ApProgXimate Audio: A Distributed Interactive Experiment in Sound Art and Live Coding

Chris Kiefer

Department of Music & Sussex Humanities Lab, University of Sussex, Brighton, UK.

School of Media, Film and Music,
Silverstone Building,
Arts Road,
University of Sussex,
Falmer,
East Sussex,
UK
BN1 9RG

c.kiefer@sussex.ac.uk

01273 877484

Chris Kiefer is a Lecturer in the School of Media, Film and Music, working in Digital Technologies and Digital Performance at the Sussex Humanities Lab. His research background is in music, computing, machine learning and interaction design, stemming from his PhD studies in musician-computer interaction at the University of Sussex, and research projects at Goldsmiths and Brighton University in audiovisual interactivity, brain-computer interfacing, pervasive sensing and musical instrument design with children with disabilities. Chris is a practising musician, performing improvised electronic music with code and custom-made instruments under the name *Luuma*. His research projects focus on networked notation for musical ensembles, the design of multiparametric instruments, uncontrol, participatory design, participatory research methodology and gesture analysis.

ApProgXimate Audio: A Distributed Interactive Experiment in Sound Art and Live Coding

ApProgXimate Audio is a browser-based sound art experiment, a distributed and participatory piece where code and livecoding are fundamental to the participants' experience. The piece is based on a coding technique called Approximate Programming. It allows participants to explore complex sound synthesis algorithms through relatively simple techniques, by writing and editing code and through screen-based mouse interaction. It foregrounds code as the primary medium of engagement. This article explains the technical background behind the piece, and then explores the design philosophy through three themes that emerged during its creation: accessibility and paths to engagement in work based on live coding, the blurred boundaries between tools, instruments and artworks, and finally issues surround the web-browser as a venue for creative work

Keywords: live coding, genetic programming, internet, sound art, participatory

Introduction

ApProgXimate Audio is a web-based distributed and collaborative artwork that allows participants to explore, create and share sounds. Participants engage with the piece through the medium of code, within the framework of a programming technique called *Approximate Programming*. (Kiefer, 2015). The piece is the most recent in a series of artistic experiments in applications of this technique. You can visit *ApProgXimate Audio* at http://approgx.luuma.net, using the *Chrome* browser.

Approximate Programming (which will be described in more detail below) is a system for designing algorithms by combining small pieces of code into a larger algorithm, with a process that is controlled a vector of numbers. This vector of numbers can be manipulated by a variety of processes; it can, for example, be hand selected, changed with a screen-based interface, or controlled by a digital musical instrument. It

allows the user to create a space of creative possibilities, and then explore this by manipulating numbers as well as manipulating code. The term *approximate* is used because of the bottom-up and emergent nature of this workflow and because of the highly nonlinear and complex relationship between inputs and outputs; it's challenging to start with a blank slate and precisely specify the code and vectors to achieve a desired output. Instead, the user moves towards a desired result by shaping a space of possible outputs through exploration and refinement of inputs. These aspects of the system can be deemed *approximate*, in that the process of creating a desired output could not be carried out in a precise or highly specified manner, but the system does have enough predictability to roughly map out an area to explore and then refine this area. The system is, however, very predictable in the way that the mappings function; they are deterministic, so a particular set of inputs will always give the same output.

The system runs in realtime; changing code has an immediate effect on the output, so this is considered to be a form of live coding system. The author has used this system previously for live musical performance, generative audiovisual display, and for sound design and composition. Recently, technologies have emerged that can facilitate the creation of an Approximate Programming system to create sound, based solely within a web browser. WebAudio (W3C, 2015) supports sound synthesis in the browser, with either pre-defined unit generators or customised signal processing written in Javascript. Since its introduction, it has been used for a wide range of audio applications, a good reference is the WAC conference series (IRCAM, 2015). The potential of moving the Approximate Programming system from a live performance setting to the web where it could be used in a distributed and also participatory way seemed novel to explore. This article describes how Approximate Programming was used to create an experimental web-based sound-art piece. The piece is inspired by Mr

Doob's *GLSL Sandbox* (Doob, 2015) and Charlie Robert's *Gibber* (Roberts et al, 2015), both of which are browser-based creative live coding systems. This article begins by describing how the piece functions. It then proceeds to discuss the philosophy and motivations behind the design, and the themes and issues that have emerged during its creation, namely issues of engagement with code, distributed and collaborative art, and the boundaries between creative work, tools and instruments.

