Game Audio : Coding vs. Aesthetics

Leonard J. Paul

info [at] lotusaudio.com

Abstract
The key to creating successful immersive game audio to balance the technical and creative elements. Too often one is sacrificed at the expense of the other due to their assumed opposing natures.

In the “good old days” of video game audio, one person would wear both hats of audio coder and sound designer/composer. In today’s video game world with escalating budgets and highly complex game platforms, the work must be split between several people, which threatens to increase this fundamental rift.

In an effort to bridge this problem we should approach audio production from the creative and technical viewpoints simultaneously to arrive at the desired goal of successful game audio.

Topics
1) Prototyping
2) Peer Review
3) Audio Control Parameters
4) Voice and Memory Usage
5) Tools
6) Automated Dynamic Mixing

Video Game Audio Process Levels
Level 1: Largely unaware of existing practices. Misapplication of practices.
Level 2: Aware of existing practices. Application of existing techniques.
Level 3: Proactive practice of techniques. Apply and create new effective techniques.

Usage Terms
The term “composer” is used to mean sound artist, sound designer or otherwise the person creating audio content for the game and deciding how it should play. “Coder” is a term meant to apply to the game audio programmer or otherwise the person responsible for realizing the structures which control the audio playback in the game.

Coding vs. Aesthetics

Many things have changed from the days of “bleeps and bloops” of yesteryear to our multi-channel 3D digital surround-sound audio systems of today, but the basic context in which we create video game audio has remained primarily unchanged. There is a fundamental left-brain vs. right-brain struggle when producing audio for an interactive medium: content and code.

In the good old days, these roles were fulfilled by one person. Rob Hubbard was not only a composer, but also hand-coded his own assembly language sound driver for his Commodore 64 titles. “Hip” Tanaka wrote the music for Metroid but also engineered the sound cards on which many Nintendo titles played. Later on in the late 80’s, when FM and MIDI reigned supreme, making audio for video games no longer required such computing prowess which opened the door to your average musician. Once streamed music became more of a reality in the 90’s, this caught the ear of the “serious” musician with experience in Protools. Video game audio had matured into a serious art form.

With US video games sales at $9.3 billion in revenues against Hollywood’s $8.1 billion in 2001, many film and television composers are now taking the plunge into video game music production. With today’s increasing budgets, headcount and hardware complexity, the task of audio production is often split at least between a composer and coder if not a whole audio team.

One of the issues which is often overlooked (or denied) is that video games are not movies and that interactive non-linear media is much different than its’ linear cousin. Currently, there is an increasing need to bridge the gap between content and code as they are pushed further apart by team size and reactive project scheduling.

I propose that we attack the problem from both sides at once and forget our current preconceptions of content and code. In the early days, the coder/composer might have had an easier time and could have used an informal approach, but with larger teams, we are forced to adopt some sort of formal methodology to unify our efforts. What is needed is a real “renaissance” to dissolve our perceptual barriers between science and art or code and aesthetics.

Overview

In each of the following areas, different process levels are presented and specific cases are examined to detail new methodologies in developing video game audio. These methods are examined to find how the composer might further stretch her own skills into controlling her own content and to discover how the coder might better aid the composer by understanding the composer’s process. The idea is not to transform one into the other, but allow for a free exchange of ideas. The goal is to combat the trend that with larger teams and conventional corporate culture that we become more isolated and less willing to share and learn with our co-workers. In contrast, we should favour an effective process of sharing and open methodologies produce new systems of working which suit the specific people who work there.

Prototyping

Level 1	A first prototype is quickly built, but due to timeline constraints, it evolves awkwardly into the final project.
Level 2	A prototype is made and later thrown out, but much of the code remains the same. Some view the prototype as a "waste of time."
Level 3	Multiple iterative prototypes are rapidly made and a final model is settled upon. The prototype is completely rebuilt and reorganized, using the best elements of the model. The intermediate modules are saved for future reference to show the progression to the final design.

Prototyping occurs after research, analysis and design have gone into determining what actually needs to get done for the audio. Prototyping is a good way to find out how the design addresses such ephemeral characteristics such as mood and pacing.

