Exploring Audiovisual Linkages

Pehr Hovey
Media Arts & Technology
University of California Santa Barbara
pehr@umail.ucsb.edu

Abstract

It is becoming increasingly common for live music performance to be accompanied by visual projection of some kind. Whether the visuals are at the command of a VJ artist or the result of a static DVD they are often not directly driven by or correlated to the music. This paper covers the concept of *audiovisual linkage* whereby the visual output is driven by the musical content and thus creates the possibility of a more immersive and cohesive overall experience.
This paper presents a brief overview of applicable concepts in audiovisual perception and details recent experiments by the author in implementing live audiovisual synchronization.

Keywords: Live visuals, Audiovisual Synchronization.

1. Introduction

We are fundamentally multimodal beings. As humans, we possess a powerful set of complementary sensory systems that work together to help us understand the characteristics of our surroundings. Throughout human history we have been devising ways to be informed and entertained through the use of multisensory and multimodal events and constructs. Though we are equipped with five primary senses we as a species have so far been concerned almost exclusively with auditory and visual stimuli as our preferred avenues of artistic experience.

The need and desire to use audiovisual works as a form of entertainment predated any feasible means of recording them. Before there were motion pictures and phonographs there were theaters in the round and chamber orchestras. Before that we had tribal chants and dances. Today we have DJs and VJs.

While DJs and live musicians are usually the focal point of a concert or club night it is becoming more common to have visual projection as a performance enhancement. These visuals can range in complexity and connectedness to the music. Some performers and clubs use pre-made DVDs that loop pre-recorded material in unchanging order. Other venues employ Video Jockeys (VJs) to mix visual material on the fly, using a combination of pre-recorded material, live camera input and special effects. While the aesthetic output of modern VJ software is very compelling there can be an issue of discontinuity between the audio and visual domain, summarized by Nick Collins:

"Though a VJ works from some tracking of the audio output of the artists they accompany, there is no guarantee that the artistic agenda of the laptop musician and laptop visual artist will coincide."[1]

In addition to aesthetic/artistic differences there is an issue with temporal synchronization. It can be distracting if the apparent speed of the visuals is much different from the tempo of the music or if there is a definite beat structure in the music and the visuals appear chaotic and unstructured.

A much higher level of connectedness, or *audiovisual linkage*, is achievable with systems that are specially designed to generate the visual elements on-the-fly in response to the music as it is happening. Such systems integrate the qualities and characteristics of the music in some way to ensure that the visual result is somehow correlated to what the audience is hearing.

This paper documents the beginnings of my research and experimentation in creating live visual performance software that can be partially driven by generative music in order to create such tight synchronization. The proceedings are the records of the conference.

2. Related Work

Much research in the union of audio and visual has concerned capturing the general thematic elements of the music in color and form. Many modern computer visualizations take the form of swirling colors and pulsating lights that have their roots in synaethesia. This research looks for natural correspondences between sound and color, based on accounts from people who "see" color in response to sound.

Thomas Wilfred built one of the earliest visualization systems in the early 1920's. He created a "light organ" dubbed the Clavilux [2] that was played similarly to a piano but created swirling patterns of light on the wall. He called his new art form "Light Music" despite the absence of sound.

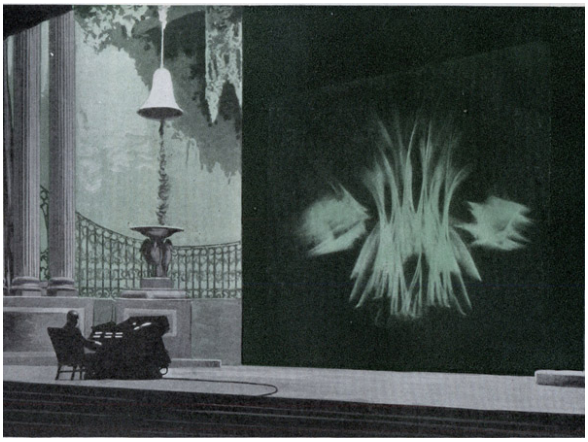

Figure 1. Thomas Wilfred's Clavilux 1924

As computing technologies evolved in the mid twentieth century artists and researchers began to write computer programs to visualize music. In the late 1970's J.B. Mitroo and Nancy Herman used FORTRAN programs to create visual displays based on audio input [3]. A pioneering DOS program called *Cthugha[4]* appeared in the 1980's when computers began to have enough processing power to create real-time visualizations.

