Many virtual environments deal with moving objects and articulated human-like 3D characters. Powerful tools for scripting complex movements and behaviors have been created, but coming up with realistic virtual humans or actors still remains a challenge.

Realistic virtual humans (and avatars) are as important as the virtual environment in which they reside, and their creation has been one of the main goals of most virtual reality systems. Methods for real-time rendering and natural ways for interaction and communication are related techniques for bringing true complexity and realism to all aspects of virtual environments.

For many years users could visit virtual worlds using a real-time interactive system, to interact and share experiences with others connected only via a simple network. However, because of the computing power such systems require, until recently you had to use high-end Unix-based computers. The powerful machines now available to consumers incorporate inexpensive high-performance graphics cards. These machines have enabled the NVE systems once dominated by Unix to be designed and implemented on a Windows OS system using only a few techniques to maintain rendering speeds.

We present Virtual Park (VPark), which consists of two systems running under Windows NT. The Attraction Builder enables the creation and editing of animations for use within the NVE system. Windows Virtual Life Network (W-VLNET) then loads and manages the attractions as a complete networked attraction. The system handles multiple attractions and users efficiently. In general, the design requirements for an NVE system include:

- Participant embodiment
- Network topology specific to virtual environments
- Data and task distribution scheme for scalability
- Dedicated communication protocol

Our goal was to develop and integrate several modules into a system capable of animating realistic virtual humans in a real-time performance. This includes modeling and representing virtual humans with high realism and simulating human face and body movements in real time. Realism becomes quite important in NVEs, where the communication among participants is crucial for their sense of presence.

**Attraction Builder**

One of the main goals for this research was to develop a complete tool for creating attractions in virtual environments in which virtual actors play the main parts. In our system, we give users a set of powerful functions to direct highly realistic virtual actors. High-level actions are provided to avoid low-level descriptions for an actor’s every move. We define an attraction as a set of geometrical and temporal descriptions of the virtual actors and objects in the virtual environment. In this work, we focus on a number of issues related to creating and animating virtual actors in attractions:

- Avatar model and animation data acquisition
- Real-time performance in rendering and animation
- Control on virtual human actions
- Effective and easy-to-use attraction creation methods
- Support for standards (Virtual Reality Modeling Language (VRML), Moving Pictures Expert Group-4 (MPEG-4)) for scalability and flexibility

**Virtual avatars**

The representative and behavioral realism of virtual humans is our system’s key feature. We provide virtual humans with crucial functions to aid their visual perception and allow them to perform normal tasks (such as walking, speaking, and interacting generally). Realistically represented avatars and actors (autonomous virtual humans acting out a part) coexist in the same
environment (Figure 1). We consider face, body, and speech animation an essential set of virtual human simulations, since they play an important role in our everyday communication. Facial expression and animation play an essential role in communicating human emotions. At the same time, speech animation and the corresponding lip movement prove even more important to aid in communication.

**Face and body models.** We used textured polygonal mesh representations of virtual actors from different sources. Face models were either from our in-house modeling tool⁴ or from the automatic method of generating clones from two photos.⁵ The latter allows participants to represent themselves effectively. In any case, face models are generated from the modification of a generic model that users can animate with the face animation module.

Bodies for virtual actors are represented using VRML97 for their segments. By supporting the Humanoid Animation Working Group’s H-ANIM,⁶ users can employ models available over the Internet. The generation of H-ANIM individualized body models from two photos is ongoing.

**Face animation.** The MPEG-4 facial animation standard is based on feature points located at several key places on the facial mesh geometry (such as the tip of a nose, the corner of the lips, and so on). There exist 68 facial animation parameters (FAPs), 66 of which are low-level parameters affecting these feature points directly. Specifying the displacement of these feature points with respect to their neutral position achieves the animation. These displacements are specified in terms (units) of certain feature distances of the facial model. For example, FAPs related to eye movement are expressed in terms of the horizontal distance between the pupils, whereas those related to lips are specified in terms of the distance between the corners of the lips. We call these feature distances facial animation parameters units (FAPUs). The two high-level FAPs are visemes (visual counterparts of phonemes) and expressions (sadness, happiness, and so on). Figure 2 shows some of the predefined expressions applied to one of our face models.

