The Musical Blob

Nathan Armstrong
University of Washington CSE
armstnp@cs.washington.edu

Daniel Chesney
University of Washington DXARTS
chesnd@uw.edu

Andrew Foster
University of Washington CSE
agf3@uw.edu

ABSTRACT
In this paper, we explain the goals, development procedures, choices, and outcomes of 'The Musical Blob' project written winter 2012 for the University of Washington Computer Science and Engineering Sound Capstone. The project was intended as an experimental fusion of several different technologies for the purpose of creating a virtual instrument for live performance.

Categories and Subject Descriptors
J.5 [**Computer Applications**]: Arts and Humanities – *performing arts*; G.1.2 [**Numerical Analysis**]: Approximation – *approximation of surfaces*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *input devices and strategies*.

General Terms
Design, Experimentation

Keywords
limb tracking, sound synthesis

1. INTRODUCTION
The Musical Blob project began with the concept of an abstract instrument that could be manipulated to produce sounds. A performer would be given a visual interface for a virtual system or simulation. By manipulating the instrument, they could produce sounds that clearly correlate to their actions. The instrument would be abstract, and the relationship between the interaction and the sound would be hidden, requiring exploration to discover how to create an effective performance.

Implementing this would require a user interface that is easy to understand, intuitive to interact with, and dynamic enough to provide variety for the performer to work with. This also allows anyone to use the system: from casual users who can simply enjoy the interaction to professional performance artists who could learn to control the instrument more precisely.

2. RELATED WORK
There have been previous projects that have investigated the creation and use of virtual instruments. Mulder et al. [1999] created two different modes for manipulating a virtual surface using hand gestures and movement. In doing so, the user simultaneously modifies the parameters sent to a sonification system, which then produces sound based on various properties of the surface. We desired a similar form of interaction, but focused on a more general interactive experience, instead of focusing all interaction solely on hand gestures.

One of the problems we came across when designing the instrument was that of rendering a volatile, amorphous object. While some conventional solutions to this type of problem may provide superb visuals, they would not necessarily follow the performance constraints required for this real-time application. Lorensen and Cline [1987] developed an algorithm for polygonization of an isosurface, allowing a surface to be constructed from a grid of scalar values. Marching Cubes ended up having a huge impact, since it permitted us to display our instrument's model with variable precision, which in turn enabled us to balance between high-quality visuals and high-speed performance.

3. SOLUTION
3.1 Design Decisions
3.1.1 Instrument Design
When proposed, the instrument itself was designated as a 'blob' or amorphous object with highly mutable structure. This provided an interface that was easy to display, understand, and interact with. We chose to restrict the interactions available to the performer to whatever we could complete by the end of our eight-week development period.

3.1.2 System Design
With the basic concept in place, we chose to divide the system into discrete pieces (see Figure 1).

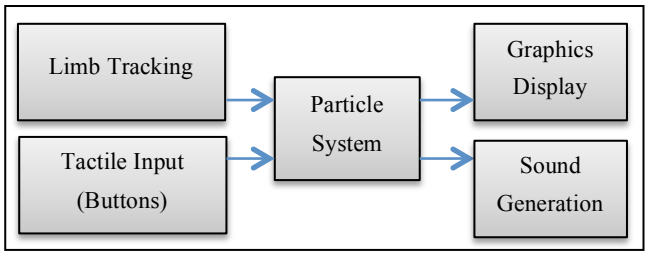

Figure 1. System Components and Flow

User input would be handled in one set of modules, which would summarize the performer's actions and provide necessary information to a particle system simulation. The particle system would use this input to manipulate the particles, and pass on appropriate data to the interface modules. One interface module simply renders the particles with the appearance of a contiguous blob, as well as any other visual data to aid the performer, such as the performer's limb positions. Another interface module uses various properties of the particle system as parameters for audio synthesis.

This design was chosen because it allowed us to be highly flexible and independent with decisions inside each module. For example, the particle system could easily be swapped out for another

system that uses different particle interactions. The graphics display could be changed to interpret the data in a different way. The audio systems can be switched between to use the parameters in different ways. As long as the data sent between each module was specified beforehand, we could easily make whatever decisions were necessary within our modules to create the greatest impact.

3.2 Implementation Details

3.2.1 Limb Tracking

In order to allow intuitive interaction with the blob, we wanted to be able to track the user's movements as one of the main inputs. Recent development with the Microsoft Kinect made it a viable option for interaction. We considered using a captured mesh of the user to interact with the blob, but the required library knowledge, installation and integration time was infeasible. We therefore chose to try tracking approximate positions of user limbs instead.

The library we used to accomplish this was *OSCeleton* [3]. The program captures data from the Kinect and uses it to track the approximate limb positions of multiple users. The data is sent via network packets to the receiving system, formatted as Open Sound Control (OSC) messages [4]. Open Sound Control played a vital role in several places in the project, providing a uniform and reliable data transfer interface.

