# OM-pd: API between PureData and OM-Sharp

MODALIDADE: COMUNICAÇÃO

SUBÁREA: Composição e Sonologia

*Charles K. Neimog*
*Universidade de São Paulo*
*charlesneimog@outlook.com*

*Rodolfo Coelho de Souza*
*Universidade de São Paulo*
*rcoelho@usp.br*

**OM-pd: API between PureData and OM-Sharp**

**Abstract**. This communication will present a library capable of integrating OM-Sharp and PureData. There are two goals: The first is to allow the processing of large amounts of data using the visual language (BRESSON et al., 2017) of OM-Sharp (OM) and the offline mode of PureData. The second is to bring PureData's audio possibilities to OM.

**Keywords:** OM-Sharp, PureData, BigData, Visual Programming.

**OM-pd: API entre PureData e OM-Sharp**

**Resumo**. Esta comunicação apresentará uma biblioteca capaz de integrar OM-Sharp e PureData. Existem dois objetivos: O primeiro é permitir o processamento de grandes quantidades de dados utilizando a linguagem visual (BRESSON et al., 2017) do OM-Sharp (OM) e o modo offline do PureData. A segunda é trazer para o OM as possibilidades de áudio do PureData.

**Palavras-chave**. OM-Sharp, PureData, BigData, Linguagem Visual.

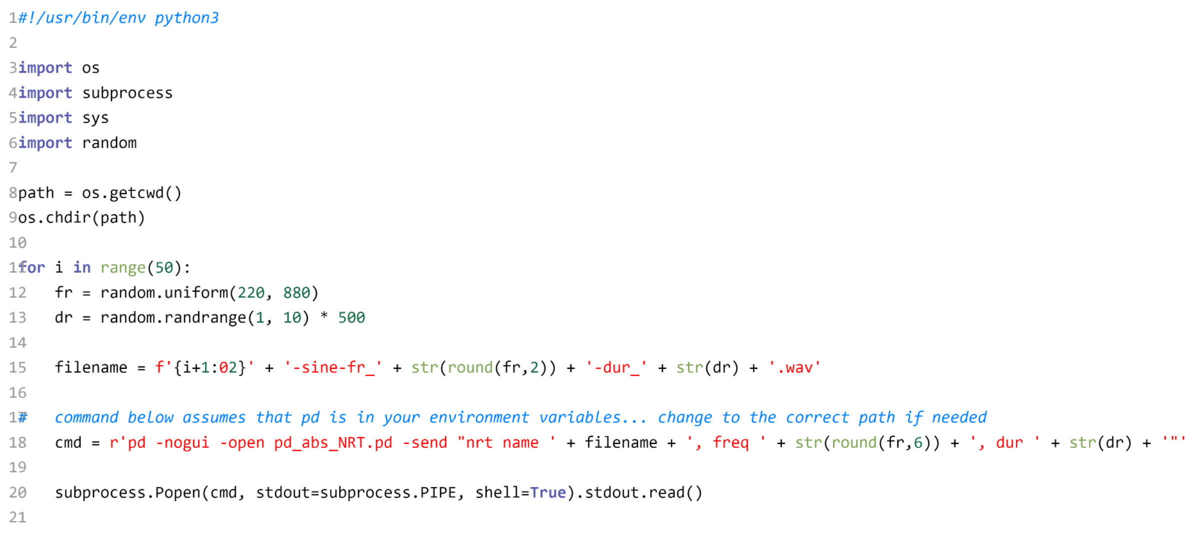
## Introduction

One of the great features of the OM's family (OpenMusic and OM-Sharp) is that they implement most of the Lisp language. Due to this feature, we believe that the OM family is a helpful tool to work with scripts and automatize repetitive works, and because of that, it can be used to build large musical structures using math, draws, scores, graphs, and other representations to control audio parameters.

However, there are some problems with OM. The most perceptible is that Lisp – the programming language in which OM is built – is not a fast language for audio processing, and the community is decreasing, so neural synthesis, for example, is not widely implemented. Thus, the possibility of making scripts/patches for big musical data processing and some audio approaches is not fully available in the OM environment.