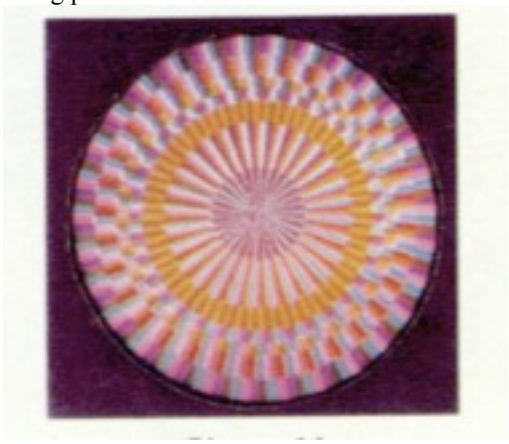

Figure 2. Mitroo Chord Progression Visualization

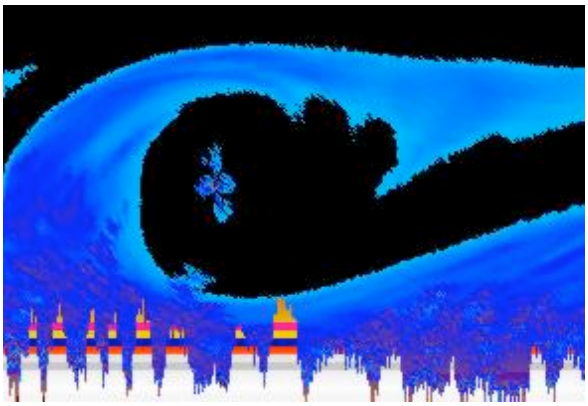

Figure 3. Cthuga '97

Nanayakkara, et al. did significant research in topics concerning audio visualization and also began developing a system similar to the one described herein, using MIDI signals to drive a Flash application [5].

3. Experiencing Audiovisual Stimuli

With the rise of digital technology we are able to receive audiovisual stimulation in many more formats and venues than prior generations. Today we are exposed to music on our computers, in our cars, on the street and in clubs and concert venues. Besides the traditional platforms of film and television, we experience visual media on animated billboards and Jumbotrons, iPhones and car headrest displays.

Many of these platforms support and depend on the marriage of audio and visual content and do not exist in a unisensory vacuum. As viewers we are always seeking correlations between the audio and visual aspects of the presentation. Perceived asynchronicities will draw our attention away from the artistic message and may negatively impact our experience.

Our brains fuse information from our visual and auditory systems in a process known as *audiovisual integration*. There are physical limits to both our sensory organs and the brain's processing ability that dictate the range of combined audiovisual stimuli that we can properly receive and comprehend.

3.1 Auditory System Characteristics

The range of audio frequencies that our ears can perceive is finite. Sources say the average hearing range is 16Hz to 16,384 Hz and a general rule of thumb (especially for loudspeaker design) is to consider a range of 20Hz to 20Khz [6]. The exact range varies depending on age and gender and higher frequencies tend to diminish in sensitivity as we age. Additionally, we are not equally sensitive to all frequencies but instead perceive frequencies around between 1Khz and 5Khz (corresponding to typical human voice) with maximum sensitivity while other frequencies require more energy for the same perceived loudness [7]. For the purposes of audiovisualization it is important to realize that elements of the music may not be as noticeable or effectual to all people.

There are also spectral-spatial limits whereby certain sounds might be masked (less detectable) by other, louder, sounds at similar frequencies. This masking is a result of the physical construction of our ears and how sensitive they are to small changes in frequency as compared to loud changes in volume. Temporal masking is also a concern as quieter sounds happening nearly simultaneously to louder sounds may be overlooked. These limitations of the human auditory system are exploited to positive effect for audio compression (as in the popular MP3 and related compression techniques) but can pose an issue if we are visualizing frequency or sound events that are not uniformly perceived.

3.2 Visual Perception

The visual system is similarly defined by operational qualities that must be taken into account. The human eye can perceive light with wavelengths in the range of 380nm (red) to 780nm (violet) [8]. Light falling immediately out of this range is known as *infrared* and *ultraviolet* and is typically invisible to humans. Video projectors and monitors are designed to produce light within the visible range, and standard digital cameras use filters to avoid picking up invisible light ranges that would cause the resultant image to not reflect what we see.

