Infinite Research · Technical Report 1 · version 1.3 ·
https://infinitedevices.org/research/aliasing-aware-field.html

Aliasing-aware compilation for Field

Naman Soni

Infinite · infinitedevices.org

pre-registered 2026-11-21 · published 2026-11-21 · data and code at commit d370e296

ABSTRACT

Nonlinear audio code aliases, and the usual remedy, oversampling, is chosen by hand and does not compose: in our rig one saturator at 8x stays below −118 dB of alias, three in series measure −46 dB. We asked whether a compiler can choose the factor itself. Before measuring, we pre-registered two hypotheses with kill criteria: (H1) a static analysis predicts alias well enough to rank kernels (Spearman ρ ≥ 0.8) and (H2) choosing the factor per subgraph saves at least 25% of CPU against oversampling the whole kernel at the factor it needs. On a corpus of 47 kernels in Infinite's Field language, alias falls by about 6(k+1) dB per doubling of the rate, where k is the number of continuous derivatives, so kernels with jumps cannot be fixed by oversampling at all. A periodic steady-state analysis predicts alias with ρ = 0.994 and names the needed factor exactly for 43 of 43 kernels, but it is exact by construction for memoryless kernels. A pre-registered held-out test on 272 kernels the author did not tune against (32 from the literature, 240 generated) confirms H1 within its scope: on the 174 informative kernels scored, ρ is 1.000 and 0.994 on the two sets, and the exact factor is chosen for 174 of 174, against 8 for "8x if nonlinear" and 37 for a lookup by smoothness class. A pre-registered extension then treats a delay line as a periodic signal shifted by N samples: 94 of 97 delay kernels (including every ADAA kernel and a new blind set of 60) are scored with ρ 0.94 to 0.99 per set and the exact factor for 93 of 94, and no earlier result changes. The scope is a steady tone; three implicit literature circuits are still declined. A pre-registered real-input test (H5) on 147 kernels × 12 music clips, scored against an exact interleaved-copy oracle on the worst 100 ms window, fails: a multisine built from each clip is enough on 71.0% of pairs against a bar of 90%, today's one-tone predictor is short on 15.5%, and a third of pairs never reach −80 dB even at 32x. H2 is killed: per-segment plans saved 17.4%, and would reach the bar only if a resampler pair cost under 0.09 interpreter instructions; and on the motivating three-saturator chain the plan measured −43.0 dB against a predicted −80.5 dB, worse than a fixed 8x. Real half-band resamplers pass energy that the next stage folds back. What survives is a compile-time suggestion of one whole-kernel factor.

Keywords: aliasing, oversampling, DSP compiler, nonlinear audio, anti-aliasing, Field, negative result

1 Introduction

Nonlinear audio code (a `tanh`, a `clamp`, an `abs`, a product of two signals) creates harmonics above the Nyquist frequency. At a fixed sample rate they fold back into the audible band as inharmonic aliasing. The common remedy is oversampling: run the nonlinear code at a multiple of the sample rate, low-pass filter, and decimate. Antiderivative anti-aliasing (ADAA) reduces alias for a single memoryless nonlinearity without raising the rate [1, 2], and has been extended to stateful systems [3], recurrent networks [4] and polynomial approximations [5]. All of these are applied by hand, one nonlinearity at a time.

In Infinite, a person picks the factor per node; the Drive and Saturator nodes run at 8x. That choice does not compose. In our rig one Saturator stage at 8x (with ADAA) stays below −118 dB of alias, while three of the same stages in series, in one 8x domain, measure −46 dB. Field, Infinite's language for user-written kernels, makes this worse: its kernels are arbitrary code and its sample path has no oversampling at all.

This report asks whether a compiler can read a kernel, predict how much it aliases at each factor, and choose the cheapest factor per part of the kernel that meets a target. The target throughout is −80 dB below 20 kHz, the one Infinite's Analog node uses. Our contributions are a measured smoothness law for alias against rate on a corpus of 47 kernels (Section 4.1), a periodic steady-state alias predictor and its evaluation (Section 4.2), and a negative result for per-subgraph rate choice with its cause (Section 4.4), and an exact oracle for alias on real in-

put with a negative result for steady-tone prediction there (Section 4.5). The data, scripts and every figure's source are public (Section 9).

