Designing Next-Gen Academic Curricula for Game-Centric Procedural Audio and Music

Rob Hamilton[1]

[1]*Stanford University, CCRMA, Stanford, CA, USA*

Correspondence should be addressed to Rob Hamilton (rob@ccrma.stanford.edu)

ABSTRACT

The use of procedural technologies for the generation and control of real-time game music and audio systems has in recent times become both more possible and prevalent. Increased industry exploration and adoption of real-time audio engines like libPD coupled with the maturity of abstract audio languages such as FAUST are driving new interactive musical possibilities. As such a distinct need is emerging for educators to codify next-generation techniques and tools into coherent curricula in early support of future generations of sound designers and composers. This paper details a multi-tiered set of technologies and workflows appropriate for the introduction and exploration of beginner, intermediate and advanced procedural audio and music techniques. Specific systems and workflows for rapid game-audio prototyping, real-time generative audio and music systems, as well as performance optimization through low-level code generation will be discussed.

1. INTRODUCTION

Computer-based representations of space and action have over the last five decades become both increasingly sophisticated and commonplace. The computer gaming industry has bloomed into a multi-billion dollar business, offering rich graphic content and dynamic control interfaces on hardware platforms ranging from desktop and laptop computer systems to home gaming-consoles and mobile devices. The advent of ubiquitous fast network connections and an explosion in modern society's use of social media have supported the rise of popular massively multiplayer online environments - distributed server-based virtual spaces capable of engaging large numbers of clients at any given time. And while there do exist large-scale music-based social gaming networks and gaming franchises focused on interactive musical gameplay [7], the majority of gaming titles still primarily incorporate music as a background support for game narrative and mood.

Even as graphics processing used in gaming systems have become faster, more realistic and more sophisticated with the increase in power available to most computing devices, the allocation of cpu cycles to sound and music concerns in most commercial gaming software pipelines has remained relatively small by comparison. One result of such limitation is that the vast majority of sound assets used in software systems still consist of pre-recorded samples, artfully combined by composers and sound designers to create dynamic soundscapes and musical experiences.

As such, one concern for composers working with interactive music systems for games and similar environments is the avoidance of undue repetition and the integration of thematic musical material with experiences unfolding in the software. Techniques such as horizontal arrangement and re-sequencing - where game data drives the playback of subsequent thematic sections of a pre-recorded composition - or vertical arrangement and orchestration - in which the density and orchestration of musical material in the form of layered recorded materials is driven by game data - allow composers to dynamically adjust their composed materials in alignment with game progress and narrative. And through the use of dynamic digital signal processing coloring both game audio and music, a greater degree of interactivity coupled with high-quality composed material can be achieved.

While such techniques allow sample-based sound architectures to display a base level of dynamic change and interactivity, true process-based or procedural audio and music systems - wherein game parameters are tightly coupled in real-time to synthesis and/or audio manipulation processes - are still a significant exception to the industry standard audio workflows.

While recent increases in gaming hardware memory and cpu speed have removed many of the oft cited barriers prohibiting the use of procedural audio systems in commercial gaming systems, one barrier that still remains however is a distinct lack of educational curricula for sound designers and composers looking to expand their artistic and commercial practice into procedural directions.

2. PROCEDURAL SOUND AND MUSIC

Data rich systems found in computer gaming or virtual reality platforms, offer an opportunity to couple motion and action in virtual spaces to sound and music generating processes. By creating music and sound in virtual environments procedurally, that is by creating and controlling sounds through the mapping of parameters of motion, action or state to sound producing systems, the sonic experience presented to users can be tightly integrated with the visual and narrative experiences on which successful cognitive immersion in game environments is based. The same user control systems that control motion or action in space can control parameters of sound and music. By coupling user action in direct as well as abstracted fashion, rich artistic environments can be created. In this way, the virtual environment itself as well as a gamer's interaction with that environment can become both an active and reactive participant in the game's sonic experience.

