Computer Methods and Programs in Biomedicine 80 Suppl. 1 (2005) S61-S70

ELSEVIER

Computer Methods and Programs in Biomedicine

www.intl.elsevierhealth.com/journals/cmpb

Building interactive virtual environments for simulated training in medicine using VRML and Java/JavaScript

D. Korošec*, A. Holobar, M. Divjak, D. Zazula

University of Maribor, Faculty of Electrical Engineering and Computer Science, Smetanova 17, 2000 Maribor, Slovenia

KEYWORDS
Medical training;
Virtual environments;
VRML;
Neonatal resuscitation;
Virtual newborn

Summary Medicine is a difficult thing to learn. Experimenting with real patients should not be the only option; simulation deserves a special attention here. Virtual Reality Modelling Language (VRML) as a tool for building virtual objects and scenes has a good record of educational applications in medicine, especially for static and animated visualisations of body parts and organs. However, to create computer simulations resembling situations in real environments the required level of interactivity and dynamics is difficult to achieve.

In the present paper we describe some approaches and techniques which we used to push the limits of the current VRML technology further toward dynamic 3D representation of virtual environments (VEs). Our demonstration is based on the implementation of a virtual baby model, whose vital signs can be controlled from an external Java application.

The main contributions of this work are: (a) outline and evaluation of the three-level VRML/Java implementation of the dynamic virtual environment, (b) proposal for a modified VRML `TimeSensor` node, which greatly improves the overall control of system performance, and (c) architecture of the prototype distributed virtual environment for training in neonatal resuscitation comprising the interactive virtual newborn, active bedside monitor for vital signs and full 3D representation of the surgery room.

1. Introduction

Learning and training using computer-based systems helps improving knowledge and skill acquisition process in many fields. Medicine is no exception [1]. Even more - as a field dealing with particularly delicate objects, namely human beings, it is often inherently limited with respect to training possibilities. There is usually little room for errors and exercise with real patients, who are expecting expert treatment instead of novices.

Traditional medical education has two stages: theoretical study and hospital apprenticeship. There is a gap between the two which could be closed by learning and training in simulated conditions so as to avoid 'learning by accident'

*Correspondence to: D. Korošec. Tel: +386 31 321 921.
E-mail: dean.korosec@uni-mb.si (D. Korošec)

0169-2607/$ - see front matter

phenomena. Computer based devices and software used in such training are often called computer simulators, although this is a very inexact term, as it should cover everything from, for example, tens of thousands dollar worth patient simulators capable of realistic gas exchange, blood flow and heart activity; over complex simulation devices for surgical and endoscopic procedures; down to software-only models and study tools or small programs for visualisation of pharmacokinetic graphs, and more [2].

In general two kinds of simulations should be distinguished first: (a) 'real-world' simulated situations, and (b) virtual environments (VEs) based on computer models and animations. With both approaches, students should work through predefined scenarios and training tasks, although involving a mentor to supervise such training is in both cases necessary to provide higher complexity and realistic assessment.

A good example of training using simulated exercises is the system of crisis management training in neonatal resuscitation [3] introduced by L. Halamek, D.M. Gaba and their team at the Veterans Affairs Palo Alto Health Care System [4]. It can provide a significant degree of immersion and a very realistic experience, but the price is high: special training facilities with instruments, monitors, mannequins, and skilled senior staff to control and supervise the training are necessary. A virtual environment alone can not completely replace such experience, but it can help students to be better prepared for such situations and thus make the whole process more efficient. The two approaches are complementary - while certain skills (e.g. intubation, heart massage, etc.) can be better trained with real equipment and mannequins, others (for example assessment of vital signs) can be more realistically pictured using interactive 3D graphics. We believe that modern medical education should include all four mentioned stages of the learning path, as shown in Fig. 1.

Let us now focus on applications of virtual reality (VR) technology, which received a lot of attention

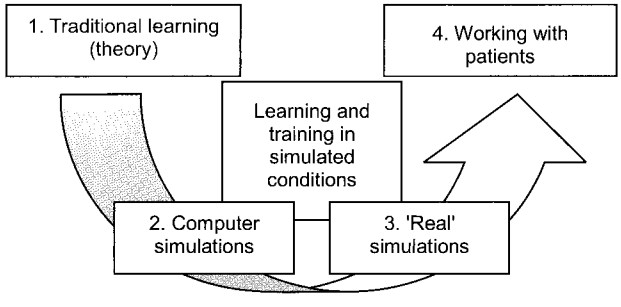

Fig. 1. Learning and training in simulated conditions can close the gap between theory and practice in medicine.

some years ago with the development of cheap computer power and prevalence of the Internet. Medicine has traditionally been one of the major application fields for VR, with medical education being the first area in which it made significant contributions. Others include remote and local surgery, surgery planning, treatment of phobias and other causes of psychological distress, skill training, pain reduction, and more [5].

