The Virtual Conductor

Gene Kim
genelkim@cs.washington.edu

Dustin Walde
dwalde@cs.washington.edu

Michael Wiktorek
wiktorek@cs.washington.edu

ABSTRACT

The Virtual Conductor is a software program that immerses the user in a 3D environment with high-level control over the generation and playback of music using hand gestures and movement. An Oculus Rift immerses the user in a 3D environment and the user's gestures are communicated through a Leap Motion Controller. Music is generated from an algorithmic composer which takes input from the gestures read by the Leap Motion. The Virtual Conductor synthesizes the concept of musical conducting with computational music generation and new immersive technologies to create a novel, immersive, and realistic set of interactions with generated music.

1. INTRODUCTION

With advances in human-computer interface technology, programs for conducting recorded music have been created in recent years. Our aim in creating the Virtual Conductor was to synthesize the idea of a natural conducting interface for music with the ability to computationally generate music for several different virtual instruments on the fly. To make the experience more natural and realistic, we use the Oculus Rift and Leap Motion controller as an interface with the virtual environment.

The hand gestures used to interact with the music combine motions from real conducting and from digital user interfaces so that the set of hand motions used feel natural to the user and can be recognized by the program.

2. RELATED WORK

There has been previous work in creating programs for conducting and for generating music, but we have not found any examples that merge the two into a single application as we have done. Preexisting examples of virtual conductors generally simulate conducting but not composition. That is, the user can influence the playback of pre-composed or pre-recorded music. The Mendelssohn Effektorium, an installation in Leipzig's Mendelssohn-Bartholdy Museum, allows the user to conduct a virtual orchestra using a baton. Their interface combines a Leap Motion Controller and a touchscreen [1]. A related but different project animates a virtual human to act as a conductor [2].

Extensive research has gone into algorithmic composition. We looked at two papers that summarize previous algorithmic composition research [3,4]. These papers go over more topics than are appropriate to review here, but several examples represent the breadth of research that has taken place in this field. Music generation algorithms have been designed using rule-based systems/logic, genetic algorithms, statistical machine learning, many variations of Markov chains, fractal models, and other variations on machine learning. These algorithms have achieved melody composition, harmonization, jazz improvisation, and the generation of variations on a theme, among other compositional techniques. We chose a probabilistic rule-based system in order to create an intuitive model that generates enough variation to keep providing interest over time.

3. SOLUTION

3.1 Overall Design

The Virtual Conductor uses a client-server model, with the music composition system and the virtual environment as the server and client, respectively. The user interacts with the virtual environment using the Oculus Rift and the Leap Motion Controller. Any changes the user makes via hand controls are sent over the network to the server, which takes those changes into account when generating further notes.

The environment is built in the Unity game engine and provides the user with 3D representations of the stage, audience, instruments, sound playback, and hands as recognized by the Leap Motion. The user can examine the environment by rotating or translating their head, just as they might while standing still on a stage in real life. The user begins in the instrument selection phase by placing selecting and placing objects onto a stage and moving the go object, as described in the virtual environment section. The selected instruments are sent as a message to the composition server, which begins composing with the given instruments and default parameters. The user can then manipulate the composition in real-time with a variety of gestures using the Leap Motion and Oculus Rift.

The virtual environment keeps the composer in sync by sending messages with its current playback sample position to the composer. The composer uses this information to fill the audio stream as necessary. The stream acts as a buffer between the composer and the playback system and ensures continuous playback even though the composer must switch between feeding the stream and composing the next sequence of notes.

3.2 Virtual Environment

The virtual environment that makes up the directly interactive part of the program consists of two distinct scenarios, instrument selection and onstage conducting.

In most interactive virtual environments, functions like selecting an instrument and placing it on a stage might be done with a two-dimensional text-based menu. However, in the fully three-dimensional environment produced by the Oculus Rift, a two-dimensional menu can become awkward at best. Giving the user only hand control makes the problem worse; buttons pushed by a user with the Leap Motion virtual hands provide no tactile feedback, and don't feel particularly natural. Our solution to this problem was to take the functionality of a game menu and build it using manipulable objects that can be grabbed and moved by the user.

Our instrument selection phase places a set of spheres labeled with instrument names in front of the user, along with a small scale model of the conducting stage, complete with individually-marked spots where instruments may be placed. Using the "grab" recognition described in the interface controls section, the user picks up and places individual instruments into their respective

positions on the stage. When done, the user then grabs a sphere labeled “go” and places it into a specially-labeled object to their left. The motions involved in grabbing and placing individual instruments provide a level of tactility that buttons or text alone could not, and further serve to immerse the user in the virtual environment. Upon receiving the “go” signal, a controller sends the number and names of the instruments to be used to the server.

