Sarlacc

Shawn Lawson
Rensselaer Polytechnic Institute
lawsos2@rpi.edu

Ryan Ross Smith
Rensselaer Polytechnic Institute
ryanrosssmith@gmail.com

ABSTRACT
Sarlacc, an audio-visual performance, features visuals live coded within the OpenGL fragment shader, that are reactive to incoming audio frequencies parsed by band, beats per minute, and Open Sound Control data. The sound component is performed using Ableton Live and analog synthesis.

Author Keywords
OpenGL; shaders; live coding; performance; audio-visual; Open Sound Control

ACM Classification Keywords
J.5. Arts and Humanities: Arts, Performing arts. D.2.6. Programming Environments: Interactive environments. I.3.6. Computer Graphics: Methodology and Techniques: Interaction techniques.

INTRODUCTION
Sarlacc, the second audio-visual performance piece created by the authors, was developed during a short, intense residency period where iterative improvisational sessions became the foundation of the creative process, and ultimately, the final work. This paper will briefly discuss tool design and interaction, aesthetic concerns, and the creation process of *Sarlacc* in general.

TOOLS
Sarlacc was developed and is performed using both author-developed software and consumer-available software and hardware.

Visuals
The visual component of *Sarlacc* is created in an integrated development environment (IDE) that runs as a webpage in the Google Chrome web-browser (see figure 1). The IDE, from top layer to bottom layer, includes: an interface written in JavaScript with JQuery, a text editing area using the ACE library, and lastly a 3D canvas element for WebGL rendering [1, 2, 3].

C&C '15, June 22-25, 2015, Glasgow, United Kingdom

ACM 978-1-4503-3598-0/15/06.
http://dx.doi.org/10.1145/2757226.2757373

Live coding is, generally speaking, software editing in real-time, generally at a low-level [4]. The live coding of *Sarlacc's* visuals occur within the aforementioned text editor of the IDE. The text editor contains OpenGL fragment shader code associated with a timer set to automatically compile the shader code every 200 milliseconds [5]. If the code is compiled without errors, it is loaded onto the graphics card and used to draw the imagery in the canvas layer. To this end, successful edits to the shader code are seamlessly reflected in the visuals, without the need for compile, play, or execute buttons. Coding errors are highlighted, while the most recent successful edit will continue generating the visuals.

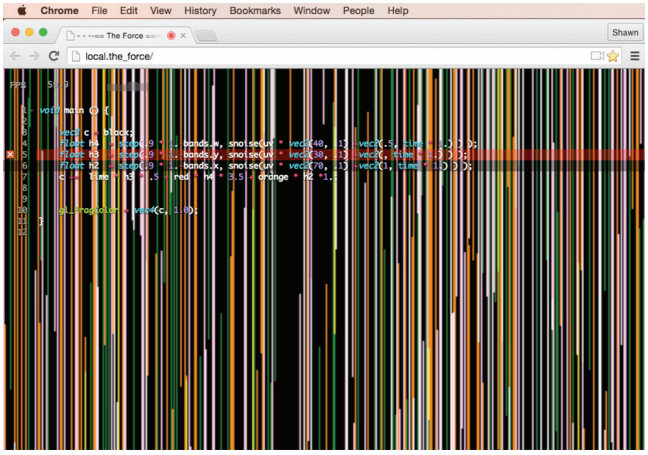

Figure 1. IDE for performing visuals runs as a webpage in Google Chrome.

The WebKit web-browser engine contains functionality for accessing any audio/microphone input to the computer [6]. The IDE uses this audio input and WebKit's frequency analysis tools to create four frequency bands: low, middle-low, middle-high, and high. This frequency band data, supplied by the audio performer, is passed into the OpenGL shader, providing a steady stream of variable control data.

For further integration with the audio an Open Sound Control (OSC) network is created between the audio performer and the visual performers computers [7]. The audio performer synchronizes sample and scene launch from Ableton Live with a Max/MSP application to send triggers synchronized to beats per minute (BPM) over the OSC network. The visual performer's IDE parses these OSC packets and makes the BPM data accessible from within the OpenGL shader.

