Motivation

• Why add sound to visuohaptic simulation?
  – It’s more realistic and engaging
  – It’s way cool
• Option 1: Play back a .mp3 file on contact
  – Different sounds for each force interaction?
  – Different sounds for each impact position?
  – Must be created by sound effects specialist
• Option 2: Synthesize sound in real-time based on haptic force

Outline

• Inspirational Videos and Demos
• Algorithm for Real-Time Interaction Sounds
• Brief Intro to BASS Audio Library
• CHAI Examples
• Resources

The Sounds of Physical Shapes


Haptic Audio Algorithm

• Modal Synthesis Model
• Obtaining Model Parameters
  – Analytically
  – Experimentally
• Convolving Sound Model with Force Profile
• Haptic and Audio Forces
• Special Effects

The Audio and Haptic Interface (AHI)

Modal Synthesis Model

- The impulse response is a sum of damped sinusoids, each associated with a frequency
  \[ y(p,t) = \sum_{n=1}^{\infty} a_n(p) e^{-\alpha_n t} \sin(\omega_n t) \]
- Each of the \( n \) angular frequencies, \( \omega_n \), is associated with a damping coefficient \( \alpha_n \) and an amplitude \( a_n(p) \) that depends on impact location \( p \).
- Model is derived from the solutions to the wave equation PDE

Analytically Obtaining the Frequency Spectrum

- Wave equation:
  \[ \left( \frac{\partial^2}{\partial x^2} + \frac{1}{c^2} \frac{\partial^2}{\partial t^2} \right) \phi(x,t) = 0 \]
- \( \mu(x,t) \) is the deviation from the surface of a point \( x \) at time \( t \) (assuming no external force)
- \( c \) is a constant related to speed of sound in material
- \( A \) is a differential operator that depends on object shape
- Solution to wave equation:
  \[ \phi(x,t) = \sum_n (a_n(x) \sin(\omega_n t) + b_n(x) \cos(\omega_n t)) \Psi_n(x) \]
- The \( \omega_n \) values come from the eigenvalues of \( A \) (\( \omega_n = -\omega_n^2 \)) and \( \Psi_n(x) \) are the corresponding eigenfunctions

Analytically Obtaining the Frequency Amplitudes (Timbre)

- Wave equation solution:
  \[ \phi(x,t) = \sum_n (a_n(x) \sin(\omega_n t) + b_n(x) \cos(\omega_n t)) \Psi_n(x) \]
- Using orthogonality to solve for coefficients:
  \[ a_n = \int_0^L \frac{\lambda_n \Psi_n(x) \sin(\omega_n t)}{\cos(\omega_n t)} dx \]
- With initial values assuming the object at rest (\( y_0(x) = 0 \)) and velocity equal to the Dirac delta function at the distance from the point to the impact location \( p \)
  \( \phi(0) = 0 \)
- Leaving us with
  \[ \phi(p,t) = \sum_n (a_n(p) \sin(\omega_n t)) \Psi_n(x) \]
- By normalizing the eigenfunctions, \( \psi_n \), can be made independent of \( n \). Since we only care about the relative amplitudes of the modes, we can drop \( \psi_n \) and

Model Parameters

- How do we get the values for \( \omega_n \), \( a_n \), and \( \alpha_n \) for a given object?
  - Analytically by solving a PDE
  - Fitting the model to empirical data
- Physical interpretations
  - The frequency spectrum \( \omega_n \) values depend on the shape of the material
  - The timbre \( a_n(p) \) values depend on location of impact \( p \)
  - The decay rate \( \alpha_n \) values depend on internal friction

Analytically Obtaining the Frequency Spectrum: Example

- For a 2D rectangular membrane of dimensions \( L_x \) by \( L_y \):\
  \[ A = \frac{\partial^2}{\partial x^2} \frac{\partial^2}{\partial y^2} - \left( \frac{\mu}{c^2} \right) \frac{\partial^2}{\partial t^2} \]
  \[ \mu \phi(x,y) + \frac{\mu}{c^2} \phi_t(x,y) = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \]
- Analytically by solving a PDE

