# Mapping Sound Synthesis in a Virtual Score

Guy E. Garnett , Timothy Johnson, Kyongmee Choi
School of Music
University of Illinois. Nevada St, Urbana IL 61801
garnett, tejohnso, kchoi3@uiuc.edu

## ABSTRACT

The Virtual Score Project is currently running in a CAVE (Cave Automatic Virtual Environment) [3] located in the Beckman Institute of the University of Illinois in Urbana-Champaign. While a variety of research has been done in the CAVE to produce applications for science or industry, there has been relatively little artistic exploration. The CAVE has a highly developed and supported visual environment, but the development and support in the area of sound has been much less emphasized. One approach is described in [1]. Part of the impetus for this project is to bring these two things more into balance, and to explore the latent artistic possibilities in such a rich new technological medium. This paper describes the approaches to sound production we have recently been exploring.

## 1.1 Technical Description

*Cave*

The CAVE is a four-sided graphic environment encompassing three walls and the floor. Users wear special glasses which allow them to see images in three dimensions. The glasses have a MotionStar [7] sensor attached to them that sends out six data streams, the three positional axes Horizontal (X), Vertical (Y), and Depth (Z), which are measured in absolute distance, and the three orientation axes Azimuth (A), Elevation (E) , and Roll (R), which are measured in degrees. Users interface with the system by wielding a special wand which has three buttons, a joystick (push controller), and a second MotionStar sensor. The graphics and computation engine of the CAVE is an SGI named Cassatt equipped with an Onyx2 Reality Monster consisting of 12 250 MHz R10K MIPS CPUs, 9 Gigabytes of physical RAM and 4 InfiniteReality2 graphics boards. [5]

## 1.2 Overview of Program

The VirtualScore is designed to utilize the graphics and motion tracking features of the CAVE to create an immersive 3-D representation of a musical score. The user is inside the score and events flow past under user control. Each event is associated with certain sonic behaviors.

## 1. 3 Soniblui

Our graphics software, called Soniblui[1], is menu driven, and these options are selected using the leftmost two buttons on the wand. The rightmost button is used to initiate a sound event. As the user draws an image in the CAVE with the wand, the gesture is recorded by Soniblui in a linked list of nodes, and is immediately displayed as a colored object in the shape of the gesture, with the orientation axes of the wand recorded and rendered visible as short white lines protruding from each node. In addition to creating the individual note events, the joystick is used to edit the position of an event or group of events, or to adjust the display of a group of events. A grid is displayed on all four projection surfaces to allow the user to measure increments of time spatially (objects that are farther away are further back in time). As the image is displayed, Soniblui sends out relative position and orientation values received from the wand to our Macintosh Titanium G4 running Max/MSP. (The wand sensor values are scaled in relation to the glasses sensor, allowing the user to stand anywhere in the space.) Using Max/MSPs and the OSC protocol [6], the position, orientation, and button values are accessed by our Max/MSP patch, and a corresponding sound event is produced. The CAVE can theoretically handle eight channels of audio and our system was originally designed to exploit this, but was later simplified to two channels.

## 1.4 Edit and Performance paradigms

There are two user modes in our system: Edit mode and Play mode. In Edit mode the user creates and edits events with gestures. In Play mode the user performs the stored composition, shaping its temporal and dynamic unfolding using basic conducting gestures. As in [4] a beat detecting algorithm is used to calculate the tempo of the playback from the conductor/performer's gestures. During playback, the score moves past the user, whose position determines the current "now" of the score. In order to have multiple sounds playing simultaneously we have used the MSP object ~poly [8] . ~poly allocates a specified number of copies (voices) of all the patches contained within it. It works much the same way as synthesizer polyphony. This was necessary to accommodate playback of stored compositions. For our purposes, we found that having 16 voices was sufficient, although in the future much more computing power will be needed to fully implement our projected system. Our choice of FM [2] allows us to handle all the sound processing on our Macintosh. [9]

## 1.5 Prototype Development

Using the Max/MSP system made it very easy for us to prototype and develop different sonic responses to the gestural data we were generating. A few of these varied approaches are presented next.

