Live Coding YouTube: Organizing Streaming Media for an Audiovisual Performance

Sang Won Lee
Computer Science and Engineering
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
snaglee@umich.edu

Jungho Bang
Electrical Engineering & Computer Science
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
bjungho@umich.edu

Georg Essl
College of Letters & Science
University of Wisconsin - Milwaukee
2442 E Hartford Ave.
Milwaukee, WI 53211
essl@uwm.edu

ABSTRACT

Music listening has changed greatly with the emergence of music streaming services, such as Spotify and YouTube. In this paper, we discuss an artistic practice that organizes streaming videos to perform a real-time improvisation via live coding. A live coder uses any available video from YouTube, a video streaming service, as source material to perform an improvised audiovisual piece. The challenge is to manipulate the emerging media that are streamed from a cloud server. The musical gesture can be limited due to the constrained functionalities of the YouTube API. However, the potential sonic and visual space that a musician can explore is practically infinite. The practice embraces the juxtaposition of manipulating emerging media in old-fashioned ways similar to experimental musicians in the 60's physically manipulating tape loops or scratching vinyl records on a phonograph while exploring the expressiveness enabled by the gigantic repository of all kinds of videos. In this paper, we discuss the challenges of using streaming videos from the platform as musical materials in live music performance and introduce a live coding environment that we developed for real-time improvisation.

Author Keywords

live coding, YouTube, streaming video, bricolage H.5.5 [Information Interfaces and Presentation] Sound and Music Computing

1. INTRODUCTION

How we listen to music has changed over a long period, with the development of recording technology and its playback media. Before the time of recording technology, listeners could enjoy music only by going to a musical performance. Since we were able to record music performances, new types of recording media have been invented, from vinyl records to compact discs and mp3. Nowadays, most people do not carry "physical" recording media for music listening. Instead, a piece of music can be streamed from a cloud server on demand, which changes the ways in which musicians are compensated. Whether we like it or not, the streaming services now dominates the current music industry. For example, the U.S. music industry made more money with streaming than physical media or digital downloads in 2015 [2]. Even a NIME music submission does not accept non-streaming media (not even electronic files like mp3) anymore. Instead, it asks participants to post documentation links from "streaming-only services such as SoundCloud, Vimeo, and YouTube".

The music recording media inspired pioneers to compose music in different ways, such as by cutting and pasting magnetic tapes, scratching vinyl records, making dents in compact discs and chopping a piece of music into audio sample files for DJing. In this paper, we begin questioning what kinds of new opportunities and challenges we have with the today's streaming media for new musical expression. We choose YouTube, a commodity video streaming service, to implement a real-time performance system. This idea is in principle not limited to YouTube. Through their application program interface (API), other streaming services could be utilized in a comparable fashion. However, YouTube is a useful example case as it allows to demonstrate the scale and effect of this kind of performance. YouTube is a grand repository of videos, including a large number of music and non-music videos. The use of YouTube videos offers musicians the ability to retrieve any audiovisual samples on the fly by searching keywords and by providing time offsets to play them from. The participatory nature of YouTube as a platform where anybody can easily generate content cultivates the potential of an artistic practice that can be accessible to both listeners and other musicians.

The idea of improvising on YouTube videos poses the question of exploring methods and challenges in musically "organizing" the vast amount of streaming media live. In this paper, we developed a performance system for a musician to live-code YouTube videos in a web browser. We review related works, discuss the challenges and opportunities of using streaming media, relate this work to previous works in live coding, and address the technical challenges and musical gestures in developing and performing a piece.

2. RELATED WORKS

2.1 Musical Experiments with Recording

The history of electronic music parallels artists' experiments using recording media in composition. John Cage's assembly of *the Williams Mix* exemplifies the use of recording media in composition, cutting segments of magnetic tapes on a piece of paper and splicing them into eight tapes that are played simultaneously as a piece [7]. In, *Studie I and II*, using additive synthesis, Stockhausen accumulates tapes on top of each other or plays tapes of different tones in rapid succession to create polyphonic structures [32, 33]. Recording non-musical sound (natural or industrial) led to a new compositional practice that uses fixed media, such as musique concrète [29] and SoundScape [30].

