Adding Force Feedback to Graphics Systems: Issues and Solutions

William R. Mark[1] Scott C. Randolph[2] Mark Finch[3] James M. Van Verth[4] Russell M. Taylor II[5]

Department of Computer Science*
University of North Carolina at Chapel Hill

ABSTRACT

Integrating force feedback with a complete real-time virtual environment system presents problems which are more difficult than those encountered in building simpler force-feedback systems. In particular, lengthy computations for graphics or simulation require a decoupling of the haptic servo loop from the main application loop if high-quality forces are to be produced. We present some approaches to these problems and describe our force-feedback software library which implements these techniques and provides other benefits including haptic-textured surfaces, device independence, distributed operation and easy enhancement.

CR Descriptors: H.1.2 [Models and Principles]: User/Machine Systems; C.3 [Special-Purpose and Application-Based Systems]: Real-time systems; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – *Virtual Reality*; I.6.8 [Simulation and Modeling]: Types of Simulation – *Distributed*.

Additional Keywords: haptic, force feedback, friction model, intermediate surface representation, scientific visualization, interactive graphics, virtual environment, virtual world.

1. INTRODUCTION

As designers of interactive computer systems work to increase the information flow between the computer and the user, sensory modalities other than vision become increasingly important. One such modality is force feedback. The sensing of forces is closely coupled to both the visual system and one's sense of three-dimensional space; the eyes and hands work in concert to explore and manipulate objects.

* CB #3175, Sitterson Hall; Chapel Hill, NC 27599. Tel. +1.919.962.1700
Authors' current organizations and contact information:
1 UNC-CH; markw@cs.unc.edu; www.cs.unc.edu/~markw
2 Spectrum Holobyte; randolph@holobyte.com; www.holobyte.com
3 Numerical Design, Ltd.; mf@ndl.com; www.ndl.com/ndl
4 Virtus Corp.; jim.van.verth@virtus.com; www.cs.unc.edu/~vanverth
5 UNC-CH; taylorr@cs.unc.edu; www.cs.unc.edu/~taylorr

Proceedings of SIGGRAPH 96 (New Orleans, Louisiana, August 4-9, 1996). In Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 447-452.

Force feedback usefully enhances the capabilities of virtual environment systems; [17] showed that force feedback increases productivity in solving rigid-body placement problems and [8] demonstrated an atomic-surface modification system which would not have been feasible with graphics alone.

Virtual environment force displays use models and algorithms described in the robotics and teleoperation literature for low-level control—see for example [9][22][23]. When combining a computer graphics engine, a simulation, and a force-feedback device into one system, there are several areas of concern in addition to that of low-level control. The force-feedback component of such a system should:

- Maintain a high update rate in the force servo loop.
- Present high quality forces without detectable artifacts.
- Transparently support different force-feedback devices.
- Interface easily and cleanly with the rest of the system.

We discuss some approaches to these problems and present the Armlib force-feedback library [13] as one solution.

2. PROBLEMS AND SOLUTIONS

It has been clearly shown that it is necessary to run the simulation and graphics loops of virtual environment (VE) systems asynchronously in order to maintain reasonable display update rates (around 20 Hz) in the presence of long simulation computations. [11][20]

Such a decoupling is even more critical for force display, where update rates of several hundred Hz are required to produce high-quality forces. The necessary rate depends somewhat on the characteristics of the force-feedback device and control algorithm, but, for example, [1] required an update rate of 500 Hz for their system. If the update rate falls below the required minimum, the user begins to notice high-frequency discontinuities and hard surfaces become either soft or unstable.

We can decouple the simulation and haptic loops on a single machine by using either multiple processors or *very* frequent context switches. However, it is often more practical to dedicate one real-time machine to the haptic servo loop, and use other machine(s) for the rest of the virtual environment tasks (simulation, high-performance graphics, etc.). This strategy allows each machine to be well matched to its task. It also allows for flexible system configuration, which is particularly useful in a research environment.

The general case of such a split system connects the force-feedback device directly to a *force server*. This server tracks the *probe* of the force-feedback device (held in the user's hand) and executes the force-feedback servo loop. The application connects to this force server through some communication channel, retrieving position information from the server and sending descriptions of forces or force fields to it. We currently use a TCP/IP Ethernet communications channel because we must connect to existing graphics and research equipment; a low-latency, high-bandwidth channel such as shared memory would be superior.