2 Pre-registered hypotheses

Both hypotheses, with the numbers that would kill them, were committed to the repository (commit `d38d29b7`) before the first measurement. Spearman's ρ [6] was chosen because a compiler needs the ranking of factors to be right more than the absolute dB.

	Claim	Metric and bar	Kill criterion
H1	A static analysis predicts alias.	Over ≥ 40 kernels, Spearman ρ ≥ 0.8 between predicted and measured alias; median absolute error ≤ 10 dB.	ρ < 0.8
H2	Choosing the factor per subgraph pays.	At equal measured alias, ≥ 25% less CPU than oversampling the whole kernel at the factor it needs, and it beats a fixed 8x on composed chains.	saving < 25%

No hypothesis is reported as confirmed without held-out validation; the corpus was written by the author of the analysis. After the corpus results, two more were pre-registered with the analysis frozen: H1b (H1's bars, met separately on two held-out sets) and H3 (the analysis picks the needed factor more often than two simple rules); Section 4.3.

3 Method

3.1 Corpus and stimulus

The corpus has 47 Field kernels: polynomials, `tanh` and sine folders at several drives, hard clips, quantisers, naive oscillators, feedback loops, linear filters around a nonlinearity, and six of Infinite's shipping presets. Each kernel is driven by a 5 kHz sine (bin 1707 of a 16 384-point frame at 48 kHz, so one frame holds a whole number of periods) at amplitude 0.5.

3.2 Measuring alias

Each kernel runs through the real Field Effect node at 1x, 2x, 4x, 8x, 16x and 32x of 48 kHz and is decimated with Infinite's minimum-phase filter chain. After warm-up, one steady-state frame is transformed. Every bin that is not a harmonic of the 5 kHz tone, below 20 kHz, counts as alias; the figure reported is its energy relative to the output's total, in dB. The needed factor is the smallest factor at or below −80 dB.

3.3 Predicting alias without rendering

A kernel driven by one steady tone produces a periodic signal, so the pass computes that periodic steady state over one input period (8192 points) with the real interpreter, then folds every harmonic analytically for every factor. Feedback through state cells is solved as a periodic fixed point. Above the grid's band, a power law fitted to the top two octaves extends the spectrum, which is what gets jump kernels right at 32x. The pass declines kernels whose output is not a function of the input phase: delay lines, tables, random numbers, beat-clock reads, and feedback with no stable periodic solution. The idea of estimating a sufficient rate from the structure of a nonlinear system is old [7, 8]; interval analysis of signal ranges in Faust [9] and its per-block rate work [10] are the nearest compiler-side ideas we found.

For a kernel with no memory driven by the rig's own tone, this prediction is exact by construction: sampling at the base rate samples the ideal periodic signal. The informative cases are kernels with state, phase accumulators, or a jump whose tail is extrapolated.

3.4 Per-segment rate plans

The planner cuts a compiled kernel into at most four segments next to its nonlinear operations (at most three cuts, at most two values crossing each cut). Each segment runs at its own factor, 1x to 32x, and each crossing

value goes down to the base rate and back up through Infinite's resampler. Plans are scored with the phase-1 analysis, weighted by how sensitive the output is to each crossing value, and the cheapest plan predicted at −80 dB or below is chosen. Every chosen plan is then rendered through the real resamplers, and **a plan only earns its saving if its measured alias meets the target**; otherwise the kernel is charged the whole-kernel cost.

CPU is a cost model, not a timing of the app's audio thread: nanoseconds per base-rate sample for one interpreter instruction, and for an up- and a down-sampler at each factor, from an isolated −O2 microbenchmark (best of five trials). The verdict run used one measurement of that model. To give the saving a spread, the model was re-measured in 10 independent runs and the rig re-run under each (Section 4.4).