2.1. Early Days of Procedural Sound

The use of procedural techniques linking game data directly to the generation of sound and music systems in games and virtual environments is not new. In the early days of computer gaming, before computer memory space was large enough to allow for the storage and playback of large pre-recorded audio files and samples, the majority of sound effects and game soundtracks were commonly synthesized in real time using Programmable Sound Generators (PSGs) driving simple synthesis processes [2].

As gaming systems became larger and more complex, the role of sound and music generation shifted significantly away from real-time data-driven synthesis and control towards pre-recorded music and sound, more in the style of a motion picture.

Andy Farnell described this paradigm shift in gaming audio and music as such:

> These computers seemed alive. There was such low latency and response from the machine, and the sounds that they made were generated synthetically. There was a tight coupling... it was like an instrument. I think sound samples came along and sound became dead. And it stayed dead for a long time. [4]

2.2. Recent Work

While many commercial video games embraced the use of sound samples and more static soundtracks, the growth of a more casual gaming practice - featuring less complex game systems often based around portable devices like the Nintendo DS gaming system or mobile devices - often showcased innovative musical game systems. Designer Toshio Iwai created Electroplankton in 2005 with a dynamic musical system that was tightly integrated into the game itself [10]. More recently, the high-profile game Spore used Pure Data to drive procedural music for its in-game character creator, mapping user action to parameters influencing the musical score in real-time [3].

With the introduction of projects such as libPD [1], a wrapper for Pure Data [16] enabling its use as an embedded audio engine, many of the difficulties limiting the use of process-driven audio and music in commercial gaming systems have decreased. Recent releases such as FractOSC[1] and SimCell [15] feature dynamic real-time music and audio systems driven by libPD. And with the massive boom of mobile applications following the launch of the Apple and Google application stores, an ever increasing number of music-based applications like Smule's Ocarina [21] and Leaf Trombone [20] - each powered by an embedded version of the ChucK audio programming language [19] - are leveraging procedural techniques as a core gameplay component.

3. EXISTING GAME AUDIO CURRICULAR DEVELOPMENT

To date there have existed significant academic and industry efforts to provide direction in the development and standardization of game-specific audio and music curricula. Perhaps most notably, in the 2011 "Game Audio

[1] http://fractgame.com

Curriculum Guideline" [9] prepared by the Education Working Group of the Interactive Audio Special Interest Group, a detailed four-year academic curriculum is proposed, stressing necessary skills for students to successfully enter the game audio workforce. Skills ranging from Sound Design, Dialog, Mixing and Composition to Interactive Audio Aesthetics and Game Music Analysis are suggested, providing a well-rounded curriculum to support industry's expectation for existing projects.

However, not only are discussions about procedural music and audio systems not prominent in the proposed guidelines, the role of Audio Programmer is considered separate from the specializations of Sound Design and Music, with the guidelines going as far to suggest:

> ...an Audio Programmer should - while having experience in audio - obtain an undergraduate degree in Computer Science first and foremost.

As technology continues to evolve at an extremely rapid pace, the use of what were once considered "advanced" techniques in this field are rapidly becoming tools that students routinely incorporate into their own sonic practices, expanding the "Jack of All Trades" mentality beyond simple audio and music creation and editing to incorporate software development and design. Today's students who have grown up surrounded by powerful computing devices are entering University settings exhibiting a comfort level with technology that simply did not exist in previous generations. For such students, the combination of artistic and technological pursuits is nothing revolutionary, just an inevitable consequence of the role technology plays in their lives.

4. THE RISE AND ROLE OF THE COMPOSER/PROGRAMMER

Rather than view musical composition and computer programming as two distinct skill sets, success in procedural music and audio concerns somewhat mandates students to possess skills from both these domains. To successfully understand, design and implement procedural sound and music systems, students must be able to conceptualize music as a non-linear system of sorts, leveraging the power of dynamic software systems to augment traditional linear audio and music workflows. Towards these goals, it becomes imperative that students of audio and music are exposed to a wide array of interactive music and audio systems and methodologies beyond the influence of static audio-visual media such as cinema.

