The Sound Group

Joe Bota
Aaron Camm
Alex Cueto

Brief Overview

Of Our Presentation

- The Physics of Sound
- Audio Formats
- Music and Dynamic Audio

The Physics of Sound

Joe Bota

Properties that Affect Sound Propagation

- Attenuation
- Reflection
- Diffusion
- Absorption
- Refraction
- Diffraction

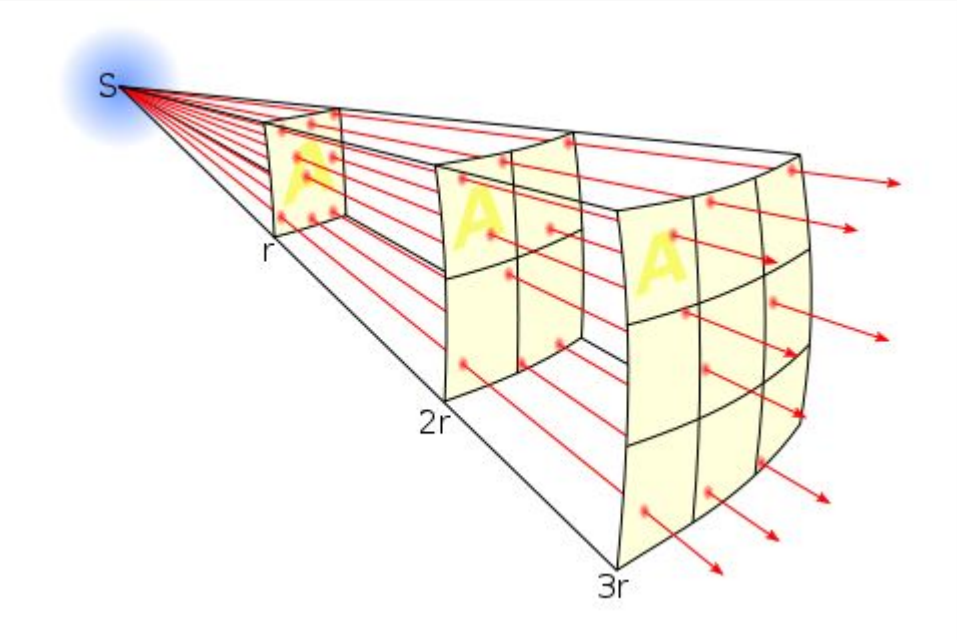
Attenuation

- This is the natural rate at which sound decays in strength.
- The simplest attenuation of sound can be represented by the Inverse-Square Law.

*Intensity = power / (4 * π * d ^ 2)*

power = power of sound source

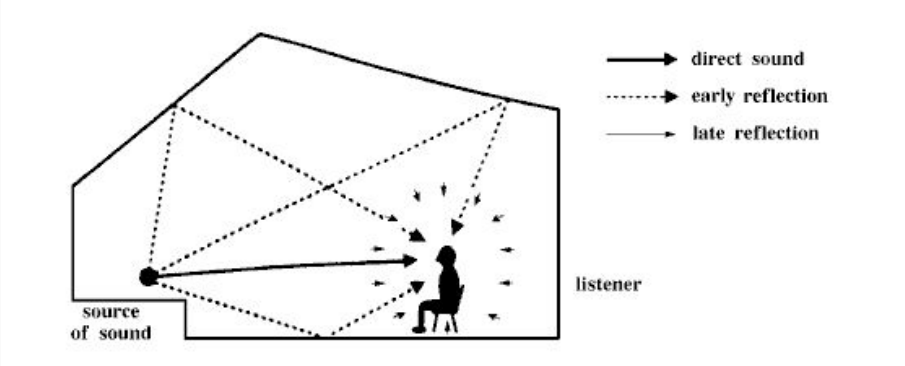
d = distance traveled



Reflection

- The degree and angle at which sound is deflected off of an obstacle.
- Can focus sound onto a central focal point or cause echoing.

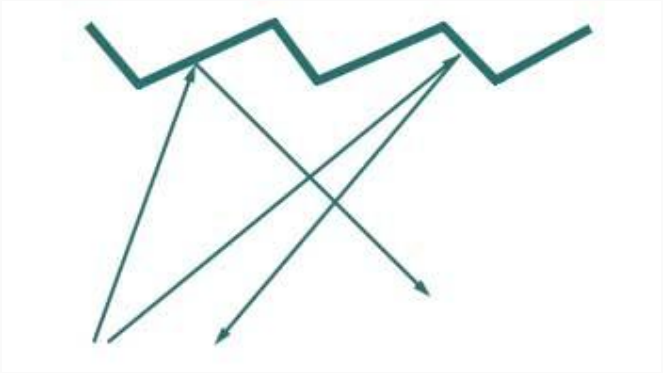
(eg. Smooth surfaces like marble have high reflection.)



