Using PureData to control a haptically-enabled virtual environment.

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

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

ABSTRACT

This paper presents a client-server model for using PureData to create and control a physically dynamic virtual environment, with which the user can interact using a hand controller haptic device.

Keywords

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

1. INTRODUCTION

Today, the commercial market for force-feedback haptic devices is finally beginning to reach consumer-level prices. In this context, it is interesting to think about how this new modality can be integrated into programs like PureData (Pd), which already supports audio and video processing.

One possible approach is to create a "haptics" external for Pd, allowing direct access to the haptic device. This would consist of a set of objects which read the device's position, and output forces to the device's motors. A diagram of how this might work is given in Figure 1. In fact, this approach is feasible and we have implemented such objects for SensAble's OpenHaptics drivers as a proof of concept. However, the results are of very low haptic fidelity.

Unfortunately, haptics has different computational demands than both audio and control data. While audio data can afford latency of a few milliseconds, making it possible to compute it in blocks of samples, haptic data must be calculated in a single-sample manner with latency of no more than 1 millisecond [6], (due to 500 Hz being close to the upper bound of the tactile frequency range.) This is essentially because a haptic device is both an input and an output device—the two data directions are inherently coupled in a closed-loop fashion, with the human operator being part of the loop. In order to provide a realistic "touch", and to simulate "stiff" walls, it is required to provide high-speed throughput, which implies real-time constraints that are not present in other modalities.

PureData Convention '07 Montreal, Canada

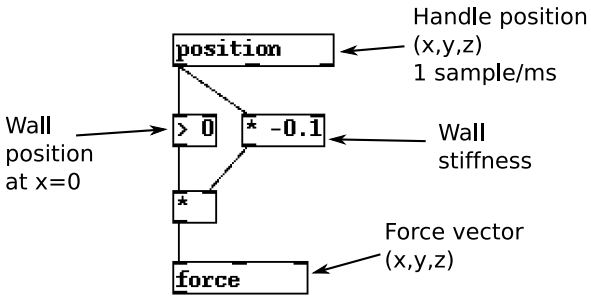

Figure 1: A PureData patch which calculates an undamped haptic *virtual wall* on the plane defined by the Y-Z axes of a 3-dimensional space. This approach, though working, cannot currently achieve the real-time throughput and steady timing necessary for good haptic fidelity.

Previously, we presented an approach to integrating haptics with existing software that makes use of a client-server model for separating haptics computations from the audio software [10]. The haptics "server" takes commands from a PureData patch running on the same or another computer through the OpenSoundControl (OSC) protocol [12]. The specified OSC messages allow the dynamic creation of 3D objects which can be touched by a haptic device, while constraints on their movement can also be specified, so that virtual mechanisms may be constructed. Here, we give an overview of this project, which we hope may be of interest to the Pd community, and present some more recent Pd-specific developments.

2. CHALLENGES FOR REAL-TIME 3D HAPTICS COMPUTING IN PD

The changes required for integrating a direct implementation of haptic servos in Pd would be non-trivial. It would require, essentially, building a third system of patch cords (in addition to the audio and control patch cords). This is because the "position" object (Fig. 1) would have to send 1 sample of the position vector every millisecond with guaranteed regularity and as little jitter as possible. In contrast, the audio system computes a block of samples at a time, and puts them on the hardware's buffer every 5 to 50 milliseconds, while the control system runs asynchronously, with no real-time requirements. A simple patch (Fig. 2) can show the difficulty in achieving this latency with Pd.

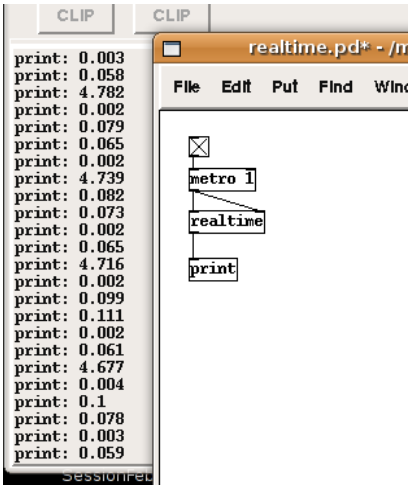

Figure 2: Demonstrating the difficulty of achieving millisecond timing in PureData: screenshot taken under Ubuntu Linux 7.04 running the 2.6.20-16-realtime kernel and the "-rt" Pd option as root without audio processing enabled, on a dual-core AMD computer.