In September of 2001 we began the CAVE and Virtual Conductor projects. The first major decision was to add OSC [8] to the CAVE app. This allowed us to transmit the wand sensor data

1 Soniblui was de veloped initially from a graphics only program called AlaskaBlui, which allowed a user to create 3-dimensional blue swirley shapes in the CAVE, without any sound.

directly and simply to the Macintosh running Max/MSP. In our earliest prototype we were using the absolute wand sensor values, that is, numbers referring to absolute locations in the CAVE. This forced us to locate and remember a position in the room where the user would be required to stand in order for our scaling ranges to work. This proved to be overly cumbersome, so code was added to calculate the wand sensor values relative to the head sensor. These relative wand sensor values were much easier to work with, and made the system much more user friendly. However, some parameters, mostly having to do with time points, are still referred to location in the time grid which is independent of the user. This is helpful becaue then the user can move around physically in space in order to gain access to different time locations.

### *1. Event updates vs. continuous control*

As in many realtime systems, we were faced with decisions about whether we would allow parameter updates continuously or only at discrete event boundaries: "note on" versus "continuous controller" data in MIDI parlance. In our first version of the system, the sound was continuous, and the user was able to play the space which represented one big FM sound environment. It was exciting to be able to shape the synthesis output gesturally, but we found this became tiring relatively quickly, and we wanted to have control over the initiation, duration and termination of a sound. So we decided to use the wand Button #3 to initiate and terminate an event, much like a mouse click. Even though we would still be receiving continuous data from the tracking devices, this gave us the opportunity to get the sensor values at the moment of initiation of the sound (which we refer to as instantaneous values). Using these numbers for some parameters makes it easy to keep certain aspects of the sound constant for its entire duration, without any special effort on the part of the user. Every time we receive a Button #3 On message we store the wand values in float boxes and bang them through once. Button #3 also allocates a new ~poly instance to handle the synthesis for that particular sound. The continuous values are coming through all the time, and for certain parameters we use these (mapping continuous Y values into frequency, for example).

In order to keep the synthesis demands to a minimum and focus on the scoring and control functionality, we developed a set of 8 FM timbres, which consist of fixed modulation index and amplitude envelopes, along with default harmonicity values and a fixed duration. This gives the user eight choices for timbral starting points, though many of the timbres behave similarly depending on the user's gestures. Later, three variations of the fixed modulation index and amplitude envelopes were developed for each timbre. In this version the user could toggle through this family of three envelopes by positioning the wand within one of three zones on the E (elevation) axis, each encompassing a 60 degree portion. We therefore use only 180 degrees out of the possible 360 since the wrist is not able physically to bend over backward, though of course we could have allowed for flipping the wand as a drum major twirls his baton. These envelopes added richness to the environment but were cumbersome and difficult to edit with all the hard-coded data. A set of Max patches was created to address this problem. Two fixed envelopes are created off-line and stored in a Max coll object in the FM synthesis engine patch. The FM engine interpolates between these fixed envelopes in real time. The original hard-coded envelopes were incorporated into this system. Now, by positioning the wand along the E axis, an FM sound is generated with interpolated envelopes for its modulation index and its local amplitude. Using these patches the user can easily create any number of fixed envelopes for any of the FM timbres. The user need only draw the desired envelopes and store them in the patch.