4 Results

4.1 Alias follows smoothness

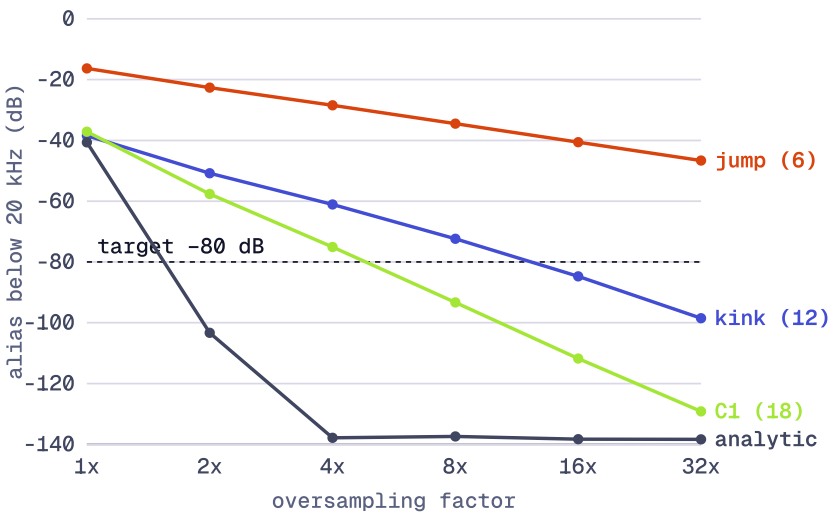

Figure 1. Measured alias against oversampling factor for one kernel of each smoothness class (jump: comparator; kink: hard clip; C1: x/(1+|x|); analytic: tanh at drive 4). The number after each class is the slope, in dB per doubling, that the smoothness law predicts. Data: phase0-baseline.csv.

How smooth the nonlinearity is sets the slope (Figure 1). Alias falls by about 6(k+1) dB per doubling of the rate, where k is how many derivatives stay continuous. Jumps (`floor`, comparisons, `%`) fall 4.9 to 6.1 dB per doubling, kinks (`abs`, `max`, hard clip) 11.8 to 12.9, and a C1 function such as x/(1+|x|) 17.4 to 17.9. Analytic functions such as `tanh` fall off a cliff once the rate clears a bandwidth that depends on drive. Three consequences follow.

- **Oversampling cannot fix jumps.** No jump kernel, and neither the naive saw nor the naive square, reached −80 dB even at 32x; the best was −50 dB. For these a compiler should not oversample but say why, and point to band-limited steps or ADAA.
- **The needed factor varies widely.** For −80 dB: 1x for 8 kernels, 2x for 9, 4x for 3, 8x for 6, 16x for 6, 32x for 5, and never for 7. One global factor wastes CPU on most kernels or fails the rest.
- **Shipping presets alias audibly at 1x.** Tube Overdrive measured −35 dB and Wavefolder −22 dB.

4.2 The predictor

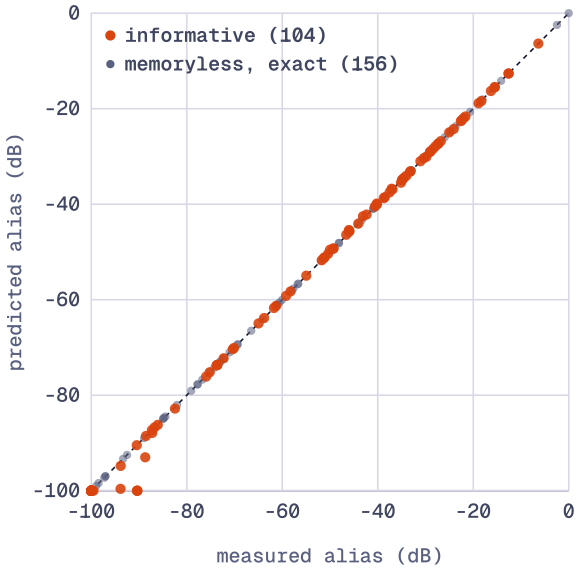

Figure 2. Predicted against measured alias for the 43 predicted nonlinear kernels at six factors (260 points, both sides floored at −100 dB, as scored for H1). On the diagonal the prediction is exact. Grey points are memoryless kernels (poly, analytic, kink, chain), for which the prediction is exact by construction (Section 3.3); the orange points (state, phase accumulator, jump tail, preset) are the informative ones. Data: phase1-predictions.csv.

Over 43 predicted kernels at six factors, the rank correlation is ρ = **0.994** (bar 0.8), the median absolute error **0.0 dB** (bar 10), and the needed factor is predicted exactly for **43 of 43** kernels (Figure 2). The analysis costs about 4.6 M interpreter runs for the whole corpus, against 91 M for rendering every factor.

