

OPENAV LAC 2026



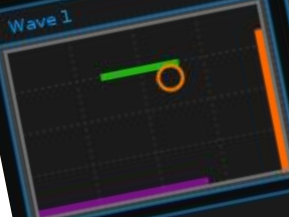
"Reflection on OpenAV Productions Software"

- + Live looping performance software
- + Audio FX plugins
- + Synths (wavetables + routing)
- + USB controller support
- + Open source + \$\$ release model
- + Linux focussed

SORCER

OPENAU

Wave 1



Wave 2



Sub-Bass



LFO



Remove



ADSR

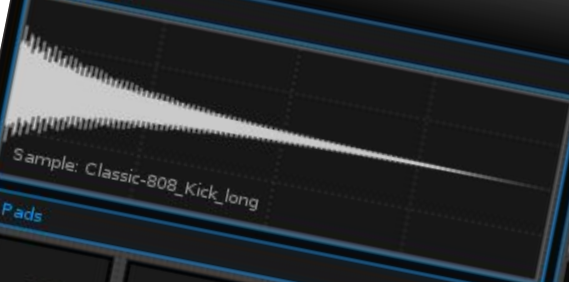


Master



FABLA

Waveform

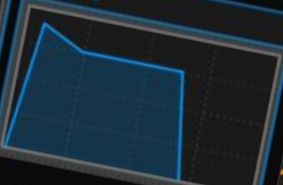


Pads

13 Classic-808_	14 Classic-808_	15 Classic-808_	16 Classic-808_
9 Classic-808_	10 Classic-808_	11 Classic-808_	12 Classic-808_
5 Classic-808_	6 Classic-808_	7 Classic-808_	8 Classic-808_
1 Classic-808_	2 Classic-808_	3 Classic-808_	4 Classic-808_

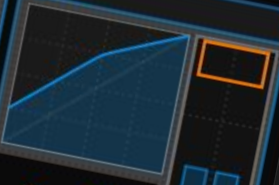
OPENAU

Source



Envelope controls: A, D, S, R, VOL, SPD, PAN

Master



Master controls: Ratio Thres, Atk, Rel

OPENAU

ADSR



ADSR controls: A, D, S, R

W1, W2, Speed, Amp

Remove



Remove controls: Cutoff, Modulation

Master



Master controls: Gain, Thres, A, R

LUPPP

OPENAU PRODUCTIONS

Drums



Sorcer



Fabla Pads



Fabla Bass



Guitar (Mic)



Cajon (Mic)



Voice (Mic)



Track 8



Master



Scene 1

Scene 2

Scene 3

Scene 4

Scene 5

Scene 6

Scene 7

Scene 8

Scene 9

Scene 10

Tap

Metro

BPM

Return

OPENAU



LUPPP

OPENAU PRODUCTIONS

Drums

Sorcer

Fabla Pads

Fabla Bass

Guitar (Mic)

Cajon (Mic)

Voice (Mic)

Track 8

Master



Scene 1

Scene 2

Scene 3

Scene 4

Scene 5

Scene 6

Scene 7

Scene 8

Scene 9

Scene 10

OPENAU

BITTA

ARTYFX
OPENAV

DELLA

ARTYFX
OPENAV

DRIVA

ARTYFX
OPENAV

DUCKA

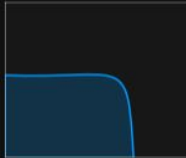
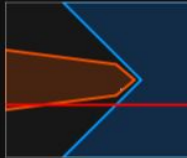
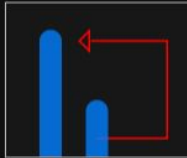
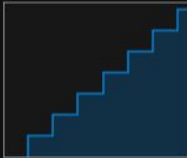
ARTYFX
OPENAV

