(cont'd)

PhantomDefinePlane(*xPnt, yPnt, zPnt, xNorm, yNorm, zNorm, stiffness*)
PhantomDefineImgTx(*xRes, yRes, xDist, yDist, *imagefile*)
PhantomDefineSphere(*xCenter, yCenter, zCenter, xRad, yRad, zRad, stiffness*)
PhantomDefineButton(*xMin, xMax, yMin, yMax, zMin, zMax, stiffness, activeside*)
PhantomDeletePlane(*planeID*)
PhantomDeleteSphere(*sphereID*)
PhantomDeleteButton(*buttID*)
PhantomActivatePlane(*planeID*)
PhantomDeactivatePlane(*planeID*)
PhantomActivateSphere(*sphereID*)
PhantomDeactivateSphere(*sphereID*)
PhantomActivateButton(*buttID*)
PhantomDeactivateButton(*buttID*)
PhantomTranslateObject(*objID, xTrans, yTrans, zTrans*)
PhantomGetButtonHit()
PhantomNumButtonHits()
PhantomSetPlaneFriction(*planeID, cs, ck*)
PhantomSetPlaneDamping(*planeID, b*)
PhantomSetSphereFriction(*sphereID, cs, ck*)
PhantomSetSphereDamping(*sphereID, b*)
PhantomSetPlaneTexture(*planeID, amp, freq, texid*)
PhantomSetSphereTexture(*sphereID, amp, freq, texid*)
PhantomSetViscosity(*b*)
PhantomSetDriftForce(*fx, fy, fz*)
PhantomReInitialize()
PhantomForcesOn()
PhantomForcesOff()

Acknowledgments

This work was primarily sponsored by the Honda R&D Corporation. Much of the work was performed at the MIT Media Lab and at Interval Research Corporation, where one of the authors, W. Plesniak, was sponsored as an Interval Fellow. Equipment support was provided by the Intel Corporation.

Damping within surfaces can also be computed according to the user-specified damping coefficient, b. We simply record five most recent points along the stylus trajectory within a surface and use them to compute an approximate velocity, $\mathbf{v}_{avg}$. From this information, the damping force, $\mathbf{F}_d = b\mathbf{v}_{avg}$, is computed and added to $\mathbf{F}_n$ and $\mathbf{F}_f$, to form the total force that a user is exerting on an object.

Texture simulation

We currently specify object textures in two ways, either algorithmically, or by using a greyscale image (a 2D array of pixels). The texture simulation procedes in the following manner: if, upon polling, the stylus is found to be in contact with a texture-mapped surface, then either a function is evaluated or a texture map is indexed to indicate the surface relief at the current sample position. A displacement value ranging from {0 - texture scale} is added to the current position which permits the stylus to continue to sink into the surface unperturbed for that offset distance. Both texturing methods are outlined below.

Algorithmically-defined textures:

If sample point collides with textured object:

map currently-polled sample (x1, y1, z1) to (u, v, z2);
evaluate texture function at (u, v) and four surrounding points;
vector cross product to approximate new normal (ntx, nty, ntz);
inverse-map (u, v, z2+offset) back to (x1', y1', z1');
replace original normal with (ntx, nty, ntz);
compute forces with new position and normal..

Image-defined textures:

If sample point collides with textured object:

map currently-polled sample (x1, y1, z1) to (u, v, z2);
index texture map at points around sample point;
interpolation yields a height, offset, at (u, v);
approximate new normal (ntx, nty, ntz);
inverse-map (u, v, z2+offset) and (ntx, nty, ntz);
compute forces with new position and normal.

Recent and future work

The server design reported above does not permit reliable network-based simulation of inter-object dynamics and thus does not permit the user to rearrange elements in the scene by physically interacting with them. We are currently redesigning the system to provide a more complete simulation of object dynamics (to be displayed both visually and haptically.) The revised design permits objects to be created and to interact with each other and with the user. Objects like spheres, planes, cubes and other obstacles can be tagged as fixed or moveable, and this status can be changed as the application runs. During the inter-object collision detection stage, the bounding sphere of each object tagged as moveable is checked for intersection with those of other objects. This operation requires $(N^2 - N)/2$ simple compares per loop, where N is the number of movement-tagged objects. Objects which may be exerting force on other objects in the scene are tagged for update, and the current force acting on each tagged object is computed and recorded.

