Interactive Sound, Environment, and Music Design for a 3D Immersive Video Game

By

David Parfit

August 17, 2005
New York University
New York, NY 10012

Submitted to the Faculty of New York University in partial fulfillment of the requirements for the degree of Master of Music in Music Technology
Abstract ................................................................................................................................. 2

Part I: Video Game Sound Design and *Orange*  3

The State of Game Audio ........................................................................................................ 3
Types of Game Play: 3D Immersive Environments and Abstracted Environments  3
Audio in a 3D First Person Game ......................................................................................... 5

Part II: Acoustic Modeling and Video Games  8

Basic acoustic principles ........................................................................................................ 8
Acoustic Modeling Implementation in Video Games ............................................................. 12
Example: modeling the reflections of a virtual church .......................................................... 16
  The Acoustic Model .............................................................................................................. 16
  Early Reflection Modeling .................................................................................................. 22
  Design .................................................................................................................................. 24
  Early Reflections Implementation ....................................................................................... 25
  Late Reflections .................................................................................................................. 27
  Limitations ......................................................................................................................... 28

Part III: Interactive Game Music  29

An Interactive Music System .................................................................................................. 30
  The Technique .................................................................................................................... 31
  Implementation (this section assumes basic knowledge of Max/MSP) ............................... 33
  Conclusions and Limitations ............................................................................................. 38

Conclusions ............................................................................................................................ 39

References ............................................................................................................................. 40
Abstract

Sound has been an integral part of video games since their inception, constantly cuing players to their virtual actions and surroundings and providing emotional guidance with music. As with graphics capabilities, processing power, and media storage, audio technology has advanced at an astonishing rate, in part due to the expansion and desires of the gaming world. Today, it is now possible to enhance a player’s aural experience to the point of complete immersion.

Among the primary concerns of game audio developers is the ratio of game play hours to the total hours of audio and music. With the extremely large amount of time a person will spend playing a single game, and the nature of sample-based game audio and music, repetition becomes a major obstacle blocking game enjoyment.

This paper is a study of game audio and music development techniques aimed at player/game interactivity, realistic sound environment implementation, and removing the repetitive nature of game sound and music. Examples of implementation are taken from the sound design and music – created by myself – for the game Orange – created by Joey Silayan. Orange was developed using the development environment Virtools Dev 3.0 and the 3D modeling software Maya.

The introduction and Part I provide a brief game audio background and knowledge base as well as sound design implementation techniques for the reduction of repetition in game audio – focusing on Orange audio implementation.

Part II of this paper describes, in detail, the Virtools implementation of a simple acoustic model of a virtual church. This example presents a viable dynamic reverberation scheme based image source reflection modeling.

The final section, Part III, outlines an interactive music system designed to extend the effective playback time of a given amount of music, and adapt to the game as the emotional environment changes. This generic system is implemented in Max/MSP as a music system proof-of-concept.
Part I: Video Game Sound Design and Orange

The State of Game Audio

Many factors contribute to the audio and music quality in a game, and unfortunately, quite a number of them are purely economic. As the gaming industry picks up speed (estimated revenue was $28 billion in 2004, with a record $7.3 billion in sales in the U.S. alone [24] and is projected to reach $36 billion by 2006 [25]), more attention and resources are given to game audio and music. It is in this recent explosion that composers and sound designers have found new careers in the gaming industry. Game audio developers are now some of the best sound designers in the world. Also with this push comes interdisciplinary exploration in both the academic and commercial worlds of audio technology resulting in a plethora of new technology developed primarily for gaming. Currently, these technologies tend to focus on the battle for a transparent aural experience and sonic variety.

As the game audio community has grown, so have the resources. Currently there exist a few international groups devoted to game audio, of which the most established are GANG (Game Audio Network Guild) and IASIG (Interactive Audio Special Interest Group) [33,39]. These groups provide knowledge dissemination, guidance, and research resources to an otherwise proprietarily dominated field.

Types of Game Play: 3D Immersive Environments and Abstracted Environments

The number of game styles, story lines, and game play methods currently on the market rivals the number of descriptions the Inuit have for snow. PCGameWorld.com
piles games under genres such as action, adventure, driving/racing, fighting, online, 
pinball, puzzle, role-playing, simulation, sports, and strategy; but this list is subjective at 
best. More important for game audio than the genre is the style of play. Is the game a 
First Person Shooter (FPS) or is the player looking at the player-character from above? Is 
the player-character in the driving game simulated as behind the wheel, or is the entire 
scene viewed from behind? Does the game immerse the player into a completely 
artificial environment? These questions all have one element in common: where is the 
listener in relation to the sound sources?

An immersive environment suggests the player is simulated as though he or she is 
inside the TV or PC monitor – inside the game, if you will. In a game such as this (a first 
person game), the audio should reach the player realistically – ideally taking into account 
the acoustic properties of the environment as well as the binaural reception and 
perception characteristics of human ears. These sound propagation characteristics 
include attenuation due to distance and air, reflection, obstruction and occlusion 
(diffraction), and sound source modeling. These topics are discussed in Part II.

Abstracted audio environments – as opposed to immersive sonic environments – 
occur in a third person game, when the player (i.e. the listener) is not peering out from the 
eyes of a player-character. These games can be either simulated 3D (not realistic 
dimensional ratios) or some type of 2D. Obviously, these characteristics impact the 
audio development drastically, although quality and non-repetition are still of utmost 
importance. This study focuses on audio implementation in an immersive 3D 
environment.
In both abstracted and immersive game audio environments, audio may or may not correspond to realistic events. In other words, there is a certain degree of realism that every event has, and the type of sound effect should reflect this. For example, footstep sounds tend to need a realistic implementation in a game while sounds pertaining to the use of magic are afforded more freedom.

