 Generating Music and Graphics from Dance
Devin M. Hilly, B.S. Computer Science, 2018
Thesis advised by Professor Holly Rushmeier, Department of Computer Science

PROJECT SUMMARY

The goal of this project was to create a program which generates music and graphics based on movement, specifically dance. The idea was to generate sounds which correspond with the musical events that one would choreograph a particular move to, and to create graphics which reflect these decisions. For example, hits in music often lead a person to choreograph a “hit” in movement, faster music typically leads to faster movement, and the proximity of a dancer to the ground can align with changes in dynamics and pitch. The graphics portion of the tool provides viewers with a visual representation of these relationships.

TOOLS USED

The project requires a motion sensor, software which can interpret the data from the sensor, software which generates music, and software which generates images. I will now cover which tools I used as well as other tools I considered before settling on the final configuration.

Hardware

The primary piece of hardware required for this project is a motion sensor. I used an Xbox 360 Kinect, Model 1414. The Kinect is light and easily transportable, which makes the tool adaptable to different performance spaces and provided me with flexibility when working on the project. That said, the Kinect is a relatively simple tool and provides less data about the subject than a production-quality motion sensor would provide. Yet because the scope of the project is small, it seemed appropriate for the amount of time I would have to master the tool.

Before settling on the Kinect, I considered using the Vicon Shogun motion capture system. When I spoke with Professor Rushmeier to propose my project, she informed me that the university had just installed a 20-camera Vicon Shogun system at the Collaborative Center for Arts and Media (CCAM). I set up an appointment with Johannes DeYoung, the faculty director at the CCAM. Professor DeYoung has taught courses on animation at the School of Art for years and was instrumental in bringing this tool to Yale. When we met, he showed me the system, which was set up inside a spacious dance studio, perfect for my project. He suggested I audit his new class,
Animation Studio, on Friday mornings, in which he was teaching students how to use the new system.

I attended the class three times, and ultimately decided not to use the system. Though the floor in the studio is perfect for working with dancers, there was no space on the walls of the room to project the graphics for my project. I also knew that I wanted to have at least one performative demonstration of the project, and the Mocap studio could not accommodate an audience. Additionally, using the system requires a complicated calibration which usually took the class over an hour of the two hour seminar to complete. Finally, because the system was new to the university, instructors and students at the CCAM were just learning how to use it, meaning it would be hard for me to get help with my project. I feared that if I opted for the Vicon system, I would spend most of the semester understanding how to use the tool, leaving me little time to experiment with the music and graphics. Ultimately the Kinect was a better tool for this project.

**Software**

I used an application called Synapse to read and interpret data from the Kinect, and the SuperCollider programming language and IDE to generate music and graphics. I had experience generating music in SuperCollider, after having taken CPSC 431, *Algorithmic Computer Music*, at Yale. I also knew that SuperCollider had graphic generation capabilities, though I had never used this feature. Realizing that SuperCollider was my best option for music and graphics, I looked into options for feeding data from the Kinect into SuperCollider.

I decided to use an application called Synapse, developed by Ryan Challinor,¹ to import and interpret the data from the Kinect. Synapse sends and receives Open Sound Control (OSC) messages to and from the Kinect, and also sends and receives messages to and from SuperCollider. The latter messages relay information about the position of different parts of the body in world space, relative to the torso, and as projected on the screen. Synapse also provides data about when different parts of the body produce a “hit” in movement. Synapse has the ability to track 14 different joints on the body—the head, torso, hands, elbows, shoulders, hips, knees, and feet.

I was drawn to Synapse both because of its ease of usability and because it was well-documented online. I used Eli Fieldsteel’s YouTube tutorial, “SuperCollider Tutorial: 13. Xbox Kinect”² as reference for much of my project. The tutorial walks

---

through how to use an Xbox Kinect motion sensor to feed data into Synapse and interpret this data in SuperCollider—my exact setup.

