IMPLEMENTATION PLAN

QuadraFuzz v1

Modern Cross-Platform Reimplementation

NIH-plug (Rust) · VST3 / CLAP / AU
macOS (Apple Silicon + Intel) · Windows · Linux

Based on complete reverse engineering of the original
Steinberg / Spectral Design plugin (July 2000)

February 2025

Contents

Contents

1 Executive Summary

A faithful reimplementation of QuadraFuzz v1 as a cross-platform audio plugin targeting modern DAWs, operating systems, and hardware. The original's entire DSP pipeline has been fully reverse-engineered — no unknowns remain. This plan covers framework selection, architecture, DSP implementation, GUI design, testing, and release strategy.

2 Goals & Constraints

2.1 Primary Goals

1. **Bit-accurate DSP** — match the original's signal processing behavior exactly using the extracted wave-shape tables, filter topology, and gain structure
2. **Cross-platform** — macOS (Apple Silicon + Intel), Windows (x64), Linux (x64)
3. **Modern plugin formats** — VST3, CLAP, AU (macOS), standalone
4. **Modern GUI** — vector-based, resizable, with optional retro skin using extracted bitmaps
5. **Open source** — MIT or GPLv3 license

2.2 Non-Goals

- No attempt to replicate the DirectX/COM interface (obsolete)
- No attempt to replicate the Wise installer or registry behavior
- No copy protection or authorization system
- Not a clone of QuadraFuzz v2 (different, expanded plugin)

3 Technology Selection

3.1 Framework: NIH-plug (Rust)

Criterion	NIH-plug	JUCE (C++)	DPF (C++)
License	ISC (permissive)	GPLv3 or commercial	ISC
Formats	VST3, CLAP, standalone	VST3, AU, CLAP, standalone	VST2, VST3, LV2, CLAP
GUI	iced, vizia, egui	Built-in (excellent)	OpenGL, Cairo
Cross-platform	✓	✓	✓
Apple Silicon	✓ native	✓ native	✓
Memory safety	✓ (Rust)	×	×

Criterion	NIH-plug	JUCE (C++)	DPF (C++)
Build system	cargo	CMake/Projucer	Makefile/CMake
Community	Growing	Large, mature	Small

Recommendation: NIH-plug with Rust. Permissive license, memory safety, native CLAP support, excellent cross-compilation. GUI via **vizia** (integrated with NIH-plug, declarative, GPU-accelerated).

3.2 Build Targets

Platform	Architecture	Formats
macOS	aarch64 (Apple Silicon)	VST3, CLAP, AU, standalone
macOS	x86_64 (Intel)	VST3, CLAP, AU, standalone
Windows	x86_64	VST3, CLAP, standalone
Linux	x86_64	VST3, CLAP, standalone

Universal Binary for macOS (fat binary with both architectures).

4 Architecture

4.1 Crate Structure

```
quadrafuzz/
├── Cargo.toml                    # Workspace root
├── crates/
│   ├── quadrafuzz-dsp/           # Pure DSP library (no plugin deps)
│   │   ├── src/
│   │   │   ├── lib.rs
│   │   │   ├── crossover.rs      # Butterworth biquad crossover
│   │   │   ├── waveshaper.rs     # Table-lookup waveshaping
│   │   │   ├── processor.rs      # Per-band processing chain
│   │   │   ├── mixer.rs          # Band summing / solo
│   │   │   ├── tables.rs         # Embedded waveshape tables
│   │   │   ├── presets.rs        # Factory preset definitions
│   │   │   └── params.rs         # Parameter defs and conversion
│   │   └── Cargo.toml
│   └── quadrafuzz-plugin/        # NIH-plug wrapper + GUI
│       ├── src/
│       │   ├── lib.rs            # Plugin entry point
│       │   ├── editor.rs         # vizia GUI
│       │   └── editor/
│       │       ├── crossover.rs  # Crossover display widget
│       │       ├── band.rs       # Per-band controls
│       │       └── skin.rs       # Optional retro skin
│       └── Cargo.toml
├── data/                         # Extracted assets
```

```
├── tests/                          # Integration tests
└── docs/                           # RE documentation
```

4.2 Key Design Decisions

1. **DSP crate is framework-agnostic** — no NIH-plug dependency. Reusable in tests, CLI, WASM.
2. **f64 internal processing for filters** — matching original's double-precision biquad. Waveshaping uses f32.
3. **No heap allocation in audio path** — all buffers pre-allocated.
4. **No `unsafe` in DSP crate** — pure safe Rust.