Tweaking the Wave Equation to Get the Sound Equation

- We will not consider position of listener, so we convert displacement function \( \phi(p,x,t) \) to sound function \( y(p,t) \)
- Interference from reflections mostly cancels position effect anyway
- The internal friction \( \alpha \) depends on the material and is assumed invariant over the object
- Higher frequencies decay more rapidly
- Quality degrades gracefully as fewer modes are used, so \( n_f \) can be decreased dynamically when computational resources are limited
- Thus, we have
  \[ y(p,t) = \sum_n a_n(p) e^{-\alpha_n t} \sin(\omega_n t) \]
Experimentally Determining Sound Parameters

- Could manually apply force with a pen at various points on the surface of an object and record sound with a microphone
- Analyze measurements in frequency domain to find modes, amplitudes, and decays
- Model is linear with force, so multiple force magnitudes are not needed
- Robot improves accuracy of sample points and force magnitudes and automates process
- ACME (Active Measurement) robot at UBC used for measuring object parameters (including sound; also deformation, texture, etc.)

Sound Map

\[ y(t, p) = \sum_{n} a_n(p) e^{-j\Omega_n t} \sin(\omega_n t) \]

- \(\omega_n\) and \(d_n\) do not depend on location of impact; \(a_n(p)\) does
- Create a map of \(a_n\) values over the surface (like a texture map)
- For a given \(p\), interpolate between map values
- Timbre varies non-uniformly over surface, so you might choose sample points for map based on a measure of "sonic distance"

Convolving Impulse Response with Force Profile

- Discrete convolution of functions \(f\) and \(g\) at time step \(k\):
  \[ (f * g)(k) = \sum_{l=-\infty}^{\infty} f(l) \cdot g(k-l) \]
- With \(f\) as the force profile \(F(t)\) and \(g\) as \(y_n(t)\), the discretized sound function for mode \(n\), with \(SR\) the sample rate
  \[ (f * y_n)(k) = \sum_{l=-\infty}^{\infty} F(l) \cdot y_n(k-l) \]
- As a recursion
  \[ h_k(0) = a_n F(0) \]
  \[ h_k(k) = e^{-j\Omega_n k} h_k(k-1) + a_n F(k) \]
  - Intuition: At each time step, all previous contributions are decayed by \(e^{-j\Omega_n}\), and contribution of most recent force has not yet decayed

Notational Changes

- Sound equation for a specific mode \(n\) and for a specific impact position \(p\) (dropping \(p\) as a variable and dropping the summation of modes):
  \[ y_n(t) = a_n e^{-j\Omega_n t} \sin(\omega_n t) \]
- For notational convenience, absorb \(c\) into \(\omega_n\) and rewrite as the imaginary part of complex waveform with complex frequencies \(\Omega_n = \omega_n + id_n\)
  \[ y_n(t) = a_n e^{-j\Omega_n t} \sin(\omega_n t) \]
  \[ Im(y_n(t)) = a_n e^{-j\omega_n t} \sin(\omega_n t) \]

Recursion Sanity Check

- We stated that this convolution:
  \[ k_n(0) + \sum_{l=1}^{\infty} \frac{\omega_n}{\Omega_n} F(l) = a_n F(0) \]
  could be rewritten as this recursion:
  \[ k_n(0) = a_n F(0) \]
  \[ k_n(k) = e^{-j\Omega_n} k_n(k-1) + a_n F(k) \]
- Check for \(k=1\):
  \[ k_1(0) = a_n F(0) \]
  \[ k_1(1) = e^{-j\Omega_n} k_1(0) + a_n F(1) \]
  \[ k_1(2) = e^{-j\Omega_n} k_1(1) + a_n F(2) \]
  \[ k_1(3) = e^{-j\Omega_n} k_1(2) + a_n F(3) \]
  \[ k_1(4) = e^{-j\Omega_n} k_1(3) + a_n F(4) \]
  \[ k_1(5) = e^{-j\Omega_n} k_1(4) + a_n F(5) \]

Active Measurement Facility

### Real-time Computation

- Decompose recursion into real and imaginary parts
  
  \[ a_{k+1} = a_k + b_k \]  
  \[ b_{k+1} = a_k - b_k \]  
  \[ a_0 = b_0 = 0 \]  

- Define coefficients \( c_r \) and \( c_i \):
  
  \[ c_r = \frac{\pi^2}{12} \]  
  \[ c_i = \frac{\pi^2}{12} \]  

- Coefficients \( c_r \) and \( c_i \) are constants (given the model parameters) and can be pre-computed

- Thus only five multiplications and three additions are needed per time step per mode

### Haptic Forces

- Force profile comes from a haptic algorithm
  - God-object
  - Proxy
  - Implicit surfaces

- Components
  - Normal force
  - Tangential force
    - Friction model
      - Stick-slip
      - etc.

