Mick Grierson - Creative Coding for Audiovisual Art: The CodeCircle Platform

Introduction

Audiovisual Art is a popular form of global expression. However, it is a discipline that occupies a liminal space between Art, Science, Music and Film. As such it can be difficult to define and challenging to master. Its primary roots are in experimental cinema, but also electroacoustic music and related practices (Garro, D 2012, Brougher, K and Mattis, O 2005). It is in many ways a technical art[1], most often requiring low-level knowledge of computer graphics and sound synthesis methods. Pioneers in the field such as John Whitney Sr, Larry Cuba and Vibeke Sorensen are considered key figures in computer graphics history (Sito, T 2013, Sitney PA, 2002), whilst also being recognised as artists and composers (Moritz, W 1986). Furthermore, a number of those studying electronic music consider themselves creators of Audiovisual Art, and it is increasing in popularity in the academy.

Although Audiovisual Art's relationship to coding is less explicit, it is clearly an historically important element of the practice. A similar relationship persists in other fields, for example, Live Coding (McLean, A 2004) and 'Creative Coding' (Lopes, D 2009), and it is likewise challenging to define the boundaries of these forms of art making. Creative Computing is becoming a more widely accepted academic discipline, and this in turn occupies similar territory. Despite these difficulties in definitions, there is increased interest in Creative Technology practices, and a thirst for technical skills that facilitate their creation.

Developing a practice in Audiovisual Art and similar disciplines can be challenging due to the requirements of learning the necessary technical skills. There are a number of reasons why these challenges are important to address. It can be argued that these forms of art are important to the continued development of a wide range of technologies and media. For example, in the case of Whitney, Sorensen and Cuba, they were amongst the first to use computers to make Audiovisual Art. Their contributions also incorporate the first computer generated title sequence (*Vertigo*, 1957, title graphics by Whitney), the first computer generated 3D graphics used in a movie (*Star Wars*, 1977, 3D Death Star / flight control graphics by Cuba) and the inspirational paradigm for the 3D computer graphics software, Maya.[2] This interaction between research and practice continues to be a feature of the discipline. As an example, our work[3] on the use of generative adversarial networks for the creation of artworks has led to an exhibition in the Whitney Museum of America Art (Broad, T and Grierson, M 2017), and is considered state of the art in technical terms.

More importantly, there are a great many people who would wish to develop these skills and apply them in their own work. Massive Open Online Courses (MOOCs) on the topic have reached very high numbers of participants (Grierson et al, 2013)[4]. However, these more global, distributed methods of learning reveal the need for better, more interactive coding tools. In order to more successfully deliver learning that can effectively develop and enhance the experience and knowledge of the creative practitioner, it is essential to provide better methods for understanding and developing knowledge of the underlying techniques and practices of the form. Improving accessibility to and functionality of creative computing platforms is a potentially valuable approach that can be used to tackle this problem. This realisation has led to the creation of a new approach to making Audiovisual Art through Creative Code, which we call 'CodeCircle' (https://www.codecircle.com).

[1] Although not exclusively
[2] Sorensen's *Maya* (1993) was funded by the National Science Foundation, led to improvements in stereoscopic 3D graphics approaches, and inspired the widely used software of the same name.
[3] By which I mean work conducted in the Goldsmiths Embodied Audiovisual Interaction Group (EAVI), specifically Terence Broad and myself in this case.
[4] Over 150,000 users have taken our "Creative Programming" MOOC – it was the first MOOC from an English University, and the first in the world on the topic of Creative Code.

CodeCircle represents an attempt to find better methods for supporting the learning and use of computer programming for interactive sound and graphics. It might be more appropriate to call it *Code Circle V2*[5], as it is the second generation of the tool, reflecting the longstanding and widely held philosophy of Creative Computing at Goldsmiths. It is a project that places embodied audiovisual interaction and creative coding at its centre. The platform fuses a number of approaches to writing software that are borrowed from the history of computing, including the notion of *Interactive Programming* (explored later in this chapter). Further, it introduces the idea of *Collaborative Code*, where creators can work together at the same time and on the same documents, which can be considered an interesting method for learning[6].

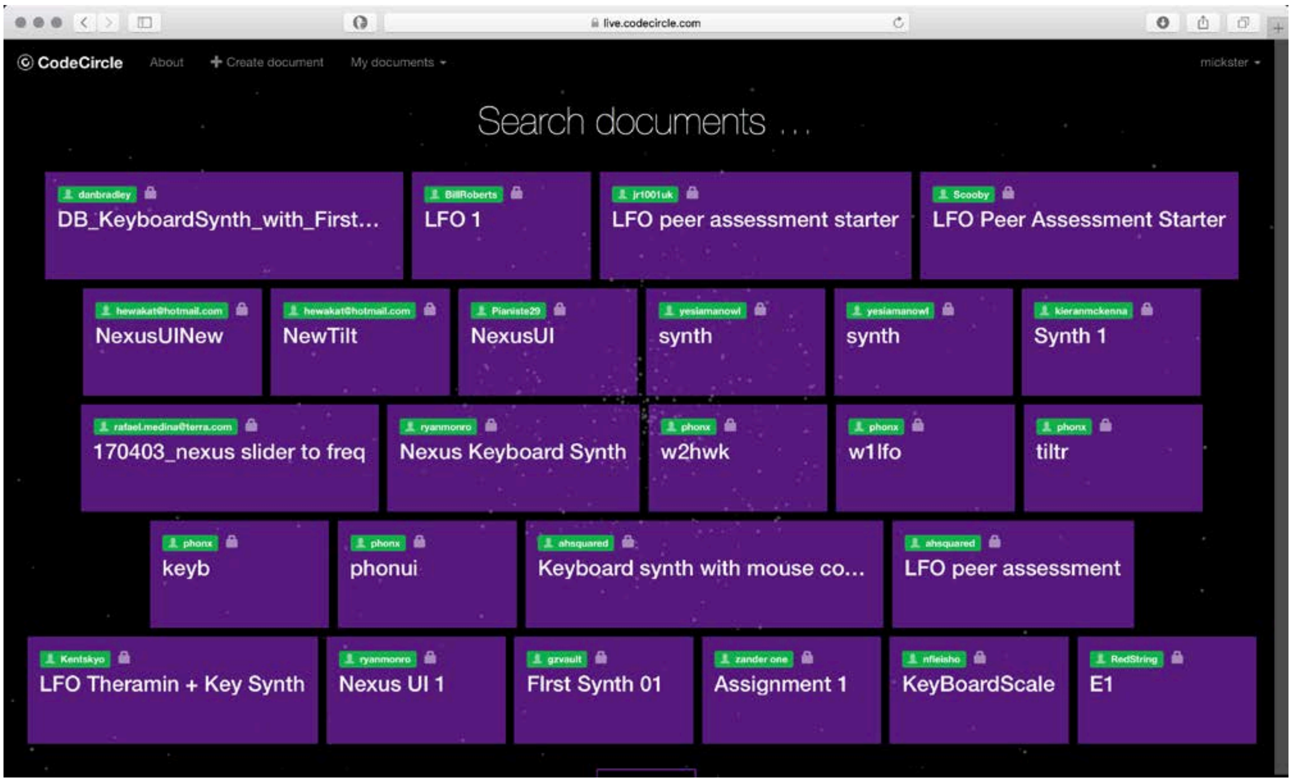


Figure 1: The Code Circle front page

CodeCircle is an online, web-based programming tool created by Goldsmiths Computing. The tool is designed to be specifically tailored to the creation of practical work in the field. This includes Computer Music, Computer Graphics, Digital Signal Processing, real time interaction, Interactive Machine Learning, Games Development and Design. All such practices share a historical link to the field of digital Audiovisual Art (see below).

CodeCircle consists of a browser-based HTML5 Integrated Development Environment (IDE) with bug detection, real-time rendering and Social features. Although many such platforms exist, CodeCircle uniquely fuses Interactive Programming with Collaborative Coding, providing Just in Time (JIT) compilation (where available) alongside real-time, socially oriented document editing in the browser. Users can work together on software that features accelerated computer graphics, buffer-level audio, signal processing, real-time user interfaces, and any other HTML5 / CSS3 / JavaScript compatible features they wish to use. Crucially, updates to the program code are simultaneously pushed to all connected users and then immediately rendered, enabling a novel

[5] It is important here to acknowledge Prof. Mark d'Inverno and Dr Matthew Yee-King, whose European Commission-funded Framework Programme 7 project, PRAISE, led to the creation of the first CodeCircle. Version 2 was a complete redesign as part of research in the Embodied Audiovisual Interaction group (EAVI) at Goldsmiths, undertaken by Matthew Yee-King, Jakub Fiala, and myself.

[6] It is our understanding that our new version of CodeCircle represents the first time this method has been used for pedagogical purposes in this field.

form of instant, interactive collaboration that is not available in any other platform to our knowledge at the time of writing.

What follows is an attempt to properly define the requirements for CodeCircle based on informed, pedagogical and creative practice needs. This is done through a brief definition of Audiovisual Art methods in the context of Creative Computing, further contextualising its position as a domain of enquiry that depends on and informs technological innovation in sound, graphics and interaction. We also briefly explore and delineate Creative, Interactive, and Collaborative coding, including key associated benefits and problems, before describing how these features form part of the design of the CodeCircle platform, and providing examples of its use and evaluation.

Audiovisual Art: A Brief Definition

Audiovisual Art is a modern artistic discipline and a global phenomenon, with leading artists including Ryoji Ikeda, Ryoichi Kurokawa, Paul Prudence, and many others who, although less well known, produce work of very high quality. The ease by which artists can distribute their work online has not diluted the field - rather it has demonstrated its potential to create new audiences, even in its most abstract, experimental forms. As mentioned above, the technical difficulties associated with its production, particularly in real time interaction contexts, have naturally led to its association with technological and scientific progress. As such it is closely aligned to other disciplines that fall within the broader academic field of Creative Computing.