Such as OM-Sox (with SoX), OM-Chroma (Csound), and others, OM-pd (with PureData) tries to solve this by building one bridge (API) between Pd and OM. The approach was initially based on Padovani's repository (https://github.com/zepadovani/pd-python_NRT), which is an example of how it is possible to use PureData in the command line with Python. We understand that one of the primary purposes of using PD in command-line mode is to allow the processing a large amount of data. The problem is that it is usually used with textual languages (as in the Python code shown in figure 1). OM-pd tries to bring this kind of possibility into the visual programming environment of OM and PD.

**Figure 1: Padovani's example of using PureData and Python. It generates 50 aleatoric sinusoids.**



```python
#!/usr/bin/env python3

import os
import subprocess
import sys
import random

path = os.getcwd()
os.chdir(path)

for i in range(50):
    fr = random.uniform(220, 880)
    dr = random.randrange(1, 10) * 500

    filename = f'{i+1:02}' + '-sine-fr_' + str(round(fr,2)) + '-dur_' + str(dr) + '.wav'

#   command below assumes that pd is in your environment variables... change to the correct path if needed
    cmd = r'pd -nogui -open pd_abs_NRT.pd -send "nrt name ' + filename + ', freq ' + str(round(fr,6)) + ', dur ' + str(dr) + '"'

    subprocess.Popen(cmd, stdout=subprocess.PIPE, shell=True).stdout.read()

```

Source: Figure of authors, code from prof. José H. Padovani.

Next, we will present how the library works and some of its possible applications.

## OM-pd

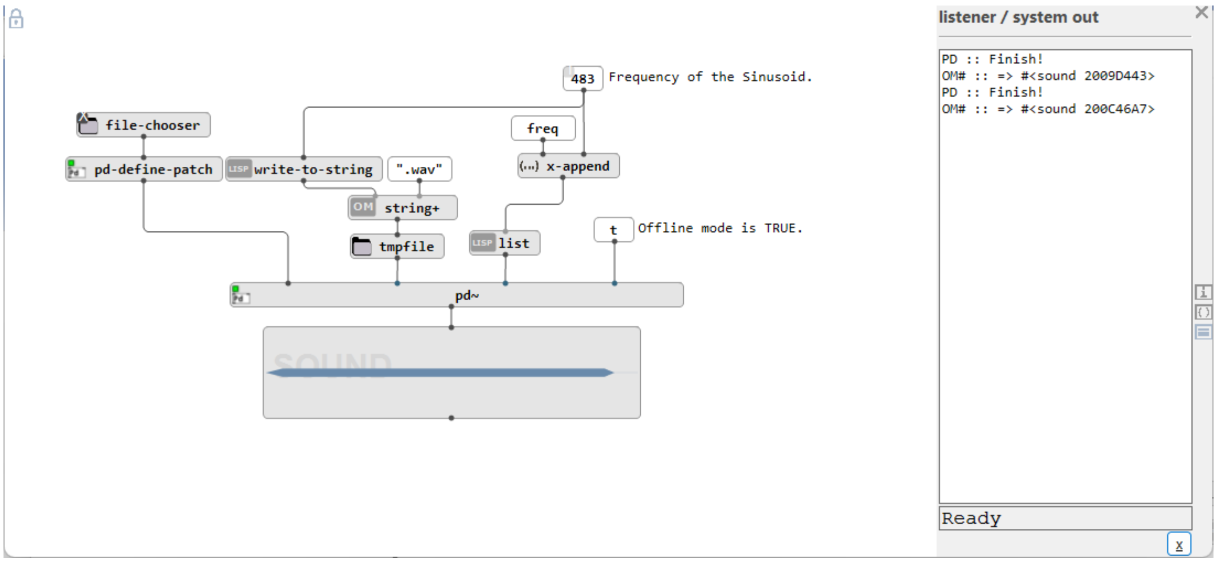
The OM-pd has few objects, they are: *pd~*, *wsl-pd~* (just for Windows), *pd-open-patches*, *pd-kill* (useful when we do something wrong in multithreading tasks), *pd-list-patches* (special box), *pd-mk-line*, and *pd-multithreading*.

