An Approach to Distributed Interactive Simulation and Visualization of Complex Systems using Cluster Computing

Denis Gračanin
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061, USA
gracanin@vt.edu

Abstract

When dealing with complex systems, interactive, real-time simulations require significant computational capabilities that can be provided by cluster computing. Current cluster computing based techniques are mostly focused on batch jobs. However, it is possible to use clusters so that an application can run and directly communicate with the remote client(s). Direct communication enables, without loss of accuracy or frame rate, real time visualization of and interaction with much larger models compared to a single machine implementation. The degree of coupling between the dependent variables in the model determines the degree of parallelization that can be achieved by evaluating the solution for each dependent variable in parallel. A distributed mass-spring simulation system was developed to serve as an open platform that can be used to improve the scalability of the simulation computation. Several techniques are used to improve scalability, both in terms of the problem size and number of clients. The developed system provides support for large scale mass-spring simulations to leverage available cluster computing and visualization resources. It can be applied to a wide range of problems related to deformable solids including many biologically related like human organ modeling and medical animation where real-time feedback is required.

1 Introduction

The widespread use of parallel and grid computing, combined with the tremendous advances in graphics capabilities, enables manipulation and visualization of large data sets. Those data sets are usually simulation results, dynamically generated as a result of interactions with users.

Interactive simulation and visualization applications use computational steering as a way to integrate modeling, simulation, data analysis, and visualization [16]. Visualization and simulation applications cover many areas such as fluid flow modeling [6], cloth modeling [7], soft tissue modeling [8], and deformable solids in general [11]. In order to improve interactivity and usability of visualization applications, several issues need to be addressed. Those include control structures, data distribution, data presentation, and user interfaces [16].

Parallel (cluster) computing requires efficient parallel algorithms and models that can exploit parallelism in a problem structure. The challenge is how to partition the problem to best utilize computational resources. In other words, the question is what work can be done concurrently and what communication among concurrent processes is required [18]. Message-Passing Interface (MPI) [2] is a communication library that provides a support for communication among the processes participating in a parallel computation.

A mass-spring model is frequently used in simulation of deformable solids. The model primarily demonstrates springs applying forces to masses and the resulting movement of those masses (particles). Typically, it also includes environmental effects. Some mass-spring simulations allow springs to have a fixed oscillation period. This feature allows a spring to be a source of energy. Mass-spring simulations are used to model the physical behavior of deformable solids such as cloth, soft tissue, marshmallows, or metal. The designer controls the physical behavior of the modeled solid by adjusting the strength of the springs and the interconnection geometry [14, 25].

Bioinformatics [22] and biomedical computing [17] related simulation often include soft tissue modeling where the mass spring model can be effectively used to achieve realistic soft tissue modeling in medical and related simulations [10].

A distributed mass spring simulation tool was developed to provide for large-scale, multi user mass spring simulations [14, 29] in real time, with distributed interactions for multiple clients. It uses MPI and a custom communication protocol to combine cluster computing and visualization resources.

The remainder of the paper is organized as follows. Section 2 provides an overview of related work in parallel computing, bioinformatics/medicine, and interactive simulation visualization. Section 3 describes the mass spring model used for simulation. Section 4 describes the implementation of the developed distributed simulation and visualization system and its use. Section 5 concludes the paper and provides directions for future work.

2 Related Work

Before describing the mass spring model used and the developed tool, an overview of related work in parallel computing, bioinformatics/medicine, and interactive simulation visualization is provided.

2.1 Parallel Computing

The need for high performance computing exists in many areas of science, engineering, medicine, and commerce. The availability of powerful processors and high-speed networks, as well as the rapidly maturing software components, provide support for high performance and high availability applications. The research on cluster computing, grid computing, and related parallel computing mechanisms provides a foundation for many simulation based applications [2, 15], ranging from atomic particles modeling to information visualization [13]. Cluster computing is often based on combining commercial off-the-shelf (COTS) computing nodes with high speed networks to form a powerful computing resource. However, visualization in grid computing environments provide a challenge for system designers [4].