Despite its popularity and success, Audiovisual Art is a discipline that is misunderstood and poorly characterised. Academics from related disciplines, such as Music, Digital Media Arts and Film, tend to think of Audiovisual Art as a subfield of their own, perhaps because that is how it has emerged from the perspective of these academic disciplines, and also perhaps because of the natural territoriality of academics with respect to the disciplinary specificity of their fields (Birtwistle, A. 2010). Authors naturally focus on those aspects of the field that fit the remit of their discipline (for example, Garro, D 2012 explores the field in terms of music, Lopes, D 2009 contextualises it as operating within Creative Coding). Filmmakers who make music, or musicians that have made films, may refer to their work as Audiovisual Art, but terms such as ‘Film’, ‘Music Video’ and ‘VJing’ are often sufficient to describe such practices. I assert that neither the historical nor the emerging canon of Audiovisual Art could be properly described by those terms.

What best characterises Audiovisual Art is the long-established practice exploring relationships *between* formal ideas and experiential observations shared across image and sound composition (Mcdonnell, M 2010, Lund, C. and Lund, H 2009). Much of this can be thought of as extending the concept of *Visual Music* (Moritz, W 1986). The earliest works of Visual Music were preoccupied with formal characteristics of music composition applied to image making, in all probability due to the lack of sufficient vocabulary to describe abstract time-based media concepts in the visual arts. In addition, grammars of visual composition, such as Graphic Design principles developed by Bauhaus tutors, including Johannes Itten, follow a line of thought derived from their colleague’s early experimentation in similar fields (Itten, J 1975).

The early work of John and James Whitney (*Abstract Film Exercises*, 1948), and certain of Norman McLaren’s animations (for example, *Dots*, 1942) are known for their extraordinary innovations in sound synthesis. These innovations were primarily achieved through the application of techniques from the visual arts to sound practice, namely drawing and animation[7]. It should be noted that in both these cases, the works mentioned aimed to create and explore an abstract, unified, audiovisual production method and outcome: abstract so as to highlight the formal relationships between the two media, and unified in such a way that neither image nor sound would be meaningful without the other. So it is clear that Audiovisual Art predates electroacoustic music as a

[7] John Whitney is recognised as one of the fathers of computer graphics, and motion graphics more generally (Moritz, W 1986).

field of enquiry incorporating experimental sound synthesis, and also that it has its own aesthetic concerns.

These explorations are now known to exploit dedicated multi-sensory cells in the brain, that only fire when strong audio-visual connections occur. It is known that these effects cause modulation in attention mechanisms, leading to a concrete, observable form of complex neurological stimulation from wholly abstract material. When added to the consideration that this stimulation is involuntary, we get much closer to a fuller understanding of Audiovisual Art as an art form. The notion of a physically affective experience, created through effective connections between abstract image and sound, lies firmly at the centre of the practice.

So to summarise, it is three things that most accurately characterise Audiovisual Art. First, the focus on the audiovisual experience over the musical or visual, for the purposes of specifically and directly modulating attention through multisensory stimulation. Second, the use of abstraction to focus the work on these experiential qualities, as opposed to other qualities such as story, characters, context - elements that might detract from the experience of the relationship itself. Third, the development and application of new technological approaches to more effectively explore these principles in practice. We can consider the CodeCircle project, presented here, as an example of such a practice, but it is also more coherently understood as an attempt to enhance pedagogy around such practices (see below).

The Importance of Code in Audiovisual Art

Audiovisual Art often focuses on pattern making - sonically, visually and as an audiovisual interaction between the two. One of the central challenges of Audiovisual Art is to improve the manner by which such patterns and relationships can be explored. Non-real time methods for Audiovisual Art have been, and continue to be, functionally identical to standard film-making and animation practice in the majority of cases. However, the problem with such approaches is that although it is far easier to generate high-quality visual art and sound in non-real time, it is more challenging to create and explore strong connections between the two streams, largely because production methods do not afford for the easy sharing of data or information across sound and image. For example, most video post-production tools do not allow audio data to be analysed by feature extraction tools, or images to be synthesised based on their relationship to sound. This is important, because, as I have stated in the past (Grierson, M 2005), textural information - for example, the timbre of a sound, or the visual texture of an image, can be considered of primary importance in Audiovisual Art and Composition - more so than simple, temporal mapping or alignment between data streams (Abbado, A 1988, Ikeshiro, R, 2013).

In order to capture and explore textural relationships between sounds and images, composers require access to raw data – the ability to both generate and analyse pixels and audio samples, including any associated features that can be derived from them. One of the most efficient and powerful methods for achieving this is through signal processing at the level of computer code. To be absolutely clear, without code and algorithms, no form of digital artmaking would be possible at all.

Therefore, code is a primary medium for the production of Audiovisual Art. Code can therefore be seen as central both in real time and non-real time audiovisual composition. It could even be said that to a large degree, the field of Audiovisual Art and Composition is **dominated by code**. One can produce a work in real time, and / or render a high definition version of a piece in non-real time, but to do so relies upon code. There are exceptions, such as analogue video synthesis and physical systems, and such practices definitely deserve special attention, but the definition and use of these approaches is not the subject of this chapter.

Creative Code and Interactivity

Historically, artists from the world of fine art are not often considered to be programmers, and vice versa. There are welcome exceptions to this, as already mentioned, and one or two key historical events that are well known, such as the 1968 *Cybernetic Serendipity* Exhibition. However, coders are often seen as technicians, and artists seen as visionaries. This is not always the case in the fields of Computer Music and Computer Graphics, where computer-use is primary. It is important to remember that John Whitney Sr., considered by many to be the first computer graphics artist, canonised the term 'Audiovisual Composition', and I consider it in much the same manner he described (Whitney, J. 1980)[8]. Furthermore, Computer Music pioneers were as preoccupied with art-making as with signal processing – for example, Daphne Oram pioneered synthesiser design[9] in the United Kingdom whilst also attempting to define electronic music as a compositional approach (Oram, D 1972).

Despite these exceptions, computer programming is not historically associated with the arts in the broadest sense, and even within the above fields, artists often see programming as a technical act, performed by technicians, not artists[10]. This view leads to the frequent pairing of artists or composers with technicians for the realisation of digital artworks[11], and tends to lend weight to assumptions that technical skill is not a requirement in order for creative acts to be produced. However, this approach is contrasted by recent practices such as Creative Code, where craft and technique are essential for the realisation of artistic expression. It is fully accepted that to many, the notion of technique being central to any creative art is in itself an anathema, but it is entirely clear it is the confluence of science, technology and art, and the mastery of that liminal space between them, that leads to Audiovisual Art, Creative Code and related practices.

In the last decade there has been a noticeable increase in the number of artists working with code as material. This can be attributed in large part to the increased accessibility of online educational resources, related creative coding software tools such as Processing (processing.org), openFrameworks (openframeworks.cc), Cinder (libcinder.org), Supercollider (supercollider.github.io), and visual data-flow languages such as Pure Data (msp.ucsd.edu), Max (cycling74.com) and VVVV (vvvv.org)[12].

These resources have made it much easier to code, whilst increasing access to the knowledge required to do so. With the use of carefully constructed languages, simplified programming syntax, and well-documented examples, these environments make it more possible for artists to hone their craft, and develop creative ideas interactively. This has allowed them to engage in more meaningful contemporary art making, ushering in the era of Creative Code.

However, such resources are not without their problems. The nature of the workflow is much the same as traditional programming in many cases, specifically those that feature textual programming. Perhaps you type code into an Integrated Development Environment (IDE). You compile your software (which can easily take minutes). You watch it run and interact with it. You attempt to improve it - without necessarily being clear with regard to the impact of your edits, then wait for your program to compile before you can experience it again. This is, in some ways, very far from an embodied experience, and not something we would associate with a natural, human-like, creative process, such as plucking a string on a Cello, or striking a Drum.

Visual data-flow languages such as Max, Pure Data and VVVV are more interactive, and perhaps as a result, more embodied than traditional approaches. For example, both Max and PD run software interactively as the program is edited. This provides a useful affordance – the experience of your edits is immediate. It is possibly for this reason that visual data-flow approaches are

[8] Although he worked mainly in the field of graphical signal processing in later life.
[9] The Oramics Machine is amongst the first music synthesisers to feature digital control
[10] This presents a subtle paradox, as one could say the same of any musician whose instrument requires technical skill.
[11] A practice which is common at a number of institutions including the eminent Institut de Recherche et Coordination Acoustique/Musique (IRCAM).
[12] VVVV is used by many Creative Coders who focus on the visual arts, but it is generally similar to Max / PD.

popular amongst musicians, as musicians often use real-time feedback and experience-led approaches in order to understand how their creative acts unfold in the world.

However, although this affordance is welcome, and in some cases, inspiring, unfortunately, the restrictions surrounding the visual data-flow paradigm can make certain forms of signal processing more challenging to learn and understand. For example, I would argue it is more challenging to compute buffer-level signal processing in visual data flow languages than it is to simply program them using traditional methods[13]. This is particularly the case if users engage with a specifically designed C++ DSP toolkit that has been designed to be easy to use, such as Maximilian (https://github.com/micknoise/Maximilian). So - to summarise - if the language and syntax of textual programming methods are good, but the interaction method is not, and if visual data flow environments have better interaction, but are not flexible enough to code bespoke signal processing routines, what are the next logical steps, and how do they impact on our design decisions for CodeCircle?

Just-In-Time and Interactive Programming

Interactive Programming describes a process whereby code compiles and runs whilst it is being written. The idea has been incorporated into CodeCircle in order to address the issues raised above regarding the lack of an interactive feedback process in traditional programming, in the hope that it will help make learning to program more experiential and embodied. The idea has many potential benefits, one being that it may allow the user to understand more quickly the relationships between their actions and any associated outcomes, increasing the likelihood that their actions might be implicitly associated with the output of the program directly. This may increase the possibility for intuitive leaps to occur, whilst also making the process of programming considerably more engaging. Useful for programming where the desired outcome or problem is not fully known or understood, this method is a strong candidate for practices such as Computer Music and Audiovisual Art. In fact, it has flourished in this domain, with Max / PD and Supercollider being prominent examples.