**Audio in a 3D First Person Game**

For convenience purposes, audio can be split into two functional groups: environmental, or ambient sounds, and reactive audio. Ambient sounds are audio such as wind, birds, background city sounds, etc… any sounds that are part of the environment that don’t necessitate player interaction – other than orientation. Reactive audio constitutes all sounds that are directly related to the player’s actions such as gunshots, footsteps, and sword swipes. These labels are not always mutually exclusive, but they provide a useful descriptive designation.

The primary characteristic of an ambient sound is its perceptual location in three-dimensional space. A simple technique of projecting the location of a sound source sonically is to pan the signal to the appropriate place in the spatial spectrum and reduce the sound intensity at a rate following the inverse square law as the distance to the sound source increases. This, however, doesn’t produce an accurate binaural experience for the listener. Alternatively, the sound source signal could be convolved with a Head-Related Transfer Function (HRTF). An HRTF is the frequency domain representation of a Head-Related Impulse Response (HRIR). An HRIR, then, is the impulse response of an eardrum. An HRIR is specific to the eardrum, however, and the technique of
measurement is somewhat tricky. Therefore, an HRTF approximating the average human’s eardrum is necessary, and in fact, general HRIRs have been measured (and thus HRTFs as well).

Applying an HRTF to a signal is accomplished via convolution. Direct convolution is achieved with a Finite Impulse Response (FIR) filter [35] (filter coefficients are readily available for measured HRIRs). Alternately, transforming the input signal to the frequency domain via an FFT reduces convolution to complex multiplication of the input signal and the HRTF. The efficiency of frequency domain convolution versus that of direct convolution (using the fastest techniques currently available) is much greater for large impulse responses \(O(N\log_2 N)\) for frequency domain methods and \(O(N^2)\) for direct convolution; \(N = \text{length of impulse response, in samples}\). However, FIR filters are easily implemented in hardware, and as the current primary implementation media is indeed hardware, FIR filters provide a valid implementation solution. HRTFs are generally implemented in hardware as an element of a sound output device. On a PC running Windows, DirectSound3D contains an interface to HRTF technology implemented as hardware.

In addition to the position of each sound, the generation scheme or playback method must be defined. In other words, is the sound to be played from a file and looped, or synthesized on the fly? The simple solution, with indifference to repetition, is to stream a looped sample from a file on disk. In a system where storage space is a greater issue than CPU power, a granular synthesis system will produce less repetitive results, with the potential cost being realism.
Granular synthesis, in this case, perhaps should be called sound splicing or effected sound splicing. It is not the technique – familiar to computer musicians – of using sound “grains” smaller than $1/20^{th}$ of a second that, when repeated, play faster than the audible audio rate of roughly 20Hz. Granular synthesis, as I describe here, is the process of placing numerous repetitions of one or two samples next to each other, adjusting various characteristics such as pitch, duration, and frequency content. This technique has proven successful at modeling ambient sounds such as wind, birds chirping, and waves on a beach. As the size of this topic is enormous, it is out of the scope of this study.

Creating a realistic 3D audio environment is analogous to a session in front of a mixing console; every sound has its place in the mix, spatially as well as in the frequency spectrum while maintaining their natural characteristics. Ambient sounds, as I have described, are only a slice of this whole; reactive sounds also require space.

Reactive sounds have the potential to be triggered many times in short succession. Footsteps, for instance, are basically continuous. This characteristic can severely annoy a player if it isn’t tackled correctly. The simple solution is to use a large set of samples and create a semi-random weighted selection system. However, most game platforms are limited in resources; and more efficient techniques are necessary.

One solution, which I implemented in the game Orange, was constructed as follows. A set of five footstep samples was used for each surface. Each sample was split into three parts: the initial heel impact, the middle scuff, and the cadential scraping of the toe breaking contact. Each part was then placed into a separate array. When a footstep event occurs, a sample is selected at random from each part array, pieced together on the
fly, and played. With five separate samples, using the method just described, the probability of triggering the same sample twice in a row is reduced from 1/5 to 1/125.

More continuous sounds, such as gunshots and random synthesized sounds, require slightly different approaches. A method similar to the footstep engine described above is suitable, but instead of breaking a sound into three temporally disparate chunks, each sound is constructed as three or more layered elements that mesh to create a single sound. Again, multiple versions of each layer are placed into arrays and selected at random for playback. In addition, attributes of each layer, such as pitch and timbre, can be altered randomly to produce a dynamic, repeatable sound.

A third repetition-reducing method of sound creation is physical modeling, or the creation and use of a mathematically modeled acoustic sound generation device that reacts to predefined stimuli such as plucking and striking. Unfortunately, physical modeling has proven to be a difficult and impractical solution for most game sound design problems. The acoustic complexity of most virtual audio sources in video games leads to overwhelmingly computationally expensive physical models. In addition, most physical models do not sound as realistic as is necessary. In the future, however, physical modeling may become a very interesting sound design addition to game audio.

Part II: Acoustic Modeling and Video Games

Basic acoustic principles

It is imperative that one must first understand the basic physical laws sound obeys in order to understand their approximate implementation in games. The science of sound
wave propagation is well understood and is the cornerstone of many 3D sound environment implementations.

A fundamental characteristic of all sound is its attenuation due to distance and air. The relationship of sound intensity to distance obeys the inverse square law. I.e. at distance \( x \) from an arbitrary sound source of intensity \( I \), the intensity of the sound is perceived as \((1/x^2)I\). This is the case because sound radiates outward from a source in a spherical pattern. In addition to intensity reduction due to distance, the air pressure and wind velocity affect sound intensity.