This number is weaker than it looks, for three reasons. First, the bar was set for an approximate analysis and is cleared trivially by the memoryless kernels, which are exact by construction. On the informative cases the error is still small: feedback (3 kernels) 0.0 dB, linear filters around a nonlinearity 0.0 to 0.5 dB, and the extrapolated tail for jumps 0.1 to 0.6 dB at 32x (Figure 3). Second, the pass knows the stimulus exactly and a compiler does not: sweeping input amplitude (0.25, 0.5, 1.0) and frequency (0.5 to 5 kHz) changed the needed factor for 31 of 38 kernels, 29 of them upward. A shipped version would have to plan at the worst corner of a declared input range. Third, the corpus was written by one person. On 18 shipping Field Effect presets that were not in it, the pass accepted 2 (factor exact, worst error 0.1 dB) and declined 16: 7 read state tables, the rest have no periodic steady state under a steady tone. Two of eighteen cannot carry a general claim, so H1 was not confirmed on this corpus alone; Section 4.3 reports the held-out test.



Figure 3. Measured (solid) and predicted (dashed) alias per factor for four kernels where the prediction is not exact by construction. Measured: the real Field Effect node rendered at each factor and decimated. Predicted: the phase-1 pass, from the program without rendering. Data: phase1-predictions.csv.

The analysis also found a bug. Two presets came out 2 dB cleaner than measured; their `drive` knob held 3.0, not its declared 2.5, because when Field recompiles a parameter keeps its value by name, so a kernel inherits any same-named knob from the one before. Picking a preset in the app goes through the same path. The data uses corrected values; the app fix is not yet released.

4.3 Held-out validation

After the corpus results, and with the analysis frozen, we pre-registered a second test (H1b, H3) and committed two held-out sets before scoring them. **L** has 32 kernels written from equations in nine papers and textbooks (waveshapers, diode and pedal models, ladder and state-variable filters, ADAA clippers), without looking at the analysis or the data. **G** has 240 kernels drawn from Field's sample-rate grammar with a fixed seed, at least half with `state` or `delay`. H1b asks for ρ ≥ 0.8 and a median error ≤ 10 dB on each set's informative (stateful) kernels; H3 asks the analysis to pick the exact factor on at least 80% of them and more often than two simple rules: A, “8x if the kernel has any nonlinear operation”, and B, the median factor of the kernel's smoothness class in the corpus.

Set	Kernels	Scored	Informative	ρ (bar 0.8)	Median error (bar 10 dB)	H1b
L, literature	32	18	7	1.000	0.0 dB	pass
G, generated	240	215	167	0.994	0.0 dB	pass

H1b and H3 pass. The analysis picks the exact factor for 174 of 174 informative kernels; rule A for 8 and rule B for 37. Sixty-eight of these kernels never reach −80 dB within 32x (runaway feedback in G) and both sides say so; without them the count is 106 of 106 against 8 and 9. Every kernel with a median error above 10 dB is one of those that never gets clean, and 29 of them predict meaningless values near +2900 dB from a diverging solve that should have been declined. Where measured alias exceeds the prediction by more than 3 dB, the measurement is at the rig's float32 floor (never above −125.9 dB).

Coverage is the limit. The pass declines kernels with delay lines, so 12 literature kernels (every ADAA kernel, the pedal circuits, Timoney's configurations) and 25 generated ones were not scored; two semi-implicit diode clippers had no periodic steady state. The seven informative literature kernels are all state-only filters. A delay of N samples is a phase shift of a periodic signal, so we pre-registered a third test (H4) before changing the analysis: each `delay(x, N)` becomes one more periodic cell, shifted by N samples instead of state's one, and a new blind set **D** of 60 delay kernels (feed-forward combs, feedback through a delay, the first-order ADAA shape) was

committed first. H4 asks for 80% coverage of delay kernels, H1b's bars per set and H3's rule pooled, and no change to anything scored before.

Delay kernels	Total	Scored	ρ (bar 0.8)	Median error	H4
L, literature	12	11	0.937	0.0 dB	pass
G, generated	25	25	0.991	0.0 dB	pass
D, delay set	60	58	0.988	0.0 dB	pass

H4 passes. 94 of 97 delay kernels are scored and the exact factor is chosen for 93 of 94, against 9 for rule A and 38 for rule B; none of the 233 kernels scored before changes. The one miss is conservative: Parker's ADAA hard clip needs 4x and the analysis says no factor reaches −80 dB. Eighteen delay kernels have outputs that grow without bound; both sides say "none", but the analysis prints meaningless values near +2900 dB, and without them coverage would be 78%. Three gaps above the float32 floor remain, all below the −80 dB target. The five kernels still declined have no periodic steady state; three are implicit circuits (a pedal and two diode clippers) solved with a one-step update, which is the zero-delay-loop problem. **H1 is confirmed for single-tone input**, including delay lines and ADAA, not for arbitrary input.

