Defining a control standard for easily integrating haptic virtual environments with existing audio/visual systems

Stephen Sinclair
IDMIL, CIRMMT, McGill University
sinclair@music.mcgill.ca

Marcelo M. Wanderley
IDMIL, CIRMMT, McGill University
marcelo.wanderley@mcgill.ca

ABSTRACT

This paper presents an approach to audio-haptic integration that utilizes OpenSound Control, an increasingly well-supported standard for audio communication, to initialize and communicate with dynamic virtual environments that work with standard force-feedback devices.

Keywords

Haptics, control, multi-modal, audio, force-feedback

1. INTRODUCTION

Audio and video systems have historically been very well integrated. In contrast, haptic displays are only beginning to be available to a wider audience. As the number of haptic controllers on the market has been increasing, there has been a corresponding interest in making use of haptics as a third sensory mode in audio/visual systems. While high-fidelity systems are often used in research for determining the limits of our haptic senses, there is also a need to introduce better integration of lower-cost haptics into existing multimedia software.

Typically, creating an experimental setup for haptics research entails programming a 3D environment using some C or C++ framework. While this is powerful, it can be unnecessarily complex for many simple needs. Some tools make this easier by allowing the user to specify the environment in a description language such as VRML [6] [17]. While this approach certainly simplifies things, it does not integrate well with systems commonly used in the audio domain. Consequently, audio-enabled haptic demos are usually limited to simple sound file playback[1]. Additionally, they are not dynamic systems that allow fast run-time prototyping. For the purpose of quickly creating interactive environments that can be easily modified, better integration with existing audio systems is called for.

Beginning with the idea of creating a "haptics" external for PureData [16], we came to the conclusion that it would be more fruitful to run a haptics simulation in a separate process, using OpenSound Control to communicate. This paper describes our reasoning behind this decision, our results to date, and some ideas we are considering for future work.

[1]See, for example, the proSENSE HapticMusic demo.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NIME07, New York, NY, USA
Copyright 2007 Copyright remains with the author(s).

2. MULTI-MODAL COMPUTING

A rendering system for a multi-modal display is often split into several asynchronous subsystems, either on separate hardware, or as operating system threads or processes. The reasons are several: asynchronous operation allows computationally demanding tasks to be performed in tandem, potentially on dedicated hardware; different sensory modes require different sets of calculations; finally, each sensory mode requires different update rates for a convincing simulation. While control changes should be apparent in an audio stream within 10 ms or less for a satisfying user experience [10], visual displays usually update at about 30 Hz, meaning that control changes are allowed up to 33 ms to be received and processed.

In haptics, the situation is more demanding: since input and output operations are directly coupled, the user becomes part of a closed system. The "display" depends entirely on the user's input movement, and reactions in control changes must be as instantaneous as possible in order to render the feel of a hard surface. It has been previously found that between a 500 Hz and 1 KHz update rate must be maintained for a good user experience [12]; the implication is that about 1 ms is allowed for input, data processing, and output. To achieve these rates, haptics computing usually requires either dedicated hardware or at least a dual-processor computer in which one processor is used exclusively for the haptics servoloop. Sometimes additional tiering is used to separate fast computations from more complex parts of the simulation [11].

2.1 The Synchronous Approach

This asynchrony has its drawbacks, which usually come in the form of latency. The alternative, synchronous approach is to use tightly integrated subsystems which operate by dividing down a master clock to the appropriate interval. For example, the Audio-Haptic Interface project [5] designed a microcontroller-based system using timer interrupts to compute haptics and audio under real-time constraints in order to achieve very low latency between the two subsystems. The result was a system with high fidelity, enabling the authors to measure the lower bound of noticeable latency in an audio-haptic system. Such a solution is specialized, requires

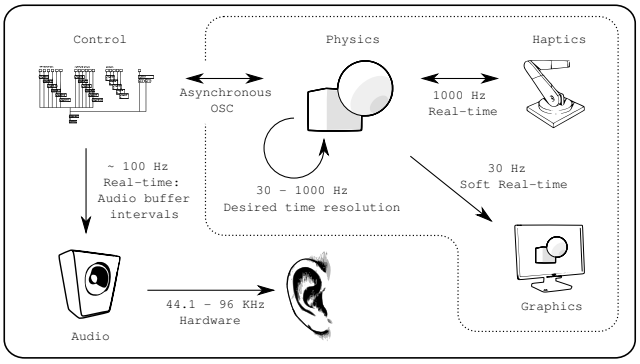

Figure 1: The system architecture. The dotted line represents a process boundary. Each subsystem within the simulation process is a separate thread.

embedded programming, and is not easily scalable to larger projects.