As sound propagates through an environment and encounters obstacles, it reflects, diffracts, and is absorbed, the amount depending on variables such as the composition, size and shape of the obstructing object.

Sound reflection is inherently more complex than the reflection of light, due to the large size and size variation of sound wavelengths (17m for 20Hz and 17x10\(^{-3}\)m for 20Khz [27]). However, at high frequencies, reflection characteristics can be approximated using the law of reflection [26]. Similarly, the law of reflection is only valid for wavelengths that are small in relation to the size of the reflecting surfaces [26]. Reflections in which the angle of incidence equals the angle of reflection (following the law of reflection) are called specular reflections, and occur on smooth, flat surfaces.

As a sound wave hits a surface, only a portion of the original sound is reflected. The rest of the energy is either absorbed in the material, or transmitted through, following the equation: energy \( I \) incident = \( I \) reflected + \( I \) absorbed + \( I \) transmitted. Sound absorption can be described as “the change of sound energy into some other form, usually heat, in passing through a material or on striking a surface,” [26]. The type of reflecting
material determines the level of absorption, and for an arbitrary material \( m \), a decimal value between 0-1 is assigned as the absorption coefficient \( \alpha \). A material that absorbs 45% of the reflected sound has an absorption coefficient of 0.45. An influential variable in the calculation of reverberation time of an enclosed space, the total absorption of an acoustic space is the product of each material’s absorption coefficient and the total surface area of that material (see Example: modeling the reflections of a virtual church).

Additionally, resulting from the disparity in sound wavelengths and inconsistent surface patterns, a material’s absorptive characteristics change with the wavelength of the reflected sound.

The amount of energy transmitted through the surface – according to the equation above – is the difference between the initial energy and the sum of the reflected and absorbed energy. The actual transmission of sound through a wall, door, window, etc. depends almost entirely on the material and the area that is exposed to the sound. In addition, as with most sound characteristics, sound transmission is frequency dependent.

Simple specular reflection is easy to imagine for flat surfaces, but for complex surfaces – where specular reflections are not feasible – a phenomenon called diffusion, or sound scattering, occurs. A large amount of diffusion in an acoustic space results in a desirable uniform sound field. Diffusion is caused by “the generous application of surface irregularities and scattering elements, such as pilasters, piers, exposed beams, coffered ceilings, sculptured balcony railings, and serrated enclosures” in a space [26]. In addition, diffusion is achieved in concert halls by alternating sound absorptive materials and reflective surfaces, or placing them in irregular patterns in the reflective areas. A space with a large amount of diffusion is considered to have “good acoustics” because the
reflections heard as delays are kept to a minimum and the sound field is uniform throughout the space.

The characteristic of sound waves (and waves in general) that allows them to aurally “bend” around corners or around obstacles is called diffraction. Again, the amount of diffraction depends on the wavelength of the sound and the size of the door, window, column, etc. If the wavelength is smaller than the opening (in the case of a window, or the space on either side of a column, etc), diffraction does not occur [32]. Similarly, as the size of the wavelength increases, the amount of diffraction increases. Therefore, low frequencies tend to propagate further, and thus are prominent behind occluding objects.

Reverberation is the result of each of the above-mentioned sound propagation effects – reflection, diffusion, absorption, and diffraction – occurring simultaneously in an enclosed space, and is perceived as a trailing off of a sound. The reflective characteristics of the many surfaces in a space result in the complex filtering and delaying effect of reverberation. The reverb time (RT) is known as the time it takes, in seconds, for the sound pressure level (SPL) to decrease 60 decibels after the initial sound has ceased [26].

Reverberation reflections are categorized as occurring early or late, depending, as you may guess, on when they occur. Early reflections are the first x reflections of a sound within the first 80-100 milliseconds. Late reflections begin when the listener is inundated with reflected sound and the reflections can no longer be heard as discrete. Late reflections are also referred to as the reverberation tail. Reverberation and reverberation time are discussed in greater detail below.
Acoustic Modeling Implementation in Video Games

An aurally successful 3D first person game, as I mentioned above, requires the implementation of a realistic sonic environment. In order to achieve this, the acoustic properties enumerated in the first section of Part II must be either modeled or approximated.

Acoustic properties of sound such as distance attenuation are fairly trivial to model; however, modeling the set of specular and diffuse reflections, and the associated frequency-dependencies in an acoustic space, is fairly challenging. Generally, modeling the sound propagation in an acoustic space requires a complex mathematical system.

Currently, acoustic models are typically constructed using geometric descriptions of a virtual space and the physical properties of wave propagation. The most common geometric methods are ray tracing, beam tracing, and image sources [14, 19, 20].

Ray tracing in acoustics is analogous to the tracing of light rays, a method of following the paths light takes from source to receiver. Rays are constructed from the sound source – modeled as obeying the wave equation – and are tracked through the virtual environment [12, 19, 20]. Reflective surfaces are defined using mathematical boundary conditions based on the physical properties of the surface. The receiver then constructs an impulse response based on the rays it intersects. In theory this method seems highly accurate, as it models sound based on wave propagation as is known in physics; however, ray tracing involves an extraordinary number of computations and is not plausible for most interactive systems.
Beam tracing is closely related to, and more efficient than ray tracing. Instead of calculating the paths of a large number of sound “rays”, these rays are collected into three-dimensional “beams”; 3D space is subdivided into convex polygons based on the propagation and reflection of sound from a source [13,14]. Dr. Thomas Funkhauser has implemented various beam tracing algorithms for acoustic modeling [13,14]. His method has proven much less computationally expensive, and has been known to construct accurate impulse responses for early reflections of a space [14].

