Training Scenario Prototyping for VR-Based Simulation of Neonatal Decision-Making

A. HOLOBAR, M. DIVJAK, D. KOROŠEC, D. ZAZULA

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

Received 3 February 2006; accepted 15 November 2006

ABSTRACT: This paper presents the design and implementation of a real-time system for virtual reality (VR)-based training in neonatal medicine, with main emphasis on simple creation of various training scenarios. This system combines an articulated 3D model of a virtual newborn with text-based descriptions of its physiological and behavioral responses, enabling medical experts to easily construct, simulate and revise an arbitrary postnatal critical situation. Afterwards, the resulting descriptions of newborn's behavior can be used for technical specifications (and even for automatic generation) of more complex behavioral models, such as finite-state automata. © 2007 Wiley Periodicals, Inc. Comput Appl Eng Educ 15: 317–328, 2007; Published online in Wiley InterScience (www.interscience.wiley.com); DOI 10.1002/cae.20121

Keywords: virtual reality; decision-making; medical training; neonatology; training scenarios

INTRODUCTION

The ideas regarding virtual reality (VR)-based environments have slowly attained their current prominence over the past few decades. Various VR systems have been developed and strenuous efforts dedicated to the study of human sensing and control in virtual environments (VE). Today, although limited in their scope, successful VR-based applications exist in the fields of the military, aeronautics and recently in medicine. By combining demonstration and exploration capabilities they provide a unique supplement to standard education and training, which can now be done anytime and anywhere, without risks.

Although far from being trivial, the actual implementation of VR systems proves to be just one of the many possible problems. The major difficulties experienced in VR-based training development originate from the gap between the trainee's expectations (specifications) and the technical possibilities (implementation). Namely, today's computers still lack sufficient power to successfully imitate reality in its

Correspondence to A. Holobar (ales.holobar@uni-mb.si).

Contract grant sponsor: Slovenian Ministry of Education, Science and Sport; contract grant numbers: 3411-98-71-0001, S2-796-010/21301/2000, and Programme Funding P2-0041.

entirety. Moreover, quite often no technical description is available on the phenomena under investigation. As a result, different compromises have to be made between the functionality and representativeness of developed VR systems. Their exact definition proves to be a very complex task depending on both objective and subjective factors, such as costs, simulation's reliability and accuracy, modularity and versatility of the training procedures, trainee's acceptance of VR, trust factors, etc. While highly crucial, some of these factors are not even tractable in the initial stages of design or implementation. Consequently, various evaluation studies of different VR-based training systems report controversial results. While the trainees usually still prefer traditional trainings [1,2], VR systems are often demonstrated to provide superior efficiency [3,4,5].

A possible solution for the encountered problems is to build general simulation systems that can be calibrated to fit the varying needs of training professionals. This, however, still calls for close cooperation among the VR system's developers and its actual users. We believe a much better idea is to raise the programming abstraction level to provide a general, simple to use simulation tool to the training providers, which can be used to create and change the different VR courses. Such an approach has at least two major advantages. First, the complex VR entities, such as autonomous virtual patients, for example, become tools in the hands of field experts and slowly gain their final form and behavior through an evolutionary process. This enables course instructors to easily capture and constantly adapt the intent behind the VR-based training software. Second, the trainee's responses, critics and recommendations can easily be integrated into the system by the didactic professionals, without the need to contact the VR system's developers. This saves energy and time, as it improves software productivity and simplifies software maintenance.

This paper outlines an approach on how to design and implement a prototype of a high-level VR-based system supporting the time-efficient construction of neonatal decision-making training. The prototype enables simple programming and control of the virtual newborn using natural, person-friendly text-based descriptions. By introducing the need for open-structured and scalable models of virtual entities, the described solutions also provide a coherent treatment of the requirements one must have in mind when building general-purpose VR-based simulators. Moreover, the presented prototype can serve as a good example of how a cost-effective, complex, real-time VE can be implemented by means of platform independent software standards.

The paper is organized as follows. The section VR-Based Systems in Neonatology outlines the state of the art in VR-based medical training. The next section describes a VR-based prototype for the training of decision-making in neonatology, along with its implementation concepts. In the section Performance Evaluation basic performance parameters are outlined, while discussion in last section concludes the article.

VR-BASED SYSTEMS IN NEONATOLOGY

In medicine the VR systems addressing surgical education are most evident. Virtual training simulators are being used in the fields of arthroscopy [6], orthopedic surgery [7], cardiac surgery [8] and open surgery [9], to name just a few of them. Some excellent general purpose training systems for minimal invasive surgery with realistic user interfaces, modelling of deformable objects (soft tissue deformations) and fluid simulation (animation of organ bleeding) have been implemented by introducing more complex VR models [10]. Exploiting the scope of existing high-tech VR hardware and software, they are all based on technical, highly demanding descriptions of the phenomena under investigation.

One of the branches of medicine which demands extensive and constant training is neonatology. Neonatologists face highly critical situations when struggling for a newborn's life: their patients are incapable of communication, while their health condition can change rapidly. Successful treatment depends on immediate and valid intervention, which can only be based on the infant's observable and measurable vital signs. The focus of the training in neonatology can be directed to either manual skills (e.g. virtual endoscopy, various intubation procedures, etc.) or decision-making, that is, descriptive procedures, where their proper order, type and timing are observed. Hand skills are usually practized using special equipment: either real instrumentation on animal subjects or expensive, computer-based haptic interfaces [10]. Contrary to the hands skills, decision-making is based on personal capabilities of assessing the health condition of a newborn. Its training is less equipment dependent, but, nevertheless, highly demanding in regard to compliance. Namely, in their training courses, all the trainees should ideally face and go through any, even though potentially seldom, critical situation with no risk to the patient, and as many times as necessary. A need for the constant upgrading of training scenarios by medical experts becomes apparent when taking into account the total number of possible postnatal complications and their various combinations.