Environments

Constructing prototypes in a different environment than the final project's can make things go faster, but may allow you to construct a prototype which is difficult to actually implement in the game. Graphic object-oriented audio environments such as Pure Data, MAX/MSP and Reaktor are great ways to hear results quickly. A desirable side effect of the composer using such packages, is that they are actually high-level coding languages and may help the composer visualize and better understand the control structures used in code.

Edit

Do sketches/prototypes first to get a feel, then set it aside and review all the elements once a good balance between effort and results is reached. Restructure and edit out anything that doesn't need to be there and don't be too attached to code or content. If you hold onto everything, then you'll be worried about changing anything for fear it will destroy the quality of the rest of the material. Not effectively editing can cripple your creativity since you can spend more time fixing things which have to be thrown out later anyways (which isn't too inspiring). Even if it does work out, by that point you could be so bitter that it doesn't matter anymore.

Tricks

One way to attack this process is to get the sketch/prototype to a certain presentable point and pretend as though you were handing it off to someone else and then giving your own feedback to your "imaginary friend." If you think your project is confusing or needs explaining, then likely you have more editing to do, or you should get a fresh start entirely.

Traps

A definite goal is to not have others too attached to the prototype and to make it clear that it is just placeholder material or a quick "hack" to show how things might work out. I've definitely been guilty of hanging onto a prototype too long. It always seems like there isn't enough time to waste rebuilding something which already works. The end result is that it breaks later in the project when there is more stress on it and takes much longer to rebuild due to the layers of hacks which were used to keep it standing beyond it's natural lifespan!

Peer Review

Level 1	People give periodic informal feedback on the audio. The composer and coder act on the information independently. The composer and coder gain outside perspectives on the audio.
Level 2	Peers regularly evaluate the audio objectively describing the good and bad points. Composer and coder discuss feedback to distribute workload. The composer and coder gain additional inside perspectives on the audio.
Level 3	Composer and coder receive and participate in regular peer reviews and are able to objectively evaluate each other's work and their work as a sum. They work and aid each other in an effective synergistic manner.

As with most software engineering and process management ideas (including this paper) everything gets thrown out once the schedule looms. However, the less time one has, the more effectively one should attempt to work. A short time spent on improving process usually results in long term rewards. Peer review should be scaled to fit everyone's available time since if everyone believes they are wasting time and would rather be doing it than talking about it, then the criticism might be more about the process rather than the work itself!

Peer review shouldn't turn into a job review. The idea is to support one's co-workers rather than undermine them. Making reviews regular and round-robin can help keep things more balanced. Corporate culture can definitely make the peer review system more difficult since there's more of an idea of an inherent hierarchical order instead of a peer system. In most cases, small review groups work best to keep things agile and efficient.

Advantages

Given good scheduling and process, peer reviews can be invaluable for finding early design flaws, identifying pipeline bottlenecks and even silly mistakes. Objective criticism is best given by peers since the language is shared. Constructive positive feedback indicates to the person being reviewed what they shouldn't edit out and helps inspire confidence in their abilities.

Learning

Constructive peer review is a great way to learn. Often, someone will point out a flaw which no one else identified. This allows for an easy exchange of ideas so that others are not repeating the same mistake. It may even raise the level of the process since people may want to come up with a new approach to a problem spread the idea to the group. Without reviews, people are more prone to try to hide their mistakes rather than finding help. From time to time, someone will like something from someone else's process and adopt it into their own working style. Improved structure and process naturally grows out of constant constructive critique, since they make it quicker to present the subject rather than get tripped up by the trivial details.

Share in each other's process and ask for help! If everyone did their job, the game would never get done. The process relies on everyone covering the gaps between job descriptions and ensure that the final job gets done as well. Everyone must be flexible to change with the latest feature set of the project and compensate quickly when major problems occur.

Audio Control Parameters

Level 1	Sound tags are placed by tagging animation frames in a text file.
Level 2	Sound tags are placed directly in the animation frames by the artist. The game also derives much of its' control parameters directly from the game state.
Level 3	The game derives some of its' control from sound tags but generates most of its' control indirectly from the game state. The audio utilizes a layer between the game state and physics to normalize values and uses basic AI to derive new parameters from combinations of existing parameters. The input parameters are processed such that they trigger the content reliably in an aesthetically pleasing manner for the composer.