Kim et al. [12] did the first work in this area, showing that teleoperation systems benefit from a decoupling of low-level force servo loops from higher-level control. Adachi et al. [1] were the first to apply the technique to virtual environment force-feedback systems. Rather than simply supplying a single force vector to the force-feedback controller, they supply an *intermediate representation* (their term, which we adopt) for a force model. This representation is updated infrequently by the application code, but is evaluated at a high update rate by the force-feedback controller.

Gomez et al. [10] demonstrate a system which takes almost the opposite approach. Their main simulation runs on the force-feedback machine and sends state updates to a graphics machine.

2.1 Intermediate representations

The kind of intermediate representation that is most useful depends on the application. A molecular modeling system might use spheres of contact. An immersive design system could send a representation of nearby surfaces. A simulation meant to teach understanding of physics force fields [3] might send equations to the server that describe the field.

Mitsuishi et al. [16] demonstrate a remote milling system which uses an intermediate representation of average tool force.

We describe two general intermediate representations, *plane and probe* and *point-to-point spring*, and our extensions to these types.

Plane and probe

In the *plane and probe* model, the force server keeps models of a plane which the probe can contact. [1] When the probe penetrates the plane, a restorative spring force that depends on the depth of the penetration is applied. The result is a surface with controllable sponginess against which the user can push (see Figure 1).

Using this model, the application computes a local planar approximation to the surface at the user's hand location each time through its main loop. The user feels a firm plane (forces updated at ~1 kHz by the force server), while the plane's position is updated more slowly (at ~20 Hz) by the application. The increase in local force update rate from 20 Hz to 1 kHz dramatically increases the maximum firmness of the surface while maintaining a stable system.

Figure 2 shows how this this technique works in one application, the Nanomanipulator, which allows the user to control the motion of a microscopic tip as it travels over a surface. [8] The initial implementation of the Nanomanipulator system performed its force computations on the graphics host, using Armlib only to read positions and send forces to the force-feedback device. This method restricted the force updates to at most the system loop update rate, which was around 20 Hz. The result was either a soft surface with sluggish response or an unusably unstable surface. Decoupling the application and force servo loops using our plane and probe model resulted in a much more stable and stiff surface.

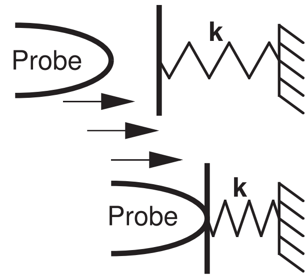

Figure 1: *A hard surface is approximated by a plane connected to a spring. When the probe encounters the plane, a spring force with spring constant **k** is applied. Very high values of **k** produce a surface that feels hard. The force is normal to the plane.*

Surface friction and texture

The surface model just described produces forces which are always perpendicular to the surface. The resulting surfaces feel like oiled glass, with the probe tending to slip off convex surface areas and into concave ones.

Previous researchers have demonstrated the importance of surface friction models in allowing the user to explore a surface without slipping. Several researchers [5][18][19] investigated models that combine static friction (which holds the probe at a fixed spot) with kinetic friction (which slows the probe's movement once it breaks free of the static friction). Adachi et al. [1] model only kinetic friction, using a velocity-based term.

Minsky et al. [15] model haptic surface textures using a friction-like technique. Rather than directly representing variations in surface height, they represent these variations using a 2D lateral force field. This field is proportional to the gradient of the surface height function. Because their force-feedback device has only two degrees of freedom, lateral forces are not proportional to the normal force, as is typical for a friction model.

We have implemented a friction model that includes both static and kinetic components and can represent simple surface textures. Adjustment of the parameters produces surfaces that feel like concrete, sand, rubber, skin, or cloth. The model is rapidly computable, allowing a high update rate. Figure 3 shows the parameters of our model graphically; an explanation follows.

