Using Pd to create a visualization instrument for percussion

John Harrison
Hack.Art.Lab (HAL)
Wichita State University
824 W. University Ave.
Wichita, KS 67213
john.harrison@alum.mit.edu

Abstract

In June of 2010 Michael Holland, director of Vortex Percussion Ensemble, approached Wichita-based art collective Hack.Art.Lab and contemporary composer Mary Ellen Childs to explore what synergy might exist between us through collaboration. To open this conversation, we targeted an earlier work of the composer's, *Still Life*, to build a live video element which might add another dimension to her piece. We chose Pd-extended as our engine to interpret sensor data from the performers, then manipulate the live video. In this paper I explore both the process and product of creating the visual element..

Supporting documents, including a video of a performance of the work, interviews of the creators, and links to all software used, are available at http://hackartlab.org/pg/groups/139/vanderbilt-collaboration/.

Keywords

wiimote, ps3eye, collaboration, video

1 Introduction

Hack.Art.Lab (HAL)[1] is an art collective based in Wichita, Kansas USA. Formed in 2008, group membership consists of about a dozen artists, engineers, and computer programmers. We have shown work throughout the US as well as at the 2009 Pd Convention in São Paulo. One of the strengths and challenges of HAL is that our individual members do not come together as a unified voice, so collaboration is not easy for us. At the same time, the group defines itself and its work through collaboration; most of the work we show was both born and exists because because of cooperation among several members and is work that could not have been realized by a single member. Resultantly, a main focus in our work is the collaborative process of the work itself.

Mary Ellen Childs[2] is an established Minneapolis-based composer. Although her works are typically for traditional western instruments, she also incorporates strong visual and theatrical components. Mary Ellen Childs is also known for forming and directing the percussion ensemble CRASH[3]. CRASH is a percussion ensemble with a strong theatrical element. Instruments may be played in unusual ways, stage props may be added, and the motions of the players are all a significant part of CRASH's performances.

Former member of CRASH and current director of VORTEX percussion ensemble, Michael Holland contacted Mary Ellen Childs and HAL to explore the possibilities of collaboration. To open this exploration, we chose her 10-minute work, *Still Life*, written in 1986. Using Pd-extended, I elected to program the new visual elements. Other members of HAL worked with me to generate the initial ideas and help develop the hardware needed. Mary Ellen Childs and I refined the final presentation and named the new work *Still Life: Revisited*.

Still Life: Revisited was performed by VORTEX on April 3, 2011 and the Wichita State University Percussion Ensemble on May 4, 2011. The performance at Wichita State University, along with interviews of the creators and performers, was archived by Wichita State University and is available with our supporting documents referenced in the abstract..

2 Video Elements

2.1 Artistic Challenge

Adding live video to a pre-existing audio work turned into more of a significant challenge than HAL initially anticipated. How could any video truly extend the work rather than being a tacky add-on without significant artistic merit? We certainly did not want our video to offer any analogy to any of the empty visualizations already offered in already-existing media players, such as Windows Media Player. At the same time, we recognized that the work was complete already and did not need a live visual element to succeed. How could we add video to the work in a way that connects to the performers meaningfully and did not compete?

2.2 Exploration

To explore this question I initially made videos compiling some of the many manipulations possible in Pd-extended and uploaded them to YouTube for others to review. However, our conversations proved most productive when we switched to focusing on other live video works rather than a discussion of possible video effects.

Eventually, inspired in part by the work of Cyrille Henry[4], we chose to explore a completely synthetically generated environment, non-deterministic in nature, and yet with organic, natural qualities of motion. We wished to stay away from obvious visual associations, preferring instead to explore abstract qualities so as not to have content that was too specific.

We wanted to create something which felt “alive” and which would offer unpredictable variety within a tight set of constraints. In this way, our visualization instrument might be considered analogous to a more traditional acoustic instrument.