The work at ACROE [2] has also shown the possibilities of strictly synchronous systems. Originally, three sensory modes were synchronized using an external hardware clock. They eventually moved to an SGI system, and currently their ERGOS haptic device depends on algorithms running in a PCI-based DSP. The physical modelling for haptic and audio processes are still calculated synchronously on the DSP, though the graphical display is updated independantly by the PC. At least one project used a very accurate 3 KHz haptic model, with 30 KHz audio [9].

2.2 An Asynchronous Advantage

Multimedia software can can often get away with performing video operations synchronously between audio frames. PureData's GEM [4] is an example of one. Such an approach would be impossible for haptics, with its higher frequency throughput, without in-depth changes to the runtime system—it would require the addition of a whole other set of patch cords running messages at the haptic rate. In the end, it would not scale to multi-core processors or distributed systems.

With communication hardware getting faster and parallel processing becoming easily available, the flexibility promised by asynchrony becomes more attractive, as its deficiencies become less important. The recent advent of dual-core and quad-core processors is evidence that this is likely to continue [20].

Taking the hint, it seems no longer reasonable to try programming synchronous subsystems for other sensory modes. A better approach is to have independent processes, possibly running on different hardware, which communicate using a standard protocol. In the next section, we introduce an environment for haptics which can be controlled from PureData, but runs as a separate process, using OpenSound Control [24] to communicate. OSC was chosen because it has particularly good support in PureData and similar programs. A diagram of the system is shown in Figure 1.

3. IMPLEMENTATION

A proof-of-concept implementation has been created to test these concepts. A photograph of a user interacting with the system can be found in Figure 2. It takes advantage of

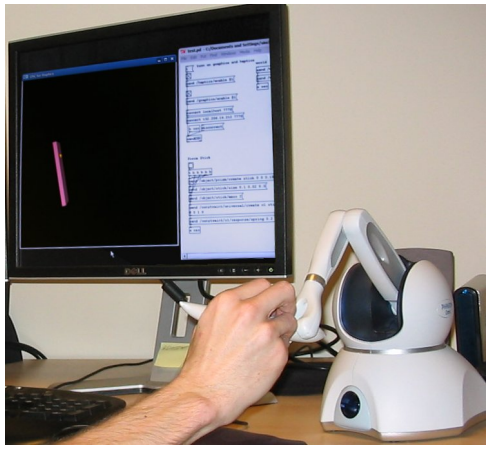

Figure 2: Using the Phantom Omni from SensAble Technologies to interact with the Force Stick example. The PureData patch that manages the scene is visible on the right-hand side of the screen.

some ready-made libraries of routines for haptics and physical dynamics, which are amazingly available as free, open-source software. The implementation is a combination of the CHAI 3D [3] haptic scene-graph API for calculating device-related forces and to communicate with the haptic display, and the Open Dynamics Engine (ODE) [19] for physical simulation of object movement and collision processing. CHAI 3D supports a number of off-the-shelf haptic devices. It also uses LibLo [7] for OSC messaging, and GLUT [8] for cross-platform graphical display, though graphics can be disabled if not needed.

A more complete specification of the proposed OSC namespace for simple objects and constraints is available [18]. The current software only responds to a subset of these messages, but work on a full implementation is currently in progress.

4. EXAMPLES

The 3D virtual environment can be specified in terms of *objects* and *constraints* on these objects. For receiving feedback from the model, any parameter of an object or constraint can be told to send itself at regular intervals. For now, only basic shapes are considered, but we found that many interesting *virtual musical instruments* (VMI) [13] can be constructed with them. Composite objects can be created by using a hierarchical naming convention, which works well in the OSC paradigm. For more complicated shapes, triangular meshes can be loaded. The specification for constraints is inspired heavily by the API for ODE.