### Audio Force

- Problems with pure haptic force
  - Spurious second "hit" sound when contact is released
    - Solution: Attenuate normal force after contact is made by \( \beta \) (tuned to 0.85)
  - High frequency jitter causes static
    - Solution: Truncate force value

### Special Effects

- Scraping / Sliding
  - Multiple contact locations at irregular locations
  - Add fractal noise: noise with power spectrum proportional to \( \omega^D \) passed through reson filter
  - Fractal dimension, \( D = 2D + 2 \), seems to correlate with perception of roughness

- Rolling
  - Similar to scraping but less understood
  - Contact area small relative to ball radius, so collisions drawn out in time
  - Using same fractal model as for scraping sounds, but passed through a low-pass filter, somewhat successful

### BASS Audio Library

- Free library for playing and recording sounds
- C/C++, Visual Basic, Delphi
- Windows, Mac
- Can play streams from a file (.wav, .mp3, etc.) or from data generated in real-time
- Directly read/write data values
- [http://www.un4seen.com/](http://www.un4seen.com/)
BASS: Initialization
Add bass.lib import library to linker dependencies, and place bass.dll in path
Include header file:
#include "bass.h"
Initialize BASS and associate it with a sound card and a frequency:
BASS_Init(1,44100,0,0,0);
To check for error conditions:
printf("Error %d\n", BASS_ErrorGetCode(1));

BASS: Playing a File
Create the stream from a file (.mp3,.wav, etc.):
HSTREAM str = BASS_StreamCreateFile(FALSE, fileName, 0, 0, 0);
Start (or restart) playing the file:
BASS_ChannelPlay(str,FALSE);
Stop (pause) playing the file:
BASS_ChannelStop(str);

BASS: Writing to a Stream
Set up stream:
HSTREAM str = BASS_StreamCreate(44100,1,0,MyStreamWriter,0);
BASS_ChannelPlay(str,TRUE);
Implement callback function:
DWORD CALLBACK MyStreamWriter(HSTREAM handle, void *buf, DWORD len, DWORD user)
{
char *cb=(char*)buf;
for (unsigned int i=0; i<len; i++)
{ cb[i] = computeCurrentSoundValue();
return len;
}
Note: It is better to return less data quickly, rather than spending a long time delivering exactly the amount BASS requested.

Implementation
In initialization:
// Precompute real and imaginary parts of e^(i*omega/Sr) for each mode
for (j=0; j<n; j++)
{
cr[j] = exp(-d[j]/SAMPLE_RATE)*cos(2*PI*f[j]/SAMPLE_RATE);
ci[j] = exp(-d[j]/SAMPLE_RATE)*sin(2*PI*f[j]/SAMPLE_RATE);
}
k = 0; Beta = 0.85;
Real-time sound generation:
int computeCurrentSoundValue()
{
// Compute audio force, attenuated so that the normal force decays gradually to avoid spurious "second hit"
float audioForce = pow(Beta, k++)*m_normalForce.length() + m_tangentialForce.length();
// Clamp the amplitude
if (audioForce > 2.0) audioForce = 2.0;
// Convolve the haptic normal force with the sound modes; see the Pai papers for explanation
float total = 0.0;
for (int j=0; j<n; j++)
{
// Modal synthesis recursion
if (k == 0) {  hr[j] = audioForce*a[j]; hi[j] = 0.0; }
else
{
tr[j] = cr[j]*hr[j] - ci[j]*hi[j] + audioForce*a[j];
ti[j] = cr[j]*hi[j] + ci[j]*hr[j];
hr[j] = tr[j];
hi[j] = ti[j];
}
total += hi[j];
}
// Convert from signed float to unsigned int, and clamp
int value = (int)(total/scale) + 128;  if (value > 255) value = 255;
return value;
}

CHAI Haptic Sounds Example: Demo

CHAI Record Player Example: Demo
CHAI Record Player Example

- **Rotating the record**
  - Compute torque applied about disk axis (z-axis)
  - Get force in the plane of the disk from force computed by proxy algorithm
  - Get radius \(r\) from record center \(c\) to proxy position \(p\):
  - Compute torque about z-axis \(\tau\):
    - Compute new angular velocity from torque about z-axis
    - Rotate the record about z-axis by \(\Delta \theta\):

\[
egin{bmatrix}
  f_x \\
  f_y \\
  f_z
\end{bmatrix}
= \begin{bmatrix}
  r_x & r_y & r_z
\end{bmatrix}
\begin{bmatrix}
  f_x \\
  f_y \\
  f_z
\end{bmatrix}
\]