**FEATURES of MUSIC**

I aimed to ensure that each fundamental element of the music was considered in my tool. While every element may not be informed by the music, whether the subject should have control over a particular element was intentional and is justified below.

**Tone Color**

Tone color, or timbre, is defined by the sonorous qualities of an instrument. It is the feature of music which is influenced by the instruments used to create sound. In SuperCollider, instruments are mimicked using synthesizers, defined as SynthDefs and declared as Synths. Any arguments to the SynthDef can be changed when the Synth is declared or while it is in use. Though I did use this feature to change qualities like dynamics and pitch, which I will discuss later, I chose not to let the user’s movement define which “instruments” were used. First, changing a Synth mid-song to resemble a different instrument is both atypical of music (even electronic music) and out of scope for this project. Second, while certain instruments are used in particular genres of music, and genres of music often correspond with certain styles of dance, I feel there are few intrinsic relationships between instruments used and qualities of movement. For these reasons I chose the SynthDefs myself, and decided which ones should be used in which musical events.

SynthDefs were taken from online sources. The SynthDefs used were a kick drum, a snare, a bass, a “gooey” synth, and a string instrument. All SynthDef authors are cited in the code.

**Rhythm**

The second musical element I considered was rhythm. Rhythm is described as the durational pattern of musical notes in time. Rhythm is an essential feature for dance; after all, being a good dancer is colloquially described as “having rhythm.” For this reason I focused on the relationship between the rhythm of the generated compositions and the movement more than any other musical element.

Movement in dance is usually choreographed to align with rhythmic elements, so I wanted to replicate this in my compositions. This turned out to be easy to

---

implement—as sound events in my tool are triggered by the dancers’ gestures, the rhythm is almost completely determined by the dancer.

The main challenge in linking rhythm and sound, then, was ensuring that musical events were generated only for significant gestures in the movement. Users would likely be moving somewhat constantly while the tool is in action, so the program needs to be able to pick out which movements are important.

Because “hits” in music often correspond with “hits” in a dancer’s movement, I began by establishing this relationship. Luckily, Synapse already supports detecting hit events, so all I needed to do was read this information into SuperCollider and trigger musical events accordingly. I decided to only generate musical hits when there are upward bodily hits in either leg, and sideways and upward hits in either arm, as these are typically the most prominently choreographed hits in dance. For the hit sound, I had the sound of a bass and snare play briefly and loudly. Though I originally generated this sound for both the hands and feet, I realized later that the musical hits I had added sounded too aggressive for a hit with the subject’s hand, as dancers frequently make dramatic gestures with their hands, even when the music is relatively tame. I removed the snare and bass sound for these gestures and added instead an ambient, gooey noise, which lasts only a few seconds after the subject performs the gesture.

**Meter**

Meter is the pattern of simple, repeating, even beats which one usually hears in the background of a song. It is closely related to rhythm, as rhythmic events usually exist in relation to the meter. Meter helps to establish the tempo of a composition, or the speed of the notes in time. It typically remains constant throughout a piece of music.

I decided to use meter in my compositions because I felt it played an essential role in distinguishing the generated noise as music rather than merely a collection of sounds. Computer-generated sound borders on sounding non-musical, so I wanted to ensure I had as many of the signature features of music as possible to make it sound appropriate and convincing. Having meter also provided me with a reference point for other musical events, and created an audible beat for the users to dance to as they were using the tool.

The meter in the composition is consciously determined by the user and is established at the beginning of the composition. Though I had originally planned on having the program figure out the appropriate meter based on the speed of the dancer’s movement, this was nearly impossible in practice. While discussing this with Professor Rushmeier, she proposed that I instead give the dancer the option to consciously
determine the meter which would be played during the piece. We decided that I would try programming the tool to pick up on four isolated movements by the dancer (claps, taps of the foot, hits of the head), which would become the meter for the composition.