Building a computer-based training system for medical decision-making means that each training scenario can be represented by a predefined sequence of time events which change the patient's health state. While several aspects of the scenario's events can be distinguished, the ones altering the patient's heart rate, respiration conditions, skin color and behavior (responses) are the most evident. In the past, different control strategies have emerged for vital signs. A very promising solution was applied by Chi et al. in Reference [11], where parallel state-machines, called Parallel Transitions Networks, were introduced to control virtual patient behavior over time. Another possible solution was presented by Stansfield et al. [12], where injury models based on decision trees were used to map the virtual patient's most likely outcome. Both solutions are based on technically complex descriptions of training scenarios. Unfortunately, the medical (non-technical) experts lack the specific computer knowledge necessary to create these training scenarios. Consequently, all possible scenarios must be foreseen and prepared by software developers in close cooperation with different medical experts. Another, much more effective route is to define all possible VR events in a simple, text-based format enabling easy definition and modification of training scenarios by medical experts.

There is yet another important aspect when developing scalable scenario training. Namely, the trainee under evaluation should recognise the simulated conditions, assess the infant's condition and demonstrate the necessary procedures. As there are many possible postnatal complications, there are numerous feasible interventions. Consequently, all the trainee's actions in VE must also be defined by the medical experts. Moreover, as each intervention changes the course of training, the definitions of scenario events and trainees' interventions overlap.

In the sequel, a scalable prototype of VR-based training environment for neonatal decision-making, called virtual delivery room (VIDERO) is described. Following the need for versatility in the training process, the main focus is on simple control of the virtual newborn, and definition of training scenarios and possible interventions. However, in order to understand the interpretation of the scenario's events, the prototype's modular and scalable architecture is first outlined.

VIRTUAL DELIVERY ROOM

VIDERO was implemented in VRML [13] and Java programming language. The audio-visual interpretation of the virtual scene is performed by publicly available VRML/X3D browsers, while the standard VRML nodes are augmented with Java and JavaScript program scripts [14]. The modules controlling the training scenarios and trainee's interventions are implemented in a standalone Java application and connected to the VE by means of external authoring interface (EAI) [15]. From the trainee's viewpoint the prototype consists of two windows (Fig. 1): (a) a Java pop-up menu from which the trainee can select a proper intervention and sets up its parameters, (b) a VRML/X3D browser window displaying a 3D virtual environment (Fig. 1). The room is fully equipped and contains a surgical table, where a virtual newborn lies, radiant heater, a virtual monitor displaying the newborn's heartbeat and respiration rates and a cart with several medical instruments, such as endoscope, oxygen mask, laryngoscope, scalpel, tubes, etc. An example of VIDERO supported training was already described in Reference [16].

The Newborn and Its Vital Signs

Following the specifications of H-anim [17], a human modelling standard, the newborn's body consists of separate segments (e.g. forearms, hands, feet, etc.) which are connected to each other by joints (such as the elbow, wrist and ankle, etc.). Each part of the body is controlled by one or more corresponding vital sign mechanisms. In the current prototype version, six of the most crucial newborn's vital signs (Table 1) are simulated.

All vital signs are implemented at three layers: representation, regulation and control layer. The VRML nodes [13], which capture geometrical interpretation and simple response logic, form the first (representation) layer. The core of this layer consists of one or more Timer nodes (subsection Enhancement of Time-Driven Event Generation Mechanism) which drive different Interpolator nodes [13] and, thus, animate the geometrical parameters of the vital signs (i.e. skin color, orientation of the arms, etc.).

The second (regulation) layer is generally implemented in a single Script node [14] per vital sign and controls all the Timer nodes in the corresponding representation layer. Dictating the Timers' active and non-active period it, interpolates the length of different stages in the vital signs' animation cycles (e.g. the length of the heartbeat interpulse interval). Its function is slightly different in the case of skin color and facial movements, where the Script node dynamically creates a suitable interpolator (e.g. the interpolator of skin color) starting with the current parameter state (e.g. current skin color) and