Finally, the objects that are tagged for update have their states recomputed, and the graphics host is notified of the new information at a rate of approximately 30 updates/sec (not every control loop cycle). This permits us to keep the haptics loop as tight as possible so that the simulation runs smoothly, and allows us to update the graphics loop at the approximate limit of graphics performance.

Haptics Library: summary

The SPI haptics library uses these models and others to provide a flexible platform for quickly mocking up haptic simulations using the PHANToM haptic interface, that accompany computer graphic stereo display of the same objects. The complexity of computer graphic rendering, handled by the client, does not interfere with server control loop speed; thus visual simulations rendered by the client can be arbitrarily elaborate. The calls currently available to client applications are listed below:

Available Client Calls
PhantomPollPos(*&x, &y, &z, button*) PhantomGetForce(*&fx, &fy, &fz*) PhantomSetNumReadAttempts(*n*) PhantomOpen() PhantomClose() PhantomSetVolumeLimits(*xMin, xMax, yMin yMax, zMin, zMax*) PhantomGetVolumeLimits(*&xMin, &xMax, &yMin &yMax, &zMin, &zMax*)

MIT Artificial Intelligence Laboratory Technical Report AITR-1596, and *MIT Research Laboratory for Electronics Technical Report No. 612*. Published in the *Proceedings of the First Phantom Users Group Workshop*, 1996.

Within each cycle, the location of the stylus is reported, and its collision with all objects is determined. If the stylus is in contact with any object, the force in the direction of the object surface normal is computed as well as forces from specified surface properties (friction, damping, texture). Force contributions from all objects are added to global drift or damping forces, and the result is displayed to the user. Object state, as shown above, is updated as necessary. Any requests from the client are then addressed before position polling repeats.

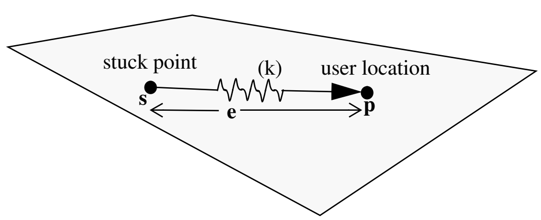
There are obviously many ways to model the surface and bulk properties of materials, and the computation of forces generated when we interact with them. Because the only directly measured state provided by the PHANToM is position, indirect methods must be used for computing physical quantities involving velocity and acceleration. Below, we provide a brief description of some simple modeling which has yielded satisfying simulation of some haptic phenomena.

Friction and damping modeling

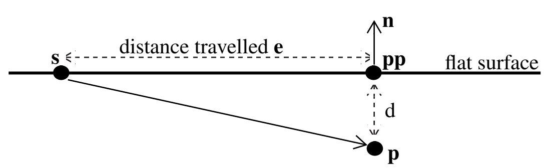
Frictional force, $\mathbf{F}_f$, can be approximated as follows; a “stuck” point, **s**, at which a user first contacts a surface, or the point at which velocity along the surface becomes zero, is recorded. We model a second surface (with stiffness k) at this point, oriented so that its normal opposes the user’s direction of motion. Static friction is modeled as the resisting force required to “break through” this second surface, before being able to continue motion across the real surface. After sufficient force has been applied to overcome static friction, movement is permitted and **s** is dragged along. Then, kinetic friction is modeled as a smaller opposing force whose magnitude depends on the amount of normal force, $\mathbf{F}_n$, exerted by the user.

Thus, if $k\mathbf{e} < (c_s\,|\mathbf{F}_n|)$, then $\mathbf{F}_f = k\mathbf{e}\mathbf{f}_t$; else $\mathbf{F}_f = (c_k\,|\mathbf{F}_n|)\,\mathbf{f}_t$, where c_s is the coefficient of static friction, c_k is the coefficient of kinetic friction, and $\mathbf{f}_t$ is the unit vector in the direction of the tangential component of force. The stiffness coefficient, k, of the “resisting” surface is currently not user-specified. Computation of the distance travelled, **e**, along a flat surface, textured surface, or sphere is shown in the diagram below.

