Report: audio-haptic research, summer 2008

Stephen Sinclair
Input Devices and Music Interaction Laboratory

August 13, 2008

1 Introduction

This report summarizes lab-related work and activities which took place throughout the period between March and Septembre, 2008. Time was divided between several projects which will be outlined here. Sodme work took place at MPB Technologies Inc. as part of my NSERC IPS2 agreement. Collaboration with the Multi-modal Interaction Laboratory and the CIM within the Enactive framework is also described.

2 Work on DIMPLE

Continuing work on this project which was the focus of my master's thesis. I wanted to explore haptic interaction with rigid body systems for creating virtual musical controllers. It was deemed needed to rewrite much of the back-end in order to maintain stability. This has now mostly been completed. The basic functionality is entirely back: spheres and prisms can be constructed and interacted with haptically, and all desired ODE joint types have been added. Working now on cleaning up the source for an interim release, and adding spring response feedback to the "hinge" joint. Figuring out how to handle responses for the other joint types as well. This will constitute what I consider the 0.1 release. Planned for 0.2 is to add back the "extra" features discussed in last year's Enactive '07 paper: object grabbing, signal input from PureData, and textures. Looking into use the texture algorithm discussed here in section 5. Object grabbing will be improved now that I have solved the angular spring problem, discussed in section 4. When all major features have been implemented, documentation has been written and a more user-friendly GUI has been constructed I will consider it the 1.0 release. I plan to work on this continually throughout the coming year, but it cannot be considered a main focus after the 0.2 release, which will implement many features already claimed in published and submitted papers. (These features have been implemented previously, but only as proof of concept—integrating them properly into the new framework is another job.)

3 Work on ERGOS device

The ERGOS device was purchased last year, which has the unique capability of running haptics on a DSP at 40 KHz. My initial plan is to get it working with DIMPLE simply by using a bridge between the DSP and CPU. (Running at approx. 1 KHz for now.) Later I plan to try getting the whole haptic portion of DIMPLE running on the DSP, communicating with the other portions of the program over the PCI bus, which would allow 40 KHz haptics in DIMPLE. The software provided by ACROE however constituted a somewhat complete C++ framework for developing OpenGL-driven displays which communicate with the Toro board. Much of this functionality conflicts with what's already available in DIMPLE. So what I've begun working on is a smaller encapsulation of the required PCI bus communication called "libtoro", based on Damien Couroussé's work, which will function as a very simple, barebones driver to the analog I/O lines of the Toro. This will enable easier construction of a CHAI3D driver for the ERGOS. The driver will have to know something about the configuration of the ERGOS device, such as which attachments are coupled to which keys, so a configuration interface will have to be created as well. So far, libtoro is able to initialize the card and load firmware, so mainly the communication interface needs to be established.

4 Progress on 6-DOF bowing simulation

During discussions and work with Erwin Schoonderwaldt last year, we concluded that at least 5 degrees of force feedback are necessary for a full violin bowing simulation. Due to my work with the Freedom 6S controller, I am interested in achieving a better understanding of 6-DOF physics and haptics, so I was interested in attempting this project. At least a year ago I had the idea for DIMPLE that if an object is virtually coupled to the controller's end effector it would be easy to take advantage of a rigid body dynamics library to do all the hard work, only needing to apply a spring-damper system to apply the forces and torques to the device. I achieved this in 3-DOF with DIMPLE, but it turned out that doing the same with torque was difficult due to singularities in mathematical representations of orientation. In certain orientations, trying to have one object follow another's orientation by applying torque in the Open Dynamics Engine caused the second object to spin a some axis, or otherwise move chaotically. I had been trying to achieve this using Euler angles to calculate the angular difference, which is where the singularity comes from.