The *pd~*, *wsl-pd~*, and *pd-mk-line* objects work similarly. They build command lines that will run Pd. *Pd~* is the main object, *wsl-pd~* will run Pd for Linux on the Windows machine (sometimes it is challenging to compile some objects for Windows), and *pd-mk-line* will build *pure-data* classes that need to be used with *pd-multithreading* object, for multithreading processing (It allows to allocate more computer resources, the processing is faster). These three objects have one required *inlet* and seven optional *inlets*. The required inlet is the *patch* defined using the *pd-define-patch* or *pd-list-patches* object. The seven optional inlets are *sound-in* (pathname of some audio input), *sound-out* (pathname of output audio), *var* (a list of variables and values for audio manipulation), *GUI* (open or not the Graphical User Interface of PureData), *offline* (run or not in offline mode), *verbose* (show or not information), and *thread* (run or not in a threaded mode, OM 'tell' PD to execute without the wait for it finish).

Next, we will present some simple examples: First, we want to build one sinusoid controlling the frequency using OM#. It will happen with this patch in figure 2.

In this case (figure 2), we use the inlet *patch* (always required) – it is highly recommended not to use spaces character in patch pathname folders and names – the *sound-out* inlet, the *var* inlet, and the *offline* inlet – in OM#, you can use the '+' in the object to add *inlets, when* you click on the inlet created you can choose (in *pd~* object) between sound-out, sound-in, var, GUI, offline, thread, and verbose. The patch pathname will be specified using the object *file-chooser*, and the object *pd-define-patch* will define some configurations to use in the *pd~* object (the use of *pd-define-patch* is not required). In the second inlet, we have the path where the sinusoid will be saved, and the third inlet is where we choose the sinusoid's frequency, in this case, 483. Note that we need a list of lists (a list of numbers: (1 2 3 4 5); a list of lists of numbers ((1 2) (1 2) (1 2)) ) and not just a list in this inlet. In the fourth inlet, we specify that we want PureData to synthesize the sinusoid the fastest possible (offline mode).

**Figure 2 - Example of patch in OM-Sharp.**



```
PD :: Finish!
OM# :: => #<sound 2009D443>
PD :: Finish!
OM# :: => #<sound 200C46A7>
```

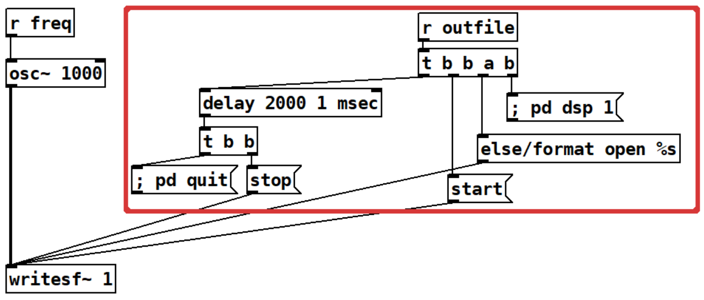
```
Ready
```

Source: Figure of authors.

On the PureData side, we have the patch in figure 3. To build a sinusoid, the part inside the red square is complex. Nevertheless, it will be the same for all similar processes (when we want to synth some audio using OM parameters without manipulating a pre-existing sample). It happens because it must deal with the exact order of the audio in PureData. To work correctly:

1. It has to turn on the *DSP*.
2. Specify the audio file path (this is sent by OM through the *sound-out* inlet, in PureData, we receive it using the '*r outfile*' object).
3. Start the recording.
4. Play the sinusoid (2 seconds in this patch, note that the *.wav* file will have 2s. The process is much faster).
5. Stop the recording.
6. Exit from PureData (this is an essential part when we are using the OM-pd).