Friction modeling on a surface



Computing the distance travelled on flat surfaces

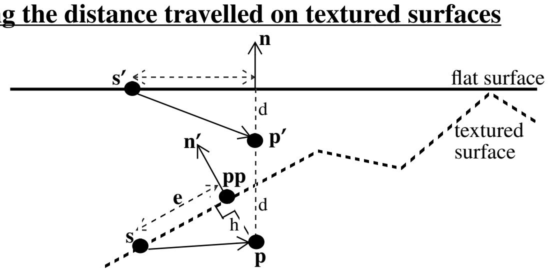


$d = (\mathbf{p} - \mathbf{s}) \cdot \mathbf{n}$

$\mathbf{pp} = \mathbf{p} - d\mathbf{n}$

$|\mathbf{e}|$ = distance travelled in surface = $|\mathbf{pp} - \mathbf{s}|$

Computing the distance travelled on textured surfaces

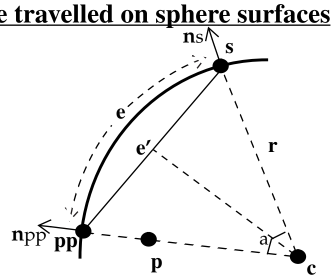


$h = d(\mathbf{n} \cdot \mathbf{n}')$

$\mathbf{pp} = \mathbf{p} - h\mathbf{n}'$

$|\mathbf{e}| = |\mathbf{pp} - \mathbf{s}|$

Computing the distance travelled on sphere surfaces



$\mathbf{e} = |\mathbf{r}|a$

$a = 2\text{asin}\{\ |\mathbf{e} - \mathbf{s}| / 2|\mathbf{r}|\ \}$

$|\mathbf{e}| = 2|\mathbf{r}|\ \text{asin}\{\ |\mathbf{e} - \mathbf{s}| / 2|\mathbf{r}|\ \}$

or, **e** can be approximated with **e′**:

$|\mathbf{e}'| = |\mathbf{pp} - \mathbf{s}|$

MIT Artificial Intelligence Laboratory Technical Report AITR-1596, and *MIT Research Laboratory for Electronics Technical Report No. 612*. Published in the *Proceedings of the First Phantom Users Group Workshop*, 1996.

SPI Haptics Library

Wendy Plesniak, John Underkoffler
MIT Media Laboratory
wjp@media.mit.edu jh@media.mit.edu

Abstract

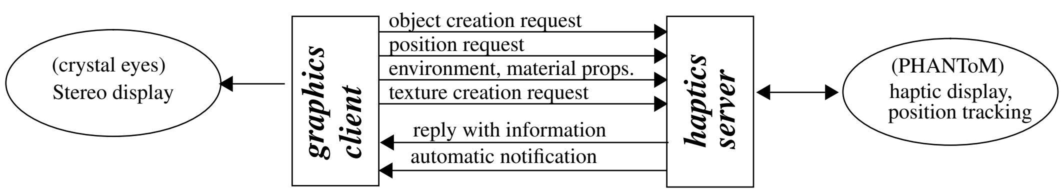
For haptics applications in which the PHANToM haptic interface is controlled by a PC, we have built a network-based client-server system so that high-quality computer graphic rendering can accompany haptic simulation. The system currently consists of a 586 (Pentium) PC to drive the PHANToM and an SGI Indigo Elan to render stereo computer graphics. The machines communicate over Ethernet using TCP/IP.

Position query and object creation commands are sent from the graphics host to the PC running the haptics simulation. Informational replies and automatic notifications are sent from the PC back to the graphics host. Thus the PC-based haptics library is made available to applications running on higher-end graphics workstations.

The haptics library supports the creation and deletion of a wide variety of primitive objects, textural and bulk material properties so that rendering a haptic scene is accomplished by making simple calls from a client program. Currently the system simulates static models, and dynamic simulations are being developed.