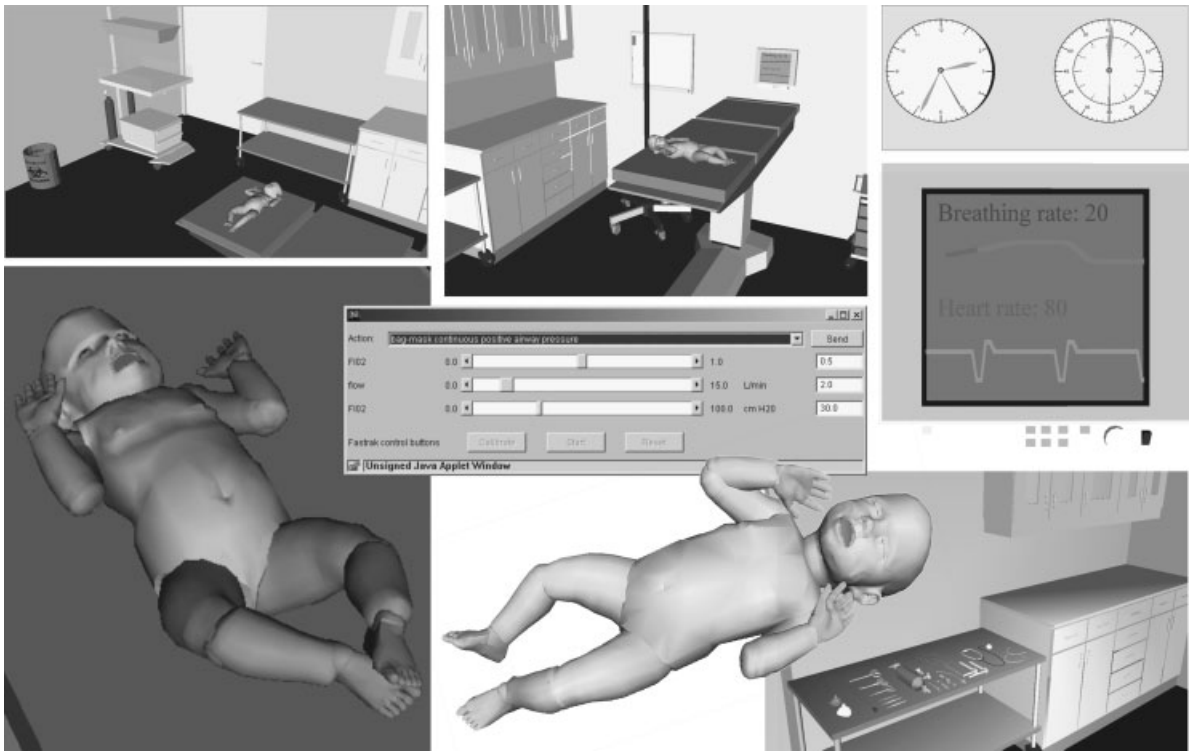

Figure 1 The VIDERO application user interface: the student's control panel (in the middle) and different views of VE with a virtual newborn, medical instrumentation, stopwatch, real-time clock and ECG /respiratory cycle monitor.

finishing with a desired state (e.g. severe peripheral cyanosis). The interpolator is automatically inserted between the corresponding Timer and geometry node (e.g. skin). As a result, the color of the skin changes arbitrarily, starting and finishing with a desired color. The newborn's mouth and eyes open and close by optional angle, likewise the skin color.

The third (control) layer is, in contrast to the previous two, no longer embedded in the VRML/X3D browser but runs independently as a Java application (Fig. 3). We can imagine it as a mediator among the training scenarios (subsection Training Scenarios Creation and Enhancement) on the one hand, and the VRML world on the other. Using the discrete-event simulation technique, a Java class called Scenario Scheduler parses loaded scenario files and fills in the list of events. The event list is observed by a dedicated dispatcher SchedulerThread, built using the Java multithreading programming technique. According to their time stamps, the events pop up and are forwarded through EAI interface [15] to the second (regulation) layer (Fig. 2).

The implementation of the newborn's respiration mechanism is described in more detail, in order to exemplify the coordination and effectiveness of the three-tier concept. Basically, the respiration is modelled as an autonomous vital function, however, various scenario- and user-induced events may

Table 1 Interpretation and Visualisation of Different Newborn's Vital Sings

Vital sign	Interpretation, visualisation
Breathing	Chest movement, different audio clips, breathing curve and current rate displayed on virtual monitor
Heartbeat	Different audio clips, ECG curve and current heart rate displayed on virtual monitor
Color of skin and lips	Skin and lips turn from normal to blue and back; for example, central and peripheral cyanosis can be simulated.
Motion	Movement of the baby's head, arms and legs with various intensity
Crying and sobbing	Replaying of the real baby crying and sobbing audio clips
Facial movements	Mouth and eyes open and close to various degree and forehead wrinkles (the newborn's face varies from calm to grimaced)

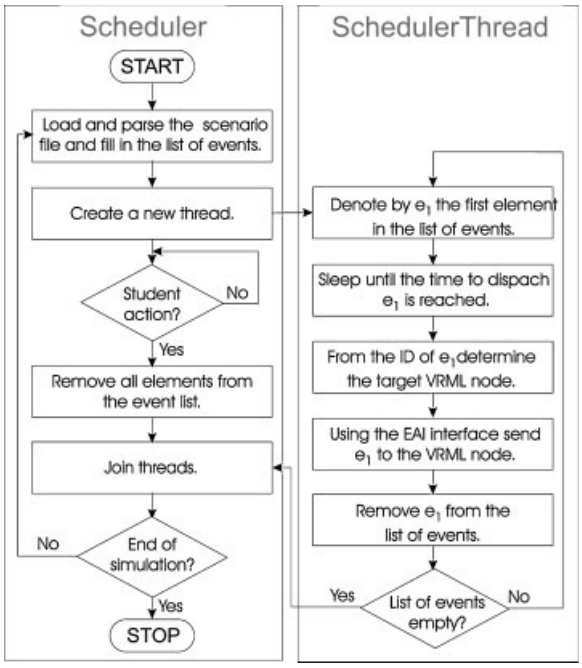

Figure 2 Flow chart of the routines that control event dispatching in the control layer. The events dispatched by SchedulerThread carry the target value of the parameter undergoing a change, and the duration (in seconds) of that change.

considerably change its predefined activity. Its implementation layers consist of the following entities (Fig. 3):

Control Layer. The SchedulerThread thread (Fig. 2) dispatches the events in regard to their triggering time. The events carrying the changes in respiration rate are, by means of the EAI interface, forwarded to the VRML/X3D browser, in particular to the Breath Looper script node.