One of the particularly interesting problems is a fast simulation of deformable objects [11]. Two widely used approaches are Finite Element Modeling and Mass Spring Systems. While finite element modeling is very accurate, it is also very slow. The mass spring systems approach is not as accurate but it is fast, simple and general enough to be used in many simulation applications. Modeling deformable objects is important for many bioinformatics and medicine related problems involving soft tissue modeling.

2.2 Bioinformatics and Medicine

Bioinformatics [9, 20, 21, 22] is one of the scientific disciplines that benefited from advances in computing capabilities [1]. However, unlike more traditional high-performance computing application areas, biology related problems are often difficult to parallelize. Huge data sets may cause even efficient, polynomial time algorithms, to execute very slowly. Many problems, like protein folding, are NP-hard. However, sufficient cluster computing capabilities may help in evaluation or in solving of some instances of the problems. Problem solving environments like SCIRun and BioPSE [17] provide support for large-scale computations, including interactive creation, investigation and steering. Another class of applications includes information rich virtual environments for biological simulations [26].

Medical simulations often require realistic modeling of soft tissues [10, 28] for scientific analysis, surgery planning, [3] and surgery procedure training [8]. Soft-tissue deformation accuracy and computation time are the main criteria used in modeling and determine the ability to provide for the real-time soft tissue deformation computation [24]. The latency (or delay) caused by longer computation time may significantly reduce or even disable the immersion of the operator and its ability to learn.

2.3 Interactive Simulation Visualization

Traditionally, visualization was used to display data that were generated “off-line” as a result of a long simulation run [6]. Interactive visualization displays data on the fly, usually during simulation, while allowing a user to select visualization parameters like levels of detail, viewpoints, angles, etc [12]. Computational steering and interactive visualization can be implemented in an immersive virtual environment [30] that provides users with a better insight and a sense of immersion.

Modeling virtual humans [23] provides an example of a class of problems that are computationally intensive but need to be solved in real-time while maintaining good visual perception [5]. Some topics include body deformations, hair simulation, cloth simulation [7], facial deformation models, speech animation, etc.

3 Mass Spring Model

Mass spring simulations model a particle as a simple circle or sphere. A particle’s minimal intrinsic property is its mass. Most mass spring simulations are a specialized form of a particle system. In other words, the simulation does not track or calculate the rotations or moments of inertia of individual masses. The only concern is a translational motion of particles. All other types of motion are ignored. Particles may also have an elasticity property.

A spring’s minimal intrinsic properties are its rest length and its spring constant. Additionally, a damping constant may be assigned to each spring. The rest length of a spring is the natural length of the spring when it is neither being stretched or compressed. The spring constant determines the stiffness of a spring. The higher the spring constant, the more difficult it is to stretch or compress the spring.

A springs damping constant is a measure of friction forces within the spring when it is changing shape. The friction forces resulting from damping dissipate some of the energy involved in changing the springs length. Without damping, a spring could oscillate forever trying to return to its rest length.

The springs in a mass spring simulation are simplified versions of real springs and behave according to the simplest spring equations. A real spring behaves according to the basic spring equations when not stretched or compressed too far; but when distorted beyond a certain amount a real spring behaves in a much more complex manner. The simplified springs in most mass spring simulations follow the basic spring equations even when distorted well beyond normal ranges.

3.1 Simulation

In the case of mass spring simulation, the goal is to simulate the motion of particles according to the well known laws of physics. Using a formula for the position $\vec{r}$ of a particle as a function of time, such that:

$$\vec{r} = \vec{P}(t)$$

then finding the formula for velocity of the particle is a simple matter of taking the derivative of the function $\vec{P}$ with respect to time. The simulator can calculate all of the forces, and thus the resulting net force a particle is experiencing at any given moment.