FILTA

ARTYFX
OPENAV

KUIZA

ARTYFX
OPENAV



MASHA

ARTYFX
OPENAV

PANDA

ARTYFX
OPENAV

ROOMY

ARTYFX
OPENAV

SATMA

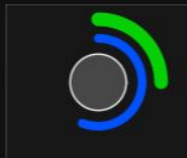
ARTYFX
OPENAV

VIHDA

ARTYFX
OPENAV

WHAAA

ARTYFX
OPENAV



Invert



Master

Gain

Tbres



Tap

BPM

return



FABLA 2.0

OPENAU PRODUCTIONS

OPENAU

OPENAU

LUPPP

Drums Sorcer

BITTA ARTYFX OPENAV

DELL

Tone

MASHA ARTYFX OPENAV

PANDA ARTYFX OPENAV

ROOMY ARTYFX OPENAV

SATMA ARTYFX OPENAV

VIHDA ARTYFX OPENAV

Time Pass Vol

Threshold Release Factor

Time Damp Dry Wet

Distortion Tone

Width Invert

Freq Drive Mix

Piano Roll Interface for FABLA 2.0. The interface shows a grid of notes with piano keys labeled A, B, C, D, 1, 2, 3, 4, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16. A list of layers is visible, including '36-Ludwig26Kick-5.wav', '36-Ludwis26Kick-1.wv', '36-Ludwis26Kick-2.wv', '36-Ludwis26Kick-3.wv', '36-Ludwis26Kick-4.wv', and '36-Ludwis26Kick-3.wv'. The interface also includes buttons for 'PANIC', 'Follow', 'REC', and 'Tone'.

Mixer Interface for OPENAU PRODUCTIONS. The interface shows a waveform display for '36-Ludwig26Kick-5.wav'. Below the waveform is a list of tracks with parameters for 'Gain / Pan', 'MIDI', 'Play', and 'Mute'. The interface also includes buttons for 'Follow', 'REC', and 'Tone'.

MASHA effect processor interface. It features a circular control knob with a green and blue arc, and three knobs labeled 'Time', 'Pass', and 'Vol'.

PANDA effect processor interface. It features a triangular control knob with an orange and blue arc, and three knobs labeled 'Threshold', 'Release', and 'Factor'.

ROOMY effect processor interface. It features a triangular control knob with a blue arc, and three knobs labeled 'Time', 'Damp', and 'Dry Wet'.

SATMA effect processor interface. It features a curved control knob with a blue arc, and two knobs labeled 'Distortion' and 'Tone'.

VIHDA effect processor interface. It features a double-headed arrow control knob with a blue arc, and a knob labeled 'Width'.

VIHDA effect processor interface. It features a triangular control knob with a blue arc, and three knobs labeled 'Freq', 'Drive', and 'Mix'.

LUPPP

Drums Sorcer



FABLA 2.0

Grid of 16 buttons labeled A, B, C, D, 13, 14, 15, 16, PANIC, Flows, and Pulse.



OPENAU PRODUCTIONS


Track 8 Master



OPENAU

NINJA

OSC 1, LFO 1, LFO 2, OSC 2, OSC 3, Filter, Master, Routing, ADSR 1, ADSR 2, ADSR 3, Effects



OPENAU

Master

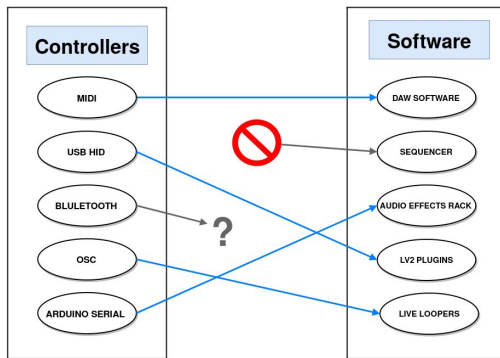
Routing section with a grid of colored buttons (orange, green, red) and a vertical slider.



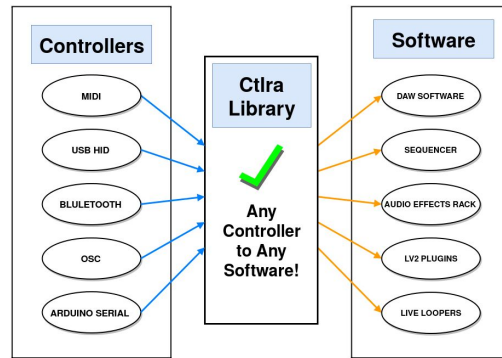
Ctla

(Hw Access)

