Spring: A General Framework for Collaborative, Real-time Surgical Simulation

Kevin Montgomery, Cynthia Bruyns, Joel Brown, Stephen Sorkin, Frederic Mazzella, Guillaume Thonier, Arnaud Tellier, Benjamin Lerman, Anil Menon
National Biocomputation Center, 701A Welch Rd, Suite 1128 Stanford, CA 94305

Abstract: We describe the implementation details of a real-time surgical simulation system with soft-tissue modeling and multi-user, multi-instrument, networked haptics. The simulator is cross-platform and runs on various Unix and Windows platforms. It is written in C++ with OpenGL for graphics; GLUT, GLUI, and MUI for user interface; and supports parallel processing. It allows for the relatively easy introduction of patient-specific anatomy and supports many common file formats. It performs soft-tissue modeling, some limited rigid-body dynamics, and suture modeling. The simulator interfaces to many different interaction devices and provides for multi-user, multi-instrument collaboration over the Internet. Many virtual tools have been created and their interactions with tissue have been implemented. In addition, a number of extra features, such as voice input/output, real-time texture-mapped video input, stereo and head-mounted display support, and replicated display facilities are presented.

1. Introduction

The benefits of computer-based surgical simulation have been widely discussed and quantitatively demonstrated by many researchers[1]. The benefits include the ability to broaden surgical training by easily providing different training scenarios, including anatomical variations (gender, size), pathologies (diseases, trauma), and operating environment conditions (emergency room, microgravity, battlefield). In addition to these benefits, the ability to objectively quantify surgical performance[2] and perhaps simulate the result of an intervention has been cited as a major benefit and drawn the attention of surgical societies as a future means of precertification. Besides these benefits, the potential to accelerate the acquisition of baseline surgical skills through the use of computer-based simulation has also been identified[3]. Perhaps the most important feature of all is that computer-based simulation can realize all these benefits without risk to any real patients.

Because of these benefits, many research groups in academia, government, and industry have been developing simulators for some time[4-13]. Each group must instill the clinical knowledge of the surgeon through the engineering technology of the computer scientist in order to realize a working and usable system. However, the technical knowledge required to produce such a simulator spans many subdisciplines including graphics, algorithm design, numerical integration methods, collision detection, networking, user interface design, and mechanical engineering. Replicating this wide breadth of technical knowledge is difficult and, for many clinical groups, represents an insurmountable obstacle to the production of a complete, functioning, useful simulator. Moreover, within other groups with broad engineering skills, achieving expertise in each of the areas required and developing their own software for each of these tasks can also increase costs and the time before the realization of a working simulator.

If the surgical simulation community instead had a common framework of shared code, then the time to realization of a working simulator would be shortened, and the barrier to entry for clinical groups of more limited engineering resources would be lessened. In essence, the entire surgical simulation community would work together on a common platform, sharing their individual expertise, and thereby accelerate the production, and ultimately adoption, of computer-based surgical simulators.

However, the challenges of building one simulator that could be used for many applications are great. Such a simulator would run the risk of trying to be *everything to everyone* and perhaps end up providing *nothing to anyone*. Beyond these factors, the technical challenge of producing a real-time, haptic-rate simulator is extraordinarily difficult in itself.