The problem is to determine the motion of the particle based on those forces. For simple forces, it is sometimes possible to solve the differential equations analytically, but for a mass spring system, the simulator must solve the problem numerically. Mass spring simulations first calculate the resulting net force on each particle. The net force determines the net acceleration, which the simulation then numerically integrates to find the velocity.

A second numerical integration finds the new position of the particle at some (usually very short) time in the future. This process repeats for each (visualization) frame. The new forces are calculated that determine new accelerations, which then determine the new velocities, and so the new positions.

The problem can be expressed as an ordinary differential equation initial value problem (ODEIVP) [27]. There are two possible sources of parallelism in ODEIVP algorithms. Those are parallelism across the method (time) and parallelism across the system (space).

Parallelism across time involves dividing the time range into smaller ranges to improve accuracy, provided it is feasible to solve each of the smaller ranges in parallel. However, the solutions for one time range typically require the solution of the neighboring range as input. This coupling and dependency between consecutive time ranges makes it rather impractical [27]. In addition to dividing the time domain into sub domains, the solution for each sub domain is solved using multiple evaluations of the derivative within that range. Each of these evaluations is called a stage. Therefore, it may be possible to compute each derivative evaluation within a stage in parallel.

In most ODEIVP algorithm formulations there is a high degree of coupling between these evaluations [15]. For example, the classic fourth-order Runge-Kutta algorithm has four stages and each stage requires the output from the previous stage. Consequently, due to internal coupling, it is not possible to leverage parallelism across the method.

Parallelism across the system attempts to evaluate the solution for each dependent variable in parallel [2]. The degree of coupling between the dependent variables determines the degree of success this approach may achieve. Higher degrees of coupling either limit parallelism or increase communication overhead between processes. The problem is how to determine the optimal distribution of computation among the available processes.

It is often possible to take advantage of the structure of many problems to formulate a good or perhaps even optimal solution to the mapping problem. A regularly tessellated geometry model with highly structured spring connections, like a rectangular cloth patch, or soft tissue, can be divided into two equal pieces along the smallest dimension and use two processors in parallel.

This technique minimizes the number of springs spanning the two processors while providing each processor an equal number of masses to track. This technique can be used recursively, subdividing each patch along its smallest dimension at each step, provided the number of processors is always an integral power of two [14].

This partitioning technique work well for distributing the work involved with each mass in the simulation. However, the computations for the spring force are not trivial. In a simple simulation involving only gravity and spring forces, the spring force requires the most computation. In most mass spring models there are more springs than masses.

Springs can be divided into two categories based on the mapping. The first category includes springs that connect masses assigned to the same processor. They are mapped to the same processor as the two masses they connect.

The second category includes springs that connect masses assigned to different processors and communication overhead cannot be avoided. The system must communicate the position of one of the two masses connected by the spring to the processor that owns the other mass. Without this communication, the relative positions of the two masses, and thus the length of the spring, cannot be calculated [14, 29].

4 Implementation

The Distributed Mass Spring Simulation system (DMSS) has been developed to serve as an open platform that can take advantage of new cluster and visualization systems as they become available. It has been designed with three main goals mind: [29]:

1. Analyze and develop software and hardware solutions to improve the scalability of the simulation computation.
2. Analyze and develop a communication protocol to support multiple interacting clients within the simulation space.
3. Develop and test client visualization and interaction models.

The first goal is to attempt to scale the simulation to make use of as many masses and springs as possible. The solution reduces the amount of calculation by adapting to the variable complexity of the simulation model. It provides more computation where necessary to maintain accuracy, but limits the amount of calculation when feasible. Additionally, a numerical integration algorithm was adapted to work in parallel.

The second goal is to provide for the real time distributed interaction and visualization of the simulation results. That required the development of the network architecture and the design of the communication protocol for use between clients and the server. Many factors were considered, including the amount of traffic that would be necessary as the size of the server simulation is increased. In addition, the protocol enables the visualizations and interactions that are required by the client interface during the simulation [29].