4.4 Per-segment oversampling

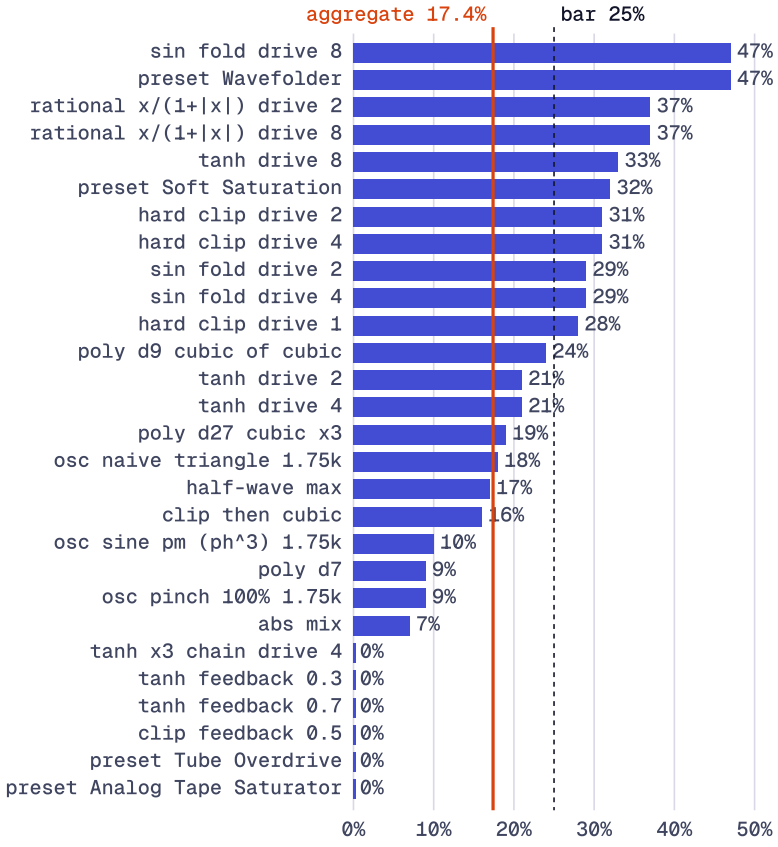

Figure 4. CPU saved by the per-segment plan against oversampling the whole kernel at the factor it needs, for the 28 kernels that needed oversampling and had a factor within 32x. A plan that missed −80 dB when measured is charged the whole-kernel cost (0%). The aggregate is cost-weighted. Data: phase2-h2-run.log.

H2 fails. The aggregate saving is 17.4% (median 19.5% per kernel) against a 25% bar (Figure 4). Re-measuring the cost model in 10 independent runs and re-running the rig under each gives 15.8% (95% CI 15.6 to 15.9), also below the bar (Table 1). Scaling every resampler cost down by a factor s, the saving reaches 25% only below s = 0.0156, where a 2x pair costs 0.09 interpreter instructions; the three-stage chain never beats a fixed 8x at any s. The repeats measured a cheaper interpreter instruction (about 2.7 ns, against 3.5 ns in the verdict line), which lowers the saving; the verdict line is kept as the pre-registered result. Staging pays where a cheap linear stage

surrounds one nonlinearity, because the linear part runs at 1x. But a resampler pair costs as much as 6 to 14 interpreter instructions at the same factor, and most Field kernels are short; the fixed cost of each crossing eats most of what staging saves.