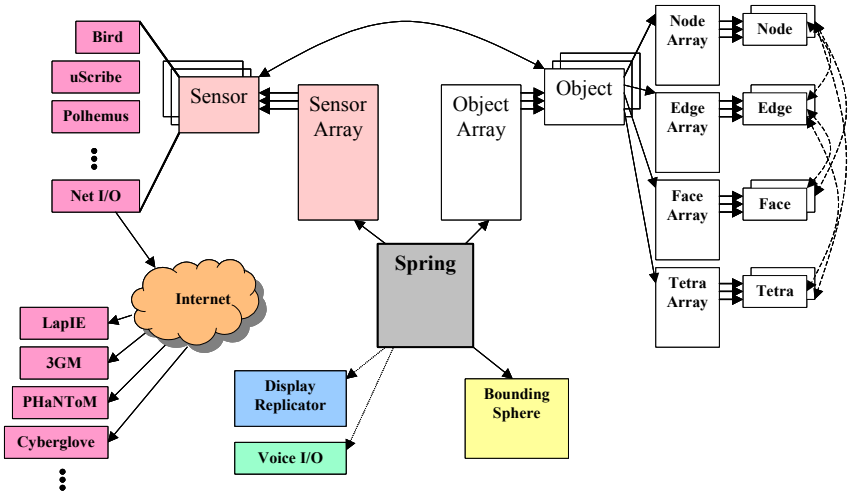
We have developed a surgical simulation framework named *Spring*. This evolving framework was designed to be a general simulator with a broad base of technological features and a broad range of potential applications, with the emphasis on real-time performance. During its production, we have developed a number of applications to ensure that the system is usable and useful for application. It is our goal to release this code to the surgical simulation community in open source form to, at the least, provide a useful example code to compare implementation details with others. At most, we hope that other groups may use it as a common framework in which to insert their expertise and enable the sharing of all our talents with the wider community.

2. Methods

The *Spring* surgical simulator code is cross-platform and runs on Unix (Sun Solaris, SGI Irix, and Linux) and Windows (98, NT, 2000) platforms. It is written in C++ and uses OpenGL for graphics; GLUT[14], GLUI[15], and MUI[16] for user interface; and supports parallel processing using the pthreads facility of POSIX. It allows for the relatively easy introduction of patient-specific anatomy and supports many common file formats, including SMF[17], Wavefront OBJ, VRML, Mesh, and Cyberware formats. It performs soft-tissue modeling[18], some limited rigid-body dynamics, and suture modeling[18]. The simulator interfaces to many different interaction devices and provides for multi-user, multi-instrument collaboration over the Internet in latency-dependent or latency-moderated modes. Many surgical and non-surgical virtual tools have been created and their interactions with tissue have been implemented. Collision detection is provided through an enhanced[19] bounding-sphere algorithm[20]. In addition, extra features such as voice input/output, real-time texture-mapped video input, stereo and head-mounted display support, and replicated display facilities are implemented.

2.1 Architecture

An overview of the architecture is provided in Figure 1. It consists of a main program (Spring), an object representation structure (ObjectArray class, with individual Objects comprising arrays of Nodes (vertices), Edges (springs), Faces (triangles), and Tetra (tetrahedral) elements that are cross-linked), an abstraction of tracking/haptic devices (Sensor class) with the individual interface subclasses including networked devices, a collision detection subsystem (BoundingSphere), and other features (Voice I/O, DisplayReplicator, etc).



2.2 Main Program

The main program (Spring) contains code for interfacing to the operating system (through GLUT, GLUI, MUI, Posix) and contains callback functions for keyboard, mouse, and other input. In addition, it contains the main display and simulation functions and all menu creation routines. Finally, it also has the ability to create objects and interface to the (optional) voice I/O and DisplayReplication subsystems. In essence, it contains the main thread(s) of control, data structures, and interface routines.

2.3 Anatomy Acquisition

Anatomy can be introduced from many sources and is provided by reading in world description files, containing lists of objects and their attributes. These data could be acquired from serial-section, volumetric (CT, MR), or surface scans (Cyberware), segmented using any tools[21], followed by mesh generation and reduction (Qslim[17], or others). Once the geometry is built using these tools, the world description file is produced, which allows one to specify all the individual objects in the world to be created, as well as their attributes. The attributes for each object consist of the geometry information itself, graphical attributes (textures, material properties, etc), simulation properties (dynamics, numerical method, spring constants, etc), collision detection and resolution attributes (detection method, faces to consider, etc), and other attributes. In this way, a single file can be used to set all parameters for a particular simulation and points to other files, in standard, industry-wide formats (where available), for the individual information.

