

experimental haptics

CS-277

experimental haptics

"Haptic Audio"

Christopher Sewell

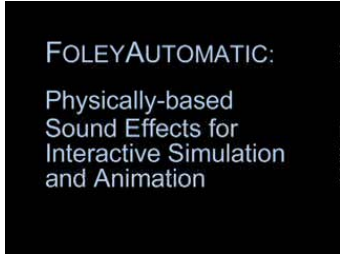
## Motivation

- 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

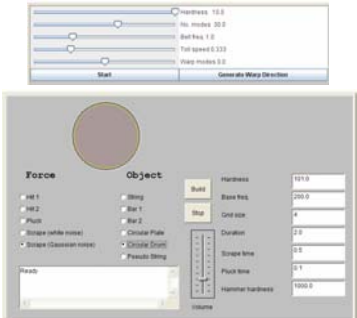
- Inspirational Videos and Demos
- Algorithm for Real-Time Interaction Sounds
- Brief Intro to BASS Audio Library
- CHAI Examples
- Resources

## The Audio and Haptic Interface (AHI)



D. DiFilippo and D. K. Pai, "The AHI: An Audio and Haptic Interface for Contact Interactions," in Proceedings of UIST'00 (13th Annual ACM Symposium on User Interface Software and Technology), November 5-8 2000, San Diego, California.

## The Sounds of Physical Shapes



K. van den Doel and D. K. Pai, "The Sounds of Physical Shapes," Presence, 7:4, The MIT Press, 1998. pp. 382-395.

## Haptic Audio Algorithm

- Modal Synthesis Model
- Obtaining Model Parameters
  - Analytically
  - Experimentally
- Convolution Sound Model with Force Profile
- Haptic and Audio Forces
- Special Effects

## Modal Synthesis Model

- The impulse response is a sum of damped sinusoids, each associated with a frequency

$$y(\mathbf{p}, t) = \sum_{n=1}^{\infty} a_n(\mathbf{p}) e^{-d_n t} \sin(\omega_n t)$$

- Each of the n angular frequencies,  $\omega_n$ , is associated with a damping coefficient  $d_n$  and an amplitude  $a_n(\mathbf{p})$  that depends on impact location  $\mathbf{p}$
- Model is derived from the solutions to the wave equation PDE

## Model Parameters

$$y(\mathbf{p}, t) = \sum_{n=1}^{\infty} a_n(\mathbf{p}) e^{-d_n t} \sin(\omega_n t)$$

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

## Analytically Obtaining the Frequency Spectrum

- Wave equation:

$$\left( A - \frac{1}{c^2} \frac{\partial^2}{\partial t^2} \right) \mu(\mathbf{x}, t) = 0$$

- $\mu(\mathbf{x}, t)$  is the deviation from the surface of a point  $\mathbf{x}$  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:

$$\mu(\mathbf{x}, t) = \sum_{n=1}^{\infty} (a_n \sin(\omega_n ct) + b_n \cos(\omega_n ct)) \Psi_n(\mathbf{x})$$

- The  $\omega_n$  values come from the eigenvalues of  $A$  ( $\lambda_n = -\omega_n^2$ ) and  $\Psi_n(\mathbf{x})$  are the corresponding eigenfunctions

## Analytically Obtaining the Frequency Spectrum : Example

- For a 2D rectangular membrane of dimensions  $L_x$  by  $L_y$ :

$$A = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \quad (\text{Laplace operator}) \quad \omega_{n_x, n_y} = \pi \sqrt{\left(\frac{n_x}{L_x}\right)^2 + \left(\frac{n_y}{L_y}\right)^2}$$

$$\mu_{n_x, n_y}(x, y, t) + \mu_{n_y, n_x}(x, y, t) = \frac{1}{c^2} \mu_{n_x, n_y}(x, y, t)$$