Diffusion

- When a surface is not perfectly reflective, diffusion will occur, scattering the sound waves.
- This will weaken and obfuscate sound.

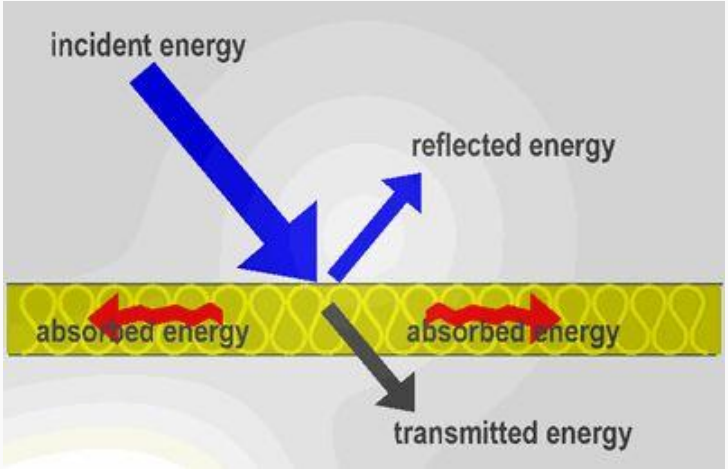
(eg. Rocky surfaces with crags and rough edges cause sounds to diffuse.)



Absorption

- Sometimes, objects can absorb sound, causing them to lose energy and decay faster.
- This also causes a sound's strength to decay faster.

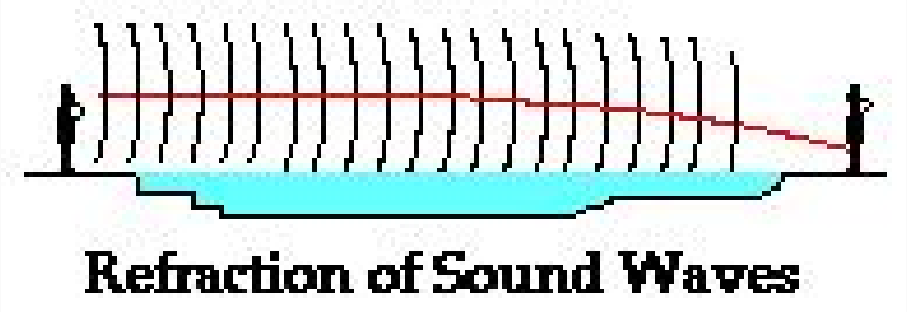
(eg. Foam has high absorption.)



Refraction

- Bending of sound when passing through an uneven medium.
- Very rare with sound.

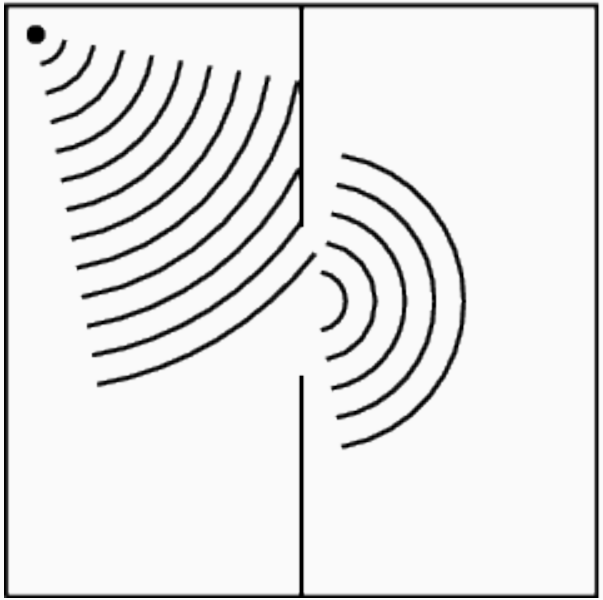
(eg. Sound passes over a lake, thus bending it towards the shore because of the water below.)



Diffraction

- The “shadow” that a sound casts when passing by an obstacle.
- A sound has reduced volume when not in the direct line of sight.

(eg. A sound wraps around an object in order to reach the listener, thus resulting in the sound seeming further away.)



