Project in musical communication and music technology

Simple Sampler

Alexej Ismailov - alexeji@kth.se
Mats Lexell - mlexell@kth.se

Musical communication and music technology
(DT2213)

2014-06-25

Abstract

The goal with this project has been to develop an expressive simple sampler. It’s suppose to, with the click of a button be able to sample any tone, preferrably your voice, and automatically pitch it to each key of a keyboard so that it becomes playable. The entire project was programmed using pure data. As a result we have come up with an application that not only works but gives surprisingly good results. We have evaluated three types of pitching algorithms and come to a conclusion of which one that was preferable.

The idea

The idea of this project is to create a sampler that enables to user to record a sound. This sound is then analysed, and fine tuned to fit to the standard pitch notation (where F_{A4} = 440Hz). The sound is then pitched and assigned to each key of the keyboard, allowing the user to play his own sampled sound as a keyboard. The biggest challenges with implementing this will be to find a way to pitch the sound without reducing it’s qualities too much.

Method, preparation and implementation

We implemented this idea using pure data as a programming language. We divided the work into different implementation parts. We needed to create the following pieces to our program:

A pure data patch that could:

1. Record a sound and save it to a file.
2. Analyse the recorded sound and determine it’s exact pitch.
3. A patch that using the pitch information could fine tune a recorded sound to fit standard pitch notation, as well as pitching the sound into every key.
4. Use a connected MIDI instrument to replay all the sounds to it’s corresponding key.
5. Apply dynamics to the sound.

The first step was implemented quite easily in pure data. With the help of the [adc~] object we could read audio from the computer microphone. We mainly used a built in webcam on a MacBook pro 13” (mid 2010 model) as a soundsource. Even though we could have used better microphones we wanted our application to work with low quality microphones since they are often included in most computers and mobile devices. We used the object [writesf~] to record and store an audio file from the input. The original file was named “foo.wav”

The next step was to determine the pitch of the audiofile. This was approximated by replaying the sound to the through the [fiddle~] object.

The third step was the biggest challenge. We managed to find three different patches that each could pitch the sound to the desired pitch. These patches were called “soundtouch~”, “fft-pitchshifter~” and “pitchshifter~”. A problem we encountered with the soundtouch patch was that it only allowed pitching a note at the most down two octaves and up a single octave. We solved this by connecting several instances of soundtouch objects into each other, allowing us to pitch the already pitched sound. The other patches were easier to implement. For all patches it was possible to pitch less than a semitone in order to fine tune to standard pitch notation. All samples was saved with each corresponding MIDI note name to .wav files. For example th file “fft69.wav” was an A4, midi note 69, using the “fft-pitchshifter~”. The algorithms are explained further in the coming section “The three different pitching algorithms”.

The fourth step was implemented using the “notein” object to send mididata and “readsf~” to replay the audio from the samples. This way it was possible to replay the samples with a midikeyboard.

In the fifth step we introduced a ADSR to post processing the audio being replayed.

Evaluation method

For our evaluation we examined the three different pitching algorithms. We made two different recordings for each algorithm. The first one with a sung note and the second one with a spoken phrase. The pieces performed was, for the sung note, Bach Prelude No.1 and, for the spoken phrase, Mr Sandman. They were recorded using a midifile so that the performances was the same for all of the audiofiles. We made a judgment on how we found the different sound. We then sent out a survey for our course comrades to make a judgment as well.