WITHOUT CTLRA



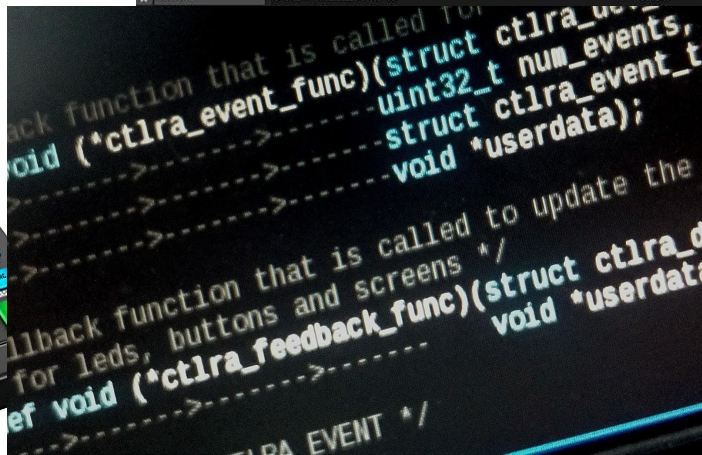
WITH CTLRA



"Next Gen" Controllers!

- USB HID devices
- "HD" Screens
- Huge functionality!
- Application support...

Mixxx (Ctrlra branch)



評価	ジャンル	キー	BPM	長さ	ビットレフ
.....	R&B	D ♭	116.085	04:37	160 m
.....			0	03:49	128 m
.....			0	03:00	192 m
.....	Rock		0	05:02	160 m
.....	Pop		0	06:09	192 m
.....	Pop		0	04:34	192 m
.....	J-Pop		0	03:58	192 m
.....			0	04:08	192 m
.....	Easy Listening		0	03:45	192 m
.....	Pop		0	04:09	192 m
.....	Metal	Cm	130.755	04:01	192 m
.....			0	08:43	192 m
.....			0	07:22	192 m
.....			0	05:25	192 m

3%

**Of all of the ideas I'd like to build...
motivation, time, and skills!?**

Community!

Motivation & Learning



2009



2010



2011



2012



2013



2014



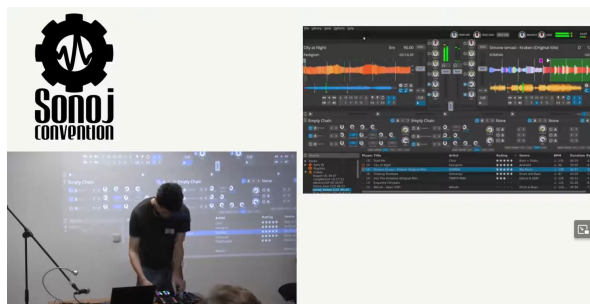
2015



2016 "minilac"



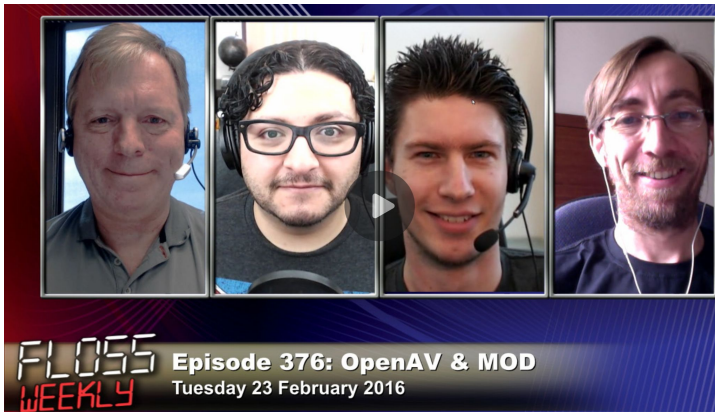
2017



SONOJ '17



2018



[Floss Weekly](#)



[MOD Device Collabs](#)



[Ubuntu User Mag](#)

[OpenAV Release System](#)



Harry van Haaren, open source musician



From issue 23 / 2014
The Producer
 Sam Tuke
 He performs drum 'n' b founder Harry van Haaren start model. His brand "Ope production, together wi dragged him away from Germany, to investigati







! LLMs !