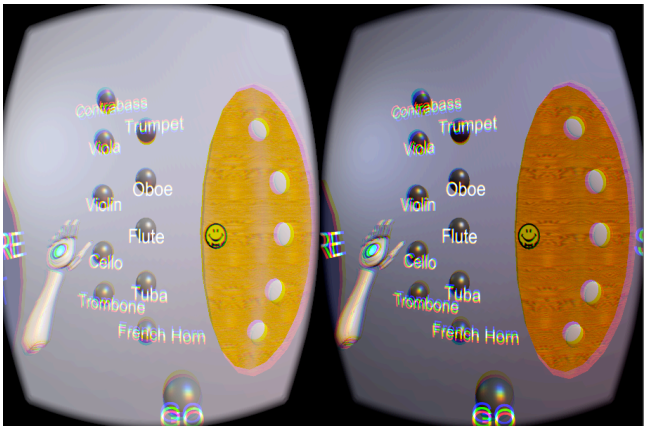

Figure 1. Instrument Selection

The second scenario we simulate in our virtual environment is, as the name of the program suggests, a stage on which the user is free to conduct a musical ensemble composed of the instruments selected in the instrument selection scenario. Many of the challenges in this scenario involved the implementation and feel of Leap Motion controls.

A particularly notable issue regarding the Leap Motion controller was where it should be placed relative to the user. A natural solution places the controller directly on the Oculus Rift, facing forwards, so that in both the virtual environment and real world, the user’s hands will be detected in the same region in front of their field of vision. This solution feels the most natural and provides the most intuitive level of control over the virtual hands provided by the program. However, we found that this solution made for extremely unreliable hand gesture detection, since the user’s fingers are obscured from the view of the stereo infrared camera on the Leap Motion controller by the user’s hands and arms. Instead we used a solution that leaves the Leap Motion controller on the desk in front of the user, but moves the virtual region of hand control along with the user’s virtual head. While this forces the user to move their hands separately from their head in a slightly non-intuitive way, it provides the best compromise between range of motion and accuracy of gesture detection. This gave us access to gestures like pointing and making a fist, while also allowing the user to direct their hand motions at any instrument in their visual field.

We used the head pointing information from the Oculus Rift to differentiate between controls for individual instruments and for the ensemble as a whole. Individual instruments detect when the user’s virtual head is pointing directly at them. This gives the user a way to select individual instruments to control without having to use a hand gesture that could be used for more direct musical control. A gesture controller constructs control messages based on whichever instrument is currently being looked at. If two hands are used in a gesture, the gesture controls the ensemble. In this manner, the user has direct control over the musical articulation of individual instruments as well over the ensemble as a whole.

Raw audio samples sent by the server are converted to the correct format and then written into an audio buffer in a single object on the stage in front of the user, which plays them back as a continuous stream of music. The Unity game engine provides support for three-dimensional sound, which initially led us to attempt to stream audio into individual buffers for each instrument simultaneously for a more realistic soundscape. However, we found that this approach led to significant synchronization issues, which led us to our current single audio source approach. Even with a single audio source, the music produced on stage sounds three-dimensional in the sense that if the user turns their head, they will hear the audio pan from one ear to the other. While this is a compromise, we consider it realistic enough for the level of immersion we wish to produce.

3.3 Interface Controls

Hand gestures are used in two different in-game states: instrument selection and music conducting. Gestures are detected using the Leap Motion VR package in Unity with code written by us specifically for this project. After instrument selection is complete, whenever a conducting command is detected in Unity, the command is sent to the composition server. Conductor controls include single instrument volume change, full orchestra volume change, music style selection, instrument role selection, instrument play and rest, and a separate mode for setting tempo. In instrument selection, grabbing an instrument is initiated when a hand is close enough to an instrument while that hand's thumb point is close enough to the point or joint of another finger on that hand. Once an instrument is grabbed, the instrument follows the motion of the hand until the thumb tip position breaks a different distance threshold from the other fingers' joints.

All other controls are available in the conducting state. Volume change, style selection, and role selection controls are determined by a mapping from the directions of the palms and the velocity of the hands in the same direction. Volume controls are mapped to one or both hands palm up or down, style change is a left hand facing right or a right hand facing left, and the role selection is mapped to one palm facing forward or one hand backward. Each set of controls has a different velocity threshold in the direction the palm is facing to trigger the control. Volume controls send a change in volume for the corresponding instruments, style selection sends the next style in the set of available styles available, and role selections sets the selected instrument as a background, foreground, or solo role, and sends that change to the server.

Play and rest controls are one hand controls. Rest stops the instrument from playing, and play makes it continue playing again. To tell an instrument to rest while it is selected, a fist is made with the single hand in scene. Fists are detected differently than the grabbing in instrument selection. The stop command checks if the positions one of the fingertips to the center of the hand is below a given threshold to detect a fist. Pointing is detected when the distance of the fingertip farthest from the center of the hand is a greater than the third farthest by a large enough margin.

Finally, the tempo controls allowed the user to set the tempo making hand motions similar to those of a real conductor. Making fists with both hands places other controls on pause while the user sets the tempo. Tempo is marked every time the hand or hands change from moving downward to moving upward. When the allotted time for taking the tempo is completed, the final tempo is set based on a number close to the median value of the recorded

times. The check removes most outliers caused by inaccurate readings or mistakes by the user. Pointing with both hands cancels the tempo recording at any time and returns the controls to the default set of volume, style and role controls.