Another disadvantage to such an approach is that the haptics system would run within the Pd process. While not inherently a problem, it is often the case that haptic algorithms are computationally expensive, and it is helpful to dedicate a CPU purely to the servo. Often a haptics servo is executed on seperate hardware, such as a dedicated DSP, to guarantee timing. Thus, the haptic process should at least be separable, and communications should be asynchronous (between modalities), so as to allow uninterrupted real-time performance.

Since force-feedback haptics usually implies some sort of 3D rigid or deformable body simulation, a final difficulty in implementing virtual reality systems purely in Pd is that a high-level approach would generally be preferred. Pd lends itself well to *description* and *control* of the environment, but low-level mathematics libraries, such as those discussed in the next section, are typically implemented as C externals, providing access to functionality through messages or special objects and data structures.

3. PHYSICAL MODELLING IN PD

The state of the art for physical modelling in Pd is represented by a set of objects designed by Henry Cyrille called "pmpd" [5], which was later optimised into a single object called "msd" by Nicolas Montgermont. These objects allow the patch to specify mass-spring systems representing a scene, which can then be stepped through time using triggers from a "metro" object or similar.

This mass-spring system is quite similar in concept to CORDIS-ANIMA [1], which has successfully been used to model physical systems for simultaneous audio, visual, and haptic representations. Therefore it is logical to presume that pmpd may also be used as such. However, there are a couple of limiting factors in this regard. Firstly that pmpd depends on the PureData scheduler, which, as described above, is not adequate for haptics computation. Secondly,

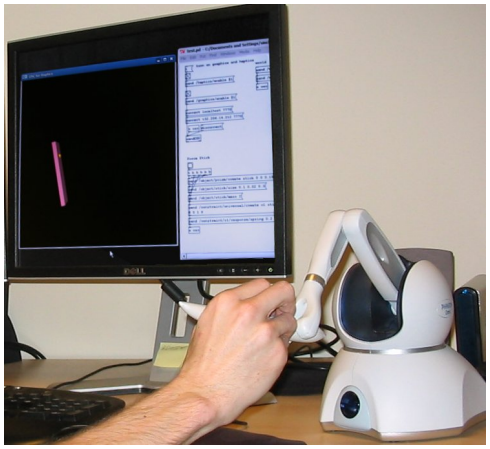

Figure 3: Using the Phantom Omni from SensAble Technologies to interact with an object. The PureData patch that manages the scene is visible on the right-hand side of the screen.

that we are interested in the simulation of rigid bodies, which are not supported by pmpd. Having said this, however, if PureData's scheduler could be made to meet haptics requirements, it could be very interesting to interface pmpd or msd with a haptic device. Another possibility could be to modify them to run their own scheduler at the haptic rate, but then a threaded system such as may be required begins to resemble the client-server model discussed here.

4. A VIRTUAL ENVIRONMENT SERVER

The need to separate physics and haptics processing from the Pd process and scheduler leads to the approach introduced in section 1, which is to create a sort of "haptics server", written in a natively compiled programming language, which implements a set of commonly-used algorithms, and allows a client to control and communicate with the simulation using messaging. Since the target here is to communicate with PureData or other audio software, a natural choice is to make use of the increasingly popular Open Sound Control (OSC) protocol. OSC's hiearchical addressing scheme lends itself particularly well to a simulation scene-graph consisting of objects and attributes.

PureData informs the server about what type of scene to construct, and the server instantiates objects on demand. Objects created in this virtual world can be touched and manipulated by a haptic controller, and can also collide with each other using a physical dynamics engine. Messages can be sent back to Pd, so that it can give audio or visual feedback of events. Thus, virtual objects simulated by this method are reminiscent of Mulder's Virtual Musical Instruments (VMI) [7]. Mulder stated that the haptic sense was missing from his early VMI, but with the system described here, that modality can be implemented.

The remainder of this paper will describe what kind of objects can be instantiated, and give some examples of how a Pd patch can create and receive feedback from the environment using OSC messages. Some notes on the implementation will follow.

A picture of the system in use is shown in Figure 3.