The third goal is to improve the user's experience by providing mechanisms for visualization and interaction. One of the results is a textured visualization that allows the client to view the textured mass spring system model. In addition, alternate methods of interaction are available, including the use of first person perspective and the use of hardware devices such as a SpaceBall.

The DMSS has a client-server architecture designed to enable real time interaction with a large scale mass spring simulation and use of distributed cluster computing resources. Clients connect individually to the simulation server through a network connection and interact with the server by sending updates to their client-controlled interaction device. The server incorporates this updated information into the virtual model, and performs simulation steps based on the physical model. As simulation steps are completed, updated mass information is distributed to all participating clients. The key components in this architecture are the server, the clients, and the network architecture and protocol that connect them.

4.1 DMSS Server

The main purpose of the DMSS Server is to "execute" the physical model that drives the simulation and is displayed at DMSS Client applications that are connected to the DMSS server. The clients are driven by streaming scene updates from the server at the end of each animation step completed at the server.

The server application is executed at the command line with only two parameters, the port to listen to for client connections and the port that updates will be sent out to clients on. All other configurable parameters of the simulation are set by way of XML-based scene files that are uploaded by the client prior to the execution of a new simulation. All models that are to be simulated by the server are loaded during runtime by a client.

The DMSS server is responsible for performing the numerical integration step required to animate the mass spring model. A number of physically modeled forces are allowed by this animation model, including viscosity, gravity, electromagnetic forces, and spring forces.

The current animation routines are fairly rudimentary, and thus in order to satisfy the first goal, a number of improved integration techniques have been examined to improve performance characteristics. The integration model that is of most interest is an adaptive algorithm that is able to adjust the number of computations necessary to achieve a result that is sufficiently accurate. The accuracy of the computation can be adjusted as necessary to speed up the calculations.

In addition, work is being done to further satisfy the first goal of the DMSS by migrating the animation server to a clustered architecture consisting of anywhere from tens to thousands of processing units. The implementation is based on the freely available MPI software library, which is available for a variety of platforms and operating systems.

The clustered server implementation is designed to handle mass spring systems that are far more complex than are possible on a single processor desktop machine. This speed is necessary in order to allow for real time interaction with a large scale simulation. It has always been possible, given enough time, to simulate the types of large scale simulations that will be possible with the clustered DMSS; however, the computation generally had to be done offline before real-time viewing was possible. With the clustered DMSS server it is hoped that simulations that previously were too complex to be interactive will now allow for real time interac-

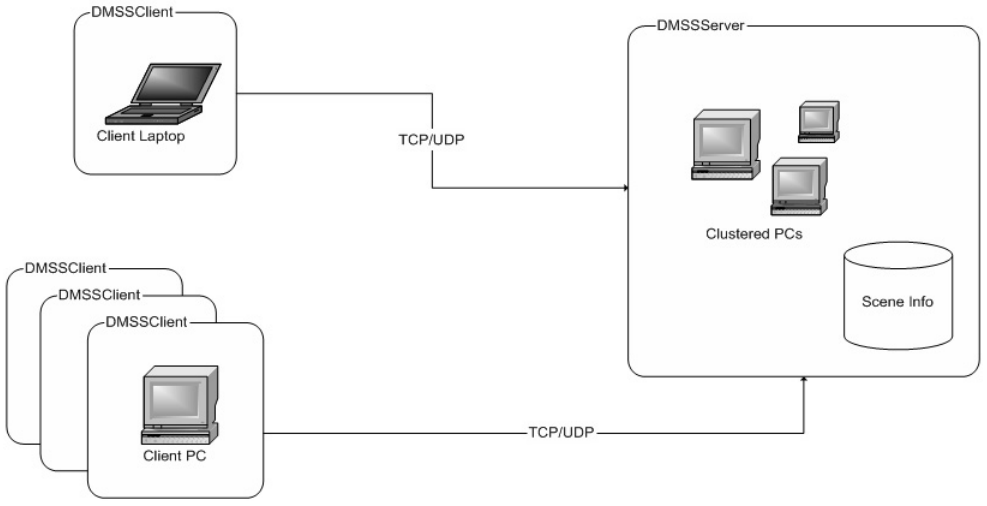

Figure 1. DMSS deployment [29].

tion from multiple clients.