On the other hand, one has to admit today that VR in general has not quite lived up to its initial expectations. Why don't we see more VR-based systems in our professional activities (not to mention daily lives) yet? While one reason is certainly that VR hardware is still extremely expensive and immature, we believe that the main reason for such situation is the lack of good software standards and their efficient implementation on common computer platforms. Practically only one such solution, Virtual Reality Modelling Language (VRML) [6], exists currently for building open standard applications. Most VRML applications that have been built using this open standard for Web VR are still mainly static visualisation of 3D objects and perhaps some simple animations. While their educational value should, of course, not be underestimated, as many excellent examples also in the field of medicine and biology demonstrate [7], the full potential of VRML lies further ahead.

In the past, we have carefully studied VRML and used it to create various virtual environments. Our challenge and aim has always been to make them as dynamic and interactive as possible, because we believe the success of experience with virtual environment is strongly related to these two components. The culmination of our efforts is a virtual delivery room [8], which we designed to support training of proper procedures in resuscitating newborns.

This paper presents some aspects and lessons learned while implementing a dynamic 3D representation of vital signs of a virtual newborn [9] with VRML, and is organised as follows: after briefly introducing a three-level scheme for implementing the dynamic interactive virtual environment in the following section, we present the main content in the third section, where we describe (a) the performance issue of the VRML `TimeSensor` node and its solution, (b) the virtual newborn as the main object, and (c) the complete architecture of our system for training in neonatal resuscitation. In the last section some conclusions based on our experiences are given together with a brief announcement of the X3D standard, which will soon replace and extend VRML.

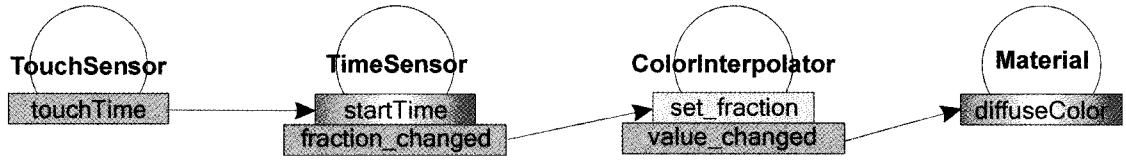

Fig. 2. Example of a basic animation cycle - gradual change of the object's colour is initiated by clicking on an object, related to `TouchSensor` first. This then triggers the `TimeSensor` to start sending events to the `ColorInterpolator` node, which scales received values of the cycle fraction to an appropriate colour space. The `EventOut` fields are dark, the `eventIn` fields are bright grey and exposed fields are iridescent.

2. Methods

2.1. Virtual Reality Modelling Language

Despite its attractive name, VRML [10] is neither virtual reality nor a modelling language. It is not virtual reality because it does not require or preclude immersive 3D experience. Also, its geometric modelling features are too scarce (and, on the other hand, some others are too advanced) to consider it a modelling language. VRML is (a) a 3D file interchange format including hierarchical transformations, light sources, viewpoints, basic animation, material properties, and texture mapping; (b) a language for publishing 3D Web pages; (c) a model for integration of images, 3D objects, text and sound into three dimensions; (d) perhaps an element of the very beginning of the future cyberspace as the on-line virtual community.

The major part of a VRML file is a hierarchical 3D description of a scene, called the scene graph. Its elements are nodes; more than 50 types of nodes are defined as standard. Each node contains a set of fields keeping data of predefined types.

2.2. Three levels for implementation of dynamic objects with VRML

Basic animation cycle - Level 0

The VRML 2.0 standard, which soon evolved into the still valid VRML 97 standard, has introduced considerable improvement over the original proposal, i.e. event routing mechanism, which enables animation, interaction, and behaviour. Fields may receive (`EventIns`) and/or generate (`eventOuts`/exposed fields) events (messages) via routes between the nodes (`ROUTE` statement) [11]. Simple animations can be predefined with the following sequence: an instance of the `TimeSensor` node, in which duration of the animation (`cycleInterval` field) is defined, can continually send events to an instance of the interpolator node. From this interpolator, which keeps the range of changing values, the output messages with current interpolation values are then (again continually) routed to the appropriate field of the scene graph node. Animation is obtained because the VRML browser permanently renders (frame by frame) the scene graph and thus immediately reflects any changes of values within it.

In our prototype training system (described in Sections 0 and 0 below), all animations at the level of geometric objects were implemented using this mechanism; an example is schematically depicted in Fig. 2.

