AUTOMATED GRAPHICS FOR CONTEMPORARY PERFORMANCE

Alexander M. Pratt [*]
Department of Computer Science
Brown University
`alexander_pratt@brown.edu`

ABSTRACT

Today's most popular musical artists have captivating music that is met with equally impressive displays of visuals. When coupled, concert-goers become involved in a superb audio-visual experience. The cost of these visuals and Visual Disc Jockeys may be expensive. With hope to offer industry standard, out-of-the-box, performance ready visuals at an affordable cost, the Automated Graphics for Contemporary Performance (*AGCP*) project was created. *AGCP* is a signal processing system for automated graphics that is primarily intended for contemporary electronic music performance.

1 INTRODUCTION

After spending a semester in MUSIC 1210 at Brown Univeristy, I wanted to apply my knowledge from the course both in a technical and creative application. I had a few avenues of thought on how I could achieve this goal. Inspired by Fred Again.., I thought it would be creative to leverage the audio and visual processing capabilities of MaxMSP (Puckette (1988); Dobrian (1999)) to create a musical performance using video and audio clips of my friends from home, college, and Boston that I have missed since moving to Providence, Rhode Island for graduate school. I was unable to pursue this option due to a lack of authentic clips of moments that would create a cohesive performance.

While in the mindset of visuals for electronic performance and reminiscing on memories with friends, I remembered seeing artists like Disclosure in Washington D.C., Swedish House Mafia at Ultra Music Festival in Miami, and Don Diablo in Amsterdam for Amsterdam Dance Event. For these performances, music that I enjoyed was met with an impressive display of visuals that mirrored the sublime when coupled. After learning new skills of visual signal processing using Max's VIZZIE extension, I felt confident in my ability to design an audio-visual application. My project was motivated by the question, "How can artists who do not have ample funding for state-of-the-art

[*]Additional forms of contact include my personal email, alexandermarcuspratt@gmail.com, or my website, `alecpratt.github.io`

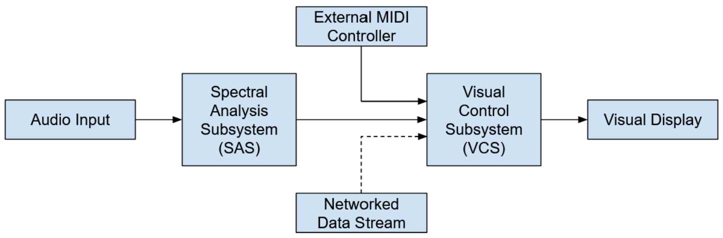

Figure 1: An overview of the Automated Graphics for Computer Performance project system.

Visual Disk Jockey's (VDJ's) have access to industry standard, out-of-the-box performance ready visuals?". Based on this question and my skills in MaxMSP, I designed the Automated Graphics for Contemporary Performance (*AGCP*) Project.

AGCP is a signal processing system for automated graphics that is primarily intended for contemporary electronic music performance. The system performs various spectral analyses on an audio input stream, and uses the output of the analyses to control a graphical display using presets. In entirety, this system enables it's user to play the music they want, while outputting industry-standard, high-quality graphics.

1.1 Course Background

This project contributes to the fulfillment of requirements of MUSC 1210: Real-Time Systems. This is a graduate level seminar course offered at Brown University that focuses on the exploration of audio and visual signal processing techniques using MaxMSP. The culmination of the course is noted by a final project where students leverage their acumen of audio and visual signal processing in either an artistic direction or in development of a signal processing technology - *AGCP* is the latter.

2 Technical Approach

The design of *AGCP* can be broken down into three subsystems: the Spectral Analysis Subsystem (SAS), the User Input Subsystem (UIS), and the Visual Control Subsystem (VCS). An audio stream is the input to *AGCP* and a visual display is the output. Each of the subsystems are discussed below.

2.1 Spectral Analysis Subsystem (SAS)

The input to *SAS* is an audio stream. The subsystem can receive it's input audio stream from any audio driver source available on the computer hosting the application. This flexibility enables audio to be input to *SAS* using an internal audio routing application (e.g. Virtual Audio Cable (Muzychenko (1998)), JACK (Vehmanen et al. (2003)) or an external audio interface. There are two purposes of *SAS*: to detect the current section of music in the audio stream and to detect when certain instruments are performed.