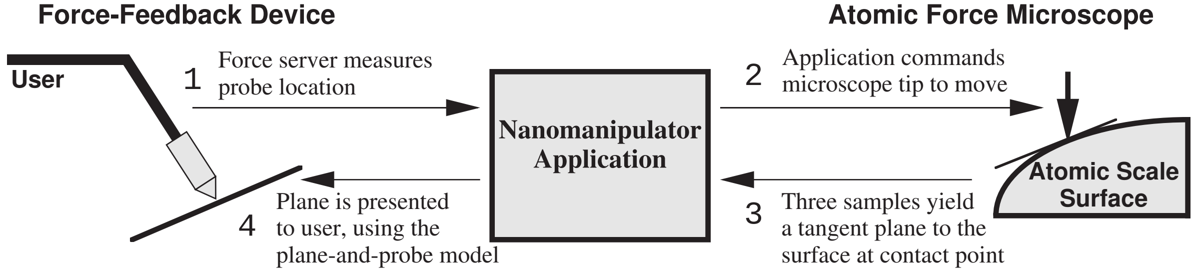

Figure 2: *The Nanomanipulator application uses surface height readings from the microscope tip to determine a local plane approximation which is sent to the force server.*

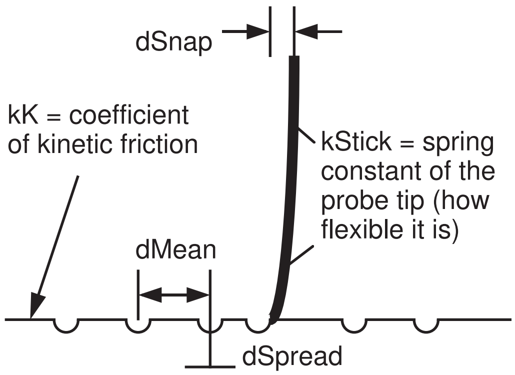

Figure 3: *Surface friction model. The tip slides across the surface against kinetic friction* ***kK*** *until it hits a snag. Snags populate the surface with mean distance* ***dMean*** *between them, uniformly distributed within* ***dSpread****. The tip sticks in the snag, bending with spring constant* ***kStick*** *until moved more than* ***dSnap****, then it jumps free.*

Our friction model is that of a surface populated by snags being probed by a flexible tip. When the tip is not stuck in a snag, it moves across the surface opposed by a friction force that is proportional to the normal force (with coefficient of kinetic friction *kK*). When the probe encounters a snag, it sticks there until the probe moves more than *dSnap* units away from the sticking point, in any tangent direction. While it is snagged, a force tangent to the surface pulls the tip towards the center of the snag. This force is proportional to both the normal force and to the distance from the snag center (with spring constant *kStick).*

The snags tend to hold the probe in place on the surface. This tendency provides a natural "station keeping" on surfaces with high snag density (such as sandpaper).

The snags are placed around the surface with a mean distance between snags of *dMean*, uniformly distributed within *dSpread*. In fact, we populate the surface with snags dynamically. After leaving a snag, the tip will encounter another placed with uniform probability between *dMean-dSpread/2* and *dMean+dSpread/2* units away from the first snag, regardless of the tangent direction traveled. Additionally, if the forward motion of the tip (movement away from the previous snag) ceases, it is considered to have encountered a snag at the point where forward motion stopped. Although actual surfaces could be measured to determine parameters for our model, in practice we have explored the parameter space interactively in order to produce different surfaces.

This parameterized snag distribution, which controls the transition from kinetic to static friction, is what sets our friction model apart from previous static/kinetic friction models. Salcudean and Vlaar [18] based their kinetic-to-static transition entirely on probe velocity. Salisbury et. al. [19] transitioned immediately, without providing steady-state kinetic friction. Our technique allows simulation of simple surface textures in addition to modeling standard friction. It provides these benefits while using only simple computations that allow us to maintain a high update rate.

Multiple planes

One plane often suffices to model a smooth surface, as it can be continually positioned in the correct orientation to provide the normal to the surface at the point of contact. However, a model of an object with a sharp inner edge (such as the inside corner of a box) requires multiple planes to constrain probe motion in several directions at once. Armlib extends [1] by providing this multi-plane capability, although as [24] points out, this technique can result in errors when the planes are not at right angles to each other.