Table 1. The cost model re-measured in 10 independent runs of the –O2 microbenchmark (each the best of 5 trials), and the H2 rig re-run under each. Aggregate saving 15.8% (95% CI 15.6 to 15.9) against the 25% bar. Data: phase2-cost-repeats.csv.

run	ns / instr	up 2/4/8x	down 2/4/8x	aggregate saving	median	tanh x3 beats 8x
1	2.67	10.1/21.4/40.7	8.9/20.7/44.3	15.7%	17.8%	no
2	2.66	10.1/21.5/40.8	9/20.8/44.2	15.9%	18.5%	no
3	2.66	10.7/23/42.5	9.5/22.2/46.4	15.3%	17.8%	no
4	2.67	10.1/21.6/40.9	8.9/20.8/44.2	15.6%	17.5%	no
5	2.67	10.1/21.5/40.9	8.9/20.9/44.3	15.9%	18.7%	no
6	2.66	10.1/22/41.5	8.9/20.8/43.9	15.9%	18.7%	no
7	2.65	10.1/21.5/40.8	8.9/20.7/43.9	15.9%	18.7%	no
8	2.66	10/21.5/40.7	8.9/20.8/44.3	15.9%	18.7%	no
9	2.76	10.3/21.8/40.7	9.3/20.8/44.6	15.9%	18.0%	no
10	2.75	10.4/22.1/41.9	9.2/21.2/45.4	15.7%	17.7%	no
mean over 10 runs (95% CI, t with 9 df)				**15.8% ± 0.1**	**18.2%**	

The three-stage `tanh` chain, the case that motivated the study, failed outright (Figure 5). The planner chose 2x, 4x, 8x and 1x segments at a modelled cost of 597 against 4272 for the whole kernel at 32x, and predicted –80.5 dB. Measured, it gave –43.0 dB, worse than a fixed 8x (–46.1 dB at cost 1048). Rendering the same plan with ideal brick-wall resamplers gives –74.0 dB, so about 31 of the 37 dB gap is the real half-band filter: its transition band passes energy around 20 to 28 kHz, which the next `tanh` folds back. The analysis treated each crossing as an ideal low-pass. Modelling each crossing's measured response would make these plans honest, but it would raise their factors, not the saving.

render	measured alias	audio
No oversampling (1x)	-12.6 dB	phase2-audio/tanh3-1x.wav
Whole kernel at a fixed 8x	-46.1 dB	phase2-audio/tanh3-fixed8x.wav
The planner's staged plan (2x | 4x | 8x | 1x)	-43.0 dB	phase2-audio/tanh3-staged.wav
Whole kernel at 32x (reference)	-136.0 dB	phase2-audio/tanh3-32x.wav

Figure 5. The three-stage tanh chain on the rig's 5 kHz test tone, rendered by the real resamplers. Each clip is one steady-state period (16 384 samples at 48 kHz) repeated to 2 s, peak-normalised to –18 dBFS with 10 ms fades; the dB figure is the alias the rig measured on that render. Aliases are the faint tones that do not belong to the 5 kHz harmonic series. Data: phase2-audio/renders.log and the .f32 and .wav files beside it.

Staging also changes the sound of chains. Band-limiting between stages removes ultrasonic content that the next stage would otherwise have folded into harmonics. Against a 32x render of the whole kernel, staged chains differ by –33 to –46 dB in harmonic amplitude; whole-kernel oversampling at any factor differs by less than –140 dB.

4.5 Real input

Every result above uses one steady tone. Music is not one tone, so we pre-registered a fourth test (H5) before running it. Twelve clips (drums, bass, plucks, chords, noise and a pad, each at peak 0.9 and 0.25, 16 384 samples) were written as Field programs and committed first. 147 nonlinear kernels (all of L and D, 60 of G by a fixed seed) were run on all 12, for 1764 kernel-clip pairs.

An exact oracle. A kernel's alias on a clip has no closed form, so it is measured. The clip is upsampled 64x; for factor F, 64/F interleaved copies of the kernel each run at F times 48 kHz on one phase of the 64x signal, so together they give the true output at every factor from the same input. Test and reference are both band-limited to 20 kHz, and the metric is the worst 100 ms window within 30 dB of the loudest: what a listener hears is the worst moment, not the average. The needed factor is the smallest of 1x to 32x at or below −80 dB.

H5 asks whether a predictor built from the clip, a multisine with the clip's power spectrum on a 46.875 Hz grid (P6), is enough (at or above the needed factor) on ≥ 90% of pairs and exact on ≥ 70%, and beats today's predictor (P0, one 5 kHz tone at 0.5); it is killed below 70% enough.

