## 13.4 Rendering Audio in 3D

Thus far, we’ve learned about the physics of sound, the mathematics of signal processing and the various technologies that are used to record and play back sounds. In this section, we’ll explore how all of this theory and technology

can be put to use in a game engine, in order to produce realistic, immersive soundscapes for our games.

Any game that takes place in a virtual 3D world requires some sort of *3D audio rendering engine.* A high-quality 3D audio system should provide the player with a rich, immersive and believable soundscape that matches what’s going on in this 3D world, while supporting the story and remaining true to the tonal design of the game.

* The inputs to this system are the myriad *3D sounds* that emanate from all over the game world: footsteps, speech, the sound of objects bumping into one another, gunfire, ambient sounds like wind or rainfall and so on.
* Its output is a handful of sound channels that, when played in the speakers, reproduce as believably as possible what the player would actually hear if he or she were really there in the virtual game world.

Ideally we’d like our audio engine to produce its output in full 7.1 or 5.1 surround sound, because this gives the ears the richest possible set of positional cues. However, audio engines must also support stereo output for players who don’t have fancy home theater systems—or who just want to play their game using headphones so they don’t wake their neighbors.

A game’s audio engine is also responsible for playing sounds that do not originate in the virtual world. Examples include the music track, sounds made by the in-game menu system, a narrator’s voice-over, the voice of the player character (especially in first-person shooters) and possibly certain ambient sounds. We call these *2D sounds.* Such sounds are designed to be played “directly” in the speakers, after having been mixed with the outputs of the 3D spatialization engine.

### 13.4.1 Overview of 3D Sound Rendering

The primary tasks performed by the 3D audio engine are as follows:

* *Sound synthesis* is the process of generating the sound signals that correspond to the events occurring in the game world. These might be produced by playing back pre-recorded sound *clips*, or they might be *procedurally* generated at runtime.
* *Spatialization* produces the illusion that each 3D sound is coming from the proper location in the game world, from the point of view of the listener. Spatialization is accomplished by controlling the *amplitude* of each sound wave (i.e., its gain or volume) in two ways:

◦ *Distance-based attenuation* controls the *overall volume* of a sound in order to provide an indication of its *radial distance* from the listener.

◦ *Pan* controls a sound’s relative volume in each of the available speakers in order to provide an indication of *direction* from which the sound is arriving.

* *Acoustical modeling* heightens the realism of the rendered soundscape by mimicking the early reflections and late reverberations that characterize the listening space, and by accounting for the presence of obstacles that partially or completely block the path between the sound source and the listener. Some sound engines also model the frequency-dependent effects of *atmospheric absorption* (Section 13.1.3.2) and/or HRTF effects (Section 13.1.4).
* *Doppler shifting* may also be applied to account for any relative movement between a sound source and the listener.
* *Mixing* is the process of controlling the relative volumes of all the 2D and 3D sounds in our game. The mix is driven in part by physics and in part by aesthetic choices made by the game’s sound designers.

### 13.4.2 Modeling the Audio World

In order to render the soundscape of a virtual world, we must first describe that world to the engine. The “audio world model” consists of the following elements:

* *3D sound sources.* Each 3D sound in the game world consists of a monophonic audio signal, emanating from a specific *position*. We must also provide the engine with its *velocity*, *radiation pattern* (omnidirectional, conical, planar) and *range* (beyond which the sound is inaudible).
* *Listener.* The listener is a “virtual microphone” located in the game world. It is defined by its *position*, *velocity* and *orientation*.
* *Environmental model.* This model either describes the *geometry* and *properties* of the surfaces and objects present in the virtual world, and/or it describes the *acoustic properties* of the listening spaces in which gameplay takes place.

The *positions* of the source and listener are used for *distance-based attenuation*; the *radiation pattern* of the sound source also factors into the distancebased attenuation calcuation. The *orientation* of the listener defines a reference frame in which the *angular position* of the sound is calculated. This angle in turn determines the *pan*—the relative volumes of the sound in the five or seven main speakers of 5.1 or 7.1 surround sound, respectively. The *relative velocity* of source and listener is used when applying a *Doppler shift.* And last but not least, the *environmental model* is used for modeling the acoustics of the listening space and to account for partial or complete blockage of the sound path.

### 13.4.3 Distance-Based Attenuation

Distance-based attenuation reduces the volume of a 3D sound as the radial distance between it and the listener increases.

#### 13.4.3.1 Fall-Off Min and Max

The number of sound sources in a typical game world is very large. Due to hardware and CPU bandwidth limitations, we couldn’t possibly render them all. And we wouldn’t want to, because thanks to distance-based fall-off all sounds beyond a certain distance from the listener can’t be heard anyway. For this reason, each sound source is usually annoted with fall-off (FO) parameters.

The *fall-off min* (“FO min” for short) is a minimum radius, which we’ll denote *r*min, within which the sound doesn’t fall off at all and is heard at full volume. The *fall-off max* or “FO max” is a maximum radius, denoted *r*max, beyond which the sound source is considered to be silent and can therefore be ignored. Between the FO min and FO max, we need to blend smoothly from full volume down to zero.

#### 13.4.3.2 Blending to Zero

One way to blend from maximum volume down to zero is to use a linear ramp between FO min and FO max. Depending on the type of sound, a linear fall-off might sound just fine.

In Section 13.1.3.1, we learned that sound intensity, which is closely related to our perception of “loudness,” falls off with radial distance according to a 1*/r*2 rule. Gain, which is proportional to the amplitude of the sound pressure, falls of as 1*/r*. So really the right thing to do is to use a 1*/r* curve to blend the gain of a sound from full volume down to zero.

One problem with the function 1*/r* is that it is asymptotic—it never quite reaches zero, no matter how large *r* gets. We can fix this by shifting the curve slightly downward so that it crosses the *r* axis at *r*max. Or we can simply clamp the sound intensity to zero for all *r > r*max.

#### 13.4.3.3 Bending the Rules

When making *The Last of Us*, Naughty Dog’s sound department discovered that attenuating character dialog using the 1*/r*2 rule caused speech to become unintelligible too quickly for characters that were only a modest distance away. This was a serious problem, especially during the stealth sections of gameplay, where hearing the enemies’ ambient conversations was important both as a tactical tool and as a means of advancing the storyline.

To solve this problem, the sound department at Naughty Dog utilized a sophisticated fall-off curve that causes dialog to roll off more slowly near the listener, more quickly in the mid-range, and then more slowly again as the distance to the listener grows very large. This allows speech to be audible over longer distances, while still retaining a natural-sounding fall-off.

The dialog fall-off curves were also adjusted dynamically at runtime, based on the current “tension level” of the game (i.e., whether the enemies are unaware of the player, are searching for him or are engaged in direct combat with him). This is what allows the voices in *The Last of Us* to project over longer distances during stealth gameplay, while not rising to overpowering levels when combat breaks out.

Finally, a “sweetener” reverb could be optionally enabled to allow character voices to bleed around corners, even when the direct path is 100% obstructed. This tool is incredibly helpful in situations where modeling realistic fall-off is less important than ensuring that the player can hear a conversation clearly.

There are all sorts of ways to “cheat” when designing your 3D audio model. But no matter what you do, always remember this simple lesson: Never be afraid to do whatever it takes to satisfy the *needs of your game.* Don’t worry— the laws of physics won’t be offended!

#### 13.4.3.4 Atmospheric Attenuation

As we saw in Section 13.1.3.2, low-pitched sounds are attenuated by the atmosphere less than high-pitched sounds. Some games, including Naughty Dog’s *The Last of Us*, model this phenomenon by applying a low-pass filter to each 3D sound whose passband slides toward lower and lower frequencies as the distance between the sound source and the listener increases.

### 13.4.4 Pan

*Panning* is a technique used to provide the illusion that a 3D sound is coming from a particular direction. By controlling the volume (i.e., gain) of the sound in each of the available speakers, we can induce the perception of a *phantom*

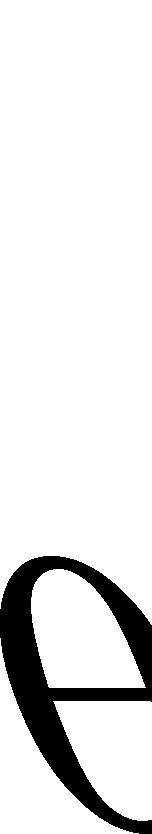
**Figure 13.29.** Speaker layout for 7.1 pan.

*image* of the sound in three-dimensional space. This method of panning is called *amplitude panning* because we are providing angular information to the listener by adjusting only the amplitudes of the sound waves produced at each speaker (as opposed to using phase offsets, reverb or filtering to provide positional cues). It is sometimes referred to as *IID panning* because it relies on the perceptual effects of interaural intensity difference (IID) to produce a sound’s phantom image.

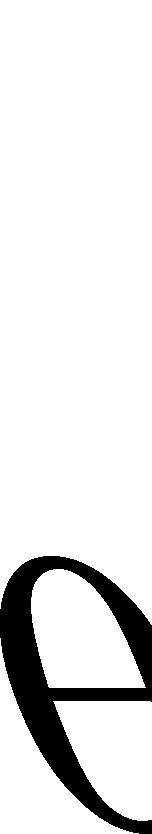
The term “pan” comes from early technology that used a “panoramic potentiometer” (variable resistor) or “pan pot” to control the relative volumes of the left and right speakers of a stereo system. Dialing the pan pot to one extreme would produce sound only in the left speaker; dialing it to the other extreme would drive the right speaker exclusively; and centering the pan pot dial would distribute the sound equally to both speakers.

To understand how pan works, we envision our listener located at the center of a circle. The speakers are positioned at various points on the circumference of this circle, so we’ll call it the *speaker circle* in this book. The radius of the circle approximates the average distance between the listener and any one speaker.

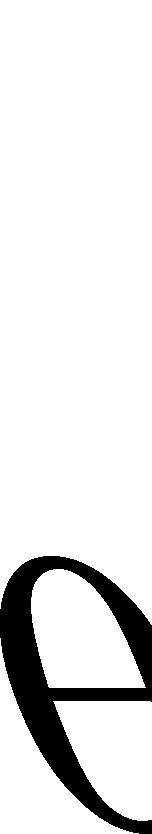
For a stereo sound system, the front and right speakers are located roughly at ±45 degrees to the left and right of center. For stereo headpones, they are positioned at ±90 degrees (and the radius is much smaller). For 7.1 surround sound, we consider only the seven “main” speakers, as the LFE channel provides no positional cues. These speakers are located roughly as shown in Figure 13.29. When panning to a 5.1 system, we simply omit the surround left and surround right speakers.



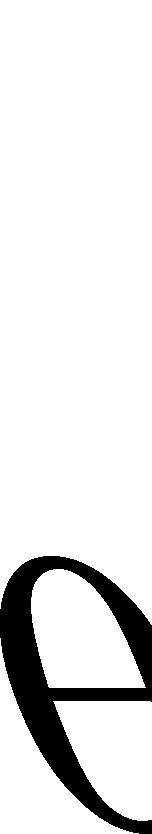
*s*



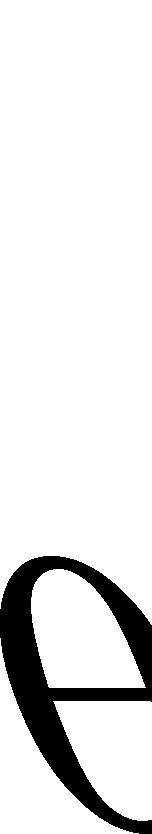
2



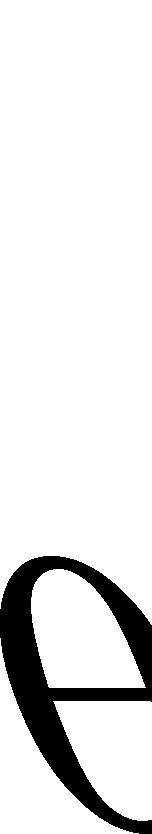
1



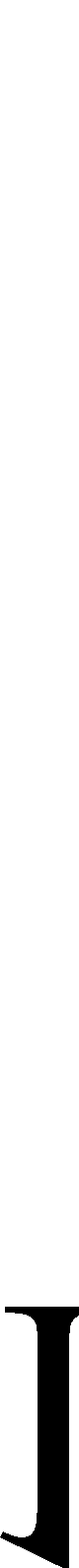
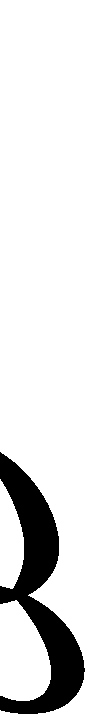
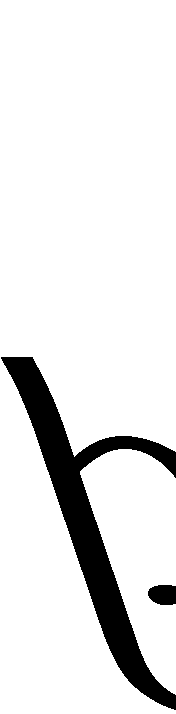
*s*



2



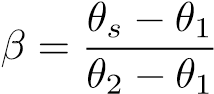
1



**Figure 13.30.** Treating the sound as a point source, the pan blend percentage *β* is calculated between the two speakers immediately adjacent to the source.

For the time being, let’s treat each 3D sound as a point source. To pan a sound, we first determine its *azimuthal* (horizontal) angle. The azimuthal angle must be measured in the local space of the listener, so that an angle of zero corresponds to the position directly in front of the listener. Next, we figure out which two speakers around the circumference of our circle are *adjacent* to this azimuthal angle. We convert the angle into a percentage of the arc between the two speakers. Finally, we use this percentage to determine the gains of the sound in each speaker.

To formulate this mathematically, let’s use the symbol *θs* for the azimuthal angle of the sound. We’ll call the angles of the two adjacent speakers *θ*1 and *θ*2. The percentage *β* is then calculated as follows:

*.*

This calculation is illustrated in Figure 13.30.

#### 13.4.4.1 Constant Gain Panning

Our first instinct might be to use the percentage *β* to perform a simple linear interpolation between the gains of the two speakers. Given the gain *A* of the unpanned sound, the gains of that sound as played in each speaker would be calculated as follows:

*A*1 = (1 − *β*)*A*; *A*2 = *βA.*

This is known as *constant gain* panning, because the net gain *A* = *A*1 + *A*2 is constant, independent of the values of *θs* and *β*.

The main problem with constant gain panning is that it does *not* produce the perception of constant *loudness* as the sound moves around the acoustic field. Gain controls the amplitude of the sound pressure wave, and therefore controls the *sound pressure level* (SPL). However, as we learned in Section 13.1.2, human perception of loudness is actually proportional to the *intensity* or *power* of a sound wave, both of which vary as the *square* of the SPL.

As an illustration of the problem, imagine that our sound is panned to the halfway point between our two speakers. Constant gain panning would have us set the gains *A*1 and each. But this yields a total power of

. In other words, the loudness of the sound

will be *one-half* of what it would have been, had the sound been panned to only the left or the right speaker.

#### 13.4.4.2 The Constant Power Pan Law

Clearly in order to keep the perception of loudness constant as a sound’s image moves about the listener, we need to keep the *power* constant. This rule is known as the *constant power pan law*, or just the *pan law* for short.

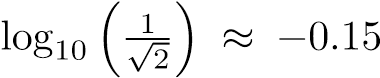
There’s a very easy way to implement the constant power pan law. Instead of linearly interpolating the gains, we use the sine and cosine of the blend percentage *β* to calculate them:

;



Consider again a sound image that is panned to halfway between the two speakers (). With constant power panning, the two speakers’ gains will be set to. This yields a total power of

. This works for any value of *β*, so the power *A*2 is constant no matter where our sound image is placed around the circle.

Sound designers often apply a “3 dB rule” to account for the pan law: If a sound is to be mixed equally to two speakers, the gain in each speaker should be reduced by 3 dB relative to the gain that would be used if the sound were to be played in only one speaker. The value −3 dB arises because . Voltage gain (or amplitude gain) is defined as

20log10(*A*out*/A*in), and 20 × −0*.*15 = −3 dB. (The 20 in front of the logarithm arises because a decibel is one-tenth of a bel, multiplied by two to account for the fact that we’re dealing with *A*2 and not *A*.)

#### 13.4.4.3 Headroom

Panning causes sounds to be rendered entirely by one speaker in some situations, and by two (or more, as we’ll see) speakers in others. Let’s say a sound is being played equally by two adjacent speakers, and its volume is so loud that each speaker is outputting its maximum power. What happens when that sound pans around to only one speaker? The answer is that we’d probably blow out the speaker, because our constant power pan law requires us to use more gain for one speaker than for two.

To prevent this problem, we need to artificially lower the maximum gains of our sounds across the board, such that the worst-case scenario of playing the sound in one speaker won’t overdrive that speaker. The practice of artificially reducing the maximum range of volume is known as “leaving oneself some *headroom*.”

The concept of headroom also applies to *mixing*. When two or more sounds are mixed, their amplitudes add up. By leaving some headroom in our mix, we can accomodate worst-case scenarios where a large number of high-volume sounds play simultaneously.

**13.4.4.4 To Center or Not To Center?**

In cinema, the center channel was historically used for speech; only the sound effects would be panned to the other speakers around the room. The idea behind this practice was that the characters in the movie are usually on-screen when they speak, so the audience expects to hear their voices front-and-center. This approach has the nice side-effect of separating out the speech from the rest of the sounds in the film, meaning that loud sound effects won’t use up all the available headroom and drown out the dialog.

In 3D games, the situation is quite different. The player generally wants to hear dialog coming from the “correct” location around him or her. If the player swings the camera by 180 degrees, the dialog should likewise swing about the sound field by 180 degrees. As such, games usually do not assign all dialog to the center speaker; instead, it is included in the pan for both sound effects and dialog.

Of course, this brings us back to the headroom problem—loud gunfire can now completely drown out the speech. At Naughty Dog, we overcame this problem by “splitting the difference” and always playing *some* of the dialog in the center channel, as well as panning some of it to the rest of the speakers along with the sound effects.

#### 13.4.4.5 Focus

When the source of a sound is far away from the listener, we can treat it as a point source. We simply calculate a single azimuthal angle and feed it into our constant power panning system. However, when a sound source approaches or actually enters into the circle that defines the radial distance of the speakers from the listener, it can no longer be accurately modeled as a point source represented by a single angle.

Consider the case of moving toward and past a sound source. At first, the sound source appears entirely in the front speakers. As it passes the listener, we somehow need to transfer the sound to the rear speakers. If we model the sound as a point source, our only option is to “pop” the sound from the fronts to the rears.

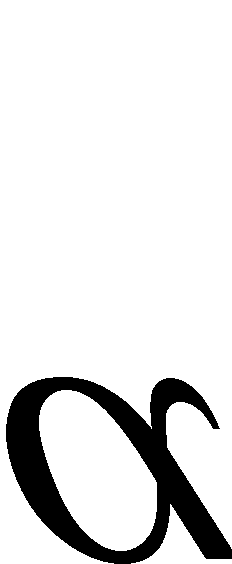
Ideally we’d like the sound’s image to gradually “spread out” around the speaker circle as it approaches. That way, as it nears the listener, we can start playing more of it in the side speakers. When the sound source is coincident with the listener, it can be played in all seven (or five) speakers. And once it passes, we can smoothly transition the sound to the rears, dropping the front gains to zero as it recedes behind the listener.

We can do this kind of thing and more if we model a sound source not as a point on the speaker circle but as an *arc*. Looking at it another way, we can think of each sound source as having an arbitrary shape in 3D space, and its *projection* onto the speaker circle *subtends* a certain angle, defining a “pie wedge” shape within the circle. This is analogous to the concept of *solid angle* often used in the calculation of ambient occlusion in 3D graphics—see [http:// en.wikipedia.org/wiki/Solid\_angle](http://en.wikipedia.org/wiki/Solid_angle) for details.

We’ll call the angle subtended by an extended sound source the *focus angle*, and we’ll denote it *α*. A point source can be thought of as an “edge case” in which *α* = 0. The focus angle is depicted in Figure 13.31.

To render a sound with a nonzero focus angle, we must first determine the subset of speakers that either intersect its projected arc on the speaker circle, or are immediately adjacent to the arc. Then we must divide the sound’s intensity/power among these speakers in order to induce the perception of a phantom image that extends across the projected arc.

We can divide the sound amongst the relevant speakers in various ways. For example, we could arrange for all the speakers that lie *within* the focus “pie slice” to receive equal maximum power, and then apportion less of the sound to the two speakers immediately adjacent the arc to create a fall-off. But no matter how we do it, we must remember to always obey the constant power pan law. So, we must set the gains in such a way that the sum of their



**Figure 13.31.** The focus angle *α* defines the projection of an extended sound source on the speaker circle.

squares (i.e., the sum of their powers) equals the squared gain of the original unpanned sound source.

#### 13.4.4.6 Dealing with Verticality

In both stereo and surround sound set-ups, the speakers all lie roughly in a horizontal plane. This arrangement makes it tricky to position sounds above or below the plane of the listener’s ears.

The ideal of course would be to model a true “periphonic” sound field by using a spherical speaker arrangement. The little-known *Ambisonic* technology [(http://en.wikipedia.org/wiki/Ambisonics](http://en.wikipedia.org/wiki/Ambisonics) is capable of accomodating both planar and spherical speaker arrangements. However, it is not supported by any game console—at least not yet. So we’ll need to make do with a planar speaker arrangement.

It turns out that the concept of *focus* can be leveraged to simulate some degree of verticality in our sound imagery. We simply *project* all sounds onto the horizontal plane, and then use a nonzero focus angle for any sounds whose projections fall too close to or within the speaker circle. An elevated sound that is far away will be rendered in virtually the same way as one that is not elevated. But as the elevated sound passes overhead, we blend it across multiple speakers, thereby producing a phantom image within the speaker circle. If we combine this with distance-based attenuation and frequency-dependent atmospheric absorptions, we can provide the listener with enough cues to make the sound seem to be located above or below the listener.

#### 13.4.4.7 Further Reading on Pan

The basics of the constant power pan law can be found here: [http://www. rs-met.com/documents/tutorials/PanRules.pdf.](http://www.rs-met.com/documents/tutorials/PanRules.pdf) The following site is also a great resource on the topic: [http://www.music.miami.edu/programs/mue/ Research/jwest/Chap\_3/Chap\_3\_IID\_Based\_Panning\_Methods.html.](http://www.music.miami.edu/programs/mue/Research/jwest/Chap_3/Chap_3_IID_Based_Panning_Methods.html)

The paper entitled “Spatial Sound Generation and Perception by Amplitude Panning Techniques” by Ville Pukki of the Helsinki University of Technology, available at [https://aaltodoc.aalto.fi/bitstream/handle/123456789/ 2345/isbn9512255324.pdf?sequence=1,](https://aaltodoc.aalto.fi/bitstream/handle/123456789/2345/isbn9512255324.pdf?sequence=1) provides a clear description of the spatialization problem and outlines the vector based amplitude panning (VBAP) method, as well as providing an extensive bibliography for further reading.

David Griesinger’s paper, “Stereo and Surround Panning in Practice,” also makes for a very interesting read; it is available at [http://www. davidgriesinger.com/pan\_laws.pdf.](http://www.davidgriesinger.com/pan_laws.pdf) David’s website is chock full of research on sound perception and audio reproduction technologies.

### 13.4.5 Propagation, Reverb and Acoustics

Even if we were to implement distance-based attenuation, pan and Doppler, our 3D sound engine still wouldn’t be able to generate a realistic soundscape. This is because a lot of the auditory cues we humans use to sort out what kind of space we’re in come from the early reflections, late reverberations and head-related transfer function (HRTF) effects caused by sound waves taking multiple paths to reach our ears. The term “*sound propagation modeling*” can be applied to any technique that is designed to take into account the ways in which sound waves propagate through a space.

Many different approaches are used, both in research and in interactive media and games. These technologies fall into three basic categories:

* *geometric analysis* attempts to model the actual pathways taken by sound waves,
* *perceptually based models* focus on reproducing what the ear perceives using an LTI system model of the acoustics of a listening space, and
* *ad hoc methods* employ various kinds of approximations to produce reasonably accurate acoustics with minimal data and/or processing bandwidth.

The following paper does a good job of surveying many of the techniques that fall into the first two categories: [http://www-sop.inria.fr/reves/Nicolas. Tsingos/publis/presence03.pdf.](http://www-sop.inria.fr/reves/Nicolas.Tsingos/publis/presence03.pdf) In this section, we’ll briefly discuss LTI systems modeling, and then turn our attention to a few ad hoc methods, because they tend to be more practical for use in real games.

#### 13.4.5.1 Modeling Propagation Effects with an LTI System

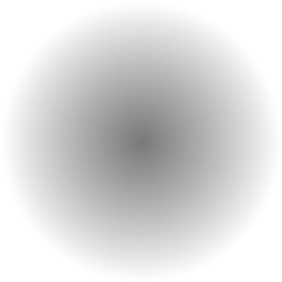
Imagine that I am standing in a room containing various objects made of various materials. A sound is made in the room. It reflects and diffracts and bounces around the room, and eventually reaches my ears. If you think about it, it doesn’t really matter which specific paths those sound waves took. The only thing that affects my perception is the specific superposition of the dry direct sound waves and the various time-shifted and possibly muffled or otherwise altered wet indirect waves.

It turns out that all of these effects can be modeled with a linear timeinvariant (LTI) system. Theoretically, if we could measure the *impulse response* of the room for a given pair of points that represent the source of the sound and the listener, we can determine exactly how *any* sound we might play at that source location should sound if heard at the listener position. All we need to do is convolve the dry sound with the impulse response!

*p*wet(*t*) = *p*dry(*t*) ∗ *h*(*t*)*.*

This technique seems like a silver bullet at first blush. However, it is actually more difficult and less practical than it may at first seem. It’s pretty easy to determine the impulse response of a space in real life—you can record the sound of a short “click” that approximates the unit impulse *δ*(*t*), and the recorded signal will approximate *h*(*t*). But in a virtual space, we’d need to perform a complex and expensive simulation of each play space in order to determine *h*(*t*). Also, to model the room’s acoustics accurately, we’d need to perform this calculation for a large number of source-listener point pairs throughout the game world, and once calculated the size of this data would be immense. Finally, the operation of convolution is itself not inexpensive, and game consoles and sound cards have in the past lacked the horsepower to do this for every sound in the game in real time.

Modern gaming hardware is getting more powerful all the time, and a convolution-based approach to propagation modeling is becoming more feasible. For example, Micah Taylor et al. created a real-time demo of convolution reverb that produced promising results—see [http://software.intel.com/ en-us/articles/interactive-geometric-sound-propagation-and-rendering.](http://software.intel.com/en-us/articles/interactive-geometric-sound-propagation-and-rendering) That said, most games still don’t use this approach, but instead they rely on various ad hoc methods and approximations to model environmental reverb.



**Figure 13.32.** It’s a good idea to cross-blend between reverb settings based on the position of the listener.

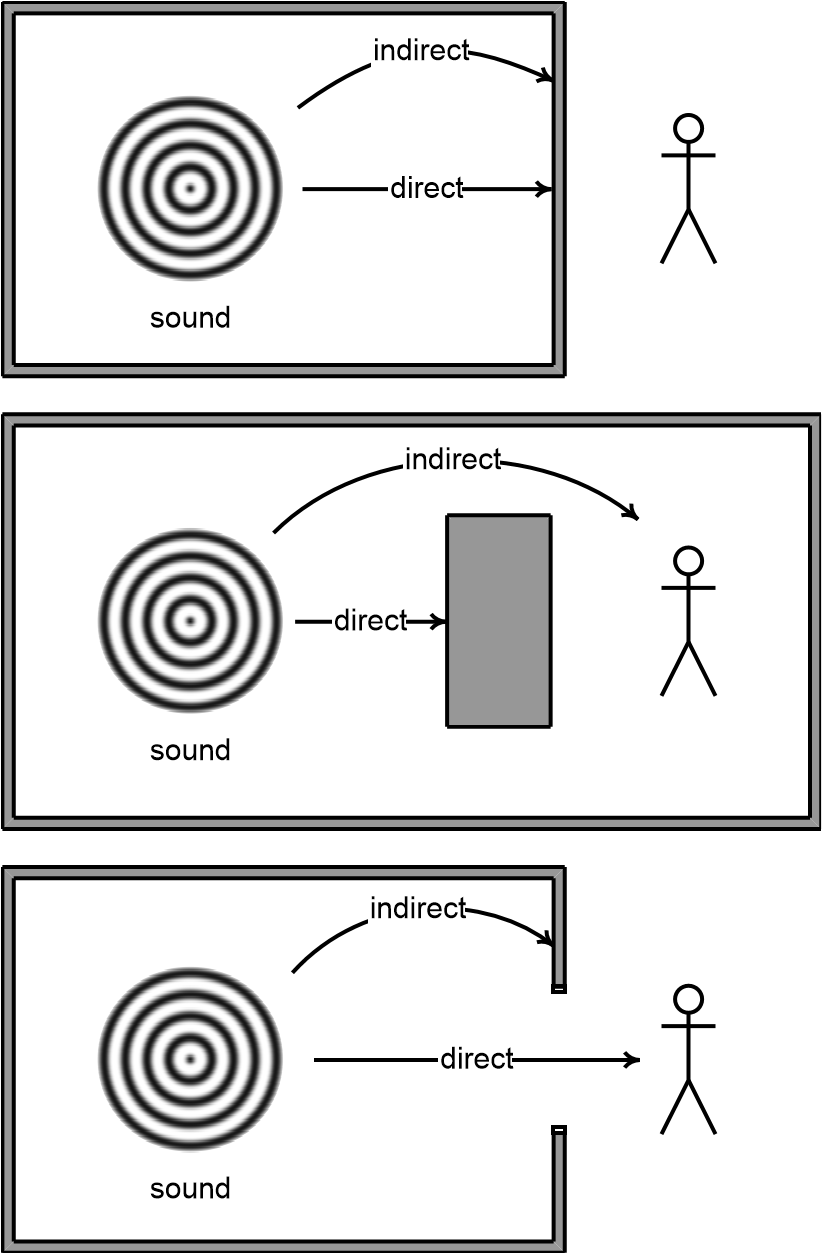
#### 13.4.5.2 Reverb Regions

One common approach to modeling the wet characteristics of a play space is to annotate the game world with manually placed regions, each of which is tagged with appropriate reverb settings such as pre-delay, decay, density and diffusion. See Section 13.1.3.4 for a discussion of these parameters. As the virtual listener moves through these regions, we can light up the appropriate reverb mode: If the player enters a large tiled room, we can bump up the echos; when the player enters a small closet, we can virtually eliminate the reverb to produce a very dry sound.

It’s a good idea to smoothly cross-blend between reverb settings as the listener moves through the play space. We can use simple linear interpolation to perform this cross-blend for each parameter. The blend percentage is best calculated using a measure of how far “into” the region the listener is. For example, imagine moving between an outdoor space and an indoor space through a doorway. We could define a region around the doorway within which the blend occurs. If the listener is entirely outside the blend region, the blend percentage should yield 100% of the outdoor reverb settings and 0% of the indoor settings. If the listener is standing at the halfway point within the blend region, we’d want a 50/50 mix of the reverb settings. Once the listener passes out of the blend region inside the building, we’ll have reached a 0% outdoor / 100% indoor blend. This idea is illustrated in Figure 13.32.

#### 13.4.5.3 Obstruction, Occlusion and Exclusion

When using regions to define the acoustics of our play spaces, we typically assign a *single* impulse response function or a *single* collection of reverb settings to each region. This captures the essence of each play space (e.g., large tiled hall, small closet lined with coats, flat outdoor plain, etc.). But it results in a less-than-perfect reproduction of the acoustics that arise due to obstacles. For



**Figure 13.33.** From top to bottom: occlusion, obstruction and exclusion.

example, imagine a square room with a large pillar in the center. If a sound source is located in the corner of the room, a listener will perceive a very different timbre as he or she moves about the room, depending on whether the direct path is obstructed by the pillar or not. If we use a single set of reverb parameters for this room, we cannot capture these subtleties.

To address this problem, we can attempt to model the geometry and material properties of the environment in some way, determine how sound waves are affected by the obstacles in their path, and then use the results of this analysis to alter the “base” reverb settings associated with the room.

Figure 13.33 shows the three ways in which the objects and surfaces in the game world can affect the transmission of sound waves:

* *Occlusion.* This describes a situation in which there exists no unfettered path from the sound source to the listener. A listener might still be able

to hear a fully occluded sound, if for example there is only a thin wall or door between it and the source of the sound. Either the dry and wet components of an occluded sound are both attenuated and/or muffled, or the sound is entirely silent from the point of view of the listener.

* *Obstruction.* This describes a case in which the direct path between the sound and the listener is blocked, but an indirect path is available. Obstruction can occur for example when a sound source passes behind a car, pillar or other obstacle. The dry component of an obstructed sound is either entirely absent or greatly muffled, and the wet component may be altered as well to account for the sound waves having to take a longer, more reflected path to the listener.
* *Exclusion.* This describes a case in which there is a free direct path between source and listener, but the indirect path is compromised in some way. This can happen if a sound is produced in one room and passes through narrow opening such as a door or window to reach the listener. In an exclusion situation, the dry component of the sound remains unaltered but the wet component is attenuated, muffled or, for very narrow openings, entirely absent.

#### *Analyzing the Direct Path*

Determining whether the direct path is blocked or not is not difficult. We simply cast a ray (see Section 12.3.7.1) from the listener to each sound source. If it is blocked, the direct path is occluded. If not, it is free.

If we wish to model sound transmission *through* walls and other obstacles, ray casting can still be used. We cast a ray from source to listener, and for each contact we query the material properties of the impacted surface to determine how much of the sound’s energy it absorbs. If it allows some energy to pass through, we can cast another ray starting on the other side of the obstacle and continue tracing the path to the listener. Once all of the sound’s energy has been absorbed, we can conclude that the sound cannot be heard. But if the ray makes it all the way to the listener without losing all sound energy, we can attenuate the gain of the dry sound component by the corresponding amount to simulate transmission of the sound.

#### *Analyzing the Indirect Path*

Determining whether the indirect path is occluded is a much more difficult problem. Ideally, we’d perform some kind of search (A\* perhaps) to determine whether or not a path exists from the source to the listener, and also how much attenuation and reflection is introduced by each viable path. In practice, this *path tracing* method is rarely used because it is processor- and memoryintensive. And at the end of the day, we game programmers aren’t really interested in creating physically accurate simulations that will win us Nobel prizes in physics. We merely want to produce a soundscape that is *immersive* and *believable.*

Never fear, all is not lost. There are all sorts of ways in which we can obtain an *approximate* model of the indirect path of a sound. For example, if we are using *reverb regions* to model the overall acoustics of the various spaces in our game (see Section 13.4.5.2), we could leverage these regions to determine whether an indirect path exists. For example, we could use some simple rules of thumb:

1. If the source and listener are in the same region, assume an indirect path exists.
2. If the source and listener are in different regions, assume the indirect path is occluded.

Using these assumptions combined with the results of our direct path ray cast, we can differentiate between the four cases: free, occluded, obstructed or excluded.

#### *Accounting for Diffraction*

When any wave passes through a narrow opening or interacts with a corner, it spreads out as shown in Figure 13.34. We call this phenomenon *diffraction.* Because of diffraction, sounds can be heard around corners as if a direct path existed, as long as the angular difference between the direct path and curved path is not too great.

One way to determine whether sound can diffract in order to reach the listener is to cast a few “curved” rays around the central “direct” ray. Most collision engines don’t support curved path tracing, but we can emulate a curved

**Figure 13.34.** Diffraction causes the dry component of a sound to be clearly audible even when the direct path is blocked.



**Figure 13.35.** Curved ray casts can be approximated using multiple straight-line rays.

path by using multiple straight-line ray casts. Figure 13.35 shows a simple example, in which five rays are cast from the sound source to the listener—one direct ray, plus two “curved” traces comprised of two straight-line ray casts each. Technically speaking we’re employing a *piecewise-linear approximation* to each curved path we wish to trace.

If the direct ray is occluded but the curved traces can “see” the listener, this tells us that the listener is within the “diffraction region” around a nearby corner, and should hear the sound as if it is not occluded.

#### *Applying the Model Using Reverb and Gain*

Thus far, we’ve discussed how to determine whether the direct and indirect paths are blocked or not. This analysis can also tell us something about the acoustic impact of an occlusion or obstruction. (For example, sound passing through a wall can be muffled; sound taking a long “bouncy” path might introduce a lot of reverb.) The question now is: How do we apply this knowledge when rendering the sound?

One simple approach is to simply attenuate the dry and wet components of the sound individually, based on whether the direct or indirect paths are totally or partially blocked, respectively. To finesse the results, we can also apply more or less reverb to each component of the sound, based on whatever heuristic information we gathered when determining the path(s) taken by the sound. The needs of every game are different, so this is one of those times when trial and error is your best and only option!

#### *Blending Obstructed Sounds*

If you were to go off and implement everything we talked about in the sections above, you’d notice a glaring problem. As a sound source moves between the four states described above—for example, from being free to being obstructed—the timbre and loudness of the sound will seem to “pop.” There are a number of ways to smooth out such transitions. You could apply a little *hysteresis*, meaning that you delay the response of the sound system to changes in the obstructed state of each sound, and then use this short delay window to smoothly cross-blend between the two sets of reverb settings. But the delay might be noticeable, so this isn’t an ideal solution.

For the *Uncharted* series and *The Last of Us*, Naughty Dog’s senior sound programmer Jonathan Lanier invented a proprietary system that he called *stocastic propagation modeling.* Without giving away any trade secrets, I can tell you that this system involves casting a bunch of rays to each sound source, some direct and some indirect, and accumulating these hit/miss results over many frames. From this data, we generate a probabalistic model of the degree of occlusion experienced by both the dry and wet components of each sound source. This allows us to smoothly transition a sound from being fully obstructed to fully free without noticeable “pops.”

##### 13.4.5.4 Sound Portals in *The Last of Us*

For *The Last of Us*, Naughty Dog needed a way to model the actual pathways that sounds take through the environment. If an enemy NPC is speaking while standing in a long hallway that connects to the room the player is in, we wanted to be able to hear the sound of his voice coming from the *doorway*, not “through the wall” along a straight-line path.

To do this, we used a network of interconnected regions. There were two kinds of regions: *rooms* and *portals*. For each sound source, we found a path from the listener to the sound by using connectivity information provided by the sound designer when laying out the regions. If both the sound source and listener were in the same room, we’d use the tried and true method of performing obstruction/occlusion/exclusion analysis that we used on the *Uncharted* series. But if the sound source was in a room directly connected to the listener’s room via a portal, we would play the sound *as if* it were located in the portal region. We found that we only needed to go “one hop” in the room connectivity graph to make this work for all real situations that arose in the game. Obviously I’m leaving out a lot of important details here, but Figure 13.36 illustrates the basics of how this system worked.

##### 13.4.5.5 Further Reading on Environmental Acoustics

Audio propagation modeling and acoustics analysis are areas of active research, and more and more advanced techniques are being applied in the game industry as hardware capabilities continue to improve. A few links are listed below to whet your appetite, but a Google search for “sound propagation” or “acoustics modeling” will provide many more hours of enjoyment!



Portal

Region

Fake

Sound

Source

**Figure 13.36.** The portal-based audio propagation model used in *The Last of Us* by Naughty Dog,

Inc.

* “Real-Time Sound Propagation in Video Games” by Jean-François Guay of Ubisoft Montreal: [http://gdcvault.com/play/1015492/Real-time -Sound-Propagation-in;](http://gdcvault.com/play/1015492/Real-time-Sound-Propagation-in)
* “Modern Audio Technologies in Games” presented at GDC 2003 by A.

Menshikov: [http://ixbtlabs.com/articles2/sound-technology;](http://ixbtlabs.com/articles2/sound-technology)

* “3D Sound in Games” by Jake Simpson: [http://www.gamedev.net/page/ resources/\_/technical/game-programming/3d-sound-in-games-r1130.](http://www.gamedev.net/page/resources/_/technical/game-programming/3d-sound-in-games-r1130)

### 13.4.6 Doppler Shift

As we saw in Section 13.1.3.5, the Doppler effect is a change in frequency that’s dependent upon the *relative velocity* between source and listener: **v**rel = **v**source − **v**listener. This frequency change can be approximated by simply timescaling the sound signal. This results in the “chipmunk effect” with which Alvin and the Chipmunks have made us all so familiar—by speeding up a sound, the pitch also rises. Because our sound signals are digital (i.e., sampled discrete-time signals), this kind of time scaling can be accomplished via sample rate conversion (see Section 13.5.4.4). However, this is not strictly the correct thing to do, because the speeding up or slowing down of the sound can become noticable.

The ideal solution is to apply a pitch shift without affecting the time axis. This can be done in a number of ways, including the *phase vocoder* and *time domain harmonic scaling* approaches. A complete description of these techniques is beyond our scope here, but you can read more about them at [http://www. dspdimension.com/admin/time-pitch-overview.](http://www.dspdimension.com/admin/time-pitch-overview)

Time-independent pitch shifting technology is an extremely powerful thing to have in your audio engine, in part because it also allows you to perform frequency-independent time scaling. So not only can you alter the pitch of sounds without changing timing for Doppler, you can also speed up or slow down sounds without altering their pitch for all sorts of other cool effects.