3 pmpd

Perhaps for his own artistic exploration, Cyrille Henry developed Physical Modeling for Pure Data (pmpd)[5]. In Cyrille's words: “pmpd is a collection of [...] objects [which] provide real-time simulations, specially physical behaviors. pmpd can be used to create natural dynamic systems, like a bouncing ball, string movement, Brownian movement, chaos, fluid dynamics, sand, gravitation, and more.”[6]

What seemed powerful about pmpd for us was that we could create objects, relationships and an environment with physical properties such as mass, gravity, and spring damping. Then, by exerting forces or otherwise altering the environment in some way, we could create natural, fluid motions. Since we were not describing the motions directly, the motions might be more organic in their behavior and offer a certain natural unpredictability or chaos within their ordered motion.

Following Henry's model in his own works with chdh such as Adaptations[7], we chose to work with large numbers of primitive objects --- lines and spheres in our case. Through simple manipulation of the environment, we could create complex expressive motion of these lines and spheres as a visual contribution to the piece.

4 Sensors

It was obvious to us that live sensor data from the musicians was needed to effect the environment. The question then became what data we would collect, what sensors we might use to collect that data, and how we could map that data meaningfully for the work.

4.1 Accelerometers

Accelerometers attached to drum sticks seemed like a powerful way to capture both the motion of drum sticks and their impact on drums. Moreover, various cookbook implementations already exist to capture accelerometer data into a computer using a microcontroller such as an Arduino [8][9]. Even commercial off-the-shelf products such as V-beat Drumsticks[10] exist which could perhaps be re-purposed to import data to a computer.

We quickly ruled out V-beat drumsticks after trying a pair as our performers considered the V-beats not of a quality acceptable for their use. And, as wires connecting the performers to a computer would limit the performers' movement, we wanted to consider only wireless implementation. It would be possible to connect an Arduino and accelerometer to a wireless transmitter such as an Xbee[11]. but perhaps an easier and less expensive way to achieve similar results might be to modify a Wii Remote Controller (wiimote) and Wii Nunchuck[12] for our purposes.

4.2 Wiimote and Nunchuck

A wiimote has a built-in accelerometer and a Bluetooth transmitter which can send the accelerometer data to a computer with a Bluetooth receiver. A Nunchuck also has a built-in accelerometer and physically attaches to a wiimote so that the Nunchuck's accelerometer data can also be sent through the wiimote's Bluetooth transmitter. With one wiimote and one Nunchuck, we could capture accelerometer data from both hands of one performer.

We could not capture meaningful accelerometer data by attaching the wiimote and nunchuck to the upper arms of the performers. However, the devices were too cumbersome to attach to the lower arms, hands or sticks. To overcome this problem, we made a small circuit board consisting only of pads for the legs of the accelerometer IC itself. After soldering new accelerometer ICs to our boards, we then removed the original accelerometers from their pads on the wiimote and Nunchuck and soldered long wires to the original pads. We soldered the other ends of the wires to our small board so that the wiimote and Nunchuck would receive their accelerometer data from our accelerometers outside of the wiimote and nunchuck casing. Our board and IC measured less than a centimeter and were extremely light. We attached the accelerometers to the performers' sticks, then ran the wires to the backs of the performers were we strapped the wiimote and Nunchuck.

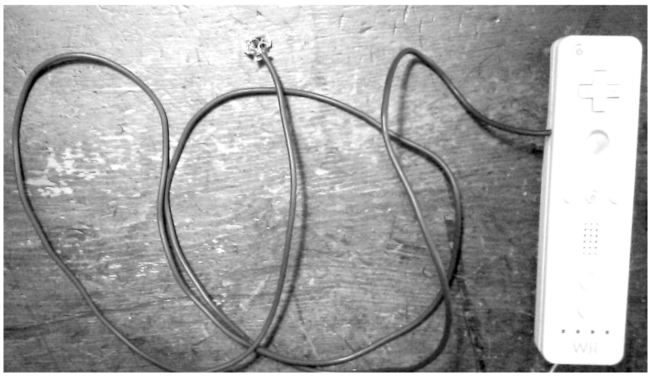
Illustration 1: Wiimote with external accelerometer

4.2.1 [wiimote]

Initally we tried Pd's [wiimote] object[13] for capturing accelerometer data from the wiimote and Nunchuck. We found that object caused instability with Pd when the instance was under moderate to heavy CPU load. We then switched to Linux Laptop Orchestra's [disis_wiimote][14], since it uses a threaded implementation. Its performance was stable.