The characteristic that makes beam tracing viable for interactive applications is the ability to construct a beam tree\(^1\) representing the sound propagation and reflection prior to exploration by the listener. Thus, the only calculations executed in real time are those necessary for the movement of the listener. However, the primary limitation beam tracing succumbs to is the beam tree must be reconstructed every time the sound source(s) moves. This recalculation adds enough complexity to stymie interactive possibilities with average computational power capabilities.

The use of image sources to model wave propagation provides a simpler, yet limited solution. The concept of image sources is very straightforward: for each reflective surface, a virtual sound source is constructed by mirroring the sound source across the surface [14, 19, 20]. Therefore, as the listener travels through the space, the reflection is modeled by the sound emanating from the virtual source (taking into account attenuation based on boundary characteristics). However, this method models only specular reflections and becomes difficult in oddly shaped spaces. Also, as the number of

\(^1\) A beam tree is generally implemented as a Binary Space Partitioning (BSP) tree
reflections modeled increases, the complexity increases exponentially \( (O(N^R); N = \text{number of surfaces}, R = \text{number of reflections}) \) [14, 19].

Each of the above-mentioned methods provides different degrees of aural acceptability for early reflections, but not late reflections. Calculating aesthetically pleasing and accurate late reflections using geometric reflection-modeling methods is an extremely difficult problem and beyond the scope of this paper. However, artificial reverberation is a possible solution for statistically computed late reflection approximation. The reverberation time of a space (as discussed above) can be fairly accurately calculated using Sabine’s formula (see Example: modeling the reflections of a virtual church), and an appropriate reverberation tail can be constructed that estimates the diffusion characteristics of the space.

Instead of building a complex artificial reverberation scheme consisting of nested allpass filters and delay lines, convolution reverberation is an alternative. A measured impulse response approximating the space can be convolved with the dry sound source to create an approximation of the sound in the space. Reverberation impulse responses are measured by recording a single, short impulse such as a gunshot in an acoustic space (it is best if the impulse contains all audible frequencies). This impulse response, if measured correctly, captures the reverb time and frequency response of the space.

It is not necessary to assume convolution reverberation and acoustic modeling as mutually exclusive. It is possible to model the early reflections in an acoustic space, and utilize the quality of convolution reverberation for the late reflections. This method allows for dynamism and positional reference via the early reflections, yet high quality diffuse and late reflection approximation.
Direct geometric acoustic modeling is only a partial solution to the problem of realistic acoustic modeling. In addition to the direct and specular sound wave paths, diffracted paths must also be considered. As we discussed above, diffraction occurs at edge boundaries, creating derived wave patterns also subject to additional reflection and/or diffraction. As predicted in the Geometric Theory of Diffraction (GTD), the aural effect of diffraction is low-pass filtering of the original sound; the intensity of which is dependent on the angle at which the listener sits from the diffracting boundary. Nicolas Tsingos et al. have developed a modeling technique extending their previous beam tracing method (described above) to include the effects of diffraction [29]. Variables affecting the diffraction intensity are the size and dimensions of the obstacle (or opening), distance from the source to the obstacle and the obstacle to the listener, the possibility of multiple obstacles, and reflected occlusion.

Current literature and implementations of environmental acoustic modeling have been primarily isolated to the academic domain. Commercial engines exist mainly as proprietary software, and game development companies tend to subject proprietary implementations to strict anti-dissemination rules. However, publicly accessible 3D audio engines do exist in some capacity. As an example, Anilogix AN-Sound 3D is a modern game audio engine implementing the following features: static and dynamic sound occluders; 3D environmental reverb full support (EAX 2.0); geometric reverb, obstruction and occlusion; multiple 3D environments; intelligent 3D voice management; true 3D spatial audio, real time direct and reflection path calculations for 3D environments; and mixing of multiple 3D audio environments in real time [30].
In the following section, I describe a simple acoustic modeling technique, viable for real time interactive applications.

**Example: modeling the reflections of a virtual church**

*The Acoustic Model*

Joey Silayan, a former New York University Tisch Interactive Telecommunications student, created the virtual church and the associated 3D environment referenced in this section as an element of the game *Orange*. As mentioned above, Joey and I developed *Orange* and all associated projects using Virtools Dev 3.0 and Maya.

For simplicity, it is assumed that the targeted virtual church is constructed of three basic materials: wood, wallboard, and glass. These materials were assigned based solely on their visual representation. This designation is necessary in order to assign the appropriate absorption characteristics to each reflective surface (see Table 1).

<table>
<thead>
<tr>
<th></th>
<th>125 Hz</th>
<th>250 Hz</th>
<th>500 Hz</th>
<th>1000 Hz</th>
<th>2000 Hz</th>
<th>4000 Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walls</td>
<td>0.3</td>
<td>0.1</td>
<td>0.05</td>
<td>0.04</td>
<td>0.07</td>
<td>0.1</td>
</tr>
<tr>
<td>Wood</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
<td>0.2</td>
<td>0.15</td>
<td>0.1</td>
</tr>
<tr>
<td>Glass</td>
<td>0.3</td>
<td>0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.07</td>
<td>0.04</td>
</tr>
</tbody>
</table>

*Table 1: Absorption coefficients [17].*

As Virtools does not provide real time signal processing up front, the reflections were not filtered in real time. Therefore, assuming the bulk of the frequency of the footstep sound is concentrated between 250-2000 Hz, the absorption coefficients in this range were averaged for this calculation. In addition to the three materials enumerated
above, the area of the floor that contains pews is considered somewhat of a sound trap, and a separate absorption coefficient is estimated at 0.5.