2.4 Object Representation

The world description information is used to create objects with the given properties. Each object contains attributes as listed above and also contains arrays of Nodes, Edges, Faces, and Tetras. Note that, for a particular object, it may not contain all of these elements. Blood could be modeled as a particle system and represented as an object consisting only of Nodes. A suture is modeled as an object with only Nodes and Edges. Most 3D objects can be modeled with Nodes, Edges, and Faces, while Tetras are presently only required for some forms of cutting and volume preservation. An extrusion algorithm is also implemented to provide some 3D structural dynamics from a surface-only mesh.

2.5 Simulation Core

The core simulation code within each object processes the dynamics of each object (deformable, rigid-body, or suture dynamics) at every simulation timestep. Providing rigid-body kinematics within a dynamic solver is an open area of research, therefore limited support is currently provided. Suture dynamics are modeled by considering the suture to be comprised of short, linear segments (edges), forming an articulating object. This articulating object has constraints imposed by its contact with other objects (virtual instruments, tissue, other points of collision) and computes the locations of intermediate nodes as a weighted, bi-directional follow-the-leader algorithm[18].

For deformable objects the simulation system considers the nodes as point masses and edges as spring/dampers to form a 3D mesh for mass-spring simulation. These edges can be considered as linear, piece-wise linear, or non-linear 1D springs/dampers. While we currently do not provide support for torsional springs[22], their introduction would be straightforward. Each edge can have different spring and damping coefficients and the nodes can provide different mass distributions to provide for some support of anisotropic, heterogeneous tissues.

For deformable objects, a number of numerical methods have been implemented. The traditional Euler and Runge-Kutta (2^{nd} and 4^{th} order) have been implemented. In addition, a quasi-static method, appropriate for heavily damped tissues and low interaction velocities, assumes the tissue to always be in static equilibrium and ignores dynamic inertial and damping forces for a corresponding increase in simulation performance. Other simulation methods are also under development. Relative performance numbers for each method is given in Table 1.

Numerical Method	Node updates per second	Edge updates per second
Euler	178,000	530,000
Runge-Kutta 2^{nd} order	97,000	288,000
Runge-Kutta 4^{th} order	46,000	136,000
Quasistatic	220,000	651,000

Table 1: Performance of Numerical Integration Techniques (5000 node, 15000 edge, 10000 face object)

An open area of research in soft tissue deformation concerns taking as large an integration step size as possible[23], while maintaining stability of the numerical method. To address this issue, we have produced an antidivergence feature that, when numerical divergence is detected, continues to halve the step size of the numerical method until stability is acheived. While this violates the time-accurate goal of the simulation, it was judged more important to maintain stability under such degenerate conditions.

Other features include tracking the region of deformation and only processing within this region in order to provide a significant (possibly order of magnitude) increase in performance[18]. This is accomplished by breadth-first ordering the nodes from a point of interaction into levels, processing the nodes in this level-based order, then ceasing computation when a level has no nodes that move more than a threshold. These regions may increase to include other levels as the deformation increases, or decrease in size as the nodes of the object at a particular level return to rest.

This scheme allows for many objects, each with large geometries, to be on the screen with high resolution graphical display, while the simulation is only processing what is needed. In addition, the simulation can use the information of the extent of the deformed region for other purposes, such as to indicate to the collision detection system to update its structures for those nodes.

Volume preservation code is also provided for each object. Based on this attribute, the object can decide to disable volume preservation or perform global volume preservation (compute object volume at each timestep and induce a corresponding force on each node in the direction of its normal), or to preserve volume within the deformed region alone.

Finally, it is also possible to produce links between objects. In this way, for example, an object with deformable dynamics (skin) could be linked to an object with rigid-body kinematics (bone). This is accomplished through the use of *tie-nodes*: the skin has a node which exists in the same position as a node in the bone and the two nodes can thereby pass forces between their respective objects.

2.6 Sensors

A Sensor is an abstraction of a 6D tracking and/or haptic device. It contains a position in 3D space, along with orientation information (rotation matrix) and contains an array of floating-point activation values to store information from any buttons, handles, or other controllers associated with the device. A sensor can be linked to a particular object (typically a virtual instrument) and, when that object is updated, it will be transformed by the sensor data. In addition, the activation values can be used to specify hinge angles (for example, to indicate and display how opened or closed the scissors are) or telescoping/plunger depth (for a syringe or resectoscope handle) for the object's subparts (the subpart id is denoted by a field in each Node).

