Teaching Sound Synthesis in C/C++ on the Raspberry Pi

Henrik von Coler and **David Runge**
Audio Communication Group, TU Berlin
voncoler@tu-berlin.de
david.runge@campus.tu-berlin.de

Abstract

For a sound synthesis programming class in C/C++, a Raspberry Pi 3 was used as runtime and development system. The embedded system was equipped with an Arch Linux ARM, a collection of libraries for sound processing and interfacing, as well as with basic examples. All material used in and created for the class is freely available in public git repositories. After a unit of theory on sound synthesis and Linux system basics, students worked on projects in groups. This paper is a progress report, pointing out benefits and drawbacks after the first run of the seminar. The concept delivered a system with acceptable capabilities and latencies at a low price. Usability and robustness, however, need to be improved in future attempts.

Keywords

Jack, Raspberry Pi, C/C++ Programming, Education, Sound Synthesis, MIDI, OSC, Arch Linux

1 Introduction

The goal of the seminar outlined in this paper was to enable students with different backgrounds and programming skills the development of standalone real-time sound synthesis projects. Such a class on the programming of sound synthesis algorithms is, among other things, defined by the desired level of depth in signal processing. It may convey an application-oriented overview or a closer look at algorithms on a sample-wise signal processing level, as in this case. Based on this decision, the choice of tools, respectively the programming environment should be made.

Script languages like Matlab or Python are widely used among students and offer a comfortable environment, especially for students without a background in computer science. They are well suited for teaching fundamentals and theoretical aspects of signal processing due to advanced possibilities of debugging and visualisation. Although real-time capabilities can be added, they are not considered for exploring applied sound synthesis algorithms in this class.

In a more application-based context, graphical programming environments like Pure Data (Pd), MAX MSP or others would be the first choice. They allow rapid progress and intermediate results. However, they are not the best platform to enhance the knowledge of sound synthesis algorithms on a sample-wise level by nature.

C/C++ delivers a reasonable compromise between low level access and the comfort of using available libraries for easy interfacing with hardware. For using C/C++ in the development of sound synthesis software, a software development kit (SDK) or application programming interface (API) is needed in order to offer access to the audio hardware.

Digital audio workstations (DAW) use plugins programmed with SDKs and APIs like Steinberg's VST, Apple's Audio Units, Digidesign's RTAS, the Linux Audio Developer's Simple Plugin API (LADSPA) and its successor LV2, which offer quick access to developing audio synthesis and processing units. A plugin-host is necessary to run the resulting programs and the structure is predefined by the chosen platform. The JUCE framework [34] offers the possibility of developing cross platform applications with many builtin features. Build targets can be audio-plugins for different systems and standalone applications, including Jack clients, which would present an alternative to the chosen approach.

Another possibility is the programming of Pd externals in the C programming language with the advantage of quick merging of the self-programmed components with existing Pd internals. FAUST [6] also provides means for creating various types of audio plugins and standalone applications. Due to a lack in functional programming background it was not chosen.

For various reasons we settled for the Jack API [18] to develop command line programs on a Linux system.

- Jack clients are used in the research on binaural synthesis and sound field synthesis, for example by the WONDER interface [15] or the SoundScape Renderer [14]. Results of the projects might thus be potentially integrated into existing contexts.
- Jack clients offer quick connection to other clients, making them as modular as audio-plugins in a DAW.
- The Jack API is easy to use, even for beginners. Once the main processing function is understood, students can immediately start inserting their own code.
- The omission of a graphical user interface for the application leaves more space for focusing on the audio-related problems.
- The proposed environment is independent of proprietary components.

A main objective of the seminar was to equip the students with completely identical systems for development. This avoids troubles in handling different operating systems and hardware configurations. Since no suitable computer pool was available, we aimed at providing a set of machines as cheap as possible for developing, compiling and running the applications. The students were thus provided with a Raspberry Pi 3 in groups of two. Besides being one of the cheapest development systems, it offers the advantage of resulting in a highly portable, quasi embedded synthesizer for the actual use in live applications.

The remainder of this paper is organized as follows: Section 2 introduces the used hard- and software, as well as the infrastructure. Section 3 presents the concept of the seminar. Section 4 briefly summarizes the experiences and evaluates the concept.

