Anyone for Tennis?

Tom Molet*
Amaury Aubel*
Tolga Çapin*
Stéphane Carion†
Elwin Lee†
Nadia Magnenat-Thalmann†
Hansrudi Noser*
Igor Pandzic†
Gaël Sannier†
Daniel Thalmann*
*Computer Graphics Lab
EPFL
Lausanne, Switzerland
{aubel, capin, molet, noser, thalmann}@lig.di.epfl.ch
†MIRALab
University of Geneva
Switzerland
{Elwin.Lee, thalmann, pandzic, sannier}@cui.unige.ch

Abstract

In this paper we present a virtual tennis game. We describe the creation and modeling of the virtual humans and body deformations, also showing the real-time animation and rendering aspects of the avatars. We focus on the animation of the virtual tennis ball and the behavior of a synthetic, autonomous referee who judges the tennis games. The networked, collaborative, virtual environment system is described with special reference to its interfaces to driver programs. We also mention the virtual reality (VR) devices that are used to merge the interactive players into the virtual tennis environment, together with the equipment and technologies employed for this exciting experience. We conclude with remarks on personal experiences during the game and on future research topics to improve parts of the presented system.

1 Introduction

At the opening and closing session of Telecom Interactive '97 in Geneva, Switzerland, we presented in real time a virtual, networked, interactive tennis game simulation. The videotapes were demonstrated in the opening ceremony of the Virtual Humans '97 Conference at Los Angeles, at the Virtual Technologies booth of the SIGGRAPH'97 exhibition, as well as at various other conferences. This demonstration was a big challenge because, for the first time, we had to put together several different computer-related technologies and corresponding software. This had to work in real time at a specific moment on the exhibition site with nonpermanent installations. In this demonstration, the interactive players were merged into the virtual environment by head-mounted displays (HMDs), magnetic sensors, and data gloves. The University of Geneva player was ''live'' on stage at the opening session and was separated by a distance of approximately 60 km from the other player in the Computer Graphics Lab of EPFL at Lausanne, Switzerland. For managing and controlling the shared, networked, virtual environment, we used our Virtual Life Network, which is a general-purpose client / server network system that uses realistic virtual humans (avatars) for user representation. These avatars support body deformation during motion. The virtual humans also represent autonomous virtual actors such as the synthetic referee, who is part of the tennis game simulation. A special tennis ball driver animated the virtual ball by detecting and treating collisions between the tennis ball, the virtual rackets, the court, and the net.

The paper is organized as follows. Section 2 discusses the creation of the virtual players. Section 3 details the procedures that were used to model and implement the real-time body deformations. Section 4 gives an overview of the

Presence, *Vol. 8, No. 2, April 1999,* 140–156

networked virtual environment system used as software basis for integration. The autonomous referee and game automaton controlling the ball animation are detailed in Section 5. Section 6 discusses the particular motion-capture and VR-feedback approaches that were built for the virtual tennis application. Sections 7 and 8 respectively present the experiments and the conclusions.

2 Creation of the Virtual Players

The modeling and deformation of virtual humans is an exhaustive task (Badler, Phillips, & Webber, 1993). Because we can use scanning devices to create the human shape, the modeling of the avatars seems to be obvious. However, with these methods we have no more information than the shape of the virtual human in default posture, which is not sufficient to compute deformations during the animation (motion of bodies, animation of faces). We use two different modeling tools in the production of the virtual players, and these tools are linked with the animation we need to realize on each part of the body. One is a surface-based sculpting program used to model heads (LeBlanc, Kalra, Magnenat-Thalmann, & Thalmann, 1991). The other is our software, BodyBuilder (Shen, Chauvineau, & Thalmann, 1996), dedicated to the creation of human bodies using metaballs attached on a skeleton. Using texture-mapping and texture-fitting methodology enhances the final realism.

2.1 Sculpting the Player's Face, Hands, and Feet

The method we use is similar to traditional clay or wax sculpting. The designer starts from an irregular triangle mesh, modifying it with simple tools like adding, deleting, modifying, assembling, and splitting primitives of the surface. The use of local and global deformations on the 3-D head provides the designer with a sense of real sculpting by enabling the gradual transformation of the object. This kind of modeling suits our needs in facial animation because the user creates the heads knowing the animation requirements. Thus, users are able to create more detail on the animated parts of the head (lips, eyelids, etc.) and less detail on the other nonanimated parts (hair, forehead, etc.). (See Figure 1.)

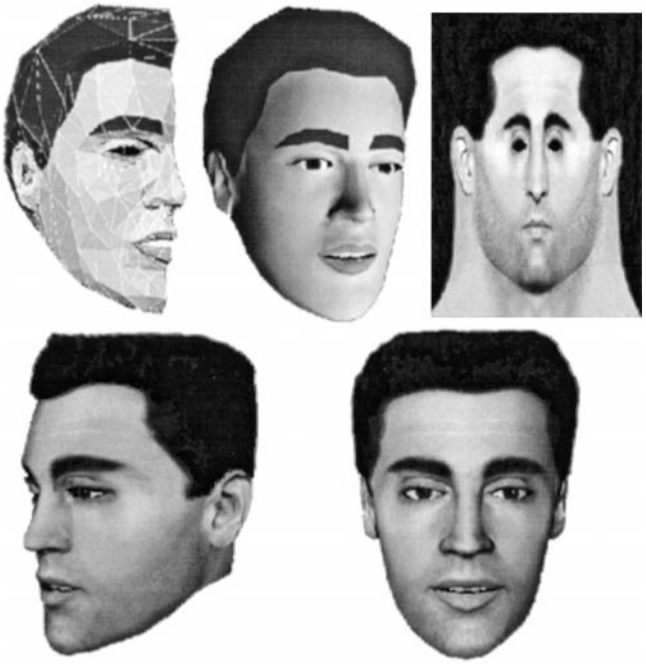

Figure 1. *Description of images from left to right, top to bottom: (1) One-half of the head is created. (2) The other half, the eyes, and the teeth are added. (3) The texture is designed. (4 & 5) The texture is adjusted to the 3-D model.*

For the tennis players, we constructed the heads from an already existing head that is precisely defined in terms of the complexity of the deformable regions. The idea is to deform the template to get the head we want to design. It is obvious that the best solution is to start with a template that is as close as possible to the target face. For hands and feet, the same approach is used.

2.2 Body Creation of a Tennis Man and Woman

The BodyBuilder software is dedicated to modeling the human body by creating a human envelope. The human form is obtained by attaching metaballs into the

Figure 2. *Creation of the body.*

Figure 3. *Body surface with textures.*

joint articulations of a skeleton entity (created / animated by the user). A spline surface is computed using a ray-casting method on the body envelope. The surface is then triangulated and assembled with the head, hands, and feet (Figure 2). The user can easily deform, scale, rotate, and translate the metaballs to model the global shape of the body.