One place in academia where the role of composer/programmer has been fostered and promoted throughout its nearly 50-year history can be found within the field of Computer Music. From Max Mathews and John Pierce's early experiments with computer-generated sound and music at Bell Labs [11], working with composers such as Jean-Claude Risset and John Chowning, there was a deep-rooted understanding that both computational and musical systems of thought were integral in order to truly understand the sonic possibilities afforded by computers. Modern computer music programs can be found within countless Music Departments at Universities around the world and routinely teach the integration of composition and sound design with interactive visual systems.

5. A MULTI-TIERED PROCEDURAL CURRICULUM

The design and development of procedural audio systems are by nature complex. For sound designers and composers, accustomed to working with audio and music tools within relatively straightforward asset creation production workflows, the dynamic and volatile nature of procedural systems can present an overwhelming challenge. The tight coupling between game and audio engines necessary to procedurally bind action to audible event requires practitioners at some level to be capable software designers and audio engineers, in addition to traditional sound design and music composition skill-sets. And while industry development teams can allocate specialists to each of these roles, students or individuals working outside of a fully-funded professional environment will likely need to bootstrap their own projects, simultaneously playing multiple roles across multiple specialties.

When introducing procedural audio systems and their integration into game engines as part of an academic curriculum, a multi-tiered approach in which the need for students to engage game-engine code is introduced gradually can offer students the experience of composing and designing for dynamic systems before necessitating any hands-on programming skills. Within the context of a Computer Music or Interactive Media program, such a combination of software-based musical and visual media is already becoming relatively common.

For the purposes of this paper a three-tiered approach is defined, made up of the following tiers:

- Rapid prototyping with Open Sound Control (Beginner)
- Embedded audio systems with libPD (Intermediate)
- Generating low-level audio code (Advanced)

The amount of student interaction with actual game-engine code can itself be presented in a multi-tiered fashion, introducing procedural systems with no game-engine coding first, and gradually increasing student exposure to game coding paradigms. This technique and the tools described below were explored during Stanford University courses such as *Compositional Algorithms, Psychoacoustics, and Spatial Processing* and *Computer Music Improvisation and Algorithmic Performance* (2013), as well as during the 2014 *Designing Musical Games::Gaming Musical Design* CCRMA Summer workshop, an intensive weeklong seminar in designing and building procedural game environments held at Stanford's Center for Computer Research in Music and Acoustics[2].

6. TIER 1: RAPID PROTOTYPING WITH OPEN-SOUND CONTROL

One technique for quickly prototyping procedural game audio interactions makes use of the Open Sound Control (OSC) protocol [22] to pass game-state parameters in real-time to powerful interactive audio systems written in computer-music programming languages such as Supercollider, ChucK, Max/MSP or Pure Data. By compiling OSC libraries into a game's codebase and hooking individual engine events, control systems or game-state data, a game's parameter stream can be mapped to control complex audio systems. Students are still required to build sound engines in these higher-level audio programming languages, however as these languages were designed to be used primarily by musicians, the learning curve and overall complexity is significantly lower than programming similar systems in low-level languages.

Using OSC and a music-based programming environment, student composers and sound designers can focus their energy on crafting musical systems and interactions, and not on the integration of low-level audio code. To facilitate this rapid-prototyping approach, a series of Open-Sound Control enabled game environments have been developed, allowing students to explore familiar game environments such as Minecraft, and Quake III, with more flexible implementations designed to integrate with the Unreal and Unity game development platforms.

6.1. Osccraft

Released in the Summer of 2014, Osccraft is a modification to the popular game Minecraft that embeds oscP5, a bi-directional Java OSC library, into the Minecraft Forge mod framework. Osccraft outputs real-time OSC messages passing parameters of player motion (3D translation and rotation), block events (creation, destruction and hit interactions), as well as tracking AI-controlled mobs and World events such as time (across a day-night cycle). Osccraft can also be run with multiple networked clients, allowing for the creation of complex multiuser interaction schemata.

As the core game-mechanic of Minecraft is the creation and manipulation of blocks, users can rapidly build and modify game environments on the fly with no additional code or software use other than the game itself. As part of a procedural sound and music curriculum, students can use Osccraft to build and interact with game topographies, structures and entities on-the-fly, testing sound treatments and processing techniques via OSC without exiting the game environment.