4.2 DMSS Client

The DMSS Client is a graphical application that creates a virtual environment by provided a 3D visualization of active simulations and allows users to interact. The client application is designed to run in an immersive CAVE based environment or on a regular workstation using a keyboard and mouse for interaction.

The client application has a number of options which are used to control the visualization; however, the main options that must be specified when the client is executed is the IP address of the server and the port that it is listening on. From this point onward communication with the server is transparent to the user.

Figure 2 shows the interface that is presented in the current DMSS client implementation. The two spherical objects represent the probes manipulated by two users in order to interact with a deformable object, in this case a cloth. All movements of other users probes are reflected in the visualization that is displayed to each user. Other interactions such as starting and stopping the simulation and opening a new scene file are carried out through keyboard operations.

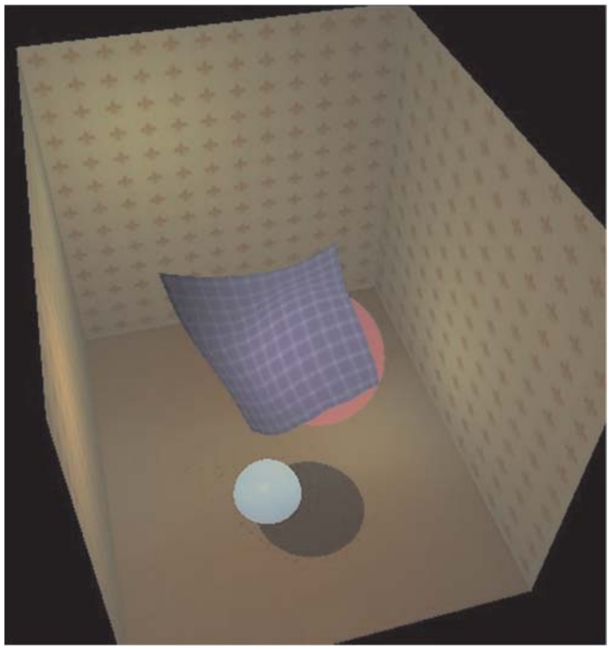

Figure 2. DMSS user interface [29].

4.3 Implementation Issues

In the initial implementation of the server, the Client Handler Thread would block after receiving a probe update until an animation step was completed and the scene data area was released by the Animation Thread. This caused the responsiveness of the server to probe updates to be very jumpy at times, especially when many updates were sent over a short period of time. The Update Queue improved responsiveness by eliminating the block; thereby allowing the Update Queue to respond to pending updates more quickly. The addition of the Update Queue had no effect on the quality of simulation results.

Another problem that was discovered in the initial implementation of the protocol was that the original Probe Update message sent by clients over the Control Channel was to cause a confirmation message to be sent from the server to the client. Because probe updates are such a frequent message sent out by the client, this caused excess network traffic to be generated. As a result, the protocol was modified so that no confirmation message from the server is required when a Probe Update message is sent.

The issue of how to ensure that client probes are removed from other clients scenes when a client exits presented some challenges as well. Ideally the server would send out a message to all clients whenever a new client probe has been added or removed from the scene. However, the current protocol does not provide for any reliable communication initiated by the server to the clients; all communication from the server to the client goes out over the unreliable Update Channel.

As a result, the server sends out a heartbeat message to each client for any inactive probes, and that the client should simply remove probes that have not been updated within a prescribed timeout period. As a consequence, the client can reliably remove any other client probe that has exited the simulation, and at worst a clients probe may flicker on and off if a heartbeat message is missed. Regardless, the server is always aware of what probes exist in the simulation, so there is never any affect on the simulation itself.

