An Event-Driven Framework for the Simulation of Complex Surgical Procedures

Christopher Sewell[1], Dan Morris[1], Nikolas Blevins[2], Federico Barbagli[1], and Kenneth Salisbury[1]

[1] Department of Computer Science, Stanford University
{csewell, dmorris, barbagli, jks}@cs.stanford.edu
[2]Department of Otolaryngology, Stanford University
nblevins@stanford.edu

Gates Building 1A
Stanford, California, USA 94305

Abstract. Existing surgical simulators provide a physical simulation that can help a trainee develop the hand-eye coordination and motor skills necessary for specific tasks, such as cutting or suturing. However, it is equally important for a surgeon to gain experience in the cognitive processes involved in performing an entire procedure. The surgeon must be able to perform the correct tasks in the correct sequence, and must be able to quickly and appropriately respond to any unexpected events or mistakes. It would be beneficial for a surgical procedure simulation to expose the training surgeon to difficult situations only rarely encountered in actual patients. We present here a framework for a full-procedure surgical simulator that incorporates an ability to detect discrete events, and that uses these events to track the logical flow of the procedure as performed by the trainee. In addition, we are developing a scripting language that allows an experienced surgeon to precisely specify the logical flow of a procedure without the need for programming. The utility of the framework is illustrated through its application to a mastoidectomy.

1 Introduction

The traditional method of training surgeons has followed the apprenticeship "see one, do one, teach one" model [1]. However, in recent years simulation-based training has become more commonplace as an adjunct to this method, and its value more widely accepted [2]. It can be a safe, cost-effective, customizable, and easily accessible tool for gaining experience in surgery.

One type of surgical training focuses on developing hand-eye coordination and motor skills necessary for specific tasks, such as grasping and suturing, which are frequently performed in a number of procedures. For minimally invasive laparoscopic operations, such skills have traditionally been developed using box trainers. However,

C. Barillot, D.R. Haynor, and P. Hellier (Eds.): MICCAI 2004, LNCS 3217, pp. 346–354, 2004.

recently computer-based simulations, such as LapSim [3], have been validated and shown to also be effective in developing these skills [4] [5]. Simulators have also been developed to train other specific, isolated tasks, such as catheter insertion [6]. The behavior of such existing surgical simulators tends to be fixed by the program's designers and is not adaptable by the medical schools that purchase them.

In addition to developing specific technical skills, however, a surgeon must also learn the optimal sequence in which to perform these tasks, accounting for differences due to variations in patient anatomy or to consequences of previous actions, and how to respond to unexpected or stressful situations. Recognizing this need, mannequin-based simulations of complex scenarios in specialties such as anesthesiology [7] have been developed. Nevertheless, such simulations are expensive and are limited in use due to the high demands of time and effort needed to prepare and run the simulations. Thus, there is a need for cost-efficient, easily accessible computer-based simulations to provide cognitive training for complex scenarios.

Markov chains have been used to identify specific physical actions in surgical simulations (such as fundoplication), and to objectively rate the skill level with which they were performed [8]. Kumar et. al. used a state machine called a “task graph” to sequence appropriate robotic augmentation computational primitives, such as compliant motion, for surgical fine manipulation tasks based on the history of user control and environmental interactions [9]. In addition, finite state machines have been used to control simulations of digital actors in video games and animated movies, defining actions undertaken by the actors depending upon previous events in the simulation.

We believe that using such models to track the logical flow of events in a surgical simulator will help develop the trainee's cognitive skills in dealing with complex scenarios. Ideally, the program should allow the user to interact with the simulated patient in a free-form manner, while still keeping track of the order in which actions are performed to ensure that safe and effective techniques are taught. In addition, the ability to detect and track discrete events can allow for an increased level of realism without the need for an exceedingly complex real-time physical simulator. For example, by detecting an “incision event” that triggers a “bleed action”, the simulation can provide a realistic response to cutting a vein without the physical simulator having to account for the complete dynamics of blood flow through veins. Events could even trigger the loading of an appropriate pre-recorded response from a database, greatly reducing the demands on the real-time simulator.

Furthermore, our system provides the user with control over the design of the simulated procedure. Recognizing the need for customizable surgical simulators, the Teleos authoring tool [10] was designed to make it easier to create new surgical simulation environments. Several frameworks, such as CAML [11] and GiPSi [12], have been proposed to facilitate seamless integration of independently developed physical simulation modules. This idea can be extended to provide scripting tools to allow a professor of surgery using the simulator to easily modify or create scenarios for his/her students, as well as to tailor the performance metrics and logical flow to his/her style and preferences.