The Osccraft mod code is freely available and open-sourced both as a re-obfuscated and compiled .jar file and as source code. The project repository can be found on github at https://github.com/robertkhamilton/osccraft

6.2. q3osc

q3osc [8] is a heavily modified version of the open-sourced ioquake3 gaming engine featuring an implementation of Open Sound Control for bi-directional communication between a game server and one or more external audio servers. By combining ioquake3's internal physics engine and robust multiplayer network code with a full-featured OSC packet manipulation library, the virtual actions and motions of game clients and previously one-dimensional in-game weapon projectiles can be repurposed as independent and behavior-driven OSC emitting sound-objects for real-time networked performance and spatialization within a multi-channel audio environment.

[2] https://ccrma.stanford.edu/workshops/designingmusicalgames2014

Fig. 2: A multi-player chromatic marimba built within *Osccraft* for the 2014 "Designing Musical Games::Gaming Musical Design" Workshop at Stanford University's CCRMA.

6.3. Unity3D

For students with some background in software development or game art and animation, a more hands-on approach can be pursued in which game scripts and art assets can be manipulated at a high level. The Unity3D game programming environment can be easily extended to embed OSC libraries using a variety of programming languages, including Javascript, C# or Boo. A number of fully-featured OSC libraries with Unity-specific wrapper classes are freely available including OSCSharp[3] and UnityOSC[4].

To introduce programming concepts without overwhelming relatively inexperienced students, a simple 2D sidescroller game environment can be easily designed in Unity with key game interactions hooked to OSC-generating events. For the Designing Musical Games workshop, a simple game titled "World of Mush" was prototyped and distributed to students, with basic OSC hooks for player position and trigger zones including mushroom-shaped jump-pads already included. Students were encouraged to add new OSC hooks by reproducing simple code elements included in the project, exploring new creative game interactions and building dynamic sound processes using the Pure Data programming environment.

[3] https://github.com/valyard/OSCsharp
[4] https://github.com/jorgegarcia/UnityOSC

6.4. UDKOSC

For students more comfortable with software development and programming, full 3D game engines such as the Unreal Development Kit (UDK) [17] or the more recently released Unreal Engine 4 provide next-gen graphics capabilities and robust game architectures. Introduced in 2011, UDKOSC is a modification to the UDK featuring OSC input and output streams for the controlling, tracking and sonification of in-game actions and motions [6]. UDKOSC was designed to support the creation of immersive mixed-reality musical performance spaces as well as to serve as a rapid prototyping workflow tool for procedural/adaptive game audio professionals [14, 18]. Data points tracked in UDKOSC include actor, projectile and static mesh coordinate positions, game events such as collision, and real-time ray tracing from player actors to identify interaction with specific object types and classes.

Fig. 2: Parameters of avatar skeletal-mesh bone translation and rotation for a bird-like avatar are tracked with *UDKOSC* in the mixed-reality musical performance work *ECHO::Canyon*

UDKOSC is a set of UDK custom class files, which when compiled, take the form of a custom game-type within the UDK. Gestures, motions and actions generated by actors in game-space are hooked and output as OSC messages to be transformed in real-time into control messages for complex audio and musical software systems. While the UDK restricts access to Unreal's core C++ codebase, it exposes UnrealScript as a higher-level scripting language. UnrealScript allows developers to bind Windows Win32 .dll's to UnrealScript classes, enabling external blocks of code to interact with the scripting layer. Using this DllBind functionality, UDKOSC binds a customized version of Oscpack to a series of UnrealScript classes and mirrored data structures, passing bidirectional data both into and out from the game engine. Real-time game data can be streamed over UDP to any given IP-address and port combination and control messages from external processes can be streamed into the game engine.

OSC input can be used to drive actor velocity and rotation in three-dimensions, and can be targeted towards numerous entities individually or simultaneously through the use of OSC bundles. Input data for the control of game pawn, player or camera actors is designed to offer detail and flexibility in the manner in which actors are moved and controlled within the game engine. In this way, detailed choreographed positioning and action can be carried out, such as the flocking actions of flying actors or intricate camera motion and cut scenes.