The newest client implementation will be based on the DIVERSE (Device Independent Virtual Environments—Reconfigurable, Scalable, Extensible). It is a highly modular collection of complimentary software packages, containing both end-user programs and C++ APIs, designed to integrate distributed simulations with heterogeneous virtual environments (VEs) [19]. The main characteristics include:

- Allows an application to be run, unmodified, on all supported platforms,
- Supports application-independent interfaces optimized for each platform, (For example, a desktop interface can be used on a desktop system, and an immersive interface can be used in an immersive system, without needing to modify the underlying application.)
- Provides emulators of immersive systems to support development and debugging of immersive systems on non-immersive systems, and,
- Includes tools that allow non-programmers to display and interact with their data.

Diverse automatically handles most of the details of graphics display and device access and supports the separate development of applications and user interfaces.

The Diverse Toolkit (DTK) [19] provides many utilities, the most distinguishing of which is a novel implementation of remote shared memory. In addition to providing a general Inter-Process Communication (IPC) application-programming tool, DTK remote shared memory is the IPC method with which DTK provides the seamless distribution of DTK I/O device services. It provides a standard coding interface to I/O device data for local (same computer) and remote access, without requiring that local and remote access to be coded differently.

5 Conclusion and Future Work

The developed DMSS tool provides support for large scale mass spring simulations to leverage available cluster computing and visualization resources. Several techniques are used to improve scalability, both in terms of the problem size and number of clients. Users can collaborate in manipulation of the mass spring simulation models.

This approach can be applied to a wide range of problem related to deformable solids including many biologically related like protein decomposition, human organ modeling, interactive surgery simulation and medical animation where real-time feedback is required. Collaboration and team training aspects of the tool are of particular importance.

The biologically related problems are the focus of future research. The effect of latency on multi user collaboration will be evaluated to see how effectively collaborative tasks, including manipulation of complex biological systems, can be performed in real time.

Acknowledgements

Christopher L. Hines, Kevin A. Reichert, and Jason Dudash worked on the development, implementation, and testing of the DMSS as a part of their graduate studies at Virginia Tech.

References

[1] D. A. Bader. Computational biology and high-performance computing. *Commun. ACM*, 47(11):34–41, 2004.

[2] R. H. Bisseling. *Parallel Scientific Computation: A structured approach using BSP and MPI.* Oxford University Press, Oxford, 2004.

[3] M. Bro-Nielsen, D. Helfrick, B. Glass, X. Zeng, and H. Connacher. VR simulation of abdominal trauma surgery. In J. D. Westwood, H. M. Hoffman, D. Stredney, and S. J. Weghorst, editors, *Proceedings of the Medicine meets virtual reality: Art, science, and technology*, pages 117–203, Amsterdam, 1998. IOS Press.

[4] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, and J. Wood. Visualization in grid computing environments. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 155–162, Washington, DC, USA, 2004. IEEE Computer Society.

[5] V. Bruce, P. R. Green, and M. A. Georgeson. *Visual Perception: Physiology, Psychology and Ecology*. Psychology Press, Hove and New York, 2003.

[6] J. X. Chen, D. Rine, and H. D. Simon. Advancing interactive visualization and computational steering. *IEEE Computational Science and Engineering*, 3(4):13–17, Winter 1996.

[7] L. Chittaro and D. Corvaglia. 3D virtual clothing: from garment design to web3D visualization and simulation. In *Web3D '03: Proceeding of the eighth international conference on 3D Web technology*, pages 73–ff, New York, NY, USA, 2003. ACM Press.

[8] K. S. Choi, H. Sun, P. A. Heng, and J. C. Y. Cheng. A scalable force propagation approach for web-based deformable simulation of soft tissues. In *Web3D '02: Proceeding of the seventh international conference on 3D Web technology*, pages 185–193, New York, NY, USA, 2002. ACM Press.

[9] J. Cohen. Bioinformatics— an introduction for computer scientists. *ACM Comput. Surv.*, 36(2):122–158, 2004.