We functionally classify instruments into a number of different categories, based upon their methods of operation. Instruments are single pieced (e.g., scalpel, dilator, probe), hinged (scissors, endoscopic scissors, graspers), multihinged (3-prong grasper- where we specify each hinge's location, axis of rotation, and activation value upon which it depends),

dependent-hinged (hand, multiaxis endoscopic grasper- where the location of one hinge is dependent upon another), telescoping (resectoscope, syringe- with one part that slides in relation to another), lasso (resection loop- with a part that constricts based upon the activation value) and multitools (with working channels for the insertion of tools of the kinds stated above). Note that these classifications merely define how the instrument articulates based upon its activation values. We will later discuss the interactions of these tools with other objects.

The Sensor superclass is inherited by subclasses that communicate with their individual devices. These include non-haptic devices such as electromagnetic trackers (Ascension Flock-of-Birds and pcBird, Polhemus FasTrak), inertial trackers (Intersense InterTrax), armature-based trackers (Immersion Microscribe), composite devices (Virtual Technologies/Immersion CyberGlove) or a computer mouse. Haptic devices supported include devices from Immersion (3GM, Laparoscopic Impulse Engine, Bimanual Laparoscopic Device) and SensAble Technologies (Phantom).

Because many devices can only be interfaced through methods available only on PC platforms, and due to the desire to have the simulation capable of running on the most appropriate (perhaps non-PC) hardware, the system also provides for a network-based module that communicates with a sensor or *hapticserver* program running on a different, perhaps dedicated computer over the network. In this way, we can decouple the interfacing restrictions of these devices from the simulation itself.

In either case, this method of network-based sensor- and hapticservers inherently provides for multi-user and multi-instrument interaction and supports collaborative procedures. However, when performing a collaboration at some distance, it is necessary to also replicate the display of the simulation, as described in section 2.10 below.

2.7 Collision Detection

Over time, a number of collision detection methods were implemented. First, a node-node force-sphere model was introduced. Later, static partition methods, Axis-Aligned Bounding Boxes[24], and Oriented Bounding Boxes[25] were implemented. While these methods worked well in many cases, a more general scheme that better supported deformable objects was sought. For this reason, we moved to a Bounding Sphere algorithm[20], with enhancements for deformable objects[19]. The generality of this method, and its fast update capability, provided a reasonable tradeoff for many cases. As with any hierarchical method, ultimately the detection method decomposes to testing collisions between primitive elements, such as face-face (surface collisions), edge-face (suture wrapping over vessel), edge-edge (suture wrapping onto itself), etc.

To further increase performance, a number of other enhancements were made. First, each node within the bounding sphere tree can be individually enabled. In this way, large portions of the tree can effectively be ignored when appropriate. In order to easily support this, a world description file can identify the list of faces to be used for collision detection. Moreover, objects (virtual instruments) can be created to have internal faces which are invisible to the user, but upon whom the collision detection system relies. In this way, virtual instruments can be very detailed graphically, but a dramatic decrease in the number of collision detection tests can be realized, leading to large performance increases[26]. Finally, a fast path within the collision detection subsystem was created to provide a quick rejection test for appropriate applications.

When a collision is detected, the collision detection subsystem places, in each colliding object, a collision pair list denoting the details of the collision (which primitive elements collided, the intersection point if available). In this way, the collision detection subsystem merely provides a general service of detection and enables a more general system for collision response.