4.2.2 wiimote Accelerometer Hardware

The wiimote uses an ADXL330 accelerometer which can measure between +/-3g. The Nunchuck uses an LISL02AL accelerometer which can measure between +/- 2g. However, both accelerometers are pin-compatible with each other and are also compatible with the ADXL326BCPZ, which can measure +/-16g. We opted for this ADXL326 so that we could measure significant drum impact.

Our alterations to the wiimote and Nunchuck along with using the [disis_wiimote] object seem like a successful way to capture the motion of drum sticks as well as their impact on drums. However, there were concerns about latency and sampling rates. We continued exploring other methods and eventually did not pursue this method for this project.

4.3 Multiple Microphones

Electret microphones are an inexpensive, easy way to capture audio data when the audio quality of the captured sound is not critical. To capture both the attacks and resonances of the individual instruments, we made a simple circuit to power 6 electret microphones using USB power. We connected the microphones to a Delta 1010LT PCI sound card[15] and experimented attaching the microphones to different parts of the various percussive instruments. We then had the performers rehearse, observed them and compared what we observed from the performers with the data captured by the microphones.

We expected that we could detect what instrument had been struck at a given time by comparing the amplitudes of the microphones and choosing the instrument whose microphone showed the loudest amplitude, but this turned out not to be the case. Perhaps due to room and instrument resonances, we got inconsistent results. Sometimes a nearby microphone would show an initial amplitude higher than the microphone attached to the instrument which had been struck.

We might have solved this problem by comparing the exact time at which each microphone detected a sharp amplitude increase. In theory the one showing the increase first would be the one that was struck first. Instead we chose to use only one electret microphone placed near the performers. It successfully captured the attacks and amplitude of the entire ensemble. We also incorporated other methods for capturing data for this project.

4.4 ps3eye

The Ps3eye camera has become extremely popular in the hacker community because of its low price and excellent performance. Using open source drivers, it is possible to get this camera to capture 320x240 resolution at 125 fps. For our computer, an i7 running 32-bit Ubuntu 11.04 (Natty), it was necessary to recompile the provided webcam drivers to achieve these performance results[16].

4.4.1 Near-infrared Tracking

Looking for a way to more accurately track the exact motion of the sticks, we attached four infrared LEDs around the diameter of the sticks and modified the webcams to detect only near-infrared light[1][18]. Using Gem's [pix_multiblob] and one webcam we could track a performer's set of two sticks. [pix_blobtracker] uses the resulting iemmatrix to keep track of which blob represented which stick.

The infrared LEDs on the sticks were powered by CR2032 button-cell batteries. Using a simple battery holder attached to the sticks, these batteries could be directly connected to the LEDs without resistors. They would successfully power the LEDs for about 1 hour. With this setup we were able keep our alternations to the sticks small enough and light enough that they could still be used by professional performers.

Illustration 2: drum stick with four infrared LEDs

5 Pd implementation

5.1 Communication amongst multiple instances

Capturing and analyzing the data from a single webcam running at 125 frames per second (fps) created a significant load for 1 core of our i7 CPU. The high fps was necessary for minimal latency but the draw on the single core did not allow much room in that core for more analysis or functionality.

Pd is a single-core application, so more generic examples of this problem are common and typically solved by running multiple instances of Pd. The Host OS will then shift the various Pd instances among the available cores. In our case, 3 Pd instances could each be independently responsible for capture and analysis of each of the 3 webcams. They could then transmit this information to a master Pd instance, which could then interpret the data and show the video.

