Digital Audio Effects 2010

The Jamoma Audio Graph Layer

Timothy Place[1] , Trond Lossius[2] and Nils Peters[3]

[1] 74 Objects LLC, [2] BEK - Bergen Center for Electronic Arts, [3] McGill University, CIRMMT

Abstract

Jamoma Audio Graph is a framework for creating graph structures connecting unit generators to process dynamic multi-channel audio in real-time. These graph structures are particularly well-suited to spatial audio contexts demanding large numbers of audio channels. This framework forms part of the Jamoma layered architecture for interactive systems, with current implementations of Jamoma Audio Graph targeting the Max/MSP, PureData, Ruby, and AudioUnit environments.

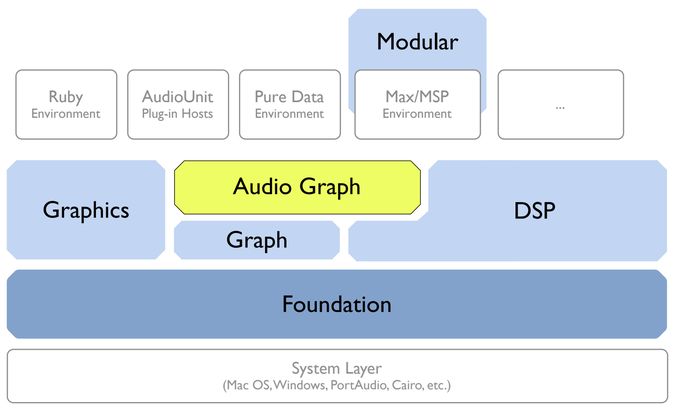
The Jamoma Platform

A comprehensive infrastructure for creating computer music systems, with a clear separation of concerns, structured in a layered architecture of several frameworks, including:

Jamoma DSP - Framework for creating a library of unit generators.

Jamoma Audio Graph - Open source C++ framework providing the ability to create and network Jamoma DSP objects into dynamic graph structures for synchronous audio processing.

Graph and unit generator is decoupled.



Requirements:

Connections between objects capable of delivering multiple channels of audio.

Support for audio signals with multiple sample rates and vector sizes simultaneously within a graph.

Audio signal sample rate, vector size, and number of channels must be dynamic (able to be changed in realtime).

Live coding/Live patching - Dynamic modification of an audio graph at runtime.

Ability to transport representations of an audio graph between different programming languages.

Support for multi-threaded parallel audio processing.

Liberal licensing for both open source and commercial use.

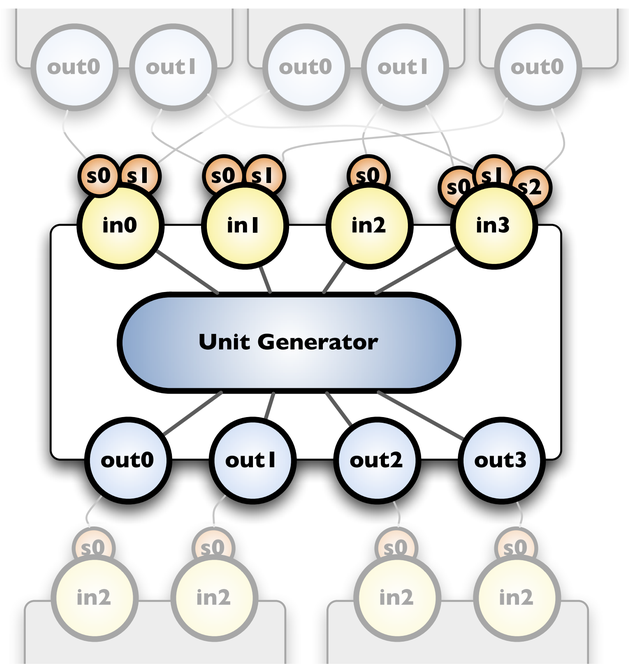
Cross-platform

Implementation

Pull mechanism - Audio processing driven from a `terminal object' or `sink' at the bottom of the chain

`TTAudioGraphObject` - Node class for the graph, wraps a unit generator with other supporting objects

Layered architecture - the audio graph object itself only knows about its inlets; the inlets only know about their sources; the sources know from what object and from which outlet they are connected.



Building the graph

The connections of the graph must be established by passing a reference to a source object, as well as the outlet and inlet numbers for the connection, to the downstream object's `connect()` method.

Similarly, connections may be cut by passing the same information to the downstream object's `drop()` method.

Connections may be created or dropped at any time before, after or during the graph being processed.; there is no global signal chain compilation, the graph may dynamically change over the course of its operation and performance.

Processing the graph

is driven by the object at the end of the chain according to a two step process:

First, a `preprocess` method is propagated up the chain. With the objects in the graph prepared, the audio can be pulled by a terminal object using the `process()` call on each of its sources.

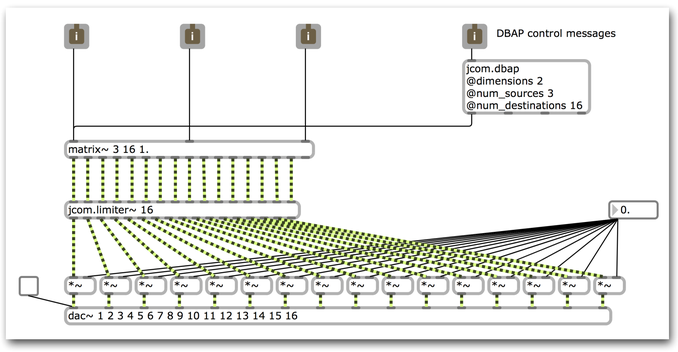
Since Jamoma Audio Graph is using a pull-based architecture, outlets are passive. They are simply buffers storing the output calculated by the wrapped unit generator.