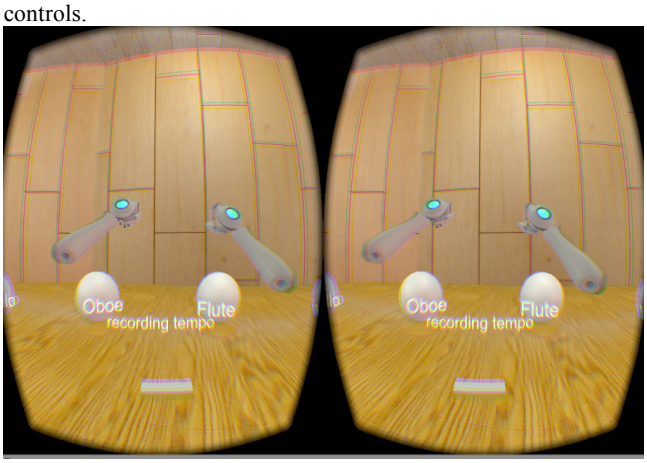

Figure 2. Tempo Recording

3.4 Composition System

The music composition system is written in Java on top of the JFugue library, which provides an interface to the Java midi library using music theory concepts such as note names, tempo, and scales. The composer is designed to generate a new sequence of notes to follow its previous generation like a stream. This way, the user can change the parameters of the currently playing composition. The generation of a sequence of notes takes two steps: 1) generating the chord and 2) selecting notes for each instrument.

Chords are generated based on probabilistic transitions between diatonic chord functions (tonic, subdominant, dominant) always starting with a tonic chord. The transition probabilities follow basic music theory rules on the likelihood of transitions. Once a chord function is chosen, a random chord is selected uniformly from an enumerated set of chords that fit within that function.

Notes are selected independently for each instrument. In order to ensure that the composition as a whole is harmonious, each instrument is given a role (melody, harmony, or bass line). This role influences how likely that instrument is to play a given note in the scale. For each note, a duration is selected by sampling from a discrete distribution of possible note lengths.

The pitch is selected by sampling from a probability distribution for the pitch which is created based on the previous pitch distribution, the previous played note, the instrument range, the current chord, and the instrument's role. The previous distribution and the last played note make the sampling more likely to result in a note close to recent notes, which leads to natural musical lines. The instrument range acts both as a limit on the range of notes playable and as a bias toward the center to lessen the likelihood of an instrument staying in an extremely high or low register for many notes. Note selection is repeated until the sequence of notes satisfies the length of the sequence for filling the buffer.

The composition system maintains an internal model of the state of the composition so that continuous compositions are coherent and so that the model parameters can be changed to reflect conducting control messages in between portions of streamed compositions.

3.5 Conclusion

Our evaluation metric was whether or not we have a working project. Does the algorithmic composition engine produce something that sounds even a little bit good? Does the front end correctly render the MIDI output of the back end into sound? And can the user's hand motions change the sound output of the program in the way we want? If the answers to any of these questions are "no", then we have more work to do.

Based on this metric we were successful in our project. However, we made many tradeoffs in the design of our system. Below are detailed descriptions of the major tradeoffs we made in this project.

1) Our controls were not as responsive as we had hoped. This was because the Leap Motion requires many hours of careful calibrating or large datasets of example gestures to create a highly precise gesture. Even with the most well designed gestures, the Leap Motion data is not precise enough to be a basis for a reliable interface.

2) Increased response latency due to lack of midi support in Unity. Algorithmic composition systems require a way to represent notes and instruments rather than raw sound. Midi is currently the best available audio representation system for this task. However, Unity doesn't officially support midi sounds. We worked around this issue by converting the midi composition to a raw sound stream. This creates a long delay between the user actions and corresponding changes in the generated music.

3) We compromised with an *almost* 3D soundscape due to synchronization challenges in playing each instrument as a separate audio stream.

There are improvements that could be made in every aspect of our system: the virtual immersion, responsiveness of controls, coherence, complexity, and versatility of the composition system, and system latency. Still, our system remains a novel and enjoyable system that effectively synthesizes computational composition with a natural modeling of conducting in a virtual setting.

4. REFERENCES

[1] WhiteVOID. 2014. Mendelssohn Effektorium –Conducting a virtual orchestra. http://www.whitevoid.com/#/main/interactive_structures/mendelssohn-effektorium.

[2] A. Nijholt, D. Reidsma, R. Ebbers, M. ter Maat, "The Virtual Conductor: Learning and Teaching about Music, Performing, and Conducting," Eighth IEEE International Conference on Advanced Learning Technologies, 2008.

[3] Fernández, J.D., Vico, F.: AI methods in algorithmic composition: a comprehensive survey. arXiv preprint arXiv:1402.0585 (2014)

[4] P.S. Langston, Six Techniques for Algorithmic Composition, Bellcore Technical Memorandum #ARH-01 3020, 1988.