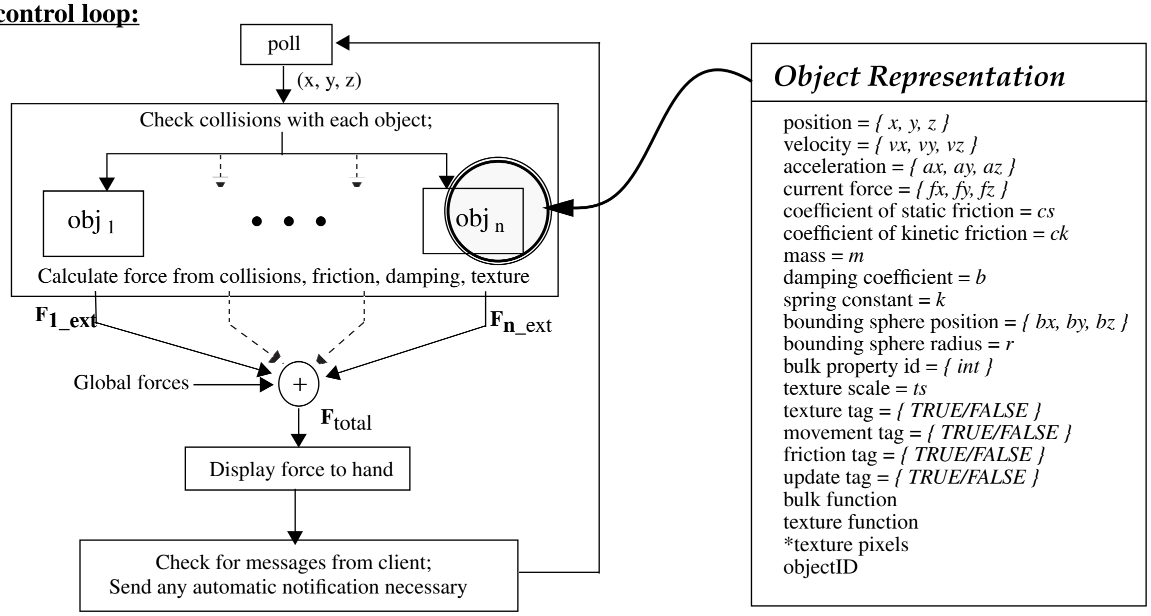
Haptics client-server

Our network-based client-server system has been developed for applications demanding both high quality haptic and visual rendering. The haptics server recognizes commands from a client graphics machine for object creation and deletion, activation and deactivation, setting global drift forces and global viscosity, specifying surface damping and friction (static and kinetic), creating textures on objects, and for position-query. In turn, the client receives confirmations of successfully-completed commands, requested position information, and automatic notifications when buttons tagged as "active" have been touched. This system, which models a world of static objects, has proven to be robust; the only noticeable visual latency occurs when network traffic is heavy. The basic system is shown below.



The haptic control loop, which accommodates a servo-loop of about 4000 pts/sec, is constructed as shown below:

Haptic control loop:



(cont'd)

PhantomDefinePlane(*xPnt, yPnt, zPnt, xNorm, yNorm, zNorm, stiffness*)
PhantomDefineImgTx(*xRes, yRes, xDist, yDist, *imagefile*)
PhantomDefineSphere(*xCenter, yCenter, zCenter, xRad, yRad, zRad, stiffness*)
PhantomDefineButton(*xMin, xMax, yMin, yMax, zMin, zMax, stiffness, activeside*)
PhantomDeletePlane(*planeID*)
PhantomDeleteSphere(*sphereID*)
PhantomDeleteButton(*buttID*)
PhantomActivatePlane(*planeID*)
PhantomDeactivatePlane(*planeID*)
PhantomActivateSphere(*sphereID*)
PhantomDeactivateSphere(*sphereID*)
PhantomActivateButton(*buttID*)
PhantomDeactivateButton(*buttID*)
PhantomTranslateObject(*objID, xTrans, yTrans, zTrans*)
PhantomGetButtonHit()
PhantomNumButtonHits()
PhantomSetPlaneFriction(*planeID, cs, ck*)
PhantomSetPlaneDamping(*planeID, b*)
PhantomSetSphereFriction(*sphereID, cs, ck*)
PhantomSetSphereDamping(*sphereID, b*)
PhantomSetPlaneTexture(*planeID, amp, freq, texid*)
PhantomSetSphereTexture(*sphereID, amp, freq, texid*)
PhantomSetViscosity(*b*)
PhantomSetDriftForce(*fx, fy, fz*)
PhantomReInitialize()
PhantomForcesOn()
PhantomForcesOff()