With some of my days at MPBT dedicated to 6-DOF modeling, I decided to attack this problem again during Septembre and August. This time I used a quaternion (4-vector) representation of orientation, which is known to be singularity-free. I found it is also used as the internal representation in ODE, which was convenient, since it provided functions for quaternion manipulation. This algorithm did work: I was able to use quaternions to compute the necessary torque to move one object towards a target orientation. This is a watershed moment for 6-DOF haptics, since it means that virtual coupling can be used to easily calculate forces *and* torque necessary for a 6-DOF haptic device, greatly simplifying the task of haptic interaction by taking advantage of rigid body engines already able to preform these calculations. Further work is needed to determine optimal spring/damper coefficients according to an objects mass and shape, but for the moment trial and error is working okay. Some work will be needed to eliminate a "flipping" effect which the orientation reaches a particular point where it seems to want to apply a 360 degree rotation, but for the moment I am avoiding this. Also, I found it was necessary to apply stiffness asymmetrically to the haptic device since the end effector does not have the same motor configuration for each axis.

The algorithm works as follows: The Freedom 6S API provides orientation as a 3×3 matrix. This can be converted losslessly to a quaternion. The object's orientation can be retrieved directly as a quaternion from ODE. The "difference" between the device orientation q_d and the object's orientation q_o can be calculated using ODE's `dQMultiply2()` function, which performs a quaternion multiplication (cross product) with an inverse of the second quaternion:

$$q_\Delta = q_d \times q_o^{-1} \tag{1}$$

The "distance" required to go from q_o to q_d can then be retrieved by the latter 3 elements of q_Δ. (I put "distance" in brackets; I am not entirely sure what the units are here, since I lack a formal understanding of the mathematics.) A spring damper algorithm uses this "distance" metric with stiffness and damping coefficients to apply to the object by ODE's `dBodyAddTorque()`, and the opposite (negative) stiffness and damping coefficients to apply to the Freedom 6S with the `f6s_SetForceTorque()` function. In practice I also had to divide down the device stiffness by some magnitude, otherwise the haptic device would fly away or oscillate vigorously. Various combinations of mass, stiffness, and damping coefficients also caused the algorithm to explode, and subsequently the program to crash, so some formal understanding of how these coefficients interact will need to be achieved. Thus trial and error was used to determine working coefficients.

One stumbling block: most literature I found on quaternions did not discuss this algorithm but instead discussed what's called a "slerp", or a linear interpolation between two orientations. This "slerp" algorithm does not provide torque, but merely the intermediate steps needed for a smooth animation between two orientations. This is useful for video game camera movement, for example. However I found one or two forum posts discussing this calculation, but they recommended the following: $q_\Delta = q_d^{-1} \times q_o$. This stumped me for some time because it resulted in a singularity as with the Euler algorithm. Finally I discovered that equation 1 worked better, so I will need to determine whether this was just a difference in notation, whether I had my variables reversed, or something else. (Quaternion multiplication is not commutative, so reversing the variables would have an unexpected effect.) There is also some information on quaternion calculation of torque in the engineering literature on things like space craft attitude control, but I haven't yet read this in depth. I searched for some time for papers mentioning the use of quaternions for haptic rendering of torque and found a few brief mentions here or there but not very much, meaning either it is novel, or considered obvious.

In any case, using ODE to represent the haptic device's end effector, I created a scene with a "bow" and four "strings". In the collision function I calculated the bow velocity, pressure, and position, and used these modulate the STK *Bowed* object which implements a bowed string digital waveguide model. The user can then use the haptic device to play a virtual violin, where each strings makes a different frequency.

Further work is needed: to guarantee stability in the algorithm while preserving stiffness; to integrate a bowed string friction model; and finally to more closely couple the friction algorithm and the sound-producing audio-rate simulation. Some work on this latter part has been done by Mounia Ziat as part of the Enactive project, and I also have code from Stefania Serafin on her system. I am working on a proposal for a special project for the fall semester to explore these in detail, which I'll describe again in section 6.

5 Work on texture study

Last year I decided to try a straight-forward geometric texture rendering algorithm, to see how it could be used to interact with a texture sonically and haptically. I succeeded in creating a geometric texture renderer and used it for generating Dirac impulses for a modal synthesizer. Another use will eventually be to integrate it with Mathieu Lagrange's "rolling synthesizer".