"So what" for Linux Audio?

Ctrlra code review

SUMMARY

Severity	Count	Issues
 Critical	5	Memory leaks, use-after-free, buffer overreads, NULL dereferences
 High	5	Resource leaks, buffer overflows, missing bounds checks
 Medium	5	Integer overflows, inconsistent patterns, magic numbers
 Low	5	Code quality, documentation, dead code

(while watching the keynote yesterday? Context switching...)

Mem leaks, UseAfterFree, Buffer Overflows!

"Modern" software engineering?

Review the RTFX codebase in order to understand what DSP digital signal processing errors there might be in the DSP code, as well as any software bugs or issues like C++ memory leaks, etc.

SUMMARY OF FINDINGS				
Severity	Issue	File	Line	Status
CRITICAL	Variable shadowing in BitCrusher	dsp_bitcrusher.hxx	85	MUST FIX
CRITICAL	Buffer overflow risk in Delay	dsp_delay.hxx	45-47	MUST FIX
CRITICAL	Logic error in Distortion	dsp_distortion.hxx	120	MUST FIX
HIGH	Hardcoded sample rate in AnalogFilter	AnalogFilter.cxx	38-40	SHOULD FIX
HIGH	Memory management inconsistency	dsp_delay.hxx	45-47	SHOULD FIX
MEDIUM	Unused BPM handling	bitta.cxx	95-105	NICE TO FIX
MEDIUM	Numerical precision constants	global.h	25-30	NICE TO FIX
LOW	Commented code in Waveshaper	waveshaper.cxx	70-75	CLEANUP

<input type="checkbox"/>	CRITICAL BUGS SUMMARY - DSP Code Review	bug	dsp	enhancement	#57 · harryhaaren opened 1 hour ago	
<input type="checkbox"/>	HIGH: AnalogFilter Uses Hardcoded Sample Rate	bug	dsp		#56 · harryhaaren opened 1 hour ago	
<input type="checkbox"/>	CRITICAL: Distortion DSP Logic Error - Always Processes Audio	bug	critical	dsp	memory	#55 · harryhaaren opened 1 hour ago
<input type="checkbox"/>	CRITICAL: Delay DSP Buffer Overflow Risk	bug	critical	dsp	memory	#54 · harryhaaren opened 1 hour ago
<input type="checkbox"/>	CRITICAL: BitCrusher DSP Variable Shadowing Bug	bug	critical	dsp	memory	#53 · harryhaaren opened 1 hour ago

Now create a PR with one commit for each fix, tag the fix appropriately with a "fix #<id>" to auto close it on merge.

✓ PR Created Successfully!

I've created a **draft Pull Request** with **4 individual commits**, each addressing one of the critical DSP bugs identified in the code review.

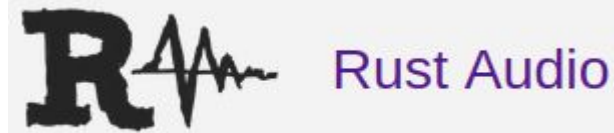
Technical Stuff

(Here be language wars ;)

Best tools for the job?

Something that doesn't let me make "silly" mistakes

- Silly mistakes in complex code-bases aren't silly
- But they are still mistakes, and cause bad UX/crashes...



how many of these bugs would exist if the code was written in Rust?

Rust's Safety Guarantees vs. C Bugs

● CRITICAL BUGS - All Prevented in Rust

Bug	Rust Prevention
1. Memory Leak in <code>ctla_dev_connect()</code>	✓ Prevented - Rust's ownership system ensures every allocation has a corresponding deallocation. The compiler will catch if a value is allocated but not consumed.
2. Use-After-Free in <code>ctla_dev_disconnect()</code>	✓ Prevented - Rust's borrow checker prevents using a value after it's been dropped. In <code>dev.disconnect(dev)</code> , <code>consume dev</code> , you cannot use it again.
3. Buffer Overread in <code>ctla_get_vendors()</code>	✓ Prevented - Rust slices (<code>&[T]</code>) carry their length and bounds are checked at runtime. Bounds-checked.
4. NULL Pointer Dereference in <code>ctla_impl_accept_dev()</code>	✓ Prevented - Rust has no null pointers. <code>Option<T></code> is used instead, and the compiler will catch dereferencing a <code>None</code> value before dereferencing.
5. Resource Leak in Device Connection	✓ Prevented - Rust's <code>Drop</code> trait ensures resources are cleaned up when a value goes out of scope. Resources would be wrapped in types that implement <code>Drop</code> .