5 DSP Implementation

All algorithms verified against the original binary's disassembly. See `dsp-routines.md` for complete pseudocode.

5.1 Crossover Filter Bank

Butterworth biquad IIR, Direct Form I, f64 coefficients and state, f32 I/O.

- Bandpass: order 5 (cascaded sections) · LP/HP: order 3
- Band width expansion: 1.15× · Nyquist guard: 0.48–0.49 × sample_rate
- Denormal protection: flush |y| < 1e-15 to zero
- Frequencies in log2(Hz) space

```rust
pub struct BiquadSection {
    b0: f64, b1: f64, b2: f64,
    a1: f64, a2: f64,
    x1: f64, x2: f64,
    y1: f64, y2: f64,
}

impl BiquadSection {
    pub fn process_sample(&mut self, x0: f64) -> f64 {
        let y0 = self.b0 * x0 + self.b1 * self.x1
                 + self.b2 * self.x2
                 - self.a1 * self.y1 - self.a2 * self.y2;
        self.x2 = self.x1; self.x1 = x0;
        self.y2 = self.y1; self.y1 = y0;
        y0
    }

    pub fn flush_denormals(&mut self) {
        if self.y1.abs() < 1e-15 { self.y1 = 0.0; }
        if self.y2.abs() < 1e-15 { self.y2 = 0.0; }
    }
}
```

5.2 Waveshaping Engine

5 tables × 257 floats (256 curve + 1 drive comp). Index: |sample| × 254.0. Linear interpolation. Odd symmetry. Hard clamp ±1.0.

```
pub fn process_sample(&self, x: f32) -> f32 {
    let sign = x.signum();
    let abs_x = x.abs();
    if abs_x >= 1.0 { return sign; }

    let table = self.tables[self.current_table];
    let scaled = abs_x * 254.0;
    let idx = scaled as usize;
    let frac = scaled - idx as f32;
    sign * (table[idx] + frac * (table[idx+1] - table[idx]))
}
```

5.3 Waveshape Tables

#	Name	Drive Comp	Source
0	Linear	2.0 dB	`data/wavetables/linear.csv`
1	Mild Overdrive	6.1 dB	`data/wavetables/mild_overdrive.csv`
2	Soft Saturation	7.4 dB	`data/wavetables/soft_sat.csv`
3	Tube Saturation	6.9 dB	`data/wavetables/tube.csv`
4	Hard Clip	10.2 dB	`data/wavetables/hard_clip.csv`

Note: The Tube table (index 3) is a sampled curve with a distinctive dip around index 100–110. It cannot be approximated by any simple function — must use extracted data directly.

5.4 Per-Band Processing

For each band, each channel, each sample:

```
output = waveshape(sample × input_drive × band_drive) × output_level × drive_comp
                                    × band_gain
```

All gains: `10^(dB / 20)`. Output mixer: additive band sum (normal) or single band copy (solo). No dry/wet mix.

6 Parameter System

6.1 Plugin Parameters (Host-Visible)

Parameter	Range	Default	Unit	Auto
Band 1–4 Gain	−12 to +12	0.0	dB	✓
Input Drive	−20 to +20	0.0	dB	✓
Output Level	−20 to +20	0.0	dB	✓
Shape Mode	0–4	0	enum	✓
Solo Band	−1 to 3	−1	int	✓
Freq 1–5	25–22050	20..20k	Hz	✓
Shape 1–4	−12 to +12	0	—	✓

The original's "Modify" indirection and Edit/Delete/Create params are replaced by the DAW's native preset browser.