**Body animation.** Virtual human body animation is achieved by applying a set of degree-of-freedom (DOF) values to each body joint over time. To comply with VRML97 and to provide independence with respect to proprietary embodiments, we chose to directly animate H-ANIM compliant models. These models can be obtained from a number of sources and tend to increase in number, thus greatly enhancing representation flexibility for users. Compliance with MPEG-4 is also ensured by using body animation parameters (BAPs) for animation. A BAP specifies the relevant body joint DOF at a given instant in time. At the users’ level, however,
some predefined gestures are available from the user interface so that they have rather high-level control over virtual actors (Figure 3).

**Speech animation.** We obtain text-to-phonemes using the Microsoft Speech software developer’s kit. From the phonemes generated, corresponding visemes are generated as a set of FAPs so that they can be processed by our face animation engine. The system moves the generated visemes to a buffer to be synchronized with phonemes.

**Adding scenarios to an attraction**

Even with a powerful set of animation engines, it’s still a challenge to provide a simple but effective user interface to control, manipulate, and animate objects and virtual humans. We considered an interface based on script language. Although powerful, it required specialized computer programming expertise from users. We chose instead a graphics user interface in combination with a high-level action specification. In this approach, users can direct an actor by choosing an action from the menu. The selected action is then simulated by any of the engines provided.

Here, an action may be an emotion, a gesture, or a sentence. Each high-level action is considered a basic unit of animation. The system then assembles collections of basic units into an animation sequence comprising an attraction together with a geometric scene description. Users are given tools enabling them to adjust the animation’s duration, move in the time line, edit animation units, play back to see current status of the animation at any time, and load predefined animations and save current ones. Figure 4 shows the time line for an animation sequence composed of six basic action units—note that several action units apply simultaneously to an actor.

The key to linking the entire animation sequence and the animation engines lies in a timer that synchronizes the actions to feed them into an animation engine at the correct time, shown in Figure 5.

Building actions on a single time line maintains the system’s ease of use by letting users visualize what might happen at any point on the time line.

Control points may be added to the time line to change its sequence, so the time line isn’t limited to running in a linear, start-to-finish order. Interactivity can be added as follows:

- Break points stop the animation—the user can continue at the same time or choose another time.
Trigger points allow interaction with the scene to control the time line. They can be set either by selecting a specific object or by moving inside the object’s proximity range.

Index points, which are labeled on the time line, allow, for instance, multiple-choice dialogs for users to select an option to continue (perhaps an answer to a question). Figure 6 shows an example course through the time line when control points are activated.

Scenarios are added to an attraction using the Attraction Builder to combine all the separate elements into a single scenario. Figure 7 shows how the various applications combine to produce the final scenario.

You can build a scenario in three stages:

- **Combine graphical elements.** Import the separate VRML files from a standard modeling package into the Attraction Builder. The system then manipulates the objects to set the scene.
- **Add the animation.** Files can be added either using the Attraction Builder’s key-framing feature, which allows the positioning and orientation of objects and the recording of those positions, or by importing pre-recorded animation files.
- **Incorporate the interaction.** This involves adding control points into the time line.

### W-VLNET NVE system

VPark is a framework for distributed virtual environment applications in which human-like embodiments represent real humans. While the Attraction Builder creates or describes permanent attraction components including self-animated virtual actors, the W-VLNET system loads and manages these attractions for connected users to join the attractions as their embodiments.

### System architecture

The W-VLNET system builds on a previous architecture implemented on Unix (specifically, Irix from SGI). The previous system, also called VLNET, was written and optimized for Unix and uses a multiprocess and shared memory architecture. A shared memory protocol was used as the communication medium for processes (each process performing a different task). Although the W-VLNET system is similar, we redesigned the underlying communication architecture since Windows doesn’t fully use the shared memory architecture. Therefore, we implemented the architecture using a special communication system, and the concurrent task management used threads instead of processes.