Our eyes sample the world at a very fast rate, which allows our view to seem continuous and *real*. When we view a synthetic moving picture on a television or projection screen it must refresh at least as fast as our eyes scan or else it will not appear continuous. The minimum update rate for a movie to appear continuous has been shown to be 24 frames per second (fps), which is the standard frame rate of film. For digital productions the rule of thumb is to strive for as fast a frame rate as possible, with many applications running at between 30 and 60 fps.

While it is important for the entire frame of a motion picture to update at a very fast rate our visual system has limits of the perceived velocity of objects on screen with regards to motion comprehension. If objects move too fast it becomes difficult for us to deduce relationships in movement between frames and other objects and thus understand the scene [9].

When designing visual animations it is important to remember the limits of the visual perception system to ensure that the audience sees what the artist intended.

4. Implementing Audiovisual Linkages

I have identified two broad scenarios in which audiovisual linkages can be attempted. The most challenging situation involves producing visual output in response to pre-recorded or otherwise opaque audio signals. This is also the case with most musical performances. In this example the visual artist does not have much more knowledge of the musical structure and timeline than the audience does.

Attempts at correlating visual stimuli with the auditory signal fall on a continuum between fully manual (the artist makes all decisions based on their interpretation of the piece) and automatic based on expensive (and not necessarily accurate) signal characterization via digital signal processing. Most club VJ'ing is a manual process and the output of popular music visualization software, such as the iTunes visualizer, falls into the automatic category.

The second and potentially easier scenario exists when the visual artist has direct knowledge of the inner structure of the music so as to efficiently derive meaningful cues for the visuals. This can take the form of having access to the full score in advance and manually sequencing visual events (as in the case of a music video) or receiving information on the fly as the music is performed.

4.1 MIDI in Live Visuals

Musical Instrument Digital Interface (MIDI) files are one particular example of using notation information to create the multimedia output. Rather than storing a quantized representation of the resultant audio waveform these files represent the song as a score of note numbers and velocities to be recited back using a software synthesizer or sample player. The advantage of this method is that the representation of the song can be on the order of kilobytes instead of megabytes, and the same 'song' can be made to sound very different depending on the soundbanks and software synthesizers used.

While MIDI is usually associated with audio output the existence of precise musical notation information creates great potential to design tightly synced visual presentations that do not need to be pre-sequenced for a particular track. Nanayakkara, et al. provide a good background on the subject of using MIDI to visualize the structural dynamics of music [5]. Their work was concerned with how to represent different elements of music using colors and abstract forms with special attention paid to historical research in synaethesia.

An example of a more direct and utilitarian mapping between music and video is *Midi Jam*, a free software program that depicts each midi instrument in the song as an animated cartoony form [10]. The General Midi specification defines a set of standard instruments that have unique identifying codes [11]. A properly constructed MIDI file makes such direct visualization very easy since it the software will know which string of notes corresponds to which instrument.

A downside of deriving the visualization directly from the notation (as in MIDI) is that it tends to lead to direct analogies that remain firmly entrenched in the mold of traditional music scores and spectra. To do anything more than show the rhythm of the kick drum or the frequency distribution (as in a spectrum analyzer frequently seen in music players and CD players) requires a higher-level knowledge of the macrostructures of the piece. This could be estimated using sophisticated processing of the raw notation[5] but it is far more desirable to get this information directly from the musician.

This high level musical information processing is what I have been working on with Aaron Mcleran. He creates generative music and beats using probabilistic processes in Max/MSP that lends itself well to effectively informing a visual performance.

Though the music has a decidedly electronic / experimental feel it is structured after the common characteristics of western music. The program has logical abstractions for typical instrumental and rhythmic components such as *kick drum*, *snare,* or *bass guitar* which each have their own specially tailored probability tables and control structures. The main program does not create sound itself but instead uses MIDI notes to represent the spatial-temporal features of the music and control software

synthesizers in Ableton Live and Reason. In this way a wide variety of sounds can be created and further processed independently of the stochastic control structure.

I also collaborated with the MAT Rhythm Seminar conducted by Matthew Wright in which a group of performers use their laptops to process sampled sounds in novel ways and attempt to synchronize globally using a clock signal sent out over OSC.