Interactive Programming tools most often use a Just In Time (JIT) approach. JIT platforms have been around for decades in a variety of forms (McCarthy, J 1960). Contemporary C++ compilers such as LLVM now have the capacity for JIT compilation that can enable Interactive Programming (CLANG). The contemporary C++ framework, JUCE, now includes an Integrated Development Environment (IDE) based on the principle (The Projucer, 2015). The Chuck programming language can also be described as an Interactive Programming platform (Wang, G 2008).

Such approaches have become very powerful in the domain of JavaScript web applications, particularly since contemporary browsers began to support JIT compilation, greatly accelerating JavaScript, which is an otherwise interpreted language. Modern browsers now support JIT compilation of accelerated graphics (the graphics framework, webGL was finalised 2011), and buffer-level audio DSP (the webAudio framework was finalised 2014, and is now the subject of a regular academic conference, WAC (The WebAudio Conference)). Platforms such as glslsandbox.com and shadertoy.com are excellent examples of contemporary Interactive Coding platforms that rely on such processes. These are used by learners and visual artists alike to develop and demonstrate their skill, and tools such as livecodelab.net use the same approach (graphically at least), coupled with a hugely simplified yet elegant syntax to achieve the same results with children who are learning to code. What is crucial here is that these platforms are becoming a central mechanism whereby people learn to code for creative purposes. I would argue that these platforms, specifically shadertoy.com, glslsandbox.com and livecodelab.net, represent a step change in how people now engage with Audiovisual Art and Creative Code.

Live Coding and Interactive Programming

[13] Consider implementing a time-domain convolution reverb in Max or PD and you might see what I mean. What is just a few lines of code in C++, can be very challenging to program in Max or PD.

Although there is a great deal of crossover between Interactive Programming as an approach, and the Live Coding movement represented by collectives such as TOPLAP, Interactive Programming is a separate field of interaction research. In addition to the use of ideas from the field of Interactive Programming, Live Coding highlights concerns that are fundamentally performative, aesthetic, cultural and conceptual in nature - such as the imperative for performers to show their code to audiences, and the importance of performer or domain specific language design. Interactive Programming is an interaction technique, and not concerned with aesthetics or culture, instead being focussed on problem solving issues, including learning, usage, experience and output. It is clear, as has already been mentioned, that Live Coding is a form of practice that shares a number of similar approaches and methods with Audiovisual Art and Creative Code generally. As a result, concerns raised here may well apply across such practices. However, it is important to recognise that Interactive Coding is a technical solution to an interaction problem that has been the subject of debate for over 50 years, whereas Live Coding is considerably more than just this – it is an artform with a number of different technical and aesthetic challenges[14].

Importantly, the JavaScript-based, Interactive Programming tools mentioned earlier use more-or-less immediate interpretation i.e. they execute code as soon as the user finishes typing. It is perhaps this which is most interesting to our discussion with respect to embodied interaction, learning and use in the context of programming. However, this approach may not always be a useful or recommended Live Coding technique. For example, this specific method makes it difficult for the user to dictate when the code will be executed – it is executed immediately, and following any further edits, it will execute from the start once more. In Live Coding, specifying precisely when code runs can be central to a live performance.

Nevertheless, it is clear that there are shared concerns between the more or less technical field of Interactive Programming, and Live Coding, as evidenced by the excellent livecodelab.net, and other impressive platforms such as *Tidal*[15]. However, it is clearer to say that Interactive Programming is a technique that features in some forms of Live Coding as well as other fields of practice, but that on its own is not 'Live Coding'. This does not mean that Interactive Programming platforms such as CodeCircle cannot be used for Live Coding, just that they might not be a very good choice for doing so, as the program will keep being interrupted. It is fair to say that CodeCircle could be used as a Live Coding platform, but as such, being that it currently only features Interactive Programming methods as a means to solve interaction issues, it might not be the best platform available for such purposes.

Collaborative Coding

As the name suggests, Collaborative Coding describes a process where different people collaborate to write a single piece of software. This is a common practice, made more effective with version control tools such as git[16] (by Linus Tovalds, creator of Linux), svn and similar. These tools provide a central online repository where all code is stored. Users make copies of this repository, make edits, and then submit edits back to the repository. Any conflicts between different user's edits are managed at that point. This process helps to ensure that edits to code by different people can be more effectively controlled and integrated, for example, when they happen at similar times, or when they relate to the same precise section of any particular piece of code being edited. It is safe to say that almost no software is or should be written without the use of some form of version control as described above[17].

[14] For an introduction to the culture and practices of Live Coding, visit https://toplap.org
[15] https://tidalcycles.org
[16] https://git-scm.com
[17] At Goldsmiths, all computing students must use git version control, regardless of the complexity of their task

A similar, more interactive kind of collaborative coding has recently emerged in new software development platforms such as cloud9 (c9.io), collabedit.com, and etherpad.org. Users can see each other's edits immediately, or at least as soon as possible after they occur. This kind of interaction can be experienced in google's popular 'google docs' platform[18], but it is not supported by any major professional development environments, such as Windows Visual Studio, or Xcode. This may be with good reason, but as little is currently known regarding the possible impact of this new approach, it remains a largely unexplored territory.

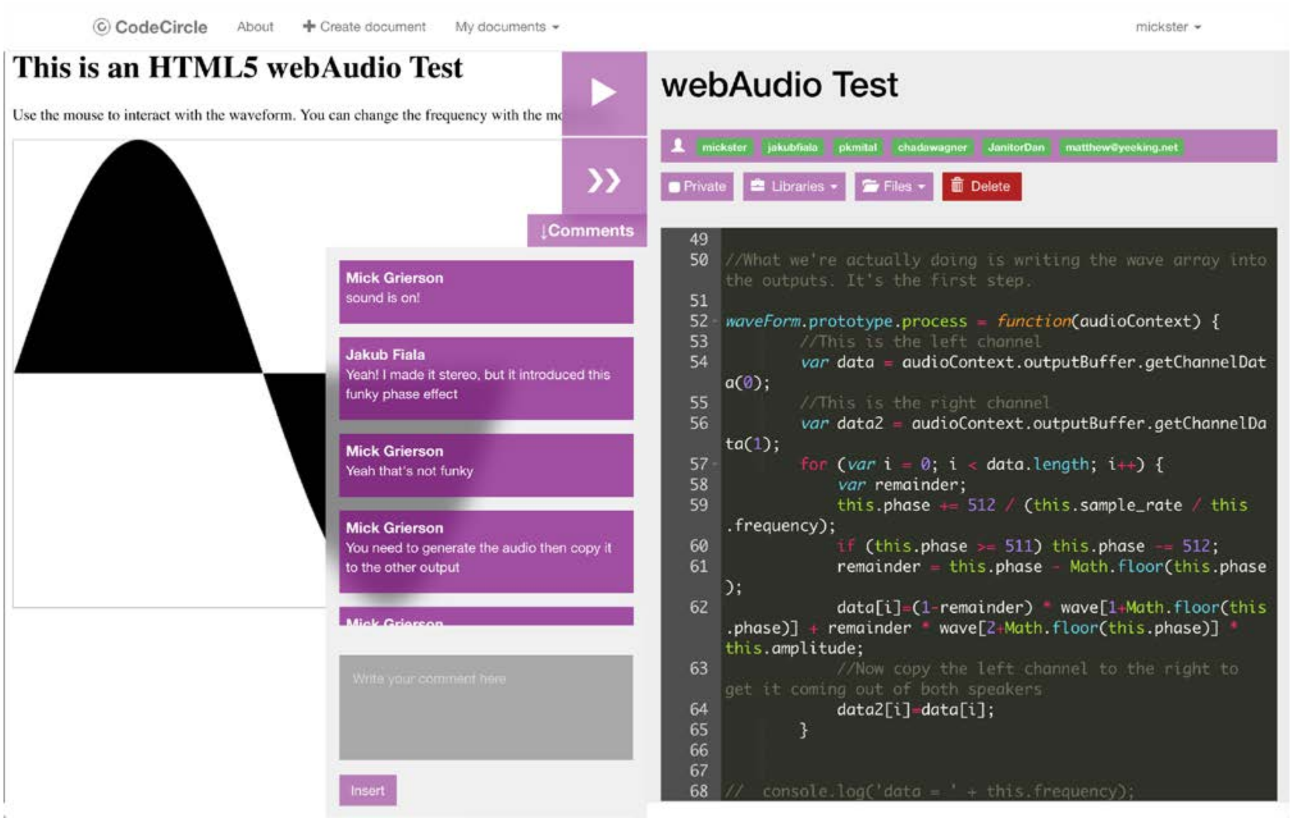
The advantages of such collaborative approaches are numerous. Programming can be a highly challenging task, and code sharing is common. Supporting code sharing in this way seems eminently sensible, allowing users to work together on complex programming problems – for example, composers and artists often work together when generating various forms of art. However, computer programming is more or less considered a solitary task – and it need not be. In education and the creative arts this new method may be hugely important. For example, teachers could comment on student work, make suggestions, and monitor group work in real-time with the advantage of knowing precisely what each student has contributed. It can also facilitate peer learning, rewarding students for social interaction and assisting other learners through mutual shared experience. Finally, it may bring to light collaborative approaches reflecting improvised composition and performance, that would otherwise not emerge in the Creative Computing discipline. As we will see, it is a founding element of CodeCircle for precisely these reasons.

CodeCircle

CodeCircle has been developed using the Full-Stack web development framework, Meteor (www.meteor.com), in collaboration with Dr Matthew Yee-King, and Jakub Fiala (Fiala et all, 2016). Meteor is an excellent tool for creating websites that feature multi-user interaction. This is due to its support for reactive architectures. Users interact with databases that automatically update themselves when clients or servers cause or detect changes to database elements. We adapted this approach, applying it to the development of a web-based programming tool, rather than using it for the purposes of website design specifically. We do this by referencing code documents as database objects, whilst recording code edits as user-specific database entries. Crucially, we render the document in the browser, with the code window on top and to the right, as can be seen in figure 2.