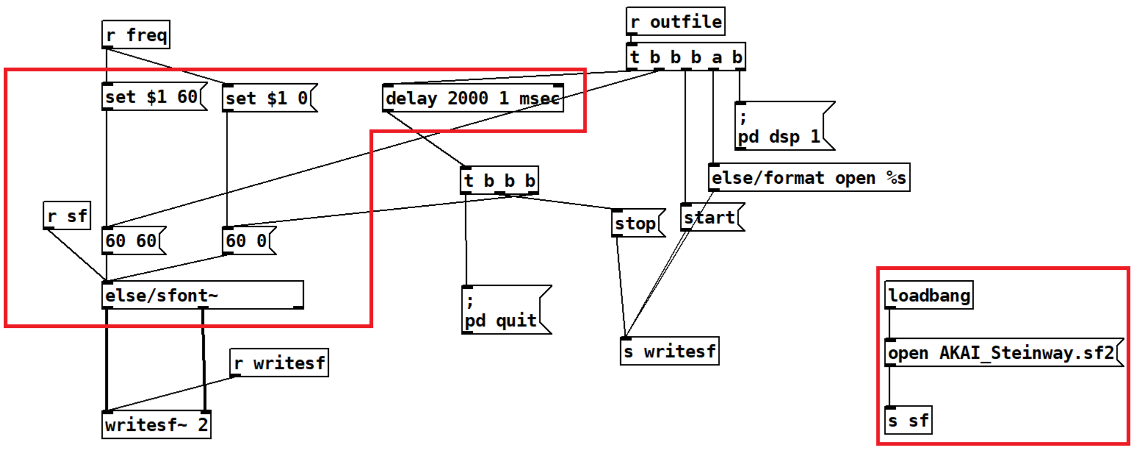
**Figure 3 - Patch in PureData to build sinusoid.**



Source: Figure of authors.

Another example is to change the patch to build a note using SoundFont. First, we must load the SoundFont (using loadbang) and add some note-in/off to *else/sfont~*. All the similar processes will happen with the same structure.

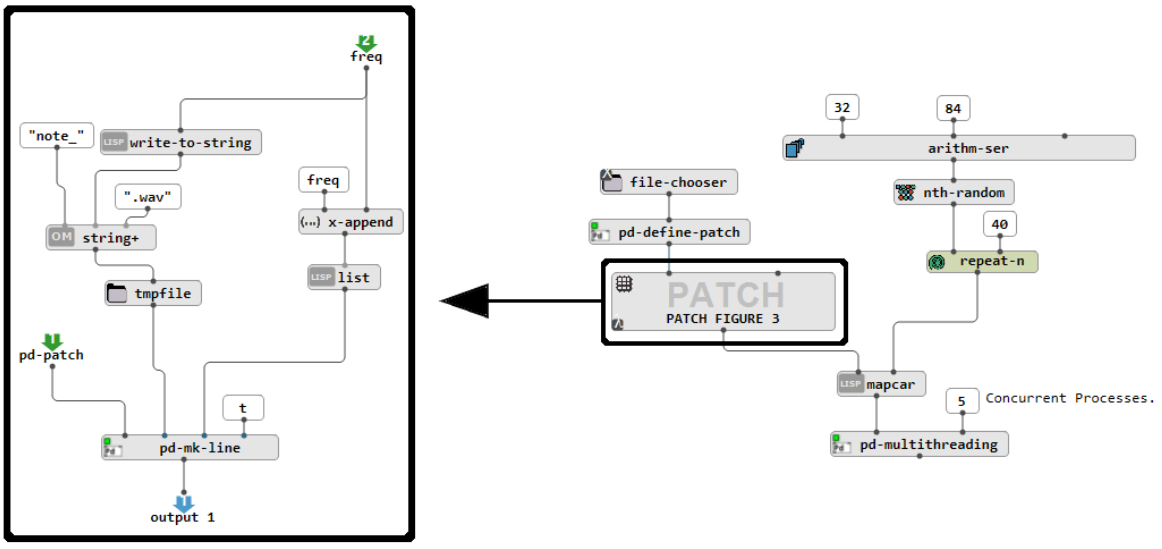
**Figure 4 - The patch builds a SoundFont note.**



Source: Figure of authors.

The patches presented can work in multithreading. For that, we need to change the OM patch, as represented in figure 5. In this case, for example, we build 40 samples of 2 seconds in about 10 seconds.