2 The Conceptual Framework

Our system can be divided into four domains: the user, the physical simulator, the event engine, and the instructor and performance evaluator. It is a hybrid system with both continuous and discrete components. The framework is illustrated in Figure 1A.

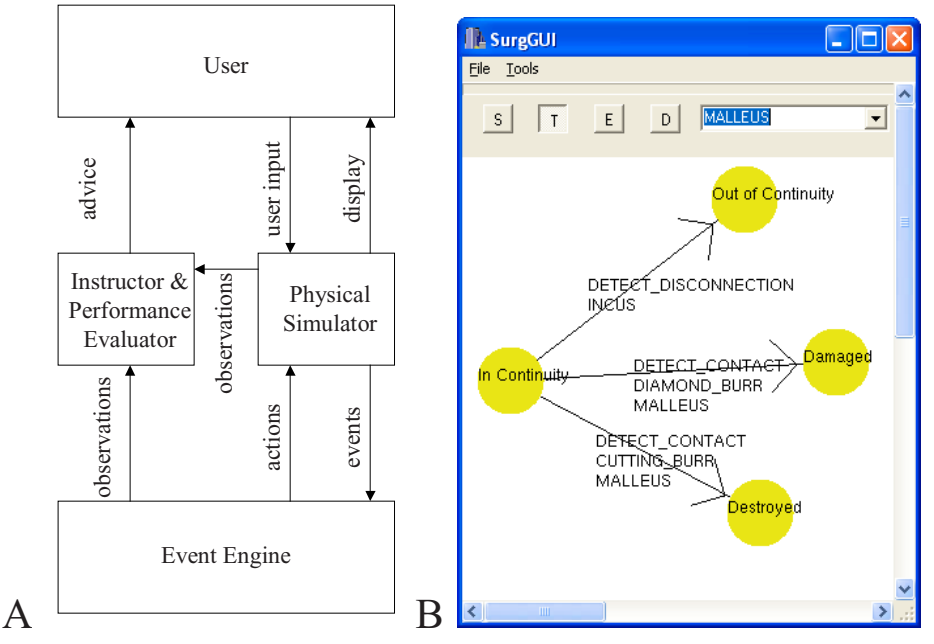


Fig. 1. A) Diagram of the flow of information among the four domains of our simulation conceptual framework, as described in Section 2. B) An example of a simple state machine developed using our graphical scripting environment. Here, the user has specified differing levels of trauma caused by contacting the malleus with two different types of drills. However, if the incus is removed, the hearing chain becomes disconnected, and the malleus enters a state in which there are no such out-transitions, since drill contacts no longer result in hearing loss.

2.1 User and Physical Simulator

The user is the surgeon in training. He/she provides input to the physical simulator, such as movements of a pointing or haptic device. In return, the user may receive graphic, haptic, and auditory feedback from the simulation. The physical simulator is responsible for the rendering of the physical models, which represent the anatomy of the simulated patient, and of the surgical instruments. The rendering should be responsive to continuous interactions between the instruments and the anatomical structures, and among the various structures. Depending on the degree of realism

required, a physical simulator of virtually any level of complexity may be used. The only requirement is that it implement the interfaces defined by the other domains.

2.2 Event Engine

In addition to the continuous interactions between the instrument and the anatomical structures, and among the different anatomical structures, there are discrete events that occur during the course of a procedure. These events may trigger specific actions. According to our terminology, "events" are inputs to the engine resulting from interactions with the continuous physical simulation, while "actions" are outputs of the engine that specify a change to the physical simulation. The behavior of the physical simulator can be constrained by the history of past events. For example, if an artery leading into an organ has been cauterized, a subsequent "incision event"' affecting that organ may no longer produce a "bleed action". The relevant aspects of that history, defined by the space of possible current and future events, can be modeled as a state in a finite automaton. Transitions between states (or self-transitions) are triggered by the detection of pre-defined events, and either a transition itself or entry into a state can result in the execution of a specified action.