Figure 2 shows the main interface for the document editor. On the right hand side one can clearly see the code edit window. This is a version of the ACE code editor (https://ace.c9.io/). The ACE code editor includes highlighting for different languages (HTML5, CSS3, JavaScript, Coffee Script etc.). We have also included support for dynamic error checking using JSHint (http://jshint.com). This combination is powerful – providing instant, in-place feedback on code edits, allowing users to understand the implications of their creative coding decisions more fluidly. Figure 2 also shows that the file has been edited by a number of authors, represented by individual usernames on a green background in the upper right section of the document window. By parsing the database, it is very simple to discover which edits were created by which users. This is of the highest importance with respect to assessing document ownership and individual effort. In this way, group projects can be undertaken with confidence that convincing evidence will be available to indicate the provenance of any code excerpts within a document. Furthermore, information about how people code can be analysed offline and used to help better understand the kinds of strategies people might use in certain circumstances. We have used this approach to gather data on thousands of users in order to better understand the creative, exploratory strategies that might help people develop better work (see below).

[18] https://www.google.com/docs/about/



This is an HTML5 webAudio Test

Use the mouse to interact with the waveform. You can change the frequency with the mo

webAudio Test

Mick Grierson
sound is on!

Jakub Fiala
Yeah! I made it stereo, but it introduced this funky phase effect

Mick Grierson
Yeah that's not funky

Mick Grierson
You need to generate the audio then copy it to the other output

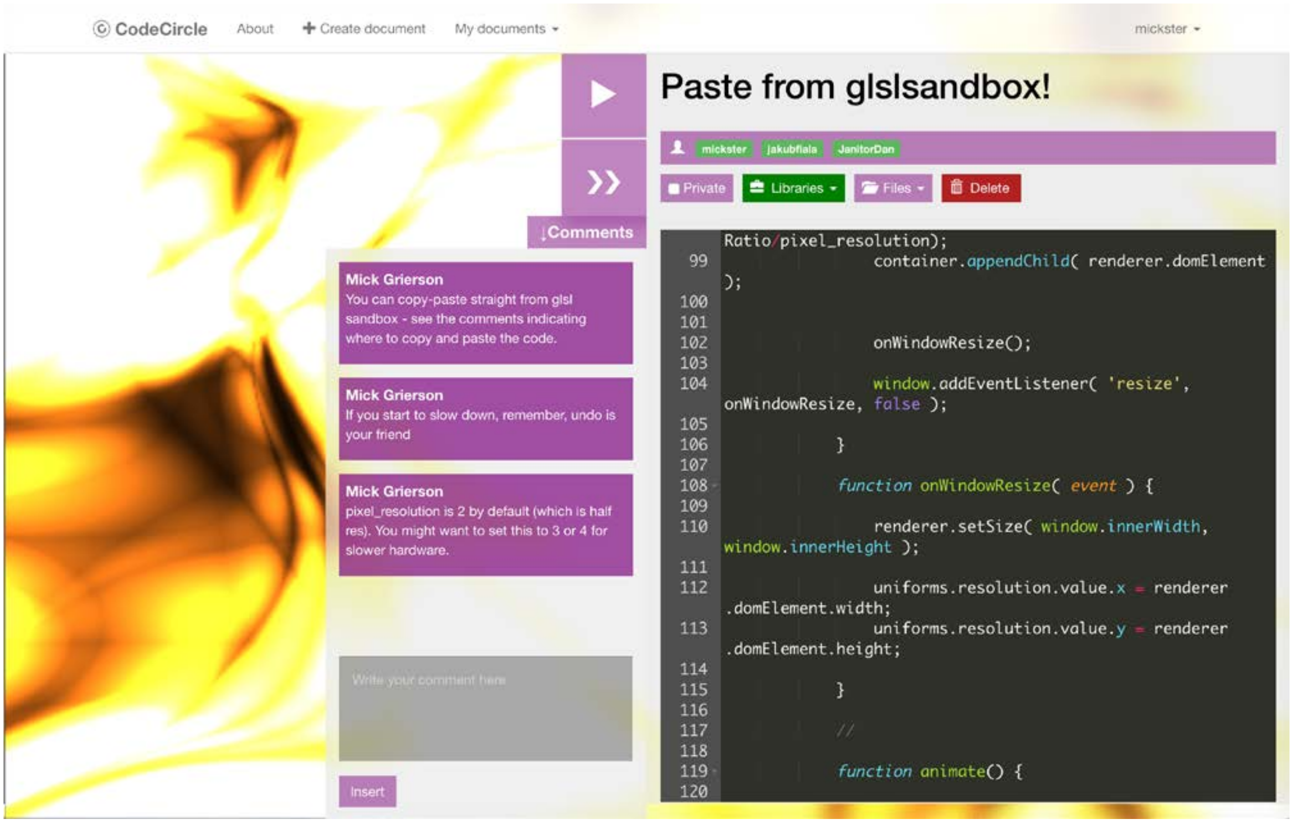
Mick Grierson

Write your comment here

```

//What we're actually doing is writing the wave array into the outputs. It's the first step.

waveForm.prototype.process = function(audioContext) {
        //This is the left channel
        var data = audioContext.outputBuffer.getChannelData(0);
        //This is the right channel
        var data2 = audioContext.outputBuffer.getChannelData(1);
        for (var i = 0; i < data.length; i++) {
            var remainder;
            this.phase += 512 / (this.sample_rate / this.frequency);
            if (this.phase >= 511) this.phase -= 512;
            remainder = this.phase - Math.floor(this.phase);
            data[i]=(1-remainder) * wave[1+Math.floor(this.phase)] + remainder * wave[2+Math.floor(this.phase)] * this.amplitude;
            //Now copy the left channel to the right to get it coming out of both speakers
            data2[i]=data[i];
        }


//  console.log('data = ' + this.frequency);
```

Figure 2: The basic CodeCircle Interface. The code editor is on the right. Above the editor are file operations, including elementary permissions. The code is rendered underneath.

Figure 2 also shows an interactive session featuring buffer-level signal processing. A discussion regarding how to implement stereo audio is visible in the collapsible comments pane. Here we can see the potential educational value in terms of remote tutorial support.



Mick Grierson
You can copy-paste straight from glsl sandbox - see the comments indicating where to copy and paste the code.

Mick Grierson
If you start to slow down, remember, undo is your friend

Mick Grierson
pixel_resolution is 2 by default (which is half res). You might want to set this to 3 or 4 for slower hardware.

```
Ratio/pixel_resolution);
            container.appendChild( renderer.domElement
);


            onWindowResize();

            window.addEventListener( 'resize',
onWindowResize, false );

        }

        function onWindowResize( event ) {

            renderer.setSize( window.innerWidth,
window.innerHeight );

            uniforms.resolution.value.x = renderer
.domElement.width;
            uniforms.resolution.value.y = renderer
.domElement.height;

        }

        //

        function animate() {
```

Figure 3: WebGL shader code executing in the Code Circle platform

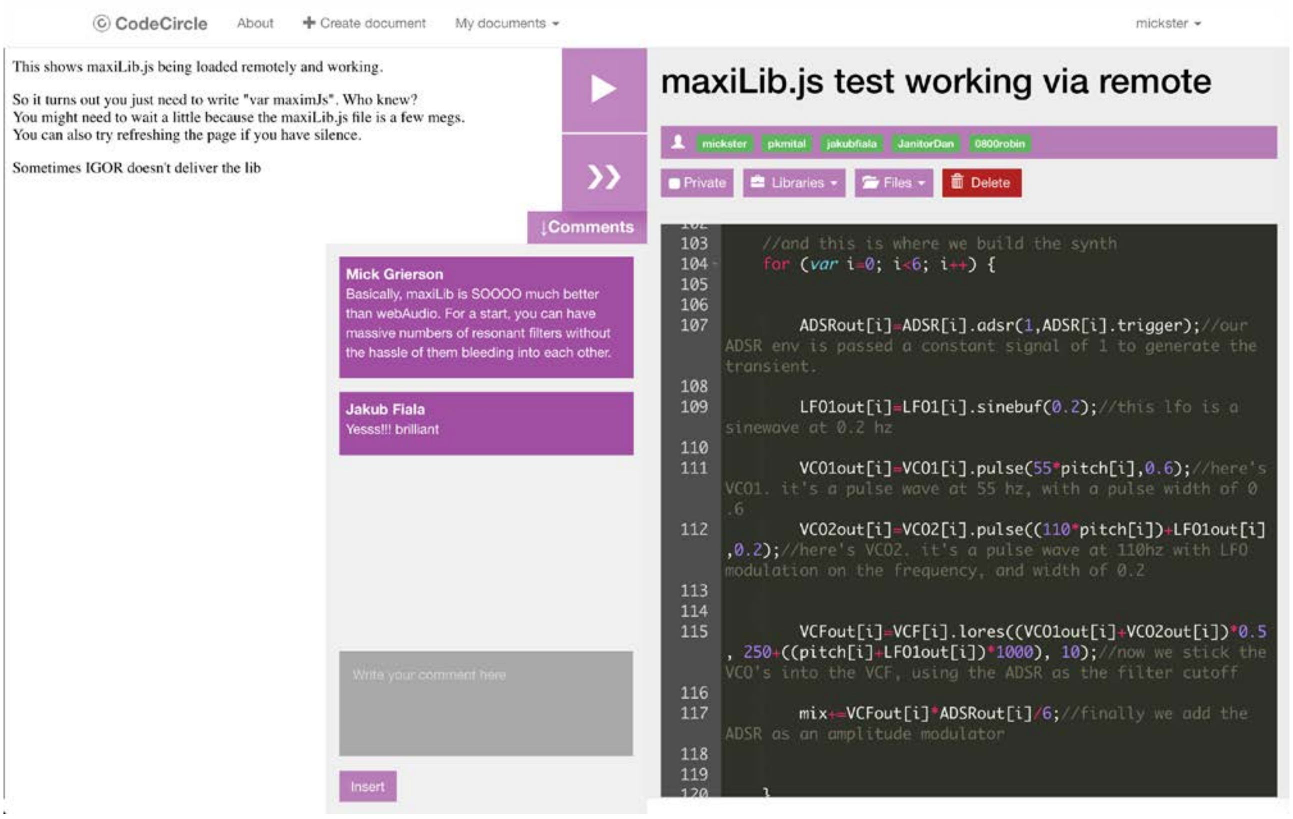
Figure 3 shows the potential for accelerated computing provided by the platform. The screenshot shows a template created in order to instruct students regarding how to implement a basic fragment shader. The template has been specifically designed to be compatible with glslsandbox.com, a popular Interactive Coding platform - documents can be copied and pasted directly into CodeCircle. In addition, the comments section gives specific advice to beginners regarding how to improve the performance of the graphical content - a factor that varies from machine to machine.