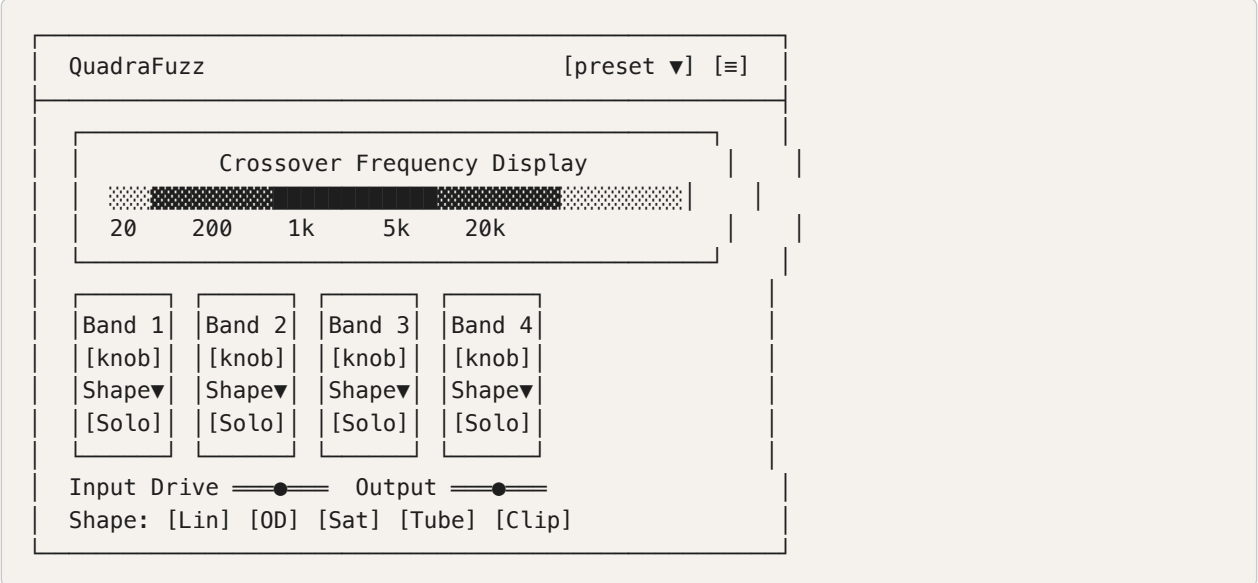
6.2 Factory Presets (16)

Drums (5): DrumSmasher, AnalogDrums, DrumsOfDoom, DrumSqueeze, Tightener

Guitar (11): GrungeKord, Resofuzz, BigNoiseKord, Acoustifuzz, KordBright, ChordRez, PowerChord, KordKrunch+Hi, Basic Lead, Cutting Lead, 60s Lead

7 GUI Design

7.1 Layout



7.2 Color Palette

Element	Color	Usage
Background	`#1a1a1a`	Main panel
Surface	`#2a2a2a`	Control areas
Accent	`#e8a020`	Active elements, highlights
Band 1	`#c05040`	Low frequency band
Band 2	`#e8a020`	Low-mid band
Band 3	`#40a060`	High-mid band
Band 4	`#4080c0`	High frequency band
Text	`#e0e0e0`	Labels, values

8 Implementation Phases

8.1 Phase 1: DSP Core (Week 1–2)

Task	Est.	Pri
Biquad filter section (Direct Form I, f64)	2h	P0
Butterworth coefficient computation	4h	P0
Crossover filter bank (N-band splitting)	4h	P0
Waveshape table embedding (5 × 257 floats)	1h	P0
Waveshaping engine (table lookup + interpolation)	2h	P0
Per-band processor (drive → shape → gain)	2h	P0
Output mixer (sum + solo)	1h	P0
Full processing chain integration	3h	P0
Factory preset definitions	1h	P0
Unit tests (impulse response, DC, sweep)	4h	P0
Total	**24h**	

8.2 Phase 2: Plugin Wrapper (Week 2–3)

Task	Est.	Pri
NIH-plug scaffolding + parameter declarations	3h	P0
Parameter → DSP mapping (dB conversion, etc.)	2h	P0
Sample rate change handling (filter rebuild)	2h	P0
Block size management	1h	P0