The use of recording media has been expanded to the live performance practice. In Riley's live performances of *The Gift*, Baker and his quartet played live to the tapes to create a tape delay sequence over which the live performers also improvised [16]. Turntablism, whose precursor is early ex-

NIME'17, May 15-19, 2017, Aalborg University Copenhagen, Denmark.

perimental music, is a well-known performance practice in which a musician performs by physically manipulating vinyl records, turntables (typically two), stylus, and mixer [14]. Yasunao Tone took a destructive approach in using recording media where he damaged audio CDs and hacked the CD player to create noise music [36]. Granular synthesis is a modern compositional technique in computer music, which manipulates sound samples temporally [40]. It has frequently been used in live performance [37], leading to dedicated interfaces for real-time granular synthesis in NIME contexts [8, 25, 26]. The use of video samples in composition enables unique audiovisual performances that often result in video mash-up not only in VJing cultures [12] but also in context of NIME. For example, 'sCrAmBlEd?HaCkZ!' takes audiovisual samples from popluar videos and remix them based on the similarity to live input [18]. MadPad realizes the grid of video samples that mimics an MPC controller on a tablet [19].

With the emergence of streamed forms of audiovisual samples, we started to witness interactive music systems that utilize streamed audiovisual samples. MusicMapper realizes a 2D representation of chopped audio samples streamed from SoundCloud for remixing music [6]. TextAlive Online generates a music video using kinetic typography by analyzing audio from music streaming services [17]. Gibber, a web-based live coding environment, implements a function called "`freesound`" that lets a musician search an audio sample from the freesound website and play a sequence of the sample [27]. This paper was directly inspired by a web-based live coding performance, *The Last Cloud*, which manipulates various media on the web (audio, video, and multiple web pages) and the grid of videos [35]. It is apparent that recording media in its various manifestations have inspired musicians to create new forms of composition and performance practices. Hence it is important to note that while the list of creative works in this category is vast, our review here can by no means be complete.

While music streaming services dominate how people listen to music, it is not clear if it helped us create a new art practice. This paper is an instance in response to the anticipation of novel performance practices using streaming media.

2.2 Why Live Code?

How do we perform with streaming media? While audiovisual performances can be performed live with various kinds of interfaces, from a MIDI controller to gesture recognitions, we choose live coding to organize YouTube videos musically. The reason is simple– it is the most basic way to control the large scale video streaming. Live coding is a highly interactive programming practice where a programmer writes a program that is currently running and the outcome of which is typically electronic music [10]. Typically, live coding has been known as a practice that generated sounds through algorithmic means that were developed by interactively writing the code on stage. Hence the product of live coding is typically sound synthesis and temporal algorithms that controls the generated sound. However, the product of live coding in music performance can be expanded in various ways. For example, the outcome can be real-time notation [23], musical instruments [5, 22], audience participation [21] and visualization [11]. Furthermore, live coding has been expanded to non-musical domain: choreography [31], textile patterns [9], scientific simulation [34], and programming education [28]. Live coding allows a musician to control audio-visual-temporal dimensions of YouTube videos with a degree of automation. In addition, the canonical rule of live coding to reveal the screen and show the code to the audience will help them understand the algorithms of musical gesture by comprehensible naming convention, such as `loop, jump,` and `speed`.

3. OPPORTUNITIES AND CHALLENGES

Before we introduce the performance system, we would like to review the opportunities and challenges that this media poses to musicians.

3.1 Infinite Audiovisual Samples

One of the biggest benefits of including streaming media in interactive applications is that it provides a grand repository of audiovisual files. The repository will not only have music, but also, various kind of contents depending on the characteristics of the service. For example, YouTube videos can be music, live concerts, tutorials, commercials, movies or possibly any video that users create. Therefore, the process of exploring and selecting video samples that will be used in a piece is a significant part of the composition. Musicians can also create materials by themselves and make them readily available in certain cases asynchronously (posting a video) or synchronously (e.g., YouTube Live). Potentially, audience members can contribute to provide audiovisual samples during (or prior to) the concert; a similar approach has been attempted under the context of mobile phone orchestra [24]. Musicians do not need to "own" sonic materials on their local machine but only has to draw on samples from the services. Not only this will reduce the size of the application but vastly expands the material a musician can improvise with on the fly. Similarly, the advantages and challenges in streaming computer music applications from a cloud server have been examined in [15].