One of the most difficult things with interactive audio is deciding where and how to get your control parameters. One can either derive them from explicit tags inserted into the game world data or implicitly from various AI parameters and other game states. If sound tags are primarily used, then one gains more control over the nature of the input, but this requires much extra time due to repeated effort in maintaining the tags and generating enough tags to produce dynamic audio results. If game state is used to trigger the audio, then one is at the mercy of the rest of the game to produce audio-friendly input. I have personally found that this has commonly resulted in a tuning nightmare when the audio is left hanging after the game is retuned late in the project!

Composer's Vision

What is needed is to produce a separate audio layer which filters the input variables and produces parameters which play the content in the fashion the composer desires. This is one of the most important ways in which the coder lends his musical and audio knowledge to the project. The coder needs to be aware of all significant changes in game code which jeopardize the behaviour of the audio. The composer may, in addition to the coder, take an active role as well. For example, the composer might sit with the camera placement artists or camera logic coder to decide on which camera angles not only favour the graphics but also the audio.

Impact Over Authenticity

One of the most important considerations when creating game audio is to not be too attached to what is realistic instead pursue what sounds good. Commonly, the sound which 3D positional audio creates by default needs to be tweaked to make it sound better. If the doppler-shift isn't pitching enough, crank it up! In his postmortem on *Halo*, Marty O'Donnell identifies a common problem in 3D positional audio in which the speaking character's voice can sometimes annoyingly jump around in the surround mix when the camera view switches from the speaker to a listener. Tying the listener to a static location instead of the camera would definitely fix this, but requires extra logic to switch listeners and require someone to define the listener's location.

The goal is to create a layer which is tuned to produce audio which supports the composer's vision. Often, shortcuts are taken, but given enough time, the audio game code should almost be viewed as a system which is "playing" the content in the style which the composer desired.

Voice and Memory Usage

Level 1	Composer has no way of accurately knowing the audio memory map and voice utilization in the game. He updates a spreadsheet to work it out manually.
Level 2	Composer is able to output the audio memory map and current voice utilization on demand but can only guess the frequency at which samples are being used.
Level 3	Composer is provided debug output which keeps a running total of the number of times each sample is called. It creates a "Bank-for-the-Buck" rating table by combining the frequency a sample is called and its' associated memory usage. Large samples which are rarely used appear at the top of the table and small samples which are often used appear at the bottom.

Usually voice usage and memory utilization issues are a sore point between the composer and coder. The composer often has no way to accurately know how many voices are being used and the coder is bitter that the composer is wasting voices and increasing overhead on the audio driver. On some systems, DSP chips can actually become saturated when over-utilized causing them to distort unpleasantly or stutter their output. The composer needs to be aware of the limits of the materials they use to create the audio and the coder needs to present all necessary information clearly and not hinder the creative process.

Voice Usage

Some methods to reduce voice utilization are volume culling, sound sphere radius reduction, voice stealing by priority, instance capping and sub-mixing.

1) Volume Culling

Volume culling involves shutting down a voice when its volume reaches a certain threshold near zero. Done correctly, this has the desirable side effect of clearing up the mix and reducing processor overhead. Done incorrectly, it can introduce voice "popping" where it is easy to hear when voices get culled and possible thrashing when a voice hovers around the threshold and is repeatedly stopped and started due to the culling algorithm. To reduce the possibility of clicks, the voice is enveloped before it is stopped.

2) Sound Sphere Reduction

Sound sphere radius reduction should be simple given individual control over 3D sound volumes. The composer should be able to reduce various groups and individual 3D sound sources in the world building tool.

3) Voice Stealing

Voice stealing by priority requires a driver which supports this functionality and some pre-planning on the side of the composer. This works by defining good priority values for each sample and allowing the sound driver to decide how to steal a voice if none are available to play a new voice.

4) Instance Capping
Instance capping causes the sound driver to steal the voice of the oldest instance of a sample group when the maximum number of instances is reached. An example is in button sounds, where the number of instances is set such that it doesn't sound like menu sounds are being cut off as well as not allowing the user to trigger enough voices to distort the game platform's audio output.

5) Sub-Mixing
Sub-mixing is likely the easiest. If two samples are commonly used together and rarely separately then the composer can mix the two sounds together, save on memory as well as reduce the number of voices used.