Once triggered, some action functions may just execute once. For example, positioning an instrument closer than a certain distance from a sensitive structure may result in a warning message being displayed to the trainee. Other action functions place a message in a list that is periodically read by the physical simulator, resulting in a persistent change in the rendering of the physical models. A third type of action function removes messages from the list. For example, detecting an "incision event" at a certain location will result in a message being inserted into the list stating that bleeding should occur at that location. As the physical simulator renders the scene, it reads the list and simulates the bleeding at the given location. Later, detecting a "suture event" at that location will result in the removal of that message, and the physical simulator will no longer render the bleeding.

The event engine receives specifications of events to detect and actions to execute from the input script (described in the following section), and it communicates with the physical simulator through the message list and a standardized API. For action functions that affect the physical simulation, only a few relatively high-level parameters (such as the location and rate of bleeding) are specified in the script and passed on to the physical simulator via the message list by the event engine. The physical simulator is free to implement the requested action in any way. As long as it can read and recognize the messages in the list, any physical simulator, of any degree of sophistication, can be plugged into the system in a modular fashion. Likewise, the event engine passes on only a few high-level parameters (such as the location, length, depth, and force of an "incision event") for events to be detected from the script to the physical simulator. The physical simulator must only implement a specified API, consisting of a number of specific event detection functions that accept these parameters and return true or false.

2.3 Performance Evaluator

A surgical training system should also provide feedback to the user about his/her performance, scoring it and supplying constructive criticisms. Many events may be directly tied to a performance metric, such as contact of a cutting instrument with a nerve resulting in a comment that care needs to be taken around nerves and a reduction in score proportional to the amount of nerve damage. Some such events may only be of evaluation significance and not have a direct effect on the physical simulation, such as using a functional but non-optimal instrument for a certain task. Other metrics may not be tied to events, but rather to an observation of the state of the physical simulation. For example, if the indication for surgery was removal of a tumor, it is obviously relevant how much of the tumor structure remains at the end. Yet other metrics may consider the observed state of the simulation when a specific event occurs, such as the degree of exposure of a structure when it is manipulated or removed.

3 The Scripting Language

A scripting language is being developed to allow surgeons to use the system described in the previous section to design specific procedures and scenarios for training.

The first section of a script in our language simply lists the file names and types (deformable layer, volumetric mesh, or surface mesh) of the models representing the instruments and the anatomical structures, and associates with each a name that can be referenced from elsewhere in the script.

The second section declares variables for metrics to be tracked, which may be given an initial value and ranges for acceptable values. These variables are modified by certain action functions, as described below.

Finite automata are described in the third section. An automaton may be associated with any or all of the structures named in the first section. Associated with each state may be a list of action functions, defined by the event engine's API, that are executed upon entry into the state. These functions may include several parameters, such as a structure name from the first section, a location in Cartesian or spherical coordinates, and/or an expression indicative of something such as rate or severity of the action.

Such expressions may include numerical constants, arithmetic operators, variables declared in the second section of the script, and reserved words (such as FORCE or LOCATION for the haptic force or instrument position at the time of execution of the action function). For example, the BLEED function may be called with three parameters, e.g. “BLEED SKIN LOCATION '5*FORCE'”, specifying that the skin will bleed at the current instrument contact position at a rate directly proportional to the force of the contact. More sophisticated parameters of the resulting blood simulation are handled at a lower level, abstracted from the surgeon, by the physical simulator. The “UPDATE_STATUS” function takes as a parameter an expression that modifies a performance metric variable defined in the second section.

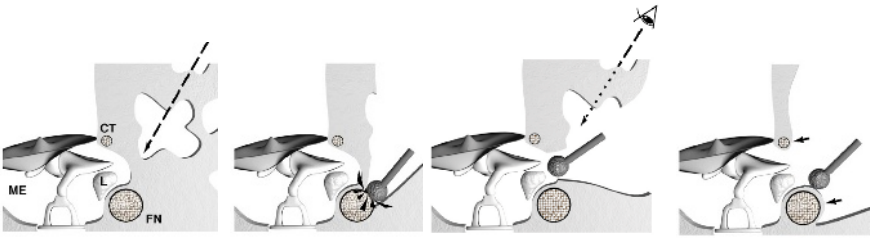


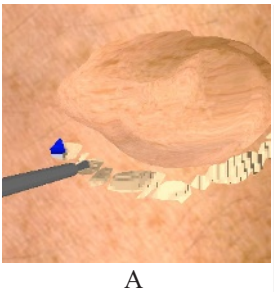
Fig. 2. A) Schematic cross-sectional view of the temporal bone illustrating a lesion (L) within the middle ear (ME) cavity. The goal of this simulation is to access this lesion, in the direction shown by the arrow. The chorda tympani (CT) and the facial nerve (FN) are shown. B) An "injurious" action has occurred, with the burr contacting the facial nerve. C) A "dangerous" action has occurred, when the surgeon has drilled away bone without first establishing clear exposure of the region. D) Correct bone removal has occurred. The nerves have been avoided, and adequate exposure has been established. Failure to adequately thin the bone may be considered a "discouraged" action, since there remains some doubt as to the location of the nerves.