Basically, any game where immersion and sound are integral to the experience.

Metal Gear Solid

Assassin's Creed

P. T.

Games that use some degree of sound propagation.

A “Ubisoft-esque” Approach to Sound Propagation

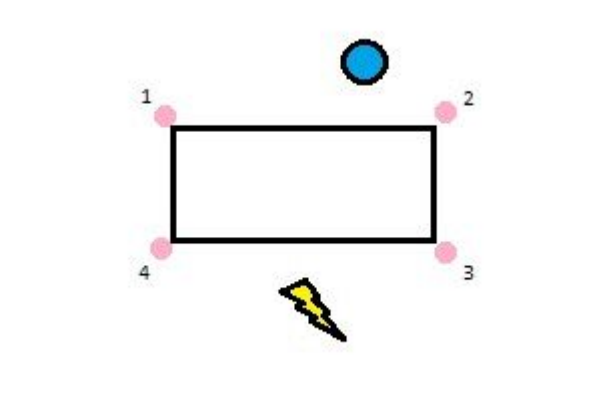
This is an adaptation of a technique discussed at GDC 2012 by Jean-Francois Guay of Ubisoft Montreal. It is absent of some optimizations for the sake of simplicity.

- Uses attenuation and graph theory to emulate diffraction.
- Relatively low cost, even without optimizations.
- Can be implemented relatively easily into Unity.

UBISOFT®

Breakdown of the Methodology

- Populate the environment with nodes at each corner of geometry.
- Generate a graph between all nodes where edges do not collide with geometry
- Calculate the shortest distance between each node using an algorithm like Dijkstra's algorithm.
- During gameplay, when a sound is emitted out-of-sight of the player, find all line-of-sight vertices near player and sound, and determine shortest path to each other.
- Use this distance to calculate intensity of sound and then emit sound from the vertex closest to the player from within the shortest path.



A Rough Implementation

- **IsInRangeOf** functions calculates whether a node can be reached directly via Raycast. This is what makes it valid.
- **TotalDistance** takes two indices of nodes and retrieves their distance along with each nodes' distance from the player and sound source.
- **activePlayerNode** is the node in which a sound will ultimately emit. It is effectively the closest node to the player along the graph.
- **PlayAudio** plays a sound bit from the activePlayerNode and uses the closestDistance to determine the sound's volume.

```
45      private void CalculateActivePlayerNode() {
46          // Arrays with valid nodes to the sound source and player
47          bool[] inRangeOfSource = new bool[nodes.Length];
48          bool[] inRangeOfPlayer = new bool[nodes.Length];
49          float closestDistance = outOfRange;
50
51          // Find all valid nodes
52          for (int i = 0; i < nodes.Length; i++) {
53              inRangeOfSource[i] = IsInRangeOfSource (nodes[i]);
54              inRangeOfPlayer[i] = IsInRangeOfPlayer (nodes[i]);
55          }
56
57          // Cycle through pairings until the smallest distance
58          // is found between the source and the player.
59          for (int i = 0; i < nodes.Length; i++) {
60              float temp = outOfRange;
61              if(inRangeOfSource[i]) {
62                  for (int j = 0; j < nodes.Length; j++) {
63                      if(inRangeOfPlayer[j]) {
64                          temp = TotalDistance (i, j);
65                      }
66
67                      if(temp < closestDistance) {
68                          closestDistance = temp;
69                          activePlayerNode = nodes[j];
70                      }
71                  }
72              }
73          }
74
75          PlayAudio (closestDistance);
76      }
```

More Code Snippets

- **nodeDistances** is the distance between two nodes, given their indices. This is meant to be calculated upon the room's instantiation.
- **IsInRangeOfSource** merely says whether a node can be reached without obstruction.

```
78      private float TotalDistance(int i, int j) {
79          float distanceToSource = Vector3.Distance (gameObject.transform.position, nodes[i].transform.position);
80          float distanceToPlayer = Vector3.Distance (player.transform.position, nodes[j].transform.position);
81
82          return distanceToSource + distanceToPlayer + nodeDistances[i,j];
83      }
84
85      private bool IsInRangeOfSource(GameObject node) {
86          gameObject.transform.LookAt (node.transform);
87          RaycastHit hit;
88          float distance = Vector3.Distance (gameObject.transform.position, node.transform.position);
89          Physics.Raycast (gameObject.transform.position, gameObject.transform.forward, out hit, distance);
90
91          return hit.collider.gameObject.Equals (node);
92      }
```

Quick Demo

- The formula for attenuation is linear instead of Inverse-Square, for the sake of simplicity.
- Does not account for height. This could be fixed by implementing some of the optimizations specified in Ubisoft's powerpoint.