The UDK is available at no cost for educational and non-commercial use and until early 2014 was updated monthly with new features ranging from enhancements to the lighting and modeling tools to integration with Apple's iOS, allowing works created in the UDK to be published to mobile devices such as the iPhone and iPad.

7. TIER 2: EMBEDDED PROCEDURAL AUDIO ENGINES

While the use of Open Sound Control alongside external interactive music programming languages allows students to quickly experiment with game parameter mappings and novel compositional and sound design techniques, this approach becomes problematic when the goal of a given game audio project is the release of a commercially "shippable" game product. To build game audio and music systems that can be released as commercial game products to end users, the audio generating codebase needs to be compiled into one distinct project. However rather than leap from flexible computer-music style programming languages directly into sound and music systems based around low-level audio programming, there exists a middle-ground wherein a fully featured dynamic audio programming language can be embedded within a host application, communicating with the game engine using hooks similar to the Open Sound Control implementations explored in our first tier approaches. While the use of languages like ChucK has been successfully explored within mobile musical gaming applications, at this time, Pure Data and the libPD project offer perhaps the most mature and stable engine of this type.

7.1. libPD

libPD is an embeddable library that wraps Miller Puckette's Pure Data audio programming language and allows it to be compiled into interactive media-rich environments. The libPD project has been ported to a number of platforms including iOS and Android and allows sound designers and composers to build dynamic sound projects using Pure Data's graphical patching paradigms. The Pure Data user and developer community are both large and robust, making the Pure Data language a strong choice for students looking to learn more about interactive music systems.

Binding libPD into a Unity3D or iOS project is relatively straightforward and painless thanks to the project's predefined wrapper classes for CSharp and iOS. During the Designing Musical Games CCRMA workshop, participants with no prior libPD or Unity experience were able to build a Unity Scene with libPD hooks into player motion and rotation attributes, compile that project for iOS and subsequently push the Unity project with an embedded libPD engine onto an iOS device.

8. TIER 3: PROCESS OPTIMIZATION AND LOW-LEVEL CODE GENERATION

One major concern when working with dynamic audio environments like PD in commercial projects is the dynamic nature of the system itself and the potential for performance issues. The same inherent flexibility that makes PD a great choice for prototyping and development makes it less optimal for projects with fixed non-changing audio demands. For advanced students building fast and stable audio and music projects, the need to use highly optimized code with small memory and cpu footprints may likely lead them away from a project like libPD and towards functional specification and code generation workflows using projects such as FAUST and HEAVY.

8.1. FAUST

FAUST, or Functional Audio Streaming, is a functional specification language developed at GRAME that allows sound designers to define synthesis and signal processors at a high level, prototype and test those processes as compiled objects for dynamic audio programming languages like PD and Supercollider, and then subsequently compile them down to highly optimized C++ code for use directly in a game-engine [5]. FAUST features both a desktop based development environment as well as an online FAUST compiler, capable of rendering FAUST objects in a variety of library and plugin forms [12].

8.2. HEAVY

Another option to generate efficient and optimized compiled code objects from a higher-level language specification can be found in Martin Roth and Joe White's HEAVY project[5] (formerly known as Tannhäuser PD)[6]. HEAVY analyzes Pure Data patches and generates efficient compiled software objects or C++ code. HEAVY sprang out of Roth's previous work at RJDJ, a mobile music application company that used highly optimized versions of Pure Data and libPD in their shipping musical products. At the time of this writing, HEAVY is still in a closed Beta state.

9. CONCLUSIONS

There exists a growing need in commercial game development for flexible and dynamic audio systems to complement the highly procedural and generative visual aspects of modern gameplay. As demand grows for sound designers and composers comfortable working in dynamic and procedural sound and music paradigms, so too will demand grow for educational curricula designed to introduce and refine the techniques and workflows necessary to build these intricate and subtle systems. This paper has presented a multi-tiered approach for the teaching of procedural sound and music systems. By presenting students with a series of gaming platforms and dynamic sound and music generation techniques at different steps along their educational path, students can quickly begin experimenting with the artistic and creative aspects of building procedural sound systems without being intimidated or slowed down by the intricacies of software development. And as their skill set and comfort level with procedural and programming systems grows, the projects and workflows outlined in this paper can provide a steady stream of challenges and attainable goals.

10. REFERENCES

[1] Brinkmann, P. and McCormick, C. and Kirn, P. and Roth, M. and Lawler, R. and Steiner, H.-C., Embedding pure data with libpd, Proceedings of the Fourth International Pure Data Convention, pp. 291–301, June, 2011.

[5] http://heavy.enzienaudio.com/

[6] http://tannhauserpd.section6.ch

[2] Collins, K., FromBitstoHits:Video Games Music Changes its Tune, Film International, vol. 12, pp. 4–19, January, 2012.

[3] Danks, M., PD in video game Spore; forum communication, http://lists.puredata.info/pipermail/pd-list/2007-11/056307.html

[4] Farnell, A., Keynote Lecture, Satellite Workshop at the 14th International Conference on Digital Audio Effects (DAFx), Paris, France, DAFx/Ircam, Paris, France, 2011, http://vimeo.com/35254559.

[5] Y. Orlarey, D. Fober, and S. Letz, An algebra for block diagram languages, in Proceedings of International Computer Music Conference, pp. 542–547, 2002.

[6] Hamilton, R., UDKOSC: An immersive musical environment, in Proceedings of the International Computer Music Conference, Huddersfield, UK, pp. 717–720, August, 2011.

[7] Hamilton, R. and Smith, J. and Wang, G., Social Composition: Musical Data Systems for Expressive Mobile Music, Leonardo Music Journal, volume 21, 2011.

[8] Hamilton, R., q3osc: or How I Learned to Stop Worrying and Love the Game, In Proceedings of the International Computer Music Association Conference, Belfast, Ireland, 2008.

[9] Education Working Group of the Interactive Audio Special Interest Group, Game Audio Curriculum Guideline v.10, Interactive Audio Special Interest Group (IASIG), March 2011.

[10] Iwai, T., Electroplankton User Manual, Nintendo of America, Inc., 2005.

[11] Mathews, M. V., The Digital Computer as a Musical Instrument. Science, New Series, Vol. 142, No. 3592 (Nov. 1, 1963), pp. 553-557.

[12] R. Michon, Y. Orlarey, The Faust Online Compiler: a Web-Based IDE for the Faust Programming Language.

[13] Onen, U. and Stevens, R. and Collins, K., Designing an International Curriculum Guideline for Game Audio: Problems and Solutions, Journal of Game Design, Development and Education, pp. 38 – 47, 2011.

[14] Paul, L. J., Video Game Audio Prototyping with Half-Life 2, in Transdisciplinary Digital Art, Sound, Vision and the New Screen, Adams, R. and Gibson, S. and Arisona, S. editors, Communications in Computer and Information Science, vol. 7, pp. 187–198, Springer Berlin Heidelberg, 2008.

[15] Paul, L. J., The Procedural Sound Design of Sim Cell, presented at the AES 137th convention, AES 2014, Los Angeles, CA, 2014.

[16] Puckette, M., Pure Data, in Proceedings, International Computer Music Conference, San Francisco, pp. 269–272, 1996.

[17] Epic Games, Unreal Development Kit (UDK), https://www.unrealengine.com/products/udk

[18] Verron, C. and Drettakis, G., Procedural audio modeling for particle-based environmental effects, in Proceedings of the 133rd AES Convention, Audio Engineering Society, San Francisco, 2012.

[19] Wang, G. The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality. PhD Thesis, Princeton University, 2008.

[20] Wang, G. and Oh, J. and Salazar, S. and Hamilton, R., World Stage: A Crowdsourcing Paradigm for Social / Mobile Music, In Proceedings of the International Computer Music Conference.. Huddersfield, UK, 2011.

[21] Wang, G., Ocarina: Designing the iPhone's Magic Flute, Computer Music Journal. 38(2):8-21, 2014.

[22] Wright, M., Open Sound Control: an enabling technology for musical networking, Organised Sound, vol. 10, pp. 193–200, 2005.