**Figure 5 - The patch builds 40 samples using sfont~ in multhreading.**



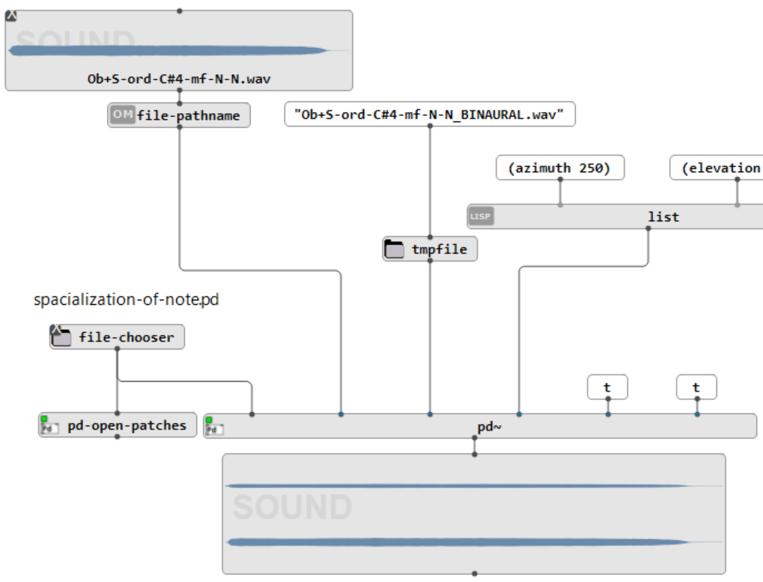
Source: Figure of authors.

One second way to work with OM-pd is to manipulate samples. Here one sample will be spatialized. It is possible to use the patch of figure 6 in OM and figure 7 in PureData. In *pd~* object, we have (left to right):

1. The *patch* is in the first inlet.
2. The pathname of the audio that will be manipulated (*sound-in*).
3. Where the audio manipulated by PureData will be saved (*sound-out*).
4. The list of parameters and values (*var*).
5. The offline mode is true (*offline*).
6. The verbose mode is true (*verbose*).

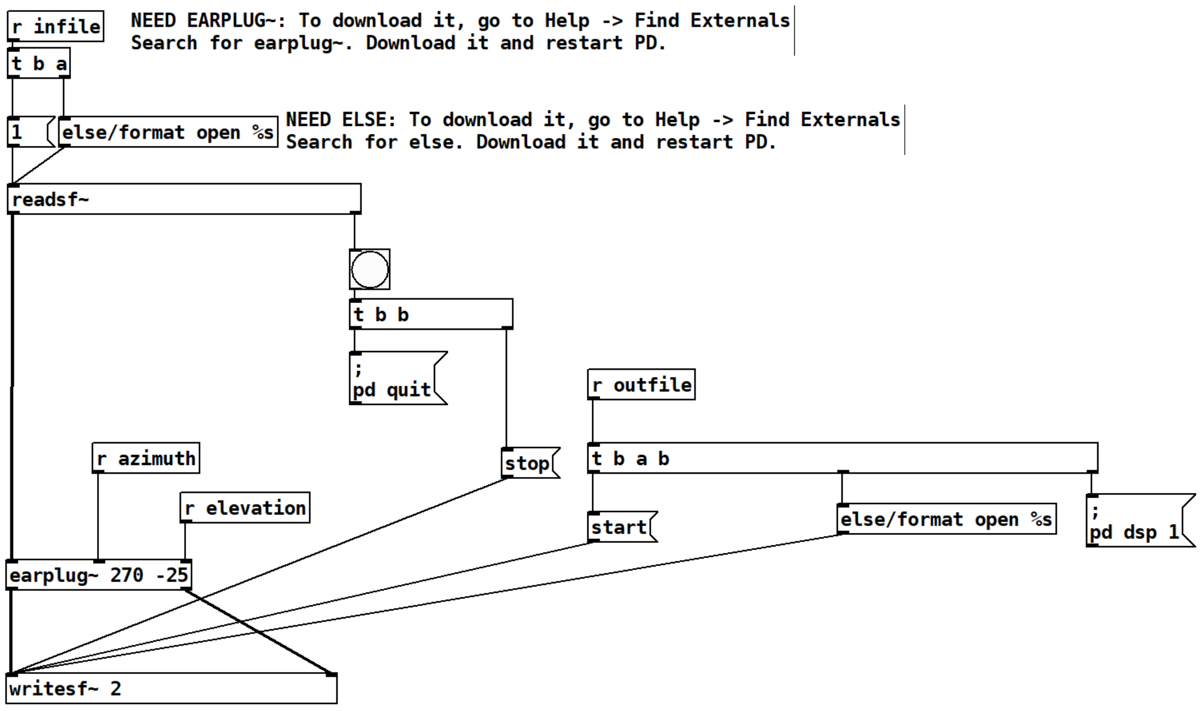
In OM, it is essential to note that the parameters azimuth and elevation are part of one list with the variable's name followed by the value (a list of lists). These values will be received in PureData using the *r* object and the variable's name. The variable azimuth, for example, is received using the object *r + azimuth* (*r azimuth*).

