Mosaic, an openFrameworks based Visual Patching Creative-Coding Platform

Emanuele Mazza **María José Martínez de Pisón**
Laboratorio de Luz, Universitat Politècnica de València
emanuelemazza@d3cod3.org mpison@pin.upv.es

ABSTRACT

Mosaic is an open source multiplatform (osx, linux, windows) live coding and visual programming application based on openFrameworks. This paper describes the initial ideas that have driven its development, its internal structure and the basic libraries it is comprised of, the levels of complexity that can be developed with Mosaic, (from beginner, to very complex developments) and the main contributions to the field of live coding/visual programming and also to the field of creative coding teaching/learning.

We present the development framework, which integrates two paradigms: visual programming (diagram) and live coding (scripting), to show its features and potential, most significantly, the learning feedback generated through to the relationships it establishes between human-machine. In other words, amplifying access routes to that relationship/interrelation promotes augmented-thinking through feedback.

Keywords:

Mosaic, openFrameworks, live-coding, visual-programming, creative-coding teaching/learning

1. INTRODUCTION

Mosaic is a visual programming and live coding environment, based on openFrameworks. In the data flow environment, the possibility of programming in different scripting languages (Lua, Python, BASH, GLSL) is inserted through specific objects that can be activated simultaneously (see Figure 1).

Mixing visual diagram and scripting concepts (diagram and language) allows us to combine the possibilities of two media to offer variable learning curves for variable user levels. Visual programming helps to understand and visualize schematically what the machine is doing (Tanimoto 2013, 31-32), because objects show what's happening with the information flows being generated or modified, which provides a more intuitive and flexible feedback. Scripting, on another level, allows us to explore and understand the specific processes that are involved in more detail and more deeply. This hybridization depicts a fertile two-way cross-pollination situation that amplifies the human-machine communication, providing something similar to the benefits introduced by the Dual Coding Theory (Paivio 1990, 57).

The project, at alpha stage right now, is being developed through a modular structure of independent ofxAddons. For example, ofxVisualProgramming is the central code for visual programming, and is kept isolated to encourage contributions, simplify error correction and improve the quality of the code, but most of all to encourage a collaborative community-driven development of Mosaic project.

Describing all the Mosaic characteristics could be a very complex task (OF and ofxAddons ecosystem is extremely large, and the possibility of multiple scripting languages increases its complexi-

ty), so the idea behind the interface design is to avoid such "highly complex" situations, embodying a direct and natural interface of a drag&drop connect/disconnect interface (mouse / trackpad) on the most basic level of interaction, adding editing code (keyboard) at an intermediate level (scripting), and up to the most advanced level for experienced users (external device communication, automated interaction, etc.). The main features of Mosaic are summarized in the following sections.

