

Scientific Report from ConGAS Short Term Scientific Mission (STSM) to Stockholm

Details

Applicant

Smilen Dimitrov
Department of Medialogy, Aalborg University Copenhagen (AUC)
Lautrupvang 15
DK-2750 Ballerup, Denmark

Host institution

Department of Speech, Music and Hearing (TMH), KTH
Stockholm, Sweden

Mission period

February 26 - March 10, 2007

Purpose

Purpose of the STSM to KTH in Stockholm was to work on combining an interface for scratching, Skipproof (developed by Kjetil Falkenberg Hansen, KTH), with a physical model of a violin (developed by Stefania Serafin, AUC). Eventually, such a merge would potentially allow a user to experiment in producing violin sounds, through re-mapping of movements derived from turntable scratching.

1 Objectives

The merging of the two pieces of software would, in general, allow users to explore whether control, derived from one type of a physical interaction with an instrument, would be applicable to a different type of instrument - in this particular case, the intent is to be able to use the control derived from hand motions during turntable scratching, and apply them to parameters of a physical model of a violin.

The Skipproof program uses the playback speed and fader volume of a turntable (either as recordings or as modelled parameters), in order to render a scratching sound from an arbitrary sample. These two parameters could be readily applied to the "pitch" and "volume" of a violine, however it would also be interesting to see whether there are any different playable combinations - like applying these control parameters to bow velocity or force, which are available as control parameters of the used physical violin model.

As such, most of the work carried out was in solving the programmatic issues in being able to map the desired control signals to the desired audio rendering engine - and providing an initial user interface for it.

1.1 Changes

As Skipproof had code for interfacing with a data acquisition hardware (like the Toaster from La Kitchen), the development went beyond the issues of connecting a scratching and violin models, and into providing a generic interface, where four different types of signal inputs (recorded, modelled, or live - via sensor input) could be patched to two different audio rendering engines (violin physical model, or Skipproof playback engine).

2 Work description

2.1 Timeline

The STSM was held in weeks 9 and 10 in 2007. The daily working process was straightforward - morning briefings, and then work during the day (possibly interrupted by lunch breaks, meetings, or guest lectures).

As mentioned previously most of the work dealt with solving programmatic issues in connecting the two pieces of software. At the end of the stay, a little experiment was performed, where an attempt was made to use sensor equipped turntables as a controller.

- Day 1 - information gathering, accomodation
- Day 2, 3 - work on porting the physical violin model from a Max/MSP environment to a PureData environment
- Day 4-10 - development of a common interface, to allow patching between Skipproof and violin physical model engines
- Day 11-12 - implementation of a circuit, and experiment with usage of signals (generated by a sensor enhanced turntable, and sampled with a Toaster data acquisition hardware) to drive the audio engines through the common interface

2.2 Method

The method used in the development during the STSM, was simply to try and assemble a working prototype - 'working' in the sense that a user would be able to experience the different audio rendering engines, and be able to recognize when control signals were applied; however, without greater inquiry into the further playability or interaction issues. The criterion for whether the prototype was 'working' was left solely at the judgement of the developers; no further user tests were performed (however, there are plans of conducting a test of the patches and mappings, related to the Wacom tablet, by a 'serious wacom-player', although there are not many of them).

3 Results

The results from this STSM are mostly of programmatic character. First of all, porting of the violin physical model, from the original MSP object for Max/MSP, to a DSP object in PureData, was performed.

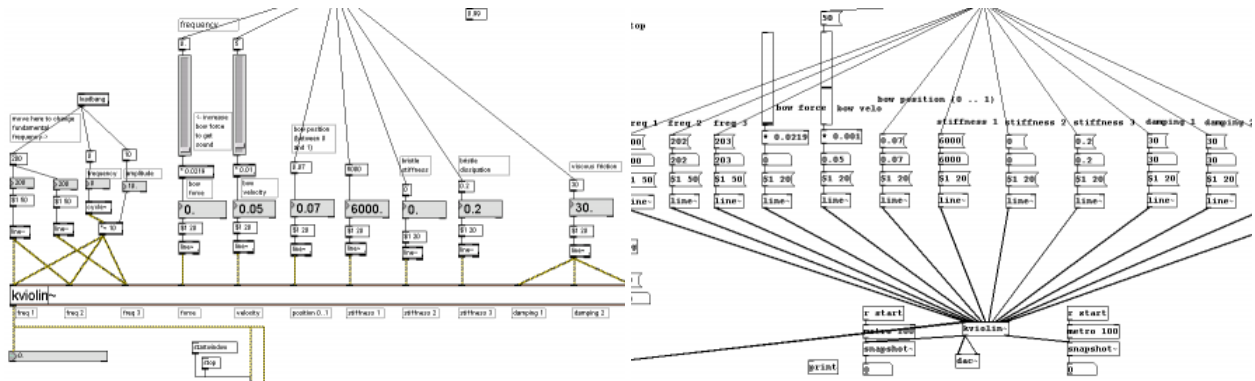


Figure 1. The physical violin model object in Max/MSP environment (left) and in PD environment (right)

As there are quite a few similarities between the code structure of dsp/audio objects in Max/MSP and PD, the procedure was quite straightforward. An attempt was made to port this code to the flex library, whereby one would have a single source code project for compiling both PureData and Max/MSP objects; however, this possibility was judged as too time consuming, and was not further explored.