2.8 Collision Resolution/Interactions

As each object is processed by the simulation and any collisions noted, each object's collision handling routine is called. A probing interaction (pick, dilator, hand) induces an instantaneous displacement of the deformable faces to resolve the collision (relying on the antidivergence algorithm to ensure stability in degenerate cases). Grasping tools (forceps, endoscopic graspers) attract nodes of the surface to the tool tip when that tool is active, with the rest of the object is processed as a probing interaction as above. Piercing (needle, syringe) subdivides the surface at the point of entry of the tool tip, with the rest of the object considered as a probing interaction (hence, the tip of the syringe is "sharp" and pierces into tissue, while the rest of the syringe merely bumps the tissue upon interaction). Cutting interactions[28] (scalpel, scissors, endoscopic scissors) have edges that are denoted as sharp. When one of these edges comes in contact with the tissue, the tissue is cut and the mesh subdivided. If a non-sharp edge or face comes in contact with the tissue, then a probing interaction is produced (the back of a scalpel can be used to probe, while the cutting edge can be used to slice the tissue). A cauterizing interaction (roller ablator, loop cautery), when active and in contact with tissue, progressively yellows, browns, then blackens the area of contact by changing the color of the contacted faces and using blended textures to achieve the desired graphical result.

In each case, the collision resolution algorithm processes each of the elements of the collision pair list, performs their interaction function, and computes the resulting haptic force of that interaction upon the tool. Then, the interaction forces are averaged to calculate the overall force vector that should be realized upon the virtual instrument.

2.9 Display

A number of display devices are supported. Traditional CRT-based displays in monoscopic or stereoscopic (using CrystalEyes or NuVision glasses) modes are supported. In addition, projection displays, such as the Immersive WorkBench (FakeSpace), as well as custom displays such as the Surgical WorkBench[29] are supported. Head-mounted displays are also supported, where user-based tracking is achieved by tying the viewing position to one of the Sensors. In this way, we can integrate user-based tracking into the environment as easily as we integrate other tracking and haptic devices. In addition, by attaching the viewing location to a sensor linked to an instrument, we can trivially obtain an endoscopic view from any given instrument

2.10 Other features

As briefly stated above, the system also supports a number of other features. Voice input and output is achieved by *Spring* connecting to a *voiceserver* computer over the network. In this way, the user can select surgical instruments by speaking their name and indicating other commands in an easy, hands-free manner.

A mechanism for replicating the display of the simulation is required for collaborative viewing. A DisplayReplicator class can, at each screen refresh, copy the data from the screen (in stereo or monoscopic modes), compress it, and send it to a remote client to view the live (possibly stereo) video imagery of the simulation at real-time rates.

In other applications, it is sometimes necessary to receive live video and texture map this video data in real-time within the environment. Therefore, Spring can also connect to a videoserver to receive live video and texture map that data live onto an object.

Finally, in surgical simulation, sometimes other senses besides vision and tactile are necessary in order to reproduce the experience of a particular surgical procedure. For this reason, limited support for audio output is also provided.

3.0 Applications

A number of applications have been developed during the production of this simulator, including a microsurgery simulator[30,31]; a clinically evaluated, haptic-rate hysteroscopy simulator[26,27] simulating cervical dilation, endometrial ablation, and resection of intrauterine polyps; an intraoperative assistance environment with an advanced system for surgical assistance with voice input and output and virtual "hanging windows" for the live display of CT data, vital signs, and live endoscopic video; a surgical simulator for rat dissection and astronaut training system[32] with hand-based interaction with objects in the virtual environment; and its use for patient-specific surgical planning[33,34]. New applications under development include a colonoscopy simulator[35], a stent placement simulator), as well as a cleft-lip surgery simulator.

4.0 Conclusion

We have sought to develop a generalized framework for surgical simulation that can support the requirements of many surgical simulation applications. The simulator has been refined and enhanced during the development of a number of applications and supports multi-instrument, generalized interactions with networked haptics within a collaborative, multi-user environment. We hope that the description presented here enhances the discussion of the technical details of simulation and leads to the greater proliferation of surgical simulation applications.