Acknowledgments

This work was primarily sponsored by the Honda R&D Corporation. Much of the work was performed at the MIT Media Lab and at Interval Research Corporation, where one of the authors, W. Plesniak, was sponsored as an Interval Fellow. Equipment support was provided by the Intel Corporation.

Damping within surfaces can also be computed according to the user-specified damping coefficient, b. We simply record five most recent points along the stylus trajectory within a surface and use them to compute an approximate velocity, $\mathbf{v}_{avg}$. From this information, the damping force, $\mathbf{F}_d = b\mathbf{v}_{avg}$, is computed and added to $\mathbf{F}_n$ and $\mathbf{F}_f$, to form the total force that a user is exerting on an object.

Texture simulation

We currently specify object textures in two ways, either algorithmically, or by using a greyscale image (a 2D array of pixels). The texture simulation procedes in the following manner: if, upon polling, the stylus is found to be in contact with a texture-mapped surface, then either a function is evaluated or a texture map is indexed to indicate the surface relief at the current sample position. A displacement value ranging from {0 - texture scale} is added to the current position which permits the stylus to continue to sink into the surface unperturbed for that offset distance. Both texturing methods are outlined below.

Algorithmically-defined textures:

If sample point collides with textured object:

map currently-polled sample (x1, y1, z1) to (u, v, z2);
evaluate texture function at (u, v) and four surrounding points;
vector cross product to approximate new normal (ntx, nty, ntz);
inverse-map (u, v, z2+offset) back to (x1', y1', z1');
replace original normal with (ntx, nty, ntz);
compute forces with new position and normal..

Image-defined textures:

If sample point collides with textured object:

map currently-polled sample (x1, y1, z1) to (u, v, z2);
index texture map at points around sample point;
interpolation yields a height, offset, at (u, v);
approximate new normal (ntx, nty, ntz);
inverse-map (u, v, z2+offset) and (ntx, nty, ntz);
compute forces with new position and normal.

Recent and future work

The server design reported above does not permit reliable network-based simulation of inter-object dynamics and thus does not permit the user to rearrange elements in the scene by physically interacting with them. We are currently redesigning the system to provide a more complete simulation of object dynamics (to be displayed both visually and haptically.) The revised design permits objects to be created and to interact with each other and with the user. Objects like spheres, planes, cubes and other obstacles can be tagged as fixed or moveable, and this status can be changed as the application runs. During the inter-object collision detection stage, the bounding sphere of each object tagged as moveable is checked for intersection with those of other objects. This operation requires $(N^2 - N)/2$ simple compares per loop, where N is the number of movement-tagged objects. Objects which may be exerting force on other objects in the scene are tagged for update, and the current force acting on each tagged object is computed and recorded.

Finally, the objects that are tagged for update have their states recomputed, and the graphics host is notified of the new information at a rate of approximately 30 updates/sec (not every control loop cycle). This permits us to keep the haptics loop as tight as possible so that the simulation runs smoothly, and allows us to update the graphics loop at the approximate limit of graphics performance.