The most obvious result, is the development of an interface in PureData which allows for starting and mapping of parameters between Skipproof and the physical violin model; this interface has the working title of PhSkpf (from PPhysical [referring to the violin model] and SKiPproofF).

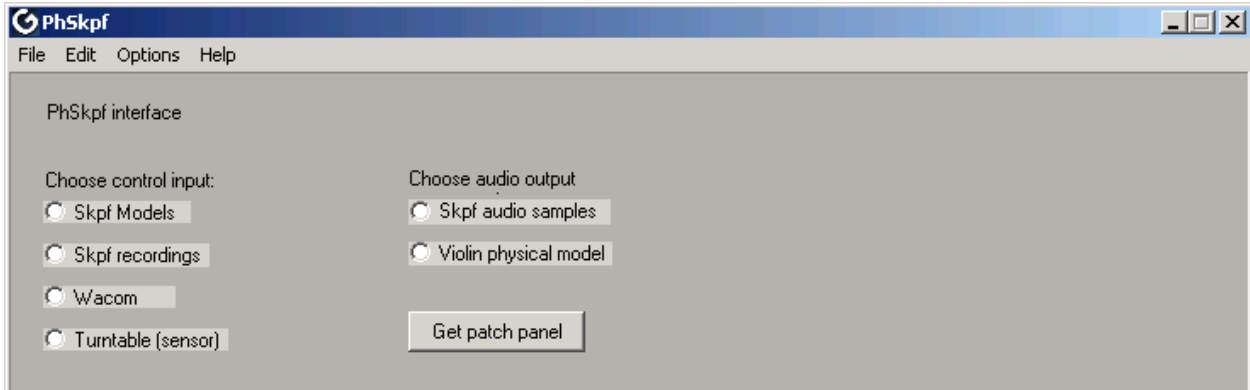


Figure 2. Starting screen of the PhSkpf interface

As PureData does not offer directly ways to build a GUI, an extension library called gripd was used (same one as the one underlying Skipproof). Initially, one can choose between 4 control inputs (Skipproof models, Skipproof recordings, Wacom tablet, and sensor equipped turntable); and 2 audio outputs (Skipproof sample playback engine, or violin physical model). Depending on the choices, Skipproof interface may be additionally started, or parameters be drawn on the PhSkpf screen.