Threads are more prevalent in Windows and impose fewer restrictions on the design since they don’t require shared memory to communicate. We combined the control and execution of threads with the communication architecture to create a faster communication mechanism. The communication uses a first-in, first-out (FIFO) buffer, which allows flow control over the communication while maintaining a fast transfer of data. The FIFO buffer basically allows any thread to communicate with any other thread.

The thread manager creates a thread for each task to be executed. By assigning a priority value that depends on the task type, the thread manager prevents tasks that should be executed quickly from being blocked by simple tasks (such as GUI control). The thread manager also controls the termination of threads and limits the number of executing threads, since too many running con-
currently would cause a system to slow down. In practical tests, the upper limit is much greater than required.

Because it’s not realistic to customize the thread manager for each specific task, the remaining control tasks for each thread are left to the module to determine and manage. These tasks include mutual exclusion, global memory control, and wait states. Because threads can run on multiprocessor systems and be distributed across these multiple processors, the only requirement for the individual modules themselves is that they be separated enough to use the multiprocessor architecture but singular enough that it doesn’t create dependencies. Figure 8 shows the communication among the main modules running through the system manager (the collective of the thread manager and the FIFO buffer).

**Plug-ins**

We designed the communication and thread managers with plug-ins in mind. We also designed it to be easily expandable, which was the key limitation of the previous system. The plug-ins used are much the same as other plug-ins—adding the change, adding, and editing of any module without recompilation. Since all the main components are also plug-ins, the system can be upgraded without users requiring major changes to the software. We encourage users to design plug-ins for the system to do a specific task they might require. A software developer’s kit designed to aid users in understanding the plug-in concept specific to this system is available.

**Scene manager**

In its basic form, the scene manager resembles the one used in the Attraction Builder, and in the same way the system manager controls the system, the scene manager controls all aspects of the scene. Although OpenGL Optimizer controls the actual scene graph, many additional interactions must be taken care of. Since there are multiple clients with multiple avatars, the database needed to manage this is quite complex. Instead of the system controlling, for example, Avatars 1, 2, 3, and so on, it controls Avatar 1 for Client 1 and Avatar 1, 2, and 3 for Client 2. This causes more complications in the overall design, especially since the Attraction Builder is effectively a linear, single-threaded system, whereas the W-VLNET system processes highly concurrent tasks throughout its architecture.

**Database control.** The first layer of the two-layer database, the client layer, contains a simple reference to the client. Below this, the Item layer (an Item being either an object or an avatar) contains the references for all objects and avatars in the system, including their names, scene graph reference(s), and locking switches. It’s important to keep track of these objects and avatars in a very strict fashion, since many complex things can happen (for example, a client could leave or join, crash, get disconnected, and so on), and these can have an adverse effect if not handled correctly. As mentioned previously, concurrent tasks can be performed on the same object or avatar. Hence, it’s necessary to keep track of whether users are interacting with an object or avatar. The inability to keep track of these events will also have awkward outcomes (such as strange animations or objects and avatars ending up in different positions on different clients).

**Avatar loading and animation.** We use bodies compatible with H-ANIM and MPEG-4 in conjunction with faces compatible with MPEG-4 (these are the same ones used in the Attraction Builder). Each client is expected to load at least one representative avatar, which also must be uploaded to the server and distributed to the other connected clients. These avatar files are compressed into Zip files to facilitate their transfer to the server (normally seven to eight times smaller than uncompressed files), but even these files are between 600 Kbytes and 1 Mbyte. Hence, we implemented a twofold caching mechanism to reduce wait times and bandwidth use. The first caching works in the normal way, finding whether a copy of the file exists locally (this mechanism also works for object files), then transferring just the smaller basic information (such as posture or position). If users have a small network connection to the server, the second caching mechanism supplies a default avatar representation (also stored locally) for all avatars, hence reducing the requirement to download other client’s representative avatars.

Animation is also more complex on the NVE system, since it’s done completely on a frame-by-frame basis. Each frame (of either BAPs or FAPs) is compressed using a simple lossless compression technique, then sent directly to the server and distributed to other clients using a sequential numbering system. This means that all clients can stop their animation at any time or adjust it accordingly—there’s no set time for which an animation can last. This works equally well both for animation streams stored in files and for motion tracking units. The lossless compression reduces a body animation’s...
In some highly interactive attractions, such as Virtual Dance, the animation of virtual humans can be directly produced by mimicking the users’ gestures using real-time motion capture systems.

Overall packet size, since animation of all the joints can produce up to 296 values 4 bytes in size (with other necessary data the packet becomes almost 1 Kbyte). Because the normal maximum transmission unit (MTU) is 556 bytes, this is too large for normal Internet transmissions, where the restricted MTU size is usually 576 bytes. The lossless compression uses the upper and lower limits of each of the 296 values and reduces the size of each value to the maximum bit value required. Even in worst-case conditions we use only a maximum of 110 values. Hence, the packet can be compressed roughly to within the MTU restriction. A quantizer value could be used to reduce this value, at a cost of reducing the accuracy of the animations, but a better approach would be to use either Huffman or arithmetic coding to produce better lossless compression. Quantizing the values reduces the packet size very little, at a cost of very poor animations.

**Picking and object manipulation.** Full interaction with a virtual environment requires object picking and manipulation. Since the system uses a collaborative environment, an object moved by one client must move for all other clients.

Picking is done on the basis of the client’s representative avatar. When it moves toward an object, the system selects all objects within the view frustum and a specific range (variable, with default of 1 meter) as pickable objects. The database of pickable objects changes dynamically as the avatar moves around. The pick mode cycles through all pickable objects stored in the database, then picks whichever object users select (either by pressing a button or by moving once the object was picked). The system moves the object with the avatar as it moves, much in the way an object moves in real life. Users can deselect the object by unpicking it. The system sends all object movements to the other clients so that their databases are completely up to date.

**Proximity, collision detection, and gravity.** Proximity detection provides greater interaction within the virtual environment. The proximity function is a collaboration of several common functions—proximity, collision detection and response, and gravity—which can be turned on or off as necessary according to requirements and computing power. Actually, gravity isn’t directly combined into the task but does work in hand with collision detection and response. Gravity is applied to each object or avatar (defined by the server as the default object) apart from the basic scene—the system stores the speed of each in the database and applies a simple gravitational equation to each. This equation is designed to be fast (in real time) and to move the object or avatar a large finite distance each time it’s applied.

Proximity and collision detection are handled in the same loop. The latter checks for all impending collisions of objects and avatars with others, then implements the response mechanism to prevent their actual intersection.

The proximity function is involved in this calculation also, checking whether an object or avatar is less than a set distance away. This is used mainly in conjunction with the attraction player, which plays files output from the attraction builder on the local client. Since proximity triggers are specified in the Attraction Builder they must be present in the NVE system, making it more complex since any user can trigger sensors and those events must be handled correctly. The attraction player (according to its input file) specifies a proximity sensor to be applied to either an object or an avatar and a set of trigger conditions (triggered by an object only, an object or avatar, or the local user’s representation) plus the proximity distance.

Collision detection and response at this time (to preserve the real-time aspect) is simple. Collision detection occurs by placing a bounding box around each object, then detecting simple intersections. The response mechanism is currently designed only to stop an object or avatar from causing an intersection. This response mechanism and the application of gravity work together—when the gravity mechanism is used, the response mechanism must be implemented. We expect to develop more complex response mechanisms in the future.

**Real-time motion tracking**

In some highly interactive attractions, such as Virtual Dance, the animation of virtual humans can be directly produced by mimicking the users’ gestures using real-time motion capture systems.

**Body posture and tracking.** The real-time motion capture engine is based on a set of 14 magnetic sensors that measure the motion of the major human limbs (head and spine, shoulders, elbows, hips, knees, and ankles). Optionally, two digital gloves track wrist and finger movements. The sensors’ raw measurements are converted into anatomical angles suited to skeleton hierarchies using an efficient technique. This converter is driven by orientation measurements to remove as fully as possible dependencies on the distorted (non-linear) position measurements of magnetic sensors. Only one sensor position recovers the virtual human’s position. The key features of this engine are:

- **an automatic instant sensor calibration procedure,**
- **human-specific optimizations such as dedicated evaluation for shoulders and hips twisting and corrections for skating across a floor,** and
- **control of the whole spine using three sensors.**

The motion-capture engine is a dedicated external...
application that sends BAPs to the W-VLNET core, which in turn applies the posture to the virtual human before final scene rendering. That way, we can spread the computational load over separate processors. This introduces a slight lag (~0.5 second) between the performed movement and the rendered related posture. However, we found it more effective than the pipelining solution, where all steps are performed within the same application, creating a lag that varies between 0.3 and 0.7 second depending on the rendered scene complexity.

**MPEG-4 BAPs.** The human motion-capture process is built on top of a proprietary skeleton structure that models joints using Euler angle sequence decompositions. These angles resemble the MPEG-4 BAPs. To animate MPEG-4 or H-ANIM hierarchies, we translate the joint angles from our internal hierarchy to the MPEG-4 BAPs. These computations consist basically of finding the MPEG-4 counterparts (or indexes) for each joint angle and applying simple data encoding (our internal angles are float values and MPEG-4 parameters are encoded as long int). In a few cases (for example, finger parameters), there’s a slight posture difference between our internal model and the MPEG-4 default postures. We adjust for this default difference by adding angle offsets prior to encoding. These offsets are identified by setting the proprietary hierarchy in the MPEG-4 default posture using key-framing.

After encoding all parameters, the new posture information is sent to the client application using the transmission-control protocol (Figure 9). A simplified virtual human representation can be displayed within the external motion-capture application to provide diagnostic-level feedback. We use this feature mainly to determine incorrect sensor positioning and other hardware-related problems.

**Networking**

The network topology used for this NVE system is based on the client-server approach. This approach assumes that all the clients requiring common interaction are connected to one server. The server hosts one or more attractors and scenes, contains the master scene database, and controls the distribution of data to all clients.

**Client connection.** Each client connects to the server via a single entry port. As soon as the connection is established, the server moves the client to another port to keep the entry port free. The new port is then established as the control port between the server and the client and is used as a secure data exchange for network interaction. This connection uses TCP.

The server then exchanges information with the client to establish its identity and the channels it wishes to connect to. It tests the data connection to determine a rough estimate of the bandwidth, then sets up several channels according to the client’s request, as follows:

- Stream is used for data that must be transmitted rapidly and steadily. It requires no retransmission of data, in case of loss or error. The port is connectionless using the user datagram protocol to transfer data.
- Update resembles the stream channel, requiring only a UDP connectionless port, but it has error control (using retransmission), so data sent is treated with more care.
- File data (or large amounts, greater than 1 Kbyte) are transferred over a TCP-oriented port. Complete error control and packet resend are implemented for this channel.
- Control is used during the client-server phase and stays connected, using TCP, until the end of the session.

The client generally will need the update channel and has no option for the control channel. However, it can deny connection for the stream and file channels—this might be to preserve bandwidth or CPU processing time (or perhaps for the application if it’s not required elsewhere). The update channel transfers data such as object or avatar transforms and avatar animations. The stream channel mainly handles audio or video connections that might be required, for instance, in sending real-time voice communication.

The system disconnects in reverse order. The channels are disconnected first and then the control port sends a command to the server to disconnect completely. The server can force disconnection in the same way, which ensures a clean disconnection of ports and lets the server accept new connections without restarting.

**Scene graph initialization.** Once the main network connection has been established, the scene can be sent to the client. It’s actually split into two sections, the world and the attraction. The world is just a single object (normally a grass plane) that serves as an absolute reference for the remaining objects and avatars (especially when using gravity). The attraction is a complex set of objects and avatars placed directly on the world (with an offset if desired); overall control of it is handled by the attraction player plug-in, but the scene graph loads objects or avatars.

All objects or avatars are checked against a database using a caching mechanism to avoid unnecessary downloads. Both the attraction and world files are compressed to obtain maximum transfer and uncompressed directly into the cache.
Server database. After the client connects and downloads the main world and attraction—and once the main static objects and avatars (common throughout the server’s online status) have been downloaded into the scene graph—the system consults the server database to determine all the dynamic objects or avatars in the scene. All clients at any time during their connection can add their own avatar(s) and object(s) into the scene to enable greater interaction.

The scene database contains information about objects and avatars that have been uploaded by clients, their transformation matrices, and their file references in the server’s local cache (which works exactly like the client cache). This information is distributed to each client when that client uploads the information. When a client connects for the first time, the system searches the database and sends all details to let the new client be completely up to date with the scene’s current state.

Avatars also have an extra field that stores the avatar’s body posture. Neither audio nor video streams are stored in the database.

When a client disconnects, the system removes its uploaded avatars from the database and distributes informational messages to all other clients. However, since objects may still be in use by other clients, their ownership moves to the server (which has its own client ID) to avoid problems with later connecting clients.

Communication protocol. We used a common communication protocol over the update channel to enable simple message passing. This protocol uses a generic packet that contains fields for common data types and three generic fields that provide access for other units.

- Message type identifies a message packet, declaring its contents.
- Animation stream uses 400 bytes for different types of animation and data (FAP, BAP, text, and so on).
- Message string is a 32-byte text identifier (such as a filename).
- Message value 1, 2, and 3 handles general values and references.

- Transformation matrix is a 4-by-4 float value, used because most objects or avatars will require transformations in nearly every packet.

The stream channel uses a simplified version of this, with a message identifier and a transformation matrix for each packet of 500 bytes of compressed data. The file channel splits all data into manageable packets and sends them directly over the channel. If the receiving end doesn’t acknowledge that all data was received correctly, a packet-by-packet resend message is transmitted to the sending end. The control channel receives undefined messages regarding the server’s state (connections, load, client status, and so on). The client then has a rough database of the connected clients (to reduce network load, updates to the client database are done on very large time steps).

Multimedia objects

Different types of data can be exchanged between clients, using the server as a network switch. The list of data types and streams currently added is as follows:

- Audio stream is transferred at 16 kilobits per second (Kbps) and compressed using the G.728 audio compression codec. However, for systems with larger bandwidths or with smaller bandwidths but greater CPU power, the G.711 (64 Kbps) and G.723.1 (5.3 Kbps and 6.4 Kbps) audio codecs both function on the same audio channel. Each audio stream is given a reference object so that 3D audio can be created.
- Speech communication over this type of system is useful for clients connected over low bandwidths. The speech itself is transmitted as plain ASCII text and is passed to a text-to-speech engine. This not only converts the text into its audio equivalent but it creates the corresponding visemes.

Attraction playing and management

To make the system completely clear and comprehensible, not only for the system’s design but for the plug-ins that might be added later, the attraction itself belongs to the server. Because the server has its own client ID as well as a server ID, it’s easy to add avatars and objects to the database. Once the client has connected to the server, the attraction player plug-in loads the attraction into the scene graph using APIs from the scene manager. The attraction player uses the same caching mechanism and compression techniques as the scene manager. Loading an attraction consists of loading the autonomous avatars, animated objects, script, and proximity and touch sensors placed around the scene. The server then sends timing information to the attraction player plug-in to enable it to synchronize with the other connected clients. Since the time-line is set to a linear time placement (although not necessarily linear when played), the time reference applies to a specific set of postures and placements for avatars and objects, respectively. This timing signal is sent to all clients every second (by default) to enable clients to maintain synchronism (although each client maintains its own timer).
The scene manager sends the attraction player information about the scene (such as proximity sensors that have been triggered), and therefore the attraction player controls the attraction itself, without any intervention from the scene manager. If a generic NVE system is required, the attraction player plug-in can be removed without affecting the rest of the system. Likewise, the server attempts communication with the attraction player, and if there’s no response it continues regardless.

**Attractions**

We’ve built two showcase attractions to demonstrate the effectiveness of our approach and to underline the broad range of possibilities it offers to creative minds.

**Virtual Theater**

The first of two case study attractions, the Virtual Theater, allows users in a real sense to join the cast, resulting in a real-time, dynamic theater production that changes within a predefined framework according to the interactions of the digital actors and the participants.

In our theater, up to three digital actors perform a predefined script in the absence of any users’ avatars. A user’s avatar enters the scene and either watches the predefined drama unfold or approaches the digital actors to initiate interaction (Figure 10). Proximity triggering of the script causes the digital actors to cease their current activities and interact with the user’s avatar.

Lewis Carroll’s *Alice in Wonderland* provides the basic script, and the scene is based on Carroll’s photographs of Alice herself from the 1860s, taken from the collection of the National Museum of Photography, Film, and Television in Bradford, UK. The Alice scenario is character driven rather than plot based and provides a variety of immediately recognizable characters representing different ages, genders, shapes, and social types, so most of the museum’s visitors should be able to identify with one or more of the characters. Its episodic narrative lets a nonlinear storyline be enacted without contradicting expectations of the basic scenario. The story’s imaginative imagery allows features possible only within animation, whether computer or cell based. The unnatural setting also gives license to freely experiment and play with words and actions.

We adapted Carroll’s text to provide opportunities for interaction between avatars and virtual actors, as well as between avatars. The latter breaks the time line of the piece, which returns to a linear nature when the users cease their interaction. The scene progresses without users present, allowing people to join the theater at any time. Visitors to this exhibit can choose to act as Alice, the Queen of Hearts, or Tweedledee or experience the different perspectives of all three. The museum has worked with a theatrical production company to research ways of making the attraction lively and accessible. It’s also conducting visitor research into how best to provide maximum engagement between the participants, fluency with the proposed interface, and understanding of the storyline. With the museum’s education department, we’re also researching potential applications of the virtual theater in schools and live-link events.

**Virtual Dance**

The second case study is an attraction for a teacher to teach dance (Figure 11). The teacher is attached to a motion tracking system at the University of Geneva while the student is attached to one at EPFL. Our in-house cloning system cloned the teacher and the student to obtain virtual copies of them. An overlaid music sequence enabled the teacher and student to synchronize with each other. Each can see the other (virtually) on a screen (Figure 12), so the teacher can evaluate the
student and the student can study the teacher.

The system is fully interactive, allowing participants to talk to each other. This scenario receives maximum benefit from an NVE system and would be difficult to replace with other conventional systems (such as videoconferencing). The scenario isn’t limited to two participants, provided the motion-tracking equipment were available. A teacher could have a whole class of students. A lightweight head-mounted display could be used to increase the teacher’s sense of submersion and enable a clearer perception of the situation, although it should be rugged and well secured to the teacher so that large dance movements aren’t restricted.

Future work

We’ve integrated various pieces of research to form a Windows framework for users to build believable attractions effectively and easily. Our W-VLNET multi-threaded system is capable of not only running the attractions but connecting two or more users in an NVE made more real by the integration of a system that completely animates the user’s virtual representation. The two attractions we created and tested let us visualize our research. In future work we aim to augment the overall experience by improving the collision models, the depth of the multimedia inputs (including video), and the audio perception in the environment. Better compression methods and server filtering in real time should improve the transmission rates for the system.

Acknowledgments

The VPark project (Advanced Communications Technologies and Services, or Acts, project number AC353) is funded by the European Community. The Swiss partners are sponsored by their Federal Office for Education and Science.

References


Chris Joslin is a research assistant and PhD candidate in MiraLab at the University of Geneva. He earned an MS in electronic and communication engineering from the University of Bath (UK) in 1997. Currently he’s working on networked virtual collaborative environments and specializing in networking, multiprocessing, video, and audio compression combined with virtual environments.