Multiple probes

It is sometimes necessary to simulate a probe that is larger than a single point. An application that allows users to feel around in a virtual room with their hand is an example of such a system. One virtual probe is created for each finger on the hand and one for the palm, allowing the user to feel multiple contacts between the world and the hand (for example, resting the hand flat on a virtual desktop). Since the points might be contacting different objects, each has its own local surface with which it can collide. For example, when pulling out a chair, the thumb may rest on top of the chair while the index finger pulls it away from the table. Since the force-feedback device has only one physical probe, the user experiences the sum of the virtual probe forces; the effect is similar to sticking a single finger in a very stiff glove.

Point-to-point springs

Some applications allow the user to pick and drag objects which are subject to complex forces. Examples of such objects include rigid bodies participating in a many-body simulation and atoms in a protein, which are subject to forces determined by a molecular dynamics simulation. Often the calculation of the forces acting on the object is so complex that it can only be performed once or twice a second. Furthermore, there is usually no rapidly-computable local approximation to these forces which will remain valid for the entire interval between full calculations.

We approach this problem by implementing a compliant connection between the application loop and the force-feedback servo loop. The technique uses a simulated spring to connect the probe endpoint to the appropriate body in the simulation, as shown in Figure 4. The user experiences forces which are both reasonable and stable.

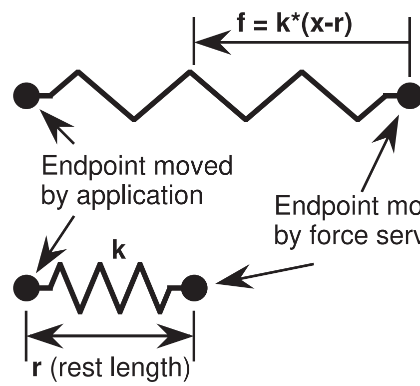

Figure 4: *Point-to-point spring, which couples one point moved by the application (at ~1 Hz) to another moved by the force server (at ~1 kHz). The spring constant is* ***k****.*

The method is based on that used in [21] for mouse-based interaction. Another member of our lab, Yunshan Zhu, implemented it for a force-feedback device using two asychronous loops on the graphics host. That success led us to integrate the technique into Armlib's force server.

In this method, the application controls the motion of one endpoint of the spring at its slower update rate, while the other endpoint follows the probe motion at the force update rate. The spring applies force both to the probe (pulling the user's hand towards the point of contact) and to the application (typically adding forces into the simulation). Adjustment of the spring constant controls the tightness of the coupling between application and probe; a weaker spring produces small forces in the application while a tighter spring causes more discontinuity in the force when the application endpoint moves.

In order to prevent the user from moving the probe too rapidly, we may in the future add adjustable viscosity to the force-server loop. Viscosity would tend to keep the probe from moving large distances (and thus adding large forces) between simulation time steps.

Multiple springs

Using a single point of contact between the application and the force server, it is only possible to specify forces, not torques. This restriction is overcome by attaching springs to multiple application points and multiple virtual probes. Acting together, multiple springs can specify a force and general torque on the probe and the application model (see Figure 5).

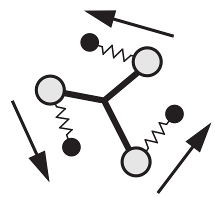

Figure 5: *Torque from multiple springs.*

2.2 Preventing force discontinuity artifacts

As pointed out in [1], the plane-and-probe model works well only when the plane equation is updated frequently compared to the lateral speed of probe motion. As shown in Figure 6, this restriction is most severe on sharply-curving surfaces. A sharp discontinuity occurs in the force model when the probe is allowed to move large distances before the new surface approximation is computed. If the discontinuity leaves the probe outside the surface, the probe drops suddenly onto the new level. Worse, if the probe is embedded in the new surface, it is violently accelerated until it leaves the surface (and sometimes the user's hand).