3.2.2 Tactile Input

There are several operations or setting that we wanted to provide. We considered implementing these options by watching for user poses or gestures, but ultimately decided that those would be too easy to trigger accidentally, and would not necessarily be straightforward to capture. We chose instead to provide a two-part interface: a keyboard would be used for technical control, allowing direct access to core system functionality, and a set of Wii Remotes would be used for performance control, enabling the user to select modes of interaction with ease.

Keyboard interaction was straightforward to implement, but for connecting to the Wii Remotes, we selected the library *wiiuse* [5]. The library provided a way to connect to and poll the remotes, and left potential for future development using the built-in accelerometer. In the final implementation, the remotes were bound to specific limbs, enabling per-limb interaction.

3.2.3 Particle System

The particle system was entirely custom-written for the project, and used a simple physics system to form particle interaction. Particle systems were selected over the manipulation of premade models due to the relative simplicity of integrating user interaction, ease of display, and capacity for parameter extraction. At each simulation step, data about the particles – such as position, acceleration, centroid, etc. – are sent to the output modules via Open Sound Control packets.

3.2.4 Graphical Display

The particle data is taken in here and rendered to a visible format. Based on the concept of *metaballs*, each particle is given some field of influence in the form of a scalar function of distance to the particle. A surface can be formed by tracing the function along a specific value, or *isolevel*, forming an *isosurface*. When two particles approach each other, their fields merge, allowing a surface to adapt smoothly around both particles, and giving a sense of fluidity in the visualization. However, in order to find and render an isosurface, a method is needed to take the sum isolevels generated by all particles and transform it into a set of polygons that can then be easily rendered by a graphics engine, such as OpenGL. We selected to use the *Marching Cubes* algorithm for this purpose after finding a working implementation that seemed to be efficient enough for our purposes. One optimization we were able to make that allowed higher-quality visuals involved separating particles into groups which would be rendered together, allowing the normally coarse granularity of Marching Cubes to be made much finer to fit the size of each set of particles.

3.2.5 Audio Synthesis

Sound was synthesized and/or processed in real time with the SuperCollider language and environment. Several audio modules (patches) were created for use with the project, each accepting the same set of data from the particle system as parameters. Some of the patches also utilized live audio input from a shotgun microphone placed near the user.

Several patches made use of granular synthesis. This method chops an audio source (either synthesized by the program or sent in through a microphone) into smaller pieces called grains and replays those grains many at a time. Variations in each grain's duration, pitch, and envelope (attack, sustain, release) change the overall sound, as does controlling the number of grains that can be played at the same time.

3.2.5.1 A Detailed Example

What follows is a more detailed explanation of one of the patches, by way of example. It does not use granular synthesis and consists of two distinct sounds: a continuous synthesized pad and scattered percussive samples.

The source for the pad is a *UGen* (unit generator) that generates impulses at a given frequency and with a specified number of harmonics. There is one instance of the pad corresponding to each particle on the screen. The X position of each particle maps to the frequency of its corresponding UGen, rounded down to every 4th semitone. The Y-distance of each particle from the centroid of the system is mapped to the amplitude of the corresponding sound. The signal is then sent through a resonating filter with hard-coded cutoff and bandwidth settings.

The percussive sounds are taken from a buffer containing a recording of a single note played on an out-of-tune clavichord. The sound is triggered by any particle's impulse magnitude exceeding a hard-coded threshold, and the playback speed is determined by that particle's Z position.

All output from this patch is sent through a reverb bus to make the final sound less raw.

4. CONCLUSION

Our goal was to create a modular, tactile, interactive synthesizer, and we succeeded in creating one. This success is tempered by the rough look and feel of the system. Functionally, the project is complete, but aesthetically it leaves a lot to be desired. The display should ideally display a representation of the player or at least the player's hands. Sonically, the modules could each use some work to respond more dramatically to typical user input. We could collect usage statistics to better calibrate our audio modules. Modularity is a good thing, but the musical blob is spread across so many different interconnected technologies that any number of unexpected problems can crop up at any time. Consolidating would have made the project more practical and portable, but given our time and budget constraints, we worked with existing hardware and libraries as much as we could.

5. REFERENCES

[1] Axel G. E. Mulder, S. Sidney Fels, and Kenji Mase. 1999. Design of virtual 3D instruments for musical interaction. In *Proceedings of the 1999 conference on Graphics interface '99*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 76-83.

[2] William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.* 21, 4 (August 1987), 163-169. DOI=10.1145/37402.37422 http://doi.acm.org/10.1145/37402.37422

[3] github, 2012. OSCeleton repository. https://github.com/Sensebloom/OSCeleton

[4] Open Sound Control, 2012. Open Sound Control. http://opensoundcontrol.org/

[5] SourceForge, 2012. wiiuse repository. http://sourceforge.net/projects/wiiuse/