocol for distributed virtual environment systems. *Proceedings of VRAIS'96, Santa Clara, CA*, pages 214–221, 1996.

[15] M. Macedonia and M. Z. et al. Npsnet: A network software architecture for large scale virtual environments. *Presence*, 3(4):265–287, 1994.

[16] I. Pandzic, N. M. Thalmann, T. Capin, and D. Thalmann. Virtual life network: A body-centered networked virtual environment. *Presence*, 6(6):676–686, 1997.

[17] D. Schmalstieg and M. Gervautz. Demand-driven geometry transmission for distributed virtual environments. *Proceedings of EUROGRAPHICS'96*, 15(3):421–432, 1996.

[18] D. Snowdon. Aviary: Design issues for future large-scale virtual environments. *Presence*, 3(4):288–308, 1994.

[19] K. Watsen and M. Zyda. Bamboo - a portable system for dynamically extensibe real-time, networked, virtual environments. In *Proceedings of IEEE VRAIS'98, Atlanta, Georgia*, 1998.

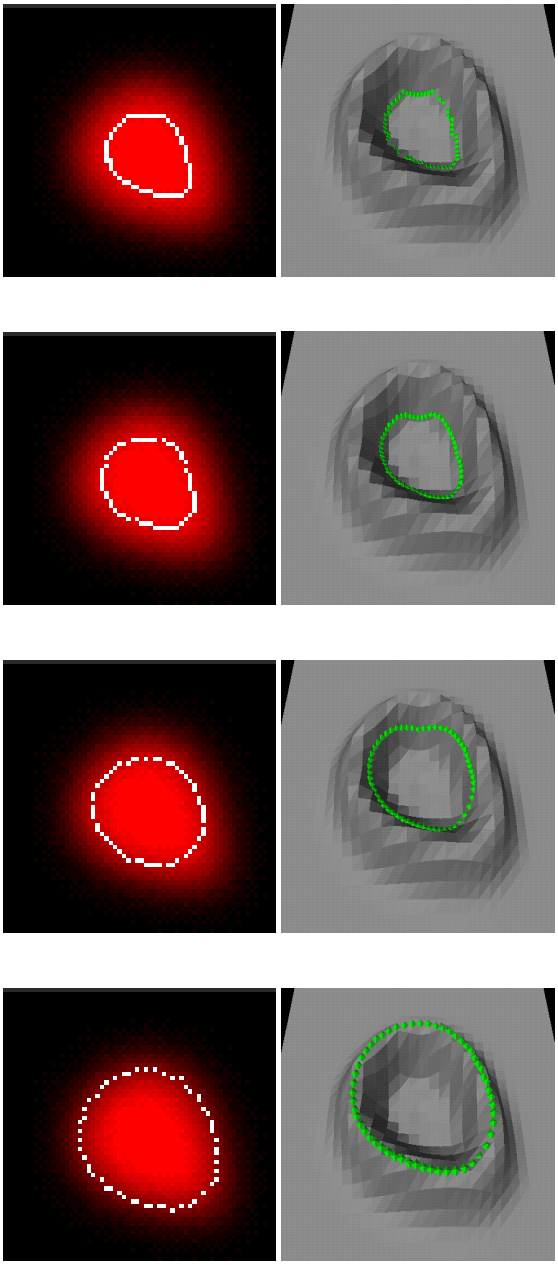


Figure 10. The snake. The left column shows the snake in the image while the column shows the snake remotely visualized as a 3d scene.