This also proved challenging, as the only isolated movement Synapse can detect is a hit. Though Synapse allows programmers to adjust the sensitivity of detection of these hits, the detection sensitivity is specific to the joint tracked, making it impossible to track less sensitive hits in that joint for other purposes (eg. the hits I discussed earlier). I also feared that even if I did increase the sensitivity of the hit detection, that the system would over-interpret movements as hits, leaving the subject out of control once again. Lastly, I realized that a system such as this would either require the dancer to be very precise about the timing of their movements, or would ignore in-between beats altogether, registering only the total time between the first and last beat and dividing this by a certain number of counts.

I leveraged this last quirk to my advantage and decided to detect only the “beginning” and “ending” of the tempo-determining gesture. As deciding the tempo should be the very first thing the dancer does after they are detected by the sensor, the “beginning” point for the tempo would have to be the moment of detection. The sensor detects the user when they assume the position shown in the figure on the right, so I decided that the tempo would be established based on the time it takes the subject to lower their arms from this position. My tool creates a noise the moment the user is detected, and from there the subject lowers both arms in whichever fashion they please, counting 4 counts in their head as they lower them. Once their arms have reached their sides, which I estimated at about 100 pixels below their torso, the meter begins to play.

Pitch

Pitch refers to the frequency of a sound, or the highness or lowness of a note. Though not all instruments vary in pitch, most compositions feature at least one instrument which does. Though there is not a strict correlation between pitch and movement in dance, there is a correlation between pitch and mood. Higher pitches tend to signify happier music, and lower pitches sound sadder. Similarly, high-energy dances usually
features many jumps, leaps, and other movements which bring the dancer higher in space, while somber choreography tends to be lower to the ground. These relationships are very general and are certainly not always reflected, but I still aimed to respect them in my music.

The first way I did this was by creating a low bass sound when user is lower in space. This is detected by their position relative to the screen. The tool detects when the user has entered the “lower” space for the first time, and then keeps track of when they go back into the “higher” space in the event that they come down again. It triggers a bass sound each time they re-enter the lower portion of space.

I also wanted to have constant melodic sounds which moved up and down with the user through the course of the composition. This was achieved by using a three instrument chord of strings. The strings are introduced eight, twelve, and sixteen counts after the beginning of the composition. The pitch of each string is determined by the vertical position of the dancer’s torso. At each of the three points in time, the dancer’s torso position is rounded to the frequency of the nearest note in the chromatic scale, which ranges from 440 to 830 Hz. As the dancer’s vertical position fluctuates between 0 and 480 pixels, I scaled their position appropriately.

The chord pitches worked better in theory than in practice. First, the dancer’s vertical position remained relatively constant while they were dancing, so there were often duplicates amongst the three notes. Second, I tried modulating the chords after all three notes were determined, but as most of the numbers corresponding to the vertical positions a person moves through in space are not frequencies which appear on the chromatic scale, the music sounded very atonal. The sounds were unpleasant enough that I removed the modulation, and the chord remained in its initial position for the entire composition.

**Dynamics**

Lastly, I considered the dynamics of the song, or the loudness and softness of sound. Though most sounds in the composition are played at about the same amplitude, I varied the dynamics of the string instruments slightly. I decided that a similar relationship to the pitch, where amplitude moves with the vertical position of the dancer, was appropriate for this musical element.

**FEATURES of GRAPHICS**

Having never done anything with computer graphics, I spent some time experimenting with graphics in SuperCollider. When I felt comfortable with this, I assigned myself what I thought would be a simple task: drawing lines which followed the strokes of the
dancers’ hands and feet. Generating graphics which were influenced by the data received from the OSC messages turned out to be much more challenging than I expected, as most of the examples I found online did not interface with users. I had trouble figuring out how to make changes to the graphics window within an OSCdef, as scheduling events within the OSCdef was throwing errors. I finally realized that I could modify variables used within the window inside the OSCdef, and then refresh the window somewhere else in the program, which would then reflect those changes. I set the window to refresh each 1/20th of a second, thinking that the program would draw a connected line throughout the window. Instead, the stroke of the pen, which resembled a rectangle, would disappear and reappear every time the window refreshed, creating the effect of a flickering rectangular shape which followed the hands and feet. Liking this effect, I then added points for the head and torso to imitate a full figure.