Typically such instances are independently run, perhaps by a Bash script, then communicate to each other or another instance using sockets. More recently, Miller Puckette introduced the [pd~] object to achieve similar results.

5.1.1 [pd~]

Initially, I tried using the [pd~] object, which took some (undocumented) fumbling to get to work with pd-extended. However, I opted against this approach as it requires audio processing to be running in the master patch and this appears to create a bottleneck, limiting the performance of the video rendering in Gem. And, although not relevant for this particular project, [pd~] also works only in distributing information instances of Pd running between multiple cores on a single computer and not between instances of Pd running on multiple computers networked together.

5.1.2 [inter-inst-comm]

With earlier projects, I have split tasks among cores by using the earlier-described method of creating multiple instances of Pd which communicate using sockets. While this is a successful method, implementation of this method from scratch can be cumbersome and error-prone. To help address implementation for this project and future projects I built a generic abstraction `[inter-inst-comm]`.

To use the abstraction `[inter-inst-comm]`, first initialize communication in each patch by adding the object: `[inter-inst-comm <id#> <send-socket> <rec-socket>]`. The id# is user-assigned. For example, if two patches need to communicate to each other using sockets 6000 and 6001, to initialize one patch might contain `[inter-inst-comm 1 6000 6001]` and the other might contain `[inter-inst-comm 1 6001 6000]`.

If the first patch also needed to communicate to a third patch using sockets 6002 and 6003, initialization for this in the first patch might look like `[inter-inst-comm 2 6002 6003]`. In this way, the id# allows the first patch to make a distinction between communicating with the second patch and communicating with the third patch.

After the initialization object exists in a patch, transfer a message from one patch to another patch by adding `[s2i <id#> <message-name>]` to the patch. For example, to send a message named `my-message` between the two patches we have already initialized, add `[s2i 1 my-message]` somewhere inside the first patch. Then the second patch will automatically receive the message with a simple `[r my-message]` when the first patch sends a message `[s my-message]`. An example set of patches further demonstrating `[inter-inst-comm]` is available with the support materials for this paper.

1Not all ps3eye webcams are able to be modified in this way[17].

5.2 Dynamic Patching

5.2.1 Motivation

After some experimentation we decided that our visualizations would consist of three independent tree structures consisting of lines and spheres. For full control, we would need the ability to address the properties of each of the lines and spheres in each tree independently. Moreover, each tree structure would contain its own environment for gravity, damping and other forces and properties. A "master" environment would effect properties for all the trees.

Our concept meant that the trees would each be able to dynamically generate their lines and spheres as well as change their properties through messages. Such messages could be generated from analysis of input signals from the microphone and the three ps3eye cameras.

5.2.2 Implementation

I created an object which consisted of the properties needed for any generic sphere and line. I then dynamically generated instances of that object using dynamic patching at run time. As I needed in performance, I would generate or destroy the instances of the objects during performance.

While this implementation worked, perhaps a better way might have been to dynamically generate all the needed objects at creation time as opposed to run time, then turn the objects on and off as needed. This latter method prevents hiccups that can be caused by the graph regeneration that Pd does when a new instance of an object is created. And although I have not experienced stability issues with dynamic patching, any stability issues that might exist with this unsupported feature will likely show up before the performance instead of during the performance.

5.3 Text files and [qlist]

After creating an interface which could create nodes and assign properties, I needed a way to change multiple properties simultaneously for different sections of the piece. To solve this I used Pd's [qlist] and devised a simple set of commands which my patch could interpret. I stored the commands in a text file then recalled them through the interface of the Pd patch.