The problem with this mechanism is twofold: (a) only basic cyclic animations can be created and (b) the user has no control over its performance. The first point can be successfully addressed by a scripting interface (SI), as described in the following subsection. But the second issue (performance control) is rooted deeper in the philosophy of VRML. The problem and our proposed solution are presented in Section 0.

Scripting - Level 1 (SI)

A scripting interface (SI) has been defined to extend the abilities of VRML beyond those of the predefined nodes [11]. The `Script` node can be used to implement additional complexity in an external programming language, for example performing calculations, improving animation or communicating with other local and remote programs. No programming language is prescribed in VRML standard, but most browsers support Java [12] and JavaScript [13]. The main advantage of JavaScript as a VRML scripting language is that the source code can be inserted directly into the `Script` node, avoiding the need for compilation and use of many small source files. On the other hand, the execution of Java byte-code is faster than other scripting languages. And since only byte-code is transmitted across the network, it makes it more difficult for someone to take the source code and modify it. As a general rule, JavaScript is the right solution for tiny scripts that do not do very much; for large scripts that perform complicated calculations, access the network or construct GUI, Java is the better solution [14].

A script node can contain an arbitrary number of fields, and processes all received events by a method (user program) specified in the `URL` field.

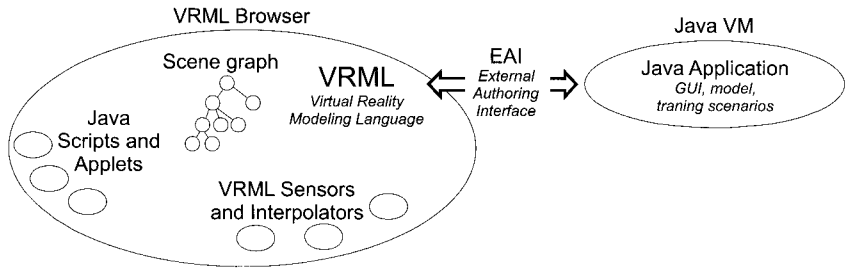

Fig. 3. Typical application built with VRML and Java runs within the Web browser.

Along with the actual value also the timestamp is received. The events generated by the `Script` method have the same timestamp as the event which triggered them to enable scheduling and synchronisation mechanisms, although a certain amount of time is needed to process the events on a real system during execution.

In our prototype training system we used scripting to implement representations of all vital signs (see the example for respiration in Section 0).

External programs - Level 2 (EAI)

Script programs (Level 1) are activated upon receiving events and, therefore, under control of the VRML browser. As result, they can significantly decrease its rendering performance. Sometimes we also need to implement extensive program logic as part of a complex VR application. To solve both issues, we can connect an independent application written in any program language with the VRML browser using the External Authoring Interface (EAI) [15]. Although not officially part of the VRML 97 standard, it is supported by most browsers [16], especially for Java [12].

If applications run independently of the VRML browser multi-tasking/threading is managed by the operating system. In the ideal case, users should thus have better control over allocation of processing time between rendering (VRML browser) and program logic (external program). However, because most practical applications involve Java where often even both VRML browser and Java Virtual Machine (JVM) run as plug-ins within the HTML browser (Netscape or Internet Explorer) it is difficult to treat each one as a separate process (Fig. 3).

Most of the system functions not directly related to 3D rendering are usually contained in external applications - in the case of our training system for neonatal resuscitation system (Section 0) this includes the interface to a motion detecting device, multi-user audio conferencing, part of the user interface (dialogs), control over training scenarios, and storing of all data (student motion and actions, mentor interventions, conversation) during the training.

2.3. Implementation of complex behaviour over all three levels

When creating complex interactive VEs, all three implementation levels coupling VRML with Java/JavaScript (or other general purpose programming language) have to be used. But, as usually in programming, there is more than one possibility to implement certain object behaviour. It is therefore important to know about the possibilities and limitations of each particular implementation level, to be able to spread the design of our planned system behaviour among them in the most efficient manner. A list of, in our view, main advantages and disadvantages of each level is summarised in Table 1.

3. Results

3.1. Performance control of Level 0: Extension of the `TimeSensor`

Implementation of the basic animation mechanism is left to the browser, which handles the described 'continuous' timer-to-interpolator and interpolator-to-node message passing (as in example of Fig. 2) in its own way based on its own internal scheduling. But in practice not all animation cycles have the same importance: in our application, for example, we wish to assign high priority to animations related to the virtual baby and lower priority to some other, less important virtual objects. But even if priorities were available this would not solve the performance issue completely. In our observation, the VRML browsers consume all available processor time (100% CPU usage) even if only a simple animation is predefined in the scene. This behaviour is caused by the true nature of 'continuous' message passing as it is implemented in the browser: 'continuous' in this case means 'the highest possible frequency of consecutive discrete events'. Quite an elegant solution in the ideal case and on ideal computers,