**Figure 6 - OM patch to generate audio spacialization.**



Source: Figure of authors.

**Figure 7 - OM patch to generate audio spacialization.**

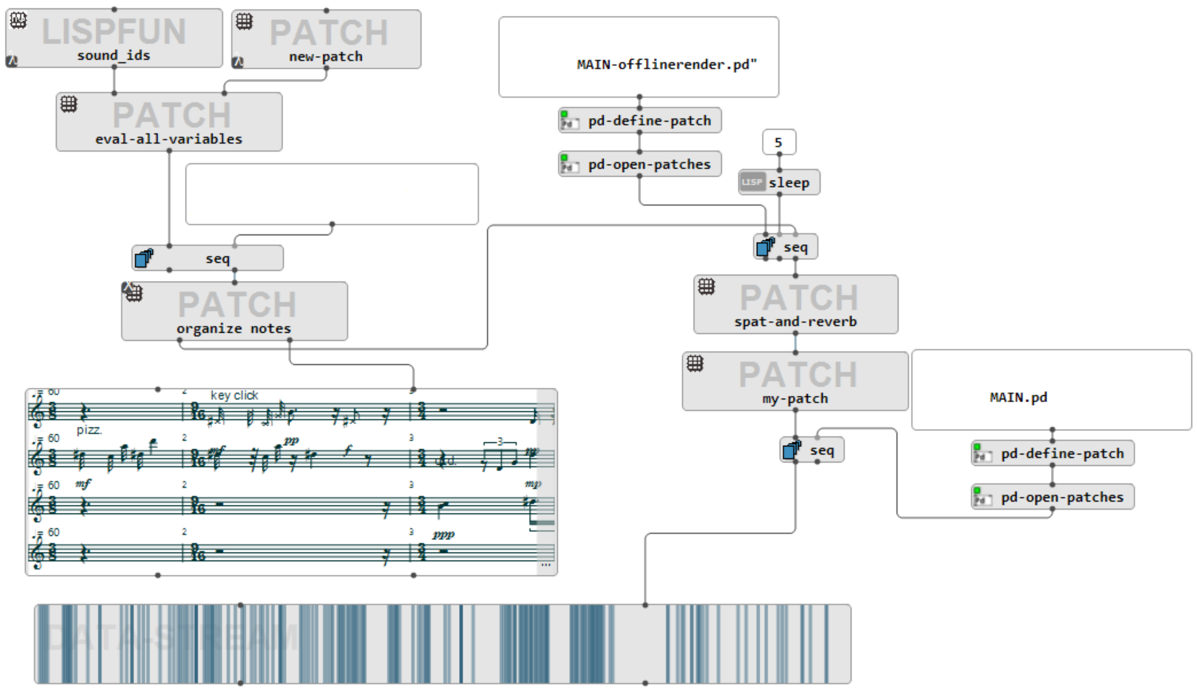

r infile
NEED EARPLUG~: To download it, go to Help -> Find Externals
Search for earplug~. Download it and restart PD.
t b a
1
else/format open %s
NEED ELSE: To download it, go to Help -> Find Externals
Search for else. Download it and restart PD.
readsf~
t b b
; pd quit
r outfile
r azimuth
stop
t b a b
r elevation
start
else/format open %s
; pd dsp 1
earplug~ 270 -25
writesf~ 2

Source: Figure of authors.