2.1.1 Detection of Music Section

Understanding the current section of music of the audio stream is needed to determine how graphics can be displayed. When *SAS* detects that a song is in it's verse, there should not be any graphics processed or displayed. When *SAS* detects that a song is in it's chorus, there should be graphics processed and displayed. To achieve this goal, *SAS* acts as a toggle between graphics being on and off. When in a verse, *SAS* will notify the entire system that modified graphics cannot be displayed.

$$v_{rms.,avg.} = \sqrt{\frac{v_{rms,1} + v_{rms,2} + ... + v_{rms,n}}{n}},\ n = 64\, samples \tag{1}$$

To determine the section of music in the audio stream, the average root mean square (RMS) of the audio signal over a short duration of time is computed as shown in equation 1 and compared to a static threshold. If the threshold is surpassed, *SAS* interprets that the song is in a chorus and graphics are enabled. Otherwise, graphics are not enabled. This method relies on the generalization that most modern produced songs have louder choruses in comparison to verses. Depending on this generalization meant this method was unreliable, causing both false-positives and false-negatives. Further developments to this method are discussed in the 'Future Work' section.

2.1.2 Instrumentation Detection

The spectral analysis in *SAS* is the most critical component of *AGCP*. The subsystem performs a Short-Time Fourier Transform (STFT) (Gabor (1946)) on the audio stream. This specific STFT uses 2048 bins of 21.5 Hz. The bins from the STFT are then grouped into three frequency ranges intended

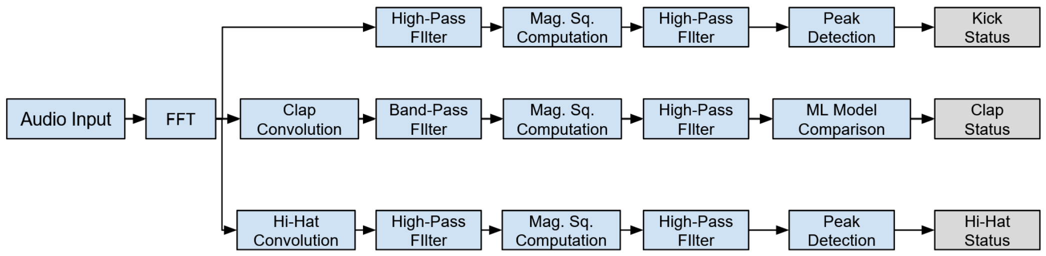

Figure 2: An overview of the signal processing required for each instrument grouping.

to encapsulate select percussive instruments. These frequency groupings represent the frequencies most relevant to a kick, clap, and hi-hat. Each of these groupings are additionally processed to determine if in a certain moment a kick, clap, or hi-hat of the song is currently playing. *SAS* outputs a binary value representing whether each type of percussive instrument is ’on’ or ’off’. This output is sent to the Video Control Subsystem to control graphics processing parameters.

Kick Processing. The kick is processed using the lowest grouping of bins from the STFT. These bins represent frequencies ranging from the sub-bass through the lower-mids. Initially, the audio stream associated with the kick is passed through a high-pass filter (HPF). This filter is intended to remove rhythmic and frequency content from a sub-bass, which is very common in contemporary music production. The presence of a sub-bass can interfere with the peak-detection in this signal processing chain. After the HPF, the input signal is multiplied with itself to better distinguish the amplitude peaks from other audio information in this frequency range. This technique is described in various papers including Maragos et al. (1993) and Bello et al. (2005). Next, another HPF is applied to remove any sub frequencies that were amplified as a result of the signal being squared. At this point, peak detection is applied using a static threshold. If the signal exceeds the threshold, a value of true is sent to enable any kick-related graphics control.