## 1.6 Gestural Mapping/Sound Synthesis

We have two signal processing engines: FM synthesis and sample playback. Most of the research into gestural mapping strategies was done with FM synthesis in mind. However, when we later added the sound sample option, we tried to keep our gestural model as similar as possible to the FM approach. Aiming at what would be most intuitive for the user, the gestural model evolved gradually. The mapping used in the first prototype was: frequency (X), amplitude (Y), modulation index (Z), panning (A), and on/off cue (E). We soon changed to frequency (Y), which, along with panning (A) are so intuitively natural that they have remained until the present. By trial and error we found appropriate scaling ranges for our FM parameters. Our design was guided by our desire to minimize any intuitive discrepancy between the gesture and the sonic effect, knowing that it was necessary to keep things simple enough for the user to be able to get meaningful results from their movement. Our second prototype featured the following mapping scheme (the Sampler synthesis portion of the table refers to an alternative synthesis engine that was developed as an alternative to FM):

Play mode :

X - overall dynamic control
Y - speed control ( ~pvoc [10] )
Z - reverb
A - panning

Edit mode :

FM synthesis :

X - carrier frequency
Y - amplitude
Z - modulation index/harmonicity
A - panning

Sampler synthesis :

X - time compression
Y - amplitude
Z - chorus effect
A - panning

Some things were hard to hear, like the chorus effect in sampler mode, and the values had to undergo a lot of tweaking. Other things seemed to work well from the start. In FM synthesis, it is difficult to predict the effects of certain parameter changes, such as harmonicity. It required a lot of experimentation to be able to reliably produce certain timbral qualities. In our original sampler mode we only had one pre-loaded sample. Now we have eight different samples, and , as in FM mode, changing the current sample or changing the FM timbre has to be done manually by someone at the computer keyboard. After our most recent spate of research we have settled on the following mapping arrangement for FM synthesis:

Table of Mappings for FM Synthesis (in Edit Mode)

Instantaneous Values:

| | |
|---|---|
| X | harmonicity quadrant (scaling range) |
| | global amplitude scaling vector (see below) |
| Y | global amplitude scaling vector |
| Z | reverb (dry/wet mix) |
| | global amplitude scaling vector |
| E | interpolated envelopes (see above) |
| R | harmonicity value |

Continuous Values:

| | |
|---|---|
| Y | carrier frequency |
| A | panning |

In our sample playback model, the sensor values are mapped into parameters that control the quality of the sample playback. The table of mappings is as follows:

Table of Mappings for Sample Playback

Instantaneous Values:

| | |
|---|---|
| X | chorus quadrant (scaling range) |
| | global amplitude scaling vector |
| Y | sample playback duration (frequency tuning) |
| | global amplitude scaling vector |
| Z | reverb (dry/wet mix) |
| | global amplitude scaling vector |
| E | low pass filter |
| R | chorus depth |

Continuous Values:

| | |
|---|---|
| A | panning |

In Sample Playback, X and R are used in the same manner as FM, with X delineating three quadrants of scaling ranges and R producing the specific value within a particular scaling. The mapping of Y is also analogous to the FM model: here it maps into a parameter that controls the perceived pitch of the sample. E has been mapped into a low pass filter allowing the user to lighten or darken the sound with a twist of the wrist, a simple but effective technique. The rest of the mapping is the same as FM. The mapping for both methods of signal processing was chosen to give the user intuitive gestural control over a rich sound environment that would be computationally inexpensive.