2 Technical Outline

2.1 Hardware

A Raspberry Pi 3 was used as development and runtime system. The most recent version at that time was equipped with 1.2GHz 64-bit quad-core ARMv8 CPU, 802.11n Wireless LAN, a Bluetooth adapter, 1GB RAM, 4 USB ports 40 GPIO pins and various other features [11].

Unfortunately the on-board audio interface of the Raspberry Pi could not be configured for real-time audio applications. It does not feature an input and the Jack server could only be started with high latencies. After trying several interfaces, a Renkforce USB-Audio-Adapter was chosen, since it delivered an acceptable performance at a price of 10 €. The Jack server did perform better with other interfaces, yet at a higher price and with a larger housing.

Students were equipped with MIDI interfaces from the stock of the research group and private devices. The complete cost for one system, including the Raspberry Pi 3 with housing, SD card, power adapter and the audio interface, were thus kept at about 70 €(vs. the integrated low-latency platform bela[5], that still ranks at around 120 €per unit).

2.2 Operating System

First tests on the embedded system involved Raspbian [12], as it is highly integrated and has a graphical environment preinstalled, that eases the use for beginners. Integration with the libraries used for the course proved to be more complicated however, as they needed to be added to the software repository for the examples.

A more holistic approach, integrating the operating system, was aimed at, in order to leave as few dependencies on the students' side as possible and not having to deal with the hosting of a custom repository of packages for Raspbian. Due to previous experience with low latency setups using Arch Linux [7], Arch Linux ARM [2] was chosen. With its package manager pacman [9] and the Arch Build System [1] an easy system-wide integration of all used libraries and software was achieved by providing them as preinstalled packages (with them either being available in the standard repositories, or the Arch User Repository[3]). Using Arch Linux ARM, it was also possible to guarantee a systemd [13] based startup of the needed components, which is further described in Section 2.5. At the time of preparation for the course, the 64bit variant (AArch64) - using the mainline kernel - was not available yet. At the time of writing it is still considered experimental, as some vendor libraries are not yet available for it. Instead the ARMv7 variant of the installation image was used, which is the default for Raspberry Pi 2.

2.3 Libraries

All libraries installed on the system are listed in Tab. 1. For communicating with the audio hardware, the Jack API was installed. The jackcpp framework adds a C++ interface to Jack. Additional libraries allowed the handling of audio file formats, ALSA-MIDI interfacing, Open Sound Control, configuration files and Fast Fourier Transforms.

Table 1: Libraries installed on the development system

Library	Ref.	Purpose
jack2	[18]	Jack audio API
jackcpp	[26]	C++ wrapper for jack2
sndfile	[19]	Read and write audio files
rtmidi	[32]	Connect to MIDI devices
liblo	[23]	OSC support
yaml	[20]	Configuration files
fftw3	[22]	Fourier transform
boost	[4]	Various applications

2.4 Image

The image for the system is available for download[1] and can be asked for by mailing to the authors in case of future unavailability. Installation of the image follows the standard procedure of an Arch Linux ARM installation for Raspberry Pi 3, using the blockwise copy tool dd, which is documented in the course's git repository [31].

2.5 System Settings

The most important goal was to achieve a round-trip latency below 10 ms. This would not be sufficient for real-time audio applications in general, but for teaching purposes. With the hardware described in Section 2.1, stable Jack server command line options were evaluated, leading to a round-trip latency of 2.9 ms:

```
/usr/bin/jackd -R \K
        -p 512 \
        -d alsa \
                -d hw:Device \
                -n 2 \
                -p 64 \
                -r 44100
```

[1] https://www2.ak.tu-berlin.de/~drunge/klangsynthese

As these settings were not realizable with the internal audio card, the *snd-bcm2835* module - the driver in use for it - was blacklisted using */etc/modprobe.d/**, to not use the sound device at all.

For automatic start of the low-latency audio server and its clients, systemd user services were introduced, that follow a user session based setup. The session of the main system user is started automatically as per systemd's user@.service. This is achieved by enabling the linger status of said user with the help of loginctl [8], which activates its session during boot and starts its enabled services.
A specialized systemd user service [30] allows for Jack's startup with an elevated CPU scheduler without the use of dbus [21]. The students' projects could be automatically started as user services, that rely on the audio server being started by enabling services such as this example:

```
[Unit]
Description=Example project
After=jack@rpi-usb-44100. service
[Service]
ExecStart=/path/to/executable \
    parameter1 \
    paramter2
Restart=on-failure
[Install]
WantedBy=default.target
```

2.6 Infrastructure

For allowing all students in the class the access via SSH, a WIFI was set up, providing fixed IP addresses for all Raspberry Pis. Network performance showed to be insufficient to handle all participants simultaneously, though. Thus, additional parallel networks were installed.