Clap Processing. The clap is processed using the middle grouping of bins from the STFT. These bins represent the middle through upper-middle frequencies. Since claps or snares generally exist in the same frequency bands as the harmonic frequencies of many instruments (Bello & Sandler (2003)), it is important to be able to distinguish between noise and intentional percussive instruments. A Finite Impulse Response (FIR) Convolution occurs between a clap sample and the clap audio stream. This convolution exposes the clap frequencies that may be muddied by harmonics from vocals and instrumentation in this frequency band. After convolution, a similar process that is applied to the kick occurs. The signal is passed through a band-pass filter (BPF) to reduce frequency content to the convolved clap signal. The signal is then squared, and it is passed through another HPF. Unlike kick processing, which uses peak detection, the clap signal uses a Machine Learning (ML) object, `bonk~` (Puckette et al. (1998)), that determines if a clap is being played based on the current frequency response. Initially, peak detection was used, but after adding the ML model, detection became much more accurate. The output of the ML model is sent to enable any clap-related graphics control.

Table 1: Frequency Groupings for Instrument Detection Processing Chains

Grouping	**Frequency Band (Hz)**
Kick	0 - 101.5
Clap	1670 - 2530
Hi-Hat	6162.5 - 7237.5

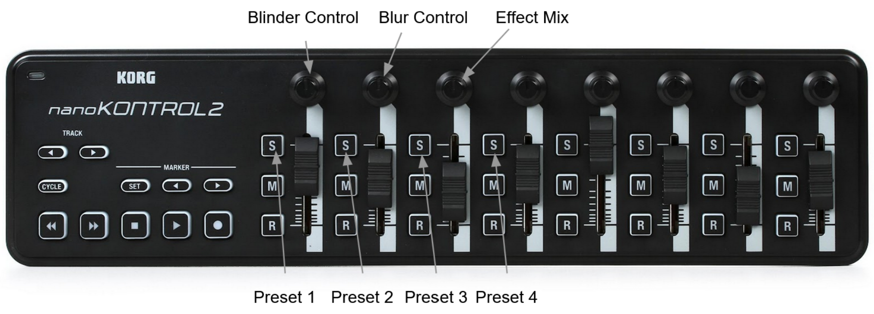

Figure 3: An example use of the KORG nanoKontrol2 MIDI controller for *AGCP*

Hi-Hat Processing. The hi-hat is processed using the upper grouping of bins from the STFT. These bins represent the upper-middle frequencies to the extent of the audible range. This processing chain shares features of the chains used for the kick and clap. Like clap processing, a FIR Convolution is applied with a hi-hat sample to the audio stream prior to any processing. This is done to better detect a hi-hat's presence in comparison to white noise or upper harmonics from other instrumentation in the audio stream. Like kick processing, a HPF is applied, the signal is multiplied with itself, and another HPF is applied to the audio stream. Peak detection is performed on the resulting signal against a set threshold. If the signal exceeds the threshold, a value of true is sent to enable any hi-hat-related graphics control.

2.2 User Interface Subsystem (UIS)

UIS is composed of two components: an external MIDI controller (KORG nanoKontrol2) and a networked data stream (TouchOSC). At least one, but not both, of these components are required for proper functionality. This section overviews how the midi controller and networked data stream can be used within *AGCP*. The 'Future Work' section describes further enhancements to *UIS* that could prove beneficial for future use of *AGCP*.

2.2.1 External MIDI Controller

The external MIDI controller is a hardware option to directly interface with *AGCP* using a KORG nanoKontrol2 MIDI Controller. The radial dial and pads from the device control different aspects of *AGCP*. The pad buttons are used to select predefined presets for visual effects in the Video Control Subsystem. The radial dials are used for controlling additional effects that are not controlled by *SAS* including a blinker effect and blur. Figure 3 contains an example layout that was used for development of *AGCP*.

2.2.2 Networked Data Stream

The networked data stream has the same functionality as the external MIDI controller, but it uses a UDP connection and smartphone application instead of a physical hardware connection to *AGCP*. For data transmission, TouchOSC is used to send OSC (Wright & Freed (1997)) data into the application. The TouchOSC application mirrors the layout and functionality of the external MIDI controller. The MaxMSP application receives the OSC data and applies it to *ACPG*. The difference in design between the KORG nanoKontrol2 and the TouchOSC interface is that a slider is used instead of a radial dial for 'Effect Mix'. This design decision was made to effectively use space on a smartphone screen.