The first example will give details of what kind of OSC namespace can be used to define a very simple instrument. A few extra examples of more complex instruments are described briefly.

4.1 Force Stick

Modeled on the original Force Stick described by Verplank [23], this simple VMI consists of a rectangular prism with one end on a rotating hinge. The hinge has varying types of active feedback. For instance, Verplank suggested several effects that can be achieved: *pluck*, *ring*, *rub*, *bang*, *strike*, and *squeeze*. Using OSC, such a system can be constructed

as follows:

Initialize the object (named "stick"), and specify its shape, position and mass:

/object/prism/create stick
/object/stick/size 0.02 0.1 0.3
/object/stick/position 0 0 0.15
/object/stick/mass 2

It can be seen here that once an object is created, it becomes part of the OSC namespace. It can then accept OSC methods which modify it. It is also immediately introduced into the simulation, appears on the screen, and can be touched and manipulated with the haptic device. Next, add a constraint (a hinge) located at the bottom of the prism, named "motor", with a damped spring response:

/constraint/hinge/create motor stick world 0 0 0 0 1 0
/constraint/motor/response/spring 20 1

The constraint is located at (0,0,0) and its axis points along (0,1,0), the X-axis. This is the axis around which the stick will rotate.

The constraint is defined to be between the object `stick`, and `world`, indicating a fixed position in space. The stiffness of the spring action is defined as 20 N·m/rad, and the damping coefficient is 1 N·m·s/rad. Henceforth, the object will not move in space except in rotation around the line segment defined by (0,0,0) and (0,1,0). Around this single axis, the "motor" will respond according to the given parameters.

To change the behaviour of the object when it is pushed or pulled by the device proxy, a different `response` message can be sent to the `motor` constraint. For instance, to get a *squeeze* type of response, a negative linear response can be used.

/constraint/motor/response/spring -10

This will reverse the usual spring response, so that the stick tends to fall away from the original location, and must be pulled back to the center. An object can be grabbed using the device's button.

To create a musical response, for example, by modifying the timbre or pitch of a synthesizer based on force, the following message will indicate that the system should send messages every 30 ms:

/constraint/motor/force/magnitude/get 30

This will cause the force exerted by the stick's constraint to be sent to the audio system at regular intervals. Conversely, the force exerted *on* the stick can be retrieved by,

/object/stick/force/magnitude/get 30

Generally speaking, any object property can be retrieved similarly, either one time or at regular intervals.

A *plucking* response can also be specified:

/constraint/motor/response/pluck 10 5

This will indicate that the constraint should resist the device when it reaches an angle of 10 degrees, with a stiffness of 5 N·m/rad. In other words, a weak rotational "virtual wall" will be created at the 10-degree angle on this hinge constraint. Since the proxy can push through it, the feeling of plucking is created by "letting go" as it passes a certain point beyond it. In this case, one might wish to trigger a wavetable or send an impulse to a physical model in response to each pluck:

/constraint/motor/response/pluck/get 1

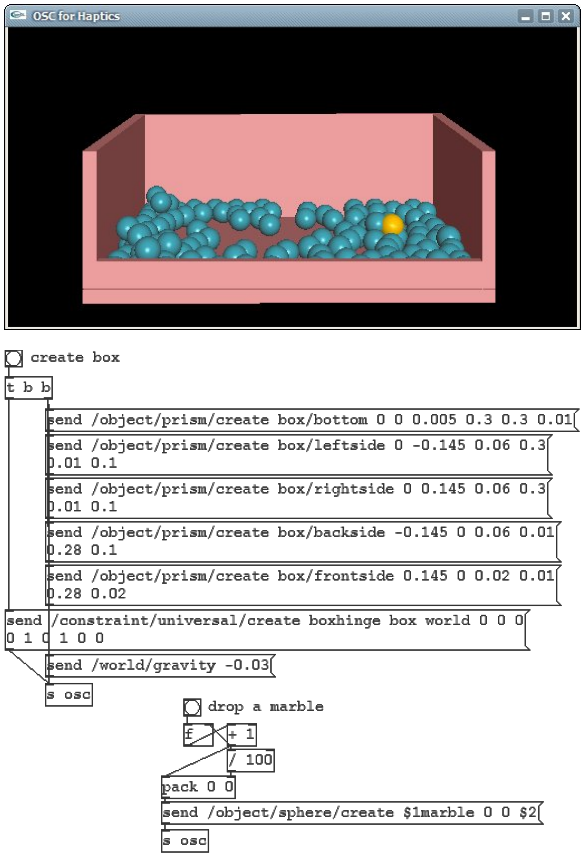

Figure 3: The Bucket of Marbles example, and the PureData patch that created it. (The audio portion of the patch is not shown.) The light-coloured sphere represents the haptic proxy.

Other constraint responses, such as textures or non-linear springs, may be specified in a similar manner.

4.2 Other Examples

A few other examples of virtual instruments that are possible to specify using the OSC namespace semantics follows:

Bucket of Marbles: The PebbleBox [15] is a controller using audio analysis to detect collisions between small polished pebbles, which can be used to excite some synthesis engine, such as physical modelling of water or ice cubes.

A picture of the patch used to create this VMI can be seen in Figure 3. Note that two previous virtual implementations of the PebbleBox have been created in the Enactive Network [9]. One of these simulations used the same physical dynamics engine (ODE) as the implementation discussed here.

Elastic Theremin: O'Modhrain and Chafe [14] discussed how a simple elastic band allowed the kinesthetic sense to aid in playing the Theremin. Something similar could be implemented by specifying a fixed object's "gravity well" so that a weak force is constantly present on the force-feedback device which is proportional to its position in space.

Scanned Synthesis: Verplank et al. [22] introduced a synthesis technique where an audio buffer is scanned at audio rates, while its contents are manipulated at "haptic rates". (That is, the speed at which we can manipulate objects with

our muscles.) This could be accomplished by specifying a deformable plane, and asking the mesh to report the positions of its nodes. In 3D, the extra dimensionality could be used for timbral effects or stereo separation.

Scraping: van den Doel et al. [21] created a virtual environment in which textured objects could interact to create audible scraping, sliding, and rolling sounds. OSC could be used to specify parametric textured surfaces, and for sending back interaction profiles, (stochastic or otherwise), that would inform a synthesis algorithm.

Also, If small-scale interactions can be calculated by the physical model in advance, for instance by using velocity information for prediction, cues could be sent to the audio process early, using OSC time-stamping for precise local timing.

5. DISCUSSION AND FUTURE WORK

We have shown that OSC can be used to specify a virtual instrument, which is constructed and modelled in a dedicated process, and subsequently to modify and get feedback from it to inform an audio (or visual) engine. Here we explore some issues that might arise in working this way.

Firstly, inter-process and inter-computer communication is necessarily slower than tightly integrated systems which share memory. Usually, OSC messages are transmitted using UDP/IP, even when communication is between processes on the local computer. This protocol may have a variety of physical transports, which may introduce latency between events in the physical model and sensory responses in the audio and visual modes. Experiments have shown that the Just Noticeable Difference (JND) for audio-haptic latency is about 24 ms [1], implying that messages must be received and processed by the audio subsystem within this time frame.

We have not yet performed formal tests on actual latency observed when using OSC to communicate between haptic and audio processes. However, informally we have observed few problems with small numbers of objects when the two processes are communicating over a local loop-back UDP connection on a dual-core computer. An example such as the PebbleBox simulation could be used to measure how many objects can effectively be tracked this way.

It is important to remember that OSC is a purposely verbose protocol, and that it is transport-independent. In other words, it puts clarity over efficiency, with the assumption that the actual transmission medium should be chosen to be fast enough for whatever use it is put to. For instance, one could easily imagine an implementation which uses interprocess shared memory, or named pipes, both which would likely be very fast mediums on a local computer. Thus, the advantages of having clear, human-readable message names outweighs the disadvantages of its verbosity.

6. ACKNOWLEDGMENTS

This work was sponsored in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canadian Foundation for Innovation (CFI), and the Enactive Network European Project.