<INSERT FIGURE "codeScreenShot" ABOUT HERE>

Approximate Programming

Approximate Programming is an experimental coding technique, aimed at allowing the programmer to interact with code not just through the medium of text, but via any process that produces continuous numerical output, be it another algorithm, a dataset or a digital musical instrument. The technique has roots in the field of gene expression programming (Ferreira, 2001). Gene expression programming describes a process that converts a vector of numbers (a gene) into an algorithm represented in tree form, given a predetermined set of functions mapped to signifiers in the gene. These genes are evolved using artificial evolution techniques to create an algorithm that fulfils a specific set of goals. Approximate programming inherits the gene expression programming techniques of translating a gene into a code tree, however there is no specific goal other than to explore a space of creative possibilities, so rather than using evolutionary processes, an interactive evolution approach (Dorin, 2001) is followed. The gene can be directly manipulated by the user, while they monitor the output in realtime. Overall, one could think of it as a technique combining live coding, gene expression programming,

and interactive evolution.

Approximate Programming takes two inputs. The first is the gene, as described above. The second is a set of *component functions.* These are typically small functions that specify some sort of low-level behaviour or low-level mathematical process. These functions are built into a more complex algorithm, according to the numbers in the gene. The larger the gene, the more complex and detailed the resulting algorithm is likely to be. The output of this algorithm will depend on what the component functions do, and on the structure of the algorithm given the numbers in the gene. The full details of the transformation process are given in (Kiefer, 2015).

In use, this technique challenges the user to explore a non-linear space of possibilities. The predominant mode of use is exploratory; it's rarely possible to translate directly or intuitively from a set of component functions to a desired outcome. However, through a process of exploration and fine-tuning of both the gene and component functions, a player can map out the parameter space and shape the process to approach a desired result. It provides an alternative to the conventional top-down processes employed in algorithm design, and exploits the assumption that in creative contexts, algorithm design may not need to be as precise as is needed for engineering applications.

The algorithms produced by Approximate Programming could be employed in a range of contexts; to date this technique has been used to produce sound and to produce video. The sound generating example was a live performance system, running in SuperCollider (McCartney, 2002). The component functions were live-coded in the SuperCollider IDE, and the system was controlled with a multiparametric controller. This controller consisted of a set of 8 motion sensors mounted on armature wire, streaming a vector of 48 variables. This vector was used as the input to the system, and

converted into code and then sound. Moving the motion sensors would change the gene and therefore the sound. The video example was an autonomous audiovisualiser used in live performance. Blocks of audio were taken repeatedly from a live audio stream and used as the gene, which was converted into GLSL code and run as a GPU shader (Rost et. Al, 2009). MFCCs were analysed in the audio stream, and used to control parameters in the shader. Both of these systems were capable of generating some very interesting outputs, with complex, organic qualities that evolved over time. They also suffered from issues due to nonlinearities in the parameter spaces sometimes resulting in disproportionate shifts in the outputs relative to the size of changes in the input. The piece described in this article is, in part, an attempt to alleviate this issue by foregrounding the complex nature of the system and giving the participant detailed control. The piece sits in the audio domain. It allows participants to listen to, interact with, code and share 'scenes', which comprise the code and gene necessary to produce a particular sound. The system has a tendency towards producing complex sounds that evolve over long periods. There's an element of risk and complexity in its use; due to possible nonlinearities in the mappings, there are settings could result in little sound or silence. The degree of nonlinearity will depend on the component functions that are being used. Some combinations of component functions will always provide a *safe* space to explore, with no risk of silence, while other component function sets may be highly nonlinear. The user can however move away from these *unsafe* zones by returning to previous settings; the mapping system is deterministic so the user will never get stuck in a silent zone.

Live Coding and Accessibility

A strong theme that emerged during the development of this piece was accessibility of participants to code and coding processes. In a piece where interaction

with code is a fundamental part of its experience, how can one make this accessible to participants with a range of experience in coding (from none to expert)? This echoes an on-going debate in live coding performance concerning the relationship between the audience and code; does it matter that the audience have different levels of understanding? And does it matter whether code is foregrounded to the audience? In *ApProgXimate Audio*, an attempt is made towards addressing these issues by providing multiple points of entry into the piece. Broadly, there are two 'modes' available in the interface. In 'listen mode' the participant can choose a scene and listen to the sound. They are presented with a set of moveable bars that allow them to explore manipulations of the sound. In this mode, the participant cannot see any code. However, switching to 'create mode' reveals the full inner mechanism for creation of a sound. In this mode, the participant can engage with the piece in levels of increasing difficulty. At all times, the full algorithm that produces the sound can be seen, and changes are shown in realtime. The participant is presented with the full gene, whose values can be moved with the mouse to change the algorithm and sound. They are also given full control of the component functions. These can be switched on and off to change the set of functions being combined into an algorithm. Participants can also edit these functions and add their own using the provided sound synthesis library and the full scope of Javascript. In this way, the participant has multiple entry points into the piece, ranging from simply listening to mouse-based exploration to coding.