2.3 Visual Control Subsystem (VCS)

VCS takes input from both *SAS* and *UIS* to control the visual effect processing chains. The input from *SAS* controls various parameters within VIZZIE effect modules (e.g. visual delay, blur, kaleidoscope, fold, mirror). The inputs from *UIS* control four different aspects of *VCS*: the visual preset,

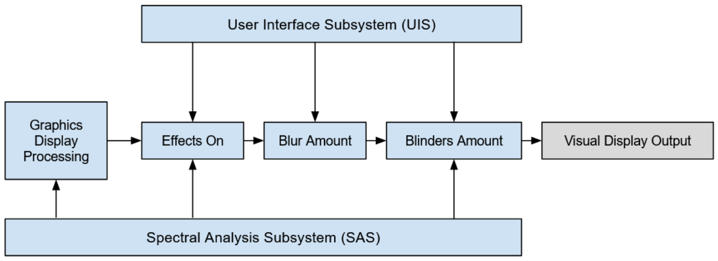
User Interface Subsystem (UIS)
Graphics Display Processing
Effects On
Blur Amount
Blinders Amount
Visual Display Output
Spectral Analysis Subsystem (SAS)

Figure 4: *VCS* and its interface with *SAS* and *UIS*.

effects mix, blinders, and blur. The visual preset is determines the source video and graphics processing that is used. The latter three properties are post-processing techniques that are applied prior to the visual display output.

Visual Presets. At the time of writing, there are four presets that can be used for performance. The preset selection is received from *UIS* to determine the appropriate processing path. Each preset is associated with a source video file. After the selection of a preset, *VCS* selects the correct source video file and switches to the graphics chain associated with the preset. *VCS* then uses the data from *SAS* to determine how to apply the effects.

Effects Mix. *UIS* sends *VCS* a value in range of zero to one hundred percent that represents the effects mix. This value controls the mix of the source video file and the processed graphics that are displayed. This parameter enables a performer to intentionally filter out any graphics that they do not displayed in their performance. Additionally, this parameter provides a performer with an artistic choice of navigating between verse and chorus graphics at their own discretion.

Blinders. *UIS* sends *VCS* a value in range of zero to one hundred percent that represents the blinders mix. Blinders are often used in concert or festival settings to ”blind” the audience. These bright, generally white or grayscale, lights add tension and atmosphere to a performance. When non-zero, the blinders mix enables data representing claps or hi-hats being ”on” from *SAS* to trigger blinders output. A value of zero has no blinders, even if a clap or hi-hat is toggled. A value of one hundred has the screen turn fully white when a clap or hi-hat makes impact.

Blur. *UIS* sends *VCS* a value in range of zero to one hundred percent that represents the blur mix. Blur is a simple effect from the VIZZIE library. It provides a smoothing of the frames to make the visual display look like it is blurred and slowed. This effect can be used in tandem with the effects mix and blinders to introduce the chorus effects in a slowed, blurred way during the build. Using blur hints at the chorus visuals, without fully exposing the intensity. Blur can also be used in the verse to make the base video file have variation or take on new character.

3 Conclusion

AGCP is an audio signal processing program that utilizes spectral analysis and graphics processing to create a performance-ready set of graphics for contemporary music performance. The project is composed of three major subsystems that interact directly with one another: the Spectral Analysis Subsystem (SAS), the User Interface Subsystem (UIS), and the Video Control Subsystem (VCS). The interaction of these subsystems creates a cohesive visual product that can be consumed in music venues across the globe. The project currently works, but is fairly resource intensive. Following recommendations that are listed in this section and ”Future Work” may help with the accuracy of the spectral analysis, heighten visual experiences, and dramatically increase the performance of the system.

3.1 Audio Input Design Analysis

During the development process for *AGCP*, the input audio stream was tested using a `playlist~` object with WAV files. This approach was sufficient for testing, but would not work for a real-time performance. To transition to application to a performance capable system, a design choice had to be made with respect to audio source. The choices were to use an external audio interface or internal routing from an application. The latter option was chosen, which had trade-offs.