Regulation Layer. The BreathLooper node receives commands from the scenario and interpolates the breathing rate in time. Calculation of the new breathing rate depends on its current value and on the event value received from the control layer. An arbitrary interpolation strategy can be applied. The current prototype version uses linear interpolation of the respiration cycle (inspiration, pause after inspiration, expiration, and pause after expiration). In each respiratory cycle, new lengths for all respiration stages are calculated and forwarded to the corresponding Timer nodes in the representation layer (duration of inspiration to BreathIn node, length of pause after inspiration to BreathInPause node, etc.).

Representation Layer. The Breath(In/Out) and Breath(In/Out)Pause nodes dictate the changes in the geometrical nodes representing corresponding respiration stages. The Breath(In/Out)ChestInterp and Breath(In/Out)CurveInterp nodes move the newborn's chest and respiration curve on the virtual monitor, respectively, while the Breath(In/Out)Audio nodes replay audio clips of the real newborn breathing. This way the length of audio clips, the movement of the newborn's chest and the curve on the monitor are synchronized with the inspiration and expiration, respectively.

Enhancement of Time-Driven Event Generation Mechanism

Time-driven VRML animations are typically built by using one or more TimeSensor nodes [13], which generate continuous and discrete time events. To achieve the best performance, most VRML/X3D browsers generate time-related events as often as possible [14], and, by saturating the computer's processor, decrease the responsiveness of the virtual world. Another weakness of TimeSensor node is that it does not support priority settings. All TimeSensor nodes in the scene generate the events with the same rate and there is no mechanism to slow some of them down. Most applications, however, consist of a few detailed animations and a number of less important time-dependant actions.

In order to overcome the aforementioned problems, a prototype of a time-dependant node, called Timer [18], was developed. It exhibits exactly the same programming interface and behavior as the standard TimeSensor node, but differs in an additional field, called delay, which sets the minimum time interval between two consecutive continuous time events. Setting the frequency of each individual instance of the Timer node reduces the CPU load and ranks the Timer instances by their importance. Comparison between the Timer and standard TimeSensor node is further illustrated in Figure 6.

Training Scenarios Creation and Enhancement

VIDERO training scenarios are defined in plain text files. Each line of such files describes one event, changing the state of the newborn's vital signs. Most of currently supported events are specified in the following format:

Triggering time [s]	Parameter ID	New parameter value (e.g. 180) or the percentage of change (e.g. 130%)	Time to reach the new value [s]

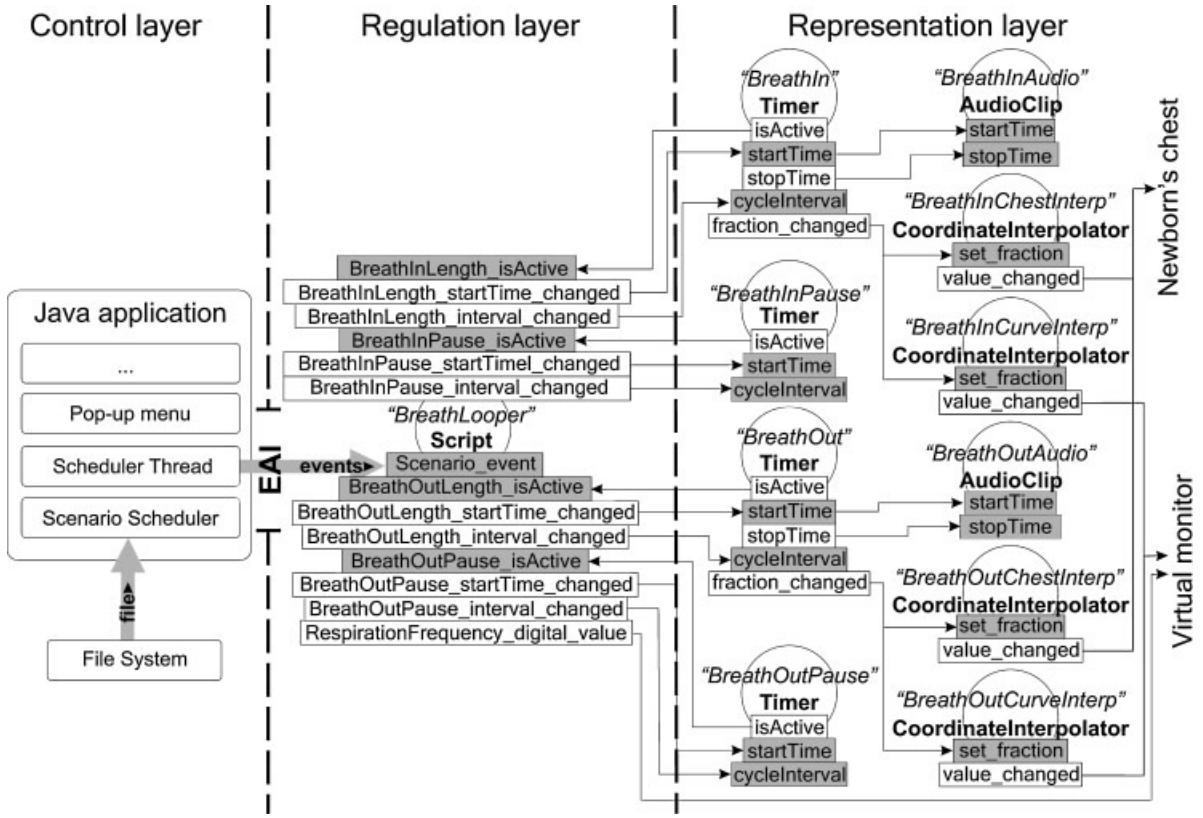

Figure 3 Scheme of the newborn's respiration mechanism, implemented in VRML and Java, follows the principle of three layers: VRML nodes are depicted by circles with their names (italic) and types (bold); their EventOut fields are white and the EventIn fields bright gray, while arrows indicate routes among the nodes.