Task	Est.	Pri
Factory preset registration	1h	P0
State save/load (DAW session recall)	2h	P0
Basic DAW testing (REAPER, Bitwig, Logic)	3h	P0
Total	**14h**	

8.3 Phase 3: GUI (Week 3–5)

Task	Est.	Pri
vizia setup + main layout	4h	P0
Crossover frequency display (custom widget)	8h	P0
Draggable frequency handles	4h	P0
Per-band control strip (gain knob, shape, solo)	4h	P0
Global controls (input drive, output, shape mode)	2h	P0
Preset selector integration	2h	P1
Resizable window support	2h	P1
Visual polish + animation	4h	P1
Retro bitmap skin (optional)	6h	P2
Total	**36h**	

8.4 Phase 4: Testing & Validation (Week 5–6)

Task	Est.	Pri
Run original in Wine, capture reference signals	4h	P1
Automated comparison (< 0.01 dB tolerance)	4h	P1
Edge cases (silence, DC, Nyquist, extreme gains)	2h	P0
CPU profiling + optimization	3h	P1
Memory leak / safety audit	1h	P0
Multi-DAW compatibility testing	4h	P0
Total	**18h**	

8.5 Phase 5: Release (Week 6–7)

Task	Est.	Pri
CI/CD pipeline (GitHub Actions)	3h	P0
Cross-compilation (macOS Universal, Windows, Linux)	3h	P0
Installer / package scripts	2h	P1
README + user documentation	2h	P0

Task	Est.	Pri
License selection + legal review	1h	P0
GitHub release with binaries	1h	P0
Total	**12h**	

8.6 Timeline Summary

Phase	Duration	Cumulative
Phase 1: DSP Core	1–2 weeks	Week 2
Phase 2: Plugin Wrapper	1 week	Week 3
Phase 3: GUI	2 weeks	Week 5
Phase 4: Testing	1 week	Week 6
Phase 5: Release	1 week	Week 7

Total estimated effort: 104 hours over 7 weeks (part-time, 15h/week)

9 Performance Targets

Metric	Target	Original (est.)
CPU (stereo, 44.1 kHz)	< 1% single core	2–5% (PIII)
Latency	0 samples	0 samples
Memory	< 1 MB	500 KB
State size	< 2 KB	N/A
Build time (release)	< 60s	N/A

10 Testing Strategy

10.1 Unit Tests (DSP Crate)

- **Biquad**: impulse response matches analytic Butterworth
- **Waveshaper**: known input/output pairs from extracted tables
- **Crossover**: energy preservation (sum of bands ≈ input)
- **Full chain**: silence in → silence out, DC → expected DC
- **Denormal**: sustained silence doesn't cause CPU spikes

10.2 Integration Tests (Plugin)

- **Parameter sweep**: all parameters through full range without artifacts

- **Sample rate change**: no clicks, pops, or state corruption
- **Preset recall**: state save/load round-trips exactly
- **Block size variation**: consistent output regardless of size (1–8192)

10.3 Reference Comparison

- White noise → capture output → compare spectral characteristics
- Sine sweeps at various drive levels → compare harmonic content
- All 16 factory presets → verify output matches within tolerance

11 Future Enhancements (Post-v1.0)

Enhancement	Description
Oversampling	2×/4× for reduced aliasing at high drive
Per-band shapes	Independent waveshape mode per band
Spectrum analyzer	Real-time FFT in crossover view
MIDI learn	Map parameters to MIDI CC
Sidechain	External input for frequency-dependent processing
User tables	Loadable custom waveshape tables
A/B comparison	Quick toggle between two parameter states

12 Risks & Mitigations

Risk	Impact	Likeli.	Mitigation
Butterworth numerical issues at extreme freqs	High	Low	Validate against `biquad` crate
vizia limitations for custom crossover	Med	Med	Fall back to raw GPU / egui
Apple notarization	Med	Med	Code signing in CI
NIH-plug API changes	Low	Low	Pin dependency versions
Can’t run original in Wine	Med	Med	Use extracted tables + spot-checks