The Maximilian C++ DSP framework has been converted to JavaScript via *emscripten* (https://github.com/kripken/emscripten), a tool that transpiles[19] C++ code to JavaScript. This provides comprehensive, buffer-level support of professional level synthesis, sample manipulation, granular synthesis, FFT / iFFT-based manipulation, Music Information Retrieval and Atomic (wavelet-based) synthesis, and many other features not supported by the webAudio framework. The webAudio framework is an excellent innovation, standardising digital audio across all modern browsers, but there are some specific electronic and computer music approaches that it does not support. Transpiling offers the opportunity for users to such methods, with the same functionality as complex C++ signal processing libraries, whilst interactively programming in a web browser.

In addition to buffer level signal processing and accelerated computer graphics, the platform supports comprehensive asset loading and manipulation. Samples, images, video and other assets are uploaded to the platform, and then stored in the database alongside the document. When a document is forked (copied by another user), these assets migrate with the document. When a user downloads any document, all assets are contained within it, and the document can be run in any web browser with all such assets, without the need of a webserver[20].

[19] As opposed to compiling – specifically the library is compiled into bytecode first, and then translated or 'transpiled' to asm.js, which is a low-level form of JavaScript.

[20] This functionality represented a considerable technical challenge, specifically relating to cross-domain requests and other issues.



This shows maxiLib.js being loaded remotely and working.

So it turns out you just need to write "var maximJs". Who knew?
You might need to wait a little because the maxiLib.js file is a few megs.
You can also try refreshing the page if you have silence.

Sometimes IGOR doesn't deliver the lib

Mick Grierson
Basically, maxiLib is SOOOO much better than webAudio. For a start, you can have massive numbers of resonant filters without the hassle of them bleeding into each other.

Jakub Fiala
Yesss!!! brilliant

maxiLib.js test working via remote

```
    //and this is where we build the synth
    for (var i=0; i<6; i++) {


        ADSRout[i]=ADSR[i].adsr(1,ADSR[i].trigger);//our ADSR env is passed a constant signal of 1 to generate the transient.

        LFO1out[i]=LFO1[i].sinebuf(0.2);//this lfo is a sinewave at 0.2 hz

        VCO1out[i]=VCO1[i].pulse(55*pitch[i],0.6);//here's VCO1. it's a pulse wave at 55 hz, with a pulse width of 0.6
        VCO2out[i]=VCO2[i].pulse((110*pitch[i])+LFO1out[i],0.2);//here's VCO2. it's a pulse wave at 110hz with LFO modulation on the frequency, and width of 0.2


        VCFout[i]=VCF[i].lores((VCO1out[i]+VCO2out[i])*0.5, 250+((pitch[i]+LFO1out[i])*1000), 10);//now we stick the VCO's into the VCF, using the ADSR as the filter cutoff

        mix+=VCFout[i]*ADSRout[i]/6;//finally we add the ADSR as an amplitude modulator


    }
```

Figure 4: C++ DSP code transpiled and running in the CodeCircle platform.

CodeCircle in the wild

This new, interactive version of CodeCircle has been in active development online since late 2015. During that time we have used it for the delivery of two MOOCs, one as part of an international collaboration with the Los Angeles based online learning provider, *Kadenze*, and a second with UK MOOC provider, *Futurelearn* (produced by Dr Matthew Yee-King). The Kadenze programme was a ten-week course in Audiovisual Art, supporting thousands of learners. Learning materials on audio signal processing, visualisation techniques, computer vision, algorithmic composition methods, and other key topics were presented as runnable documents on CodeCircle (which are still freely available at codecircle.com). Students were instructed to make copies of example code, and work through video tutorials delivered by the Kadenze.com platform to explore Audiovisual Art-making principles. All assignments were completed on the CodeCircle, allowing us to perform statistical analysis to more fully understand how students learn to program for creative practice.

Analysis of activity by users attempting to complete specific tasks in audiovisual processing, undertaken by Dr Matthew Yee-King, Professor Mark d'Inverno and myself, have demonstrated a potentially important finding we have reported at the 2017 ieee EDUCON conference (Yee-King, M Grierson, M & d'Inverno, M 2017). Our results suggest that students who are engaged in programming for the purposes of creative activity of their own devising, such as the creation of artworks and similar projects, tend to achieve higher grades in programming tasks than students engaged in more traditional Science, Technology, Engineering and Maths (STEM) exercises. The evidence for this comes from the number of delete operations carried out by highly achieving learners. In our study, delete operations are significantly statistically correlated with higher grades, and also with creative arts methods[21].

This result suggests that the platform itself is useful for exploring creative approaches to technology-enhanced creativity, a subject that sits right at the heart of Creative Computing, and Audiovisual Art specifically. This work is ongoing, but does appear to evidence that the design of CodeCircle may help to more effectively reveal the significance of this liminal, poorly-defined space that Creative Computing practices occupy between Science and the Arts.

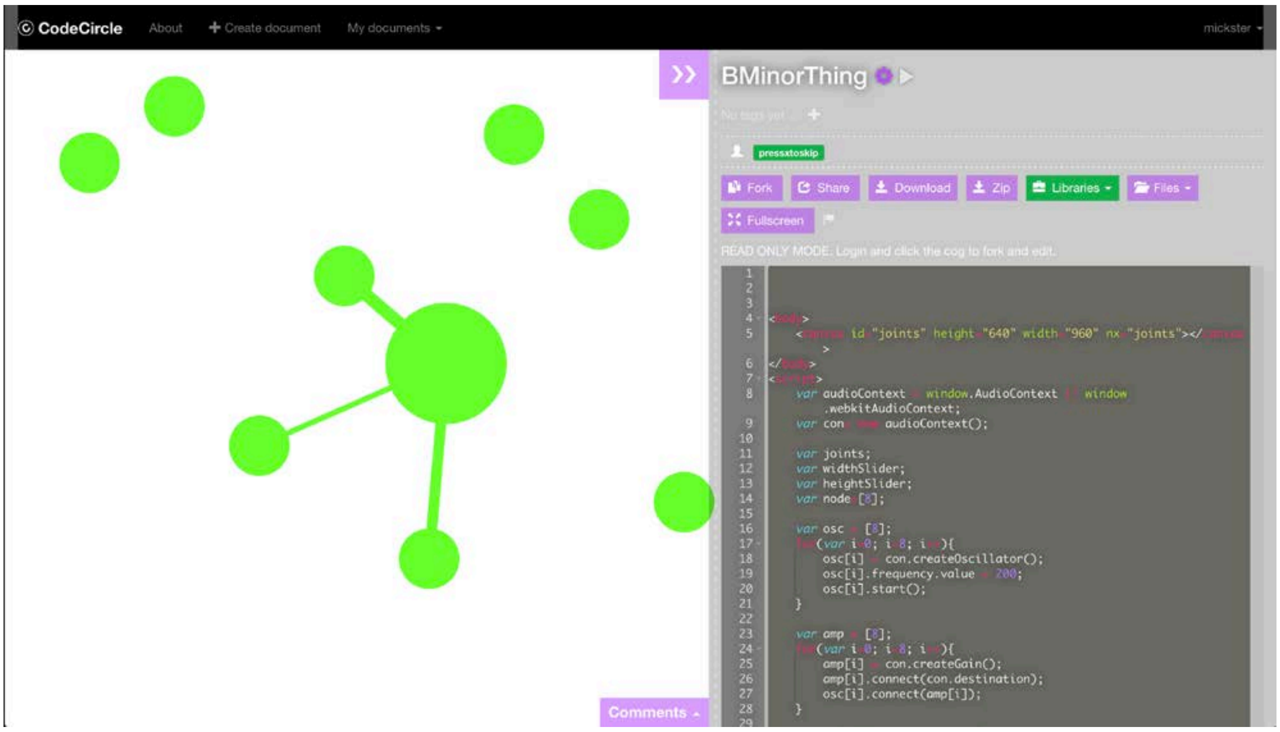
We have also been able to use CodeCircle to explore methods that are only now emerging in the field of Audiovisual Art and Creative Code, such as Machine Learning (ML) for electroacoustic music. For example, we present work towards rapid prototyping of electronic musical instrument interfaces using Interactive Machine Learning (IML) in "Rapid Prototyping of New Instruments with CodeCircle" (Zbyszyński, M et al, 2017). This work shows how creative coders can use machine learning to help develop better interactive tools through the use of our own machine learning libraries embedded in the CodeCircle interface (the rapid-mix-api). The machine learning Application Programming Interface (API) we have developed is based on Rebecca Fiebrink's *Wekinator* (Fiebrink, R, and Cook, P, 2010), and has been developed as part of the project, RAPID-MIX[22].

Examples of Work Created on the Platform

Overall there are currently several thousand users on the CodeCircle platform, each with a number of documents. This creates something of a problem, in that it is quite hard to find specific examples of good practice amongst the available material. However, by browsing the platform, and looking at the data in order to discover documents that are more popular than others, it has been possible to extract a number that are of interest. Because the platform is online permanently, it is possible to retrieve specific work by student learners that demonstrate the potential of the platform by URL, and this has been done on occasion. Importantly, as is the case with other Creative Coding tools mentioned earlier in this chapter including Max, PD, openFrameworks etc., users most often use the platform to sketch out ideas, and only a very small number of public documents feature completed works. Documents produced by the community include basic demonstrators for sound generation and composition methods, audiovisual instruments as well as a few presentable pieces. Below are some examples of such works. A video of some of these examples, and a great number of others can be found at https://www.doc.gold.ac.uk/~mus02mg/cc.mp4.

The example *BMinorThing* by user 'pressxtoskip' demonstrates proximity based audiovisual synthesis, where the volume of specific partials relates to the distance of a single circle in relation to a series of static circles. The first circle is controlled with the mouse, and as this circle approaches other circles in the composition, the amplitude of a specific partial or set of partials is increased as a function of the distance. Visual links between each element in the system are drawn thicker as objects get closer together, reinforcing the relationships between each partial and the dynamic system that controls them. This creates a simple audiovisual instrument that can be used as a basis for generating harmonic textures.

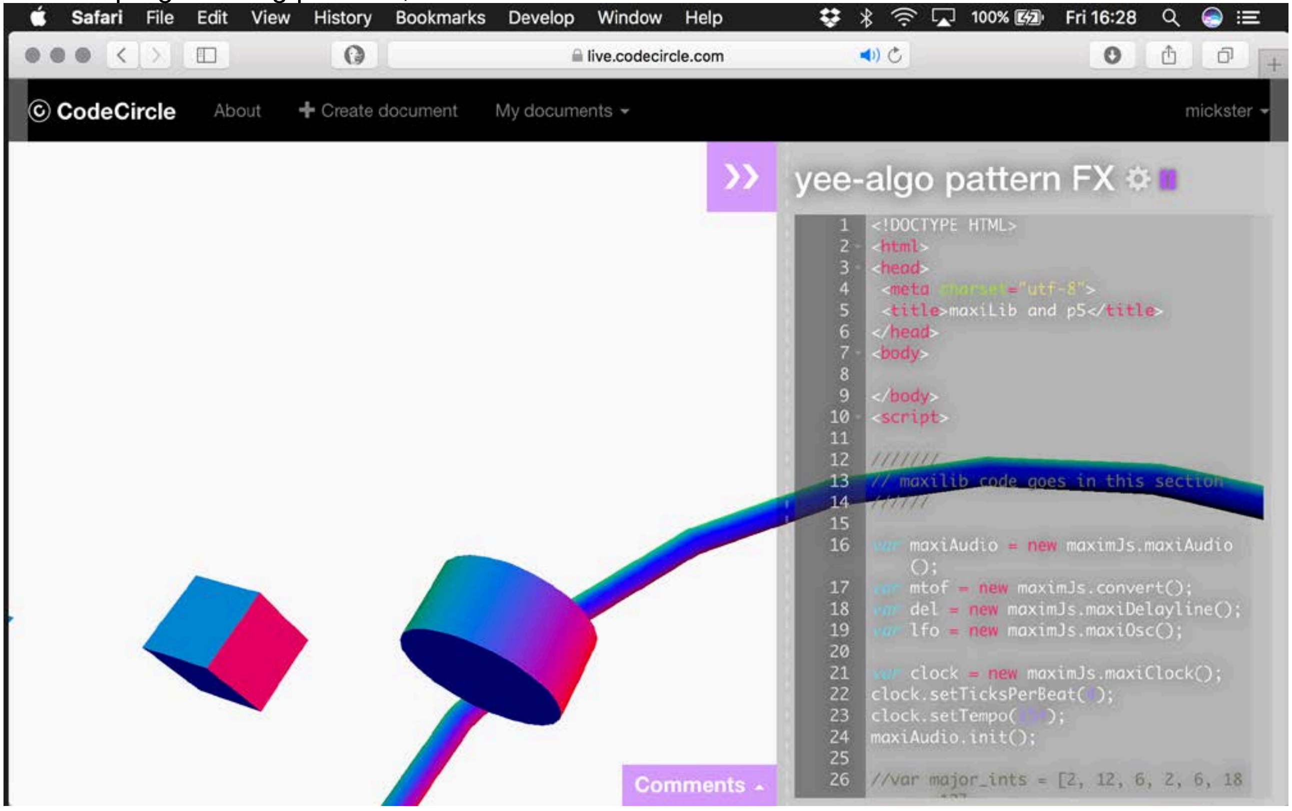
[21] d'Inverno, Yee-King and myself reflect on how this might relate to John Dewey's notions of experience, creativity and 'Inquiry', put forward in *Art as Experience* (1934), and *Logic: The Theory of Inquiry* (1938). See (Yee-King, M., Grierson, M., & d'Inverno, M., 2017)
[22] Funded by the European Commission, Horizon 2020. The IML tutorials are freely available on CodeCircle.com, tagged *RAPID.*



```



<body>
    <canvas id="joints" height="640" width="960" nx="joints"></canvas
        >
</body>
<script>
    var audioContext = window.AudioContext || window
        .webkitAudioContext;
    var con= new audioContext();

    var joints;
    var widthSlider;
    var heightSlider;
    var node=[8];

    var osc = [8];
    for(var i=0; i<8; i++){
        osc[i] = con.createOscillator();
        osc[i].frequency.value = 200;
        osc[i].start();
    }

    var amp = [8];
    for(var i=0; i<8; i++){
        amp[i] = con.createGain();
        amp[i].connect(con.destination);
        osc[i].connect(amp[i]);
    }
```

Figure 5: BMinorThing by ‘pressxtoskip’, https://live.codecircle.com/d/cxuENqhG9ifkfkLHN

Yee-algo pattern FX is a simple Audiovisual Composition by user ‘cyleung274’, the username for whom appears to be a pun on Cycling74, which is the name of the company who make the popular creative programming platform, Max.

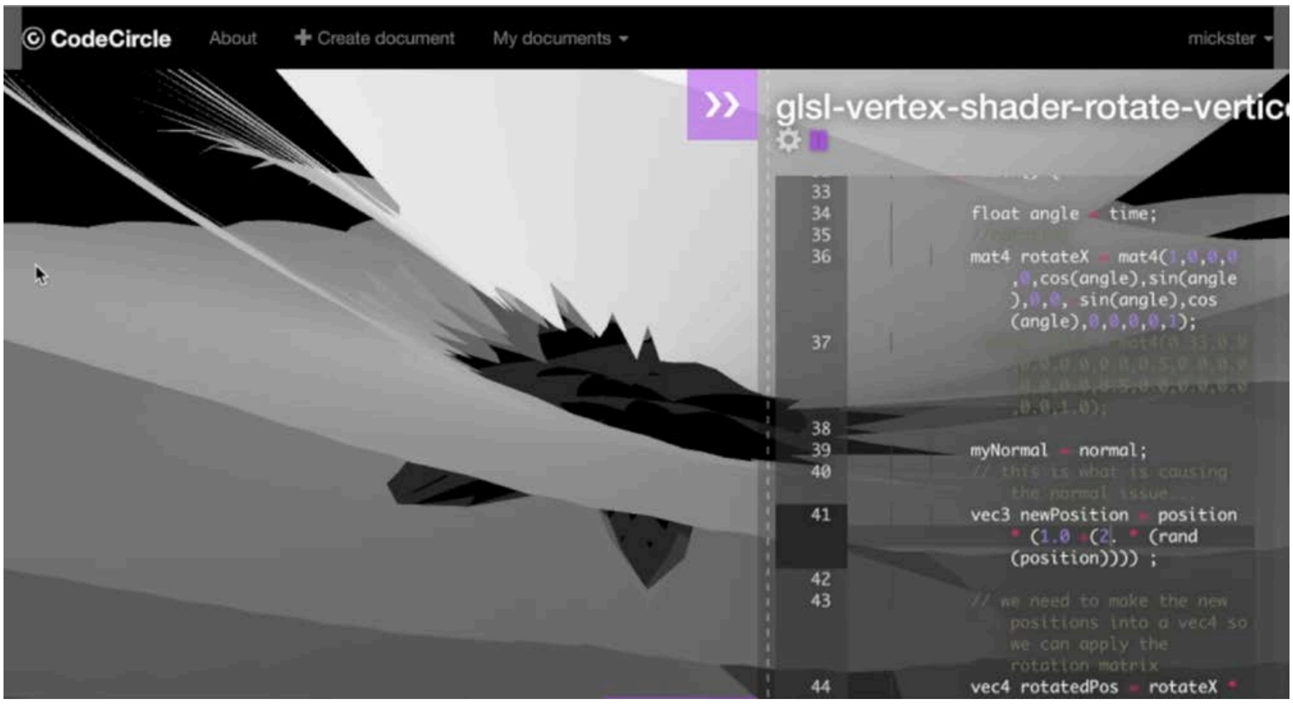


```
<!DOCTYPE HTML>
<html>
<head>
 <meta charset="utf-8">
 <title>maxiLib and p5</title>
</head>
<body>

</body>
<script>

///////
// maxilib code goes in this section
//////

var maxiAudio = new maximJs.maxiAudio
    ();
var mtof = new maximJs.convert();
var del = new maximJs.maxiDelayline();
var lfo = new maximJs.maxiOsc();

var clock = new maximJs.maxiClock();
clock.setTicksPerBeat(4);
clock.setTempo(154);
maxiAudio.init();

//var major_ints = [2, 12, 6, 2, 6, 18
```

Figure 6: *Yee-algo pattern FX* by ‘Cyleung274’ https://live.codecircle.com/d/LyRCpnLw9tf9YoGiA

This work features the use of webGL and Maximilian, used together to create a scene with ‘techno’ aesthetics, using neon-like primary coloured geometry on a white background, with synthesized audio.

Other webGL work includes a series of experiments with GLSL, where accelerated graphics is used to render complex scenes quickly. One such example is shown below. It is by a user on the second year Creative Computing programme at Goldsmiths, and shows a 3D sphere being deformed by frequency modulation.



```
float angle = time;

mat4 rotateX = mat4(1,0,0,0
    ,0,cos(angle),sin(angle
    ),0,0,-sin(angle),cos
    (angle),0,0,0,0,1);
[illegible]

myNormal = normal;
// this is what is causing
    the normal issue...
vec3 newPosition = position
    * (1.0 + (2. * (rand
    (position)))) ;

// we need to make the new
    positions into a vec4 so
    we can apply the
    rotation matrix
vec4 rotatedPos = rotateX *
```

Figure 7: A sphere being deformed by Frequency Modulation on the CodeCircle platform. This work was created by a second year undergraduate student enrolled on Goldsmiths' Creative Computing programme.

The artist Memo Akten (www.memo.tv) produced a tutorial for the CodeCircle Audiovisual Art MOOC on Kadenze, where he presented a series of code examples that helped users learn how to build dynamic systems, specifically particle systems[23]. A screenshot of the final work can be seen below. On the video, you can see the dynamic behavior of the particle system, and hear how it impacts on the sound synthesis approach. This example incorporates some excellent programming, and, although it is only an example, represents a very interesting approach to generating Audiovisual Art.

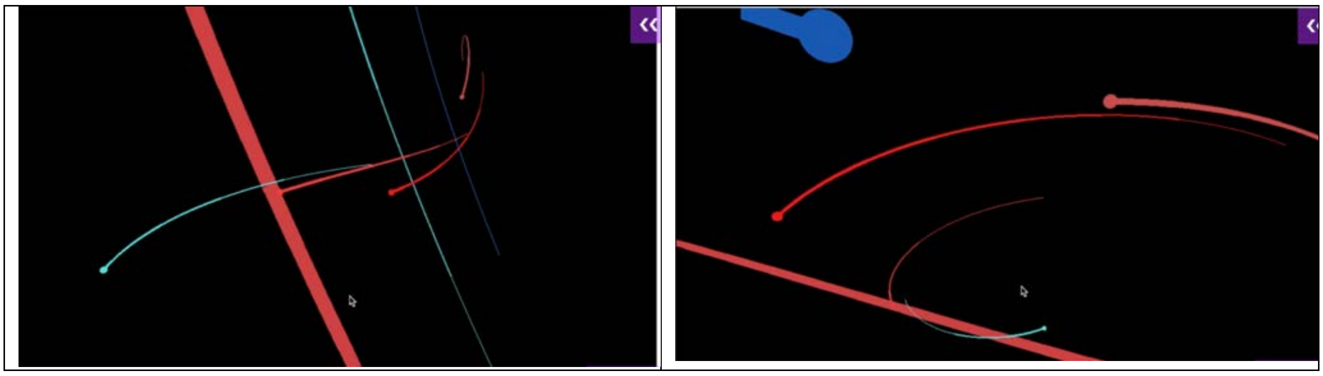
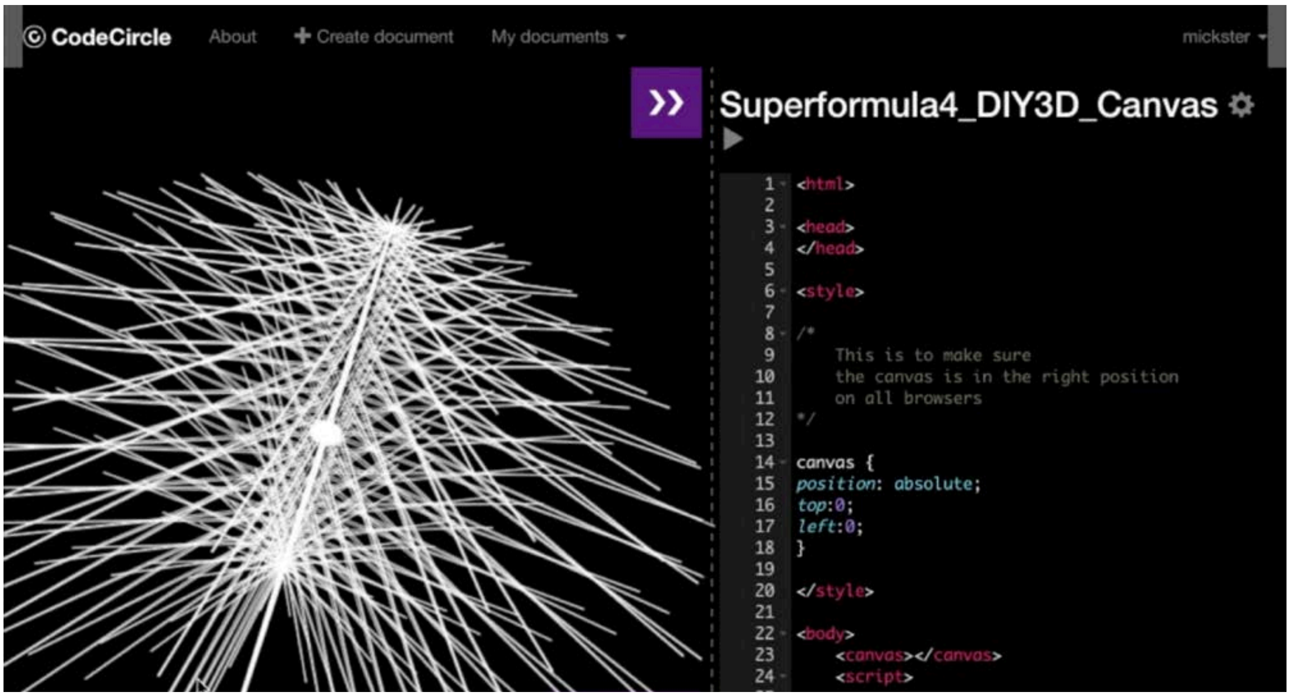


Figure 8: Two screenshots from Memo Akten's Particle system example.

[23] A particle system is a method often used in computer graphics whereby individual elements, or 'particles' appear like a group, often by being programmed to behave in certain specific ways, or by following certain rules. They can be thought of as different to swarms, as they are not necessarily attempting to emulate life of any kind. They are often used to generate clusters of physically modeled objects, like elements in an explosion.

Figure 9 shows a 3D superformula algorithm being explored by a student, who are themselves extending one of the given examples provided on the platform. In my experience, students sometimes consider the superformula mysterious and challenging. It allows a large number of shapes to be generated by quite simple means, and was patented by Johan Gielis in 2005[24]. However, it is nonetheless remarkably simple, being an extension of quite basic methods for shape generation in spherical coordinate systems, useful for creating curved shapes with polar coordinates. It is similar in fact to the approach Whitney describes in *Digital Harmony* (Whitney, J 1980). The work has been extended by a number of students at Goldsmiths, including Kingsley Ash, who used the superformula for Audiovisual synthesis[25]. The superformula is also used by the Audiovisual Artist Paul Prudence in a number of his pieces, including the phenomenal work, *Cylotone II*[26].



```
<html>

<head>
</head>

<style>

/*
    This is to make sure
    the canvas is in the right position
    on all browsers
*/

canvas {
position: absolute;
top:0;
left:0;
}

</style>

<body>
    <canvas></canvas>
    <script>
```

Figure 9: A wireframe Superformula on CodeCircle

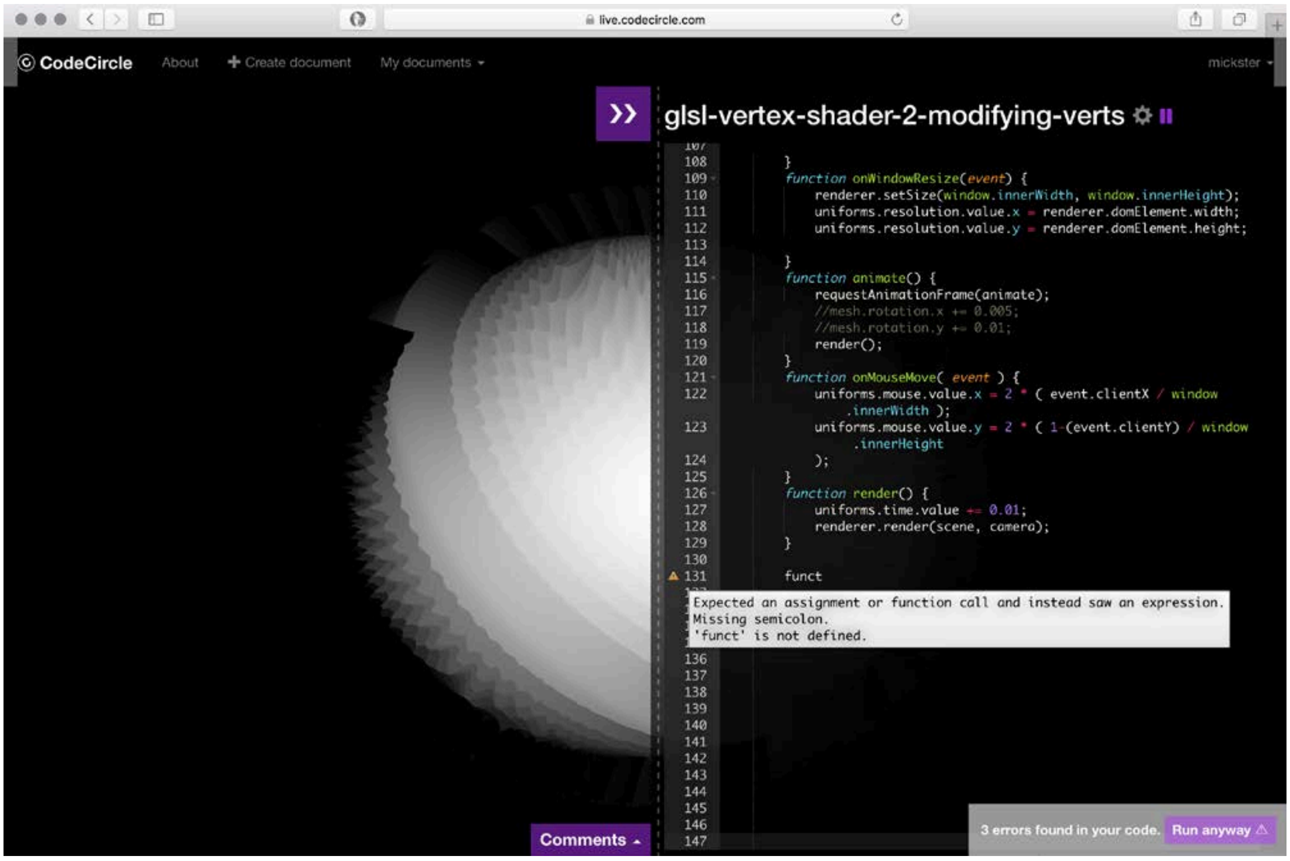
There are a range of very interesting audio-only works on the platform, some of which can be heard in the video referenced above. These explore a range of approaches to sound making, and are offered up with full source code – as all CodeCircle documents are – to the community. One example features the use of DSP methods taken from chiptune aesthetics. This is fascinating, particularly for those not familiar with early digital audio synthesis on personal computing platforms. The example makes great use of bitwise operations, where calculations are performed based on individual bits, rather than bytes[27]. In this way, the mathematical operations, rather than being performed on real numbers, are performed on 8 bit number representations.

There are a great many more documents on the platform, and they contain a range of fascinating approaches by students on our MOOCs, undergraduate and postgraduate programmes. For a greater selection of such examples, please visit the platform at http://www.codecircle.com, and see the example video here: https://www.doc.gold.ac.uk/~mus02mg/cc.mp4.

Conclusion

[24] EP patent 1177529, Gielis, Johan, "Method and apparatus for synthesizing patterns", issued 2005-02-02
[25] https://freshyorkshireaires.wordpress.com/portfolio/kingsley-ash-superformula004/
[26] http://www.paulprudence.com/?p=553
[27] https://live.codecircle.com/d/xCZs38ihyxxRBupdW

From our initial experiments using the platform, a key question that has arisen has been “Is language design *really* more important than interaction approach?” The reason for this question is best explained with reference to earlier arguments in this chapter regarding interactivity. With respect to well-known Creative Coding platforms such as those already discussed, language design has been the fundamental area of concern. However, our experience demonstrates the possibility that with real-time, Instant, Interactive Programming, fuelled by powerful JIT compilation, the use of more complex syntax may be made easier, as the distance between the act of creation and the experiencing of the outcome is as small as is practicably possible. Our platform supports any language or framework that can be run in JavaScript (including C++). However, as feedback and debugging is detailed and instantaneous (see Figure 5), it may not be necessary to sacrifice the power and detail of a language in order to increase ease of use. What this means is that although language design is important for a number of reasons, it may be that the interaction method has a greater impact on the ability of users to use code to create art.



```
108         }
109         function onWindowResize(event) {
110             renderer.setSize(window.innerWidth, window.innerHeight);
111             uniforms.resolution.value.x = renderer.domElement.width;
112             uniforms.resolution.value.y = renderer.domElement.height;
113
114         }
115         function animate() {
116             requestAnimationFrame(animate);
117             //mesh.rotation.x += 0.005;
118             //mesh.rotation.y += 0.01;
119             render();
120         }
121         function onMouseMove( event ) {
122             uniforms.mouse.value.x = 2 * ( event.clientX / window
                    .innerWidth );
123             uniforms.mouse.value.y = 2 * ( 1-(event.clientY) / window
                     .innerHeight
124             );
125         }
126         function render() {
127             uniforms.time.value += 0.01;
128             renderer.render(scene, camera);
129         }
130
131         funct
```

```
Expected an assignment or function call and instead saw an expression.
Missing semicolon.
'funct' is not defined.
```

3 errors found in your code.

Figure 10: Note the white pop-up window and hazard symbol above, clearly indicating code errors on line 131, and providing relevant reparatory advice.

Indications that transpiled C++ code may out-perform some embedded browser features require significant scrutiny. The notion of C++ code running in JavaScript may cause many developers to be sceptical, mainly as a result of the generally poor performance of JavaScript when compared to C++, but there is at least some comfort in the possibility that whatever the truth of this, the general approach could yield significant benefits in terms of functionality, accessibility and learning.

On the subject of Collaborative Coding, it seems clear that the social features can promote engagement and improve learning. My own experience is that the approach has led to very quick improvements in collaborative projects. The ability to witness changes to the code in real time, and understand the outcome of those changes immediately, is a feature that has generated great excitement, energy and engagement amongst users.

A possibly negative aspect of real-time collaboration in this context has been the potential for anarchic and irresponsible action. Users sometimes found that other users would break code in their absence, or occasionally, in front of them. On the other hand, there is an element of this that might add to creative outcomes; notions of hacking, breaking and cracking are an important part of both the creative arts and also computing culture, and the humorous outcomes it can lead to are potentially compelling. CodeCircle allows users to backup, duplicate and hide their documents, yet it is clear that such breakage can be compelling - there is something magical about the notion of a random editor copy-pasting a running 3D world into another person's document. So CodeCircle projects sometimes become deliberately broken. However, the platform allows all such adjustments to documents to be easily repaired, Exactly how we will allow such interactions to manifest is still emerging, but at present, users must protect their documents to ensure that they work, and it may be that this happens by default in the future.

The CodeCircle project has allowed us to explore better methods for embodied, globally oriented Audiovisual Artwork creation. The platform potentially improves access and support for creative practitioners exploring Audiovisual Art, and other related areas of Creative Computing practice, whilst simultaneously allowing us to better understand the impact of interaction and language design in the context of technology art practices. The platform is free to use, and contains a large number of interactive coding examples that we continue to use in teaching and research. Finally, the platform received funding in November 2016[28] to facilitate its development and maintenance as a teaching resource. This will allow the project to continue for the foreseeable future, aiding the dissemination of knowledge, methods and approaches that remain core to Audiovisual Art making and related creative practices.

References:

Abbado, A. (1988). Perceptual Correspondences of Abstract Animation and Synthetic Sound. *Leonardo. Supplemental Issue*, 1, p.3.
Birtwistle, A. (2010). *Cinesonica: sounding film and video*. Manchester: Manchester University Press.
Broad, T. and Grierson, M. (2017). *Autoencoding Blade Runner: Reconstructing Films With Artificial Neural Networks*, SIGGRAPH, 2017
Brougher, K. and Mattis, O. (2005). *Visual Music: Synaesthesia in Art and Music since 1900*. London: Thames & Hudson.
Carvalho, A. and Lund, C. (2015). *The audiovisual breakthrough*. Berlin: Collin&Maierski Print GbR.
Chion, M., (1994). *Audio-vision: sound on screen*. New York: Columbia University Press.
Daniels, D., Naumann, S. and Thoben, J. (2010). *Audiovisuology. an interdisciplinary survey of audiovisual culture*. Köln: Walther König.
Fiala, J., Yee-King, M., and Grierson, M., (2016) Collaborative coding interfaces on the web. Proceedings of the International Conference on Live Interfaces, pages 49–57
Garro, D. (2012). From Sonic Art to Visual Music: Divergences, convergences, intersections. *Organised Sound*, 17(02), pp.103–113.
Fiebrink, R., and Cook, P.R., (2010) The Wekinator: a system for real-time, interactive machine learning in music. In Proceedings of The Eleventh International Society for Music Information Retrieval Conference (ISMIR 2010)(Utrecht)
Grierson, M. (2005). *Audiovisual composition*. Canterbury: University of Kent at Canterbury, *Thesis.*.
Grierson, M, Yee-King, M and Gillies, M, (2013), *Creative Programming for Digital Media and Mobile Apps*, Coursera, accessed 1/10/2016
https://www.coursera.org/learn/digitalmedia
Ikeshiro, R. (2013). Studio Composition: Live Audiovisualisation Using Emergent Generative Systems. Doctoral thesis, Goldsmiths, University of London, Thesis

[28] From HEFCE's Catalyst Fund Call A for Small-scale, 'experimental' innovations in learning and teaching http://www.hefce.ac.uk/lt/innovationfund/

Itten, J. (1975) *Design and Form: The Basic Course at the Bauhaus and Later*, New York : Van Nostrand Reinhold.
Le Grice, M. (1981). *Abstract film and beyond*. Cambridge, MA: MIT Press.
Lund, C. and Lund, H. (2009). *Audio.Visual: On Visual Music and Related Media*. Stuttgart: Arnoldsche Art Publishers.
McCarthy, J (1960), "Recursive functions of symbolic expressions and their computation by machine". Communications of the ACM. April 1960.
Mcdonnell, M. (2010). Visual Music - a composition of the 'things themselves. In *Sounding Out 5*. Bournemouth University.
McLean, A. (2004). Hacking Perl in Nightclubs, perl.com, accessed 1/10/2016 < http://www.perl.com/pub/2004/08/31/livecode.html>
Moritz, W. (1986). *Towards an Aesthetics of Visual Music. Center for Visual Music.* Available at http://www.centerforvisualmusic.org/TAVM.htm [Accessed March 24, 2017].
Moritz, W. (1997). *The Dream of Color Music, And Machines That Made It Possible*. *Animation World Magazine.*
Oram, D. (1972). *An individual Note: of Music, Sound and Electronics*, London: Galliard
Lopes, D. (2009). *A Philosophy of Computer Art*. London: Routledge
Rees, A.L. (2013). *A history of experimental film and video: from the canonical avant-garde to contemporary British practice*. Basingstoke, Hampshire: Palgrave Macmillan.
Richardson, J., Gorbman, C. and Vernallis, C. (2013). *The Oxford handbook of new audiovisual aesthetics*. Oxford: Oxford University Press.
Sitney, P.A. (2002). *Visionary film: the American avant-garde 1943-2000*. Oxford: Oxford University Press.
Sito, T (2013). *Moving Innovation: A History of Computer Animation*. Cambridge: MIT Press
Wang, G. (2008). The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality. Thesis, Princeton University.
Whitney, J. (1980) *Digital Harmony: On the Complementarity of Music and Visual Art*, McGraw-Hill
Yee-King, M., Grierson, M., & d'Inverno, M., (2017) STEAM WORKS: Student coders experiment more and experimenters gain higher grades, IEEE International Engineering Education Conference, Athens 2017
Zbyszyński, M, Yee-King, M and Grierson, M, (2017), Rapid Prototyping of New Instruments with CodeCircle, Proceedings of the New Interfaces for Musical Expression Conference, Copenhagen.