Table 1
The main properties of levels for implementation of dynamic behaviour in VRML

Level	Advantages	Disadvantages
Level 0	Basic VRML mechanism Simple and standard Supported by all browsers	Functional limitations Inability to control browser performance
Level 1: Scripting	Solves functional limitations of Level 0 Standard Supports several languages	Synchronous operation with respect to the VRML browser core Can respond only to triggered events Interferes with the performance of the rest of the VRML visualisation Does not support external fields
Level 2: EAI	Full functionality Asynchronous operation	Incompatibility of versions Not supported by all browsers No access to unnamed nodes No access to meta-information on the nodes No access to simple fields

but it causes a lot of troubles in a more complex real VRML application running on real machines and in parallel to other programs.

The solution of the animation performance problem lies at its core: the `TimeSensor` node. We know now that all its instances in the scene are treated equal by the browser and there is no option to either define their priorities or to bound the frequencies with which they generate messages for interpolators. Having these two serious limitations in mind we decided to create a new implementation of the `TimeSensor` node; we called it `Timer` [17]. The `Timer` node is an instance of the `Script` node and has almost exactly the same interface as the original `TimeSensor` node, except for the extra field `delay` of type `SFType`. Its default value is set to 0.1 second (meaning ten triggered messages per second).

The `Timer` node is written in Java language using threads. Our idea at first was to join the triggering mechanism for all timers in a common `TimerQueue` class, which maintains the list of all active instances of the `Timer` nodes ordered according to the triggering time. The event triggering thread constantly checks the list, comparing the current system with the triggering time of the first timer in the queue. When these two become equal, appropriate events are sent from the `eventOut` fields of the corresponding `Timer` node, and later one is reinserted in the queue at a place set according to its delay parameter. The triggering thread then enters `sleep()` until the time for a new timer event comes. However, the mechanism with the single triggering thread proved to be quite inefficient in case of larger number of nodes. CPU load itself was not the main problem, but we started to lose events at higher triggering frequencies (shorter delays) and larger number of nodes - the 10 ms resolution of the Java `sleep()` function [in the Java Virtual Machine (JVM) within the Netscape Communicator we used] proved to be inadequate.

Therefore we slightly adjusted the design and developed a second replacement for `TimeSensor` - `Timer2`. Here each instance has its own triggering thread. Although the opposite had been expected, our tests have shown that there is little difference in CPU load between `Timer` and `Timer2`. However, the ratio between the actual and demanded frequency of events proved (as shown in Fig. 4) to be much better in the case of `Timer2`. This proved that (at least with the JVM used) multithreaded code with longer `sleep()` delays is more efficient (can exploit CPU better) than one single thread with shorter `sleep()` delays.

Performance tests showed that CPU load can be significantly reduced using our `Timer2` (and `Timer`)

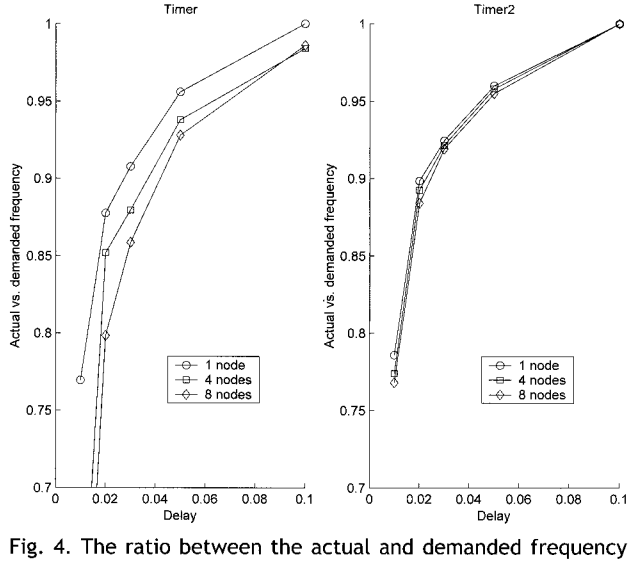

Fig. 4. The ratio between the actual and demanded frequency of generated events with respect to the set timer delay (1/frequency) and different number of timer instances. Left: `Timer` node, right: `Timer2` node.

instead of `TimeSensor` nodes. At window sizes up to approx. 500×300 pixels (Pentium II 200 MHz, 64 Mb RAM, Windows NT 4.0, Netscape Navigator 4.6 and VRML Cosmo Player 2.11) the observed CPU load is less than 5% for a single cube object with four interpolators (position, colour, orientation and size) and 10 Hz imposed message triggering frequency. We suggest setting this frequency slightly higher than the number of frames our VRML browser is able to render per second, which is, of course, scene and computer dependent.