Creating human bodies is a difficult task that requires anatomical knowledge and experience in muscular deformations, because metaballs act almost like the muscles under the skin envelope. Thus, creating the nice shape of a human body in a specific posture requires knowledge of the muscle shapes in that posture (Figure 3). Also, the exact proportions of the body need to be preserved because the human eye can easily detect the slightest defect. Once a body is created (male or female), we can easily create another by using the first as a template for the body creation. Although BodyBuilder generates high-quality deformation, its computations are too heavy for real-time applications. Therefore, real-time deformations are made using another technique (presented in Section 3) that is based on the skin mesh provided by BodyBuilder.

2.3 Texturing

As we use a modeling method to create the avatars, we cannot just texture the head and the body using simple projections. If the textures have features (nose, eyes, hair), we want them to match the corresponding details on the 3-D object (Figure 3). This is a difficult operation when the images and the 3-D objects are slightly different. We created an interactive texture-fitting software (Sannier & Magnenat-Thalmann, 1997) to realize this kind of precise texturing (Litwinowicz & Miller, 1994). The idea is to allow the designers to correlate the main features of the texture directly with the 3-D object. These features are projected on the 2-D image where the designer can adjust their position; the texture is then applied to the object using these points as references. The user is able to warp the texture locally

and directly onto the 3-D object, which provides more flexibility.

3 Body Deformations

Many attempts have been made to produce fairly realistic virtual humans, but few of them have been oriented towards real-time animation. As a result, most examples of real-time animated humans are often unrealistic in the sense that they rarely show human characters whose skin is properly deformed. In this section, we shall describe an efficient method that we have been using to render and animate realistic tennis players in real time. By relying on cross-sectional data and deforming skin contours, we succeed in representing a player in a most accurate fashion, while preserving a consistent frame rate. This approach deals only with the body itself, which means the hands, head, and feet are not taken into consideration.

3.1 The Model Data

The surface model for the virtual players is conceptually simple, containing a skeleton and an outer skin layer. The envelope is basically a set of points, and the underlying skeleton is a hierarchical model comprising a total of 32 joints corresponding to 74 degrees of freedom (DOF) (Boulic, Capin, Huang, Kalra, Lintermann, Magnenat-Thalmann, Moccozet, Molet, Pandzic, Saar, Schmitt, Shen, & Thalmann, 1995).

Because the human limbs and trunk exhibit a cylindrical topology, a natural centric axis comes out in each body part. If we consider, for example, a human leg, the natural axis runs vertically. Now, if we scan this leg along its axis (in a top-to-bottom fashion, for instance) and look at cross sections of this limb at regular times, the outermost points of each cross section would form a drawing similar to Figure 4. Essentially, these very points make up our model's envelope.

To obtain this data, we use the BodyBuilder software described in Section 2. This tool allows the construction of highly realistic human models from implicit surfaces. Once a model is rendered, it is possible to sample its skin surface and then output cross-sectional data (Shen et al., 1996). BodyBuilder divides a human body into 12 logical parts: neck, torso, shoulders, forearms, waist, pelvis, thighs, and calves. We are also provided with skin contours for 12 body parts. For the sake of clarity, when speaking in the following of a (cross-sectional) contour, we will mean the set of outermost, coplanar points (vertices) of a given cross section.

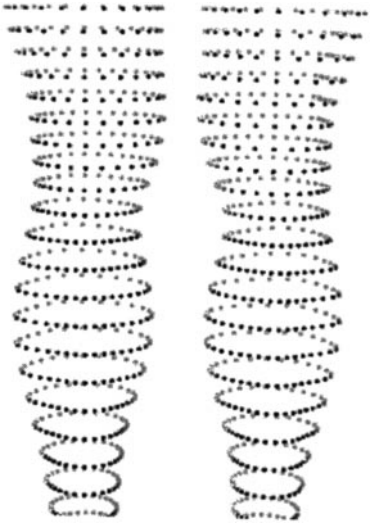

Figure 4. *Cross-sectional data for the legs.*

Each body part is then composed of a certain number of cross-sectional contours, which in turn are made up of a fixed number of vertices. However, a contour belongs to only one body part, and contours belonging to the same part have an equal number of points. Therefore, a triangle strip can be constructed from two adjacent cross sections by simply connecting their points. Thus, it is possible to construct an entire triangle mesh for each body part directly from the contours' points. Figure 5 shows the resulting mesh when assembling all the body parts. The connections between distinct parts can clearly be seen in some regions (e.g., the shoulder area).

3.2 Animation by Deforming the Contours

During the tennis game, joint angles are updated permanently in the virtual skeleton. We associate every joint with a cross section and make sure every joint lies

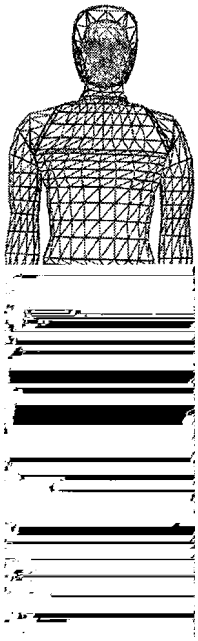

Figure 5. *Body mesh in at-rest position.*

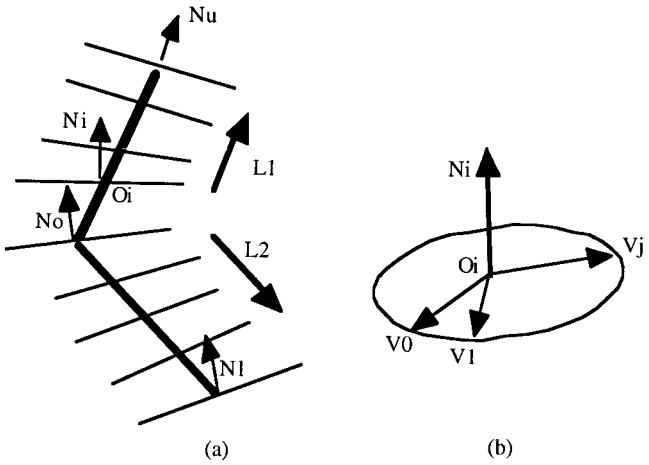

Figure 6. *Cross-sectional plane orientation.*

in the cross-sectional plane to which it is mapped. This particular mapping helps us deform the cross-sectional contours that lie between two successive joints.

The basic idea behind contour deformation is to use the angle between two connected segments in the skeleton to drive the position and orientation of the cross-sectional planes in-between. In Figure 6(a) (which could illustrate the case of the leg, for instance), L_1 is the direction of the upper segment (possibly the thigh) and L_2 is the lower one (possibly the calf's axis). Let N_u and N_l be the normal vectors of the cross-sectional planes at the segments' ends. We set the orientation of the cross-sectional plane that passes through the joint to be the bisection plane of the two links. Let this bisection plane normal be N_0. Suppose now that O_i and N_i are the center and normal, respectively, of the *i*-th cross-sectional plane along the upper segment. N_i can then be computed by direct interpolation of two end normals N_0, N_u. Knowing the normal vector N_i, it becomes straightforward to compute the new local coordinates of each vertex V_j belonging to the *i*-th contour (Figure 6(b)).