3.2 Platform Dependent Practice

Many of the challenges in using streaming recording originate from the fact that the application depends on a specific platform. Typically, commercial services, such as Spotify, SoundCloud, Vimeo, or YouTube, are available options for musicians. The challenges of using such services are left to the musicians and are listed in the following subsections.

3.2.1 Limited Control of Streaming Services

The limited functionality in manipulating streaming media musically is a technical challenge for musicians to perform with those snippets. Typically, such streaming services provide an application program interface (API) to allow access to the media via code. The API is, however, not designed for artistic practices. The musicians' challenge is then to re-purpose the API, to develop the range of musical expression, and to compose (or improvise) music. In the later section of this paper, we take YouTube as an instance and explore what kinds of features are available for musicians.

3.2.2 Terms and Conditions

There are legal issues that arise from the fact that a musician does not "own" the content. As they do not own audiovisual snippets and the platform, musicians need to agree to terms and conditions of the platform. While copyright infringement can be the aesthetic of the piece as in [18], these terms may prohibit certain artistically desirable uses of the media. We use YouTube as an example to describe the challenge. Based on its terms [4], we cannot distribute any part of the service or the content without authorization, except the means that they provide (embeddable player) - (4A). For example, downloading selected videos automatically and manipulating them in non-YouTube players is clearly a violation of the terms if it were for the public performance. Another term indicates that we cannot "modify"

any part of the service (contents, the API functions of the player) - (4B). Fortunately, the use of streaming media in the performance practice that this paper suggests seems to be within the range of the "fair use" , such as remixing as a part of new expression [1]. We find performing the piece in public is fine according to 4E of their terms ("showing YouTube videos through the Embeddable Player on an ad-enabled blog or website") [4]. Many of these problems are uncertain and vague and it is interesting for us that our performance can create gray areas they may tighten or loosen their terms or pose philosophical questions to the communities. For example, performing a piece on a web browser in public is legal but distributing the performance video via YouTube seems problematic (4A). Another interesting term that we can potentially violate is that 4H. - "You agree not to use or launch any automated system,(...) that accesses the Service in a manner that sends more request messages (...) than a human can reasonably produce...". Is the use of Is live coding an automated system or a human performance? Is a live coder (or a musician with a musical instrument) human or an automated system? How many requests can a human reasonably produce via live coding their API functions? Another interesting fact is that not all the countries allows streaming services. For example, according to [38], YouTube is blocked in China, Iran and North Korea. We currently submitted a performance proposal to a conference in one of these countries and will document the whole process if the proposal gets accepted.

While these challenges may hinder artists who musically manipulate streaming media, such constraints challenge them to find ways to be expressive, given the technical and legal boundaries. However, this is not different from any other kind of computer music practices, which re-purpose existing objects physically or electronically for music making. In fact, musicians have used a number of commodity hardware and software (e.g., turntables, tape recorders, programming languages, laptops, microcontroller kits, and mobile phones) in the musical context, which later leads to artworks, scholarly research, and commercial products. It has been studied that such constraints can yield a creative use of technology and the development of new musical styles [13]. Therefore, we rather see this as a different set of challenges in expanding musical expression with this new media.

4. DESIGN AND IMPLEMENTATION

To explore the musical aesthetic enabled with the idea of live coding YouTube videos, we present a performance system in which a live coding musician can retrieve YouTube videos and organize them musically. The performance system has been used by the authors to perform a structured improvisation piece that demonstrates the motivation of the performance practice. This is built in a web browser and is publicly available online through the following URL:`https://livecodingyoutube.github.io`[1].