<table>
<thead>
<tr>
<th>Surface</th>
<th>Average Absorption Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall</td>
<td>0.065</td>
</tr>
<tr>
<td>Wood</td>
<td>0.213</td>
</tr>
<tr>
<td>Glass</td>
<td>0.143</td>
</tr>
<tr>
<td>Pew Area</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 2: Average absorption coefficients calculated by averaging the absorption coefficients in the frequency range of 250-2000 Hz.

Measurements of the church were conducted using relative distances. The exploring character, Archie, is estimated at 6 feet tall, or roughly 1.8 meters. Thus, the dimensions of the church, in meters, were calculated by dragging Archie from surface to surface and obtaining distance values in “Archies”. These values were then multiplied by 1.8 m/Archie to achieve distances in meters. In addition to these two units, Virtools has a native unit, which I will call the Vir. The conversion factor between meters and Virs was also calculated using Archie:

\[ 1\text{ Archie} = 1.8\text{ m} = 0.64\text{ Vir} \]

thus:

\[ 1\text{ m} = 0.36\text{ Vir}; \text{ and } 1\text{ Vir} = 2.8\text{ m} \]

See Figures 2 and 3 for diagrams of the church dimensions.
**Figure 2:** Dimensions of the front/back and roof of the church

**Figure 3:** Dimensions of the sides of the church.

*NOT DRAWN TO SCALE*
With the dimensions of the church and the absorption coefficients for all materials in hand, the estimated reverberation time $T_T$ (the time it takes for the reverberation of a single impulse to fall below -60 dB SPL) can be calculated using Sabine’s formula [17]:

$$T_T = (0.16 \text{ s/m}) V / S_e$$

$V$ = total volume,
$S_e$ = effective absorption area

Before we can use this equation, however, the effective absorption area, $S_e$, must be calculated. $S_e$ is calculated by summing the products of each different surface area by its corresponding absorption coefficient $\_ [17]$

$$S_e = _1 S_1 + _2 S_2 + _3 S_3 + \ldots$$

View Table 3 for the total absorption area and the reverberation time calculations.

<table>
<thead>
<tr>
<th>Absorption Area Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Back Wall (with door)</strong></td>
</tr>
<tr>
<td>door area</td>
</tr>
<tr>
<td>wall</td>
</tr>
<tr>
<td><strong>Side Wall 1</strong></td>
</tr>
<tr>
<td>total wall space (including windows)</td>
</tr>
<tr>
<td>total window space</td>
</tr>
<tr>
<td>total wood column</td>
</tr>
<tr>
<td>effective wall</td>
</tr>
<tr>
<td><strong>Side Wall 2</strong></td>
</tr>
<tr>
<td>same as side wall 1</td>
</tr>
<tr>
<td><strong>Front Wall</strong></td>
</tr>
</tbody>
</table>

back wall total area = \(5.75m \times 16m = 92m^2\)
alter stairs (area that covers part of the wall) = \(12m \times 1m = 12m^2\)
effective wall = \(92m^2 - 12m^2 = 80m^2\)

**Ceiling**
sides (small horizontal area) = \(2 \times (2m \times 17.5m) = 70m^2\)
A frame roof = \(2 \times (8m \times 17.5m) = 280m^2\)
A frame wall = \(2 \times (1/2 \times 12m \times 5.5m) = 66m^2\)

**Floor**
total floor = \(16m \times 17.5m = 280m^2\)
pew area = \(9m \times 12m = 108m^2\)
total area – pew area = \(280m^2 - 108m^2 = 172m^2\)

**TOTALS:**
total wall material = \(77.75m^2 + (2 \times 39.13m^2) + 80m^2 + 280m^2 + 70m^2 + 66m^2\)  
= \(652m^2\)
total glass = \(44.82 \times 2 = 89.64m^2\)
total wood material = \(16.68m^3 \times 2 + 172m^2 = 205.36m^2\)
total door area = \(14.25m^2\)

Total Absorption, \(S_e\) = \((0.065)652m^2 + (0.143)89.64m^2 + (0.213)205.36m^2 + (1)14.25m^2 + (0.5)108m^2 = 167.2m^2\)

**Reverberation Time Calculation**
Reverberation Time, \(T_r\) = \((0.16 s/m)V/S_e\); \(V = \) total volume, \(S_e = \) effective absorption area

= \((0.16 s/m)(2188m^3)/(167.2m^2) = 2.09\) seconds

**Table 3:** Total absorption area calculations and the reverberation time calculation

This calculated reverberation time serves as a model for the artificial reverberation applied to the footstep sound.
Early Reflection Modeling

To obtain a maximum and realistic efficiency, only the highest energy, single-surface reflections were modeled. For this scenario, the image source method is the most efficient and simplest to implement. The five single-surface reflections modeled occur at the following surfaces: the back wall, front wall, left side wall (including windows), right side wall (including windows), and alter.

The sound source is the footstep of Archie; Archie is also the sound receiver. As the source is at the same (x, z) coordinates as the listener at all times, the single-surface specular reflections are perpendicular to the surface (if viewed from above). Virtual sources are created by mirroring the source across the five surfaces listed above (see Figure 4).
As the character explores the church, the delay and attenuation due to distance and absorption are calculated for each virtual source in real time. The absorption attenuation is applied to the sound at exactly 1/2 the distance from the virtual source to the listener (at the surface boundary). When the sound source is triggered (Archie’s foot hits the ground and the footstep sound is played), each delay (calculated in ms) and
attenuation is applied to the original signal and the sound is played (from the direction of the appropriate surface.

\[
\text{Delay} = \frac{\text{distance to virtual source}}{\text{speed of sound} = 0.344\text{m/ms}}
\]

The attenuation due to distance is calculated following the inverse square law:

\[
\text{Initial intensity, } I, \text{ at distance } r: \quad I_r = \frac{I}{r^2}
\]

The attenuation due to absorption is calculated using the aforementioned absorption coefficients. The final intensity value is converted to amplitude by taking the square root of the intensity value. Final amplitude \(a\):

\[
a = \sqrt{\left(\frac{1}{1/2 d_v^2}\right) \times (1 - \alpha) \times (1/(1/2 d_v^2))}
\]

\(d_v = \text{distance to virtual source}\)
\(\alpha = \text{absorption coefficient}\)

**Design**

The development software Virtools Dev 3.0 accommodates the construction of interactive 3D environments. In addition to visual tools, Virtools provides a sound engine capable of 3D positional sound that includes a limited set of sound propagation parameters. Thus, Virtools is potentially an acceptable environment for constructing a simple acoustic model.

Interactivity in Virtools is achieved through the internal “scripting” interface which consists of elements called \textit{behaviors} or \textit{building blocks} (see Figure 5) with inputs and outputs strung together in an execution path. Sounds are triggered using specialized
behaviors that, among other parameters, designate the object to attach the sound to (if the sound is desired as a point source).

![Diagram of two behaviors in an execution path]

**Figure 5:** An example of a two behaviors in an execution path

Virtools manages scripts on a frame-by-frame basis. A frame in Virtools is analogous to a video frame, only the frame rate is dynamic in Virtools. Understanding the frame rate is important because a script can only execute one time per frame, and if a script or behavior is inefficient, it will reduce the frame rate – sometimes bringing the interaction to a standstill.

Virtools is extendible via custom behaviors and an SDK. The SDK includes templates for the creation of behaviors and many examples.

**Early Reflections Implementation**

The delay times and amplitude attenuation are calculated using a custom behavior, *CalculateReflections*, written in C++ using Microsoft Visual Studio 6.0. *CalculateReflections* is built as a dynamically-linked library (DLL) and copied into the Virtools application directory.

When Archie reaches the church, a church-Archie intersection occurs – detected with a separate Virtools behavior – and the *CalculateReflections* behavior is turned on. The input to this behavior is the character Archie. The position of Archie is tracked – in
Virtools units – through the environment using his associated World Matrix. At each frame, Archie’s location is compared to the static location of the five reflective surfaces, the distances are deduced in meters, and the delays and attenuations are calculated for each surface.

Example CalculateReflections code:

```cpp
/*==========================================================================
  // left wall delay and attenuation calculation
  //-------------------------------
  // find distance between Archie and the left wall, mat is the world matrix
  dVir = (LEFTWALL + ((-1)*mat[3][2]));

  // convert to meters
dMeters = dVir * CONVERSION;

  // find delay in milliseconds
dDelayInMS = (dMeters * 2)/SPEEDOFSOUND;

  // check if Archie is in front of a window - of he is, use the glass absorption coefficient
  if( (((double)mat[3][0] > (double)WINDOW1A) && ((double)mat[3][0] < (double)WINDOW1B)) || (((double)mat[3][0] > (double)WINDOW2A) && ((double)mat[3][0] < (double)WINDOW2B)) || (((double)mat[3][0] > (double)WINDOW3A) && ((double)mat[3][0] < (double)WINDOW3B)))
    // reflection should use glass absorption coefficient
    dAmp = sqrt((1/(dMeters*dMeters))*(1-GLASS)*(1/(dMeters*dMeters)));
  else
    // if not, use wall absorption coefficient
    dAmp = sqrt((1/(dMeters*dMeters))*(1-WALL)*(1/(dMeters*dMeters)));

  // convert to float
  fDelay1 = (float)dDelayInMS;
  fGain1 = (float)dAmp;

  // set output parameters
  beh->SetOutputParameterValue(4,&fDelay1);
  beh->SetOutputParameterValue(5,&fGain1);
/*==========================================================================

These five delay/amplitude pairs are updated every frame and placed in the output buffers. In the Virtools script, each output delay value is sent to an individual delayer.
behavior, which delays the input, a time value (in this case, a millisecond value), before activating the output.

At the moment Archie’s foot touches the ground and the initial sound is triggered, all five *delayer* behaviors are triggered simultaneously. Each *delayer* behavior, in turn, triggers a *play sound instance* behavior that contains the original footstep sound with the modified amplitude value and the correct 3D positioning (i.e. the reflection from the left wall is attached to the left wall, etc.). The result is a subset of the early reflections calculated and triggered at an acceptable interactive rate (see Figure 6).

![Figure 6](image)

**Figure 6**: The script for triggering the delayed footstep sounds and the late reflections

**Late Reflections**

Ten late reverberation samples, minus the early reflections, were preprocessed using the previously calculated reverberation time of 2.09 seconds and the Waves RVerb Direct X plugin. As the initial footstep sound was triggered, a late reverberation sample
was randomly selected from a Virtools Array, attached to the rafters of the church, and played. The 3D attachment achieves a more distinct sense of spatialization.

**Limitations**

A difficulty arose with the measurement of distances. Most distances, because they were first measured in “Archies”, then converted to meters, were basically estimates. Also, unfortunately, the entire church was a single 3D object. Therefore, the position could not be calculated on the fly, and was statically measured, with a relatively high margin of error.

During the early reflection calculation, it was assumed that the sound source and receiver were at the same location when in fact they were roughly 1.7 meters from each other (on the vertical axis). This difference is small when Archie was a large distance from a surface, but the error became noticeable at close distances.

As the Virtools environment does not currently permit real time audio signal processing, no filtering of the initial footstep sound was achieved; thus the distinct reflective qualities of the different surfaces were not readily apparent other than the level of attenuation due to absorption.

The intersection detection algorithm within Virtools was not robust. Occasionally, while inside the church, the intersection detection between Archie and the church failed. Also, the custom behavior DLL CalculateReflections must be copied into the Virtools Web Player directory on every machine that desires playback of this project file.
Despite these limitations, this method acts as a proof-of-concept for this simple acoustic modeling technique and will likely prove desirable at higher orders of reflection modeling.

Part III: Interactive Game Music

In addition to an accurate and interesting audio environment in terms of sound effects and sound wave propagation modeling, music affects the game play extraordinarily. In order to ship a complete package, the music must reflect and enhance the desired emotions and avoid extreme repetition. If the music is neglected, the entire game may prove unsuccessful.

Every gamer, at one point or another, has wanted to turn off the music in a game because it had become unbelievably repetitive. Historically, game music was minimal, and thus became an aural burden very quickly. Still, even with modern budgets and technology, the quantity of music composed for a single game is much smaller than the hours of expected game play, and thus the music retains the same repetitive properties as before. In order to combat this problem, composers and developers can work together to utilize the interactive nature of games for music. Interactive game music, or adaptive music, adjusts in detail based on decisions made by the player. An interactive score, ideally, will follow the action of the game in a way not possible with a limited amount of static music.

Music in games follows the same criterion for success as a film score – emotional guidance, subtlety, production quality, and musical quality, for example – but the scenes
are not fully composed when the music is written. A film is a static form (theoretically), and is exactly the same regardless of the viewer’s actions. A game varies, however, and this major difference necessitates new scoring techniques.

The difficulty with an interactive score is retaining musical quality (given that the music has quality when played statically). Generally, the easiest form of interactivity requires loop-based music. This style achieved popularity in FPSs such as Quake II and Unreal (as static loop-based music – without interactivity). Interactive scores in this style add and subtract elements of the music as the player’s environment changes, and can potentially create a successful immersive musical environment. However, static tempi and loop-based music remove the human characteristics inherent in free performance. It is possible to avoid a static tempo, but the engine loses robustness without a steady tick.

An interesting side effect of an interactive score is that the player essentially directs the music. Because the player traverses the environment at his or her own free will in most cases, the structure of the music is at the mercy of the player.

The following section describes, in detail, the interactive music system created for the game Orange.

**An Interactive Music System**

Described in this section is an interactive music system implemented in the audio routing and processing environment Max/MSP. Max/MSP is an ideal tool for constructing a system of this nature as it specializes in the development of audio processing and organization – and contains many accurate synchronization tools.
Initially, the interactive music engine was to be built using Virtools; however, stable timing synchronization was not possible.

As with much interactive music, I constructed this system with the idea of using segments, or loops, to shift the mood. To simplify, I composed the music for two moods, corresponding to the two potential moods in the game *Orange*. These two emotional states can be described as *relaxed* and *intense* (relaxed = Archie wandering, intense = Archie/Anna verbal joust). In addition, the system takes into account the transitions between the states. As the intensity value changes, different audio segments are added and subtracted appropriately according to a simple logic system.

**The Technique**

A composer might ask, “How do I compose music without knowing the overall temporal qualities?” This can be the most difficult task of the system as the common perception is that music is generally constructed as a static unit. However, much 20th and 21st century music is composed algorithmically; much of which has little or no formal structure. Algorithmic composition is, in fact, exactly what interactive game music composition is. However, extremely aleatoric music, or music incorporating some element of chance, has historically appealed to a narrow audience. In the case of game music, it is necessary to please a much larger crowd, including young adults; so the music content (melody, harmony, form, and rhythm) tends to sit in the middle of the mainstream.

I composed the music for this interactive music system as a set of compatible audio segments, each lasting exactly 12 seconds. The segments were divided into two
types: the relaxed-style and the intense-style, with some fitting in between. As the elements are mixed on-the-fly, it helps to auralize the potential mixes as the segments are composed. This includes spatializing and equalizing the elements based on the potential combinations. Also, in order to reduce repetition, a single segment can be performed and recorded a few times, with small variations. This is analogous to having a few samples of the same sound — as discussed in Part I. Thirty-two audio segments were composed for this interactive music system.

It is helpful to visualize the system as a digital mixing console where each segment is assigned stereo faders and when the segment is triggered, the gain is increased to the appropriate level. As mentioned above, as the mood changes (via a user-controlled slider, in this case — simulating changes in the mood of the game), segments are added and removed based on a partially randomized logic system.

The logic system is based on the idea that as the intensity value is increased, the intense audio segments are added to the mix, and vice-versa; all the while maintaining interesting music. This is accomplished using probabilities, and by defining certain cutoff points within the intensity spectrum where some audio segments would no longer have any chance of playing, while others would obtain an increased chance of playing. Ideally, the intensity spectrum would affect the music continuously, but I found that changing the music at the point where the segments repeat is satisfactory, and more realistic than fading the segments in and out.

To achieve interesting music when the intensity value is not changing, I implemented mild randomness. Segments were grouped depending on their similarity and necessity. If it is important that a single element, such as the bass, is playing most or
all the time, then a group of bass lines can be created and a single line chosen every cycle. During playback, segments are chosen from these groups based on a pre-defined probability. This type of system is very similar to those used to diminish repetition described in Part I.

**Implementation (this section assumes basic knowledge of Max/MSP)**

As I mentioned, the interactive music system was implemented as a set of Max/MSP patches. The system was divided into five main parts: the “mixing” console, the segment loader, the playback utility, the intensity retrieval (from the user), and the logic system. The main patch binds these three sections together.

The initial patch – the one that loads when the file is opened – contains a subpatch to load all the segments, the playback elements (one `groove~` playback object for each segment – for looping), a subpatch containing the logic system, stereo faders for each segment, and a synchronizing engine for playback – also a subpatch (See Figure 3.1).
Figure 3.1: The initial Max/MSP patch. Hidden are the playback synchronization mechanism and the logic system.

When the play button is clicked, all the loaded segments begin playing at the same time (with gain 0.0 initially). In addition, metronomes (Metro objects in Max/MSP) are started with click (bang) times of 12 seconds. These are used for synchronizing the transitions with the ends (and beginnings) of the segments.

The intensity value is controlled with the green slider on the top of the patch. The slider outputs a number between 0-100 (see Figure 3.2).

Figure 3.2: The intensity slider – the source of user input to the interactive music system.
When the user alters the intensity by moving the slider, the value is sent to the subpatch labeled *control*. This is the logic system (see Figure 3.3).

![Figure 3.3: The control subpatch.](image)

The control subpatch contains the mood divisions in a 100-value scale. As the image above denotes, the divisions occur at 5, 15, 30, 50, 75, and 90. When the intensity slider falls in between two of these values, the appropriate subpatch is triggered, thus triggering the appropriate set of audio segments.
Each logic subpatch randomly picks the segments out of the appropriate groups according to numbers assigned to each segment. The logic subpatches each have 64 outputs – one for each of the 32 segment values, and one for whether to turn that segment on or off (1 for off, 2 for on).

**Figure 3.4:** The logic subpatch labeled *Slowest*. The *random* objects are triggered once every 12 seconds.

The above image shows the random audio segment selection. For each odd output, a number corresponding to a segment is sent out. The output to the right of the number contains either a 1 or a 2 distinguishing whether to turn the segment on or off at the next
boundary (as discussed above). Every segment is either turned on or off at each boundary.

The outputs of the logic subpatches are sent out of the control patch and into the playback synchronization mechanism (see Figure 3.5). This system is divided into four subpatches to conserve space. Each has 16 inputs and 32 outputs. The 32 outputs of each control the faders. The need for the synchronization arises from the need to change the segments only at the segment boundaries (every 12 seconds).

Figure 3.5: One of the synchronization subpatches. Each of these eight triggers can either turn on a segment or turn it off depending on the 1 or 2 it receives from the logic output. The segments are turned on or off only at segment boundaries via this synchronization.
This completes the logic path of this interactive music system. It appears, from the user’s viewpoint, that as they move the intensity slider to the positive, the music gains momentum and drive, and as they decrease the value, the music loses the heavy drive.

**Conclusions and Limitations**

An interactive music system such as this one stretches the useful amount of music to the greatest extent (if each segment was placed next to each other temporally, the total would be roughly six and one-half minutes as opposed to an indefinite amount – but for creative music, each segment would not be played like this).

This interactive music system is valuable as a proof-of-concept, but is much too simple for realistic use. However, it serves as a base for further development, and works well in this respect.

Also, Max/MSP has not been incorporated into any sort of gaming structure, so this tool is not viable for game integration. However, this model serves as an example for such an integration tool.

The complexity of this system relies heavily on the number of audio segments; therefore, a realistic system using a high number of segments would dramatically increase the complexity of this patch.

Additionally, in order to accurately portray an adequate number of potential game scenarios, more than two moods are needed. However, many games are fairly simple, and only require two or three music states; but this is not consistently the case. A possible solution is to implement a number of 32-segment systems and link them.
In general, this music system helps prove interactive music is not out of the reach of game composers.

**Conclusions**

Many repetition-reducing techniques have been discussed in this study, and each has one major aspect in common. Each method manages to remove much of the undesirable repetition through mild randomization and simple real time selection. The footstep engine selects samples at random from sample bins; the music does much of the same – selecting segments from predefined groups.

Even though game audio may seem to be a series of highly specialized methods, the cohesion is visible from a broad point of view. With these ideas, a total system could be constructed for all game audio development. Reducing the repetitive qualities of game audio and increasing realism does not require revolutionary tactics, but the results are worlds above static repetition.

However, game audio development is an extremely large and potentially complicated field, and without the right audio elements, a game cannot compete in the modern market. As available resources increase, the idea of accurate acoustic modeling and highly complex interactive music systems will become reality. These capabilities are already mildly present in the industry, and will only expand.
References


[2] Alexander Brandon, “Pushing the Envelope”, Game Developer, June 1, 2004, AURAL FIXATION; Pg. 46, 823 words.


[25] Peter Evans, “No longer just a game: 300 studios, 5,000 employees and growing”, National Post's Financial Post & FP Investing (Canada), May 21, 2005 Saturday, Toronto Edition, FINANCIAL POST; Pg. FP1 , 1669 words.


[31] http://nintendo.about.com/od/e32005coverage/a/e3revdetails.htm


[34] http://interface.cipic.ucdavis.edu/CIL_tutorial/3D_HRTF/3D_HRTF.htm


[37] http://cnx.rice.edu/content/m12022/latest/

[38] http://www.gamasutra.com/features/19991102/gameaudiosupp/tools_03.htm