It often makes little sense to interpolate the normal vector of every cross-sectional plane along a skeleton's segment. As an example, let us consider a walking human. In this case, most of the deformations are obviously concentrated in small areas surrounding the joints, such as the knees, ankles, or elbows. Therefore, it is preferable to apply our deformation system only on the contours that reside in the vicinity of a joint, so as to reduce rendering time as much as possible. We practically determined the number of contours to be deformed in a heuristic fashion. However, there exist some cases for which it would be more realistic to use a fully deformed model (when twisting an arm, for example).

3.3 Implementation

We rely on the Performer library to have an efficient implementation. This real-time graphics toolkit lets applications achieve maximum graphics performance from all Silicon Graphics workstations. With this library, it is easy to create a 3-D scene with many objects coming from different modelers. Morphing, however, is the only deformation capability that is available in Performer.

Consequently, we define our own structure to be compatible with the Performer data arrangement (linear). By using an index array to define triangle meshes, we avoid duplicating vertices and achieve an efficient memory management. It is also easy with Performer to apply a texture onto a 3-D surface. In our model configuration, we are able to apply different textures on each body part. Thanks to dedicated hardware, texture

mapping turns out to be a good way to increase visual realism at quite a low cost in computational terms.

A single-CPU Octane workstation computes the deformations of all body parts for a model containing approximately 14,000 vertices in 11 milliseconds. Thus, we manage to run a walking sequence (all body parts involved) of a body model composed of 13,500 textured triangles at 36 frames per second.

Finally, because the computational cost of our deformation method is directly proportional to the number of vertices / contours, it is particularly well suited for this level of detail.

4 Integration and Networking

There is an increasing interest in networked virtual environments (NVEs); various successful systems have been developed (Carlsson & Hagsand, 1993; Macedonia, Zyda, Pratt, Barham, & Zeswitz, 1994). Virtual Life Network (VLNET) is a general-purpose client / server NVE system that uses highly realistic virtual humans for user representation (Capin, Pandzic, Noser, Magnenat-Thalmann, & Thalmann, 1997). VLNET achieves great versatility through its open architecture with a set of interfaces that allow external applications to control the system functionality.

Figure 7 presents a simplified overview of the architecture of a VLNET client. The VLNET core performs all the basic system tasks: networking, rendering, visual database management, and user management including body movement and facial expressions. The previously described body deformation module is integrated in the client core. When actors are animated, each client updates the skin shapes of all visible virtual actors within the client's field of view. A set of simple shared-memory interfaces is provided through which external applications can control VLNET. The VLNET drivers also use these interfaces. The drivers are small service applications provided as part of the VLNET system and can be used to solve some standard tasks, such as generating walking motion and supporting navigation devices like a mouse or SpaceBall. The connection of drivers and external applications to VLNET is established dynamically at runtime based on the VLNET command line.

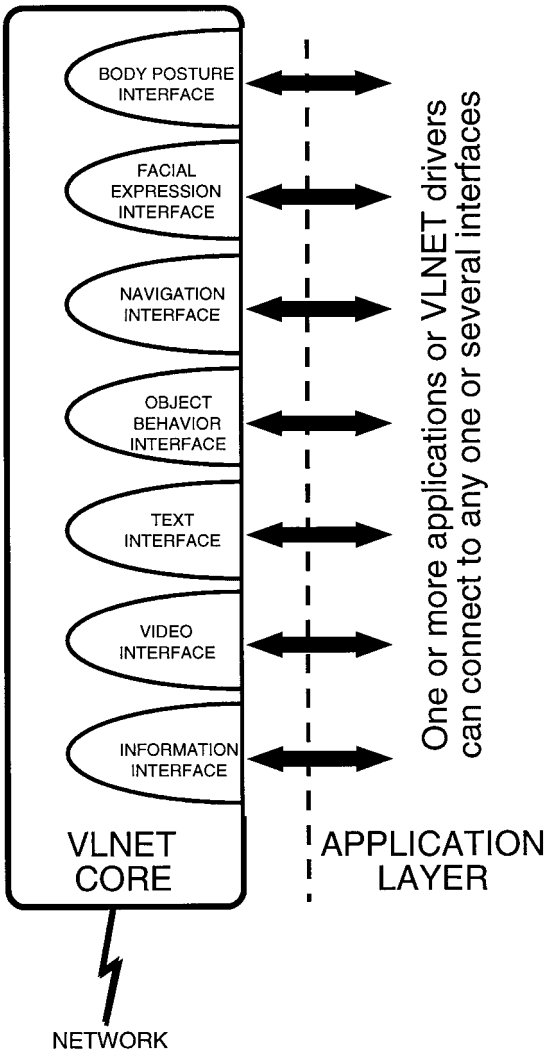

Figure 7. *Simplified view of VLNET client architecture.*

The *Facial Expression Interface* is used to control the expressions of the user's face. The expressions are defined using the Minimal Perceptible Actions (MPAs) of Kalra, Mangili, Magnenat-Thalmann, and Thalmann (1992). The MPAs provide a complete set of basic facial actions. By using them, it is possible to define any facial expression.

The *Body Posture Interface* controls the motion of the user's body. The postures are defined using a set of joint angles corresponding to the 72 DOF of the skeleton model (Boulic et al., 1995) used in VLNET.

The *Navigation Interface* is used for navigation, hand and head movement, basic object manipulation, and basic system control. All movements are expressed using matrices. The basic manipulation includes picking objects up, carrying them, and letting them go, as well as grouping and ungrouping objects. The system control provides access to some system functions that are usually accessed by keystrokes, such as changing drawing modes, toggling texturing, and displaying statistics.

The *Object Behavior Interface* controls the behavior of objects. It is currently limited to controlling motion and scaling, defined by matrices passed to the interface. It is also used to handle the sound objects, i.e., objects that have prerecorded sounds attached to them. The Object Behavior Interface can be used to trigger these sounds.

The *Video Interface* streams video texture (as well as static textures) onto any object in the environment. The Alpha channel can be used for blending and achieving effects of mixing real and virtual objects and persons. The interface accepts requests containing the bitmapped image and the ID of an object on which the image is to be mapped. The image is then distributed and mapped on the requested object at all sites.

The *Text Interface* sends and receives text messages to and from other users. An inquiry can be made through the text interface to check if there are any messages, and the messages can then be read. The interface gives the ID of the sender for each received message. A message sent through the text interface is passed to all other users in a VLNET session.

The *Information Interface* is used by external applications to gather information about the environment from VLNET. It provides high-level information while isolating the external application from the VLNET implementation details. It also allows two ways of obtaining information: the request-and-reply mechanism and the event mechanism.

We have given a brief description of VLNET that allows us to show how external programs can be interfaced to the VLNET system. The focus of this presentation was on VLNET interfaces. For more details on VLNET, the reader is directed to Capin et al. (1997) and Pandzic, Capin, Lee, Magnenat-Thalmann, and Thalmann (1997).