Note that we add the *r infile* in the PD patch. This object will receive the pathname of the audio that will be spatialized (second inlet in OM patch). It will open the audio and then play it (this order is done using *t b a*). OM-pd will always send the input file for last. Therefore, we must load all settings before the *infile* (we suppose that with the *infile*, you will load the audio, play it, and use the bang outlet of *readsf~* to stop the recording and exit of PureData). To load settings is possible to use the *loadbang* object or the receive (r) with the *om-loadbang* argument (*r om-loadbang*).

The third possibility is to use real-time. In this mode will be possible to record (to use in OM) or hear. One example is making a PD patch to play the *data-track* (a bi-dimensional representation of data), like in figure 8. Here, the *data-track* organizes sounds of one score and sends data using the Open Sound Control (OSC) protocol to PureData. An excerpt from the example below is available at: https://bit.ly/ANPPOM2022.

**Figure 8 - Render Score using the Orchidea samples and PureData.**

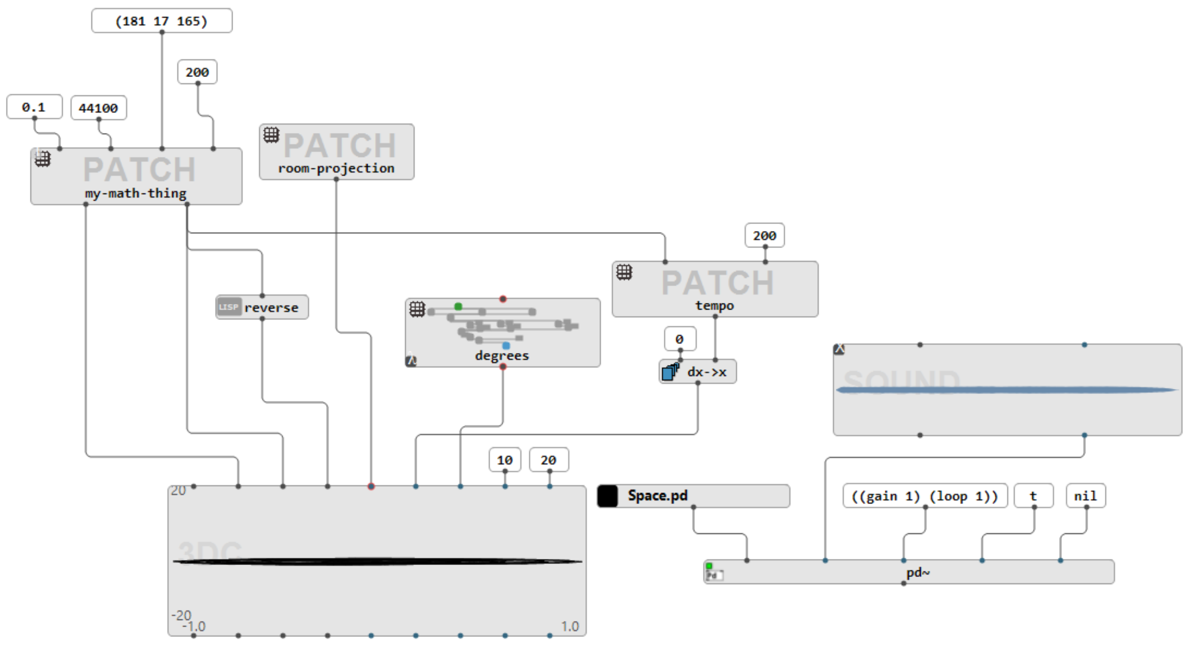


Source: Figure of authors.

Another example is to render spatializations using the 3DC (three-dimensional space graph in OM) and the IEM Plugin Suite and *vstplugin~* from the University of Graz (Austria). This process could be helpful when we want to hear or render trajectories generated by some math process in OM. The difference between running Pd by itself is that we can load audio files and specify configurations inside OM without changing the interface (OM to PureData).

The patch of figure 9 works similarly to OM-Spat (EINBOND et al., 2021) from Ircam, and it is free and OpenSource tools (IEM plugins are free and OpenSource too). An audio example is available at *Example - Figure 9* (in https://bit.ly/ANPPOM2022).