[10] H. Delingette. Towards realistic soft tissue modeling in medical simulation. *Proceedings of the IEEE*, 86(3):512–523, Mar. 1998.

[11] A. Duysak and J. J. Zhang. Fast simulation of deformable objects. In *Proceedings of the Eighth International Conference on Information Visualisation*, pages 422–427, 14–16 July 2004.

[12] C. D. Hansen and C. R. Johnson, editors. *The Visualization Handbook*. Elsevier, Amsterdam, 2005.

[13] M. C. Hao, U. Dayal, D. Cotting, T. Holenstein, and M. Gross. Accelerated force computation for physics-based information visualization. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 59–66, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[14] C. L. Hines. Parallel mass-spring simulation. Technical report, Virginia Polytechnic Institute and State University, Falls Church, VA, Jan. 5 2004.

[15] C. Hughes and T. Hughes. *Parallel and Distributed Programming Using C++*. Addison-Wesley, Boston, 2004.

[16] C. Johnson, S. G. Parker, C. Hansen, G. L. Kindlmann, and Y. Livnat. Interactive simulation and visualization. *IEEE Computer*, 32(12):59–65, Dec. 1999.

[17] C. R. Johnson, R. MacLeod, S. G. Parker, and D. Weinstein. Biomedical computing and visualization software environments. *Commun. ACM*, 47(11):64–71, 2004.

[18] G. E. Karniadakis and R. M. Kirby II. *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation*. Cambridge University Press, Cambridge, United Kingdom, 2003.

[19] J. Kelso, S. G. Satterfield, L. E. Arsenault, P. M. Ketchan, and R. D. Kriz. DIVERSE: A framework for building extensible and reconfigurable device-independent virtual environments and distributed asynchronous simulations. *Presence*, 12(1):19–36, Feb. 2003.

[20] D. E. Krane and M. L. Raymer. *Fundamental Concepts of Bioinformatics*. Benjamin Cummings, San Francisco, 2003.

[21] S. A. Krawetz and D. D. Womble. *Introduction to Bioinformatics: Theoretical and Practical Approach*. Humana Press, Totowa, New Jersey, 2003.

[22] A. M. Lesk. *Introduction to Bioinformatics*. Oxford University Press, Oxford, UK, 2002.

[23] N. Magnenat-Thalmann and D. Thalmann, editors. *Handbook of Virtual Humans*. John Wiley & Sons, Ltd, West Susses PO19 8SQ, England, 2004.

[24] K. Montgomery, C. Bruyns, S. Wildermuth, C. Hasser, S. Ozenne, D. Bailey, and L. Heinrichs. Surgical simulator for hysteroscopy: A case study of visualization in surgical training. In *Proceedings of the IEEE Visualization 2001 (VIS'01)*, pages 449–587, 21–26 Oct. 2001.

[25] H. Ohanian. *Physics*. W. W. Norton & Company, New York, 1985.

[26] N. F. Polys, D. A. Bowman, C. North, R. Laubenbacher, and K. Duca. Pathsim visualizer: an information-rich virtual environment framework for systems biology. In *Web3D '04: Proceedings of the ninth international conference on 3D Web technology*, pages 7–14, New York, NY, USA, 2004. ACM Press.

[27] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, 1988.

[28] A. Radetzky, A. Rnberger, M. Teistler, and D. Pretschner. Elastodynamic shape modeling in virtual medicine. In *Proceedings of the International Conference on Shape Modeling and Applications*, pages 172–178, 1999.

[29] K. A. Reichert. Construction of a large-scale distributed mass-spring simulation. Technical report, Virginia Polytechnic Institute and State University, Falls Church, VA, Jan. 6 2004.

[30] L. Renambot, H. E. Bal, D. German, and H. J. W. Spoelder. CAVEStudy: an infrastructure for computational steering in virtual reality environments. In *Proceedings of the Ninth International Symposium on High-Performance Distributed Computing*, pages 239–246, 1–4 Aug. 2000.