Catherine Guastavino and Rafa Absar from the multimodal interaction lab wanted to use this algorithm for testing subjects in a target-finding study using sonic and haptic cues. It was decided that before doing target-finding with texture gradients, it would be necessary to do a JND study on haptic textures displayed with the Freedom 6S to know when subjects could tell the difference between various amplitude and wavelength settings of a sinusoidal texture. This study has now been designed and is in the process of being carried out. It took longer than expected for two reasons: it was decided after designing the methodology to change to a different experimental procedure, (from a staircase algorithm to the "constant" method); and a problem with the vertical sensor slipping on the device meant that it was returned to MPBT and a replacement device was acquired. (Work on the software was not really stopped for this last problem, but it was a concern and probably contributed to delays in the run-debug cycle of this project.) The target finding study is ideally planned to be completed by Sept. 1 also, though specifics are still to be discussed.

5.1 Algorithm

I'll outline the texture algorithm here as a start to documentation on the technical aspects of this project. There are several ways to render textures haptically. I wanted something bitmap-based so that I could display textures created by external 2D algorithms, but similar principles could be used for generative textures by essentially replacing the bitmap look-up routine with a mathematical function. In any case, texture rendering amounts to modulating some aspect of the force feedback according to the contact point on a surface. This can be the normal of the force vector, or the stiffness, or the friction coefficients, for example. Combining height offsets and normal mapping can result in "geometric" rendering of the texture. I surmised this last would be the most "realistic" of these. Previously DIMPLE was using a stiffness modulation which provided a sense of texture but no change in lateral forces, and thus was extremely dissatisfying to me, so I wanted to determine the difficulty level and computational needs of geometric rendering. Texture rendering is one of the few areas where haptic rendering is actually more computationally efficient than visual rendering, because only points close to the haptic cursor need be considered. I wanted to perform geometric rendering without sacrificing this property.

I started with a kind of naïve "brute force" approach: using a macroscopic surface of only 16 by 16 "texels", I rendered it as a height field on the screen, and tried to calculate forces reflecting off the normal of the closest triangle to a point. I determined the closest triangle by projecting the point's location on the XY axis of the surface, then using the Z value for the bitmap look-up. This determined the texel, but since each texel is composed of two triangles, I had to then determine which triangle was closest to the point by comparing the squared distance to the center of each.

The coefficients of the plane of the triangle are then determined, as well as its normal. The distance to the triangle is then calculated along the normal by taking the intersection between a line and a plane:

$$d = \frac{|Ax + By + Cz + D|}{\sqrt{A^2 + B^2 + C^2}}$$

where d is the depth below the surface, x, y, and z are the components of the haptic cursor position, and A, B, C, and D are the coefficients of the triangle's plane. This depth value is then used as a unilateral spring damper to create a virtual wall through the triangle's plane. Additionally the Z location on the surface is also calculated by substituting the XY components of the haptic cursor location into the plane equation, however this is only used for visual display in order to avoid drawing the haptic cursor below the surface.

While this worked, it was found that in valleys the haptic device would begin bouncing back and forth between two walls. This was because triangles pointing toward each other would create forces sending the cursor toward the opposite wall, leading to oscillation. To avoid this, force shading was implemented: In this approach, the normals of the texel's corner point and four closest corner points were calculated to form a diamond shape surrounding the cursor location. The normal at the cursor location was then interpolated between these four points, creating a smooth transition from one texel to the next, and also had the benefit of giving valleys an upward-facing normal, avoiding the bouncing problem. Each point normal was calculated by averaging the normals of the point's 6 surrounding triangles. This is possibly a wasteful approach and could be optimized, but was chosen because by forming a square shape it was possible to linearly interpolate between them; triangular interpolation is more complicated and I decided to avoid it for now.

A last problem was encountered when miniaturising the texture. I found that as texels got smaller and smaller, the stiffness coefficient k used to render the surface needed to be increased by several magnitudes in order to feel the same stiffness. This was non-physical and also could potentially lead to loss of precision, so I investigated the problem. I realized it was because I was basing the stiffness entirely on the cursor's depth below an angled plane which was bounded by the size of the triangle. Therefore, the smaller the triangles became, the more the depth was limited, leading to a limited final force. (In other words, if the depth were larger, the cursor would be under the next texel, so the depth was effectively bounded by the texel size.)