A toolkit that does this much better than from scratch can be found here:

https://www.assetstore.unity3d.com/en/#!/content/40200

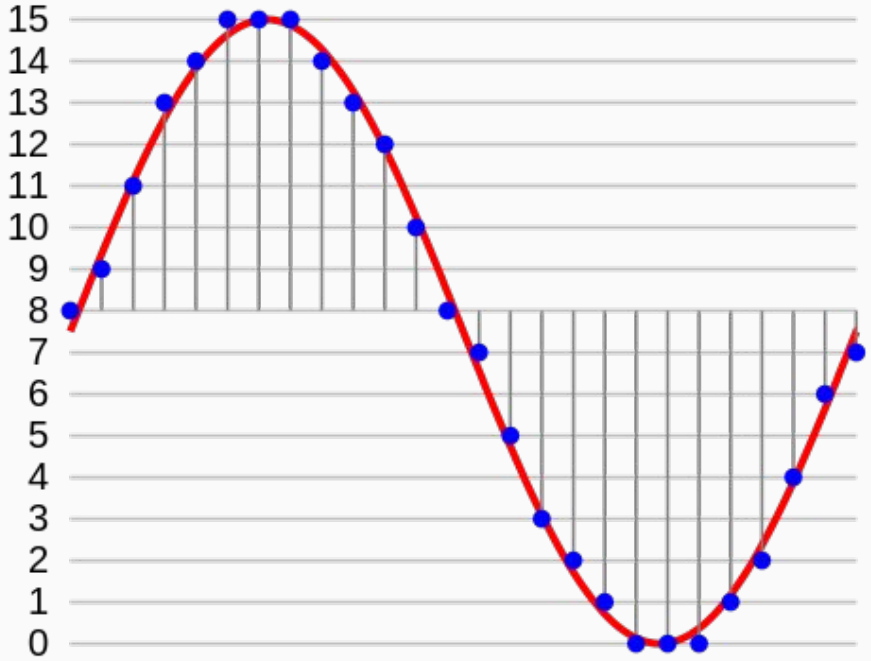
(Unfortunately, it costs $25.)

Audio Encoding

Aaron Camm

Audio Encoding - Sampling

- Majority of digital recording system work in the similar matter
  - Utilizes a system called Pulse Code Modulation (PCM)
- Audio signals is fed into an Analog-to-digital converter
  - Takes measurements of audio signal at regular intervals
  - stores each measurement as a number
- Measured data is sent to a Digital to Analog converter
  - Converter recreates audio signal from measured data that was recorded



Audio Encoding - Sampling

- Problem with storing PCM measurements
  - Aims to be the most accurate reproduction of audio
  - Includes Audio data that humans can't hear

- Results in a very large size
  - a 'CD - quality' audio (16 - bit, 44100 Hz sample) would be 10 mb per minute.
  - The problem of file size is solved through an encoding method.

Audio Encoding- Types

- Audio is Encoded in three main types.
  - Lossless Uncompressed Encoding
    - Ex: LPCM, PCM
  - Lossless Compressed Encoding
    - No Sound information is loss
    - Ex: FLAC, ALAC
  - Lossy Compressed Encoding
    - Irrelevant sound information is discarded.
      - Utilizes Psychoacoustic
    - Ex: MP3, Vorbis

Audio Encoding - Lossy Compression

- Removes Sound information that Human Ears' can't distinguish.
- Human can't hear sound below 20Hz or above 20kHz, approximately.
  - Frequencies between 1 to 5 kHz are best perceived at low volumes.
- Removing sound information that is being masked by another signal.
  - when a loud signal occurs, other signals close in frequency or time are difficult to hear.

Audio Encoding - Lossy Compression (MP3)

- Separates a PCM waveform measurements in frames,
  - usually 576 samples each frame.

- Each frame is divided into a bandpass filters to set of 576 frequency ranges
  - These frequency bands are between 20 Hz and 20000 Hz, approximately

Audio Encoding - Lossy Compression (MP3)

- Performs Fast Fourier Transforms
  - check to see if any sounds are at a close frequency to louder sounds. (Masking)

- Modified Discrete Cosine Transform
  - frames are sorted based on their different "window" patterns (steady or constant)
    - each steady pattern are described via 3 short windows (each 192 samples)
    - each constant pattern is described with 1 long window (each 576 samples)
    - each window is then turned into a set of spectral values
      - each value represent the energy across the range of frequencies.

