An Open-Source C++ Framework for Multithreaded Realtime Multichannel Audio Applications

Matthias Geier\textsuperscript{1} Torben Hohn\textsuperscript{2} Sascha Spors\textsuperscript{1}

\textsuperscript{1}Quality and Usability Lab
Telekom Innovation Laboratories
Technische Universität Berlin

\textsuperscript{2}Linutronix GmbH

Linux Audio Conference
April 15\textsuperscript{th}, 2012
APF
Audio Processing Framework

- collection of C++ code
- http://tu-berlin.de/?id=apf
- open source, GPLv3
- mostly platform-independent
  - tested on Linux and MacOSX
- quite generic
- heavy use of templates
- mostly header-only
APF Components

- MimoProcessor → *topic of this talk!*

- biquad & cascade of biquads (IIR filter)
  - several methods for denormal prevention

- (yet another) C++ wrapper for JACK

- iterators

- other tools

- delay line → *coming soon!*

- partitioned convolution (FIR filter) → *coming soon!*
MimoProcessor

Target Applications

- block-based audio applications
- many inputs/outputs
- both realtime and non-realtime
- interactive applications

example application areas:
- sound field synthesis
- multichannel echo cancelling
- beamforming
MimoProcessor Example Application

Generic MIMO system

\[
\begin{align*}
&h_{1,1}(t) \\
&h_{1,2}(t) \\
&h_{1,3}(t) \\
&h_{1,4}(t) \\
\text{output 1} \\
&h_{2,1}(t) \\
&h_{2,2}(t) \\
&h_{2,3}(t) \\
&h_{2,4}(t) \\
\text{output 2} \\
&\text{output 3} \\
&\text{output 4}
\end{align*}
\]
MimoProcessor Features

- different audio backends (realtime and non-realtime)
  - **JACK**
  - *Pure Data* (and *Max/MSP*) external using *flext*
  - *MEX*-file for *octave* (and *Matlab*)
  - read/write multichannel audio files
  - *PortAudio* → coming soon!
  - ...and users can provide their own backends!

- “automatic” multithreading
- optional crossfade
- dynamic number of inputs/outputs (if supported by the audio backend)
- safe communication between realtime and non-realtime threads
Realtime & Non-Realtime Threads

- realtime thread:
  - audio callback function
  - limited computation time per audio block
  - blocking functions are forbidden, e.g.
    - allocating/deallocating memory
    - reading/writing files/sockets
    - creating/joining threads
    - waiting for mutexes
    - ...

- non-realtime thread:
  - GUI, network, reading/writing files
  - everything else
MimoProcessor Components

- LockFreeFifo<Command*>  
  - push(), pop()  

- CommandQueue  
  - using 2× LockFreeFifo  
  - push()/wait(), process_commands()

- Command  
  - abstract base class  
  - execute(), cleanup()

- RtList<Item*>  
  - using CommandQueue  
  - add()/rem(), begin()/end()/size()

- Item  
  - abstract base class  
  - process()
MimoProcessor Components

- MimoProcessor\<Derived, policies see below >
  - interface_policy
    - jack_policy
    - pointer_policy
  - thread_policy
    - posix_thread_policy
  - sync_policy
    - posix_sync_policy
  - xfade_policy
    - raised_cosine_policy (default)
    - disable_xfade
Code Example

examples/jack_minimal.cpp

```
#include "apf/mimoprocessor.h"
#include "apf/jack_policy.h"
#include "apf/posix_thread_policy.h"
#include "apf/posix_sync_policy.h"

class MyProcessor : public apf::MimoProcessor<MyProcessor
   , apf::jack_policy, apf::posix_thread_policy, apf::posix_sync_policy>
{
    public:
        typedef MimoProcessorDefaultInput Input;
        class Output;

        MyProcessor();

        void process()
        {
            _process_list(_output_list);
        }

    private:
        rtlist_t _input_list, _output_list;
};
```
class MyProcessor::Output : public MimoProcessorOutput
{
    public:
    typedef MimoProcessorOutput::Params Params;

    explicit Output(const Params& p)
        : MimoProcessorOutput(p), _combiner(_parent._input_list, _internal, _parent)
    {}

    virtual void process()
    {
        _combiner.copy(my_predicate());
    }

    private:
    struct my_predicate
    {
        // trivial, all inputs are used
        bool operator()(const Input&) { return true; }
    };

    combine_channels<rtlist_t, Input, InternalOutput> _combiner;
};
MyProcessor::MyProcessor()
 : MimoProcessorBase()
 , _input_list(_fifo)
 , _output_list(_fifo)
{