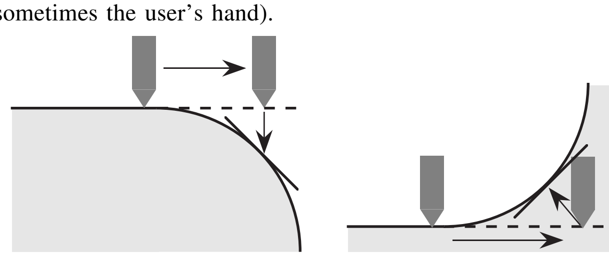

Figure 6: *Probe motion that is rapid compared to the surface curvature causes a sharp discontinuity when the new plane equation arrives. Case **a** shows the free-fall that occurs for convex surfaces. The more severe case **b** shows the sudden force caused by being deeply embedded in the surface.*

To solve the problem of extreme forces when the probe is embedded in the new surface, we have developed a *recovery time* method. This method is applied during the time immediately after new surface parameters arrive. If the probe is outside the surface at the time the parameters change, the system works as described above, dropping suddenly to the surface. If the probe is within the surface, then the normal direction for the force remains as above but the force magnitude is reduced so as to bring the tip out of the surface over a period of time, rather than instantaneously. This period of time is adjustable, and serves to move the probe out of the surface gently, while still maintaining proper direction for the force at all times. Figure 7 illustrates this algorithm.

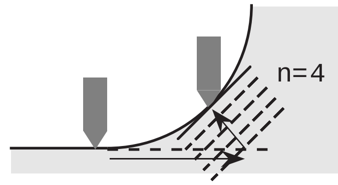

Figure 7: *When a new plane equation would cause the probe to be embedded in the surface, the recovery time algorithm artificially lowers the plane to the probe position then raises it linearly to the correct position over **n** force loop cycles.*

This method allows the presentation of much stiffer-feeling surfaces (higher spring constant) without noticeable discontinuities. By using recovery times of up to 0.05 second, the Nanomanipulator application was able to increase the surface spring constant by a factor of 10.

A recovery-time algorithm is also required in the point-to-point spring model. When the only adjustable parameter is the spring constant, there is a trade-off between how tightly the probe is tied to the application endpoint (higher **k** is better) and how smooth the transition is when the application moves its endpoint (lower **k** is better). We avoid this tradeoff by allowing the application to specify the rate of motion for its endpoint after an endpoint update. When the application sets a new position for its endpoint (or a new rest length for the spring), the point smoothly moves from its current location to the new location over the specified number of server loop iterations.

2.3 Flexibility and extensibility

Our force-feedback software has evolved from application-specific device-driver routines [4], through a device-specific but application-independent library controlling our Argonne-III Remote Manipulator, to the current device-independent remote-access library, Armlib.

Armlib provides connectivity to widely-used graphics engines (SGI, HP and Sun workstations) over commonly-used networks (Ethernet and other TCP/IP). It supports commercially-available force displays (several varieties of SensAble Devices PHANToM [14], and Sarcos Research Corporation Dexterous Master), as well as our Argonne-III Remote Manipulator from Argonne National Laboratories. Armlib supports the simultaneous use of multiple force-feedback devices, for multi-user or multi-hand applications. The application selects the device(s) it needs to use at runtime.

Armlib structure

Armlib provides device independence at the API level by using a cartesian coordinate system with an origin at the center of the device's working volume. Forces and positions can be automatically scaled so that software will work unchanged with devices of different sizes.

The device independence extends to Armlib's internal structure (Figure 8). Device-dependencies are compartmentalized in a set of simple low-level "device-driver" routines, which handle the reading of joint positions, the writing of joint forces, and the serializing of the robot link configuration. Higher levels of the library, including the intermediate representation servo loops, function in cartesian space. The conversion from joint space to cartesian space and back is handled by a common set of routines which utilize a Denavit-Hartenberg based description of each device to compute the forward kinematics and Jacobian matrix at runtime (see e.g. [9]). These routines effectively discard most torque information for three DOF devices such as the standard PHANToM.

The compartmentalization of device dependencies facilitates the addition of both new device types and new library capabilities. Because code for intermediate representations uses only cartesian space, this code works automatically for all devices. The ease of making changes is illustrated by the fact that it took only two days to add support for the PHANToM device to our library, and less than two days to add the code for our spring-based intermediate representation.