5.0 Acknowledgements

The authors would like to thank the numerous other individuals that have contributed to the production of this system. Other developers (Michael Madison, Bharath Beedu, Jeremie Roux, CJ Slyfield, Yanto Muliadi, and Tyler Kohn), together with clinical collaborators (Simon Wildermuth, Michael Stephanides, Stephen Schendel, Leroy Heinrichs, Parvati Dev) have greatly contributed to this work. In addition, other technical collaborators (Jean-Claude Latombe, Richard Boyle, and Alexander Twombly) also contributed to this effort. This work was supported by grants from NASA (NCC2-1010), NIH (NLM-3506, HD38223), NSF (IIS-9907060), and a generous donation from Sun Microsystems.

References

1. Satava, R; "Robotics, Telepresence, and virtual reality: a Critical Analysis of the future of surgery", Minimally Invasive Therapy v1:357-363, 1992.
2. P. Gorman, J. Lieser, W Murray, R Haluck, and T. Krummel, "Evaluation of Skill Acquisition Using a Force-Feedback, Virtual Reality-based Surgical Trainer", Medicine Meets Virtual Reality 1999, Ed: J Westwood, IOS Press, 1999, pp. 121-123.
3. R., O'Toole, Playter, R, Krummel, T, Blank, W, Cornelius, H; Roberts, W; Bell, W; Raibert, M; "Measuring and developing suturing technique with a virtual reality surgical simulator", J. Amer Coll Surgeons, v189(1), 1999, pp 114-127.
4. D. Baraff and A. Witkin. Dynamic simulation of nonpenetrating flexible bodies. *Computer Graphics*, 26(2):303– 308, 1992.
5. E. Keeve, S. Girod, and B. Girod. Craniofacial surgery simulation. In *Proceedings of the 4th International Conference on Visualization in Biomedical Computing (VBC '96)*, pages 541–546, Sept. 1996.
6. U. G. K¨uhnapfel, H. K. C¸ akmak, and H. Maaß. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics*, 24:671–682, 2000.
7. G. Picinbono, H. Delingette, and N. Ayache. Non-linear and anisotropic elastic soft tissue models for medical simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2001.
8. D. Terzopoulos and K.Waters. Physically-based facial modelling, analysis, and animation. *The Journal of Visualization and Computer Animation*, 1:73–80, 1990.
9. N. Ayache, S. Cotin, and H. Delingette. Surgery Simulation with Visual and Haptic Feedback. In *Robotics Research*, Springer, 1998, pp. 311-316.
10. J. Berkley, S. Weghorst, H. Gladstone, G. Raugi, D. Berg, and M. Ganter. Fast Finite Element Modeling for Surgical Simulation. *Proc. Medicine Meets Virtual Reality (MMVR'99)*, ISO Press, 1999, pp. 55- 61.