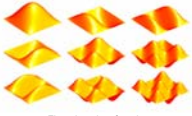
$$\Psi_{n_x, n_y}(x, y) = \sin(\pi n_x x / L_x) \sin(\pi n_y y / L_y) \quad \lambda_{n_x, n_y} = -\omega_{n_x, n_y}^2 = -\pi^2 \left( \left(\frac{n_x}{L_x}\right)^2 + \left(\frac{n_y}{L_y}\right)^2 \right)$$

Check with  $n_x = 1$  and  $n_y = 1$ :

$A\Psi_{n_x, n_y} = \lambda_{n_x, n_y} \Psi_{n_x, n_y}$  (Definition of eigenfunctions)

$$\left( \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right) \sin(\pi x / L_x) \sin(\pi y / L_y) = -\pi^2 \left( \left(\frac{1}{L_x}\right)^2 + \left(\frac{1}{L_y}\right)^2 \right) \sin(\pi x / L_x) \sin(\pi y / L_y)$$

$$= -\frac{\pi^2 \sin(\pi x / L_x) \sin(\pi y / L_y)}{L_x^2} - \frac{\pi^2 \sin(\pi x / L_x) \sin(\pi y / L_y)}{L_y^2} = \left( -\frac{\pi^2}{L_x^2} - \frac{\pi^2}{L_y^2} \right) \sin(\pi x / L_x) \sin(\pi y / L_y)$$

$$= \left( -\frac{\pi^2}{L_x^2} - \frac{\pi^2}{L_y^2} \right) \sin(\pi x / L_x) \sin(\pi y / L_y) = \left( -\frac{\pi^2}{L_x^2} - \frac{\pi^2}{L_y^2} \right) \sin(\pi x / L_x) \sin(\pi y / L_y)$$


First nine eigenfunctions

## Analytically Obtaining the Frequency Amplitudes (Timbre)

- Wave equation solution:

$$\mu(\mathbf{x}, t) = \sum_{n=1}^{\infty} (a_n \sin(\omega_n ct) + b_n \cos(\omega_n ct)) \Psi_n(\mathbf{x})$$

- Using orthogonality to solve for coefficients:

$$a_n = \int_0^s \frac{v_0(\mathbf{x}) \Psi_n(\mathbf{x})}{c \alpha_n \omega_n} d^k x \quad b_n = \int_0^s \frac{y_0(\mathbf{x}) \Psi_n(\mathbf{x})}{\alpha_n} d^k x \quad \alpha_n = \int_0^s \Psi_n^2(\mathbf{x}) d^k x$$

- With initial values assuming the object at rest ( $y_0(\mathbf{x}) = 0$ ) and velocity equal to the Dirac delta function at the distance from the point to the impact location  $\mathbf{p}$  ( $v_0(\mathbf{x}) = \delta(\mathbf{x}-\mathbf{p})$ )

$$a_n = \frac{\Psi_n(\mathbf{p})}{c \alpha_n \omega_n} \quad b_n = 0$$

- Leaving us with

$$\mu(\mathbf{p}, \mathbf{x}, t) = \sum_{n=1}^{\infty} (a_n(\mathbf{p}) \sin(\omega_n ct)) \Psi_n(\mathbf{x})$$

- By normalizing the eigenfunctions,  $a_n$  can be made independent of  $n$ . Since we only care about the relative amplitudes of the modes, we can drop  $a_n$  and  $c$

$$a_n = \frac{\Psi_n(\mathbf{p})}{\alpha_n}$$

## Twinking the Wave Equation to Get the Sound Equation

- We will not consider position of listener, so we convert displacement function  $\mu(\mathbf{p}, \mathbf{x}, t)$  to sound function  $y(\mathbf{p}, t)$ 
  - Interference from reflections mostly cancels position effect anyway

$$\mu(\mathbf{p}, \mathbf{x}, t) = \sum_{n=1}^{\infty} (a_n(\mathbf{p}) \sin(\omega_n ct)) \Psi_n(\mathbf{x}) \quad y(\mathbf{p}, t) = \sum_{n=1}^{\infty} (a_n(\mathbf{p}) \sin(\omega_n ct))$$

