**Real Time Audio Programming in C**

Markus Hädrich

markus.haedrich@tu-berlin.de

Thomas Resch

thomas.resch@campus.tu-berlin.de

**Goals and Topics of this Seminar**

• Pure Data

• Programming Experience in C

• Reusable, maintainable and readable source code

• Object oriented programming in C

• Understanding and writing Audio Plugins

• Some operating system basics: audio interrupt, stack & heap

• SDK/API independent code

• Documentation (Doxygen)

• Versioning (Git)

• Testing

**Organisation**

• Projektarbeit in Gruppen, maximal 4 Studentinnen/Studenten zusammen.

• Einzelleistungen müssen erkennbar und entsprechend gekennzeichnet sein.

• Bewertung erfolgt individuell.

• Test Environment ist Pure Data (Läuft unter Windows, Linux und OSX).

• Kein Support für die Einrichtung des Programming-Environment unter Windows!!

• Implementierung eines Algorithmus (filter, sound synthesis, ..) als Objekt für Pure Data.

• Die Gruppenarbeit muss als (Link zu einem) Git-Repository im Stil des rtap\_limiter~ Projekts eingereicht werden (wird auf dem Seminar Git zu finden sein).

• Trennung von SDK und Algorithmus, wie im rtap\_limiter~ Beispiel Projekt!

• Doxygen Comments und Dokumentation wie im rtap\_limiter~ Beispiel Projekt!

• Unit-Tests wie im rtap\_limiter~ Beispiel Projekt!

• GUI in Pure Data mit Kommentaren wie im Beispiel Projekt. Das Endergebnis muss ohne

Raten zu müssen bedienbar sein mit Beispiel-Presets und Sounds.

• Abgabe der Seminar-Arbeit bis zum Semesterende am **30.9.2021**! Es wird danach keine Möglichkeit zur Korrektur geben, daher möglichst schon früher abgeben um ggf. die Möglichkeit zu haben, unser Feedback einarbeiten zu können.

**Possible Topics**

• Noise cancelation

• Granular synthesis

• Port something from ak\_tools

• Source Separation

• Upmix algorithms

• Convolution Reverb

• Synthetic Reverb

• Dynamics

• Filter

• Sound Synthesis

**Why C/C++?**

(Almost) all SDK’s/API’s/Frameworks (Audio Unit, VST, JUCE, Pure Data, Unity, Max/MSP, Port Audio, ...) are in C or C++.

**Why Pure Data?**

Pure Data runs on all major operating systems (Linux, Windows and OSX).

It’s Open Source and free.

By using libPd, it is possible to run Pure Data Externals on almost every device (iOS, Android, Raspberry Pi, etc.)

**Useful Resources**

Brian W. Kernighan, Dennis M. Ritchie

“C Programming Language” [https://cs.indstate.edu/~cbasavaraj/cs559/the\_c\_programming\_lang uage\_2.pdf](https://cs.indstate.edu/~cbasavaraj/cs559/the_c_programming_lang%20uage_2.pdf)

- Standard reference work for C-programming

Jon Erickson

“Hacking – The Art of Exploitation” [https://leaksource.files.wordpress.com/2014/08/hacking-the-art-of- exploitation.pdf](https://leaksource.files.wordpress.com/2014/08/hacking-the-art-of-%20exploitation.pdf)

- First half is a nice introduction into C-programming

<https://www.tutorialspoint.com/cprogramming/index.htm>

- Very good introduction into C-programming

[https://www.cs.cmu.edu/afs/cs/academic/class/15213- s13/www/codeStyle.html](https://www.cs.cmu.edu/afs/cs/academic/class/15213-%20s13/www/codeStyle.html)

- Some style guidelines

<https://puredata.info/>

- Pure Data Home

<https://github.com/pure-data/externals-howto>

- How-to for Pure Data externals; including universal makefile for OSX, Linux and Windows; starting point for the class

[https://nontranscendentalexistence.wordpress.com/2012/07/27/de veloping-pure-data-extensions-in-visual-studio/](https://nontranscendentalexistence.wordpress.com/2012/07/27/de%20veloping-pure-data-extensions-in-visual-studio/)

- How-to for building Pure Data externals with Visual Studio under Windows

<https://ccrma.stanford.edu/software/stk/>

- good source for dsp code in C++

<https://juce.com/>

- JUCE Home

<https://cycling74.com/>

Max/MSP Home

<https://www.gribblelab.org/CBootCamp/7_Memory_Stack_vs_Heap.html>

- Memory Stack & Heap

[https://www.cs.cmu.edu/afs/cs/academic/class/15213- s13/www/lectures/18-allocation-basic.pdf](https://www.cs.cmu.edu/afs/cs/academic/class/15213-%20s13/www/lectures/18-allocation-basic.pdf)