Haptics Library: summary

The SPI haptics library uses these models and others to provide a flexible platform for quickly mocking up haptic simulations using the PHANToM haptic interface, that accompany computer graphic stereo display of the same objects. The complexity of computer graphic rendering, handled by the client, does not interfere with server control loop speed; thus visual simulations rendered by the client can be arbitrarily elaborate. The calls currently available to client applications are listed below:

Available Client Calls

PhantomPollPos(*&x, &y, &z, button*)
PhantomGetForce(*&fx, &fy, &fz*)
PhantomSetNumReadAttempts(*n*)
PhantomOpen()
PhantomClose()
PhantomSetVolumeLimits(*xMin, xMax, yMin yMax, zMin, zMax*)
PhantomGetVolumeLimits(*&xMin, &xMax, &yMin &yMax, &zMin, &zMax*)

MIT Artificial Intelligence Laboratory Technical Report AITR-1596, and *MIT Research Laboratory for Electronics Technical Report No. 612*. Published in the *Proceedings of the First Phantom Users Group Workshop*, 1996.

Within each cycle, the location of the stylus is reported, and its collision with all objects is determined. If the stylus is in contact with any object, the force in the direction of the object surface normal is computed as well as forces from specified surface properties (friction, damping, texture). Force contributions from all objects are added to global drift or damping forces, and the result is displayed to the user. Object state, as shown above, is updated as necessary. Any requests from the client are then addressed before position polling repeats.

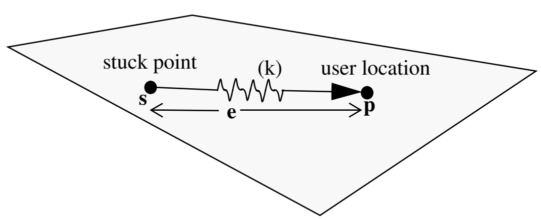
There are obviously many ways to model the surface and bulk properties of materials, and the computation of forces generated when we interact with them. Because the only directly measured state provided by the PHANToM is position, indirect methods must be used for computing physical quantities involving velocity and acceleration. Below, we provide a brief description of some simple modeling which has yielded satisfying simulation of some haptic phenomena.

Friction and damping modeling

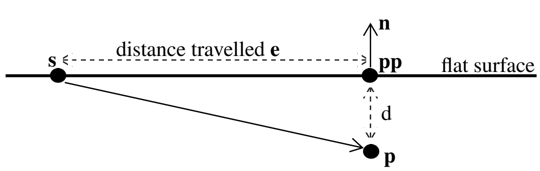
Frictional force, $\mathbf{F}_f$, can be approximated as follows; a "stuck" point, **s**, at which a user first contacts a surface, or the point at which velocity along the surface becomes zero, is recorded. We model a second surface (with stiffness k) at this point, oriented so that its normal opposes the user's direction of motion. Static friction is modeled as the resisting force required to "break through" this second surface, before being able to continue motion across the real surface. After sufficient force has been applied to overcome static friction, movement is permitted and **s** is dragged along. Then, kinetic friction is modeled as a smaller opposing force whose magnitude depends on the amount of normal force, $\mathbf{F}_n$, exerted by the user.

Thus, if $k\mathbf{e} < (c_s\,|\mathbf{F}_n|)$, then $\mathbf{F}_f = k\mathbf{e}\mathbf{f}_t$; else $\mathbf{F}_f = (c_k\,|\mathbf{F}_n|)\,\mathbf{f}_t$, where c_s is the coefficient of static friction, c_k is the coefficient of kinetic friction, and $\mathbf{f}_t$ is the unit vector in the direction of the tangential component of force. The stiffness coefficient, k, of the "resisting" surface is currently not user-specified. Computation of the distance travelled, **e**, along a flat surface, textured surface, or sphere is shown in the diagram below.