5. Razzle: a Live Visual Performance Platform

All of my experiments with audio-visual synchronization were conducted using *Razzle*, a large Java program I have been developing since December 2008. My aim with Razzle is to create a tool for live visual performance that is easily extensible. It consists of a core program that provides many services to small animation classes. I have sought to implement many reusable facilities as part of the core program so that the animations themselves can be as simple and quick to write as possible. For now I am focusing on 2D constructs using simple geometric shapes though I imagine it will expand to 3D in the future.

The core program consists of a drawing loop that runs a set of draw the current animation to the screen at up to thirty frames per second. All animations are stored in a data structure for easy access and the current animation is selected based on the state of a variable. In this way the current animation can change as fast as every frame if desired. There are many global parameters and modes that affect all animations, such as per-frame rotation. Additionally there is a Color Factory that produces a constant stream of colors for Fill and Stroke depending on the current color mode. The core Razzle program also implements recording features such as high-resolution PDF export, screenshots and movie export.

Each animation extends a base Animation class that contains many base methods that all animations will use or re-implement. Every frame the current animation's update() method can be run to advance the state of the animation, if desired. The draw() method is then executed to create the visual output. Varying the frequency of update() will cause the animation to appear to slow down without reducing the actual frame rate of the application.

Almost all parameters and important variables that affect how an animation appears or evolves over time are implemented using custom data types that implement the Observer pattern. This lets the GUI widgets that control them react to their change over time, as well as making it easier to save their state to preset files for later retrieval. Each parameter is either a *Boolean Parameter* (an on/off flag) or a *Numeric Parameter* (any floating point number confined to a specified range). The GUI itself is generated dynamically using Reflection, which frees the programmer from having to do anything special to get an animation parameter into the user interface after it has been declared and implemented. This increases animation turnaround time between iterations when animations are being developed.

5.1 Experimental Results in Audiovisual Collaboration

Our first collaboration was to make a series of introductory videos for the Media Arts & Technology colloquium series. Since there are potentially dozens of videos to be posted online we felt it would be interesting to make dozens of unique but aesthetically related videos *jingles* so no two videos would start exactly the same way. I devised an animation in java that received MIDI signals that advanced the animation state and then a final 'bang' to show the department logos on cue. By varying the period and duration between update signals the animation would appear to correlate roughly with the *swing* and *groove* of the music.

In this, my initial foray into generative visual synchronization, we controlled and varied the timescale of animation evolution but not aesthetic structure of the animation itself. The final appearance was essentially pre-determined once it started running, we just controlled when it finished.

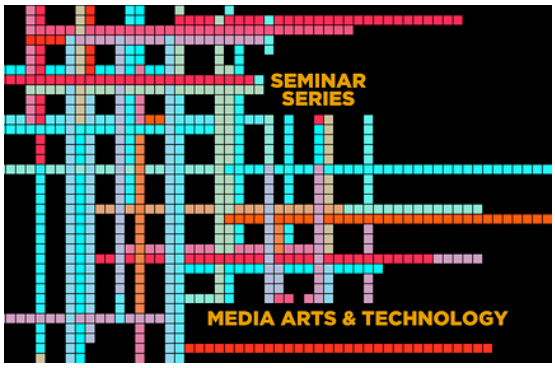

Figure 4. Colloquium Series Intro Videos

The most recent collaboration has explored more fine grained and powerful synchronization with an aim at live performance applications. We upgraded the communications infrastructure of utilize the expressive power of Open Sound Control (OSC) instead of the more note-bound MIDI protocol. This enabled us to devise named signals that more naturally translated to higher-level macro events. OSC provides the additional benefit of being more easily portable between computers as it operates as a network protocol. The power of OSC was humorously realized when both of us were unknowingly working on our programs simultaneously and I noticed my animations inexplicably reacting to unheard music via the Internet.

We started with the naïve approach of sending and interpreting low-level note information (pitch and velocity), as would be the case if the program was driven directly via a MIDI keyboard. The first animation

specifically written with audio reaction in mind consisted of note events plotted on a continuum similar to a spectrum analyzer. In the general case, pitch correlated to spatial distribution along the traditional X-axis while velocity affected the Y-axis distribution, which was found to be a fairly intuitive mapping. The spectrum was quantized to several dozen 'bins' to keep things evenly spread out for better visualization. I treated each bin in a way similar to a histogram so new notes accumulate in the bin, with a controllable shrinking effect slowly attenuating the entire spectrum.