The three different pitching algorithms

In our implementation we needed to pitch the recorded audio several semi tones while still maintaining satisfiable quality. We managed to find three different methods for pitching audio. We used and compared all of them in our evaluation. They were all implemented to be used together with pure data. The three different pure data patches were:

1. “Soundtouch~” [1]
2. “fft-pitchshifter~”, using the TFT algorithm (Fast Fourier transform).
3. “pitchshifter~”

soundtouch~

The soundtouch patch used the SOLA[2] (synchronized overlap-add) algorithm to pitch the audio. The SOLA algorithm works by cutting out small, about 10-100 millisecond, pieces and then tying them together by skipping or adding copies of parts in between. This allows the piece to change length without changing in pitch or dropping in quality, after this procedure it’s just a matter of changing the length back to the original, the same way you would change the speed on a vinyl player for example. This results in pitched audio without any significant drop in quality. Soundtouch was implemented in pure data and c++. It was imported as a simple pure data object, and the audio was later processed in the c++ program. The c++ file was compiled for windows, mac and linux.

fft-pitchshifter~

The fft-pitchshifter patch used FFT algorithm (Fast Fourier transform) to pitch the audio. The algorithm in practise is very advanced, but the principle is to divide the audio into small samples, calculate all the individual sinusoids using FFT, and individually repitching the sinusoids.

[1] See reference 1
[2] See reference 3

This patch was implemented entirely using pure data, which can be useful to make the main application more compatible across platforms (maybe the c++ patch is not compatible for some platforms).

pitchshifter~
We found this patch searching on pure data forums for pitch shifters[3]. We could unfortunately not find any documentation from the author of what algorithm was used, but we will still decided to test it with our application.

[3] See reference 2

Result and discussion

The results from the survey can be seen in the *Table 1* below.

Table 1

Bach Prelude No.1	Mr Sandman
Version B	Version B
Version C	Version B
Version C	Version B
Version B	Version B
Version C	Version B
Version C	Version B
Version C	Version B
Version C	Version A
Version C	Version A
Version C	Version A
Version B	Version B
Version C	Version B
Version C	Version B
Version C	Version B
Version A	Version A
Version C	Version B
Version C	Version A
Version C	Version B
Version B	Version B

The overall for each of the recordings is (as seen in *Table 2* and *Table 3*):

Table 2

Bach Prelude No.1	Count	Proportion %
Version A	1	5.26
Version B	4	21.1
Version C	14	73.7

Table 3

Mr Sandman	Count	Proportion %
Version A	5	26.3
Version B	14	73.7
Version C	0	0

We can see, in *Table 2* and *Table 3*, that the favorite for Bach Prelude No.1 was the performance using algorithm “Version C” from the patch “pitchshifter~”, and the favorite for Mr

Sandman was the performance using algorithm “Version B”, the FFT algorithm from the patch “fft-pitchshift~”. Our initial hypothesis that “Version A”, using the SOLA algorithm from the soundtouch~ patch, would be the favorite was wrong. However, in our evaluation we simply asked which one of the performances sounded best overall. Judging from the comments and from our own views one of the biggest factors when it came to perceived sound quality seemed to be how the sound was cut rather than how it was pitched. The audio cutting process sometimes seemed to produce a “clicking” noise that was bothersome. This was not as apparent using the pitchshifter and fft-pitchshifter patches.

Conclusion

We have in this project developed an application that works to a satisfactory level. We can also conclude that using the voice in sampling provides great versatility even for non singers. The best choice of pitching algorithm for this application is in our opinion the “fft-pitchshifter~”. It was favorable among the users and provides versatility since it’s implemented only in pure data.

For further development of this product, better audio cutting technology could prevent the clicking noise in the beginning of the sample. Another important development could also be to improve transitions when looping a sample. This way infinite sustain could be implemented still sounding natural.

References

1. katja's homepage on sinusoids, complex numbers and modulation. Available at: http://www.katjaas.nl/pitchshift/soundtouch~.html. [Visited 2014-06-25].
2. PURE DATA forum~, Pitch-shift for realtime audio?. Available at: http://puredata.hurleur.com/sujet-719-pitch-shift-realtime-audio, second post. [Visited 2014-06-25].
3. Time and pitch scaling in audio processing, by Olli Parviainen. Available at: http://www.surina.net/article/time-and-pitch-scaling.html. [Visited 2014-06-25].