Finally, an additional OSC network is created between a device (phone or tablet) that is running a custom-designed slider interface created with Lemur, and the visual performer's computer [8]. The Lemur app virtualizes on screen a conventional MIDI-like input device, including a set of sliders, knobs, and buttons. The visual performer uses this interface to facilitate direct control over time-sensitive visual modifications, smooth blending of on screen elements, and to *fine tune* the aesthetic feel.

Audio

The audio component features a set of pre-composed elements available for localized and global triggering, from one-shots to entire sequences, with those most salient elements available for real-time modification. The audio component also features the use of analog synthesis and other outboard gear for use as a transitional supplement, noise source, and to provide direct access to the visual element through the instantiation of discrete, discernable frequencies.

AESTHETICS

The aesthetic goals of *Sarlacc* were to construct an audio and visual build-narrative merging elements of EDM, J-Pop, Gif-Culture and mechanical and electronic-inspired audio and visuals.

The visual aesthetic experience is unequal parts pop-culture, abstract expressionism, and glitch-art. Specific to the creation of *Sarlacc*, the capability to draw sprite sheet animations was implemented into the IDE. This add-on allows the visual performer to use animated sprites, as source material in the performance. This expanded the capability of the visual imagery to include the iconic image of the *nyan cat* (see figure 2).

PROCESS

The sustained creative focus of the one-week-residency used to create *Sarlacc* revealed a new and exciting process for the authors: iterative-improvisation.

Prior to the residency the authors' process was to create small parts independently, before meeting to create short bursts of audio-visual mashups toward the discovery of audio-visual combinations that fulfilled our aesthetic goals, facilitated performance flow from one section to the next, and provided a sequential logic of sorts; a non-narrative story arc.

The constraints of the residency led to the iterative-improvisation method out of necessity, while still framed by several previously-agreed upon *aesthetic* constraints. The authors would improvise together with a single visual and audio segment until it was collectively felt that the segment had solidified into a singular audio-visual entity. Once all segments had been completed, sequence and transitions were subjected to a similarly improvisatory method.

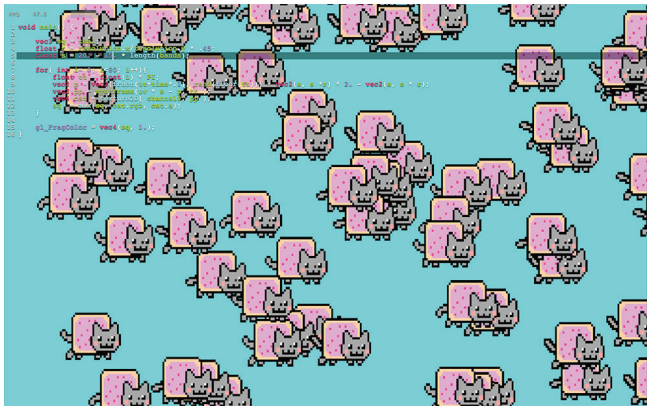

Figure 2. Example of Gif-Culture, sprite sheet animation.

CONCLUSION

Portions of the IDE were very much in play during the development of *Sarlacc*. The points at which these technologies turned from roadblocks to opportunities allowed for the iterative-improvisation method to flow smoothly.

At the end of the residency, the authors noted that the iterative-improvisation method shortened the development time of the performance work from two months to three days, and the sustained one-on-one creative contact forced a continued reflexive feedback ultimately resulting in a stronger, more cohesive work.

ACKNOWLEDGMENTS

Large thank you to CultureHub in New York City for providing us the time and space to experiment and create *Sarlacc* [9].

REFERENCES

1. JQuery library for Document Object Model (DOM) manipulation of html. http://jquery.com.
2. ACE library for text editing in a web-browser. https://github.com/ajaxorg/ace.
3. WebGL library of OpenGL for drawing graphics in a web-browser. https://www.khronos.org/webgl/.
4. TopLap, home of live coding. http://toplap.org/about/.
5. OpenGL library for drawing graphics. https://www.khronos.org/opengl/.
6. WebKit the web-browser engine in Google Chrome. https://www.webkit.org.
7. Open Sound Control (OSC) communication protocol common to music/media performance. http://opensoundcontrol.org.
8. Lemur, for creating and using OSC interfaces https://liine.net/en/products/lemur/.
9. Culture Hub, incubator for arts and technology. http://www.culturehub.org