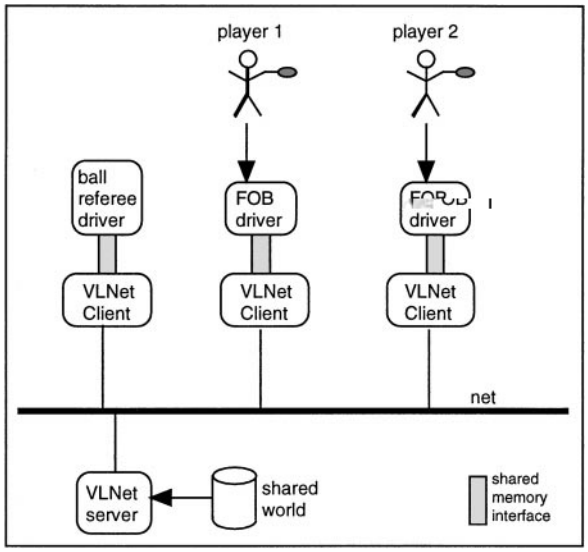

Figure 8. *The VLNET server/client/driver configuration.*

4.1 Integration of Game Simulation in VLNET

Figure 8 shows the VLNET server / client / driver configuration for the networked interactive tennis game simulation. The autonomous agents are controlled by a ball / referee driver (detailed in the following section) that accesses the shared virtual environment through shared memory interfaces of a VLNET client process. (See Figure 9.) Through the object-behavior interface of this client, the driver program is periodically updated with the actual positions of the players' rackets.

The sound events and ball position are broadcast through the same interface for each frame. Through the body-posture interface, the body data of the referee is broadcast to the other VLNET clients. The body data or joint angles are created by using the AGENTlib software (Boulic, Bécheiraz, Emering, & Thalmann, 1997) developed for motion control for virtual humans. The navigation interface can be used to map the camera of the VLNET client to the eyes of the autonomous referee in order to watch the shared virtual world from the referee's point of view.

The network bandwidth requirement is 20 kb / s per virtual human (without facial animation), 20 kb / s for the body parameters (joint angles) communication, and 10 kb / s to transfer the navigation matrices. Each ani-

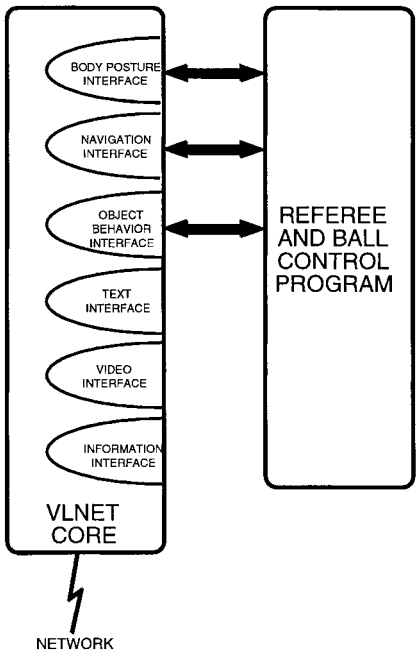

Figure 9. *The VLNET interfaces used for the autonomous referee and the ball driver.*

mated object (rackets, ball, ball's shadow) needs approximately 10 kb / s. In this application, all this information is delivered uncompressed to save computation resources for other tasks.

5 The Game

Noser, Pandzic, Capin, Magnenat-Thalmann, and Thalmann (1996) already described a networked tennis game facility in which a user with a Spaceball could play against an autonomous actor. The game was judged by a synthetic, sensor-based referee. The ball was animated by a forcefield-based particle system driven by a numerical, differential equation solver. A disadvantage of this system is the need of a relatively high frame rate to allow a reliable collision detection and response treatment that slowed down the simulation speed of the whole game.

In this paper, we address the modeling of a tennis game with a synthetic humanoid referee especially developed for a networked virtual tennis game simulation with interactive players merged into VR by two sets of magnetic sensors (described in more detail in Section 6). The real-time constraints, the network and user interface delays, and the need for several player levels led us to a multilevel game design for beginners and experienced players.

The tennis court, net, and the two rackets are all relevant to the tennis ball animation. All these elements have to be defined in a VLNET file describing the shared virtual environment that is downloaded by each VLNET client at the beginning of a networked tennis game session (Figure 10).

We implemented the physics of ball animation in a straightforward manner, based on simplified physical laws for particle-like balls. In order to compensate certain network-induced problems and to allow for several game levels, we used several methods for the ball-racket collision response. These methods range from user-friendly ''missile-like'' balls that fly automatically to the partner's racket, to physically based collision response with ball spin effects for advanced players on fast networks and computers. In particular, we implemented the following collision-response methods:

a) racket determined: determined only by racket velocity
b) friendly ball: missile-like, flying to the partner's racket
c) mixed response: mix of methods a) and b) with user-defined weights
d) physical response: approximation based on physical laws

The synthetic referee is designed for an optimal judging of real-time, interactive, and networked tennis games. This referee is not any more based on synthetic sensors as described by Noser et al. (1996), but it directly accesses information available in the animation process for speed optimization. Moreover, it makes extensive use of the VLNET interface as described earlier.

The high-level behavior of the referee and the game itself are modeled with finite state automata similar to those described by Noser and Thalmann (1997). To create the movements of the referee at motor level, we use the AGENTlib software package (Boulic et al., 1997). The resulting body postures are mapped through the

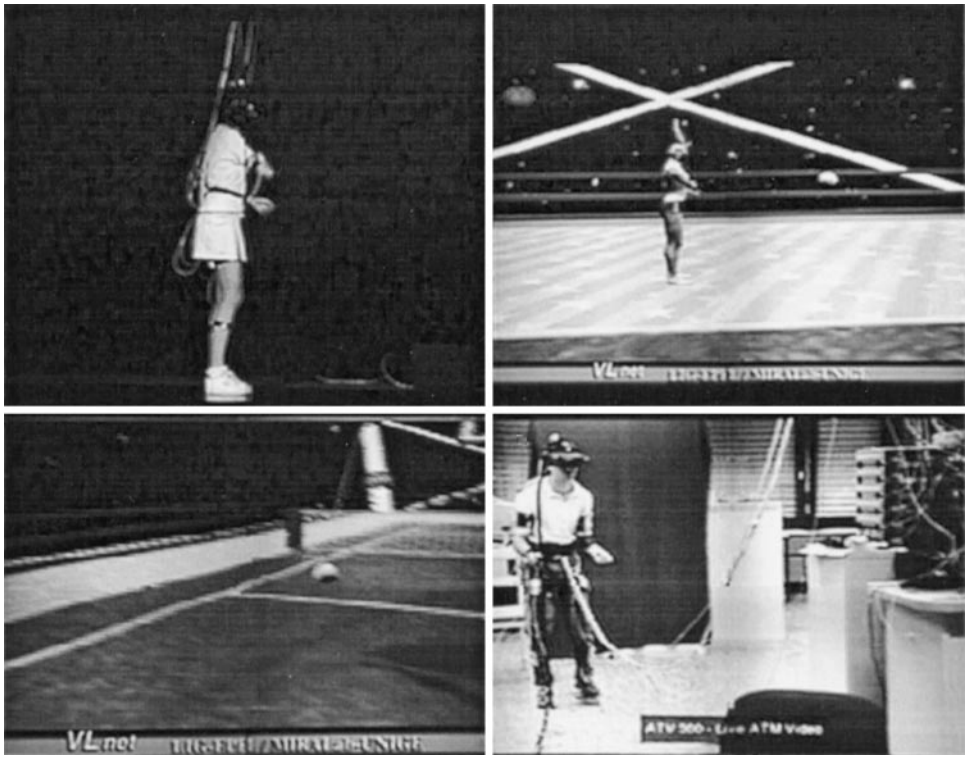

Figure 10. *Ball simulation in the network game.*

body-posture interface onto the avatar in the shared environment of VLNET.

The synthetic referee represents a very useful feature of the tennis game model as it frees the players from updating the current game themselves. Visually, it is represented by the humanoid illustrated in Figure 11. Furthermore, it is able to walk around in the virtual environment, and it can execute prerecorded motion tracks. For game result announcements, it plays back prerecorded sounds. To judge the game, the referee only needs information from the sound events produced by the ball. Therefore, it does not matter whether it judges autonomous humanoids, interactive users, or both.

The high-level behavior of the referee is illustrated in Figure 12. We suppose that the referee is already positioned correctly (next to the court where it judges the game). The corresponding humanoid stands next to the net or sits on a chair. During a match, the referee moves only its head in order to look at the players or to track the ball. At the beginning of a game or after a game fault, it looks first at the receiver and then at the server. Then, it throws the ball into the game. The server and receiver have to be at their corresponding start position.

The ball is thrown into the game approximately one meter above the actual position of the server's racket with a certain vertical upward velocity.

During the game, the referee tracks the ball with its head (eyes) and captures the ball-floor and ball-racket sound events that permit it to detect game errors. If it detects an error, it announces the type of error, updates the game, and announces the new game score (if it has changed). Finally, if the match has not finished, it announces the player who has to serve, and the whole scenario is repeated until the end of the match.

A tennis game can be described by a set of states and transitions as illustrated in Table 1, Figure 13, and Figure 14.

The game is always in one of these states. The transitions between states are triggered only by ball-floor or ball-racket collision events. To control these transitions,

Figure 11. *The autonomous referee (synthetic referee).*

the referee needs to know the identity of the actor involved for each racket-ball collision. For each floor-ball collision, the referee needs the position where it happens. With this information, the referee can manage the game automaton and recognize game errors. Figure 13 and Figure 14 summarize the transition rules for bounce events and ball-racket collision events. At the begin of an arrow, there is the ancestor state, the state of the game before the corresponding bounce event or the ball-racket collision event happens. An arrow indicates some condition that must be fulfilled in order to trigger the transition.

According to Figure 14, for example, a ball-racket collision triggers the following transitions if the ancestor state is ''service,'' i.e., the server has already hit the ball. If the receiver hits the ball, he commits an error, as the ball must first bounce on the court. Therefore, the referee interrupts the game and updates it. If the server hits the ball again (and if he has already committed a fault indicated by the Boolean variable, let = true), it's a fault of the server as twice hit the ball. In this case, he can't repeat the service, and the referee stops and updates the game. However, if it's his first error at the service (condition = not let AND server), he has a second attempt (let = true), and the service is repeated.

Figure 13, for example, describes the transitions triggered by ball-floor collision events or bounce events. If we again consider the ''service'' state as ancestor state, we see that, if the ball bounces inside the service allowed court area (condition: in), then the game state goes into the state ''serviceBounced,'' and the game continues normally. If the ball is out of the allowed court area and if it is the first error (condition: out AND not let), the server gets a second chance to serve. If, however, it is the second error (condition: out AND let), it's a fault for the server and the game has to be updated by the referee.

Regular transitions describe tennis playing without faults. At the beginning of a game, the server has to hit the ball, which must bounce first in the corresponding court area before the receiver can hit it back. Then, alternatively, the server and the receiver can play directly or let the ball bounce before returning it. Of course, each

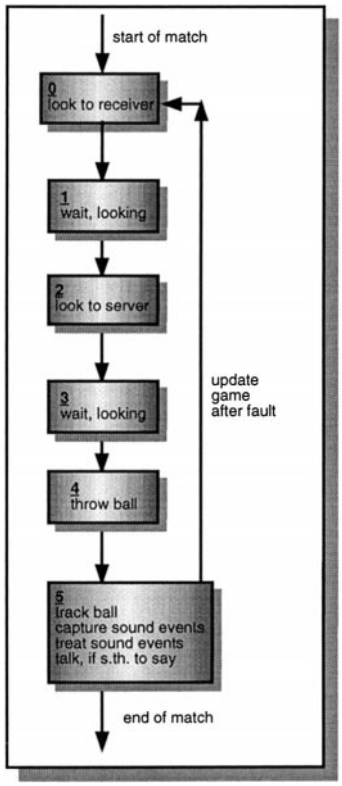

Figure 12. *Referee automaton.*

bounce has to happen inside the corresponding court area.

The referee judges the game according to international tennis rules, but with some modifications that take into account the given constraints and simplify the implementation. The placement of the actors and the first service are decided at the beginning of a match by a script. During the game, there are no change of ends as, currently, the players can't move from one courtside to the other. At the service, the referee does not control the server position. After the service, the ball has to bounce first on the opponent's side, but not necessarily in the diagonally opposite service square as demanded by the international rules. Additionally, collisions between the body of the actors and the ball or the net are ignored. A player wins a set if he has won at least six games and two games more than his partner has. He wins the match if he has won two sets. The tie-breaking procedure is not implemented.

Table I. *Tennis Game States*

States	Description
start	at the beginning of a game the players have to be at their initial play positions
service	the server has hit the ball with his racket
serviceBounced	the ball has bounced once after the service
receiver	the receiver has hit the ball
receiverBounced	the ball has bounced once after the hit of the receiver
server	the server has hit the ball
serverBounced	the ball has bounced once after the hit of the server
update	a game fault occurred and the referee updates the game and announces the score of the game

6 Immersive Technologies

6.1 Magnetic Motion Capture

The player movements are controlled using the human motion-capture (MC) technique (Molet, Boulic, & Thalmann, 1996) based on magnetic sensors. Other methods for real-time human MC are proposed by Bers (1996), Hirose, Deffaux, and Nakagaki (1996), and Semwal, Hightower, and Standfield (1998). We just had to make a specific VLNET external driver, later called *fobnet*, to control the virtual players in the shared virtual tennis scene (Figure 15). The fobnet driver uses three VLNET interfaces: the body-posture interface, the navigation interface, and the object interface. The captured posture of the player is sent to VLNET through both body posture and navigation interfaces. The posture provides the player's view parameters and his / her global position. Similarly, the object interface is used to set and update the position and orientation of the player's racket according to the current player's hand location. The ref-