For home use, students were instructed to provide a local network with their laptops, or using their home network over WiFi or cable. Depending on their operating system, this was more or less complicated.

3 The Seminar

The seminar was addressed to graduate students with different backgrounds, such as computer science, electrical engineering, acoustics and others. One teacher and one teaching assistant were involved in the planning, preparation and execution of the classes.

The course was divided into a theoretical and a practical part. In the beginning, theory and basics where taught in mixed sessions. The fi-

nal third of the semester was dedicated to supervised project work.

3.1 System Introduction

Students were introduced to the system by giving an overview over the tools needed and presenting the libraries with their interfaces. Users of Linux, Mac and Windows operating systems took part in the class. Since many students lacked basic Linux skills, first steps included using Secure Shell (SSH) to access the devices, which proved to be hard for attendees without any knowledge on the use of the command-line interface. Windows users were aided to install and use PuTTY [10] for the purpose of connecting, as there is no native SSH client. After two sessions, each group was able to reach the Raspberry Pi in class, as well as at home.

3.2 Sound Synthesis Theory

Sound synthesis approaches were introduced from an algorithmic point of view, showing examples of commercially available implementations, also regarding their impact on music production and popular culture. Students were provided with ready-to-run examples from the course repository for some synthesis approaches, as well as with tasks for extending these.

Important fundamental literature was provided by Zölzer [35], Pirkle [27] and Roads [29]. The taxonomy of synthesis algorithms proposed by Smith [33] was used to structure the outline of the class, as follows.

A section on *Processed Recording* dealt with sampling and sample-based approaches, like wave-table synthesis, granular synthesis, vector synthesis and concatenative synthesis.

Subtractive synthesis and analog modeling were treated as the combination of the basic units *oscillators*, *filters* and *envelopes*. Filters were studied more closely, considering IIR and FIR filters and design methods like bilinear transform. A ready to run biquad example was included and a first order low-pass was programmed in class.

Additive Synthesis and *Spectral Modeling* were introduced by an analysis-resynthesis process of a violin tone in the SMS model [25], considering only the harmonic part.

Physical Modeling was treated for plucked strings, starting with the Karplus-Strong algorithm [17], advancing to bidirectional [24].

FM Synthesis [16] was treated as a representative of abstract algorithms in the class. The concept was mainly taught by a closer look at the architecture and programming of the Yamaha DX7 synthesizer.

3.3 Projects

Out of the 35 students who appeared to the first meetings, 18 worked on projects throughout the whole semester. It should be noted that (with one exception) only Linux and MAC users continued.

No restrictions were made regarding the choice of the topic, exept that it should result in an executable program on the Raspberry PI. The student projects included:

- A vector synthesis engine, allowing the mixture of different waveforms with a succeeding filter section
- A subtractive modeling synth with modular capabilities
- A physical string model, based on the Karplus-Strong Extended with dispersion filter
- A sine-wave Speech Synthesis [28] effect, which includes a real-time FFT
- A guitar-controlled subtractive synthesizer, using zero-crossing rate for pitch detection
- A wave-digital-filter implementation with sensor input from the GPIOs

In order to provide a more suitable platform for running embedded audio applications, one group used buildroot[2] to create a custom operating system.

4 Conclusions

The use of the Raspberry Pi 3 for the programming of Jack audio applications showed to be a promising approach. A system with acceptable capabilities and latencies could be provided at a low price. Accessibility and stability, however, need to be improved in future versions: A considerable amount of time was spent working on these issues in class and the progress in the projects was therefore delayed considerably. The overhead in handling Linux showed to be a major problem for some students and probably caused some people to drop out. A possible step would be to provide a set with monitor, keyboard and mouse, as this would increase accessibility. The stability of the Jack server needs to be worked on, as sometimes the hardware would

[2] `https://buildroot.org`

not be started properly, leading to crashing Jack clients.