- **Playing the sound**
  - Load input stream data from .wav file
  - Write data to output stream forwards or backwards depending on sign of rotational velocity
  - Set frequency of the output stream depending on magnitude of rotational velocity

Record Player: Rotation Implementation

```cpp
// Relate force to torque applied about the disc axis
// Get force in the plane of the record
planeForce.set(force.x, force.y, 0);
// Get position of proxy on plane, get vector from center to application point
m_proxyPos = forceAlgo->getProxyGlobalPosition();
m_proxyPos.sub(m_recordMesh->getGlobalPos());
// Cross force and radius to get torque
planeForce.crossr(radius, torqueVec);
// This is the torque applied about the z-axis
m_torque = torqueVec.z;

// Compute rotational velocity from the torque, \( \tau = Ia \), \( dv = da \cdot dt \)
double time_step = 0.001;
m_rotVel = m_rotVel + m_torque/m_inertia * time_step;

// Set audio direction and frequency of stream based on rotational velocity
if (haptics_enabled && (fabs(m_rotVel)) > 0.0)
{
  if (m_rotVel < 0.0) record_direction = 1;
  else record_direction = -1;
  BASS_ChannelSetAttributes(stream, (int)(info.freq*fabs(m_rotVel)/6.5), -1, -1);
  if (!(BASS_ChannelPlay(stream,FALSE)))
    _cprintf("Play error %d\n", BASS_ErrorGetCode());
}
else
{
  BASS_ChannelStop(stream);
}

// Rotate object about z-axis
m_recordMesh->rotate(m_recordMesh->getRot().getCol2(), m_rotVel * time_step);
```

Record Player: Sound Implementation

In initialization:

```cpp
// Initialize and create the stream for writing
BASS_Init(1,44100,0,0,NULL);
stream=BASS_StreamCreate(info.freq,info.chans,0,&MyStreamWriter,0);

// Load the data from the specified file
HSTREAM file_stream = BASS_StreamCreateFile(FALSE,szFileName,0,0,BASS_STREAM_DECODE);
// Get the length and header info from the loaded file
stream_length=BASS_StreamGetLength(file_stream);
BASS_ChannelGetInfo(file_stream, &info);
// Get the audio samples from the loaded file
data = new char[(unsigned int)stream_length];
BASS_ChannelGetData(file_stream, data, (unsigned int)stream_length);

// Set playing to begin at the beginning of the loaded data
pos = 0;

// Callback function to write to output stream
DWORD CALLBACK MyStreamWriter(HSTREAM handle, void *buf, DWORD len, DWORD user)
{
  // Cast the buffer to a character array
  char *d=(char*)buf;

  // Loop the file when it reaches the beginning or end
  if ((pos >= stream_length) && (record_direction == 1)) pos = 0;
  if ((pos <= 0) && (record_direction == -1)) pos = (unsigned int)stream_length;

  // If record is spinning in positive direction, write requested amount of data from current position forwards
  if (record_direction == 1)
  {
    int up = len + pos;
    if (up > stream_length) up = (unsigned int)stream_length;
    for (int i=pos; i<up; i+=1) d[(i-pos)] = data[i];
    int amt = (up-pos); pos += amt; return amt;
  }
  // If record is spinning in negative direction, write requested amount of data from current position backwards
  if (record_direction == -1)
  {
    int up = pos - len;
    if (up < 0) up = 0;
    int cnt = 0;
    for (int i=pos; i>up; i-=1) d[cnt++] = data[i];
    int amt = cnt; pos -= amt; return amt;
  }
  return 0;
}
```

Resources

- Kies van den Doel’s “Sounds of Shapes” page:
- Java Audio Synthesis System (JASS) download with data files of parameters for various objects
- Dinesh Pai’s Papers:
  [http://www.cs.rutgers.edu/~dpai/papers.htm](http://www.cs.rutgers.edu/~dpai/papers.htm)
- BASS Audio Library:
  [http://www.un4seen.com](http://www.un4seen.com)