As I found parameters for the environment that worked for a particular section in the piece, I wrote them in these text files as commands. Examples of some of the commands are shown in Table 1:

Command	Description
1-attack-on 0;	Send 0 to message named 1-attack-on
if-debug show-nums 1;	if the debug parameter is set to 1, send 1 to message named show-nums
#This is a comment;	That was a comment
repeat 80 1-create-node;	send `[bang(` to 1-create-node 80 times
2-connect-attack-to 2-r-create-node;	take the output of the message named 2-attack and connect it to the input of message named 2-r-create-node
slide 2-r-mass-alpha-slider 0 1 8000;	Using `[line]`, slide the value of 2-r-mass-alpha-slider from 0 to 1 over 8000ms
250 slide 1-s-mass-size-slider 4 8 250;	wait 250 ms then, using `[line]` slide 1-s-mass-slider from 4 to 8 over 250 ms

Table 1: Commands in text files

6 Future Work

From a technical level, this project served as a good exploration of pmpd and dynamic patching. It provided a framework which allowed for testing using an interactive interface. Multiple states could be recalled using text files.

However, while states could be typed into text files to be recalled later, there was no system to save states in a text file. Moreover there was no way to save the states or properties of the objects themselves, only the environment.

From an artistic sense, exploring synthetically generated images in a physically modeled environment felt successful. It was also interesting to explore how to successfully add video to a pre-existing audio work. Moreover, the semi-transparent screen in front of the performers did offer some cohesion between the players and the video. It would be interesting to embark on a similar project as this, but build the video concurrently with the audio rather than having one follow the other.

In addition, there are a multitude of possibilities for projecting other than onto a screen and with a rectangular surface.

7. Conclusion

In creating the work *Still Life: Revisited*, we explored collaboration, hardware interfaces and software construction all to serve a pre-existing audio work. By using Gem and pmpd, we created organic motion of OpenGL primitives to create expressive live visuals complementing the music and the motions of the players. A semi-transparent scrim in front of the players helped us balance the visuals with the performance of the original work. In developing the work, we furthered our understanding of hardware interfaces and some software techniques.

8. Acknowledgements

Michael Holland initially introduced the idea of this project and our collaboration. Mary Ellen Childs wrote the original piece, and was fundamental in the generation and refinement of the original concept. Hack.Art.Lab, and especially Ann Resnick and Kristin Beal-DeGrandmont helped generate meaningful discussion in the development of the work artistically. Lauren Hirsh and Ivy Lanning helped in the building and placement of sensors.

Of course, the work also would not have been possible without the tireless efforts of the Pd community, including but not limited to Miller Puckette, Hans-Christoph Steiner, Iohannes Zmoelnig, and Cyrille Henry. Besides all of the wonderful code these people have produced, they have been most helpful in offering technical insight on the Pd mailing list.

References

[1] Hack.Art.Lab. http://hackartlab.org

[2] Mary Ellen Childs. http://www.maryellenchilds.com/

[3] CRASH. http://www.maryellenchilds.com/?page_id=8

[4] Cyrille Henry. http://www.chnry.net

[5] pmpd. http://www.chnry.net/ch/?120-Pmpd

[6] pmpd documentation. http://www.chnry.net/ch/IMG/pdf/pmpd.pdf

[7] Adaptations. http://www.chdh.free.fr/spip.php?article1

[8] Arduino. http://www.arduino.cc

[9] MEMSIC 2125 Accelerometer. http://www.arduino.cc/en/Tutorial/AccelerometerMemsic2125

[10] V-Beat Drumsicks www.firebox.com/product/2023/V-Beat-Drumsticks

[11] Xbee. http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/

[12] Wiimote and Nunchuck. http://www.nintendo.com/wii/console/controllers

[13] [wiimote]. http://thiscow.eu/tiki-index.php?page=puredata-wiimote

[14] [disis_wiimote]. http://l2ork.music.vt.edu/main/?page_id=56

[15] Delta1010LT. http://www.m-audio.com/products/en_us/Delta1010LT.html

[16] ps3eye Linux driver. http://bear24rw.blogspot.com/2009/11/ps3-eye-driver-patch.html?showComment=1273457202061#c5812783852330333262

[17] Good and bad lenses for the ps3eye. http://peauproductions.blogspot.com/2009/04/two-types-of-ps3-eye-stock-lenses.html

[18] Modify a ps3eye webcam for near infrared light. http://www.forcepoint.net/main/?p=24