3.2. Virtual newborn

As part of our prototype virtual delivery room, which should complement neonatal resuscitation training, we developed two dynamic virtual objects with built-in dynamic behaviour: (a) a virtual model of a baby (avatar) and (b) a virtual bed-side monitor. The virtual baby (Fig. 5) is the central and most complex object, incorporating several independent animation mechanisms to represent vital signs. These signs, relevant in neonatal resuscitation according to ref. [18], are: (a) respiration; (b) heartbeat, (c) skin colour [representing oxygenation - parts of the body can change colour from normal pink to blue in extremities (peripheral cyanosis) or all over the body (central cyanosis)]; (d) activity level (movement of extremities); (e) facial expression; and (f) crying. Table 2 shows the list of vital signs, their audio/visual representations and the implementation levels used for each of them.

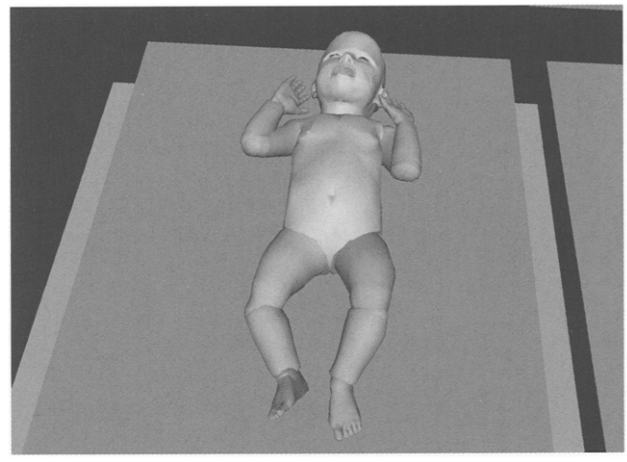

Fig. 5. View of a virtual baby.

One might think that simple periodic animations (Level 0) would be sufficient to represent these signs, but it is more complicated than that. The first reason is that we want to be able to update their control variables (like heart rate and respiration rate, for example) from an external program and change them at any given moment. The second reason is that, for example, the respiration and heart activity periods consist of several phases and cannot be treated as one simple cycle. A scheme of the complete scripting mechanism combining and controlling the phases of the respiration cycle is presented in Fig. 6.

Table 2
The interpretation and visualisations of different newborn's' vital signs. Crosses denote the corresponding levels of implementation

Vital sign	Interpretation, visualization	L0	L1	L2
Breathing	Chest movement	×		
	Sound	×		
	Graphical curve and current breathing frequency value displayed on virtual monitor	×		
	Interpolation of breathing frequency between successive respiration cycles		×	
	Control of breathing frequency over time			×
Heartbeat	Sound	×		
	Graphical curve and current heart rate value displayed on virtual monitor	×		
	Interpolation of heart rate between successive beats		×	
	Control of heart rate value over time			×
Colour of skin and lips	Skin and lips turn from normal to blue and back	×		
	Interpolation of the skin (separately for torso and extremities) and lips colour		×	
	Control of the colours over time			×
Motion	Movement of the baby's head, arms and legs	×		
	Interpolation of the motion intensity		×	
	Control of the motion intensity over time			×
Cry	Replaying of the real baby crying audio clips	×		
	Facial mimics (lips, eyes and forehead motion)	×		
	Selection of the audio clips, synchronization of the moans with facial mimics			×