This flickering figure appeared on a screen which was generated as a different random color each time the code is evaluated. In addition to tracking the dancer, each of the musical decisions I made was reflected in the graphics. I will describe these features below.

**Rhythmic hits**

Hits were represented as circles leaving the figure’s hands and feet and flying off the screen. The circles appear as a different random color each time they are generated. They move off the screen with speed, and grow in size as they move away, giving them more visual energy.

**Chords**

The chords are represented as lines which run the full width of the screen. The lines appear one at a time as each of the three notes is determined. The vertical position of the lines is determined by the user’s vertical position on the screen at the time they are generated. The color of the lines is also determined randomly.

**Modulation of pitch**

Though I removed the modulation of the chords, I decided to keep the graphics I generated for them. As the user moves lower in space, the background color of the graphics screen grows darker, and as they move higher in space, it grows lighter. This was achieved by blending the original background color with white and black as the user moves above and below their original position.

**Dynamics**
Dynamics are represented by the growth of the lines which represent the chords. The lines grow into ovals which are meant to resemble waves, alluding to sound waves. The ovals grow taller as the sound gets louder and shrink as it gets quieter.

**PERFORMANCES and FOOTAGE**

I had one performance of my piece while it was still in production. I also filmed a few improvisations of the piece and compiled them into a video. This video, along with a video of the performance, was submitted with the project.

The performance took place at the Yaledancers Fall Show in November 2017. I had previously set a deadline for myself to have a prototype of the project complete for the show, both so that I could get feedback on my progress thus far and have extra motivation for completing my work. Though it was successful in that regard, the live performance did introduce some difficulties. First, the Kinect is intended only to track users from a distance of about 5-10 feet. Although there is some flexibility with this once the user is detected, it was vital that the subject stand close to the sensor before they had been detected. Second, the cable which is meant to connect the sensor to the Xbox console is fairly short, as it is only meant to extend from the Xbox to the Kinect. This meant that the Kinect had to either be positioned in the corner of the stage or be hooked up to a USB extension cord. The USB extension unfortunately introduced too much latency into the system, so I settled for using a smaller portion of the stage. Lastly, the lighting in the theater made it was hard for the sensor to detect the dancers.

I was able to demo the project despite these issues. I unfortunately was not present at either the Thursday or Saturday show, and both shows had technical difficulties. At the Thursday show, firing up the projector automatically logged out my computer, which was locked under the administrative account. The demo went smoothly on Friday night and I was able to capture a video of the performance. At the Saturday night show, the code was correctly evaluated, but the Kinect had trouble recognizing the user, and the dancer moved in silence. Though the failure of the project in performance was disappointing, it was not unexpected, and showed that I would need to be present and operating the system at future performances.

The tool was set up a second time in December 2017 at the Off-Broadway Theater at Yale. Through the assistance of Justin Deland, the Senior Technical Director of Yale Undergraduate Productions, I was able to configure my project at the theater. I shot several performances of the project which were later compiled into a video for submission.

**CONCLUSION**
Overall, I was pleased with how the project came out. I successfully created a tool which generates music to match movement, and the relationship between the gestures and the musical events read well in performance. Though the compositions generated do not exist well on their own, they are not meant to, and the project was more concerned with concept than with how it sounded or looked.

I could not have completed this project without the help of my adviser Holly Rushmeier, CCAM director Johannes DeYoung, Senior Technical Director of Undergraduate Productions Justin Deland, dancers Brandon Rabaria, Michaela Vitagliano, and Naiya Speight-Leggett, Ryan Challinor, the developer of Synapse, and Eli Fieldsteel, whose YouTube tutorials provided immeasurable help.