   Input::Params ip;
   ip.parent = this;
   _input_list.add(new Input(ip));
   _input_list.add(new Input(ip));

   Output::Params op;
   op.parent = this;
   _output_list.add(new Output(op));

   this->activate();
}

int main()
{
  MyProcessor processor;
  sleep(30);
}
Parallel Processing

- **RtList<Item*>**: list of polymorphic base class pointers
- Virtual function `Item::process()`
- Items within one list are processed in parallel
- Fixed number of threads, specified by user
- Simple scheduling:
  - Each of the $N$ threads gets every $N$-th item
- One “main audio thread”, $N - 1$ “worker threads”
  - Communication via semaphores
Crossfade

- block-based processing
- parameter changes only at block boundaries
- artifacts due to discontinuities
- can be reduced by crossfade

- each block is processed twice
  1. with previous parameters, fade out
  2. with current parameters, fade in
- but: only if something actually changes
  - as noticed by CommandQueue

- crossfade is optional
  - can be switched off at compile time
    - MimoProcessor<..., disable_xfade>
Example

Near-Field-Compensated Higher Order Ambisonics

- implementation of a realtime NFC-HOA renderer
- circular loudspeaker array (2.5D)
- $M$-th order ($2 \times M + 1$ loudspeakers)
- stages of the algorithm (in RtLists):
  - $N$ sources/inputs
  - $N \times (M + 1)$ IIR filters
  - $M + 1$ objects which add contributions of sources per order
    - multiplication with 2 complex weighting factors
    - resulting in $2 \times M + 1$ values (per audio sample)
  - block size IFFTs (of length $2 \times M + 1$)
  - $2 \times M + 1$ outputs
- part of the SSR $\rightarrow$ coming soon!
  - What is the SSR? $\rightarrow$ see next page
The SoundScape Renderer (SSR)

- software tool for *object-based* realtime spatial audio reproduction
- several different reproduction methods
  - Binaural Renderer
  - Binaural Room Synthesis (BRS)
  - Wave Field Synthesis (WFS)
  - Vector Base Amplitude Panning (VBAP)
  - Ambisonic Amplitude Panning (AAP)
  - Generic Renderer
  - Binaural Playback Renderer (BPB)
  - NFC-HOA Renderer → *coming soon!*

- runs on Linux and Mac OSX
- uses the *Jack Audio Connection Kit* (JACK)
- graphical user interface (Qt) and network interface (TCP/IP)
- Free and Open Source Software (GPLv3)
- [http://tu-berlin.de/?ssr](http://tu-berlin.de/?ssr)
The SoundScape Renderer (SSR).

Graphical User Interface.
Important Notes

- compile with optimization!
  - e.g. g++ -03

- be aware of cache effects!
  - memory locality
  - false sharing

- look for bottlenecks with a profiler!
  - OProfile, gprof, ...
Conclusion

- **goal of MimoProcessor: to be ...**
  - unobtrusive
  - easy to use
  - re-usable in different contexts
  - easily extensible (e.g. by policy-based design)

- **parallelization: simple yet effective**
  - trade-off between effort (**to use, to implement**) and performance
  - significant gain in performance, e.g. for HOA renderer

- **unit tests are included**
  - using the *CATCH* framework

- **well documented**
  - *Doxygen* documentation also available at the website
Future Work

- Audio Processing Framework (APF)
  - include delay line and partitioned convolution
  - implement *PortAudio* policy

- SoundScape Renderer (SSR)
  - re-write core using the MimoProcessor
  - port all existing renderers
  - include the brand-new NFC-HOA renderer
Thank you very much for your attention!

Questions?

Website: http://tu-berlin.de/?apf
Blog: http://audio.qu.tu-berlin.de