6) "Bank-for-the-Buck"
Obviously, common sense must also be used when acting upon the results of the table, since a large, rarely used sample doesn't automatically mean it is unimportant and can be thrown away! However, all samples and their associated memory sizes should be periodically re-evaluated to justify their weighting. An additional offline tool might allow tweaking memory use parameters in real-time. This way, the composer can interactively change the sampling rate, sample length and other parameters to see how optimizing would affect the resulting memory map and check the mix with the results.

The coder is often better informed about voice usage and memory issues, since warnings and errors often show up for them first. The coder needs to keep the composer constantly informed on the state of voice utilization by devising techniques, such as the "Bank-for-the-Buck" table described above. Through an effective dynamic mixing system and voice culling, the coder may also be to reduce voice utilization and similarly improve the mix.

Memory Usage
Memory reduction can be done easiest by changing the length, channels and resolution of a sample. This is often an arduous auditioning process of changing values and hearing them in the game. The reason to tune audio in-game is that sounds can often sound great outside the game and not so great in the overall mix of the game and vice versa. One should optimize starting from the samples which use the largest amount of memory first.

In an effort to get more variety, asynchronous sample bank loading at run-time and memory caching might also be used. If certain sounds only play in certain contexts, then it may be possible to load them into a temporary sound memory swap space to be played and then flushed them out once they are no longer needed. Memory caching across different processor memories allows for similar functionality but reduces the latency in contrast to async loading.

Voice and Memory
With making memory and voice reduction techniques easy for the composer to audition, the resources can be used in a more optimal fashion. Hopefully the composer does not get lost in all the possibilities, but definitely making the options clear and available will increase the materials the composer has to work with.

Tools

Level 1	Composer is given a text file so they can tweak constants and coefficients for volumes, pitch bends and other parameters.
Level 2	Composer is provided a GUI tool which allows them to change variables and sample banks at runtime.
Level 3	Composer is provided a visual object-oriented language which can modify the audio parameters control structure at runtime. Similar to Reaktor, the composer is presented with a "front panel" interface layer to tweak their most common values. If the composer desires, they may delve into the "guts" of the patch to modify the control and structure of the patch itself.

Tools are always meant to make both the composer's and coder's life easier. With proper utilization of good tools, the content pipeline can be at a near optimal state. However, audio tools often leave much to be desired.

Alpha Tools

Tools are usually constructed to address a need of the composer for which the coder has a quick-fix solution. Tool development is often largely ignored in project scheduling and is commonly done during downtime at the start or end of a project. Usually the coder gets the tool so that it functionally works (ie. alpha). Most audio tools are in fact prototypes which should really be redesigned and recoded. Often there is no QA for tools, so the composer is forced to walk the fine line of asking for bug fixes but not put too much pressure on the coder such that they become annoyed. Having a tools group is a definite improvement but can threaten to distance the composer since the tools group provides tools to multiple projects and may not have time to create tool features which uniquely aid the composer's project.

Rockin' in the Real-world

One approach is to complete the tool well before the next project starts and have the coder use the last project as a real-world test data. This way the composer is at least granted that the code works on a real project (which often isn't the case). To his defense, the coder is often not given extra schedule time to maintain tools and has to work extra time to fix tools and modify them to suit the current game.

Reuse

Coders are suckers for thinking they can do everything better from scratch. The real reason is that it is much more difficult to understand someone else's code, modify it correctly and take pride and true ownership of the new system. This applies to existing formats as well. As I found with my project at Moderngroove, it is much better to use MIDI for anything to do with time-related music data rather than try to come up with one's own format. I also found that having a human-readable output from the tool is indispensable, since if worse comes to worse, the composer can still get her job done.

Tooled

Audio tools are often proprietary and game-specific which makes it difficult for the composer to learn how to use them before deliverables using them are required. Often no time is given to the composer to learn the tool since it is believed that if the tool is well designed then things should be pretty obvious anyways. Usually things are obvious to the coder and not the

composer. This combined with the fact that the last thing the coder has time for is documentation, makes audio tools very frustrating for the composer to effectively use.