**Figure 9 - 3DC class in OM-Sharp. It will control the spacialization parameters in PureData.**



Source: Figure of authors.

We presented the three possible ways to work with OM-pd, the synthesis using OM parameters, the offline and faster manipulation of one pre-existent *sample*, and the real-time render of audio data.

## Use Pd for Linux in Windows

There are objects and tools developed in Unix Systems (Linux and macOS). In some cases, it is hard to compile them for Windows. Examples are *nn~* (CAILLON; ESLING, 2021), *wavae encoder~* and *decoder~*, *ddsp~*, and others. With Windows 10/11, it is possible to use these tools without changing the workflow through the Windows Subsystem for Linux (WSL). It requires running four lines of PowerShell commands available in https://github.com/charlesneimog/OM-pd and changing the *pd~* object to *wsl-pd~* object. In the backend, what happens is that *wsl-pd~* will run the version of PureData inside a Linux virtual machine, enabling the use of these objects for offline processing without spending time discovering how to compile it for Windows. The link shows audio examples of OM-pd working with WSL using the *nn~* (in PureData) object and a Berimbau model (https://bit.ly/ANPPOM2022).

## Potential applications

OM-pd helps the organization of large structures in acousmatic music using OM tools and libraries (for example, OM-Chaos, OM-Combine, and OM-Alea). At the same time, it provides methods for audio manipulation in OM, like VST plugins, easy multithreading, and all tools developed for PureData.

It may be effective for educational applications. With Orchidea-Sol and OM-CKN, for example, it will be possible to render samples using extended techniques (see the piece *Ideias Roubadas II* in https://soundcloud.com/charlesneimog/ideias-roubadas-ii). It should help, with some more implementations, young composers to hear how music will sound using extended techniques, bringing students closer to these sounds.

In underdeveloped countries where it is challenging to acquire tools like Spat (or OM-Spat), it will be possible to use, for example, IEM Suite Plugins (free and OpenSource tool) that can provide similar services (used in figure 9) with OM.

Finally, the library is mainly developed for composers dealing with large amounts of data by reducing the need for textual programming. In unmethodical tests, for example, it has no time difference between using pedalboard (https://github.com/spotify/pedalboard) (Spotify) in Python or OM-pd and PureData for VST audio manipulation. So, we understand that OM-pd allows us to use the fast processing power and the excellent visual programming tools simultaneously, adding to PureData the easy way to make scripts in the OM. Moreover, it can automate tasks using traditional tools like scores (voices, polys, and chords) and bi and three-dimensional representations (3dc and bpf), which seems ideal for compositions that use large amounts of data.

The library is available at https://github.com/charlesneimog/OM-pd/.

## References

BRESSON, Jean; BOUCHE, Dimitri; CARPENTIER, Thibaut; SCHWARZ, Diemo; GARCIA, Jérémie. *Next-generation Computer-aided Composition Environment*: A New Implementation of OpenMusic. *In*: PROCEEDINGS OF THE INTERNATIONAL COMPUTER MUSIC CONFERENCE 2017, Shanghai, China. *Anais*... Shanghai: [s.n.], 253-8 (p.), 2017. Disponível em: https://dblp.org/db/conf/icmc/icmc2017.html. Acesso em: 30 de jun. 2022.

CAILLON, Antoine; ESLING, Philippe. *RAVE*: A variational autoencoder for fast and high-quality neural audio synthesis. *[S. l.]*, 2021. DOI: 10.48550/ARXIV.2111.05011. Disponível em: https://arxiv.org/abs/2111.05011. Acesso em: 22 jun. 2022.

GARCIA, Jérémie; BOUCHE, Dimitri; BRESSON, Jean. *Timed Sequences*: A Framework for Computer-Aided Composition with Temporal Structures. *In*: TENOR 2017, 3RD Internacional Conference on Technologies for Music Notation and Representation. A. Coruña: [s.n.], 131-6 (p.), 2017. Disponível em: https://hal.archives-ouvertes.fr/hal-01484077. Acesso em: 22 jun. 2022.