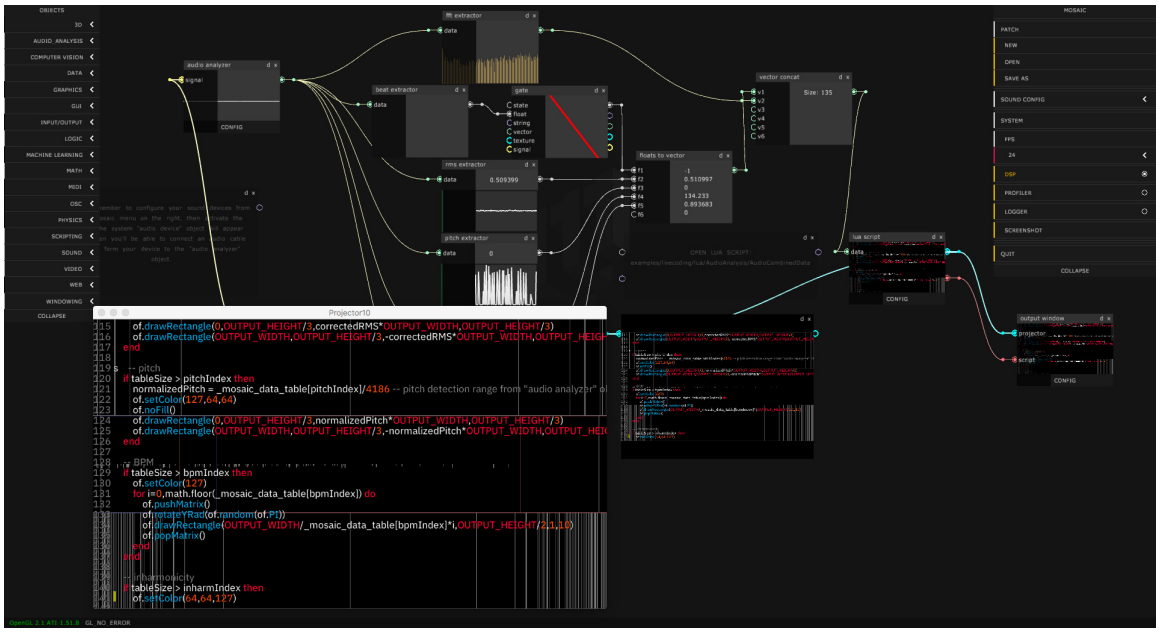

Figure 1. Mosaic Screenshot.

2. GOALS

- To submit a new development application in the field of creative programming, hybridizing live-coding and visual-programming paradigms.
- To facilitate learning and use of programming environments, promoting a human-machine interrelation that encourages augmented-thought through the learning feedback that this application provides.
- To make the software and the related documentation more widely known to promote a community-driven collaborative development.

3. MOSAIC CONTEXT

The general objective of visual programming languages is to make programming more accessible for beginners (Medlock-Walton 2014, 545), and to provide advanced users with added support at three different levels:

- Syntax, visual programming languages use blocks and a flow communication system (cords), which reduces and/or completely removes the possibility of syntactic errors (a cord cannot connect where it does not touch, the environment prevents the error).
- Visual semantics, it is plausible to consider the inclusion of visual compression mechanisms embedded in the blocks themselves (visual feedback of comprehension), or in other words, documentation of the language hybridized in the language itself.

- Pragmatic, the structure of visual programming allows the generation of case studies to analyze the interaction systems internal to the process (internal data flow/reaction/interaction) and related to the result (input -> output). Amplification of the relationship/interrelation between human-machine, boosting augmented-thought through feedback.

Today's visual programming environments also incorporate the dataflow programming concept, which translates into adding real-time live-coding capabilities, or in this case live-visual-coding, a programming environment that allows auto-parallelization (Johnston, Hanna, and Millar 2004), a programming concept that increases opportunities for shared development and multiple levels of learning (fragmentation/de-fragmentation of a program in real time)

The fundamental elements of an environment with these characteristics can be derived from the basic concepts that constitute the formulation of Reactive Functional Programming of Continuous Semantics (Elliott and Hudak 1997), which aims to abstract all the operational details not important for the programming itself (simplification of interface/use).

The key points of this formulation are related to the previously introduced dataflow programming, the dynamic modeling of values in real-time (dataflow, the cords system that we call signal transmission system), and to the possibility of controlling/programming reactive events that alter the system structure (program -> code -> visual process).

Another important point to consider is the difference between two concepts, Interactive Programming and Hot-swapping:

- Interactive Programming means programming with immediate feedback, eliminating compilation and transforming the process into a continuous loop (process of coding)
- Hot-swapping means being able to modify code WHILE it is running, and this is necessary when we are referring to Fully Interactive Programming.

Mosaic integrates live-coding (Lua, Python, GLSL, Interactive Programming without Hot-swapping) with visual programming and data flow in real time (Fully Interactive programming), combining all its elements to reinforce its native structure of auto-parallelization, leveling out the layers of complexity and equalizing the possible user levels to simplify the approach to collaborative processes.