I identified two distinct musical modes that I might operate in – dubbed *spectral* and *temporal*. As with most music, generative music can operate on many different frequencies that combine to create a rich soundscape. Plotting notes along an axis based on their natural frequency / spectral distribution worked well to visualize the dynamics of the song. Since the music was wholly generative it was simple to change the range of note pitches and velocities being sent over OSC or MIDI since they were handled independently of the local synthesizer control notes. This flexibility led to a variety of interesting appearances that pulsated with the general rhythm of the music while varying spatially in response to the instantaneous tonal quality.

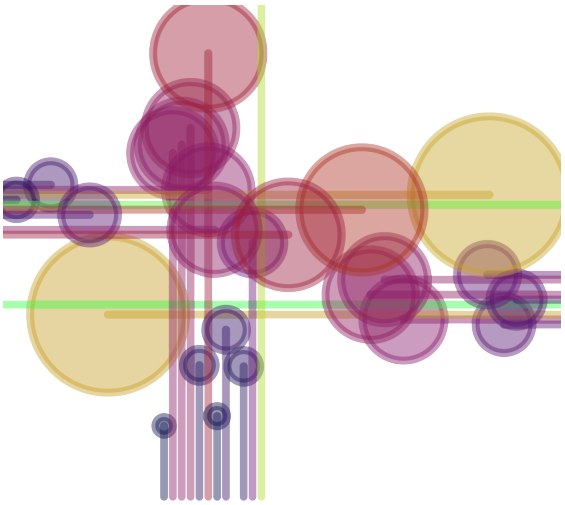

Figure 5. MIDI Notes in Razzle

An entirely different visual-musical performance scenario that I considered was percussion. Electronic drum pads are popular for many musicians to create a wide array of synthesized percussive sounds. While many of these drum pads operate by sending 'trigger' pulses to a drum computer they often have standard MIDI output as well. In this case it would be possible to connect them to the animations to have the live drumming influence the visuals. While velocity data will be just as useful as in the *spectral* context, the range of note pitches is typically limited to one note per drum pad. In this case I consider the music to be characterized as a temporally distributed pattern of repetitive notes instead of the rich frequency spectrum of other music types.

For this *temporal* case I implemented a 'clock' tick that continually moves down the X axis and wraps around to the beginning. Whenever a note is triggered it is plotted at the current location of the clock tick. The idea is that if the percussion is semi-repetitive (and especially if it is somewhat in tune with the clock signal) a temporal pattern will emerge as notes are plotted at the same couple of locations while other locations remain empty of notes.

Direct translation of musical note data onto traditional rectilinear projections was an interesting and relatively simple first step at demonstrating a basic audiovisual linkage but was not sustainable across multiple animations that do not fit with the underlying metaphor. I thus pondered system-wide macro commands that would have a meaningful effect on most animations regardless of their appearance or how they are structured.

Based on the preliminary audiovisual cognition literature review and research I have conducted I settled on a few broad concepts for how to impart musical dynamic information onto the visual landscape.

5.2 Modulating Animation Kinematics and Structure

As humans we often look for real-world parallels to the virtual worlds we see projected on screen. Video games and 3D animated movies are always striving for ultimate realism as we race to render the real world into a convincing virtual replica. Much attention has been paid to the importance of pixel resolution and overall level of textural and structural detail but issues of dynamics and kinematics are also important. Still images rendered at extremely high resolutions with very detailed models can appear very lifelike but when they are set into motion it is entirely possible for the illusion of realism to disappear without careful work on the simulation the motion of objects in space.

Newtonian physics describes the kinematics of the real world in terms of *position, velocity,* and *acceleration* and these concepts are equally important in the virtual domain. While the topic of virtual motion simulation has been thoroughly explored in the context of 3D games and animation I have found that not nearly as much attention has been paid to applying these concepts to audio visualizations. Some of my animations were written to simulate a physical concept, such as gravitational pull and rebound velocity in particles but I am more interested in how the overall motion characteristics of an animated system can be modulated in concert with the music to create the audio-visual linkage, regardless of the concept of the specific animation.

My overriding goal with depicting audio events through the modification of motion parameters is for the reactions to *influence*, but not *define* the animation. I wanted the animations to be running as usual and be enhanced by the presence of additional musical

information and control signals but not dependent on them for a pleasant aesthetic appearance. The program was initially written without any explicit external control or event correlation and I wanted to keep a baseline behavior for situations where such control data is absent or intermittent.