where parameter ID defines system event (Table 2). The change can be absolute or relative, depending on the final parameter value. As in the real world, changes cannot be executed instantly, but require some transition time. This time is specified by the fourth event parameter. For example, the line

```
25  RR  180  20
```

describes the change of the respiration rate from its current setting to 180 respirations per minute. The change starts in 25th s of the simulation and completes in 20 s, with all intermediate values calculated by linear interpolation. In the case of the CRY event, the last parameter (time to reach the final value) is skipped while the parameter value specifies the name of the file containing the audio clip. For instance

```
120  AUD  intense_crying1.wav
```

starts to play the audio clip saved in the specified file. The sound file name ''null'' is reserved for silence and can be used to stop playing audio clips. A more thorough scenario example is outlined in Figure 7.

Table 2 Events Changing the Newborn's Vital Signs and Their Valid Range

Event ID	Parameter name	Valid range
HR	Heart rate (beats per minute)	0–200
RR	Respiration rate (respirations per minute)	0–80
MOV	Degree of movement (activity)	0 (still) to 1 (high activity)
COL	Skin color	0 (blue) to 1 (pink)
EYE	Opening and closing of the eyes	0 (open eyes) to 1 (closed eyes)
MTH	Opening and closing of the mouth	0 (closed mouth) to 1 (open mouth)
AUD	The playback of one of the pre-recorded audio sound files	Name of the sound file

Specification of Possible Interventions and Their Parameters

An open and flexible structure of the scenarios enables a medical expert to play freely with the newborn's parameters and to create all kinds of critical situations. On the other hand, it is crucial that the user actions, although predefined, are scalable and flexible. In VIDERO, the set of available interventions and their outcomes is simply defined by a special text file. Each intervention is described by an ID, a text description, a set of parameters and the name of the associated scenario file (Table 3).

The intervention parameters are further described in three lines each (Table 4). The parameter's possible values comprise the minimum, maximum and default values. The number of values the trainee can select (including the minimum and maximum values) is also given. All interventions from the intervention list are automatically integrated into the user interface at the application start-up. A full descriptive example of an intervention is given in Figure 4.

Scenario Conditional Branching

When the trainee's action is applied, the so-far executed scenario is interrupted and a new scenario file containing a proper description of the newborn's response is loaded (Fig. 2). As there are many different interventions, there are also many possible transitions among the different scenarios. Usually each postnatal complication is presented by several scenario files containing descriptions of all possible responses to the trainee's interventions. The exact course of the scenario transitions depends on the value of the intervention's parameters, and on the current health state of the virtual newborn. In VIDERO the newborn's condition is precisely determined by the executed scenario. Hence, conditional transition to a new scenario can simply be determined by considering the time and parameters of trainee's intervention. Following the concept of text-based definitions, the scenario's conditional branching can be defined in a simple text-based format presented in Table 5. Each scenario file must have associated descriptions of all possible transitions to the other scenarios. The latter are added to the end of the corresponding scenario file, after the definition of the last scenario event.

Table 3 Format of Interventions in the List of Possible Interventions

Line number	Content
1	Intervention ID
2	Intervention description
3	Number of parameters (NoP)
4: $4+3\cdot$NoP-1	Intervention parameters

All interventions follow the same format and are sequentially defined in the same text file.

Table 4 Format of Intervention Parameters in the Corresponding Intervention Description

Line	Content
1	Parameter name
2	Units
3	Possible values (min, max, default, steps)

The number of conditional transitions per intervention (NoCond) is optional. However, in order to force the consistency of scenario branching, the values of all parameters must be defined for each conditional transition. The corresponding scenario file is loaded only if all the intervention parameters fall into predefined intervals. Correct definition of scenario transitions is exemplified in Figure 5, where, depending on the intervention time and parameter values, three different scenarios saved in files cpr1.scn, cpr2.scn and cpr3.scn can be activated. For example, the scenario cpr1.scn is loaded by selecting the CPR in the first 50 s and setting the values of the rate and depth parameters (Fig. 4) at between 40 and 100 comp/min, and 0 and 1 cm, respectively. The cpr_default.scn scenario file is loaded whenever an undefined combination of parameter values is chosen (e.g. the rate of 50 comp/min and the depth of 3 cm). Replacing the name of the scenario file with a reserved word ''null'' indicates no new scenario should be loaded.

PERFORMANCE EVALUATION

In order to evaluate the performance of the vital sign mechanisms the VIDERO prototype was tested on a Windows XP computer platform consisting of 2 GHz Pentium IV CPU, 512 MB RAM and a Radeon 9000 (64 MB RAM) graphics card. All the performance indices were monitored for 20 min and sampled with a frequency of 1 Hz while 10 test runs were recorded for each performance index.

