Why Spectrograms Look Smudged (And How to Fix Them)

A beginner-friendly guide to synchrosqueezing

What You Already Know

You know what an FFT does. You feed it a chunk of audio, it tells you which frequencies are present. You slide a window across your audio, stack up the FFTs, and you get a spectrogram.

But your spectrogram looks **smudged**. A pure tone doesn't look like a clean line — it looks like someone took a finger and smeared it vertically. You're wondering: *"Is this just how FFTs work, or am I doing something wrong?"*

You're not doing anything wrong. This is a known limitation of the standard approach. And there's a technique called **synchrosqueezing** that fixes it.

Why the Smudge Happens

Your FFT has bins at fixed frequencies. If your FFT size is 512 and your sample rate is 48 kHz, the bins are spaced 93.75 Hz apart:

Bin 0	Bin 1	Bin 2	Bin 3	Bin 4	Bin 5	Bin 6
0 Hz	94 Hz	188 Hz	281 Hz	375 Hz	469 Hz	563 Hz

Now imagine your signal has a pure 500 Hz tone. Where does it go?

It doesn't match any bin exactly. It falls **between** bin 5 (469 Hz) and bin 6 (563 Hz). The FFT doesn't know what to do with it, so it spreads the energy across both bins.

Result: Your clean 500 Hz tone now shows up as energy at both 469 Hz and 563 Hz. That's the smudge.

This is called *spectral leakage*, and it's a fundamental property of the FFT. It's not a bug — it's just how the math works.

The Hidden Information: Phase

Every FFT output value is a **complex number**. That means it has two parts:

- **Magnitude** — how strong is this frequency? (This is what you use to draw the spectrogram.)

- **Phase** — where are we in the wave cycle? (This is what you've been ignoring.)

Here's the thing: **the phase contains information about the true frequency.**

Think of It Like This: Engine Timing

Imagine you're tuning an engine. There's a timing mark painted on the crankshaft pulley, and a sensor that detects when the mark passes by.

You **expect** the engine to run at 3000 RPM. At that speed, the timing mark should pass the sensor at exactly the same point in each revolution.

But the engine is actually running at 3100 RPM — slightly faster than expected. What happens?

- Revolution 1: the mark passes right where you expect it
- Revolution 2: the mark arrives a little **early** (the engine completed the rotation faster than expected)
- Revolution 3: the mark arrives even earlier
- Revolution 4: the mark is now noticeably ahead of where it should be

The mark is **drifting forward** relative to your expectation. By measuring **how much it drifts per revolution**, you can calculate the **actual RPM**.

This is exactly what happens with FFT bins.

Each FFT bin is like expecting the engine to run at a specific RPM (the bin center frequency). The phase tells you where the "timing mark" is. If the phase drifts forward from one window to the next, the true frequency is higher than the bin center. If it drifts backward, the true frequency is lower.

Measure the drift rate, and you know the true frequency.

The Formula

The instantaneous frequency estimate is:

$$\hat{\omega}(a, b) = \mathrm{Re}\left[\frac{\partial_b V_f(a, b)}{i \cdot V_f(a, b)}\right] \tag{1}$$

Let's decode this:

- $V_f(a, b)$ = the complex FFT value for bin a at time window b
- $\partial_b V_f$ = how much that value changed from the previous time window
- i = the imaginary unit (a math trick that converts phase rotation into Hz)
- $\mathrm{Re}[\ldots]$ = take the real part of the result

In plain English:

"How fast is the phase changing from one window to the next?"
=
"What is the true frequency at this point?"

How to Actually Compute This

"Wait, what's that ∂_b symbol? That's a derivative!"

Good news: you don't need to compute derivatives numerically. There's a neat trick.

You compute **two FFTs** per window:

1. **Normal STFT** using your Hann window $h(t) \rightarrow$ gives you V_f
2. **Second STFT** using the **derivative of the Hann window** $h'(t) \rightarrow$ gives you V_{fd}

The Hann window is:

$$h(t) = 0.5 \times \big(1 - \cos(2\pi t)\big)$$

Its derivative is:

$$h'(t) = \pi \times \sin(2\pi t)$$

That's all. You already know how to window and FFT. Just use a different window function for the second FFT.

Then the formula becomes:

$$\hat{\omega} = \text{bin\_frequency} + \text{Re}\left[\frac{V_{fd}}{i \cdot V_f}\right] \tag{2}$$

What You Do With This Number

For every FFT bin at every time window, you now know the **true frequency**, not just the bin center.

So instead of drawing the energy at the bin center (469 Hz), you draw it at the true frequency (500 Hz).

This is synchrosqueezing: you're taking the smeared energy and "squeezing" it back to where it actually belongs.

The Algorithm

```
for each time window:
    # Step 1: Normal STFT with Hann window
    windowed = samples * hann_window
    V_f = FFT(windowed)

    # Step 2: STFT with derivative-of-Hann window
```

```
deriv_windowed = samples * hann_derivative
V_fd = FFT(deriv_windowed)

# Step 3: For each bin, compute true frequency
for each bin:
    # Complex division
    correction = Re(V_fd / (i * V_f))
    true_freq = bin_frequency + correction

    # Step 4: Put the energy at the true frequency
    target_bin = round(true_freq / freq_per_bin)
    output[target_bin] += magnitude(V_f)
```

What Changes Visually

Standard STFT	**Synchrosqueezed**
Energy spread across 2–4 bins	Energy concentrated in 1 bin
Tone looks like a fuzzy bar	Tone looks like a sharp line
Hard to distinguish close tones	Close tones are clearly separate

For an FSK signal (like your FSK4 at 50 baud):

- **Before:** each symbol looks like a blurry blob
- **After:** each symbol looks like a clean rectangle

The Trade-offs

- **Computation:** 2 FFTs per window instead of 1 (roughly 1.5–2× slower)
- **Memory:** need to store both V_f and V_{fd}
- **Quality:** significantly sharper time-frequency representation

For a 2-second audio file at FFT size 512, we're talking about 0.1 seconds vs. 0.15 seconds. Not a big deal for offline analysis.

Why Not Just Use a Bigger FFT?

You could increase your FFT size to get narrower bins. But:

- Bigger FFT = wider time window = worse **time** resolution
- You trade frequency smearing for time smearing
- Synchrosqueezing gives you sharp frequency **and** sharp time

It's not a replacement for choosing the right FFT size. It's a way to get the most out of whatever FFT size you choose.

Further Reading

If you want to go deeper:

- Thakur et al., *The Synchrosqueezing algorithm for time-varying spectral analysis* (2011) — the standard reference
- `librosa.reassigned_spectrogram` (Python) — a working implementation
- `scipy.signal.spectrogram` with phase-based reassignment — another approach

The smudge isn't your fault. But now you know how to fix it.