4.1 YouTube iframe API

YouTube provides a Javascript API for the third party developers to embed and control a YouTube player in a web page [2]. A YouTube player is embedded as an independent web page, using `<iframe>` tag. The provided API offers various functions for programmers to control the embedded player. However, the API functions are primitive and limited to simple manipulation of the videos.

[1]the code is available at `https://github.com/livecodingyoutube/livecodingyoutube.github.io`

[2]For more detail, refer `https://developers.google.com/youtube/iframe_api_reference`

Figure 1: Live Coding YouTube videos. 3X4 grid, a live coding editor (translucent) and the search result (right)

Largely, there are three types of API functions that we use: playback controls (`playVideo`, `pauseVideo`, `cueVideoById`), temporal controls (`seekTo`, `setPlaybackRate`), and volume controls (`mute`, `unMute`, `setVolume`). Not only the number of functions available is limited, but also their capabilities are restricted. For example, `setPlaybackRate` method takes a parameter to change the playback speed, and the parameter can only be predefined numbers (0.25, 0.5, 1, 1.25, 1.5, and 2); otherwise, it will round down to the nearest supported value.

4.2 Grid System and Live Coding Editor

The output of the performance system is audiovisual artifacts generated by a set of YouTube videos being played by a live coder, and the input to the system is a performer's code written live. The visual outcome and the live coding environment needs to be projected in a concert space to make the idea of using YouTube videos explicit to the audience. Therefore, a musician is in need of quickly loading videos as well as revealing the music-making process (live coding). To that end, we implemented a grid system that can automatically load videos, maintain the layout (HTML/CSS) accordingly, and simplify accessing the videos by indexing them (See Fig. 1). Without the grid system, a musician would have needed to deal with low-level code, adding, replacing, and deleting YouTube videos. Instead, musicians can use a set of helper functions that can add videos in a line and they can focus on the high-level control of videos to algorithmically organize them. The grid system provides a set of helper functions to modify the grid layout on the fly and to load multiple videos quickly and to create mash-up. Allowing multiple videos in the grid is analogous to a digital audio workstation that supports multi-tracks.

As mentioned before, the live coding editor needs to be shown to the audience, and doing so is particularly challenging, given that the grid system is the primary visual outcome, which is directly linked to the music produced by the mash-up of the YouTube videos. We designed the system to have a translucent code editor laid on top of the grid system. For the readability of the live coder, the line on which the cursor is placed is highlighted. To execute code, the live coder needs to select a block of code and press `Shift+Enter`.

In order to support efficient video search, the system provides `search` function to get the list of video thumbnails returned by the service for keywords, and the result of the query is displayed on the right side of the editor (Fig. 1). A live coder can click a thumbnail of a video and the unique identifier of the video will be pasted where the cursor is for further use in coding. This ensures the performer to retrieve

any video on the fly by keywords and possibly improvise on the selection of videos.

4.3 Latency and Buffering

Per each YouTube video, there are two dimensions the musician can control via YouTube APIs: time and dynamics (volume). The most significant difference from playing a sample audio (or video) file and a YouTube video, or streaming media in general, is that it takes time to play a video from the time that `playVideo` is called due to the network transmission. The major source of the latency is the time between making a request to a cloud server and receiving streamed data in a local machine. The latency is dependent on numerous factors: bandwidth, connection speed, and the number of connections that share resources at the moment. Fortunately, to improve the user experience of streaming, all kinds of streaming services use **buffering**. Video buffering temporarily stores data in the local memory so that a momentary insufficiency of data will not stop the playback as long as the video is buffered enough.

When `playVideo` is called for a YouTube player, the YouTube video does not start the video immediately but spends some time to secure the buffered data enough to safeguard the video from starving. While this is desirable for casual use of YouTube to have a smooth playback at the price of waiting for a few seconds in the beginning; this is problematic when a musician wants to play videos instantly or schedule one or more videos to begin playback at a specific moment due to the latency and its jitters. This latency from the buffering step can be avoided by loading and buffering a video prior to it being actually played. To initiate the buffering step of a YouTube video, when a video is added to the grid, the system automatically "plays" the video so that it immediately moves from the `unstarted` state to the `buffering` state (Fig. 2). Immediately, the video is paused, so that it will not have any audio output. Now, the video can be cached locally and can be immediately played without buffering delay (Fig. 2-c).