Object-oriented Graphical Tools and Scripting

Starting over ten years ago with the origins of Pure Data (and MAX/MSP), a new paradigm for audio creation emerged. MAX/MSP is proprietary and much larger than Pure Data, so Pure Data is likely a simpler starting point for consideration as a video game audio tool. As well, MAX/MSP is also a Macintosh only product in contrast to Pure Data which runs on many platforms. Unfortunately, the Pure Data user-interface, level of reliability and ease of installation is definitely inferior to MAX/MSP.

Basically, Pure Data is a graphical object-oriented language meant specifically for musicians. One drags components around the screen and links their inputs and outputs via virtual patch cables. This modular approach to creating audio has also been used in software packages such as Reaktor, AudioMulch and others.

Reaktor is a great environment to quickly prototype ideas. There is a “front panel” on which the user can easily modify parameters which hide the “guts” of the virtual wiring details beneath which the user can also dive in and modify. Similarly, in Pure Data, patches are hierarchical so details can stay hidden and not confuse the user. Making patches hierarchical also helps with reuse, since sub-patches can be shared and used multiple times.

So, not only are these tools good for prototyping, but they could also be good for actual use with consoles. Although they are interpreted at run-time, they are much more intuitive to use than typing in audio scripts. Since Pure Data is open source, one approach could be to port the basic functions to your target platform to get things rolling. Similar to scripts, the patches could be modified on the host computer and transferred to the game while it game is running.

Currently, it might not be practical for consoles to run synthesis engines at run-time, but a subset of operations could be used to cover basic sound requirements. Within environments such as Pure Data, it is easy to add additional custom components. One of the drawback of these modular systems is that they do not represent highly procedural constructs easily and operations which require specific ordering of instructions might be better implemented using a text-based scripting language.

Basically, these environments could be seen as composer-friendly front-ends to modular audio scripts. If the composer is more comfortable with procedural text scripting, then perhaps this is a better option. Any optimizations and techniques which apply to scripts could also be applied here. Python and Lua are good examples of scripting languages to look into for reference.

These approaches are simply meant as possible avenues in an effort to allow the composer expanded control over their content’s playback. With open-ended tools such as Pure Data and audio scripting engines, the composer could have nearly unlimited control over all their content (or at lease as much as they would ever want to!)

Automated Dynamic Mixing

Level 1	The audio content is integrated by a temporary coder on the team and the composer is given a few tools to tune her content.
Level 2	The coder introduces systems which allow the code to dynamically change the mix levels of the audio. Similar to user presets on a digital mixing board, this primarily consists of a set of presets which the composer defines for each game area.
Level 3	The dynamic mixing system's control structure is accessible to the composer who can change it's behaviour using a scripting system or tool.

The mix is often the most misunderstood element in the video game audio development process. Often it is assumed that once content is in the game that the composer's job is done. The final mix of film or television requires a significant amount of the total budget, but in game audio, it is rarely considered.

Licensed to Mix

A common problem is the chaos often results from jamming in licensed content at the last minute. I've found that licensed content is always late. Usually licensed content is required to play almost all the time, due to its' dollar value and marketing value. In addition, it is often mastered with so little headroom that everything else must be pushed to the breaking point to compete. A similar problem can also happen when there are separate people working on speech, music and sound effects all trying to make their section heard above the din. Often the composer has spent time balancing the mix with the placeholder music, but the final music usually holds some hidden surprises. If possible, it is good to make sure the composer is contractually allowed to remaster the licensed tracks to make them fit into the mix better.

Max Headroom

All too often the composer will want a sound to play louder but finds herself out of headroom. With digital mixing, the way to make it louder is to drop the levels of all other sounds playing at the same time. However, following this, how do we make the game sound "loud" or "intense" when the game is first played? The first way to tackle this is to have the composer try to set the mix at a middle-ground such that the less intense moments of the game are still easily audible and the loudest moments still have impact. Another possible way is to is to implement a type of expander which brings up the quiet sound level such that it fills the available bandwidth more efficiently. However, this doesn't take into consideration making sounds louder which are more "important."

Auto-Mix

The end result of these approaches is automated dynamic mixing which requires significant work by the coder on the game itself and the development of supporting tools. The ultimate result would be an engine which sets the mix level such that the mix is set similar to a mixer riding the levels throughout the duration of the game.