<INSERT FIGURE "listenScreenShot" ABOUT HERE>

Tools, Instruments and Interactive Artworks

Following from this strategy of multiple entry levels, questions emerge about the nature of this system and how it exists as a sound art piece. From different perspectives, it could be conceived of as a generative sound player, an instrument or an (highly constrained) integrated development environment for live coding. Live coding practice has already blurred these boundaries with the use of everyday text editing tools as instruments, by incorporating software engineering into performance, and by exposing an instrument's internal workings not only to demonstrate a performer's thought process but to reveal technical process. By presenting code and coding tools as the principle medium for engagement to all participants, this piece further blurs these boundaries.

The Web Browser as a Venue

From its inception, the Internet has been used as a medium for distributing art in various forms, and has progressively been used as a medium for interactive pieces. With developing technologies, the level and granularity of interactivity available in browsers has increased to accommodate works that use complex realtime audio and video. With *ApProgXimate Audio* it was compelling to explore the browser as a venue for an interactive sound and live coding experiment as the promise of creating software that is likely to work instantly without installation on many computers lends the possibility of broad accessibility and long-term availability. In contrast, there's a large potential variation in consistency of experience when a piece can be accessed on a variety of devices with varying levels of computational power, different qualities of audio and visual display, that sit within different surroundings.

The distributed nature of this piece means it also lends itself to participant-created content. The software allows scenes and sounds to be publically shared. This feature was included both to increase its facility as a creative tool, and to provide

another way of engaging with the piece, enabling interaction not just directly on the website, but also through sharing sounds elsewhere on the web.

Summary

ApProgXimate Audio attempts to merge live coding with interactive sound art, in a distributed web-based form. In doing so, it exposes several questions about the methodology and design philosophy behind this kind of work. What is the significance of boundaries between tool, instrument and interactive artwork? How should one approach the browser as a venue for realtime interactive multimedia experiences? How can the use of code as a medium avoid precluding accessibility to non-coding participants? The system is the most recent in a series of artistic experiments using Approximate Programming. From the author's perspective, making this piece has been a valuable step in the development of this technique. Previous experiments revealed some issues concerning nonlinearities in the mappings, however by providing a dedicated interactive environment, these issues can be foregrounded and to an extent alleviated by allowing the user to engage with the process at a very detailed level.

The code that forms this piece with is open-source, and available at https://github.com/chriskiefer/ApProgXimate.

Bibliography

Doob, Mr. 2015. "GLSL Sandbox". Accessed January 2015. http://glslsandbox.com/

Dorin, Alan. 2001. "Aesthetic Fitness and Artificial Evolution for the Selection of Imagery from the Mythical Infinite Library." *Advances in Artificial Life*, 659–68. Springer.

Ferreira, Cândida. 2001. “Gene Expression Programming: A New Adaptive Algorithm for Solving Problems.” *Complex Systems* 13 (2): 87–129.

IRCAM, 2015. Web Audio Conference. *http://wac.ircam.fr/*

Kiefer, Chris. 2015. “Approximate Programming: Coding Through Gesture and Numerical Processes”. *International Conference on Live Coding*, Leeds.

McCartney, 2002. “Rethinking the Computer Music Language: SuperCollider”. *Computer Music Journal 26, no. 4: 61-68.*

Roberts, Charles, Graham Wakefield, Matthew Wright, and JoAnn Kuchera-Morin. 2015. "Designing Musical Instruments for the Browser." *Computer Music Journal* 39, no. 1: 27-40.

Rost, R.J., Licea-Kane, B., Ginsburg, D., Kessenich, J.M., Lichtenbelt, B., Malan, H. and Weiblen, M., 2009. “OpenGL shading language”. *Pearson Education.*

W3C, 2015. “Web Audio API”. *https://www.w3.org/TR/webaudio/*