As the commands such as `playVideo`, `pauseVideo`, `seekTo`, and `setVolume` run in the main execution thread, the executions of such commands can be deferred if the main thread is temporarily stalled for other tasks, such as changing the layout or receiving data from a socket. This creates a delay between the `buffering` state and the `paused` state and can have unwanted audiovisual outcomes. To avoid artifacts, the system `mute`s the video and `unMute`s the video later (Fig. 2-a,b).

Lastly, even when the video is buffered locally and paused (at the state of 2-b), there can be a short delay between the time a user executes `playVideo` and the time when the video actually begins. We depicted this delay as "PLAY DELAY" in the Fig. 2-2. This delay adds to an error if a musician wants to execute a fine temporal control, like looping, synchronizing, or phasing.We find that this delay is dependent on the performance of the main thread, as well as the video quality of the player (resolution of the video). It is recommended to sacrifice the video quality ('tiny' - 144px in YouTube setting) for precise temporal controls especially when there are many videos on one page.

4.4 Helper Functions

The live coding environment provides helper functions that allow the musician to quickly build the grid and execute interesting musical gestures. All the helper functions are listed in the Table 1. The capability of helper functions is defined by the limited functionalities of the YouTube API. However, the advantages of using helper function are as follows: 1) It takes care of the style and appearance of the grid when adding/replacing videos in a few lines of code. 2) A live coder can select videos by index and control them in batch. These helper functions serve as the basic building blocks of the live coding environment and the rest of the functionalities are left to a live coder's performance with the expressivity of the programming language (javascript).

The first argument of most helper functions is the index of YouTube videos in the grid. The system provides multiple ways to quickly select videos and control in a batch The following example shows how a live coder can select videos in various ways, given a 2 by 2 grid (indexed from 0 to 3).

```
create(2,2,''xyz'') //create a 2-by-2 grid
play(all) //play all videos (0,1,2,3)
seek(1,30) //seek to 30 seconds in video 1
fadeOut([1,3]) //fade out video 1 and 3
speed(not(1),2) //play in 2X speed for 0,2,3
jump('i%2==1',5) //jump 5 seconds for 1 and 3
```

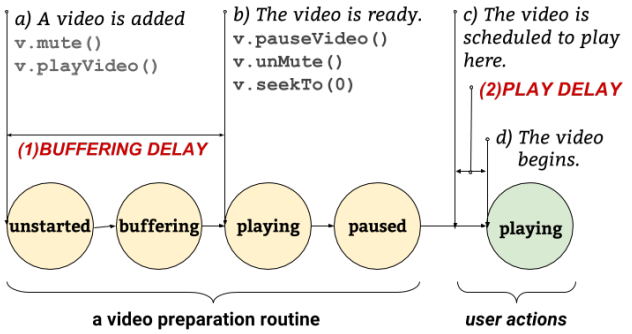

Figure 2: Video preparation routine and its state transition: To have all the videos loaded in the state that can be immediately played, each video is initially played automatically and paused in order to skip the buffering delay when played by a musician.

Table 1: Helper Functions

Category	Function	Description
Search	`search`	search string and retrieve search results from YouTube.
Grid	`create`	create a YouTube grid
	`add`	add row(s) or col(s) to the existing grid.
Playback	`play`	play the video(s)
	`pause`	pause the video(s)
	`cue`	replaces the video(s) with new video (id)
Temporal	`seek`	seeks to a specified time in the video(s)
	`jump`	jump n seconds in the video(s)
	`loop`	loop the video(s) with interval t
	`loopAt`	loop the video(s) at specified time with interval t
	`sync`	synchronize videos to the specified time or to the specified video
	`delay`	add specified delay between the video(s)
	`speed`	change the playback speed of the video(s)
Dynamics	`volume`	set volume to a specified level of the video(s)
	`turnup`	increase volume of the video(s) by the specified level
	`mute`	mute/unmute the volume of the video(s)
	`fadeIn/Out`	fade in (or out) of the video(s)

5. MUSICAL EXPRESSION WITH YOUTUBE

The basic musical gesture supported by live coding YouTube video finds parallels in early experimental music that manipulates magnetic tape and vinyl records. In particular, we

borrow musical gestures from minimalist composers including Terry Riley, Steve Reich, and Alvin Lucier, all of whom wrote important pieces in computer music with tape loops. In our premiere performance, we demonstrated these ideas. The piece embraces the juxtaposition of the compositional ideas from the 60s and the emerging media live coded programmatically. The screen recording of the performance is available at the following URL: `https://livecodingyoutube.github.io/pat2017.html` and the video is archived in [20].We discuss two important aspects in performing a Live Coding YouTube piece: 1) selecting audiovisual materials and 2) developing algorithms that can execute musical gestures.

5.1 Selecting Videos

For a particular performance, a significant portion of sonic outcome depends on the selection process of videos that will be used. As the API functions can only control dynamics, the playback offset, and its speed, whether the selection process is improvised or composed, the style of music can vary drastically based on not only how they are played simultaneously, but also the selected videos. The virtuosity of the performance systems remains in this space of choosing a set of YouTube videos. Here, we illustrate some ideas that we have presented in the performance and relate them with the existing experimental compositions.

- One can select non-musical videos and manipulate them temporally to create musical gestures, for example phasing as in Steve Reich's *Come Out.*
- A musician improvises by searching keywords on the fly or playing live-streaming channels, and leave the musical outcome to chance, like John Cage's *Imaginary Landscape No. 4.*
- A musician can perform the piece, videotape, and live-stream the performance to YouTube Live, search the live stream video, and play the video in the grid. The audiovisual artifacts captured in the video recorder is then fed back videos being played on the projection and creates the feedback loop with the delay created arbitrarily by YouTube, which is analogous to Alvin Lucier's *I Am Sitting in a Room.*

5.2 Live Coding Musical Gestures

As mentioned before, musicians are given a limited set of vocabulary by the API. However, live coding enables algorithmic and conversational interaction between the YouTube grid and the live coder. In this section, we introduce a set of musical gestures that a live coder accomplishes in the proposed performance.

5.2.1 Multitrack Mashup

As multiple YouTube videos can be played simultaneously within a few seconds, an audiovisual mashup becomes a powerful musical gesture for a musician. This is not necessarily new and there exist web applications wherein a user can mashup multiple YouTube videos by filling a form with video IDs and the starting time of each one [3]. However, the live coding system is different in a way that a musician can replace videos and control the playback on the fly. Theoretically, the grid system is designed to support any number of YouTube videos. However, we find that the grid greater than and equal to 6X6 makes the machine slow down and makes typing not responsive. Using the fadeIn/Out and replace the videos (using `cue`), one can smoothly make a transition from one video to another.

5.2.2 Loop

Using the `loop` and `loopAt` function, one can loop the whole (or a part of) a YouTube video and specify the interval and a starting point of the loop. The stacks of audiovisual loops can be accumulated gradually to create repetitive rhythms and to build up the piece. The visual outcome of looped music video creates an interesting visual collage of YouTube videos. Maintaining a precise metronome for synchronized looping of multiple videos requires the combined use of Web Audio API and the javascript timeout systems [39]. As all helper functions allow a performer to control videos individually or in batch, the video loops of different YouTube videos can be concatenated and played in an alternating fashion. Setting the interval to relatively short time-frames ($<$ 1s) produces rapid rhythmic patterns. A loop interval cannot be small enough to accomplish granular synthesis but it can support close to the tens of milliseconds when there is only one video with the lowest resolution (144 pixel). However, this is still far from the granular synthesis given the PLAY DELAY in Fig. 2-2. The lower bound increases as the grid has more videos and/or the video quality gets better. Therefore, the performer needs to understand the relationship between the micro-loops, video quality, and potential silence when the loop is too short to be covered in the main thread.

5.2.3 Delay, Reverb and Phasing

When the grid has multiple instances of one video in sync, a musician can create a delay effect by adding delays between instances. The `delay` function can be used to add this delay in the series of videos. If the delay is small enough (e.g. 1ms), it has the same effect of adding reverb to the sound. Lastly, combining delay and loop, one can create phasing effects. Technically, this is not phasing as we do not have precise control over the playback speed. However, when the two (or more) videos have a small difference in the loop intervals, it creates the phase effects where one of the samples slowly plays out of phase.

5.3 Performance Setup

The sound setup of this performance is relatively simple (stereo out from a laptop). More importantly, reliable online connection is necessary for this performance. The connection may not necessarily be high-speed, thanks to the buffering mechanism and we were able to successfully rehearse the piece 10 Mbps (download). The wireless connectivity can vary potentially with the audience's smartphones, which are not present at the rehearsals. Hence it is recommended to use a wired connection to secure online connectivity. Live streaming the performance to create feedback also requires reliable connection with minimum 2 Mbps uploading speed for 360p. As playing multiple YouTube videos simultaneously involves heavy computation and significant bandwidth usage, using a secondary projector (with screen recording for the archival purpose) will increase the computational load of the computer. As the screen projecting is essential to the performance idea, using a projector needs to be tested throughout the composition and the rehearsal process to run the piece in realistic settings Lastly, playing YouTube videos may involve random commercial banners within videos, which can be turned off with the third-party plug-in.

6. CONCLUSION

In this paper, we discussed the opportunities of using streaming media in computer music performance and introduced one instance of such opportunities to demonstrate the potential of musical aesthetics that are available to us. In this paper, we focused on the preliminary expressivity that streaming media afford by the realization of musical gestures inspired from the '60s experimental music. However,

we do believe that this basic concept can be expanded further. In particular, we are interested in liberating videos from the grid, which will allow more free placements of YouTube videos and realizing advanced visual effects, such as cross-fading, cut-out, dissolving, which may enable 'live-coded films'. In addition, we are interested in diversifying the system by implementing a custom interface to control YouTube videos and integrate them into existing performance practice such as DJing, audience participation, and collaborative improvisation with instrumental musicians. Finally, content identification on the video could allow additional levels of control and manipulation, such as automated beat synchronization, onset synchronization, or color matching.

7. REFERENCES

[1] Fair use of copyrighted materials. `https://www.youtube.com/yt/copyright/fair-use.html`. Accessed: 2017-01.

[2] News and notes on 2015 riaa shipment and revenue statistics. accessed: 2017-01. `http://www.riaa.com/wp-content/uploads/2016/03/RIAA-2015-Year-End-shipments-memo.pdf`.

[3] Youtube multiplier. accessed: 2017-01. `http://www.youtubemultiplier.com/`.

[4] Youtube,terms of service. accessed: 2017-01. `https://www.youtube.com/static?template=terms`, 2010-06.

[5] M. Baalman. Gewording, 2014. Music Performance, the International Conference on New Interfaces for Musical Expression.

[6] E. Benjamin and J. Altosaar. Musicmapper: Interactive 2D representations of music samples for in-browser remixing and exploration. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Baton Rouge, Louisiana, 2015.

[7] J. Cage. Williams mix. 1953.

[8] C. Carlson and G. Wang. Borderlands -an audiovisual interface for granular synthesis. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Ann Arbor, Michigan, 2012.

[9] E. Cocker. Live Coding / Weaving - Penelopean Mêtis and the Weaver-Coder's Kairos. In *Proceedings of the International Conference on Live Coding*, 2015.

[10] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. *Organised Sound*, 8(03):321–330, 2003.

[11] D. Della Casa and G. John. Livecodelab 2.0 and its language livecodelang. In *Proceedings of the ACM SIGPLAN international workshop on Functional art, music, modeling & design*, 2014.

[12] M. Faulkner. *VJ: Audio-Visual Art and VJ Culture: Includes DVD*. Laurence King Publishing, 2006.

[13] M. Gurevich, P. Stapleton, and A. Marquez-Borbon. Style and constraint in electronic musical instruments. In *Proceedings of New Interfaces for Musical Expression*, pages 106–111, 2010.

[14] K. F. Hansen. Turntable music. *Musikklidenskapelig Årbok*, 2000:145–160, 2000.

[15] A. Hindle. Cloudorch: A portable soundcard in the cloud. In *Proceedings of New Interfaces for Musical Expression*, London, United Kingdom, 2014.

[16] T. Holmes. *Electronic and experimental music: technology, music, and culture*. Routledge, 2012.

[17] J. Kato, T. Nakano, and M. Goto. TextAlive Online: Live Programming of Kinetic Typography Videos with Online Music. In *Proceedings of the International Conference on Live Coding*, pages 199–205, Leeds, UK, July 2015. ICSRiM, University of Leeds.

[18] S. König. sCrAmBlEd? HaCkZ!, 2006.

[19] N. Kruge and G. Wang. Madpad : A crowdsourcing system for audiovisual sampling. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Oslo, Norway, 2011.

[20] S. W. Lee, J. Bang, and G. Essl. *Live Coding YouTube - PAT showcase 2017 (Screen Recording)*. Zenodo, Apr 2017.

[21] S. W. Lee, A. D. de Carvalho Junior, and G. Essl. Crowd in c[loud]: Audience participation music with online dating metaphor using cloud service. 2016.

[22] S. W. Lee and G. Essl. Live coding the mobile music instrument. In *Proceedings of New Interfaces for Musical Expression*, Daejeon, South Korea, 2013.

[23] S. W. Lee and J. Freeman. Real-time music notation in mixed laptop–acoustic ensembles. *Computer Music Journal*, 37(4):24–36, 2013.

[24] J. Oh and G. Wang. Audience-participation techniques based on social mobile computing. In *International Computer Music Conference, At Huddersfield, UK*, 2011.

[25] S. O'Modhrain and G. Essl. Pebblebox and crumblebag: Tactile interfaces for granular synthesis. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 74–79, Hamamatsu, Japan, 2004.

[26] T. T. Opie. *Creation of a real-time granular synthesis instrument for live performance*. PhD thesis, Queensland University of Technology, 2003.

[27] C. Roberts and J. Kuchera-Morin. Gibber: Live coding audio in the browser. In *Proceedings of the International Computer Music Conference (ICMC)*, Ljubljana, Slovenia, 2012.

[28] M. J. Rubin. The effectiveness of live-coding to teach introductory programming. In *Proceeding of the ACM Symposium on Computer science education*, 2013.

[29] P. Schaeffer, F. B. Mâche, M. Philippot, F. Bayle, L. Ferrari, I. Malec, and B. Parmegiani. *La musique concrète*. Presses universitaires de France, 1967.

[30] R. M. Schafer. *The soundscape: Our sonic environment and the tuning of the world*. Inner Traditions/Bear & Co, 1993.

[31] K. Sicchio. Hacking choreography: Dance and live coding. *Computer Music Journal*, 38(1):31–39, 2014.

[32] K. Stockhausen. Studie I. 1953.

[33] K. Stockhausen. Studie II. 1954.

[34] B. Swift, A. Sorensen, H. Gardner, P. Davis, and V. K. Decyk. Live programming in scientific simulation. *Supercomputing frontiers and innovations*, 2(4):4–15, 2016.

[35] B. Taylor. The Last Cloud. Composition, 2016.

[36] Y. Tone. *Solo for wounded CD*. Tzadik, 1997.

[37] B. Truax. Real-time granular synthesis with a digital signal processor. *Computer Music Journal*, 12(2):14–26, 1988.

[38] Wikipedia. Censorship of youtube. Accessed 2017-04.

[39] C. Wilson. A tale of two clocks - scheduling web audio with precision. `https://www.html5rocks.com/en/tutorials/audio/scheduling/`, 2013-01. Accessed: 2017-01.

[40] I. Xenakis. *Formalized music: thought and mathematics in composition*. Number 6. Pendragon Press, 1992.