11. C. Bosdogan, C. Ho, M.A. Srinivasan, S.D. Small, and S.L. Dawson. Force Interaction in Laparoscopic Simulation: Haptics Rendering of Soft Tissues. *Proc. Medicine Meets Virtual reality (MMVR'98)*, Jan. 1998, pp. 28-31.
12. M. Bro-Nielsen and S. Cotin. Real-Time Volumetric Deformable Models for Surgery Simulation Using Finite Elements and Condensation. *Proc. Eurographics'96*, Vol. 15, 1996, pp. 57-66.
13. G. Szekely; M. Bajka; C. Brechbuhler; J. Dual; R. Enzler; U. Haller; J. Hug; R. Hutter, N. Ironmonger; M Kauer; V. Meier; P. Niederer; A. Rhomberg, P. Schmid; G. Schweitzer; M. Thaler; V. Vuskovic; G. Troster; "Virtual Reality-Based Surgery Simulation for Endoscopic Gynecology", *Proc. Medicine Meets Virtual reality (MMVR'99)*, Jan. 1999, pp. 351-357.
14. N. Robins, GL Utility Toolkit (GLUT)- http://www.opengl.org/developers/documentation/glut
15. P. Rademacher, GL User Interface (GLUI) Library- http://www.cs.unc.edu/~rademach/glui
16. T. Davis, Micro User Interface (MUI) library- http://www.opengl.org/developers/code/mjktips/mui
17. M. Garland and P. Heckbert, "Surface Simplification Using Quadratic Error Metrics", In *ACM SIGGRAPH 97 Conference Proceedings*, pages 43–52, 1997.
18. Brown, J; Sorkin, S; Bruyns, C; Latombe, JC, Montgomery, K; Stephanides, M; "Real-Time Simulation of Deformable Objects: Tools and Application", Computer Animation 2001, Seoul, Korea, November 6-8, 2001.
19. Sorkin, S; "Distance Computation Between Deformable Objects", Honors Thesis, Computer Science Department, Stanford University, June 2000.
20. Quinlan, S, "Efficient Distance Computation Between Nonconvex Objects", Proc. IEEE Int Conf on Robotics and Automation, pp. 3324-3329, 1994.
21. 3D Reconstruction Web Site: http://biocomp.stanford.edu/3dreconstruction
22. H. Delingette. Towards realistic soft tissue modeling in medical simulation. In *Proceedings of the IEEE : Special Issue on Surgery Simulation*, pages 512–523, Apr. 1998.
23. D. Baraff and A. Witkin. Large steps in cloth simulation. In *ACM SIGGRAPH 98 Conference Proceedings*, pages 43–52, 1998.
24. G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.
25. S. Gottschalk, M. C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. In *ACM SIGGRAPH 96 Conference Proceedings*, pages 171– 180, 1996.
26. Montgomery, K; Bruyns, C; Wildermuth, S; Hasser, C; Ozenne, S; Bailey, D; Heinrichs, L; "Surgical Simulator for Hysteroscopy: A Case Study of Visualization in Surgical Training", IEEE Visualization 2001, San Diego, California, October 21-26, 2001.
27. Montgomery, K; Heinrichs, L; Bruyns, C; Wildermuth, S; Hasser, C; Ozenne, S; Bailey, D; "Surgical Simulator for Operative Hysteroscopy and Endometrial Ablation", International Society for Computer-Aided Surgery (ISCAS), Computer-Aided Radiology and Surgery (CARS 2001), Berlin, Germany, June 27, 2001.
28. Bruyns, C; Senger, S; Montgomery, K; Wildermuth, S; "Real-Time Interactive Cutting Using Virtual Surgical Instruments", Medical Image Computing and Computer-Assisted Interventions (MICCAI 2001), Utrecht, The Netherlands, October 14-17, 2001.
29. Montgomery, K; Mazzella, F; Stephanides, M; Schendel, S; "A High-Resolution Stereoscopic Computer Projection Display for Surgical Planning", Society for Information Display 2001 International Symposium Digest of Technical Papers, Vol. 32, pp. 359-361, June 2001.
30. Montgomery, K; Stephanides, M; Brown, J; Latombe, JC; Schendel, S; "A Virtual Environment for Training in Microsurgery", SPIE- The Optical Engineering Society, v3639(1), pp. 398-403, Jan 1999.
31. Brown, J; Montgomery, K; Latombe, JC; Stephanides, M; "A Microsurgery Simulation System", Medical Image Computing and Computer-Assisted Interventions (MICCAI 2001), Utrecht, The Netherlands, October 14-17, 2001.
32. Bruyns, C; Montgomery, K; Wildermuth, S; "A Virtual Environment for Simulated Rat Dissection: A Case Study of Visualization for AstronautTraining", IEEE Visualization 2001, San Diego, California, October 21-26, 2001.
33. Montgomery, K; Stephanides, M; Schendel, S; "Development and application of a virtual environment for reconstructive surgery", Journal of Computer-Aided Surgery, v5(2), ISSN: 1092-9088, 2000, pp:90-97.
34. Montgomery, K; Stephanides, M; Schendel, S; Ross, M; "A Case Study Using the Virtual Environment for Reconstructive Surgery", IEEE Visualization, Research Triangle Park, NC, October, 1998
35. Wildermuth, S; Bruyns, C; Montgomery, K; Beedu, B; Marincek, B; "Patient Specific Surgical Simulation System for Procedures in Colonoscopy", Vision, Modeling, and Visualization (VMV01), Stuttgart, Germany, November 21-23, 2001.