Both an external audio interface and internal audio routing send their audio stream as a microphone input into Max. Using an internal routing mechanism (e.g. Virtual Audio Cable, JACK) reduces the physical imprint required for performance by removing the need for an additional cable and audio interface to connect the DJ Controller audio output into the application. Because of the reduced footprint, the internal routing option was chosen as the audio input method.

While reducing the physical footprint is useful, the amount of computing resources that internal audio routing utilizes was not thought of prior to selecting internal audio routing. *AGCP* is resource-intensive. Running *AGCP*, Serato DJ Pro, and an internal router, my ASUS Vivobook 15 with an Intel I7, quad-core processor and 16GB of DDR4 RAM would crash continuously. This makes *AGCP* unreliable for someone who is trying to run graphics in a live performance setting. Separating the audio source from the device running *AGCP* is probably ideal, since it reduces the quantity of resources that are competing for CPU and Memory while performing. After testing the program on an ASUS Vivibook Pro 16X with an Intel I9, 24-core processor with 16GB of DDR5 RAM, the program ran smoothly with no issue.

3.2 User Interface Design Analysis

The user interface design contributes to the performance issues described in the previous section. An issue of the user interface is that both the USB connected KORG nanoKontrol2 and the TouchOSC Program are polled internally by *AGCP*. This requires additional resources that prevent the application from running smoothly. Ideally, the user interface should be a light-weight process that does not interfere with the audio and visual processing.

Moving forward, it may be better to create a modular system that is external from the computer running *AGCP*. One option includes integrating the controls into a DJ controller that sends MIDI information to the device running *AGCP*. I attempted this while doing my development to see if I could reduce the need for an external device for *UIS*. I discovered that the MIDI output is read properly in Max from the DDJ-SR2 Board prior to starting the Serato DJ Pro software. After starting Serato DJ Pro, I could no longer send MIDI data from the DDJ-SR2 Board to Max. This prevented me from using the board as an input to my program. Combining a DJ controller with *AGCP* would be ideal if there is a way to make MIDI data available to other processes on a computer.

Future Work

Convert Framework to C++

MaxMSP is written in C++. This implies that *AGCP* can be translated to C++ and other programming languages. For a programming exercise, for technical purposes, and for more customizability, it may be ideal to pivot the framework of the AGCP Project from MaxMSP to C++.

External Hardware

What may be apparent from some of my discussion through this paper is the need for external hardware. A potential implementation is to create a new standalone DJ Board or external DJ Board attachment that connects to both the DJ Board and the CPU running *AGCP*. If this device has pads for controlling presets or effects, and receives data via USB or Ethernet from a series of CDJs, this data could processed and sent to the computer running *AGCP*. Relevant header information stored in music files pertaining to song structure, effects control, and audio stream would be sent into *AGCP*. Some of this information, such as song structure, completely removes the need to analyze certain aspects of music since *AGCP* will know exactly what section of music the song is currently in. Aside from reducing computation needed for *AGCP*, an external hardware device would relocate a portion of trivial processing to a microcontroller instead of on the CPU running *AGCP*.

Generative Graphics

My background in generative graphics is non-existent. However, I think that coupling *AGCP* with applications of generative graphics could elevate the project. The parameters used to control modifying the source video file could be used to control generative graphics. This would add more dimensionality to the *VCS*. Instead of being dependent on source video files, the audio stream would create the output visuals.

Instrumentation Identification

For this project, I used some machine learning to target clap sounds in the audio stream (Puckette et al. (1998)). I think that the expansion of efficient use of machine learning models could be beneficial to increasing the accuracy in which kicks, claps, hi-hats, and even other elements such as instrumentation are detected in the audio stream. With more accuracy and more instrumentation detection options, the graphics processing aspect of *AGCP* can become more dynamic. For example, an effect such as the blinker could be controlled by a piano instead of a clap or hi-hat being played. Another example could be tracking the melody of the song and utilizing that to effect something in the graphical processing chain depending on the frequency of the notes.

Preset Development

At the moment, there are four presets that are directly related to four source video files. A natural development for this project is to expand video processing capabilities in a few different directions.