Friction modeling on a surface



Computing the distance travelled on flat surfaces

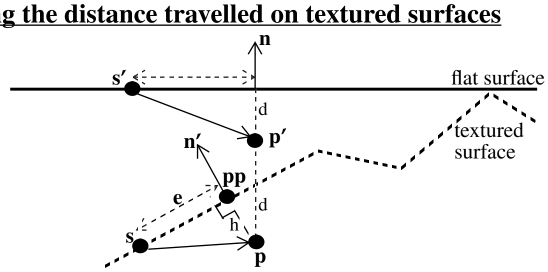


$d = (\mathbf{p} - \mathbf{s})\,.\,\mathbf{n}$

$\mathbf{pp} = \mathbf{p} - d\mathbf{n}$

$|\mathbf{e}|$ = distance travelled in surface = $|\mathbf{pp} - \mathbf{s}|$

Computing the distance travelled on textured surfaces

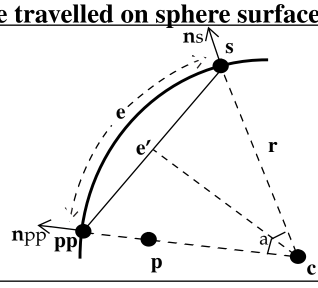


$h = d(\mathbf{n}\,.\,\mathbf{n}')$

$\mathbf{pp} = \mathbf{p} - h\mathbf{n}'$

$|\mathbf{e}| = |\mathbf{pp} - \mathbf{s}|$

Computing the distance travelled on sphere surfaces



$\mathbf{e} = |\mathbf{r}|a$

$a = 2\mathrm{asin}\{\ |\mathbf{e} - \mathbf{s}|\ /\ 2|\mathbf{r}|\ \}$

$|\mathbf{e}| = 2|\mathbf{r}|\ \mathrm{asin}\{\ |\mathbf{e} - \mathbf{s}|\ /\ 2|\mathbf{r}|\ \}$

or, **e** can be approximated with **e′**:

$|\mathbf{e}'| = |\mathbf{pp} - \mathbf{s}|$

MIT Artificial Intelligence Laboratory Technical Report AITR-1596, and *MIT Research Laboratory for Electronics Technical Report No. 612*. Published in the *Proceedings of the First Phantom Users Group Workshop*, 1996.

SPI Haptics Library

Wendy Plesniak, John Underkoffler
MIT Media Laboratory
wjp@media.mit.edu jh@media.mit.edu

Abstract

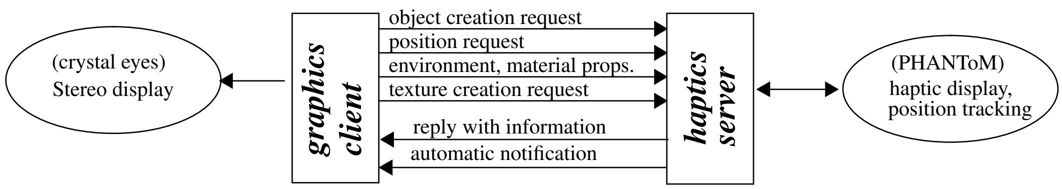
For haptics applications in which the PHANToM haptic interface is controlled by a PC, we have built a network-based client-server system so that high-quality computer graphic rendering can accompany haptic simulation. The system currently consists of a 586 (Pentium) PC to drive the PHANToM and an SGI Indigo Elan to render stereo computer graphics. The machines communicate over Ethernet using TCP/IP.

Position query and object creation commands are sent from the graphics host to the PC running the haptics simulation. Informational replies and automatic notifications are sent from the PC back to the graphics host. Thus the PC-based haptics library is made available to applications running on higher-end graphics workstations.

The haptics library supports the creation and deletion of a wide variety of primitive objects, textural and bulk material properties so that rendering a haptic scene is accomplished by making simple calls from a client program. Currently the system simulates static models, and dynamic simulations are being developed.

Haptics client-server

Our network-based client-server system has been developed for applications demanding both high quality haptic and visual rendering. The haptics server recognizes commands from a client graphics machine for object creation and deletion, activation and deactivation, setting global drift forces and global viscosity, specifying surface damping and friction (static and kinetic), creating textures on objects, and for position-query. In turn, the client receives confirmations of successfully-completed commands, requested position information, and automatic notifications when buttons tagged as "active" have been touched. This system, which models a world of static objects, has proven to be robust; the only noticeable visual latency occurs when network traffic is heavy. The basic system is shown below.



The haptic control loop, which accommodates a servo-loop of about 4000 pts/sec, is constructed as shown below:

Haptic control loop: