Audio Engineering Society

Conference Paper

Presented at the International Conference on
Immersive and Interactive Audio
2019 March 27–29, York, UK

Interactive Audio Geometry

Simon N Goodwin, **Interactive Audio Tech Consultant,** Warwick, UK

SimonNGoodwin13 @ gmail.com

ABSTRACT

One of the tough but rewarding challenges of interactive audio synthesis is the continuous representation of reflecting and occluding objects in the simulated world. For decades it's been normal for game engines to support two sorts of 3D geometry, for graphics and physics, but neither of those is well-suited for audio. This paper explains how the geometric needs of audio differ from those others, and describes techniques hit games have used to fill the gaps. It presents easily-programmed methods to tailor object-based audio, physics data, pre-rendered 3D audio soundfields and reverb characteristics to account for occlusion and gaps in the reverberant environment, including those caused by movements in the simulated world, or the collapse of nearby objects.

1 Introduction

This paper explains how to control the directional and reverberant properties of object-based audio sources and soundfields, so that they sound realistic in the world. It draws upon the author's experience making mass-market console and PC games, and insights from his forthcoming book *Beep to Boom*, part of the ***AES presents...*** series from Focal Press.

2 Graphics Geometry

There are two categories of geometric object in an interactive 3D world: **mobile** objects, like animals and vehicles, move around, while **static** objects are buildings or fixed parts of the terrain. Both types are authored in 3D graphics packages like *Maya* or *3D Studio Max*, and imported into a game in two parts – as a skeletal **mesh** of connected triangles which describe the outline of the object, and as **texture maps** which cover the facets, superimposing colour images and surface details which improve lighting and smooth out the polygonal edges.

Audio can ignore the detail in these maps, needing only a representation of the reflective properties of each triangular facet to model acoustic reflections and occlusion. But there are far more triangles in a graphical object than needed for audio purposes. On current console hardware the typical budget for a single player-character is about 100,000 triangles; racing games use around 250,000 polygons for each car, and the total number of facets in a scene, rendered 60 times a second, may exceed ten million, including informational overlays, shadows, particles and similar decorations. [1]

Such high-resolution meshes deliver finely-detailed close-up graphics, using massively parallel rendering hardware, but they're gross overkill for audio. The computational expense of working out the interaction of each sound source and every potentially-reflecting graphical facet far exceeds the real-time capability of PCs and consoles.

3 Physics Geometry

Similar discrepancies of need apply to detecting collisions between mobiles, or between mobile and static objects. Rigid body physics systems use simplified object outline representations, just enough to stop objects intercepting or overlapping one

another, and to determine which pairs are touching and the direction and intensity of their collision. Many visible objects are ignored, providing they are or can be nudged within the bounds of another, usually larger one. Collapses are handled by swapping entire meshes, usually under cover of dust.

Audio runtimes take advantage of this information to play contact sounds. A collision reports the 'materials' and a pair of sounds is triggered – for instance when a bike hits a post, wood and metal contact samples are played from the point of interception, with assets and volumes tailored according to the collision force and angle of incidence. If the angle is obtuse, a looping 'scrape' sound may be chosen and updated in pitch and volume while the contact continues.

With care, physics geometry may be used to model audio occlusion between game mobiles. In racing games, other cars reflect the sound of your own as they try to pass. Codemasters' BAFTA-winning and multi-million selling *RaceDriver Grid* [2] game models this dynamically. The game audio system sends out a bundle of eight rays – two vertical, up and down, and six in a plane around the player – to detect which objects, static or mobile, are closest, and their direction. The top vertical trace is dubbed the 'unicorn ray' as it is tilted in the direction of travel to sense oncoming objects, such as bridges and gantries, sooner than might otherwise happen.

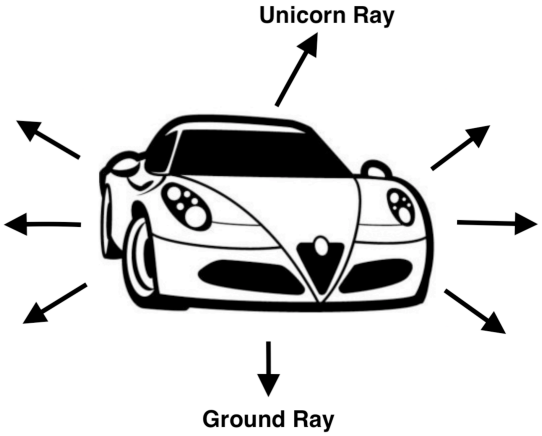

Figure 1 - Ray-tracing dynamic reflections

Even eight rays are computationally expensive. The length of the rays is limited, to save processing time by allowing the broad phase of the collision handler to sift out distant objects, and the rays are submitted together as a bundle to facilitate optimisations – it's quicker to handle a batch of rays from a common source than to process each ray individually.

An even more critical performance boost comes by not waiting for the results. Ray-tracing is performed outside the main update thread of the game, by separate threads which run asynchronously, taking advantage of specialist co-processors or multiple processor cores.

Even if the contact results arrive a few milliseconds late, that's soon enough to select and play suitable sounds without the player noticing the lag; such relatively low-priority queries help to balance the processing load and maintain a steady frame-rate for everything else. If there's time to spare, the results may arrive promptly, but if the system is fully-loaded this additional audio work may be bumped off to the next frame update, smoothing out the spike in demand without delaying graphics or the direct-path spatialisation of moving sound sources.

It's practical to use ray-tracing for the main player object in modern games, but unless you have a very small world, super-fast processors or little else going on, it remains unacceptably slow to cast rays from every source object to every listener, every frame, just to find out what might be in the way.

Other limitations of using collision geometry for sound vary depending upon the type of game. A surface might obstruct sound, but not bullets, so it will lack collision data in a shooting game, to improve speed. Race-track designers don't mark up every trackside object, but only the ones which are likely to get in the way of cars or large cast-off bits like tyres and bumpers.

An Armco crash barrier less than a metre tall will stop a car, yet it allows sounds to pass above and below. Markup adequate for physics might make this an unrealistically solid obstacle to audio. Objects further from the road still ought to reflect

sounds, but usually won't have collision physics because it's expensive and not needed for anything except audio. Some collidable objects, like fence-posts or wire mesh, might hardly impede audio. We don't expect to hear reflections from those, although they're physical obstructions. Conversely, sound waves propagate around corners, so even if a path is blocked to light some audio should leak through.

The reflectivity of a surface affects audio – a grassy knoll reflects less than a glassy shopfront – but non-audio designers are usually satisfied just to know if there's an obstruction. Marking up and checking the correct surface properties becomes a big burden for the audio team. Even if the physics, graphics or level design teams are given a way to tag obstructions with their audio properties, they may lack the skill or the will to take on this unfamiliar responsibility.

Two control values are needed – one expressing the reflectivity as a proportion, and another controlling the timbre of reflections, perhaps moderated by the angle of incidence. This may be an index into a table of materials and appropriate filter presets, or a direct tone control setting attenuation at a certain frequency. The indirect table approach is preferred because it allows each surface type to have arbitrary properties, such as a frequency response curve, and makes it easy to tweak all surfaces of a given type without selecting them individually. Often there's an existing materials database which can be used to select suitable 'wood' or 'metal' samples, and extended to include audio reflectivity.

It's one thing including support for audio properties, but quite another to get the visual artists and level designers to use it. It's often left set to the default, or whatever the previous object required. It's common for such mistakes to go un-noticed until the object is in a busy simulation, and then hard to work out exactly what to fix, assuming someone notices the problem. Compared with sliding through a tree or falling through the floor, audio reflections are a minor concern. Even though the correct handling of occlusion and reverberation is as important to sound as correct shadows and shading are to graphics, it's less immediately obvious when it goes wrong.

4 Audio Geometry

Ray-tracing is essential for mobile interactions, but neither graphics nor collision meshes give all the information we need for realistic volumetric audio.

Early attempts at side reflections were laboriously implemented in *Colin MacRae Rally* 2004 and 2005 games, using an undocumented feature of the *OpenAL* Audio API to implement directional reverberation. [3,4] Later Codemasters games adapted this technique for Microsoft's *Xaudio* and Sony's *Multistream* console audio systems, and Blue Ripple's cross-platform PC/mobile *Rapture3D.* [16]

Creative Labs' 'Environmental Audio Extensions' for *OpenAL* and *DirectSound3D* implemented up to four high-quality synthetic reverbs. Their additional 'direction' vector, also implemented in Rapture3D [16], neatly controls the spatial distribution of reverberation around the listener. If the vector has a magnitude of zero, the default, reverb is omni-directional. The maximum magnitude is 1, which directs all reverb to come from the specified direction. The following statement, expressed in the C language customarily used for audio systems programming, computes the corresponding magnitude for an angle **a** expressed in radians:

```
directionalMagnitude = (2*sinf(0.5*a))/a;
```

For example, the vector (-0.637, 0, 0) places the reflections over a 180˚ semicircle to the left of the listener, (+0.637, 0, 0) picks the opposite side; 0.90 (PI/2 radians) gives a 90˚ spread wherever you wish.

This sounded good, and the side-reflection technique has since been adopted in other racing games, but created a great deal of work for the designer, Ben McCullough. To avoid the need for real-time ray-tracing, which would have been prohibitively slow in 2004, Ben spent months creating markup text files indicating the most appropriate I3DL2 [5] reverb preset to select each side of each 'cats-eye' marker distributed at short intervals for hundreds of miles along the 88 point-to-point rally courses included in the game. [3]

Codemasters' later *Formula 1* racing games use a spatial arrangement of 'reflection lines' to modulate the timing and timbre of tapped delays applied to the output of granularly-synthesised engine sounds. [3] Such lines can be computed at runtime, or earlier. Tom Zuddock's article *Virtual Audio Through Ray Tracing* describes a simple approach to prescanning a room's geometry to work out reflection sources and delays [6] and Dave Malham's *Reflector* project seeks and applies first reflections on the fly. [7]

A more general solution is to add a third class of geometry, in the form of 'sound area meshes', to fill the gaps – or leave them open if they're porous to audio! A mesh is a group of spatially-distributed control points with properties which can be interpolated for any position therein.

An effective implementation of this concept appears in the Saracen 2 world editor and game engine, implemented by Rob Baker at Attention to Detail Ltd, for games like *Lego Drome Racers*. [8] Rob took a system originally intended for lighting and adapted it to model both the occlusion of widespread sounds like alarms, and the distribution and tone of reverberations.

The same C++ codebase supported two types of audio mesh, as well as lighting effects. Meshes could just as well be used for any volumetric parametric control, e.g. occlusion. **Sound Area Meshes** determine where particular sounds may be heard, while **Effect Area Meshes** control I3DL2 [5] reverb parameters, smoothly morphing between control values as the listener moved within and

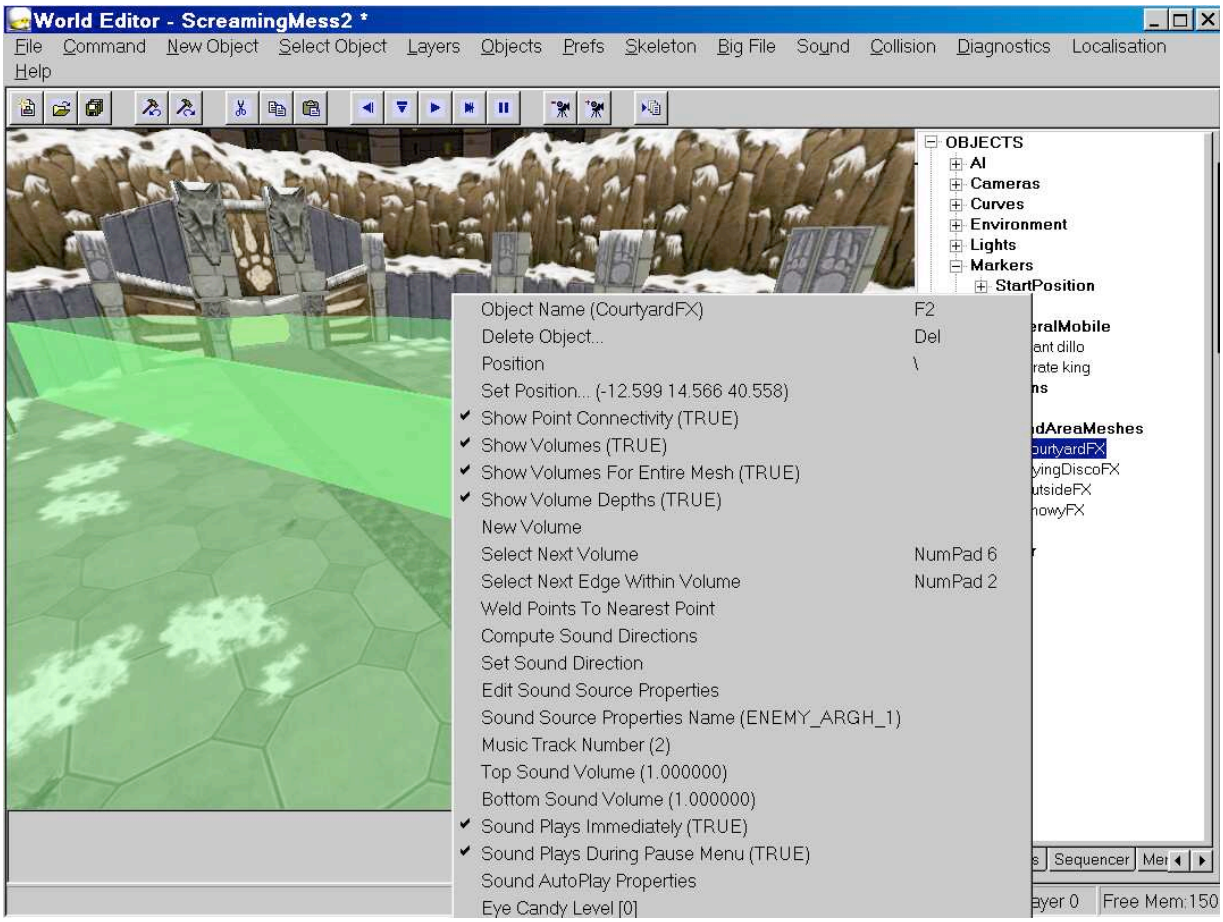

Figure 2 - Alarm Sound Area Mesh

between the meshes. I3DL2 reverb controls are exposed in many game APIs including *CryEngine, FMOD, Wwise, Xaudio, Unity* and *Unreal Engine 4*.

Both these types of mesh are designed into the environment using a special PC-hosted version of the game which incorporated a 'world editor' that allowed 3D objects to be added and moved around.

Meshes are visualised as semi-transparent volumetric overlays, so they can be seen in context. Saracen 2 used a preset alpha-blend, but the degree of transparency may vary to indicate the intensity of sound or reverberation, tailing off at the edges.

Figure 2 shows a mesh placed in a courtyard, used both to trigger and control the volume of an alarm sound played therein. The menu overlay shows the mesh properties, such as its location, visualisation and which sound sample or stream to play.

Each corner of the mesh had properties associated with it – for a playing sound it could be as simple as the required volume at that point, whereas for reverb it might set early and late reflection levels or any of the other I3DL2 properties – with the proviso that not all of those could be changed at runtime without audible glitches. Overlapping cross-fades were sometimes needed to hide these.

At runtime the game calculates which triangle of the mesh the listener is inside and interpolates between those properties to get a representative set for that position. Figure 3 outlines the process of interpolation in such a triangle; distances are in grey, vertex weights and interpolated values are in black.

The simplest mesh has one full-on point in the middle and a ring of at least three points round the outside with controls set to mute the sound or effect.

As the listener enters the ring and moves towards the middle, sound parameters like volume, timbre and reverberation are adjusted accordingly. These parameters should be expressed in psychoacoustically weighted units, such as logarithmic attenuations, so that the interpolation is perceptually as well as arithmetically linear.

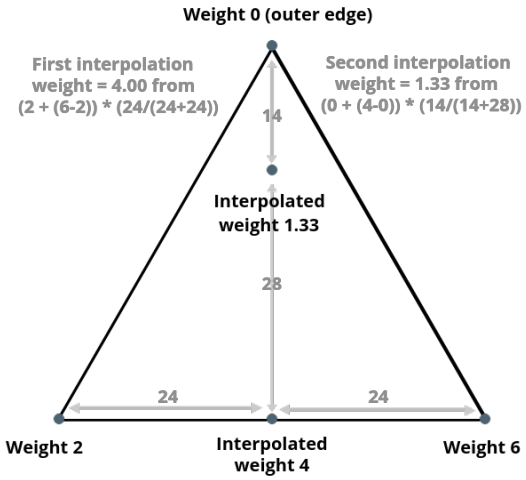

Figure 3 - 2D Audio property interpolation

5 Control Points

At first this does little more than a sphere set at Max Distance (by convention, the range at which a sound becomes inaudible) around a source, albeit with a potentially irregular outline. But as control points are added, meshes become far more capable. An intermediate ring of points allows the inner and outer volume curves to vary.

Additional points approximate any curve, allowing different shapes which depend upon location as well as distance and aesthetic factors. For instance, ambient sound could fade off at the mouth of a cave, or a sheltered area could have a quiet point within it, and varying levels around that corresponding to the occlusion caused by static geometry.

If that shelter is damaged, the mesh can be edited or replaced in the same way that a pristine building might be replaced with a shattered one by swapping physics and graphics geometry, usually under the cover of a cloud of smoke – or the audible equivalent, a big explosion. Editing is more powerful, just like deformable geometry, but a lot more expensive at runtime, which is why both techniques are commonly used.

Baker adds, *"Although the meshes were designed in 3D so the level designer could position them around*

the required objects for reference, the actual processing was all done in 2D with the vertical component ignored. Of course, this greatly simplified the processing as we were only dealing with 2D triangles rather than potentially complex 3D volumes which would have been a nightmare to author at a design level anyway. Because our race tracks were all essentially just 2D with no vertically overlapping areas this was never really a design limitation." [9]

The interpolation technique can readily be extended to 3D by locating the four nearest control points to the listener, rather than the 3D triangle, either by traversing the mesh or simply scanning through all the vertices to find the three which define a 2D plane or four for a 3D volume – not forgetting to check for the special cases when they may all be on a line or in a plane.

This cache-friendly brute-force search is amenable to optimisation as there is no need for a full sort of all the points, nor to use square roots to determine the actual distances by Pythagoras's theorem – comparing the sums of the squares, computed with fast SIMD (Single Instruction, Multiple Data) vector arithmetic, is enough to find the closest.

Disjoint clusters of points can be grouped within bounding spheres, as part of the offline data-preparation process, providing a quick way to ignore irrelevant distant clusters with a single runtime test.

Meshes are a general way to describe the properties of the space between objects, acoustic or otherwise. Irregular meshes can be applied to reverberation, creating echoic pockets in an otherwise open world. Meshes can also be used to filter or suppress sounds from outside an arbitrary volume.

Mesh control points also embody directional information. If control points to the left of the listener have higher volume than those to the right, the sound can be panned left proportionately. Extend this approach to surround by setting a suitable 2D or 3D panning vector. As the sound is distributed around the mesh, Doppler pitch shifts should be disabled for this sound object.

Hand-authoring of audio meshes is a big job, especially as non-audio designers are prone to move objects around at any point in development of a game, even at late stages, to deliberately reduce visibility to keep a consistent frame-rate. Testing and re-authoring audio metadata after such changes is costly and error-prone. But just as 'static lighting' offline processes can rebuild light and shadow maps automatically, similar systems can regenerate audio occlusion meshes before the game is tested.

It doesn't matter that they're using fine-grained graphics geometry, or some hybrid of graphics, physics and custom data, because the offline process doesn't need to run in real time. Indeed, it can be distributed across several build systems, which might be doing similar rework for the physics system, such as deriving simpler bounding zones for graphics meshes. Most of the code you need may already exist.

It's wise for such processes to generate the same data as manual markup, so the output can be tweaked by hand later if necessary. Write a simple system that catches the bulk of the common cases and can be tweaked later just where it matters most. That's more practical than to rely on an automatic one which aims to detect and handle all the possible edge-cases and leaves no escape mechanism.

6 Cheap Shapes

If you can't afford full-blown audio geometry, or don't always need its flexibility, you may get by with an arrangement of simpler shapes, such as spheres, boxes and capsules, to control environmental audio. *Colin McRae DiRT* uses trigger boxes to time its context-driven speech, with adjustments for the player's speed. [15] *Grand Theft Auto V* uses almost a thousand spheres and arbitrarily-shaped boxes to delimit its ambience zones. [10]

The density of this sort of mark-up is increasing in the quest for greater immersion and realism, and as VR and AR experiences raise expectations. Its predecessor *Grand Theft Auto 4* incorporates only 87 such zones, marked up as axis-aligned boxes. These are quicker to process at run-time because there's no

need to rotate them, but less flexible and harder to author.

Spheres never need rotating, which helps to explain why pool and snooker games were amongst the first to go into 3D. Around 200 trigger spheres control the crowds, tunnels and local ambiences in the *Colin McRae Rally* mobile game. [11]

Figure 4 - Ambience Zones in Grand Theft Auto V [10]

7 Capturing Reverberation

Stuart Ross, sound designer on the BAFTA audio award-winning *Crackdown* and *Grand Theft Auto: Vice City* games, observes that *reverb is to sound as lighting is to graphics*. [3] It's thus very subjective, combinatorial, and influenced by many aspects of the scene. It greatly enhances a sense of immersion.

Rockstar North's *Grand Theft Auto V* uses three internal reverb buses for spatialisation, each four channels wide, preset for small, medium and large reverberant spaces. Audio objects in this massive 'open world' game can be routed to any of those, in varying proportions appropriate to the environment and distance from the listener.

Meshes can be used to control synthetic reverb depth and timbre but new techniques allow more realistic reverb, providing they also support customisation. Ambisonics has revolutionised spatial audio by allowing periphonic soundfields to be authored and orientated around the listener in games and especially virtual and extended reality titles. [17]

A public database of Ambisonically-captured 3D impulse responses has been created. As well as traditional hall and church echoes, the OpenAirLib repository includes outdoor captures from a ravine, tunnel portals, and a Finnish forest in summertime and snow, intended for interactive media use. Figure 5 illustrates the setup for one of these recordings. [13]

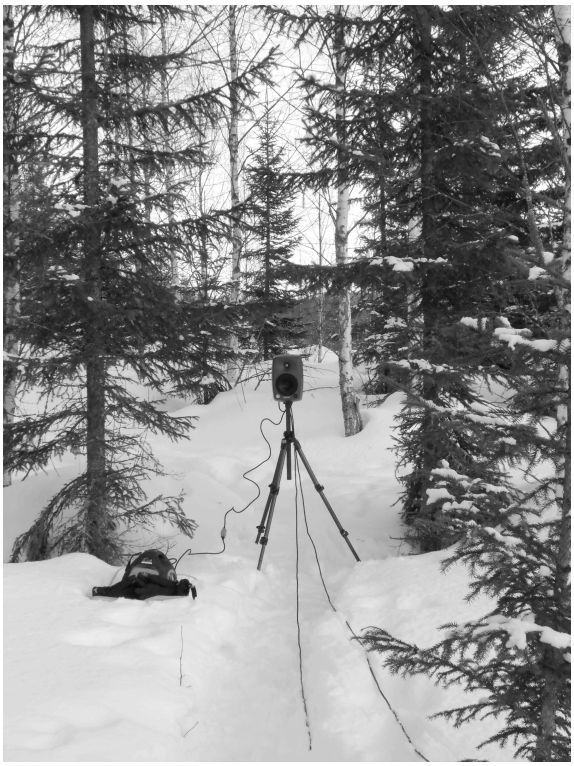

Figure 5 - AHRC-funded Ambisonic impulse response capture in Koni National Park for OpenAirLib.net [13]

The characteristic sound of a vehicle interior can be applied to an Ambisonic mix by convolving it with the 3D impulse response. The techniques to capture this are explored in a paper presented at SMCC. [12]

This is adequate – indeed transformative compared with conventional synthetic reverb. It takes good advantage of the fixed position of the driver in the cockpit. But it's still not as interactive as we'd like. Panel or window damage affects the directional characteristics and adds additional sound which may dominate the soundfield later in an event. Captured impulses can be tailored for each such configuration.

8 Directional Soundfield Manipulation

But what if we want to suppress sounds from a specific direction? This is useful if a large or nearby mobile object partially obscures sound from some directions, or when a gap in surrounding geometry means less reverberation in that direction. Such dynamic changes maintain listener interest and make the simulation more realistic and informative about the immediate spatial environment.

The following C statements sculpt the directional characteristics of a first-order ambience or impulse response. **fW**, **fX**, **fY** and **fZ** are the B-Format components, where **fW** is the omnidirectional part, and dir is a unit vector (length 1) pointing in the direction from which you wish to suppress sounds. These axes must be oriented in the conventional mathematical sequence – unlike video graphics engines, Ambisonics follows mathematical convention with a *vertical* Z axis.

Finally we need a 'dimming factor' **d**, where 0 makes no difference and 1 suppresses as much as possible of the signal in the chosen direction vector **dir**. Again in the usual C programming language, the original signal in the dimming direction, **s**, is

```
s = 0.5 * (1.414 * fW + dir.x * fX
         + dir.y * fY + dir.z * fZ);
```

This can be eliminated from the original soundfield by adjusting the component weights as follows:

```
float sd = s * d;
fW -= 0.707 * sd;
fX -= dir.x * sd;
fY -= dir.y * sd;
fZ -= dir.z * sd;
```

It's not perfect, because of the limited spatial acuity of first-order Ambisonics. This representation is not ideal for direct sound sources, but given the way sounds propagate indirectly as well as directly it's a quite realistic volume-only manipulation for varying ambient and reverberant soundfields. It's very cheap to implement, since it uses the gain adjustments associated with decoding of the existing soundfield: it needs just ten extra multiplications, three additions and four subtractions per four-channel update frame.

Higher-order systems can sift out and adjust more specific parts of the mix, including those above and below any chosen plane as well as directional parts.

This manipulation can be applied to recorded or synthesised Periphonic reverb, either by directly convolving Ambisonic impulse responses onto corresponding components of a mix of the same or higher order, or using the *AmbiFreeverb 2* [14] parametric 3D reverb, which offers control over scattering as well as spatial distribution.

This process, along with pre-delay, filtering and the insertion of computed early reflections, greatly increases the flexibility of impulse response reverb, making it more suitable for interactive applications. The process's low fixed overhead and scalability, from short mono delays via decorrelated W to 2D or 3D convolution, also commends its use on portable devices, which may vary in DSP power by a factor of 30 times across one platform like iOS or Android.

9 Conclusions

This paper has identified the key geometric data and runtime subsystems in modern computer games and considered their relevance to the dynamic representation of audible reflections and occlusion. It has shown that although existing data and tools are not ideally suited for audio processing they can be harnessed for immersive sound synthesis by a

pragmatically-tailored mixture of offline data refactoring, real-time queries and dynamic reconfiguration to meet the specialised needs of high-performance games, VR and similar interactive simulations in readily-scalable ways.

References

[1] S. N. Goodwin, *Beep to Boom: The Development of Advanced Runtime Sound Systems for Games and Extended Reality*, Routledge 2019; Chapter 22; ISBN 978 1138543904, www.simon.mooli.org.uk/b2b [Checked 2019-1-9]

[2] Codemasters Software Company, *RaceDriver Grid*, 2008 www.mobygames.com/game/grid [Checked 2018-11-9]

[3] S. N. Goodwin, *Beep to Boom: The Development of Advanced Runtime Sound Systems for Games and Extended Reality*, Routledge 2019; Chapter 21; ISBN 978 1138543904, www.simon.mooli.org.uk/b2b [Checked 2019-1-9]

[4] *OpenAL application programming interface:* www.openal.org [Checked 2018-11-9]

[5] *Interactive 3D audio rendering guidelines, Level 2.0 (I3DL2) specification:* www.iasig.org/wg/closed/icwg/21feb97.pdf [Checked 2018-10-29]

[6] T. Zuddock, "Virtual Audio Through Ray Tracing", *Dr. Dobb's Journal,* ISSN 1044-789X, December 1996

[7] D. Malham, *The Ambisonic Reflector*, sourceforge.net/projects/thereflector/ [Checked 2018-11-2]

[8] Lego Interactive, *Drome Racers*, 2002-2003 www.mobygames.com/game/drome-racers [Checked 2018-11-9]

[9] R. Baker, *Private correspondence with the author*, November 2017

[10] A. MacDonald, *The Sound of Grand Theft Auto V; Alistair, Game Developers Conference, San Francisco, 2014* www.gdcvault.com/play/1020587/The-Sound-of-Grand-Theft [Checked 2018-10-29]

[11] Codemasters Software Company, *Colin McRae Rally games:* en.wikipedia.org/wiki/Colin_McRae_Rally [Checked 2019-1-9]

[12] S. Shelley, D. Murphy, S. N. Goodwin, "Impulse Response Estimation for the Auralisation of Vehicle Engine Sounds using Dual Channel FFT Analysis"; *Proceedings of the Sound and Music Computing Conference 2013,* Logos Verlag Berlin, ISBN 978 3832534721 pure.york.ac.uk/portal/services/downloadRegister/25538798/Impulse_Response_Estimation_for_the_Auralisation_of_Vehicle_Engine_Sounds_using_Dual_Channel_FFT_Analysis. pdf [Checked 2018-11-09]

[13] *OpenAirLib impulse response library* www.openairlib.net [Checked 2018-11-09]

[14] B. Wiggins, M. Dring; AmbiFreeVerb 2, "Development of a 3D Ambisonic Reverb with Spatial Warping and Variable Scattering"; *AES International Conference on Sound Field Control* (July 2014) www.aes.org/e-lib/browse.cfm?elib=18307 [Checked 2018-11-9]

[15] Codemasters Software Company, *Colin McRae DiRT,* 2007, [Checked 2018-11-9] www.mobygames.com/game/dirt

[16] Blue Ripple Sound, *Rapture3D* for Microsoft Windows, Apple MacOS, iOS and Google Android*,* www.blueripplesound.com/products/rapture3d-universal-sdk [Checked 2019-1-1]

[17] M. A. Gerzon, "Periphony: With-Height Sound Reproduction"; *JAES,* ISSN 1549-4950, Vol. 21 Number 1, Feb 1973, pages 2–10, www.aes.org/e-lib/browse.cfm?elib=2012 [Checked 2018-11-9]