- The modes should decay exponentially as a function of time after impact by adding a damping factor  $e^{-d_n t}$ 
  - The internal friction  $\phi$  depends on the material and is assumed invariant over the object  $d_n = \omega_n c \tan(\phi) / 2$
  - Higher frequencies decay more rapidly
- Set a cut-off frequency; only use first  $n_f$  modes
  - Quality degrades gracefully as fewer modes are used, so  $n_f$  can be decreased dynamically when computational resources are limited
- Thus, we have

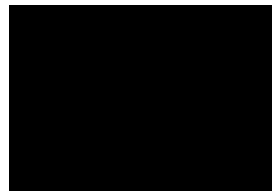
$$y(\mathbf{p}, t) = \sum_{n=1}^{n_f} a_n(\mathbf{p}) e^{-d_n t} \sin(\omega_n ct)$$

## 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.)



## Active Measurement Facility



D. K. Pai, K. van den Doel, D. L. James, J. Lang, J. E. Lloyd, J. L. Richmond, S. H. Yau, "Scanning Physical Interaction Behavior of 3D Objects," in Computer Graphics (ACM SIGGRAPH 2001 Conference Proceedings), August 2001.

## Sound Map

$$y(t, \mathbf{p}) = \sum_{n=1}^{n_f} a_n(\mathbf{p}) e^{-d_n t} \sin(\omega_n c t)$$

- $\omega_n$  and  $d_n$  do not depend on location of impact;  $a_n(\mathbf{p})$  does
- Create a map of  $a_n$  values over the surface (like a texture map)
- For a given  $\mathbf{p}$ , 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"

## Notational Changes

- Sound equation for a specific mode  $n$  and for a specific impact position  $\mathbf{p}$  (dropping  $\mathbf{p}$  as a variable and dropping the summation of modes):

$$y_n(t) = a_n e^{-d_n t} \sin(\omega_n c t)$$

- For notational convenience, absorb  $c$  into  $\omega_n$  and rewrite as the imaginary part of complex wave form with complex frequencies  $\Omega_n = \omega_n + i d_n$

$$y_n(t) = a_n e^{i \Omega_n t}$$

$$y_n(t) = a_n e^{i(\omega_n + i d_n)t}$$

$$y_n(t) = a_n e^{i \omega_n t - d_n t}$$

$$y_n(t) = a_n e^{-d_n t} e^{i \omega_n t}$$

$$y_n(t) = a_n e^{-d_n t} (\cos(\omega_n t) + i \sin(\omega_n t))$$

Euler's Formula:  $e^{ix} = \cos(x) + i \sin(x)$

$$\text{Im}(y_n(t)) = a_n e^{-d_n t} \sin(\omega_n t)$$

## Convoluting Impulse Response with Force Profile

- Discrete convolution of functions  $f$  and  $g$  at time step  $k$ :

$$(f * g)(k) = \sum_{l=0}^k g(k-l) f(l)$$

- With  $f$  as the force profile  $F(t)$  and  $g$  as  $y_n(t)$ , the discretized sound function for mode  $n$ , with  $S_R$  the sample rate

$$(F * y_n)(k) = h_n(k) = \sum_{l=0}^k a_n e^{-\frac{\Omega_n}{S_R}(k-l)} F(l)$$

- As a recursion

$$h_n(0) = a_n F(0)$$

$$h_n(k) = e^{-\frac{\Omega_n}{S_R}} h_n(k-1) + a_n F(k)$$

- Intuition: At each time step, all previous contributions are decayed by a factor of  $e^{-\frac{\Omega_n}{S_R}}$ , and contribution of most recent force has not yet decayed

## Recursion Sanity Check

- We stated that this convolution:

$$h_n(k) = \sum_{l=0}^k a_n e^{-\frac{\Omega_n}{S_R}(k-l)} F(l)$$

could be rewritten as this recursion:

$$h_n(0) = a_n F(0)$$

$$h_n(k) = e^{-\frac{\Omega_n}{S_R}} h_n(k-1) + a_n F(k)$$

- Check for  $k=1$ :