In future seminars, which are likely to be conducted after these principally positive experiences, most of the issues should be worked out and the image, as well as the repository will have been improved.

References

[1] Arch Build System - ArchWiki. https://wiki.archlinux.org/index.php/Arch_Build_System.

[2] Arch Linux ARM homepage. https://www.archlinuxarm.org/.

[3] Arch User Repository. https://aur.archlinux.org/.

[4] Boost C++ Libraries - Homepage. http://www.boost.org.

[5] buildroot homepage. https://bela.io.

[6] FAUST - Homepage. http://faust.grame.fr/.

[7] Linux Audio Conference 2015 - Workshop: Arch Linux as a lightweight audio platform - Slides. http://lac.linuxaudio.org/2015/download/lac2015_arch_slides.pdf.

[8] loginctl man page. https://www.freedesktop.org/software/systemd/man/loginctl.

[9] Pacman homepage. https://www.archlinux.org/pacman/.

[10] PuTTY Homepage. http://www.putty.org/.

[11] Raspberry PI Homepage. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[12] Raspbian homepage. https://www.raspbian.org/.

[13] systemd homepage. https://www.freedesktop.org/wiki/Software/systemd/.

[14] Jens Ahrens, Matthias Geier, and Sascha Spors. The soundscape renderer: A unified spatial audio reproduction framework for arbitrary rendering methods. In *Audio Engineering Society Convention 124*. Audio Engineering Society, 2008.

[15] Marije AJ Baalman. Updates of the wonder software interface for using wave field synthesis. *LAC2005 Proceedings*, page 69, 2005.

[16] John M Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the audio engineering society*, 21(7):526–534, 1973.

[17] Julius O. Smith David A. Jaffe. Extensions of the Karplus-Strong Plucked-String Algorithm. *Computer Music Journal*, 7(2):56–69, 1983.

[18] Paul Davies. JACK API. http://www.jackaudio.org/.

[19] Erik de Castro Lopo. Libsndfile. http://www.mega-nerd.com/libsndfile/.

[20] Clark C. Evans. YAML: YAML Ain't Markup Language. http://yaml.org/.

[21] Free Desktop Foundation. dbus homepage. https://wiki.freedesktop.org/www/Software/dbus/.

[22] Matteo Frigo and Steven G. Johnson. FFTW Fastest Fourier Transform in the West. http://www.fftw.org/.

[23] Steve Harris and Stephen Sinclair. liblo Homepage: Lightweight OSC implementation. http://liblo.sourceforge.net/.

[24] Matti Karjalainen, Vesa Välimäki, and Tero Tolonen. Plucked-string models: From the Karplus-Strong algorithm to digital waveguides and beyond. *Computer Music Journal*, 22(3):17–32, 1998.

[25] Scott N. Levine and Julius O. Smith. A Sines+Transients+Noise Audio Representation for Data Compression and Time/Pitch Scale Modi cations. *Proceedings of the 105th Audio Engineering Society Convention*, 1998.

[26] Alex Norman. JackCpp. http://www.x37v.info/projects/jackcpp/.

[27] Will Pirkle. *Designing Software Synthesizer Plug-Ins in C++*. Focal Press, 2014.

[28] Robert E Remez, Philip E Rubin, David B Pisoni, Thomas D Carrell, et al. Speech perception without traditional speech cues. *Science*, 212(4497):947–949, 1981.

[29] Curtis Roads. *The computer music tutorial*. MIT press, 1996.

[30] David Runge. uenv homepage. https://git.sleepmap.de/software/uenv.git/about/.

[31] David Runge and Henrik von Coler. AK-Klangsynthese Repository. https://gitlab.tubit.tu-berlin.de/henrikvoncoler/Klangsynthese_PI.

[32] Gary P. Scavone. RtMidi. http://www.music.mcgill.ca/~gary/rtmidi/.

[33] Julius O. Smith. Viewpoints on the History of Digital Synthesis. In *Proceedings of the International Computer Music Conference*, pages 1–10, 1991.

[34] Jules Storer. JUCE. https://www.juce.com/.

[35] Udo Zoelzer, editor. *DAFX: Digital Audio Effects* . John Wiley & Sons, Inc., New York, NY, USA, 2 edition, 2011.