- Memory Allocation

<http://www.rossbencina.com/code/real-time-audio-programming>

- 101-time-waits-for-nothing - Audio Interrupt

[https://www.cs.cmu.edu/afs/cs/academic/class/15213- s13/www/lectures/12-linking.pdf](https://www.cs.cmu.edu/afs/cs/academic/class/15213-%20s13/www/lectures/12-linking.pdf)

- Compiler & Linker

<http://libpd.cc/>

- libPd Home

**REAL TIME AUDIO PROGRAMMING, OPERATING SYSTEMS AND COMPILER**

<https://www.gribblelab.org/CBootCamp/7_Memory_Stack_vs_Heap.html>

[https://www.cs.cmu.edu/afs/cs/academic/class/15213- s13/www/lectures/18-allocation-basic.pdf](https://www.cs.cmu.edu/afs/cs/academic/class/15213-%20s13/www/lectures/18-allocation-basic.pdf)

- Memory Management, Stack and Heap

<https://www.tutorialspoint.com/cprogramming/c_type_casting.htm>

<https://www.tutorialspoint.com/cprogramming/c_pointers.htm>

- Functions & Variables, Call by value, call by reference, casting and pointer arithmetic

[https://www.cs.cmu.edu/afs/cs/academic/class/15213- s13/www/lectures/12-linking.pdf](https://www.cs.cmu.edu/afs/cs/academic/class/15213-%20s13/www/lectures/12-linking.pdf)

- Compiler & Linker

**Pure Data & LibpdExamples**

A simple Pure Data Synthesizer by Steven Nelson:

https://vimeo.com/154450683

PPP Looper Synth

by Germain Aubert & Berenger Recoules

https://www.youtube.com/watch?v=1P8a1XNBlrw

PPP Jam Session

https://www.youtube.com/watch?time\_continue=139&v=XEymJGu HoMU

Atlas – Anti Game Environment

By Binaura http://www.binaura.net/apps/atlas/

Raspberry Pi running Pure Data By Shawn Greenlee https://vimeo.com/50498017

Bela

https://bela.io/

https://blog.bela.io/2018/05/02/salt-a-programmable-eurorack- syntesizer/

https://www.youtube.com/watch?v=bo5ZEgBEapk https://www.youtube.com/watch?v=ivjHWiHN1UM https://blog.bela.io/2018/03/27/opal-rhythm-computor-dmx-krew/ https://www.youtube.com/watch?v=W\_sBoESqraE

**Pure Data**

Pure Data is a graphical programming environment (mainly) for audio signal processing. It has a control level layer (objects without ~) and a signal level layer (all objects with ~). Pure Data can be extended with objects written in C/C++.

Hello World as Pure Data Object:

#include "m\_pd.h"

static t\_class \*helloworld\_class;

typedef struct \_helloworld

{

t\_object x\_obj;

} t\_helloworld;

void helloworld\_bang(t\_helloworld \*x)

{

post("Hello world !!");

}

void \*helloworld\_new(void)

{

t\_helloworld \*x = (t\_helloworld \*)pd\_new(helloworld\_class); return (void \*)x;

}

void helloworld\_setup(void)

{

helloworld\_class = class\_new(gensym("helloworld"), (t\_newmethod)helloworld\_new, 0, sizeof(t\_helloworld), CLASS\_DEFAULT, 0);

class\_addbang(helloworld\_class, helloworld\_bang);

}

**Pure Data basics**

A window containing objects is called a Patch or Patcher. A new Patcher may be created using the menu-> File -> New or with the shortcut CTRL+ n (OSX COMMAND + n).

Inside a Patcher new objects can be created by using the Menu -> Put or using one of several shortcuts.

CTRL + 1 (OSX: COMMAND + 1) creates an empty object box.

Objects can be connected to each other with Patch Chords.

Patcher State

Patchers have 2 different states: Edit Mode or Locked, CTRL+e (OSX: COMMAND+e) switches beween the two modes (or select/deselect in the menu edit-> Edit Mode).

In Edit mode it is possible to create objects and connect them.

If the patcher is locked, the patcher can be “used”.

In order to use signal processing, the dsp button in the “Pd-window” (Pd’s console) must be checked. Be careful with Laptop speakers and ears: always decrease volume before starting the dsp in Pure Data.

**Basic Pure Data objects**

message – sends a message to an object

bang – sends the bang message to an object print – prints to Pure Data window

comment – for writing comments into a patcher

number box – float/int GUI number box +, - , \*, / - math operators

trigger

loadbang

osc~ - sinus waveform oscillator

phasor~ -rectangle waveform oscillator

dac~ - audio output, connects the patch with the selected soundcard +~, \*~, -~, /~ math signal operators