7. REFERENCES

[1] B. D. Adelstein, D. R. Begault, M. R. Anderson, and E. M. Wenzel:. Sensitivity to haptic-audio asynchrony. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, pages 73–76, Vancouver, BC, November 2003. ACM.

[2] C. Cadoz, A. Luciani, J.-L. Florens, and N. Castagné. ACROE-ICA: Artistic creation and computer interactive multisensory simulation force feedback gesture transducers. In *Proceedings of the Conference on New Interfaces for Musical Expression*, 2003.

[3] F. Conti, D. Morris, F. Barbagli, and C. Sewell. CHAI 3D. `http://www.chai3d.org/`, November 2006.

[4] M. Danks. Real-time image and video processing in GEM. In *Proceedings of the International Computer Music Conference*, pages 220–223, 1997.

[5] D. DiFilippo and D. K. Pai. The AHI: An audio and haptic interface for contact interactions. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, pages 149–158, 2000.

[6] Handshake VR. proSENSE Virtual Touch Toolbox. `http://www.handshakevr.com/`, November 2006.

[7] S. Harris and N. Humfrey. LibLo: Lightweight OSC implementation. `http://liblo.sourceforge.net/`, January 2007.

[8] M. J. Kilgard. *The OpenGL Utility Toolkit (GLUT) Programming Interface: API Version 2.* Silicon Graphics, Inc., August 1995.

[9] C. Magnusson, A. Luciani, D. Couroussé, R. Davies, and J.-L. Florens. Preliminary test in a complex virtual dynamic haptic audio environment. In *2nd Enactive Workshop*, McGill University, Canada, May 2006.

[10] T. Mäki-Patola and P. Hämäläinen. Latency tolerance for gesture controlled continuous sound instrument without tactile feedback. In *Proceedings of the International Computer Music Conference*, Miami, USA, Nov 2004.

[11] W. R. Mark, S. C. Randolph, M. Finch, J. M. V. Verth, and I. Russell M. Taylor. Adding force feedback to graphics systems: issues and solutions. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 447–452, New York, NY, USA, 1996. ACM Press.

[12] M. Minsky, O. young Ming, O. Steele, J. Frederick P. Brooks, and M. Behensky. Feeling and seeing: issues in force display. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 235–241, New York, NY, USA, 1990. ACM Press.

[13] A. Mulder. *Design of Virtual Three-dimensional Instruments for Sound Control.* PhD thesis, Simon Fraser University, 1998.

[14] S. M. O'Modhrain and C. Chafe. Incorporating haptic feedback into interfaces for music applications. In *Proceedings of ISORA, World Automation Conference*, 2000.

[15] S. M. O'Modhrain and G. Essl. PebbleBox and CrumbleBag: Tactile interfaces for granular synthesis. In *Proceedings of the Conference on New Interfaces for Musical Expression*, 2004.

[16] M. Puckette. Pure Data: another integrated computer music environment. In *Proceedings, Second Intercollege Computer Music Concerts*, pages 37–41, Tachikawa, Japan, 1996.

[17] Reachin' Technology. Reachin' API. `http://www.reachin.se/products/reachinapi/`, November 2006.

[18] S. Sinclair. OSC for haptic virtual environments: Specification. Technical Report MUMT-IDMIL-07-01, McGill University, Music Technology Area, Feb 2007.

[19] R. Smith. Open dynamics engine. `http://www.ode.org`, November 2006.

[20] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), March 2005. Available from `www.gotw.ca/publications/concurrency-ddj.htm`.

[21] K. van den Doel, P. G. Kry, and D. K. Pai. FoleyAutomatic: physically-based sound effects for interactive simulation and animation. In *Proceedings of the 28th annual conference on computer graphics and interactive techniques*, pages 537–544, 2001.

[22] B. Verplank, M. Mathews, and R. Shaw. Scanned synthesis. *The Journal of the Acoustical Society of America*, 109(5):2400, May 2001.

[23] W. Verplank. Haptic music exercises. In *Proceedings of the 2005 International Conference on New Interfaces for Musical Expression*, pages 256–257, Vancouver, Canada, 2005.

[24] M. Wright, A. Freed, and A. Momeni. OpenSound Control: State of the art 2003. In *Proceedings of the Conference on New Interfaces for Musical Expression*, 2003.