Tom Molet is a senior researcher at MiraLab, University of Geneva. He received his PhD in computer science in 1998 at the Computer Graphics Lab, EPFL (Swiss Federal Institute of Technology). His main research interests are virtual human modeling, animation and virtual reality, and more specifically, human-motion capture and performance animation. He has developed motion capture techniques based on various hardware devices, including digital gloves, magnetic sensors, and optical systems.

Nadia Magnenat Thalmann is Professor and Director of MiraLab at the University of Geneva, Switzerland. She studied at the University of Geneva and obtained several degrees including psychology, biology, computer science, and an MS in biochemistry. She also holds a PhD in quantum physics and computer graphics, from the University of Geneva completed in 1977. She has received many awards for her work including the
Moebius Prize for the best multimedia system awarded by the European Community, “Best Paper” at the British Computer Graphics Society congress in 1993, election to the 1993 International Academy in Brussels, and also to the Swiss Academy of Technical Sciences, in 1997. She has directed and produced several films, among the latest are Cyberdance (1998) and Fashion Dreams (1999). She is also editor in chief of the Visual Computer Journal. She can be contacted by email at Nadia.Thalmann@miralab.unige.ch.

Joaquim Esmerado is a PhD candidate at the Swiss Federal Institute of Technology. He obtained an MS in computer science. Research interests include realistic image synthesis, animation, virtual human interaction with virtual environments, and complex virtual objects based on knowledge.

Daniel Thalmann is Professor and Director of the Computer Graphics Lab at the Swiss federal Institute of Technology in Lausanne. He has published more than 250 papers in graphics, animation, and virtual reality. He co-authored several books including Avatars in Networked Virtual Environments (John Wiley and Sons). He is a co-editor-in-chief of the Journal of Visualization and Computer Animation. He is a co-chair of the Eurographics Working Group on Computer Simulation and Animation and member of the Executive Board of the Computer Graphics Society.

Ian Palmer is a senior lecturer in the Electronic Imaging and Media Communications Department of the University of Bradford. He has a BS from the University of Sussex, an MS in digital systems from Brunel University. In 1994 he received a Dphil in computer animation from the University of Sussex. His research interests are in intelligent virtual environments, and computer animation techniques and algorithms.

Nicholas Chilton is currently working as part of the Simula Initiative team for the Media Tech Secretariat at the University of Bradford, UK. He received an MS in virtual environments in 1995. Until 2000 he was a research assistant in the Electronic Imaging and Media Communications Department at the University of Bradford, UK. His main research interests are user-friendly humanoid avatar animations and virtual environments for theatrical uses.

Rae Earnshaw is Dean of Informatics, and Professor and Head of Electronic Imaging and Media Communications at the University of Bradford, UK. His research interests include imaging, graphics, visualization, animation, multimedia, virtual reality, media, art, design, and the convergence of computing, telephony, imaging, digital media, networking, and broadcasting. He obtained his PhD in computer science from the University of Leeds. He is a member of the editorial boards of The Visual Computer and IEEE Computer Graphics and Applications; editor in chief of Virtual Reality: Research, Development, and Applications; vice president of the Computer Graphics Society, chair of the British Computer Society Computer Graphics and Displays Group, and a fellow of the British Computer Society. He is a member of ACM, IEEE, and Eurographics.

Readers may contact Joslin at MiraLab, Univ. of Geneva, Centre Universitaire d’Informatique, 24 Rue du General Dufour, CH-1211, Geneva 4, Switzerland, email joslin@miralab.unige.ch, http://www.miralab.unige.ch.

IEEE VR 2001
13-17 March 2001
Yokohama, Japan

Sponsored by the IEEE Computer Society Technical Committee on Visualization and Graphics and the Virtual Reality Society of Japan, IEEE VR 2001 will feature panels, tutorials, workshops, demos, and exhibits.

Conference schedule:
Tutorials and workshops: 13-14 March 2001
General sessions: 15-17 March 2001
Exhibits: 15-16 March 2001