For the videos with Aaron McLeran I experimented with mapping note events to brief increases in particle velocity. In this way the particles would be always be moving based on the overall rules of the animation but would momentarily accelerate in time with the beat. The additional velocity was constantly attenuated so that the instantaneous acceleration would be followed by a predictable deceleration to return the animation state to normal until the next note. It was important for the changes to be swift so that they can correlate to a (possibly rapid) beat and not appear and simple rhythmic 'wave'. To capture tangible perceptual differences between note events I mapped the note velocity (often corresponding to loudness) to the relative magnitude of the acceleration pulse. In this way a hard-hitting kick drum would cause more noticeable and bold movements than a subtle bass guitar line.

Intelligently modulating particle velocity through brief pulses of acceleration provided convincing dynamic effects (as the viewer saw the motion changing in real-time with the music) but could also leave longer-lasting temporal traces, as the particles' trails often looked different (more spaced out or perhaps thicker) depending on how dependent on velocity they were. In this way a trail can tell the story of the beat and musical patterns can emerge that last beyond the initial attack and decay of the audio events.

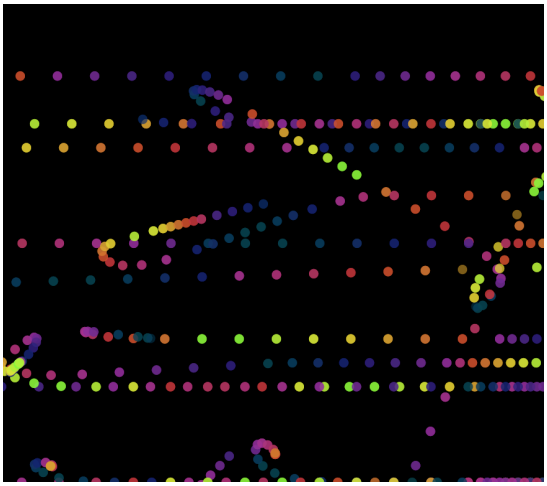

Figure 6. MIDI Notes Modulate Particle Velocity

I also looked at ways to render note events by modifying the animation structure and aesthetical characteristics. Initially experiments have involved modulating the physical size or growth speed of animation objects in response to note events. The canonical example of this was maintaining a constantly attenuated scale factor that was increased in correspondence to note velocity information and used to calculate the display scale of animation objects. I implemented this approach in an animation that consists of a constantly changing cloud of dots. With velocity cues the cloud of circles appeared to *explode* in size very briefly in response to note events. In practice the effect was very noticeable when applied to dominant instruments with high velocity such as the kick drum.

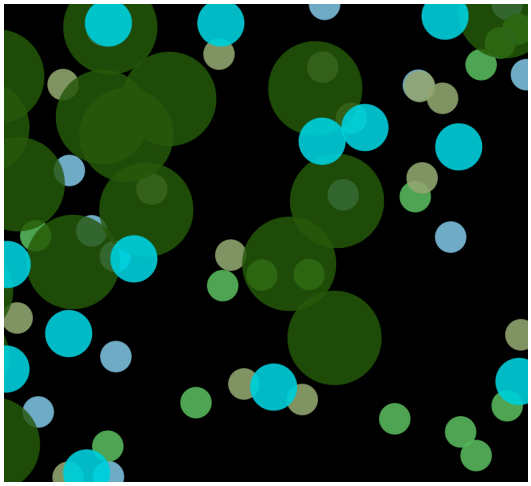

Figure 7. Dot Size Modulated by Note Events

The preceding experiments in using note-data to modulate the dynamics of an animation almost always required animation-specific code to implement. This is because each animation serves a different aesthetic purpose and is thus structured relatively different. I implemented the general architecture by specifying a *note()* method that all animations have but do not implement by default. I then went and re-implemented the function for a few animations as a test.

Implementing these *note effects* at the animation-level allows for great expressive power but it was an unfortunate extra burden to have to go write custom code for each animation that was to respond to these note events. As the number of animations increases and changes it would quickly become a nuisance to solely rely on such a setup. A more general solution was called for that would affect all animations at a global system level.