Predictor	Pairs	Enough (bar 90%)	Exact (bar 70%)
P0, today: one tone	1764	84.5%	47.9%
Rule A, 8x if nonlinear	1764	52.1%	9.1%
P1, loudest of 21 tones	1680	98.6%	39.9%
P6, multisine from the clip	1764	**71.0%**	**67.2%**

H5 fails, one point above its kill line. Real input needs far more than a tone: of the 1764 pairs, 282 need 1x and 603 do not reach −80 dB even at 32x, and one kernel (Parker's ADAA hard clip) needs 1x on quiet chords and 16x on loud noise. P6 is the most exact predictor, but it is one octave short on 384 of its 511 misses. It is enough on chords (93.9%) and noise (86.4%) and short on bass (45.9%) and plucks (58.8%), the clips whose loudest moment is much brighter than their average; P6 is built from the whole clip and the metric scores the worst window, which is the likely cause but was not tested. P1 looks safe only because it refuses: it says no factor will do for 89 of the 140 kernels it scores, and where it does name one it spends 3.1 times the CPU needed. Today's predictor, P0, picks too low a factor on 15.5% of real-input pairs.

A follow-up that builds the multisine from the three loudest windows (H6) was pre-registered with twelve new confirmation clips, committed and measured, but not run; the study closed before it. Those clips' measurements are published unscored, so they remain an unseen test set.

5 Discussion

The per-subgraph claim is dropped, and the planned next phase (rate regions on the live audio path) will not start. What is worth shipping is the simpler result: the phase-1 pass as a compile-time suggestion of one **whole-kernel** factor for Field Effect, planned at the worst corner of a declared input range, with a plain message when a kernel contains a jump that oversampling cannot fix. Two things would change the H2 verdict: much cheaper crossings (the break-even is a 2x resampler pair under 0.09 interpreter instructions, under 2% of today's 6), or much longer kernels with one nonlinearity inside a lot of linear code. Feedback loops, where bandwidth circulates through `state` and `delay`, are a separate question and remain open. Real input limits the suggestion further: no tested predictor is both safe and cheap on music, so the suggested factor should be labelled as tuned for a steady tone, and a guaranteed bound over a declared input level and bandwidth is the open problem this study leaves.

6 Threats to validity

- **Construct.** Phases 1 to 5 measure alias with one tone at one amplitude, and the needed factor moves with the stimulus (Section 4.2); the real-input test (Section 4.5) uses twelve synthetic music clips written by the author, not recordings. The metric counts modulation sidebands and limit cycles as alias; the Moog ladder preset reads −29 dB at 1x and 2x for that reason and was excluded. Kernels with a float32 phase accumulator (the naive oscillators) cannot measure below about −100 dB, because the accumulator's rounding is itself inharmonic; memoryless and filter kernels measure down to about −138 dB. Predictions and measurements are both floored at −100 dB for scoring.

- **Internal.** H2's CPU figures come from a cost model, not from timing the audio thread. A cheaper resampler would raise the saving; a compiled backend would make instructions cheaper and crossings relatively dearer, which would lower it. The cost model's spread across runs is reported (Table 1); the model itself was measured on one machine (Apple M2).
- **External.** One author wrote the corpus and the literature set (from papers, blind to the analysis), and the generated set comes from a grammar the same author wrote. The delay set D comes from a grammar by the same author. Implicit (zero-delay) circuits are outside the tested scope. Kernels are short.

7 Related work

ADAA [1, 2] and its extensions [3, 4, 5] attack alias at a single nonlinearity without raising the rate; they are complementary to choosing a rate, and the right tool for the jump kernels oversampling cannot fix. The rule that a polynomial of degree d needs d times the bandwidth is folklore; a public statement of it is [8]. Estimating a sufficient sampling rate for a nonlinear system was posed as a research topic in [7]. Faust computes signal intervals for fixed-point formats [9] and has explored per-block rates [10]. We found no system that combines a rate analysis with oversampling choice for audio, though our search was not exhaustive. Every idea here came from papers and public discussions; no other project's source code was opened.

8 Conclusion

A compiler can predict how much nonlinear audio code, delay lines and ADAA included, aliases under a steady tone, and on held-out kernels it chooses the factor far better than simple rules; and the rate at which alias falls is set by smoothness. Splitting a kernel across rates does not pay with real resamplers at Field's kernel sizes. On real music the steady-tone prediction does not carry over: the factor that real input needs depends on its loudest moment, and no predictor we tested is both safe and cheap. The useful product is one factor per kernel, chosen at compile time, with a warning when no factor will do.