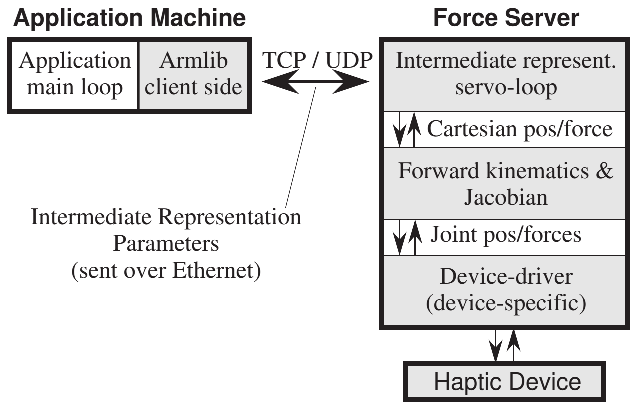

Figure 8: *Armlib's structure. Intermediate representation parameters pass between the client (application & API) and the force server over the Ethernet. The intermediate representation servo loop functions entirely in cartesian space. Device-dependencies are contained in the joint-space device-driver.*

Client/Server communications

There are two types of information passed between the application and the force server. Commands affecting system state (starting, stopping, initiating local force computation) must be delivered intact and not lost. In contrast, position reports and updates to intermediate representation parameters are sent frequently, so a lost packet can be ignored since a new one will arrive shortly. (In fact, ignoring these lost packets is the correct approach; retransmission is time consuming). We decided to use two channels between the client and server, the *command* and *data* channels. We currently use a TCP stream connection for the command channel (reliable, high overhead) and use UDP datagrams (unreliable, low overhead) for the data channel. Our client-server communications routines are well-compartmentalized, so the substitution of different protocols would be simple.

Armlib provides an asychronous *continual report* mode, in which the server sends position reports at regular intervals (using the data channel), rather than on request. This mode avoids the wait for a round-trip network message which is required by standard requests. The application can poll for these continual reports or block for them. Armlib also provides the application with a file descriptor indicating report arrival which can be `select()`'d by event-driven applications such as those written under X-Windows.

Performance

All of the intermediate representation features (plane-vs.-probe, multiple probes, recovery time, friction, and point-to-point springs) are orthogonal, and can be used singly or in combination. These tools produce a rich interaction environment at high update rates. We achieve 1 kHz on a 133 MHz Pentium processor for our custom six sense-DOF, three force-DOF PHANToM. The rate is even higher on a standard PHANToM. When we use a recovery time with our plane-and-probe model, we achieve stable hard surface stiffnesses of 2100 N/m on our custom PHANToM.

3. RESULTS AND SIGNIFICANCE

We have presented a system-based approach to solving the problems encountered when integrating force feedback into real-time computer graphics applications. Armlib combines and extends earlier work in intermediate representation of surfaces and in surface friction. It:

- Extends the intermediate representation of [1] by adding multiple surfaces.
- Introduces point-to-point springs as a form of intermediate representation on a force server.
- Extends previous work on surface friction by adding to the friction model the capability to produce simple haptic textures at high update rates.

Armlib also provides new functions and features. It:

- Presents a recovery-time algorithm to reduce sudden forces due to changes in the intermediate representation, both for local planes and point-to-point springs.
- Provides device independence, a simple interface, and easy extensibility through a compartmentalized and multi-layered design.
- Provides a fully-functional system running on readily-available networks and commercial hardware. Several groups outside our lab are already using our system.

4. FUTURE DIRECTIONS

The development of Armlib is driven by the needs of particular applications in our lab; we add capabilities as they become necessary. Use of existing library features to explore new areas is already underway. The Nanomanipulator project is working to adjust the friction parameters based on characteristics of the surface under the microscope.

One useful intermediate representation we would like to add is a 3-D linear approximation to the nearby force field; i.e. a first order Taylor series expansion of the force field about the most recent position. Such a representation would be useful for smoothly varying force fields.

Some applications might benefit from an enhancement of our plane-and-probe model to allow for half-planes, or even general convex planar polygons. This capability would allow several plane-and-probe constraints to be used simultaneously at convex points of intersection, such as the *outside* of a box. But this approach can also produce problems of its own; Zilles and Salisbury [24] provide a good discussion of some of these issues and discuss a technique to attack them.

It might be worthwhile in some applications to support simple curved surfaces as an intermediate representation type.

The implementation of the force servo loop for our plane and probe model is relatively simple. Work by Colgate and Brown [6] provides guidance on how to do better in attacking this "virtual wall" problem. Doing so would require providing our library with information about the dynamic behavior of each supported force-feedback device. We would also like to add a braking pulse like that described by Salcudean and Vlaar [18] to our virtual wall.