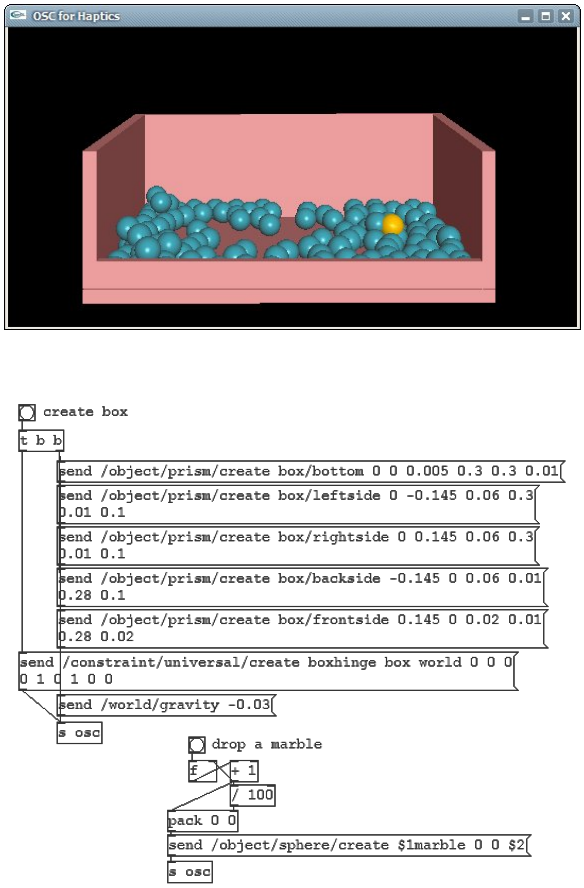

Figure 4: The PebbleBox example, and the Pure-Data patch that created it. (The audio portion of the patch is not shown.) The light-coloured sphere represents the haptic proxy, which can push on the "marbles".

5. OBJECTS AND CONSTRAINTS

Currently, only simple primitives are supported; namely, spheres and prisms. However, more complex shapes can be created by combining them into compound objects. Future work will include arbitrary mesh objects, as well as deformable surfaces, and haptic texture rendering. In the meantime, it has been found that plenty of interesting VMI can be created with these simple shapes.

For example, a model of O'Modhrain's PebbleBox [8] was created by constructing a box shape, and filling it with spheres. When the spheres collide, the collision forces are used to excite a physical synthesis, just as in the real PebbleBox. An image of this system, and the patch, can be seen in Figure 4. We have used it to excite a simple modal synthesis algorithm, and have also interfaced it to allow gestural control of a spatialization patch, with each sphere representing a sound-file "object".

In additional to events like collision, continuous properties of objects can be requested, such as acceleration, velocity, position and rotation. Any property can be requested either instantaneously, or to be reported at regular intervals over time.

To create relationships between objects, messages can be used to specify *constraints*. Constraints determine how objects can move in relation to each other, or in relation to the global coordinate system. Constraints correspond with the Open Dynamics Engine's (ODE) concept of *joints* [11]. Some examples of constraints that can be specified are hinges, ball joints, sliding joints, and universal joints.

Each of these constraints specifies restrictions on how objects can move—implicitly, each constraint has one or more free axes (e.g., a hinge can rotate in one direction). Thus, each constraint can also be given some "response", which is a force-profile function to be applied to the constraint's free axes. For example, a hinge can have a damped spring applied to its direction of movement. Constraint responses can also be textures, to provide a "grainy" feel, or can be given a virtual wall or breakable membrane (a "pluck") at some particular position.

6. NAMESPACE

Objects are created using the appropriate *create* message, and are subsequently considered part of the namespace. For example,

```
/object/sphere/create s1 0 0 0
/object/s1/force 10 0 0
```

This would create a sphere named *s1* and then apply a force of 10 N with a direction along the X axis. Haptic and visual materials for objects can also be specified this way, such as friction, or colour. To create a constraint, a similar syntax is used:

```
/object/prism/create box1 -0.1 0 0
/object/prism/create box2 0.1 0 0
/constraint/hinge/create h1 box1 box2 0 0 0 0 1 0
```

After creating two boxes at the default size, the third message creates a hinge between them which rotates around the origin (0,0,0) on the Y axis (0,1,0). Pushing on either of these boxes, the user will see them move relative to each other around this axis, as well as moving away from the origin with their collective momentum.

To specify a constraint's response, for example to give the hinge a spring, it can be attributed with the *response* message:

```
/constraint/h1/response/spring 0.001 0.0001
```

This specifies a spring with a stiffness of 0.001 N·m/rad and a damping factor of 0.0001 N·m/rad/s. This very weak spring will have the effect of making the rotation of one box pull lightly on the rotation of the other box, so that they will follow each other around the shared axis.

The final step might be to retrieve the torque on this spring, so that pushing on it with the haptic controller would yield some change in sound.

```
/constraint/h1/torque/magnitude/get 10
```

The above tells the server to report the constraint torque's magnitude every 10 ms. PureData would then receive an OSC message at regular intervals which it could use to modulate some timbral parameter.