5.3 Spin & Jitter: Global Kinematic Macros

To fully take advantage of the power of a control linkage between the music generation and the visual output I investigated simple ways to affect the animation composition as a whole regardless of how the actual animation subroutine is structured. These methods were implemented in the main program class and given meaningful names to easily trigger them remotely. Initially two methods were implemented, *Spin* and *Jitter*. Both methods are macro actions that toggle numerous animation parameters as a group to produce a temporary, repeatable

effect in one action. Each uses the *Auto-set* function, which is an object that each animation has that allows for parameters to be set to certain values at certain times automatically. The advantage of these macros is that the music program need only send a single command that triggers several individual actions behind the scenes.

Spin utilizes per-frame automatic rotation to cause an otherwise un-rotated scene to spin quickly and temporarily for dramatic effect. Global rotation was implemented early on in this program's development because I realized that simply rotating the canvas each frame can make something entirely new from existing animations. The amount of rotation per frame is easily controllable and has a very big effect on the appearance of the animation. The particulars of the animation implementation also determine how it responds to rotation. The animation components must be somehow correlated spatially between frames or else rotation will not have a noticeable effect. *Spin* takes advantage of the kinematic quantities covered earlier by quickly increasing and then decreasing the spin velocity so the animation appears to quickly accelerate radially before going back to normal.

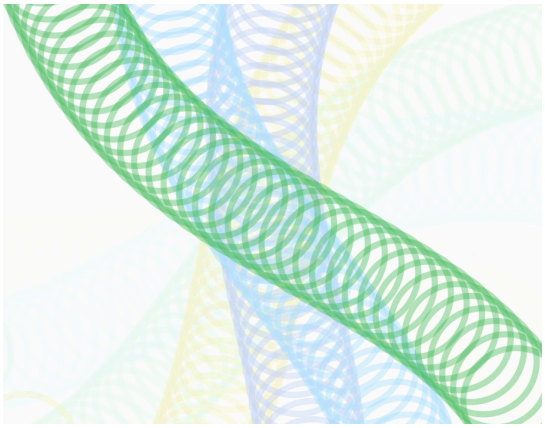

Figure 8. Example of *Spin*

Coordinate jitter is comparable to a 'rumble' effect that makes the whole canvas shake. It is implemented be means of translating the whole coordinate system randomly by a set (small) amount before the current frame is drawn. All subsequent drawing calls will be offset from the previous frame. The magnitude of the jitter is controllable and ranges from a slight vibration to all-out distortion of the animation fluidity. The *Jitter* macro triggers the coordinate jitter for a few frames and automatically turns it off. It was first used in the end of the Colloquium Series video project where the logos 'bump' on screen and later used in response to major bass events to lend the impression that the whole canvas is rocked by the music. If subsequent frames are accumulated the result is to turn previously well-defined curves into a more amorphous field of curves.

Figure 9. Example of *Jitter*

I intend to further explore globally applicable ways to affect the animation in response to external control as it is definitely preferable than requiring all animations to implement custom code.

5.4 Event Quantization

The final area of experimentation and research concerns not *what* happens in response to musical information, but *when* it happens. Music programs like Ableton Live have made great waves with their warping functionality that attempts to quantize multiple audio files to the same *beat grid*. The idea is that it is desirable that musical events happen in sync with each other and a master tempo so there is more constructive harmony and not the destructive forces of beats that hit at off-times and muddy the melody. In Live when you click the Play button the clip usually does not play immediately but instead waits until the next quantization instance so things stay in sync.

I implemented event quantization by setting up the notion of a clock signal that is received over OSC from the music program or generated internally using a Java Timer object. The clock message has a tick number that increments by one each frame which allows us to align our actions to fractions of the clock frequency using modulus arithmetic. The modulus operation allows for things to happen at slower rates than the master clock rate.

If the music obeys the clock in a meaningful way then quantizing events to the clock can really enhance the fluidity of the performance. When people are tapping their feet along with a beat and see the animation change exactly in time with the bass they make an immediate connection between the audio and visual stimuli around them.

For now only Boolean parameters are quantized by default. Unlike Numeric Parameters, which typically control the density or speed of something, these parameters control animation modalities by toggling the display of various elements of the animation or changing kinematic behavior. Thus toggling a Boolean Parameter often results in a very noticeable discontinuity in form. If the change happens in time with the beat the effect is often reinforced

positively – if it happens in isolation it can look sloppy and uncorrelated.