💡 The Bottom Line

Approximately 65% of these bugs (13 out of 20) would be prevented at compile time in Rust.

The remaining 35% would be caught by Rust's tooling (Clippy, rustdoc, compiler warnings) during development.

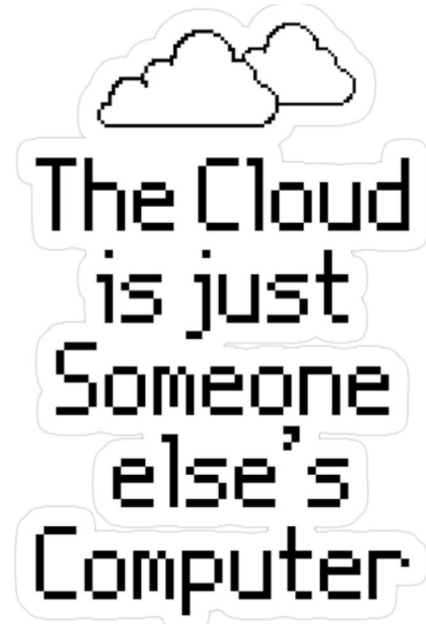
Zero of these bugs would make it to production in idiomatic Rust code.

Owning AI

... or depend on the cloud?

Owning AI

... or depend on the cloud?



Local GPUs?

Run LLMs locally on an open-source stack

~30 billion models locally : CPU only, or GPU

~16 VRAM GPU is enough - more is better!



Qwen



Gemma 4



llama.cpp

Communities & GPUs?

- **Community centric LLM inferencing**
 - Hackerspaces, "makerspaces", whatever :)
- **AI Models : new ones every day?**
 - Focus on "harness" not on model
 - Build workflows, do not focus on "prompting a model"
- **Play, Learn, Have fun**
 - Read blogs/newsletters, share experience

Positive Outlook

- **LLMs give new opportunities**

... "Mirrors of operator skill"

- **Rethinking SW engineering**

... building "agentic loops"

- **Own the hardware**

... own your coding agency

Redo OpenAV?

Start in 2026 (not 2009)

With LLM power

So what would I do exactly the same?

- + Keep attending LAC :)

Community is key

- + Attend other conferences (even more)

Wider perspectives & skills

So what would I do differently?

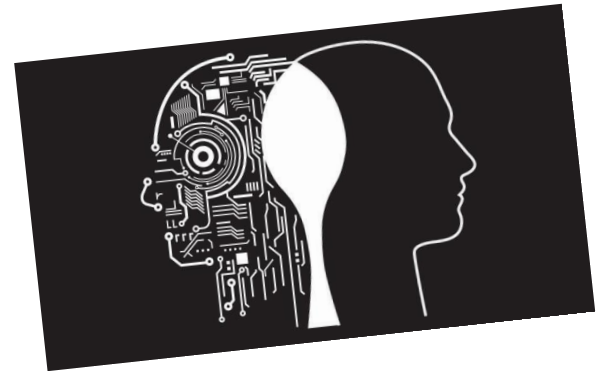
- + **Code in Rust**
 - + 90% of "silly" bugs go away!
- + **LLM (local?) to review every commit**
 - + Avoid "oopsie" logic errors
- + **More test/spec-driven development**
 - + Suits LLM agentic loop building workflows

! Thanks !

? Discussion ?



OPENAU



Bonus Slide



Summary: Rust Prevention Rate

Category	Total Bugs	Prevented in Rust	Caught by Tooling	Still Possible
● Critical	5	5 (100%)	0	0
● High	5	4 (80%)	1	0
● Medium	5	3 (60%)	2	0
● Low	5	1 (20%)	4	0
TOTAL	20	13 (65%)	7 (35%)	0