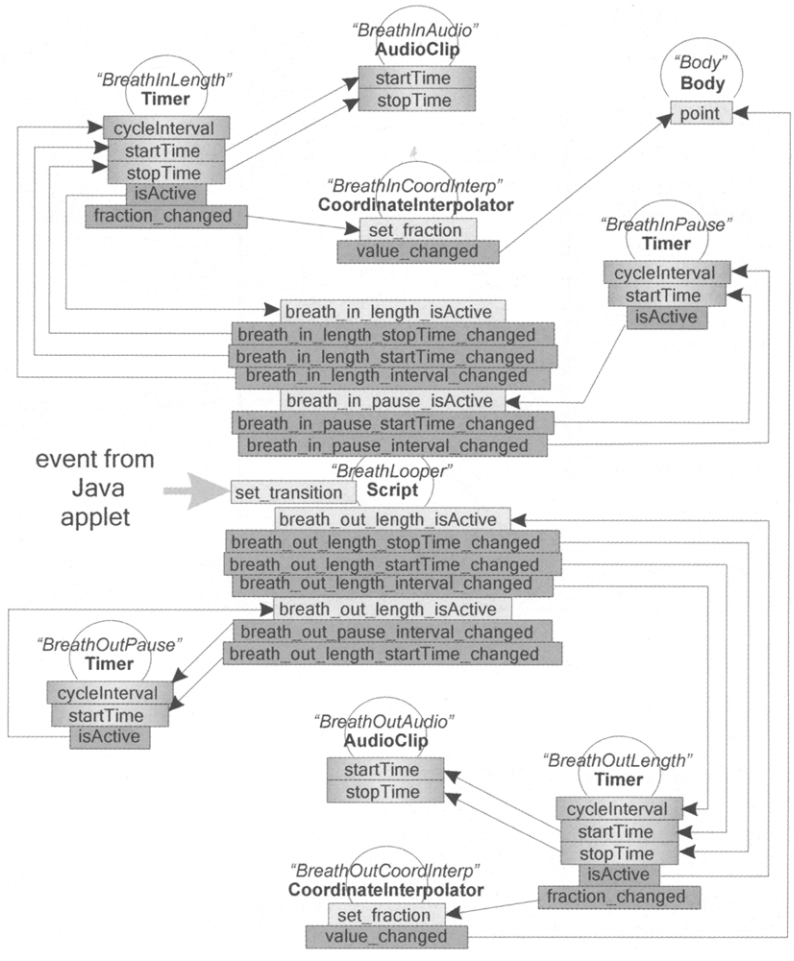

Fig. 6. Schematic structure of the newborn's respiration mechanism implemented with VRML nodes: `Breath(In/Out)CoordInterp`, and `Breath(In/Out)Interpolator` nodes move newborn's chest and respiration curve on the virtual monitor, respectively; `Breath(In/Out)Audio` nodes replay audio clips of the real newborn breathing; `Breath(In/Out)Length` and `Breath(In/Out)Pause` nodes dictate the durations of the different respiration stages; `BreathLooper` node receives the commands from the mentor's module and controls the variations of breathing frequency in time. Circles depict different nodes with their names (italics) and types (bold), field types are coded in the same manner as in Fig. 3.

3.3. VIDERO system architecture

The VIDERO (Virtual Delivery Room) prototype system for training in neonatal resuscitation is a distributed application [9], built using the Java Remote Method Invocation (RMI) mechanism [19]. The system runs on a set-up of two connected personal computers and has an option to interface with the Polhemus motion tracking device and a head-mounted display. It consists of the following modules:

- **Mentor application**, consisting of two parts: (a) 3D view of the dynamic virtual environment that the mentor shares with the student and (b) user interface for controlling the parameters of the virtual newborn, starting the training scenarios and receiving notification on student actions.
- **Student application**, must run on a different computer. The student navigates in the 3D virtual environment by a mouse or (optionally) the sensor of a motion tracking device attached to his head-mounted display. In the current prototype system the student can only choose actions from his menu-based interface. Each action has a corresponding set of (up to three) numerical parameters that the student has to set properly. The action chosen, and the value of the parameters, are shown on the mentor's screen immediately.
- **Control server** is the core program in the system. It takes care of the communication between the two main participants and possible other observers. It mediates changes of the virtual newborn parameters from the mentor to the student application and chosen actions from the student to the mentor application. Control server also initiates parts of the predefined training scenarios.