Equal	{	$h_n(0) = \sum_{l=0}^0 a_n e^{-\frac{\Omega_n}{S_R}(0-l)} F(l) = a_n e^{-\frac{\Omega_n}{S_R}(0-0)} F(0) = a_n F(0)$	Computed with original equation
		$h_n(1) = \sum_{l=0}^1 a_n e^{-\frac{\Omega_n}{S_R}(1-l)} F(l) = a_n e^{-\frac{\Omega_n}{S_R}(1-0)} F(0) + a_n e^{-\frac{\Omega_n}{S_R}(1-1)} F(1) = a_n e^{-\frac{\Omega_n}{S_R}} F(0) + a_n F(1)$	
		$h_n(1) = e^{-\frac{\Omega_n}{S_R}} h_n(0) + a_n F(1) = e^{-\frac{\Omega_n}{S_R}} (a_n F(0)) + a_n F(1) = a_n e^{-\frac{\Omega_n}{S_R}} F(0) + a_n F(1)$	
		}	Computed with recursion

## Real-time Computation

- Decompose recursion into real and imaginary parts

$$h_i(k) = e^{-\frac{\omega_c}{S_r}} h_i(k-1) + a_r F(k)$$

$$h_i(k) = e^{-\frac{\omega_c}{S_r}} h_i(k-1) + a_r F(k)$$

$$h_i(k) = e^{-\frac{\omega_c}{S_r}} h_i(k-1) + a_r F(k) \quad P \neq -1$$

$$h_i(k) = e^{-\frac{\omega_c}{S_r}} \left( \cos\left(\frac{\omega_c}{S_r}\right) + j \sin\left(\frac{\omega_c}{S_r}\right) \right) \left( \text{Re}(h_i(k-1)) + j \text{Im}(h_i(k-1)) \right) + a_r F(k) \quad \text{Euler's Formula: } e^{j\theta} = \cos(\theta) + j \sin(\theta)$$

- Define coefficients  $c_r$  and  $c_i$ :  $c_r = e^{-\frac{\omega_c}{S_r}} \cos\left(\frac{\omega_c}{S_r}\right)$   $c_i = e^{-\frac{\omega_c}{S_r}} \sin\left(\frac{\omega_c}{S_r}\right)$

$$h_i(k) = (c_r + j c_i) (\text{Re}(h_i(k-1)) + j \text{Im}(h_i(k-1))) + a_r F(k)$$

$$h_i(k) = c_r \text{Re}(h_i(k-1)) - c_i \text{Im}(h_i(k-1)) + j (c_i \text{Re}(h_i(k-1)) + c_r \text{Im}(h_i(k-1))) + a_r F(k)$$

$$\text{Re}(h_i(k)) = c_r \text{Re}(h_i(k-1)) - c_i \text{Im}(h_i(k-1)) + a_r F(k)$$

$$\text{Im}(h_i(k)) = c_i \text{Re}(h_i(k-1)) + c_r \text{Im}(h_i(k-1))$$

- 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

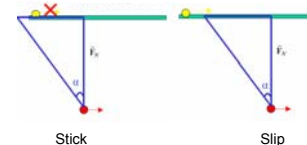
- God-object
- Proxy
- Implicit surfaces

- Components

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

$$\mathbf{f}_N = \frac{\mathbf{f} \cdot \mathbf{n}}{\|\mathbf{n}\|^2} \mathbf{n}$$

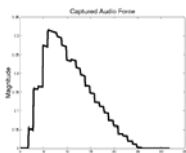
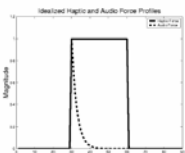
$$\mathbf{f}_T = \mathbf{f} - \mathbf{f}_N$$



## Audio Force

- Problems with pure haptic force

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



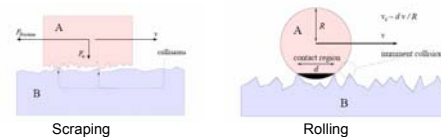
## Special Effects

- Scraping / Sliding

- Multiple contact locations at irregular locations
- Add fractal noise: noise with power spectrum proportional to  $\omega^\beta$  passed through reson filter
- Fractal dimension,  $D = \beta/2 + 2$ , 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