I think the number of video processing presets available should be increased. While there are four presets that are performance ready, surely a performer would want to use more than four visual effects for their performance. Expanding the number of performance ready presets would make *AGCP* more attractive to users.

Another pathway for development is to remove the dependency of source video files as the foundation of the visual effects processing. Creating a system where the user can predefine their own video files would be ideal. This removes a dependency upon the sample graphics that I was able to find and this allows the user to be creative with their own video files.

A key development related to presets is user customization in video processing. This includes users being able to tweak current presets and settings, and users creating their own presets and settings. This area of development would most likely be appreciated by performers who wish to create their own effect processing chains instead of being reliant upon a certain set of presets.

Robust MIDI Controller Usage

I discussed the removal of MIDI Controllers in the previous section. If MIDI Controllers are to still be used for this project, I think it is important to make the handling of MIDI controllers more robust. Currently, the software is only capable of using the KORG nanoKontrol2 right out of the box. I think it would be useful to either make a setting where the user selects their MIDI Controller from a drop-down menu of compatible controllers, or the user is given the opportunity to manually select what MIDI signals pair to what controls in a setting window. This would let the user be able to pair any MIDI Controller into the system.

Section of Music Detection

My approach to determining the song structure was fairly useful, but not perfect. This led to incorrect moments of effects appearing outside the chorus and no effects displayed during the chorus. I think that there are two approaches on how to make the detection of the song structure better: use machine learning models or analyze song files prior to performance.

The use of machine learning could be beneficial to detecting the song structure of an audio stream. Training a model on representative genres such as house, dubstep, trap, and garage could give the detection algorithm more insight into the current section of the audio stream. Not only would the RMS of the audio stream be used, but other trained aspects such as frequency response could be used in the trained model to distinguish when an audio stream is in a verse, build, chorus, or breakdown.

An alternative or complement to a machine learning model is to analyze the audio files prior to performance. Additional header data can be used to distinguish the changes of song sections. This could either be completed manually by a user in an external application, or by a trained machine learning model. The header data could be used during a performance to send information to the graphics processor. I think in order to use this concept, the user would most likely need to use an *AGCP*-specific DJ software that would be able to use the header data to effectively communicate with the processor. If an *AGCP*-specific DJ software is not used, then all of the header data would be lost as soon as the audio stream was sent out from its source.

Acknowledgment

Thank you to Dr. Butch Rovan for his help in this project. His instruction in lecture and his feedback for this project has strengthened my understanding of audio signal processing. In addition, I would like to thank the Pennsylvania State University School of Music, who provided me a foundation in Music Technology in my undergraduate education. Notable mentors include: Dr. Mark Ballora, Dr. Sarah Genevieve Burghart Rice, Prof. Curtis Craig, and Dr. Baljinder Sekhon.

References

Juan Pablo Bello and Mark B. Sandler. Phase-based note onset detection for music signals. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.

Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B. Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 2005.

Christopher Dobrian. Programming new realtime dsp possibilities with msp. *COST G-6 Workshop on Digital Audio Effects*, 1999.

Dennis Gabor. Theory of communication. *Journal of the Institution of Electrical Engineers*, 1946.

Petros Maragos, James F. Kaiser, and Thomas F. Quatieri. Energy separation in signal modulations with application to speech analysis. *IEEE Transactions on Signal Processing*, 1993.

Eugene Muzychenko. Virtual audio cable user manual. *https://vac.muzychenko.net/en/manual/main.htm*, 1998.

Miller S. Puckette. The patcher. *International Computer Music Conference*, pp. 420–429, 1988.

Miller S. Puckette, Theodore Apel, and David D. Zicarelli. Real-time audio analysis tools for pd and msp. *International Computer Music Conference*, 1998.

Kai Vehmanen, Andy Wingo, and Paul Davis. Jack design documentation. *https://web.archive.org/web/20080221160709/http://jackit.sourceforge.net/docs/design/*, 2003.

Matthew Wright and Adrian Freed. Open soundcontrol: A new protocol for communicating with sound synthesizers. *International Computer Music Conference*, 1997.