The unit generator is a Jamoma DSP class instance, responsible for actually calculating the audio to be stored by the outlet buffers.

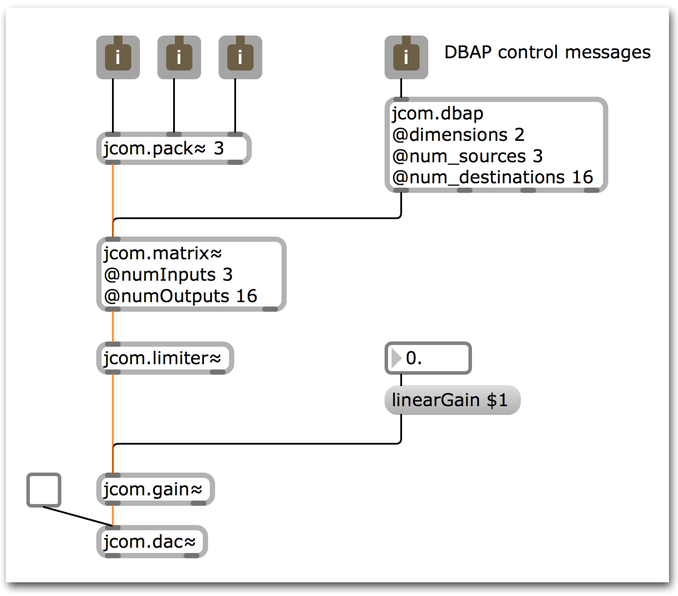
Unlike outlets, the inlets are active. When asked for a vector of audio by the unit generator, the inlets each request audio from each of their sources (other objects' outlets).

Applications

Max/MSP

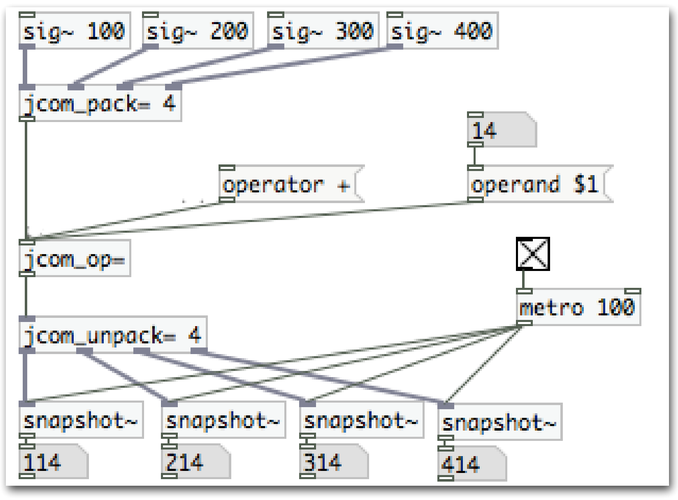


Max patch for spatialisation.

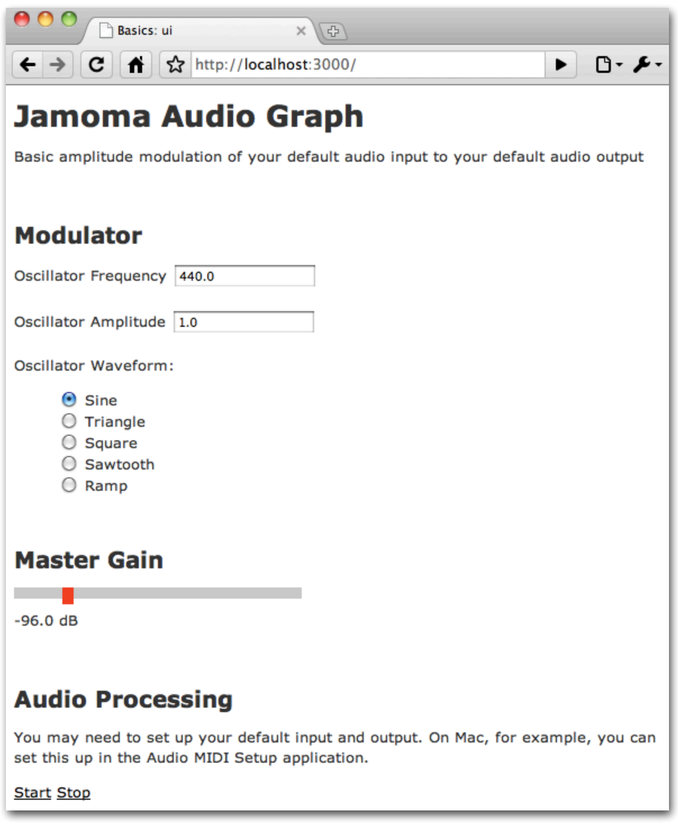


Jamoma Audio Graph simplifies the patch while increasing flexibility.

Pure Data (Pd)



Ruby on Rails



www.jamoma.org