Armlib works with very simple intermediate representations. There is a continual temptation to add progressively more complex intermediate representations and associated calculations. There is of course a tradeoff in doing so—more complex representations take longer to evaluate, thus reducing the force update rate. A possible solution is to add another layer to our system. Such a layer might be in charge of object-level contacts and dynamics, and the calculation of the plane equations for our intermediate representation. It could address the fact that under some circumstances multiple simultaneous contacts should not be treated independently [7]. This layer would still run faster than the application main loop, but would be more complex (and thus slower) than the force server's intermediate representation servo loop.

5. AVAILABILITY

The latest information on Armlib is available from our haptics research web page, http://www.cs.unc.edu/Research/graphics/force. The Armlib source code and documentation are available by FTP at ftp://ftp.cs.unc.edu/pub/packages/GRIP/armlib/. A SIGGRAPH course this year [2] presents some additional tutorial information about Armlib.

ACKNOWLEDGEMENTS

Support for this work was provided by grant number RR02170 from the National Institutes of Health National Center for Research Resources. Our SARCOS arm was provided by DARPA. The Argonne Remote Manipulator is on loan from Argonne National Laboratories.

We would like to thank other contributors to our work. Frederick P. Brooks, Jr. and William V. Wright, the investigators for our NIH grant, provided support and ideas. Other students, in particular Yunshan Zhu, Kimberly Passarella-Jones, and Chris Dimattia contributed to Armlib and the ideas presented here. John Hughes attended to our force-feedback hardware. Finally, the anonymous reviewers (and one in particular) made some very helpful suggestions.

REFERENCES

[1] ADACHI, Y., KUMANO, T., OGINO, K. Intermediate Representation for Stiff Virtual Objects. *Proc. IEEE Virtual Reality Annual Intl. Symposium '95* (Research Triangle Park, N. Carolina, March 11-15), pp. 203-210.

[2] BAILEY, M., JOHNSON, D., KRAMER, J. MASSIE, T., TAYLOR, R. So Real I Can Almost Touch It: The Use of Touch as an I/O Device for Graphics and Visualization. *SIGGRAPH 96 Course Notes #37* (New Orleans, Louisiana, August 1996).

[3] BATTER, J. J., BROOKS, F. P. JR. GROPE-I: A computer display to the sense of feel. *Proc. Intl. Federation of Information Processing Congress '71* (Ljubljana, Yugosolavia, Aug. 23-28). *Information Processing '71*, vol. 1, pp. 759-763.

[4] BROOKS, F. P. JR., OUH-YOUNG, M., BATTER, J. J., KILPATRICK, P. J. Project GROPE—Haptic displays for scientific visualization. *Proc. SIGGRAPH 90* (Dallas, Texas, Aug. 6-10, 1990). In Computer Graphics 24, 4 (August 1990), pp. 177-185.

[5] BUTTOLO, P., KUNG, D., HANNAFORD, B. Manipulation in Real, Virtual and Remote Environments. *Proc. IEEE Conf. on Systems, Man and Cybernetics* (Vancouver, BC, Oct. 1995), vol. 5, pp. 4656-4661.

[6] COLGATE, J. E., BROWN, J. M. Factors Affecting the Z-Width of a Haptic Display. *Proc. IEEE Intl. Conf. on Robotics and Automation* (San Diego, Calif., May 8-13, 1994), vol. 4, pp. 3205-3210.

[7] COLGATE, J. E., STANLEY, M. C., BROWN, J. M. Issues in the Haptic Display of Tool Use. ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1994, In *Dynamic Systems and Control 1994* (Chicago, Illinois, Nov. 6-11), vol. 1, pp. 140-144.

[8] FINCH, M., CHI, V., TAYLOR, R. M. II, FALVO, M., WASHBURN, S., SUPERFINE, R. Surface Modification Tools in a Virtual Environment Interface to a Scanning Probe Microscope. *Proc. 1995 Symposium on Interactive 3D Graphics* (Monterey, CA, April 9-12, 1995), pp. 13-18.

[9] FU, K. S., GONZALEZ R. C., LEE, C. S. G. *Robotics control, sensing, vision and intelligence*. McGraw-Hill, New York, 1987.