Dyna-mix
As one can see, this is a difficult topic, bordering on artificial intelligence issues. So we may need to take a step back. Even just allowing the composer control over setting mix presets for certain game situations and modes with adjustable slew rates between them can solve many problems. This could be used to solve a problem where the speech level is too high when the game is relatively quiet, but unintelligible when the game begins to get relatively loud. The composer could set a mix level for when the game is quiet and when it is loud and the rate of change of the volume ramping between the two for intermediate levels of background loudness.

Mixing Groups
With automated dynamic mixing we can take any available parameter from inside the audio system or from the game to decide the mix levels. The most basic approach would be to use a total voice count to decide the master mix. Eventually we might want to break the sounds into groups so that they control their own group level and make sure to have enough headroom when all groups are at full. If we keep track of the relative differences between the groups, then they can all be adjusted to keep the overall mix at a good level.

Mind the Mix
The really difficult part is if we want to tune the way in which the modulation happens so that it doesn't sound mechanical and try to make is sound more "intelligent." We can use damping values to make sure the mix doesn't swing too wildly and many other tricks to shape the output. If we allow the composer to actually change the control structure with scripts, then we obviously open ourselves up to bugs. Deciding which parameters to expose is also a large issue and they should usually be normalized and otherwise cleaned up so they are easy to use.

Information Overload
The main problem with dynamic automated mixing is that it may take too much control away from the composer. The danger being that the more the mix level is set by the behaviour of the mixing code, the less the composer can explicitly define the actual mix. Control over presets and mix coefficients is helpful, but perhaps the best eventual solution is to allow the composer control over the behaviour over the mix itself, using either a scripting language or tool. The composer should be able to decide which parameters (such as number of voices, volume of voices, groups of samples and game state) control the mix level and how to modify them. The dilemma here is that the composer may cease to understand why the mix is doing what it is doing once the control structure becomes too detailed!

Future Mix
The mix is definitely one of the more difficult areas of game audio to do well which is often why video games sound so poor in the first place. With more powerful systems, we can no longer make excuses based on the platform's audio limitations. It is difficult to say if a more static system is better than a reactive system, but hopefully we will be hearing new systems which use effective ways to control the constantly changing quality of the mix of video game audio.

Closing Thoughts
We shouldn't feel limited or restricted by our official role which we play in audio development. Once we realize that we are all creators in a creative process as well as engineers in a software engineering process, we can see both sides at once of our contributions to the end product. We need to continue to break down the difference in language and knowledge base between content and code. Jump in, try something new, learn and have fun!

Leonard Paul Bio:
Leonard Paul attained his Honours degree in Computer Science at Simon Fraser University in BC, with an Extended Minor in Music. He started in the video game industry almost ten year ago and is experienced in both audio programming and music composition. He has worked on FIFA '95 (Sega Genesis), NBA Live '95 (Sega Genesis), NBA Live 2000 (N64), Moderngroove: Ministry of Sound Edition (PS2), Need For Speed: Hot Pursuit 2 (PS2), Sega Soccer Slam (PS2) and Hitz 2003 (PS2) and other titles at Electronic Arts, Radical Entertainment, moderngroove entertainment and Black Box Games. He wrote the first audio postmortem for Gamasutra as well as writing for other publications such as the ZD Developer network. He is a professional musician and composer, having worked with film, theatre and dance. In addition to his other audio work he teaches Game Audio and Interactive Music at the Art Institute in Vancouver, Canada.

References:
Olavsrud, Thor. *Video Games: Not Just for Consoles, PCs Anymore*. Infrastructure, Web: http://www.internetnews.com/infra/article.php/1491941, Oct 31, 2002.

Paul, Leonard. Moderngroove : Audio Postmortem. Gamasutra, Web: http://www.gamasutra.com/features/20011207/paul_01.htm, 2001.

Pressman, Roger S., *Software Engineering : A Practitioner's Approach*. McGraw-Hill, 1992.

Reaktor, Software, Native Instruments. Web: http://www.native-instruments.com

Rohrl, Dave. *Two Dozen Ways to Screw Up a Perfectly Good Project*. Game Developers Conference, 2002.

Max/MSP, Software, Cycling '74. Web: http://www.cycling74.com

AudioMulch, Shareware, Ross Bencina. Web: http://www.audiomulch.com

Pure Data, Open Source Software, Miller Puckette. Web: http://www.pure-data.org

Contact:
Leonard Paul – info [at] lotusaudio.com