Comparison of the TimeSensor and Timer nodes

We first compare the performance of the Timer node (see subsection Enhancement of Time-Driven Event Generation Mechanism) to the standard TimeSensor node [13]. A simple VRML scene consisting of a 3D box, PositionInterpolator [13] and Timer node was

```
# The chest compression intervention
CPR                          # action ID
Chest compression            # description
2                            # number of parameters
Rate                         # 1st parameter: name
Comp/min                     # 1st parameter: units
0 180 100 19                 # 1st parameter: values
Depth                        # 2nd parameter: name
cm                           # 2nd parameter: units
0 4 2 9                      # 2nd parameter: values
```

Figure 4 Specification of the trainee's intervention (chest compression) with descriptions of two parameters; the first parameter (rate) represents the number of compressions per minute (comp/min). The trainee can select among 19 possible values that are equally positioned between 0 and 180 (0, 10, 20, . . . , 180). The parameter default value is 100. The second parameter (compression depth) can take values from 0 to 4 centimetres in steps of 0.5 cm (9 possible choices in total). Its default value is 2 cm.

constructed. Events from Timer node were routed via PositionInterpolator to the box's positional coordinates, forcing the box to gradually move from its original position (0,0,0) to new position (2,2,2) and back. The scene was loaded into Cortona [19] VRML browser running as a plug-in of an Internet Explorer. The results are presented in Table 6. The CPU load and the frequency of events generated by the TimeSensor node are also depicted. As expected, the TimeSensor node generates an overflow of events and overloads the CPU.

The test was further extended to a case with 50 and 100 active Timer nodes, respectively. Several instances of the scene from the previous test were loaded into the same VE (assigning different positions to different boxes). Additional interpolator nodes controlling the orientation, color and size of each box were added. Each interpolator was driven by its own Timer node, which resulted in 4 active Timer nodes per box. The CPU load of the VRML/X3D browser and the frequency of the actually generated events are depicted in Figure 6.

Table 5 Format of Scenario Transitions Defined at the End of Each Scenario File

Line	Content
1	Intervention ID
2: NoCond + 1	Intervention time interval (Time = [start time, end time]), parameters' conditional values (parameter name = [lower limit, upper limit]) and the associated scenario file
NoCond + 2	Default scenario file

NoCond denotes the number of conditional scenario transitions.

Finally, 11 instances of Timer node were included in the VIDERO application and used for the animations of different vital signs. Two different values of the delay field were chosen for each instance, limiting the frequencies of the generated events to 33 Hz and 65 Hz, respectively. The results, averaged over 20 different runs, are presented in Table 7. The Timer node, with its event-generation frequency set to 33 Hz, saved a half of the CPU time when compared to the TimeSensor node, but generated a rather insufficient number of events to provide smooth animation. The best performance/cost ratio was noticed by the Timer node with the generation frequency set to 65 Hz. It increased the CPU load by about 10% but provided very smooth animation, visually perfectly comparable to the performance of the TimeSensor node. Comparing the results of the Timer node from Tables 6 and 7 we notice an unexpected increase in the CPU load. It is, of course, due to more complex graphical entities constructing the VIDERO (e.g. virtual baby consists of several thousand polygons, whereas a simple box is a VRML graphical primitive).

Simulation of Vital Signs

In the last experiment, the time consistency of simulated vital signs, that is, the latency between the demanded event triggering times (as specified in the scenario files) and the actual responses in VRML/X3D browser, was investigated. A simple scenario file, changing the values of respiration and heart rate was created (Fig. 7). The initial values of respiration and heart rate were set to 20 inspirations per minute and to 80 beats per minute (bpm), respectively. The

```
# scenario branches triggered by the CPR intervention
CPR  # action ID
Time=[1 50]  Rate=[40 100]  Depth=[0 1]: cpr1.scn #conditional transitions
Time=[1 70]  Rate=[100 120] Depth=[1 3]: cpr2.scn
Time=[50 90] Rate=[120 180] Depth=[3 4]: cpr3.scn
cpr_default.scn # default scenario file
```

Figure 5 Specification of scenario transitions triggered by the chest compression (CPR) trainee's intervention defined in Figure 4.

Table 6 The CPU Load (the Mean ± Standard Deviation) and the Number of Events per Seconds Generated by the Timer Node Depending on the Selected Event Frequency Limit While Moving a Box Object

	Timer (65 Hz)	Timer (100 Hz)	Timer (250 Hz)	Timer (500 Hz)	Timer (750 Hz)	Time-Sensor
Event frequency limit [Hz]	65	100	250	500	750	—
CPU load [%]	0.3 ± 0.1	3.1 ± 2.6	13.9 ± 2.2	27.2 ± 3.6	42.6 ± 4.0	99.2 ± 1,8
Measured event frequency [Hz]	65 ± 0	94 ± 1	229 ± 3	464 ± 6	711 ± 9	857 ± 4

The last column depicts the results of the original TimeSensor node. Note that the event frequency limit cannot be set for the TimeSensor node.

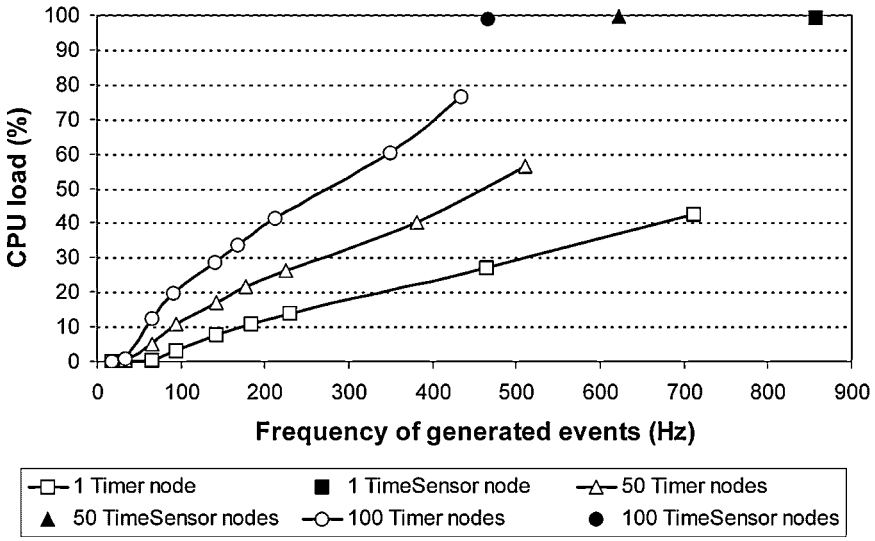

Figure 6 The CPU load depending on actual event frequency and the number of active Timer and TimeSensor nodes used in the experiments with a box object in Cortona VRML browser.