Further information on the currently supported namespace can be found on the project's website [1], and the projected specification for future work can be found in [9].

[1] http://www.music.mcgill.ca/musictech/idmil/projects/forcefeedback

7. IMPLEMENTATION

The "haptics server", named DIMPLE (for Dynamic Interactive Musically PhysicaL Environment) is implemented in C++ using a combination of CHAI 3D [2] (for haptics) and the ODE (for physical dynamics). Since each library requires its own scene graph, an instance of objects are created once in each and corresponding object positions are re-synchronized periodically. The ODE runs at some slower speed (usually 100 Hz), while CHAI's haptic loop runs at 1 KHz. The former calculates collisions between all objects in the scene and updates their positions according to physical variables. The latter calculates collision detection and friction for the haptic proxy—an object representing the haptic device. Currently the implementation uses shared memory between pthreads, but a move to separate processes may be considered in the future for more robust reliability and to allow for splitting the two tasks over different computers on a local network. OSC messaging is accomplished via LibLo [4]. The software is cross-platform, running on Linux, Windows, or Mac OS X. Unfortunately, not all device manufacturers provide drivers for all of these operating systems, but CHAI 3D supports at least 3 different devices for Windows, with Linux support for SensAble's Phantom devices.

In an attempt to investigate differences in inter-modal communication latency, DIMPLE has more recently been compiled as a Pd object. In this form, it no longer runs in its own process but is available through the usual Pd external API, loaded as a shared object. This implies that the UDP protocol and the loopback network are not needed, as OSC packets are passed using Pd's message system. Formal evaluation of the effect on inter-modal latency needs to be done. Additionally, it is hoped that this form of DIMPLE may provide a convenient way to stream information at the signal rate between the audio and haptic systems, making it possible to incorporate vibrotactile feedback into a simulation. We have previously considered using inter-process streaming solutions such as the JACK Audio Connection Kit [3] for this purpose, but Pd provides a simpler method for now. Note however that as a Pd object, DIMPLE cannot run on a separate machine, though usual techniques could be used to split audio processing to another Pd instance. It still makes use of pthreads to execute haptics and physics asynchronously, however, so multi-core computers may benefit.

8. DISCUSSION

The intent for this project is to provide a way for researchers in music technology to easily create rigid body virtual environments that interface comfortably with audio software, without having to go through the hoops of 3D programming in C++, and to provide haptics researchers a way to easily integrate sound into multi-modal demonstrations.

Future work with the environment will include user studies on the role of the kinesthetic sense in playing and learning to play virtual musical instruments.

We have also used the software for a few non-haptic tasks, purely taking advantage of the physics engine. For example, we have mapped a pressure-sensitive floor to the gravity vector, so that objects fly in one direction or another according to how a user moves his weight. We have also used spheres as "cannon balls" by applying an artificial force according to how hard a drum pad was hit, and then used the resulting trajectory to control the spatialization of sound sources as the spheres bounce around a virtual room.

In the future, more object types will be supported, as well as other kinds of haptic feedback effects, such as gravity wells (for easily locating and "picking" objects), and vibrational cues. The latter may be useful for simulating musical interactions such as bowing.

9. 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.

10. REFERENCES

[1] C. Cadoz, A. Luciani, and J. L. Florens. CORDIS-ANIMA: a modeling and simulation system for sound and image synthesis—the general formalism. *Computer Music Journal*, 17(1):19–29, 1993. MIT Press.

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

[3] Davis, P. et al. JACK Audio Connection Kit (software). Available: `http://jackaudio.org`.

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

[5] C. Henry. Physical modeling for pure data and real time interaction with an audio synthesis. In *Proceedings of the SMC*, 2004.

[6] M. Minsky, O.-Y. Ming, O. Steele, F. P. Brooks Jr., and M. Behensky. Feeling and seeing: issues in force display. In *SI3D '90: Proceedings of the Symposium on Interactive 3D Graphics*, pages 235–241, New York, NY, USA, 1990. ACM Press.

[7] A. Mulder. Virtual musical instruments: Accessing the sound synthesis universe as a performer. In *Proceedings of the First Brazilian Symposium on Computer Music*, pages 243–250, 1994.

[8] 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*, pages 74–79, Hamamatsu, Japan, August 2004.

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

[10] S. Sinclair and M. M. Wanderley. Defining a control standard for easily integrating haptic virtual environments with existing audio/visual systems. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 209–212, New York, NY, June 2007.

[11] R. Smith. Open Dynamics Engine (software). Available: `http://www.ode.org`, November 2006.

[12] 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*, pages 153–159, 2003.