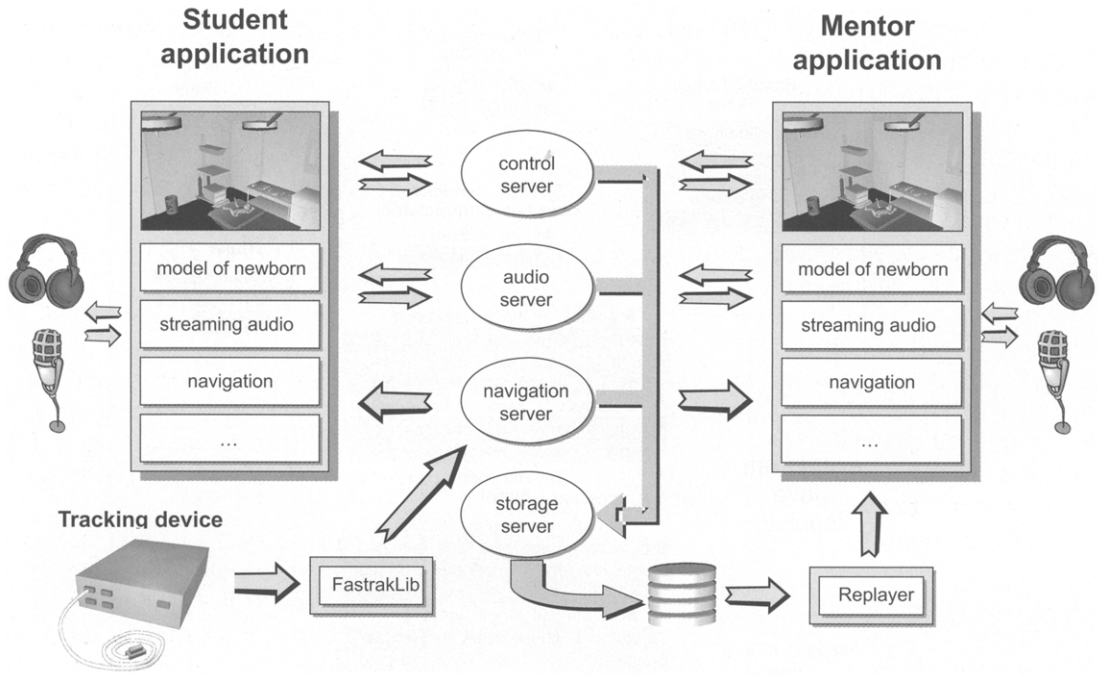

Fig. 7. Architecture of the VIDERO prototype system for training in neonatal resuscitation.

- **Navigation server** constantly synchronises the viewpoint of the mentor with the viewpoint of the student, who is in control of navigation in the 3D virtual world. This program also uses our own Java Fastraklib library [20] to read data from the (optional) Polhemus motion tracking device.
- **Audio server** is a program module which allows real-time audio conferencing between training participants [21].
- **Storage server** can be activated by the mentor to record all events during the training. This includes navigation (viewpoint changes), setting of the newborn parameters, student actions, and voice communication.
- **Replayer** is separate program which allows the stored training session to be repeated in real time exactly as it was carried out.

The structure of the complete VIDERO system is schematically depicted in Fig. 7.

4. Discussion

4.1. Technology issues

VRML advantages and limitations

VRML is the only free, widely available and open-standard solution for multi-platform 3D visualisation. It has several interesting features making it a good choice also for creating dynamic and interactive virtual environments simulating real situations. However, under the hood, VRML offers only a relatively low-level functionality for this kind of applications. Objects exist only as graphical descriptions (surfaces); no mechanisms are implied to account for their weight, material, elasticity, movement and other properties of the physical reality. Therefore also no relations among the objects can be investigated: within the generic framework even collisions can only be detected between the avatar and the objects and not among the objects themselves. All this can, of course, be justified in terms of the (non-)existing computer power that such representation would swallow - already rendering of somewhat more complicated scenes hits the limits of currently available equipment.

On the positive side, VRML offers many possibilities of interfacing to other programming languages. Thus, most of the application logic can be implemented separately in Java or C++ and the VRML browser is used only as a rendering engine.

VRML compatibility and performance issues

Besides the limitations built into the VRML specification itself there are other, more practical (and perhaps more annoying) issues. Freely available VRML browsers (like Cosmo and Cortona) do not equally support the entire VRML specification. While most of them will work well for basic rendering, unsupported features are mainly exactly those things we need for interactive dynamic simulations of the real world: efficient and deterministic event routing mechanism, various sensor implementations, 3D audio, communication to external

devices (motion tracking, data sampling, voice communication) etc.

Some issues can partly be addressed by custom developments, like in our case where we developed a replacement for the `TimeSensor` node with the ability to specify the frequency of generated output messages in order to limit the CPU usage. Preferably, such extensions should reside within the basic VRML specification.

The future of VRML: Extensible 3D (X3D)
The Web 3D Consortium has proposed a draft of the next generation X3D standard [22], which will bring closer XML, MPEG4 and VRML technologies. It remains to be seen whether new features will allow easier creation and better control of dynamic virtual environments.

4.2. New possibilities for medical training

VIDERO system for neonatal resuscitation
A computer-generated representation of a human being, a virtual newborn in our case, has one major advantage over the rubber doll, which is traditionally used in neonatal resuscitation training: several vital signs can be dynamically rendered by generating their visual and audio representation. Students, working on an example case, are now faced with more a realistic challenge - instead of listening to a verbal description of the baby's condition they have to continually assess visual and auditory clues from the baby itself and virtual devices around it.

In our model of training a teacher (mentor) remains the necessary key element: either he/she directly controls all observable parameters (heart rate, respiration rate, skin colour, facial expressions and movements) of the virtual baby or he/she predefines their trajectories over time by so-called scenarios. In our system, the scenarios are small sets of parameter changes in a specific period of time, and can be activated during a training session according to the actions the student chooses. Lists of possible actions and their corresponding parameters are again described in advance by the mentor. In this way the mentor, although physically separated from the student, is still the one fully in charge of the training process. This was the set-up we have chosen for our current prototype together with medical experts. Here the student is on his/her own, while the mentor is a more or less hidden, silent manipulator.