Table 7 The CPU Load (the Mean ± Standard Deviation) and the Number of Events Generated by the Timer and TimeSensor Nodes When Applied to VIDERO Training Environment

	Timer nodes	Timer nodes	TimeSensor nodes
Event frequency limit [Hz]	33	65	—
CPU load [%]	47.28 ± 3.34	58.73 ± 10.21	97.24 ± 0.99
Measured event frequency [Hz]	29.75 ± 2.77	59.5 ± 5.18	144.17 ± 3.6

Time	ID	New value	Transition time
0	RR	80	15
0	HR	200	15
15	RR	60	15
15	HR	140	15
30	RR	20	10
30	HR	60	10
40	RR	10	10
40	HR	50	10
⋮	⋮	⋮	⋮

Figure 7 Scenario changing the values of respiration rate (RR) and heart rate (HR).

simulation implemented our Timer nodes with the event generation frequency set equal to 65 Hz. Several scenario runs were recorded, while sampling the exact values of vital signs in time. Typical results are depicted in Figure 8. The deviations in the simulated values were found within the limits of 2 inspirations per minute and 4 bpm.

CONCLUSION

A programmable VR-based environment introduces many benefits in the medical training and can considerably shorten its development cycle and reduce its expenses. This is especially true when it comes to the training of neonatal decision-making. A computer-generated audio-visual representation of the objects under investigation (virtual baby in the case of neonatology) can provide the missing permanent view of the patient's health condition and forces the trainee to extract important data merely by observing the simulated conditions.

Although currently in a prototype stage, the VIDERO application proved to be considerably competent. Both training scenarios (the newborn's symptoms) and the trainees' interventions are defined in a plain text files providing the course instructors with an option to freely characterize and experiment with the newborn's physiological and behavioral responses. Although a bit cumbersome, the presented text-based format closely resembles the descriptions of resuscitation protocols in standard neonatal manuals and is, hence, relatively close to the medical way of thinking. Once the consensus on the training scenario for a particular critical situation is reached, the aforementioned text-based files can be used to unambiguously define technical specifications of more compact behavior models, such as Parallel Transitions Networks [11] or decision trees [12], for example. The developed prototype has, hence, the potential to be used for general decision-making training, studies of arbitrary postnatal complications, revisions of the virtual patient's behavioral models, and studies of different psychological factors influencing the trainee's proficiency.

Currently, a tool for semi-automatic GUI-based creation of scenario files is being developed. Such a tool would further simplify and considerably shorten the development and revision of complex text-based scenario files.

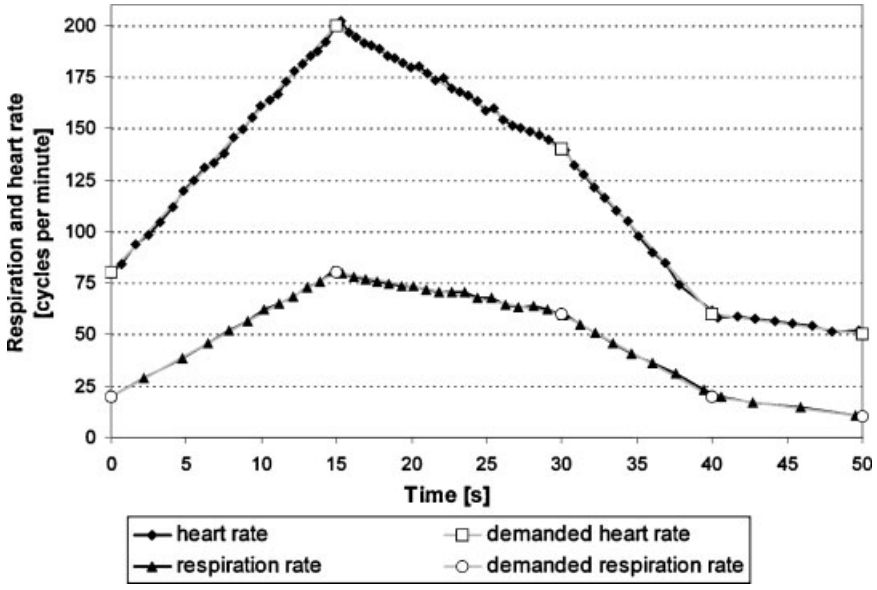

Figure 8 Deviations of simulated values of the respiration rate and heart rate (black) from the demanded values (gray).

ACKNOWLEDGMENTS

The authors are grateful to Professor Zvonko Fazarinc for his precious advice and vision of the future medical training systems, and to Professor Louis P. Halamek from Stanford University for his extensive medical and didactical support. This work was also supported by the Slovenian Ministry of Education, Science, and Sport (contracts nos. 3411-98-71-0001, S2-796-010/21301/2000, and Programme Funding P2-0041).

REFERENCES

[1] S. A. Engum, P. Jeffries, and L. Fisher, Intravenous catheter training system: computer-based education versus traditional learning methods, The American Journal of Surgery 186 (2003), 67−74.