9 Reproducibility

The study is in the Infinite repository under `docs/research/aliasing-aware-field/`. `scripts/kick-the-tires.sh` reruns phases 0 and 1 in under a minute and writes `data-rerun/phase1-predictions.csv`, which matched the committed file exactly on our machine. `scripts/run_all.sh` reruns everything; `scripts/cost_repeats.py` reruns the cost-model repeats and `scripts/make_audio.py` makes the listening clips. `ARTIFACT.md` links each claim to its command, data file and tolerance. Raw data is never edited; every figure and table here is generated from it.

Appendix A. Every H2 run, including those that did not count

Run 1 used Debug-build timings as the cost model and scored alias relative to each value's own fundamental: 0.8% saving. The relative score was meaningless for values with no fundamental (the absolute value of the input), and Debug timings over-charged the resamplers about tenfold.

Run 2, the verdict run, scored alias at the output, weighted by how sensitive the output is to each crossing value, and used the –O2 microbenchmark for costs: 17.4%.

Run 3 added a guard against cuts where band-limiting makes a one-signed value cross zero. No plan changed.

Each change was made because a measured plan disagreed with its prediction, never to move the saving. The 25% bar was fixed before run 1.

10 References

1. J. D. Parker, V. Zavalishin and E. Le Bivic (2016). Reducing the aliasing of nonlinear waveshaping using continuous-time convolution. *Proc. 19th Int. Conf. on Digital Audio Effects (DAFx-16), Brno.* dafx.de/paper-archive/details/vem_XXF5qBbfiWOH2RVVAA

2. S. Bilbao, F. Esqueda, J. D. Parker and V. Välimäki (2017). Antiderivative antialiasing for memoryless nonlinearities. *IEEE Signal Processing Letters, vol. 24, no. 7, pp. 1049–1053*. doi.org/10.1109/LSP.2017.2675541
3. M. Holters (2020). Antiderivative antialiasing for stateful systems. *Applied Sciences, vol. 10, no. 1, art. 20*. doi.org/10.3390/app10010020
4. O. Mikkonen and K. J. Werner (2025). Antiderivative antialiasing for recurrent neural networks. *Proc. 28th Int. Conf. on Digital Audio Effects (DAFx25), Ancona*. dafx.de/paper-archive/2025/DAFx25_paper_72.pdf
5. L. Gabrielli and S. Squartini (2026). PolyADAA: improving aliasing reduction in memoryless nonlinearities using Lagrange interpolation and polynomial approximation. *Proc. 29th Int. Conf. on Digital Audio Effects (DAFx26), Cambridge, MA*. www.dafx.de/paper-archive/2026/papers/DAFx26_paper_47.pdf
6. C. Spearman (1904). The proof and measurement of association between two things. *The American Journal of Psychology, vol. 15, no. 1, p. 72 ff.*. doi.org/10.2307/1412159
7. H. Thornburg. Sampling rate estimation techniques in the discretization of nonlinear systems. *CCRMA DSP seminar abstract, Stanford (Autumn 1998–1999 listing)*. ccrma.stanford.edu/~jos/mus423h/Sampling_Rate_Estimation_Techniques.html
8. R. Bristow-Johnson (2013). Re: Interpolation on WaveShaperNode?. *W3C public-audio mailing list, 13 May*. lists.w3.org/Archives/Public/public-audio/2013AprJun/0365.html
9. A. Herrou, F. de Dinechin, S. Letz, Y. Orlarey and A. Volkova (2024). Towards fixed-point formats determination for Faust programs. *Journées d'Informatique Musicale (JIM), Marseille; arXiv:2403.06527*. doi.org/10.48550/arXiv.2403.06527
10. Y. Orlarey and S. Letz (2024). The future of Faust: ondemand and co.. *International Faust Conference (IFC 2024), slides*. faust.grame.fr/community/ifc/2024/Ondemand.pdf

---

Provenance. Commit d370e296ff2d26a2d01d1afb283d9546c797de0b reproduces every number in this report. Machine: Apple M2, 8 cores, 8 GB RAM, macOS 27.2.1 arm64; Apple clang version 21.0.0 (clang-2100.4.1.1).

Raw data: https://infinitedevices.org/research/aliasing-aware-field/data/ · Source: https://github.com/n1m21n/Infinite/tree/d370e296/docs/research/aliasing-aware-field · Interactive version: https://infinitedevices.org/research/aliasing-aware-field.html