3.1. Conceptual background

Following research by Deleuze (2007, 143), a diagram or a visual scheme is defined by modulation, similar to analog relationships, while digital language or code language is defined by articulation, underlining in the notion of abstract machine (Deleuze y Guattari 2002, 103) that it have a complementary nature, incorporating both functions, inserting one into the other. Following this idea, Mosaic grafts scripting objects inside dataflow programming.

3.2. Art and coding background

Since the 90s (20th C.) at Laboratorio de Luz at the Universitat Politècnica de València, we have been interested in interactive art applications, both for conducting research into artistic practice, and for research for the development of specific applications, which is the case of GAmuza[1].

The first GAmuza (rel.001) was developed in 2009 and was implemented only as a graphical interface to facilitate the teaching and use of video tracking techniques. The latest GAmuza (rel.1.0.1),

[1] GAmuza is one of the results of the R&D research project: Visión Ampliada: Sistemas de Tracking Visual para Instalaciones Interactivas en el Campo del Arte Digital. Ref.HAR2008-02169

released in 2016, is configured as a hybrid Live OpenFrameworks Sketching IDE (Martínez de Pisón; Mazza 2016).

The accumulated experience in teaching and research has been applied to the development of Mosaic, modifying the basic concept behind the development of GAmuza towards a visual programming environment. This means that it contains all the possibilities of GAmuza (and many more as we will detail below) but with much more plasticity in its use possibilities.

As a reference, we'll highlight what was probably first visual programming software: GRAIL RM-5999 - ARPA from the RAND Corporation (Ellis; Heafner; Sibley 1969).

Developing an application today is a process that involves the work of many people, who leave their libraries in repositories, which is why communities such as github are crucial for developing projects like Mosaic. Because of this, we have chosen to integrate openFramework's ofxAddons, updating and/or modifying them through forks. The ofxAddons used are listed in the following section.

Other references for this Project are existing commercial and non-commercial visual programming and live coding software. A short analysis of the specific contribution of each software to the Mosaic project lies outside the scope of this paper, so we'll just offer a reduced list here, as well as the artistic references.

Reference visual programming systems:

Blender, Coral, DesignScript, Dynamo, Engi JS, EyesWeb, FL Studio, GenerativeComponents, GRaIL, Houdini, Isadora, MAX/MSP, Minko ShaderLab, NodeBox, Praxis LIVE (Smith 2016), Pure Data, Quartz Composer, Reaktor, TouchDesigner, VUO, VVVV.

Reference Live Coding systems:

ChucK, Extempore, Fluxus, Fragment, Gibber, KodeLife, LiveCode, Lua, Overtone, Scratch, Siren, Sonic Pi, SuperCollider, TouchDesigner.

Artistic reference:

GRASS (GRAphics Symbiosis System), an interactive, interpreted, programming language, developed on a PDP-11/45 (DeFanti 1974)

Spiral 5 PTL (Perhaps The Last), live performance (DeFanti and Sandin 1979). DeFanti added *GRASS* as input to the Dan Sandin's Image Processor (IP), creating the GRASS/Image Processor. With this combined application they created this live performance.

4. TECHNICAL FEATURES

Mosaic is being developed with openFrameworks 0.10.0 (official version from the project website) together with a selection of ofxAddons , keeping their original versions whenever possible, or using a fork to resolve possible problems (compatibility with OF 0.10.0, bugs, specific Mosaic requirements, etc.). To achieve a high portability, both for future maintenance/upgrades, and to provide a stable common base for developers interested in participating in the project, the source code of Mosaic is organized into different cores (see Figure 2), described here in detail:

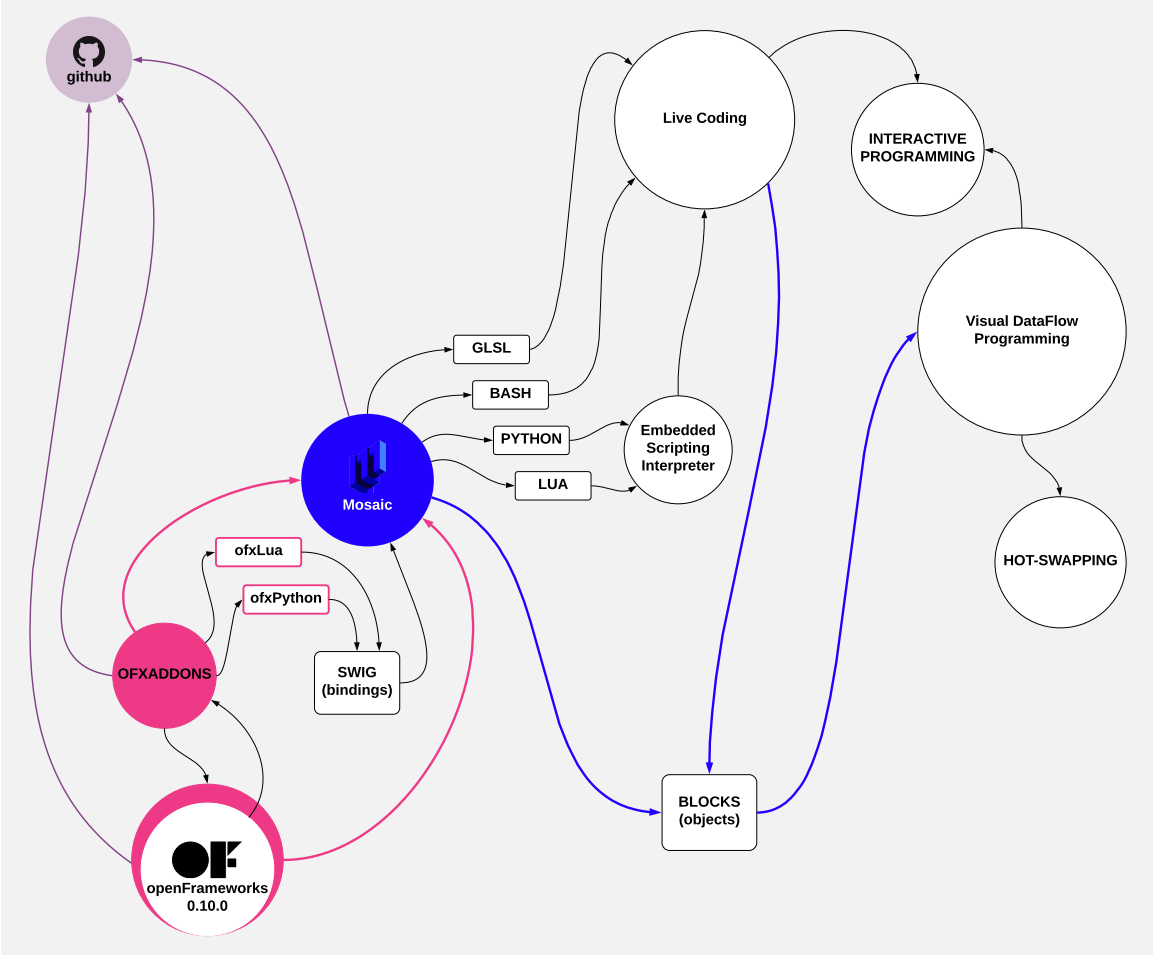

Figure 2. Mosaic Data Flow Diagram.

1. openFrameworks core, the official version is used from the download section of the official website of the OF 0.10.0. project, without touching anything of the original code
2. ofxAddons core/s, a selection of ofxAddons is made from the community (github), which are analyzed, tested outside Mosaic, integrated and retested within Mosaic. In the case of an ofxAddon that works perfectly, the original version of the author is used, whereas if problems/incompatibilities appear, the possibility of making a fork and the resulting adaptation work is studied, and if a good result is obtained, then the fork is made and a version of the ofxAddon in question is made for Mosaic.
3. "Visual Programming" core, organized and modularized in the ofxAddon structure: ofxVisualProgramming, for high portability (plug & play integration in other projects) and to create a common collaborative base. In this code core, everything that concerns the Visual DataFlow Programming system (objects and signal system) is organized, including a predefined template to facilitate the creation of new objects by the community (COLLABORATION)

To go into more detail about the possibilities of Mosaic software, we will briefly analyze its main source, openFrameworks. OF is designed according to the DIWO philosophy (do it with others), it combines a variety of advanced libraries to work with image, 3d, video and sound, it is multiplatform and provides a powerful and easy-to-use code base.

ofxAddons: ofxLua and ofxPython. These two ofxAddons are two key elements for the existence of the Live Coding system integrated within Mosaic; they allow us to obtain a binding (passing

through SWIG) of almost all of the OF APIs, and at the same time they provide us a real-time (live) scripting (coding) interpreter, both for Lua and for Python. This means that within Mosaic we can program in Lua and in Python at the same time, using OF APIs (possibility of programming almost identically to OF in C ++, but in Live) in conjunction with any of Lua and/or Python library.

GLSL, openFrameworks gives us the world of openGL Shaders, and Mosaic incorporates a simple system to use/edit them in real time (see Figure 3). For cross-platform compatibility issues, Mosaic is configured to use openGL 2.1, and consequently the Shaders it can compile (#version 120).

BASH, Mosaic incorporates the ability to open / edit / execute bash scripts (Linux and macOS), which coupled with the Visual DataFlow Programming environment, gives us direct interactive access to the operating system.

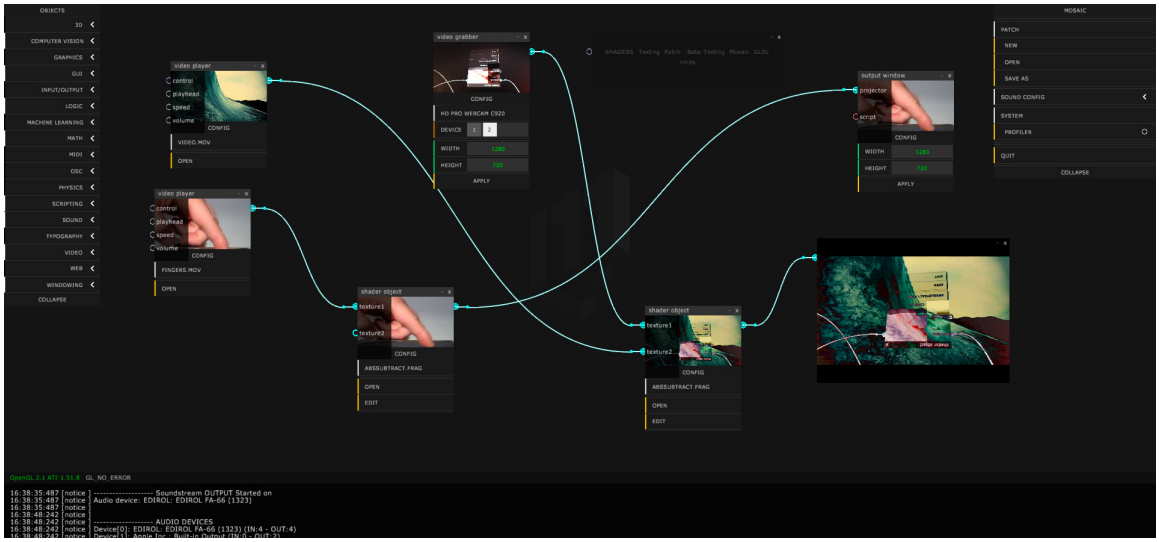

Figure 3. Shaders patch test.

Besides the ofxAddons present in the 0.10.0 version of openFrameworks, these other ofxAddons are included in the current version of Mosaic (pre-alpha 0.1.5), some originals, others adapted versions (forks):

ofxAudioAnalyzer (Zimmerman 2016); ofxAudioFile (Pisanti 2018); ofxBTrack (Tokui 2015); ofxChromaKeyShader (Koo 2014); ofxCv (McDonald 2016); ofxDatGui (Braitsch 2015a); ofxFontStash (Ferre Mesià 2015); ofxGLEditor (Hayasaka 2012);) ofxGLError (Ferre Mesià 2014); ofxHistoryPlot (Ferre Mesià 2016); ofxInfiniteCanvas (McDonald 2016); ofxLoggerChannel (Zananiri 2015); ofxLua (Wilcox 2017); ofxMidi (Wilcox 2013); ofxModal (Braitsch 2015c); ofxParagraph (Braitsch 2015b); ofxPDSP (Pisanti 2015); ofxPython (Juliá 2015); ofxSIMDFloats (Pisanti 2015a); ofxSimpleHttp (Ferrer Mesià 2014); ofxTimeline (YCAMInterLab 2012); ofxTimeMeasurements (Ferre Mesià 2017); ofxVisualProgramming (Mazza 2018).

5. CONCLUSIONS AND FURTHER WORK

We have introduced a new application hybridizing a live coding environment with visual programming, based on openFrameworks. We have explained the characteristics of its design based on independent articulated cores and the contributions it offers to the field of creative programming, teaching and self-learning. We have presented the philosophical and computer concepts on which the concept and the technology of the project is built.

Future work will involve the implementation of a distributed network communication system to

work remotely on the same projects files in a collaborative way, as well as creating an on-line community to foster collaboration with the project.

Acknowledgments

We want to thank the collaboration and support of the Laboratorio de Luz research team of the Universitat Politècnica de València and the Spanish Ministry of the Economy, Industry and Competition for its support for the 2017 call for research project: Proyectos de I+D, Programa Estatal de Fomento de la Investigación Científica y Técnica de Excelencia, Subprograma Estatal de Generación de Conocimiento. Referencia R2017-87535-P

REFERENCES

Braitsch, Stephen. 2015a. "ofxDatGui" *Github*. Development platform online (accessed August 2018) <https://github.com/braitsch/ofxDatGui>

Braitsch, Stephen. 2015b. "ofxParagraph". *Github*. Development platform online (accessed August 2018) <https://github.com/braitsch/ofxParagraph>

Braitsch, Stephen. 2015c. "ofxModal". *Github*. Development platform online (accessed November 2018) <https://github.com/braitsch/ofxModal>

DeFanti, Tom. 1974. *GRASS (GRAphics Symbiosis System)*

DeFanti, Tom y Sandin, Dan. 1979. *Spiral 5 PTL (Perhaps The Last)*. YouTube.com. Video. <https://www.youtube.com/watch?v=hw9kY85DkfE> (accessed August 2018)

Deleuze, Gilles. 2007. *Pintura. El concepto de diagram*a. Buenos Aires: Cactus.

Deleuze, Gilles; Guattari, Felix. 1987. *A Thousand Plateaus. Capitalism and Schizophrenia*. Minneapolis and London: University of Minnesota Press.

Elliott, Conal y Hudak, Paul. 1997. "Functional Reactive Animation". *International Conference on Functional Programin*g. Amsterdam.

Ellis T.O.; Heafner, F.G and Sibley, W.L. 1969. *Memorandum RM-5999-ARPA. The Grail Project: an Experiment in Man-Machine Communications*. Santa Monica: The Rand Corporation.

Ferre Mesià, Oriol. 2014. "ofxGLError". *Github*. Development platform online (accessed August 2018) <https://github.com/armadillu/ofxGLError>

Ferre Mesià, Oriol. 2014a. "ofxSimpleHttp". *Github*. Development platform online (accessed August 2018) <https://github.com/armadillu/ ofxSimpleHttp>

Ferre Mesià, Oriol. 2015. "ofxFontStash". *Github*. Development platform online (accessed August 208) <https://github.com/armadillu/ofxFontStash>

Ferre Mesià, Oriol. 2016. "ofxHistoryPlot". *Github*. Development platform online (accessed August 2018) <https://github.com/armadillu/ofxHistoryPlot>

Ferre Mesià, Oriol. 2017. "ofxTimeMeasurements". *Github*. Development platform online (accessed August 2018) <https://github.com/armadillu/ofxTimeMeasurements>

Gonzalez Vivo, Patricio and George, James. 2012. "ofxComposer". *Github* Development platform online (accessed August 2018) <https://github.com/patriciogonzalezvivo/ofxComposer>

Hayasaka, Akira. 2012. "ofxGLEditor". *Github* Development platform online (accessed November 2018) <https://github.com/Akira-Hayasaka/ofxGLEditor>

Johnston, Wesley M.; Hanna, J. R. Paul y Millar, Richard J. 2004. "Advances in Dataflow Programming Languages". *ACM Computing Surveys*, 36 (1): 1-34.

Jonas de Aguiar, Vinicius. 2017. "Diagrams and art: some thought based on Peirce and Deleuze". *Kínesis*, IX (20): 305-320.

Juliá, CarleS F. 2015. "ofxPython". *Github*. Development platform online (accessed August 2018) <https://github.com/chaosct/ofxPython>

Martínez de Pison, M. José and Mazza, Emanuele. 2016. *Live Creative Coding. Introducción a la Programación Creativa con GAmuza*. Valencia: Asociación Cultural Pluton.

Mazza, Emanuele. 2018. "ofxVisualProgramming". *Github.* Development platform online (accessed August 2018) <https://github.com/d3cod3/ofxVisualProgramming>

McDonald, Kyle. 2016a. "ofxCv". *Github.* Development platform online (accessed August 2018) <https://github.com/kylemcdonald/ofxCv>

McDonald, Roy. 2016b. "ofxInfiniteCanvas". *Github.* Development platform online (accessed August 2018) <https://github.com/roymacdonald/ofxInfiniteCanvas>

Medlock-Walton PAUL. 2014. "Blocks-based Programming Languages: Simplifying Programming for Different Audiences with Different Goals". *Proceedings of the 45th ACM technical symposium on Computer science education. SIGCSE '14*. New York: ACM

Paivio, Allan. 1990. *Mental Representations: A Dual Coding Approach*. Oxford University Press

Pisanti, Nicola. 2015. "ofxPDSP". *Github*. Development platform online (accessed November 2018) < https://github.com/npisanti/ofxPDSP>

Pisanti, Nicola. 2015a. "ofxSIMDFloats". *Github*. Development platform online (accessed November 2018) < https://github.com/npisanti/ ofxSIMDFloats >

Pisanti, Nicola. 2018. "ofxAudioFile". Github. Development platform online (accessed November 2018) <https://github.com/npisanti/ofxAudioFile>

Sandin, Dan. 1973. *Image Processor (IP)*

Smith, Neil C. 2016. Praxis LIVE - hybrid visual IDE for (live) creative coding. *Proceedings of ICLC 2016*. Ontario: McMaster University.

Tanimoto, Steven L.2013. "A Perspective on the Evolution of Live Programming". Proc. LIVE 2013, Workshop on Live Programming, IEEE Computer Society, 2013.

Tokui, Nao. 2015. "ofxBTrack." *Github.* Development platform online (accessed August 2018) <https://github.com/naotokui/ofxBTrack>

Wilcox, Dan. 2013. "ofxMidi." *Github.* Development platform online (accessed November 2018) <https://github.com/danomatika/ofxMidi>

Wilcox, Dan. 2017. "ofxLua". *Github*. Development platform online (accessed August 2018) <https://github.com/danomatika/ofxLua>

YCAMInterLab. 2012. "ofxTimeline". *Github*. Development platform online (accessed November 2018) <https://github.com/YCAMInterlab/ofxTimeline>

Zimmerman, Leo. 2016. "ofxAudioAnalyzer". *Github.* Development platform online (accessed August 2018) <https://github.com/leozimmerman/ofxAudioAnalyzer>