[2] M. Bearman, Is virtual the same as real? Medical students' experiences of a virtual patient, Acad Med 78 (2003), 538−545.

[3] M. B. Bloom, C. L. Rawn, A. D. Salzberg, and T. M. Krummel, Virtual reality applied to procedural testing: the next era, Ann Surg 237 (2003), 442−448.

[4] P. H. Cosman, P. C. Cregan, C. J. Martin, and J. A. Cartmill, Virtual reality simulators: current status in acquisition and assessment of surgical skills, ANZ J Surg 72 (2002), 30−34.

[5] P. Storm, A. Kjellin, L. Hedman, E. Johnson, T. Wredmark, and L. Fellander-Tsai, Validation and learning, in the Procedicus KSA virtual reality surgical simulator, Surg Endosc 17 (2003), 227−231.

[6] J. D. Mabrey, W. D. Cannon, S. D. Gillogly, J. R. Kasser, H. J. Sweeney, B. Zarins, H. Mevis, W. E. Garrett, and R. Poss, Development of a virtual reality arthroscopic knee simulator, Stud Health Technol Inform 70 (2000), 192−194.

[7] M. D. Tsai, M. S. Hsieh, and S. B. Jou, Virtual reality orthopedic surgery simulator, Comp Biol Med 31 (2001), 333−351.

[8] R. Friedl, M. B. Preisack, M. Schefer, W. Klas, J. Tremper, T. Rose, J. Bay, J. Albers, P. Engels, P. Guilliard, C. F. Vahl, and A. Hannekum, CardioOp: an integrated approach to teleteaching in cardiac surgery, Stud Health Technol Inform 70 (2000), 76−82.

[9] D. Bielser, and M. H. Gross, Open surgery simulation, in Proceedings of Medicine Meets Mirtual Reality (Amsterdam, 2002), 57−63.

[10] U. Kühnapfel, H. K. Çakmak, and H. Maass, Endoscopic surgery training using virtual reality and deformable tissue simulation, Comp Graph 24 (2000), 671−682.

[11] D. M. Chi, J. R. Clarke, B. L. Webber, and N. I. Badler, Casualty modeling for real-time medical training, Presence: Teleoperat Virt Env 5 (1996), 359−366.

[12] S. Stansfield, D. Shawver, A. Sobel, M. Parasad, and L. Tapia, Design and implementation of a virtual reality system and its application to training medical first responders, Presence: Teleoperat Virt Env 9 (2000), 524−556.

[13] The VRML Specifications, web site http://www.web3d.org/VRML2.0/FINAL/Spec.

[14] R. Carey and G. Bell, The annotated VRML 2.0 reference manual, Addison−Wesley Developers Press, Berkeley, 1997.

[15] The Virtual Reality Modeling Language (VRML)—Part 2: external authoring interface, web site http://www.vrml.org/WorkingGroups/vrml-eai/Specification/

[16] Z. Fazarinc, S. Divjak, D. Korošec, A. Holobar, M. Divjak, and D. Zazula, Quest for effective use of computer technology in education: from natural sciences to medicine, Comp Appl Eng Educ 11 (2003), 116−132.

[17] Humanoid Animation Working, Group of the WEB3D Consortium, 2003, web site http://h-anim.org/

[18] A. Holobar, and D. Zazula, Improved control of events in the VRML 2.0 application, In Proceedings of EUROMEDIA 2001 (Valencia, Spain, 2001), 67−71.

[19] Parallel Graphics, Cortona VRML Client, web site http://www.parallelgraphics.com/products/cortona

BIOGRAPHIES

Aleš Holobar received his BS and PhD degree in computer science from the University of Maribor, Slovenia, in 2000 and 2004, respectively. From 2000 to 2006 he was a researcher with the Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia. He is currently a Marie Curie fellow at Politecnico di Torino, Italy. His research interests include virtual reality, conceptual learning and signal processing, with current activities focused on blind source separation and biomedical signal processing.

Matjaž Divjak received MSc and PhD degrees in computer science from the University of Maribor, Slovenia, in 2003 and 2005, respectively. He worked as a teaching assistant with the Faculty of Electrical Engineering and Computer Science in Maribor from 2000 to 2005. Currently he is a postdoc researcher at INRIA in Nancy, France. His main research interests include computer vision, image processing, and interactive VR environments.

Dean Korošec received his PhD from Ecole Centrale de Nantes, France, and the University of Maribor, Slovenia, in 1999. After spending 10 years on various research and applied projects at University of Maribor, he became project manager at Nova KBM in 2003. He is currently head of informatics at Nova KBM, second largest Slovenian bank, and director of M-Pay, joint company of Nova KBM and Mobitel. His main research interests include signal processing applications in medicine, simulated medical training and mobile payment systems.

Damjan Zazula received Dipl Ing, master's, and doctorate degrees in electrical engineering from the University of Ljubljana, Slovenia, in 1974, 1978 and 1990, respectively. After being involved in industrial R&D for 12 years, he joined the Faculty of Electrical Engineering and Computer Science, University of Maribor, Slovenia, in 1987. Currently he holds a full professor position in computer science, while from 1998 to 2003 he was also appointed an associate dean of research. Dr. Zazula has spent several months as a visiting professor at the ETH in Zurich, Switzerland and Ecole Centrale de Nantes, France. His main research interests are compound signal decomposition, biomedical imaging and virtual training tools. He is a member of IEEE Signal Processing Society, EURASIP, IAPR, Slovenian Technical Society, Slovenian Society of Pattern Recognition and Slovenian Society of Biomedical Engineering.