In an attempt to extend the capabilities of the system in an intuitive way we found it useful to pair position values with an analogous orientation value: (X/R, Y/E, and Z/A). In this scenario the position value would control the coarse range of values for a parameter and the analogous orientation value would control the specific fine values. The first FM parameter we applied this to was harmonicity, mapping into it from the horizontal axis (X) and its corresponding orientation value roll (R). We divided the user's portion of the X axis (the part that fits within an average arm span) into three quadrants. From left to right each quadrant represents a harmonicity range that is progressively more sideband-rich. So the first quadrant only allows a spectrum of harmonicity ratios that produce a range of FM sounds from very pure to mildly sideband-rich. The third quadrant encompasses fairly to extremely rich sounds (noise). The instantaneous X value sets the scaling ranges for the appropriate quadrant and the instantaneous R value gives us a specific value within that quadrant. This allows the user to work with a specific family of timbres without having to worry about minute changes in the X position drastically changing the sound. We found that mapping orientation values into expressive parameters like harmonicity allows the user to obtain a wide variety of timbres while keeping most other parameters the same. Another parameter besides harmonicity that controls the richness of the sound is the modulation index, which is mapped into from E (elevation). Again, this allows the user to change the quality of sound while other parameters like frequency, reverb, etc. remain the same. The range of E values (from -90° to 90°) corresponds to interpolated envelopes, using the previously mentioned Max patches. There are two sets of these envelopes, one for the modulation index and the other for the local amplitude envelope of the current FM timbre. Z (the depth axis) maps into reverb. The further away the wand is from the user at the moment the sound is initiated, the more reverb is applied to the signal. The raionale for this choice is simply our perception that dry sounds are close and wet sounds are distant. And finally we have an absolute distance vector mapped into global amplitude. This 'global amplitude scaling vector' is a vector computed using X, Y and Z which measures the absolute distance from the head sensor to the wand sensor. This value controls the amplitude of the sound just before it is sent to the ~dac, (the global amplitude). The closer you bring the wand to the head, from any direction, the quieter the sound.

## Future Work

Future plans include making more use of the available data. Besides the relative sensor values coming from the wand, we could also make use of the absolute values of both the wand and head sensors. (For example, we could divide the space into quadrants and base the synthesis paradigm upon which quadrant the user is currently occupying.) We will refine our playback model to more expressively scale the stored values in a composition (amplitude, reverb, panning, etc.) with conducting gestures. Another addition could be having the user enter a size value, reflecting their basic arm span (small, med, large) which will globally scale all of the individual scale objects and bring the system into the reach of the current user. And we are currently developing a voice control method of choosing these options, with the help of Tom Bonura of Apple computer. To make the system widely usable we need to be able to insure users have reliable access to a CAVE, give them the ability to take their work with them and create opportunities for performance within the practical means of a composer.

## ACKNOWLEDGMENTS

Thanks to the National Center for Supercomputing Applications, and the Critical Research Initiative of the Uniiversity of Illinois for their support of this project.

## REFERENCES

[1] Bargar, R. et al. Coney Island: Combining jMax, Spat and VSS for Acoustic Integration of Spatial and Temporal Models in a Virtual Reality Installation. Proceedings of the International computer Music Conference, ICMA, San Francisco, 2000.

[2] J. Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation," Journal of the Audio Engineering Society 21(7), 1973; reprinted in Computer Music Journal 1(2), 1977.

[3] Cruz-Niera, C., Sandin, D., DeFanti, T., Kenyon, R., and Hart, J. "The Cave Audio Visual Experience Automatic Virtual Environment," Communications of the ACM, Jun. 1992, vol. 35, No. 6, pp. 65-72.

[4] Garnett, G., Jonnalagadda, M., Elezovic, I., Johnson, T., and Small, K. Technological Advances for Conducting a Virtual Ensemble. Proceedings of the International Computer Music Conference. 2001, pp. 167-169.

[5] Leetaru, K. EasyDemo Maunal. http://easydemo.ncsa.uiuc.edu/easydemomanualpdf.pdf

[6] Wright, M. and Freed, A. "OpenSound Control: A New Protocol for Communicating with Sound Synthesizers." Proceedings of the 1997 ICMC, Thessaloniki, 1997. ICMA. pp. 101-104.

[7] See www.hi-res800.com/Info/MotionStar.pdf for specification and contact information

[8] See www.synthesisters.com/download/WhatsNewInMSP2.pdf for an introduction to the ~poly object and MSP.

[9] For a complete discussion of the basic features and rationale of the Virtual Score system see Garnett, G., et al., "VirtualScore: Exploring Music in an Immersive Virtual Environment." SIMS 02, UC Santa Barbara, June 2002.

[10] For details of the MSP phase vocoder patch ~pvoc, see MSP tutorials.