Possible transitions among these states are listed, each associated with and triggered by an event detection function, again defined by the event engine's API. These functions may also take parameters similar to those taken by the action functions. For example, "2 5 DETECT_CONTACT SCALPEL MUSCLE 0.5" indicates a transition from State Two to State Five when the scalpel contacts the muscle layer with a force of at least 0.5 Newtons. Many of these transitions may be self-transitions. For greater specificity, action functions may be associated with a transition rather than with a state. Thus, an action may be taken only when there is a transition into a state from a certain subset of states rather than upon entry into the state from any other state.

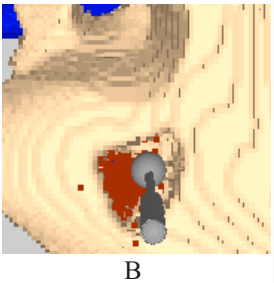
A graphical development tool allows the supervising surgeon to "draw" the automata. Variables, reserved words, and action and event detection functions defined by the event engine API can be selected from lists, and templates are provided for parameters. Performance metric variables and models may be defined via dialog boxes. The designed script is then written to a file readable by the simulation program. Figure 1B shows a screen shot of this tool.

4 Application: Mastoidectomy

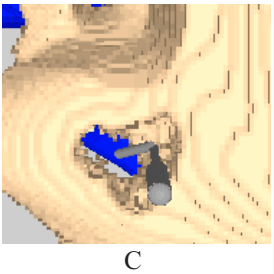
Our surgical simulation framework and scripting language have been applied to the development of a mastoidectomy training system, similar to [13]. A mastoidectomy is a procedure performed in which a portion of the temporal bone is drilled away in order to provide access to inner regions of the ear. In our simulation, the pathology is a legion located in the middle ear. The logical flow of the operation, and the various potential morbidities, are based on The Temporal Bone Dissector [14]. Our simulator's event engine uses a script of the procedure developed with the aid of an otologist. Examples of "injurious", "dangerous", and "discouraged" user actions relevant to the procedure are illustrated in Figure 2.



A



B



C

```
/* Anatomical model definitions */
DEFORMABLE_LAYER SKIN 7 7 1 7 0 0 ..\skin.bmp 1
..\Mear.3ds EAR MESH ..\skin.bmp 1
..\Mtbone.txt TEMPORAL_BONE VOXELS no_texture 2
..\Msigmoid.3ds SIGMOID MESH no_texture 2
. . .

/* Tool definitions */
..\Msuction.3ds SUCTION
..\Mcutburr.3ds CUTBURR
..\Mdiaburr.3ds DIABURR
..\Mscalpel.3ds SCALPEL

/* Performance metric variable definitions */
BLOOD 100.0 90.0 95.0
CSF 100.0 90.0 95.0

/* State machine for sigmoid */
SIGMOID
State 1
Initial State

1 1 DETECT_CONTACT CUTBURR SIGMOID
INSTRUCTION There is a severe hemorrhage.
UPDATE_STATUS BLOOD BLOOD-5.0
BLEED SIGMOID 2.0

1 1 DETECT_CONTACT DIABURR SIGMOID
INSTRUCTION There is a hemorrhage.
UPDATE_STATUS BLOOD BLOOD-1.0
BLEED SIGMOID 1.0

1 1 DETECT_CONTACT SUCTION SIGMOID
INSTRUCTION The hemorrhage is controlled.
CLOSE_UP SIGMOID
. . .
```

Fig. 3. Screen shots from the mastoidectomy simulation, along with a portion of the script. A) Detection of an incision event behind the ear triggers an action that changes to the microscope view of the exposed bone. B) Detection of a contact event between the diamond burr and the sigmoid sinus triggers an action that initiates bleeding. C) Detection of a contact event between the suction and the bleeding location triggers an action that controls the hemorrhage.