I solved this by dividing the penetration depth into two regions: depth into the texture, and depth below the surface, where here "surface" refers to the flat surface on which the texture lies. The *texture depth*, d_t, is defined as the vertical distance between the triangle surface point and the cursor's Z component, bounded by the distance between the texture and the surface point. The *surface depth*, d_s, is defined as the "left over" vertical distance between the surface point and the cursor's Z component. Force is then calculated as,

$$\vec{F} = d_t k \vec{n_t} + d_s k \vec{n_s}$$

This is a bit of a "hack" and is somewhat unphysical because the distance is no longer calculated along the texture normal. Macroscopically, under some conditions, the force feels somewhat too vertical, tending to push the haptic cursor toward the peaks; microscopically, however, (in a texture with small texels), it feels quite satisfying, providing enough lateral force to give a sense of texture while still feeling stiff normal to the whole surface.

The algorithm currently assumes only a single, upward-facing surface. In a multiobject scenario such as in DIMPLE, prism surfaces would need to have a bounding box defined some distance above them in which the texture algorithm would be turned on. Surfaces at convex angles might cause instabilities if textures are close to each other or even intersect, or if two objects are close to each other. There are likely other issues with spherical mapping of the textures as well. These issues are well-known and described in the literature.

(Indeed, this work on texture rendering is probably not novel, but it was an interesting exercise to get some hands-on experience with these issues, and to create a practical algorithm for use with DIMPLE and for exploring audio synthesis based on haptic texture interaction. There is a good deal of literature on haptic texture rendering, including work by Gianni Campion, a PhD student in Vincent Haywards lab at McGill's CIM. I will likely *not* attempt 6-DOF texture-texture interaction, which can be a much more complex problem. See for example Ming C. Lin's work at the UNC for information on that subject.)

6 Preparations for next semester

Chosing my remaining three classes has proven less than obvious, but I have finally decided to take a 500-level graphics course being offered by the computer science department. Haptic rendering often borrows or is inspired by techniques from graphics. Though I think I will already know the material for at least the first half of the course, I have not previously taken a graphics course and expect to discover some new and interesting topics. Additionally, despite being solidly focused on audio interaction with haptics, it's

interesting to be able to make my demonstration applications more visually appealing, so this may be a good chance to do so.

Considering current research previously mentioned here by Mounia Ziat as well as having my hands on code from Serafin, I have also proposed to Gary Scavone to start a "special project" MUMT-609 course in which I'll explore the application of digital waveguides to haptic rendering. I feel I'm in a unique position to explore this topic, having access to a haptic device capable of 40 KHz rendering. I'm aware of some work on this topic, but feel it could be taken further to fully close the loop between the audio and haptic rendering algorithms. Use of the same hardware means that this work will be directly comparable to real-time work at ACROE in which they take a mass-spring network approach to rendering for bowing synthesis, so comparing the two formally somehow will hopefully provide some insight. Additionally I'm as usual quite interested in seeing what happens when the performance or coupling of the system is gradually degraded. I'd like to know at which point a user will notice (or care about) the degradation, and whether or not it affects their ability to perform a musical task. (Ultimately to determine the effect of less performant haptic systems being used at "only" 1 KHz.) I will be writing a more official and well-researched proposal for this topic in the next few weeks.

7 Conclusion

Last semester I spent a good deal of time completing class assignments and working on and revising an article on DIMPLE for the IwC journal. Throughout the summer I have been playing "catch up" with many ideas that have cropped up during the last year of projects. Issues such as the angular spring algorithm have been real bottlenecks for what I wished to achieve, and I am very happy to have mostly solved these. One nagging thought throughout these projects is that I am constantly worried that despite the interesting haptic research, I wonder if it will lead to interesting *musical* results. Now that I am finding it easier to tie these two subjects together I am hoping that a more concrete thesis topic will emerge. Studying control of waveguides using haptics should provide some interesting results that will perhaps be at odds with the DIMPLE approach of using rigid body systems as modulators. I think it is possible that ACROE's idea of requiring absolute synchroneity between physical models is the only true approach to sonic interaction, but it's not necessarily the only interesting one, and this philosophy also doesn't imply that mass-spring models are the only way to go. However, eventually it would be interesting to also explore mass-spring ideas à la CORDIS-ANIMA. I'm interesting in tying everything together as much as possible, but I would also be happy if I could describe in a thesis that several approaches are possible and have advantages and disadvantages that can be concretely explained.