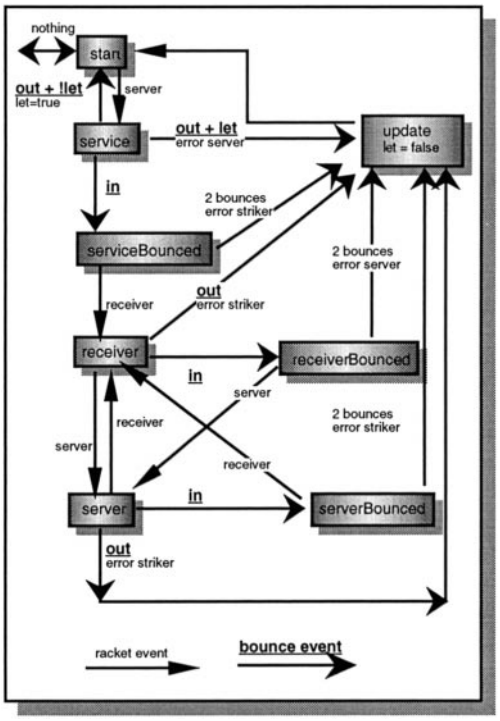

Figure 13. *Game automaton with regular events and bounce events that are faults.*

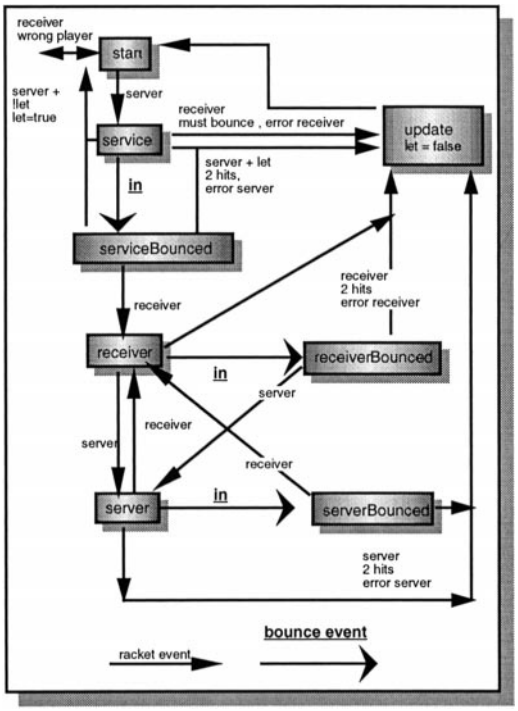

Figure 14. *Game automaton with regular events and racket events that are faults.*

eree client uses this information to detect ball-racket collision by querying its own VLNET client.

Due to the constraints of the magnetic sensing technology, we face a problem: tracking the players' positions is restricted to a smaller range than the real tennis court dimension. We experimented with two solutions to overcome this difficulty.

First, we tried to scale the measured displacement of the global location of the players in the frontal and lateral directions. However, we found that the position amplification should remain within, at most, a factor of 4 in both directions to allow proper control of the players. With amplification values greater than 4, players can hardly adjust their position to hit the ball because little movements produce great location change and thus their stability is lost. This maximum experimental value is not enough, however, for reaching all locations of one side of the court.

The second solution, allowing one player enough displacement to cover one side of the court, is to use an indirect navigation procedure. In this procedure, the global position tracking remains active to control the player's position (with or without reasonable motion amplification) in a restricted zone around the calibration position of the player (Figure 16). When the measured actor position goes outside this perimeter (as illustrated by actor 2 in Figure 16), we activate another navigation paradigm. The global position of the virtual perimeter associated with the MC restricted zone starts to move with respect to the direction indicated by the current player position (Figure 16).

When the player has reached the desired place, he / she just reenters the MC restricted zone to stop the additional navigation motor. In the latest version, we have used this navigation paradigm with amplification values set to 2 in both lateral and frontal directions. Thus, the virtual perimeter associated with the MC restricted zone is twice as large as the real perimeter.

The limitation of this procedure arises from the need to enter back inside the MC-restricted perimeter to stop

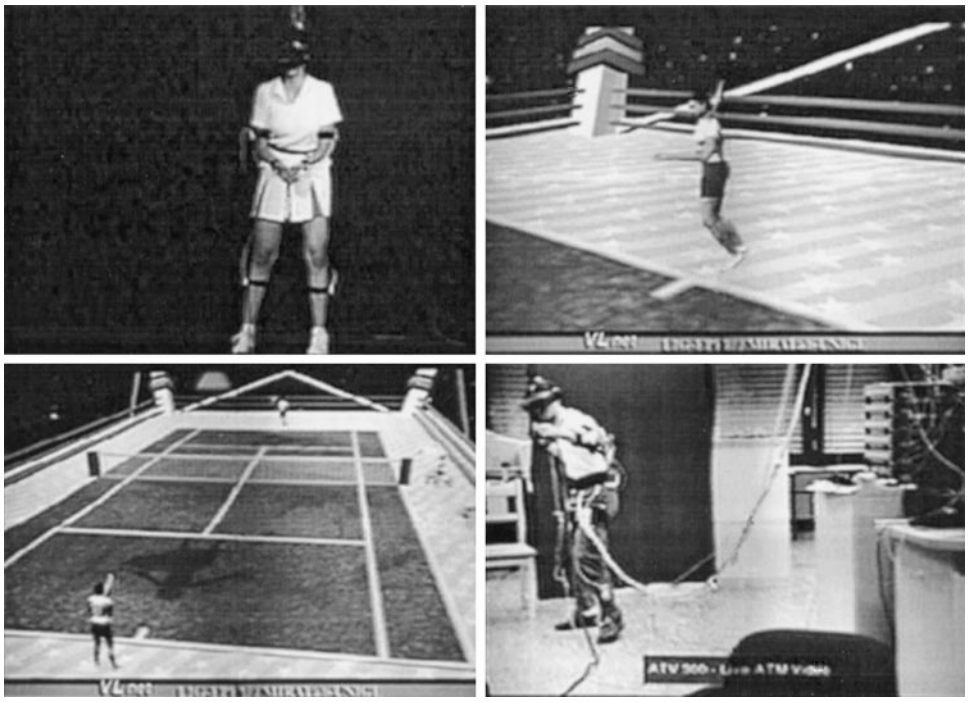

Figure 15. *Real-time tennis game using MC.*

the indirect navigation. However, this nonintuitive stop command is quickly learned by players and allows them to reach the net, giving more interest to the game.

6.2 Visual and Tactile Feedback

The head-mounted display (HMD) viewing control is easily retrieved from the current posture of the virtual human head. However, we have locked the rolling (rotation around the frontal axis) degree of freedom of the camera that is used to render the scene because our HMD has a restricted field of view and no peripheral information is displayed. It is well known that, in the absence of motions in the peripheral field of the human vision system, the head / camera rolling is perceived as a floor rotation rather than the real head rolling. This phenomenon is even strengthened by the lag between the real head motion and the rendered visual feedback. During all the experiments, only a few people reported some light motion-sickness symptoms (Hettinger & Riccio, 1992) such as a general or visual tiredness and headache.