Many of the events in this simplified procedure relate to contact between a drill and a sensitive structure. Drilling into the sigmoid sinus causes a venous hemorrhage; drilling into the facial nerve results in nerve damage; and drilling into the tegmen mastoidae causes cerebrospinal fluid (CSF) leakage. The severity of such damage is often dependent on whether a cutting burr or diamond burr drill is used. Use of the suction at the location of these problems can result in their removal from the active event list. Some events affect the patient outcome and thus the performance metrics, though there is no perceptible consequence in the physical simulation. For example, drilling into structures of the inner ear can result in vertigo and/or hearing loss. Other events may only be a matter of "style" leading to a message on the final report, such as using the slower diamond burr in "safe" areas that could have been drilled more efficiently with the cutting burr. Examples of several events are shown in Figure 3.

5 Conclusion and Future Work

Both the event-driven conceptual framework and the mastoidectomy simulator remain in development. A progressively more detailed description of the procedure is being generated with the scripting language. A collaborative environment for surgical simulation is also currently under development. After further improvements, clinical validation studies may be performed to determine the utility of the system. Eventually, the simulator may use physical models constructed from patient-specific data, such as CT scans, to provide an opportunity for pre-operative, case-specific rehearsal for even experienced surgeons. We appreciate the help of Tom Krummel, Jean-Claude Latombe, Elena Vileshina, and Phil Fong, as well as NIH Grant R33 LM07295, Stanford BioX Grant 2DMA178, and the NDSEG, NSF and Stanford fellowship programs.

References

1. Gorman PJ, Meier AH, Rawn C, Krummel TM: The future of medical education is no longer blood and guts, it is bits and bytes. *American J Surgery.* 2000 Nov. 180(5):353-356.
2. Haluck RS, Marshall RL, Krummel TM, Melkonian MG: Are surgery training programs ready for virtual reality? *J of the American College of Surgery*. 2001 Dec. 193(6):660-665.
3. Larsson A: An open and flexible framework for computer aided surgical training. *Stud Health Technol Inform.* 2001 81:263-265.
4. Hyltander A, Liljegren E, Rhodin PH, Lonroth H: The transfer of basic skills learned in a laparoscopic simulator to the operating room. *Surg Endosc.* 2002 Sep. 16(9):1324-1328.
5. Seymour NE, Gallagher AG, Roman SA, O'Brien MK, Bansal VK, Andersen DK, Satava RM: Virtual reality training improves operating room performance: Results of a randomized, double-blinded study. *Annals of Surgery.* 2002 Oct. 236(4):458-464.
6. Prystowsky JB, Regehr G, Rogers DA: A virtual reality module for intravenous catheter placement. *American J Surgery.* 1999 177:171–175.
7. Holzman RS, Cooper JB, Gaba DM: Anesthesia crisis resource management: real-life simulation training in operating room crises. *J of Clinical Anesthesiology* 1995 7:675–687.
8. Rosen J, HB, Richards CG, Sinanan MN: Markov modeling of minimally invasive surgery based on tool/tissue interaction and force/torque signatures for evaluating surgical skills. *IEEE Transactions on Biomedical Engineering.* 48 (2001).
9. Kumar R, Hager GD, Barnes A, Jensen P, Taylor RH: An augmentation system for fine manipulation. MICCAI 2000:956-965.
10. Meglan DA, Raju R, Merril GL, Merril JR, Nguyen BH, Swamy SN, Higgins GA: The Teleos virtual environment toolkit for simulation-based surgical educ. MMVR 1996:346-51.
11. Cotin S, Shaffer D, Meglan D, Ottensmeyer M, Berry P, Dawson S: CAML: A general framework for the development of medical simulation systems. *Proc of SPIE* 4037:294-300.
12. Cavusoglu MC, Goktekin TG, Tendick F, Sastry SS: GiPSi: An open source/open architecture software development framework for surgical simulation. MMVR 2004:46-48.

13. Agus M, Giachetti A, Gobbetti E, Zanetti G, Zorcolo A: A multiprocessor decoupled system for the simulation of temporal bone surgery. *Computing and Visualization in Science*. 2002 5(1):35-43.
14. Blevins NH: The promise of multimedia in otology. *American J of Otology*. 1997 May 18(3):283-284.