[10] GOMEZ, D., BURDEA, G., LANGRANA, N. Integration of the Rutgers Master II in a Virtual Reality Simulation. *Proc. IEEE Virtual Reality Annual Intl. Symposium '95* (Research Triangle Park, N. Carolina, March 11-15), pp. 199-202.

[11] GOSSWEILER, R., LONG, C., KOGA, S., PAUSCH, R. DIVER: A Distributed Virtual Environment Research Platform. *Proc. IEEE 1993 Symposium on Research Frontiers in Virtual Reality* (San Jose, Calif., Oct. 25-26, 1993), pp. 10-15.

[12] KIM, W. S., HANNAFORD, B., BEJCZY, A. K. Force-Reflection and Shared Compliant Control in Operating Telemanipulators with Time Delay. *IEEE Transactions on Robotics and Automation*, April 1992, pp. 176-185.

[13] MARK, W. R., RANDOLPH, S. C., FINCH, M., VAN VERTH, J. M. UNC-CH Force-Feedback Library, Revision C. University of North Carolina at Chapel Hill, Computer Science Technical Report #TR96-012, Jan. 30, 1996. [Available at http://www.cs.unc.edu]; Also: *ibid*, Revision C.2. May 10, 1996. [Not a TR. Available at ftp://ftp.cs.unc.edu/pub/packages/GRIP/armlib]

[14] MASSIE, T. M., SALISBURY, J. K. The PHANToM Haptic Interface: A Device for Probing Virtual Objects. ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1994, In *Dynamic Systems and Control 1994* (Chicago, Illinois, Nov. 6-11), vol. 1, pp. 295-301.

[15] MINSKY M., OUH-YOUNG, M., STEELE, M., BROOKS, F. P. JR., BEHENSKY, M. Feeling and Seeing: Issues in Force Display. *Proc. 1990 Symposium on Interactive 3D Graphics* (Snowbird, Utah, March 25-28, 1990). In Computer Graphics 24, 2, pp 235-243.

[16] MITSUISHI, M., HORI, T., HATAMURA, Y., NAGAO, T., KRAMER, B. Operational environment transmission for manufacturing globalization. *Proc. 1994 Japan-U.S.A. Symposium on Flexible Automation* (Kobe, Japan, July 11-18, 1994), vol. 1, pp. 379-382.

[17] OUH-YOUNG, M., *Force Display In Molecular Docking*. Ph. D. Dissertation, University of North Carolina at Chapel Hill, UNC-CH Computer Science TR90-004, February, 1990.

[18] SALCUDEAN, S. E., VLAAR, T. D. On the Emulation of Stiff Walls and Static Friction with a Magnetically Levitated Input/Output Device. ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1994, In *Dynamic Systems and Control 1994* (Chicago, Illinois, Nov. 6-11), vol. 1, pp. 303-309.

[19] SALISBURY, K., BROCK, D., MASSIE, T., SWARUP, N., ZILLES, C. Haptic Rendering: Programming Touch Interaction with Virtual Objects. *Proc. 1995 Symposium on Interactive 3D Graphics* (April 9-12, Monterey, Calif.), pp. 123-130.

[20] SHAW, C., LIANG, J, GREEN, M., SUN, Y. The decoupled simulation model for VR systems. *Proc. 1992 Conf. on Human Factors in Computer Systems (CHI '92)* (Monterey, Calif., May 3-7, 1992), pp. 321-328.

[21] SURLES, M. C. An Algorithm With Linear Complexity For Interactive, Physically-based Modeling of Large Proteins. *Proc. SIGGRAPH 92* (Chicago, Illinois, July 26-31, 1992). In Computer Graphics, 26, 2 (July 1992), pp. 221-230.

[22] SHERIDAN, T. B. *Telerobotics, Automation, and Supervisory Control.* MIT Press, Cambridge, Mass., 1992.

[23] SNYDER, W. E. *Industrial Robots: Computer Interfacing and Control*. Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[24] ZILLES, C. B., SALISBURY, J. K. A Constraint-based God-object Method for Haptic Display. ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1994, In *Dynamic Systems and Control 1994* (Chicago, Illinois, Nov. 6-11), vol. 1, pp. 146-150.