Our HMD has a relatively low resolution (247 × 230, 57,600 triads) compared to NTSC, and it is quite difficult to distinguish small objects from a certain distance. Therefore, to help the player track the ball, we have slightly increased the size of the virtual ball. We used several other simple yet effective methods to provide better game feedback to the player. The visual tracking of the ball is improved with the help of the ball shadow, which is rendered using a dark, flat object that simultaneously follows the ball path but is vertically projected on the ground. For depth perception and court visibility, the designers built scenes comprising a textured spherical sky and a transparent net and racket while keeping the total amount of textures under the maximum available resources to prevent a performance decline. In such an immersive environment, the collision between the ball and the racket can hardly be perceived visually by players, but the collision feedback is greatly improved by using sound event (real collision sound) and glove vibration using the Cybertouch glove option at the impact time.

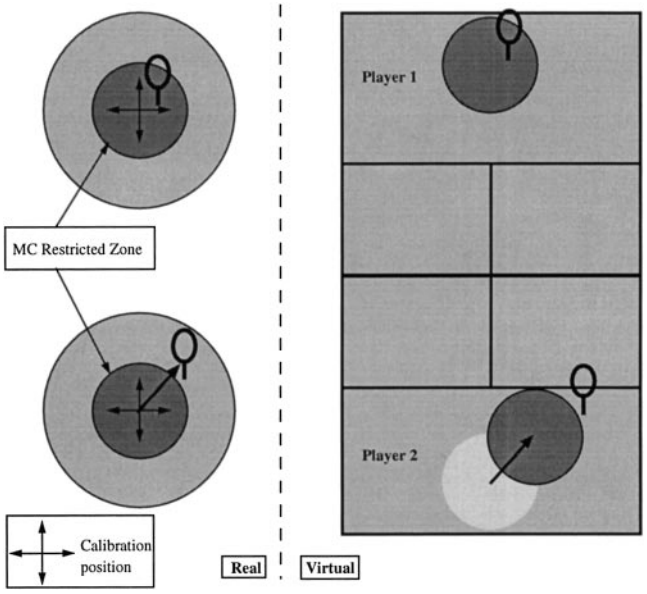

Figure 16. *Player 1 is in the MC restricted zone, and his avatar moves with reasonable motion amplification; Player 2 is outside the MC first perimeter: the virtual perimeter's global position moves.*

Special data glove gesture commands help to remotely trigger operations in a similar way as that described by Molet, Huang, Boulic, and Thalmann (1997) and without the requirement of any intermediate user. We used four basic remote commands: magnetic sensor calibration (Molet et al., 1996), service launching, changing the game level, and resetting the game. All these functions are activated by distinct hand postures.

7 Experiments and Results

7.1 Equipment and Configuration

A game session basically consists of three VLNET clients, two player clients associated with fobnet (motion-capture controller), and one client for the referee and ball driver. In addition, we usually launch a fourth client without human representation. This last client is responsible for the management of the virtual video camera, allowing the rendering of the game from various points of view for nonimmersed spectators (otherwise, only the players would enjoy the game). The frame rate at the player and referee levels should be, at least, eight frames per second (visual feedback) and postures per second (motion feedback) for proper playability. Otherwise, the ball and the players' motion are hardly perceived by participants. Similarly, the ball controller included in the referee driver needs sufficient racket position updates to be able to effectively compute the ball-racket collision detection and response. In the virtual tennis demonstration between Geneva Telecom Interactive '97 and the Computer Graphics Lab in Lausanne, we employed the following hardware. (See Figure 17.)

At the Geneva site, two Silicon Graphics Onyx 2 were used for the first player client and the Virtual Video Camera client. Both hosts were connected to one Impact over local Ethernet. The Impact contained one ATM card and was used as a router for fast communication with Lausanne. The VR devices comprised a MotionStar from Ascension with fourteen sensors, one Virtual Research VR4 HMD, two Cybergloves from Virtual Technologies, and one Spaceball (from Spaceball Technologies, Inc.) used to drive the virtual video camera.

At the Lausanne site, one Onyx was used for the second player client, and two Impacts were responsible for the referee client and for the VLNET server. These three machines used ATM cards for communicating with the server. The VR devices were identical to those used at Geneva except for the magnetic sensors: a set of sixteen Flock of Birds from Ascension Technology (only fourteen were used in the motion-capture process).

During the exhibition, the audience could simultaneously watch four different displays on large screens (Figure 10 and Figure 15) while hearing the sound emitted by the Geneva player client. Two screens contained the video images of both players filmed with (real) video cameras. The video stream of the Lausanne player was transmitted to Geneva over a dedicated ATM line. The third screen showed a copy of the subjective view of the Geneva player (bottom-right image in Figure 10) and the last display was devoted to the output of the virtual video camera conducted by one animator (top-left images in Figure 10 and Figure 15). The animator was directing the shooting of the game, in a TV-like manner (e.g., close-up view of the players and referee, general views, smooth camera motion, etc).

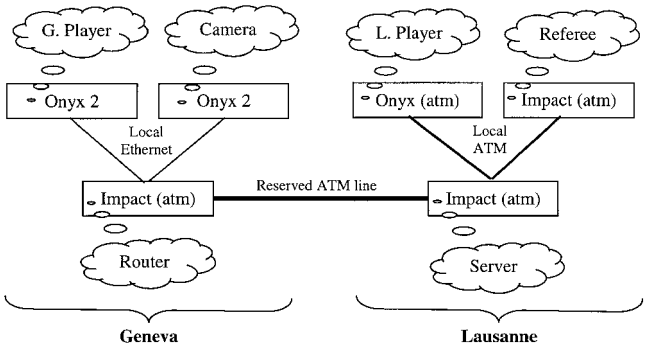

Figure 17. *Telecom Interactive'97 configuration.*

7.2 Results

For this project, a motion-capture program, a tennis ball simulation program, a virtual camera control program, and an autonomous referee program (each written by different persons in different cities) needed to be integrated within the same environment. Furthermore, new versions of software were developed simultaneously; therefore, it was difficult to integrate software within one application. Additionally, the time to develop the whole application was only four months. VLNET architecture allowed each developer to work independently and to make tests every week for half a day with new versions of other programs. The system could satisfy the requirements of this demonstration within the time constraints, and the resulting NVE simulation speed was sufficient for the high interactivity requirements of the game.

The choice of the ATM network was made mainly because of its reserved and constant flow. Over the Internet, although the bandwidth is (usually) sufficient, message-packing first delays the updates and, as a result, updates are sometimes all made at the same time—producing an unpredictable animation of the ball. Naturally, other network technology with similar functionality can replace ATM without any change at the application level.

From the player's point of view, the different game levels corresponding to different response to ball-racket collision (Section 5) were ranged from ''too easy'' for the ''missile'' ball behavior, to ''impossible'' for the physically based response. We used the missile ball technique extensively during development and earlier experiments of the game because it allowed us to play in single-player mode by just placing a second standalone racket in the opposite side of the virtual court. During the exhibition, we mostly used the mixed-response mode. This mode modifies the ''missile'' behavior by taking into account the velocity of the racket at the impact time. We activated the racket-determined method only when no fault occurred during more than ten consecutive hits. Only trained players could reach this state. No immersed player ever succeeded in hitting the ball using the physical-response technique; the ball would always fly far away from the desired place. Obviously, there are many limitations in the system compared to real tennis: the lack of visual information in the peripheral field, the MC matching between the real player and the avatar, the visual feedback frame rate and lag, and the discrete nature of the collision detection are all involved in the final result. Nevertheless, we could investigate and enjoy new game situations that would be impossible to produce in real life (like the missile ball behavior). Furthermore, one achievement of this prototype system is that it showed that VR technologies are mature enough to build entertaining applications and are no longer reserved to VR specialists. After a few hours of training, Melanie Thalmann, who had never experienced VR immersion before, managed to play ''live'' virtual tennis in Geneva.