Audio Encoding - Lossy Compression (MP3)

- Compresses the values through use of Quantization and Huffman coding.
  - 576 post-MDCT frequency bands are sorted to 22 scalefactor band
    - divides each by a quantizer and rounding
    - whether a band is rounded up or down is determined by the FFTs
    - the lower the scale is, the less space it need
  - While quantizing, uses information in scalefactor to point to a shorter variable length binary string (Hoffman Code)
    - shorter numbers are for less precise, and quickly taken from Huffman Tables
    - The shorter binary string are used for the construction of the encoding

Audio Encoding - Lossy Compression (MP3)

Simple Waveform of Chirp Sound from 20kHz to 22kHz, size is 2.5 MB in PCB encoding

Same sound as above, encoded in MP3, size is 469 KBs

Making Sweet Music...

...except not really, because we’re not composers

VG Music History in a Nutshell

- Chiptunes & PC Speakers
  - Sequence audio - written in hex
  - Limited sound channels led to distinctive sounds
  - Relied on simple melodies
- MIDI and Redbook
  - MIDI is sequence audio: The Next Generation
    - Synthesized instruments controlled by keyboard
    - Standardized equipment and software standards
  - Redbook is the CD-audio standard
    - Sounds like recorded audio because it is
    - Early redbook audio in games difficult to sync and doesn't loop - just plays off disc
- Modern Digital Audio
  - Can be sequence or recorded
  - Modern compression and digital audio recording means we don't have to rely on Redbook for recorded music
  - Easy to edit, sync, and work with

Putting Music Into Unity

- AudioSources
  - Attach to gameObjects
  - Can have multiple on each, Unity treats like an array
  - A single AudioSource cannot string together AudioClips without a gap
- AudioClips
  - Played by AudioSources
  - One AudioSource can play any number of AudioClips, but not simultaneously

Changing Properties of Audio

- Properties such as volume and pitch can change based on several factors as determined by the programmers

```
void Update () {
    land.pitch = 1.0f - (player.transform.position.x + 6.0f) / 4.0f;
    water.pitch = 1.0f - (player.transform.position.x + 6.0f) / 4.0f;


    playercontroller ps = player.GetComponent<playercontroller>();
    if (ps.isUnderwater() == true && water.volume == 0.0f) {
        land.volume = 0.0f;
        water.volume = 1.0f;
    }
    else if (player.transform.position.y > 2) {
        float newvol = 1.0f - ((player.transform.position.y - 2.0f) / 4.0f);
        land.volume = newvol;
    }
    else if (ps.isUnderwater() == false && land.volume == 0.0f) {
        water.volume = 0.0f;
        land.volume = 1.0f;
    }
}
```

Changing Music in a Scene

- An AudioSource can stop playing a clip and play a new one, but there will be a gap
  - This may change in later versions of Unity
- Multiple AudioSources can be used with crossfading for clean transitions
- For ongoing seamless transitions, two simultaneous AudioSources can be used
  - Dependent on sound editing for synchronization

Dynamic Sound Demo

- Volume and pitch changes
- Use of two simultaneous AudioSources on MusicManager object
- Used in games such as Super Mario 3D Land and the bit.trip series

What if we could do more...

Adaptive Music

- The very flow of the music changes based on gameplay events
- The songs change or even the components of a single song are rearranged
- Requires careful sound editing and use of in-game triggers
- In Unity, two AudioSources are used for clean transitions since there are more of them than with longer single clips

Fancy Adaptive Music Demo

- How it works
- How can we make it even better?
- This was a simple demo, what does this mean for finished saleable games?

Questions?

Thank You!

References

- Physics of Sound
  - http://gdcvault.com/play/1015492/Real-time-Sound-Propagation-in
- Audio Formats
  - http://www.soundonsound.com/sos/may00/articles/mp3.htm
  - http://arstechnica.com/features/2007/10/the-audiofile-understanding-mp3-compression (and its slides)
- Music and Dynamic Audio
  - http://vgsoundtest.blogspot.com/2013/04/dynamic-music-demo-rooftop-run.html (video no longer available, sadly)
  - https://www.youtube.com/watch?v=CKgHrz_Wv6o (Extra Credits)
  - http://forum.unity3d.com/threads/dynamic-music-in-unity-examples.297631/ (Unity forum, many good informational links)