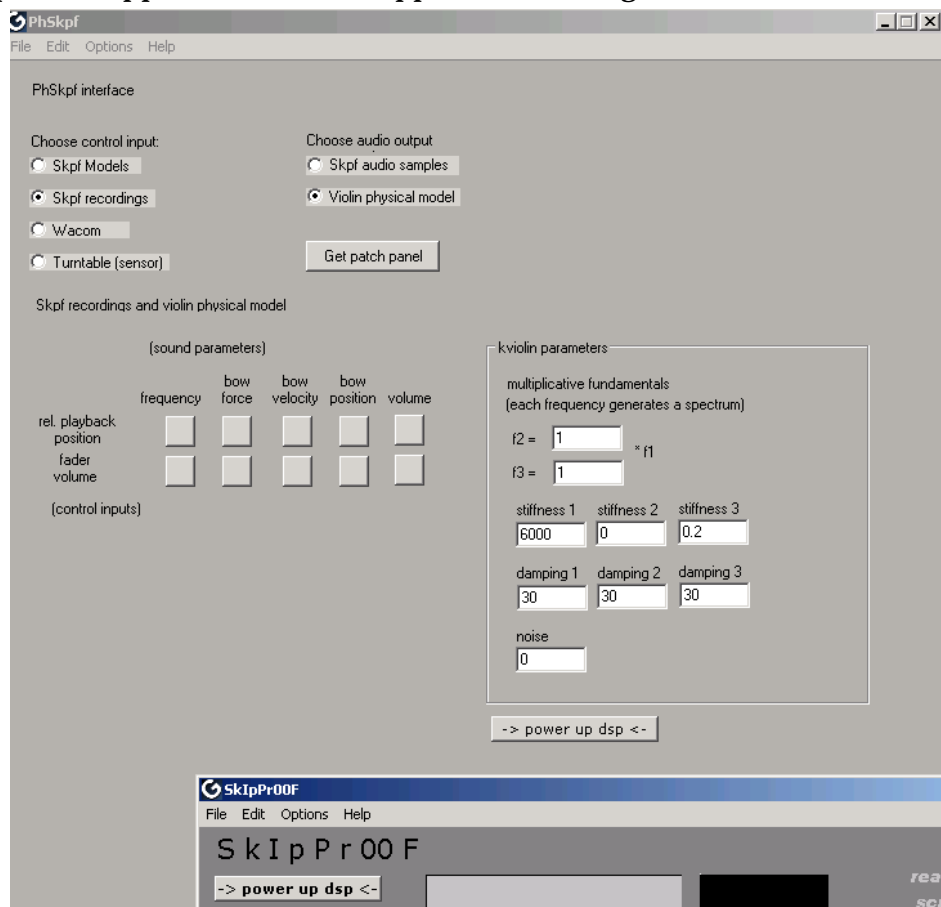


Figure 3. PhSkpf screen with violin model parameters, started Skipproof screen, and mapping patcher matrix

For each of the possible mappings between control input and audio output, a patching matrix - which is a 2D matrix of buttons - is offered as well, to allow the user to choose which input parameter should be mapped to which output parameter. This however is not a complete solution, as the user would also have to have access to scaling parameters - as sometimes a signal in the range of (0..100) must be scaled to fit in the interval (0..1); these however are not accessible through the PhSkpf GUI, and must be changed "manually" in the PD code.

The wacom tablet was added both as a way to safely test possible sensor interaction - as there is an available object that can provide Wacom tablet data in PD (known as wintablet). In addition, the Wacom can also be seen as an exciting interaction device - especially if one realizes that there are a lot of realtime parameters that could be used for musical interaction; besides the common x and y position, there is also pressure, orientation and tilt. Unfortunately, wintablet offers access only to x, y and pressure as realtime signals. Therefore, an attempt was made to develop a Wacom object, using flext (which would make it portable to both PD and Max/MSP platforms), using a library known as bbttablet. Again, this approach of developing a custom Wacom object, was deemed as time consuming, and was therefore abandoned.

Finally, an experiment was performed, to see whether a sensor equipped turntable can be used in conjunction with data acquisition hardware like Toaster, to deliver signals for PhSkpf.



Figure 4. A sensor equipped turntable, with an interface circuit for the Toaster

The sensor on the turntable seems to be an electric motor whereas the signal output from it seem to be the quadrature motor signals; meaning that one obtains a square wave voltage, whose frequency is directly proportional to playback speed. As such, it is difficult to use this signal with a data acquisition hardware like the Toaster, which typically has very low sampling rates.

Because of this, a simple circuit using a 555 timer was implemented on a protoboard, in order to try and obtain a DC signal corresponding to the playback speed of the turntable - which would make it possible to read the signal even with a slow A/D device like the Toaster. It was possible to use this circuit to demonstrate the possibility of obtaining a DC signal related to a playback speed, and using that signal to control sound output in PhSkpf (although the particular design used here to obtain a DC signal is flawed).

3.1 Discussion

Working on this level of development, the biggest issues were in the attempt to develop a GUI in PureData, which is facilitated by an external library gripd; and the process is not very intuitive nor user friendly. Further problems occurred due to a bug in using particular objects, that come from the PD extended library (in particular switch and gate), which when saved in a subpatch, tend to get saved with the wrong type of an object; which completely breaks the code on next opening of the patch. This problem managed to destroy several subpatches during development.

The particular porting of the violin physical model from Max/MSP to PureData was not that problematic, as both environments share a similar structure for realtime audio objects; however, trying to port the code to the flext library (which should then automatically compile objects for both PD and Max/MSP from a single source) proved difficult - and the same goes for the attempt to develop a Wacom object in flext.

Finally, the 555 interface circuit, in general tries to provide a DC signal, whose value is correspondant to the period of a pulse from the sensor - meaning it is inverse proportional to frequency (and eventually the playback speed). This is maybe the biggest flaw of the design of this circuit, as lower frequencies correspond to higher DC values (meaning, the DC value for zero playback speed should be infinite) - one would certainly need the inverse in order to use the control signal directly. However, for a very limited range of playback speeds, the inverse operation can be performed in software, and it can be thus demonstrated that a DC signal, derived from the playback speed measured by a sensor, can be used to drive audio output from PhSkpf.

4 End summary

4.1 Dissimination

A joint paper is planned, discussing in more detail the results gained from the STSM; a joint paper is currently planned for submission to ICMC and DAFX.

4.2 Acknowledgements

I am very grateful to Stefania Serafin, Medialogy for making this STSM possible; as well as to Kjetil Falkenberg Hansen, Roberto Bresin, and everyone else at KTH, for the great welcome and help, during the two weeks in Stockholm.