Other training modes
But the roles could also be reversed - the student would observe the mentor at work, while the mentor performs the correct procedures and comments on his actions. Our VIDERO system can be used for this kind of tutoring as well. The possibility to record a training session is another great option to improve medical training. The mentor can repeat the session exactly as it was performed and call the attention of students to particular details that were perhaps left unnoticed during the session.

The virtual baby alone can also be used as a complement to simulated training involving real devices. If students are using a real baby mannequin to practice some hands-on procedures, the computer-generated avatar on a display next to them can be used to create a more realistic impression of the simulated condition.

The future of training using virtual environments
The current state of virtual reality technology does not simply allow the setting up of various training environments at moderate costs. Extremely expensive, inefficient hardware and lack of higher level software standards put such training systems out of the domain of regular educational practice. On the other hand, numerous advantages can be found in favour of such training, some of them also mentioned in this paper, and it seems that medicine in particular is an appropriate application field. Therefore we believe that virtual environments and other derivatives of VR technology will find their place as complementary training methods in medical education of the future.

References

[1] S. Samothrakis, T.N. Arvanitis, A. Plataniotis, M.D.J. McNeill, P.F. Lister, WWW creates new interactive 3D graphics and collaborative environments for medical research and education, Int. J. Med. Inform., 47(1-2) (1997) 69-73.
[2] G. Meller, A typology of simulators for medical education, J. Digital Imag., 10(3 Suppl. 1) (1997) 194-196.
[3] L.P. Halamek, D.M. Kaegi, D.M. Gaba, Y.A. Sowb, B.C. Smith, B.E. Smith, S.K. Howard, Time for a new paradigm in pediatric medical education: Teaching neonatal resuscitation in a simulated delivery room environment. Pediatrics, 106(4) (2000) 106-110.
[4] L.P. Halamek, Development of a Simulated Delivery Room, presented at 7th Annual Medicine Meets Virtual Reality Conf. (MMVR 1999), January 20-23, 1999, San Francisco, CA.
[5] J.A. Waterworth. Virtual Reality in Medicine: A Survey of the State of the Art. Online resource at http://www.informatik.umu.se/~jwworth/medpage.html (July 1999).
[6] The VRML Specifications, http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/
[7] T. Amon, V. Valencic. VRML - enhanced learning in biology and medicine, Fut. Generation Comput. Syst., 17(1) (2000) 1-6.
[8] D. Korošec, A. Holobar, M. Divjak, D. Zazula. Dynamic VRML for simulated training in medicine. in: Proc. 15th IEEE Symp. on Computer-based Medical Systems, Maribor, Slovenia (2002) pp. 205-210.

[9] M. Divjak, A. Holobar, I. Prelog. VIDERO - virtual delivery room. in: Proc. Int. Conf. on Trends in Communications, IEEE Region 8 Student Paper Contest, Bratislava (2001) Vol. 1, pp. LIV-LVII.
[10] R. Carey, G. Bell. The Annotated VRML 2.0 Reference Manual (Addison-Wesley, Berkeley, CA, 1997).
[11] A.L. Ames, D.R. Nadeau, J.L. Moreland. VRML Sourcebook (Wiley, New York, 1996).
[12] The Virtual Reality Modeling Language: Java scripting reference, http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/part1/java.html
[13] The Virtual Reality Modeling Language: JavaScript scripting reference, http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/part1/javascript.html
[14] J. Hartman, J. Wernecke. The VRML Handbook: Building moving worlds on the web (Addison-Wesley, New York, 1996).
[15] C. Marrin, Proposal for a VRML 2.0 Informative Annex: External Authoring Interface Reference, http://www.vrml.org/WorkingGroups/vrml-eai/ExternalInterface.html (November 1997).
[16] EAI design notes, http://www.frontiernet.net/~imaging/eaifaq.html
[17] A. Holobar, D. Zazula. Improved control of events in the VRML 2.0 application, in: Proc. 6th Euromedia Conf., Valencia, Spain (2001) pp. 67-71.
[18] R.S. Bloom, C. Cropley, et al. AHA/AAP Neonatal Resuscitation Textbook (American Heart Association, 1994).
[19] Java Remote Method Invocation, http://java.sun.com/products/jdk/rmi/
[20] I. Prelog, D. Zazula, D. Korošec. Navigation inside virtual environments using Polhemus 3Space Fastrak tracking device, in: Proc. 9th Electrotechnical and Computer Science Conf. ERK 2000, Portorož, Slovenia (2000) pp. 91-94.
[21] M. Divjak, D. Korže. Visual and audio communication between visitors of virtual worlds, WSES Int. Conf. on Neural Network and Applications, Puerto de la Cruz, Canary Islands, Spain (2001) pp. 41-46.
[22] X3D - Extensible 3D, http://www.web3d.org/x3d/