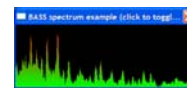
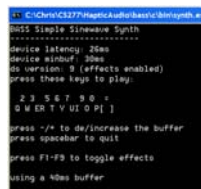
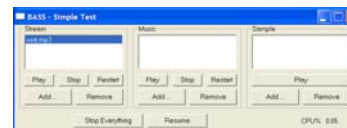


## BASS Audio Library

- 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/>



## BASS: Demos



## 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());
```

## 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



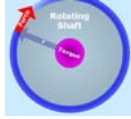
## CHAI Record Player Example

- Rotating the record
  - Compute torque applied about disk axis (z-axis)
    - Get force in the plane  $f_{plane}$  of the disk from force  $f$  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
 
$$\tau = \tau_z \quad \tau = I \dot{\omega}_{rot} \quad \Delta v_{rot} = a_{rot} \Delta t \quad \Delta v_{rot} = \frac{\tau}{I} \Delta t$$
  - Rotate the record about z-axis by  $\Delta\theta$ :
 
$$\Delta\theta = \Delta v_{rot} \Delta t$$
- 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

$$f_{plane} = \begin{bmatrix} f_x \\ f_y \\ 0 \end{bmatrix}$$

$$r = \begin{bmatrix} p_x - c_x \\ p_y - c_y \\ 0 \end{bmatrix}$$

$$\tau = f_{plane} \times r$$



## Record Player: Rotation Implementation

```
// Relate force to torque applied about the disc axis
// Get force in the plane of the record
cVector3d planeForce;
planeForce.set(force.x, force.y, 0);

// Get position of proxy on plane, get vector from center to application point
cProxyPosInPlane forceAlgo = tool->getProxy();
m_proxyPos = forceAlgo->getProxyGlobalPosition();
m_proxyPos.set(m_recordMesh->getGlobalPos());
cVector3d radius;
radius.set(m_proxyPos.x, m_proxyPos.y, 0);

// Cross force and radius to get torque
cVector3d torqueVec;
planeForce.cross(radius, torqueVec);

// This is the torque applied about the z-axis
m_torque = torqueVec.z;

// Compute rotational velocity from the torque, tau = I*alpha, dv = da*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 (fabs(m_rotVel) > 0.0)
{
    if (m_rotVel < 0.0) record_direction = -1;
    else record_direction = 1;
    BASS_ChannelSetAttribute(stream, (int)(info.freq*fabs(m_rotVel)/6.5), -1, -1);
    if (!BASS_ChannelPlay(stream, FALSE))
        _print("Play error %d\n", BASS_ErrorGetCode());
}
else
{
    BASS_ChannelStop(stream);
}

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

## Record Player: Sound Implementation

```
In initialization:
// 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
STREAM file_stream = BASS_StreamCreateFile(FALSE,srcFileNew,0,0,BASS_STREAM_DECODE);

// Get the length and header info from the loaded file
stream_length=BASS_StreamLength(file_stream);
BASS_ChannelGetInfo(file_stream, info);

// Get the audio samples from the loaded file
data = new char[(unsigned) info.stream_length];
BASS_ChannelGetData(file_stream, data, (unsigned) info.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) info.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) info.stream_length;
        for (int i=pos; i<up; i++) 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 amt = 0;
        for (int i=pos; i>=up; i--) d[amt++] = data[i];
        int amt = amt; pos -= amt; return amt;
    }

    return 0;
}
```

## Resources

- Kies van den Doel's "Sounds of Shapes" page: [http://www.cs.ubc.ca/~kvdoel/sound\\_shapes.html](http://www.cs.ubc.ca/~kvdoel/sound_shapes.html)
- Java Audio Synthesis System (JASS) download with data files of parameters for various objects <http://www.cs.ubc.ca/~kvdoel/jass/jass.html>
- Dinesh Pai's Papers: <http://www.cs.rutgers.edu/~dpai/papers.htm>
- BASS Audio Library: <http://www.un4seen.com>