8 Conclusions and Future Research

We have presented a virtual tennis game based on the VLNET framework. The game brings together different computer technologies and software including network, motion capture, virtual reality, shared environment, and controlled and autonomous virtual humans.

In future research, we want to improve the real-time deformation module in the spine region of the humanoids. This will permit us to activate the multijoint control of the spine (Molet et al., 1997) in the motion-capture process without sacrificing the look of the virtual human surface. We already tested its use at the player level, and we found that the spine, hand, and head motions were greatly improved. This motion-capture accu-

racy reflects strongly on the subjective views rendered for the HMD, because the captured position and orientation of the head induces the viewing parameters. We especially noticed the enhancement in the service phase when the ball is high above the virtual shoulder, as it is much easier to see and hit. Moreover, as the precision of the hand's spatial position increases, controlling the racket becomes more natural to players.

Using sound localization techniques could bring another improvement for better ball localization. It might, however, be too expensive in our situation, where the frame rate is a critical issue.

The main reproach about virtual tennis is that it attempts to achieve two antagonist goals at the same time. One goal is to demonstrate that current VR technology allows the creation of immersive 3-D games in which participants share the same virtual world and can act / interact using their real body movements. The other goal is to present a showcase application for many aspects of the virtual human simulation. The showcase focus diverges from the computer game logic in the sense that it requires huge resources to display the best possible virtual human as opposed to what would be sufficient to embody a virtual tennis opponent at the current HMD screen resolution. One of the main concerns in the game context is to ensure the best possible frame rate, whereas a close-up of a low-resolution virtual human would not score high viewership ratings. A nice solution would be to use levels of detail for virtual humans with separate controls at each VLNET client according to their own viewpoint locations. This way the divergent goals would be unified without compromise.

Acknowledgments

We would like to thank the designers who have contributed to this performance: Jean Claude Moussaly, Nabil Sidi Yacoub, Laurence Suhner from MIRALab at the University of Geneva, and Patrick Keller and Mireille Clavien from LIG, EPFL. We are also grateful to Chris Joslin for proofreading this document, Luciana Nedel for the referee voice, and Mireille Goud and Jacques Flumet for the ATM network support.

This research is funded by the Swiss Priority Project on Information Systems (SPP).

References

Badler, N., Phillips, C., & Webber, B. (1993). Simulating humans. *Computer Graphics and control.* New York: Oxford University Press.

Bers, J. (1996). A body model server for human motion capture and representation. *Presence: Teleoperators and Virtual Environments, 5*(4), 381–392.

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. (1995). The HUMANOID environment for interactive animation of multiple deformable human characters. *Proceedings of Eurographics '95, Computer Graphics Forum 14*(3), 337–348.

Boulic, R., Bécheiraz, P., Emering, L., & Thalmann, D. (1997). Integration of motion control techniques for virtual human and avatar real-time animation. *Proceedings of the ACM International Symposium VRST'97* (pp. 111–118). Lausanne.

Capin, T. K., Pandzic, I. S., Noser, H., Magnenat-Thalmann, N., & Thalmann, D. (1997). Virtual human representation and communication in VLNET networked virtual environments. *IEEE Computer Graphics and Applications, Special Issue on Multimedia Highways, 17*(2), 42–53.

Carlsson, C., & Hagsand, O. (1993). DIVE—A multi-user virtual reality system. *Proceedings of IEEE VRAIS '93,* Seattle, Washington.

Hettinger, L. J., & Riccio, G. E. (1992). Visually induced motion sickness in virtual environments. *Presence: Teleoperators, and Virtual Environments, 1*(3), 306–310.

Hirose, M., Deffaux, G., & Nakagaki, Y. (1996). Development of an effective motion capture system based on data fusion and minimal use of sensors. *VRST'96, ACM-SIGGRAPH and ACM-SIGCHI* (pp. 117–123).

Kalra, P., Mangili, A., Magnenat-Thalmann, N., Thalmann, D. (1992). Simulation of facial muscle actions based on rational free-form deformations. *Proc. Eurographics '92,* Cambridge (pp. 59–69).

LeBlanc, A., Kalra, P., Magnenat-Thalmann, N., & Thalmann, D. (1991). Sculpting with the ''ball & mouse'' metaphor. *Proceedings of Graphics Interface '91,* Calgary, Canada.

Litwinowicz, P., & Miller, G. (1994). Efficient techniques for

interactive texture placement. *SIGGRAPH Proceedings 94* (pp. 119–122).

Macedonia, M. R., Zyda, M. J., Pratt, D. R., Barham, P. T., & Zeswitz, S. (1994). NPSNET: A network software architecture for large-scale virtual environments. *Presence: Teleoperators and Virtual Environments, 3*(4).

Molet, T., Boulic, R., & Thalmann, D. (1996). A real-time anatomical converter for human motion capture. In R. Boulic and G. Hégron (Eds.), *Eurographics Workshop on Computer Animation and Simulation '96,* (pp. 79–94). Springer-Verlag Wien.

Molet, T., Huang, Z., Boulic, R., & Thalmann, D. (1997). An animation interface designed for motion capture. *Proceedings of Computer Animation '97* (pp. 77–85). Geneva: IEEE Press.

Noser, H., Pandzic, I. S., Capin, T. K., Magnenat-Thalmann, N., & Thalmann, D. (1996). Playing games through the virtual life network. *ALIFE V, Oral Presentations,* (pp. 114–121). Nara, Japan.

Noser, H., & Thalmann, D. (1997). Sensor based synthetic actors in a tennis game simulation. *Proceedings of Computer Graphics International 1997* (pp. 189–198). Hasselt, Belgium.

Pandzic, I. S., Capin, T. K., Lee, E., Magnenat-Thalmann, N., & Thalmann, D. (1997). A flexible architecture for virtual humans in networked collaborative virtual environments. *Proceedings of Eurographics '97,* 177–188.

Sannier, G., & Magnenat-Thalmann, N. (1997). A user-friendly texture-fitting methodology for virtual humans. *Computer Graphics International '97.*

Semwal, S. K., Hightower, R., & Standfield, S. (1998). Mapping algorithms for real-time control of an avatar using eight sensors. *Presence: Teleoperators and Virtual Environments, 7*(1), 1–21.

Shen, J., Chauvineau, E., & Thalmann, D. (1996). Fast realistic human body deformations for animation and VR applications. *Proceedings of Computer Graphics International '96* (pp. 166–174). *IEEE Computer Society Press.*