Besides quantizing specific one-shot events such as parameter settings I also looked at using the clock signal to continually quantize period events. For this I focused on synchronizing the color factory updates and the animation updates as a whole. Initially the Color Factory would update each frame which means that the colors on the screen could change very fast depending on the parameters of the color factory. This can lead to some interesting but often jarring strobing effects. To counter act this effect I implemented a system to update the color factory on the clock tick. In this way the same color will likely be re-used over multiple frames which creates a smoother appearance. I plan to study the perceptual aspects of synchronized color in greater detail but this initial step has increased the aesthetic pleasantness of the program.

As stated above, the update() method of an animation advances the state of the animation and until it is executed the animation should draw the same across multiple frames. Initially the update method was assumed to run every frame, which resulted in a visual aesthetic heavily dependent on the speed of the hardware. To mitigate this I implemented a system whereby the update() method is optionally synchronized to the clock tick. In this way faster music should result in 'faster' animations. Additionally if the tempo of the clock signal itself changes over time, as can be the case in experimental music, then the animation should follow suit. This provides an additional kind of audio-visual feedback with no extra work on the part of the animation programmer.

Clock-based event quantization has proven itself to be very effective when the music program is generating the clock ticks in a meaningful way, as with my collaboration with Aaron's Max patch. It was less effective in my collaboration with the Rhythm seminar since the clock came from an isolated computer and it was left to each individual performer to decide how they would use the clock (if at all). In these instances the color and update() synchronization helped match the animations with the overall feel of the music performance but parameter change event quantization was virtually unnoticeable.

6. Conclusion & Future Work

The major limitation to the methods used herein is the reliance on an external clock signal for even the most basic synchronization. When collaborating with a generative musician this is a trivial requirement. Traditional live music possesses no such clock and as such most of this work would not apply when doing visuals for a live band.

To bridge that gap I intend to investigate live beat-detection methods to create a separate program that analyzes an audio stream for the most basic rhythm information and generates a clock signal from this. If the calculated tempo is relatively correct then basic event synchronization, such as color, should be feasible. A system to control the phase relationship between the clock and the live audio signal would also be useful to fine-tune the synchronization.

To further tailor the visual output to the human senses I will also be taking Psychology courses in Perception. Additional information gained will influence how the animations are written and evolved to ensure they are as effective as possible.

In the specific circumstances set forth these initial experiments in audiovisual linkages worked very well. Working with computer music allowed the musician to send custom-tailored OSC messages exactly on time, which increased the accuracy of linkage. Audience members who were not conditioned to expect synchronized visuals came up to me after the show to say they noticed how music and visuals worked in tandem. Future work will build on this to make the system more generally applicable so it can be used in clubs and concerts.

References

[1] Collins, N.,(2003). Generative Music and Laptop Performance. Contemporary Music Review, Vol. 22, Issue 4, Pages 67 – 79

[2] "Clavilux," Thomas Wilfred Foundation. Available: http://clavilux.org

[3] Mitroo, J.B., Herman, N (1979). Movies from Music: Visualizing Musical Compositions. ACM

[4] "Cthuga," Available: http://www.afn.org/~cthugha/

[5] Nanayakkara, S.C., Taylor, E., Wyse, L., & Ong, S.H. Towards an Experiential Music Visualizer. ICICS 2007

[6] "Hearing range," Wikipedia. 26 May 2009 Available: http://en.wikipedia.org/w/index.php?title=Hearing_range&oldid=292468533

[7] "Equal-loudness contour." Wikipedia. 31 May 2009, Available: http://en.wikipedia.org/w/index.php?title=Equal-loudness_contour&oldid=293441314

[8] "Visible Spectrum." Wikipedia. 11 June 2009, Available: http://en.wikipedia.org/w/index.php?title=Visible_spectrum&oldid=295856886

[9] Lappin, J. S., Tadin, D., Nyquist, J. B., & Corn, A. L., Spatial and temporal limits of motion perception across variations in speed, eccentricity, and low vision. Journal of Vision 9(1):30, 1-14, http://journalofvision.org/9/1/30/, 2009

[10] Haag, S, Midi Jam., 2006. Available: http://www.gamesbyscott.com/midijam.htm

[11] "General MIDI Specification 1.0." Midi Manufacturer's Association. 1991, Available: http://www.midi.org/techspecs/gm.php