ChatGPT

Mathematical Approaches for Realistic Piano Tone Synthesis

Additive Synthesis with Harmonics and Detuned Partials

Additive synthesis models a piano tone as a sum of many sinusoidal components (partials), each with its own frequency, amplitude, and phase. Mathematically, the sound (x(t)) of a single piano note can be written as a sum of (N) damped sinusoids (harmonic partials):

[x(t) = \sum_{k=1}^{N} A_k(t)\,\sin!\big(2\pi f_k\,t + \phi_k\big),]

where (f_k) is the frequency of the k-th partial, (A_k(t)) is its amplitude envelope (decay over time), and (\phi_k) is its initial phase. In a basic model, one might start with harmonic frequencies $f_k \approx k \cdot f_1$ (integer multiples of the fundamental). However, **real piano strings are stiff**, causing *inharmonicity*: the partial frequencies are slightly shifted (not exact multiples of the fundamental) [1]. A stiff string's partial frequencies are often modeled by the formula:

[f_k = k \cdot f_1 \sqrt{\,1 + B\,k^2\,},]

where B is an inharmonicity coefficient that depends on string length, diameter, tension, etc. In a piano, B is small but nonzero, making higher partials "sharper" (higher in pitch) than exact multiples [1]. For very low strings, this effect can lead to partials slightly **flatter** than expected [1]. The result is that piano tunings are "stretched" – high notes are tuned a bit sharp and low notes a bit flat – to align their inharmonic partials with other notes [2]. To synthesize a realistic tone, one should incorporate this inharmonicity by adjusting f_k away from perfect integer ratios.

Each partial's amplitude must evolve over time to create the piano's characteristic **attack, sustain, and decay**. A simple choice is an exponential decay $A_k(t)=A_k(0)\,e^{-t/\tau_k}$ with a decay constant τ_k for each partial. In a real piano, higher harmonics tend to **die out faster** than lower ones, giving the sound a mellower quality as it sustains [3]. Thus, we can assign shorter τ_k (faster decay) to high-frequency partials and longer τ_k to low-frequency partials. This ensures the timbre evolves from bright (many harmonics at the attack) to warm (mostly fundamental in the tail). In practice, the envelopes need not be pure exponentials – some partials exhibit multi-stage decays (a rapid initial drop followed by a slower decay) due to energy transfer between string and soundboard. But even piecewise-exponential or ADSR-like envelopes per partial can work. The key is that **each partial's envelope is unique**, not a single global ADSR for the whole note [3]. In fact, analyzing real piano recordings shows that each harmonic partial follows its own amplitude trajectory [4], rather than one uniform decay. This complexity is crucial for realism.

Another critical aspect is **phase and interference**. If the piano note has multiple strings (most mid–high piano notes have 2 or 3 strings tuned to the same pitch), each string produces its own set of partials. In synthesis, one can simulate this by *doubling or tripling each partial* with slight frequency detuning between

them. For example, create two sinusoids for each nominal partial frequency, with one at f_k and another at f_k' a few cents (e.g. 0.5–1 Hz) off [5]. These close frequencies will interfere, producing slow **beats** (amplitude pulsations) with a period of a second or two, just as real unison strings do. This *thickens* the tone and adds a gentle "chorusing" effect that evolves over time [5]. To avoid objectionable **warbling**, the detuning should be very slight – skilled piano tuners keep unison strings within fractions of a hertz [5]. A 0.5 Hz detune produces a 2-second beat cycle, ensuring no rapid vibrato, just a subtle bloom in the sound [5]. It's important to set the initial phases ϕ_k of these detuned partials such that the note begins without cancellation artifacts. Often, setting all partials to start in phase (or randomizing phases slightly) is done to prevent any one moment where they accidentally cancel and then later constructively interfere (which could cause a perceived swell). In practice, many implementations simply randomize initial phases of partials to avoid a rigid, sterile sound, and this randomness can actually simulate the slight inconsistencies of a real piano strike. Humanizing the partial frequencies and phases (for example, adding tiny random fluctuations or drift) can further prevent the tone from sounding static or "too perfect" [6]. Indeed, **delicate random pitch fluctuations** over the note's duration help mimic the way real strings **"waver"** subtly as they vibrate [6]. We must use this judiciously – a small random frequency LFO or noise modulation can be applied to partials to **humanize** the tone without introducing an obvious vibrato [6].

By summing dozens of such partials, additive synthesis can produce a very realistic piano tone. This approach essentially constructs the waveform from the ground up. It requires careful tuning of many parameters (frequencies, initial amplitudes, decay rates, etc.), but it gives fine control over the spectrum at each moment in time. In practice, researchers have had success analyzing real piano samples to drive additive models. For example, one project extracted harmonic envelopes from the University of Iowa piano sample dataset and then used one oscillator per harmonic to replay those envelopes [4]. The result retained the natural attack and decay of each partial, sounding convincingly like a piano. In summary, an additive synthesis strategy for piano should include:

- **Inharmonic partial frequencies:** use a stiff-string formula or empirical data so partials are not exact multiples [1].
- **Multiple oscillators per note:** simulate two or three strings by duplicating partials with slight detuning (e.g. a few tenths of Hz difference) [5].
- **Individual amplitude envelopes:** assign each partial an appropriate decay (fast decay for high harmonics, slower for low) to mimic the changing timbre [3].
- **Transient alignment:** possibly start partials in phase at $t=0$ to create a strong attack "ping," then let interference create natural amplitude modulation later. Avoid phase settings that cause a dip then rise (preventing any **unnatural volume swell** after the initial attack).
- **Noise/transient component:** (see below) optionally add a short noise burst or non-harmonic component at attack to emulate percussive noises [7].

When done well, additive synthesis can capture the *stable harmonic structure* of a piano note and its gradual change over time [3]. Unlike a naive synth patch, a detailed additive model will have **complex, non-uniform harmonic decay** and pitch-dependent behavior, matching real acoustic pianos [3].

Modal Synthesis and Resonant Modes

Modal synthesis takes a physics-inspired view: it represents the piano string (and soundboard) as a set of resonant modes (natural vibration modes), each mode being a damped sinusoid. This is closely related to additive synthesis – in fact, mathematically it also ends up as a sum of decaying sinusoids – but the

parameters are derived from physical equations of vibration. Each **mode** is essentially a second-order system (an oscillator with frequency and damping). The modes can be excited by an input representing the hammer strike. The principle is: **excitation signal → bank of resonators → output sound** [8]. Each resonator has an impulse response that is an exponentially decaying sine wave, characterized by three parameters: its natural frequency f_k, a decay time (or loss factor) $1/R_k$, and a modal amplitude (related to how strongly that mode is excited) [9] [10]. In a real piano, the modes of the string correspond to the harmonics (with inharmonic spacing), and the modes of the soundboard contribute additional resonances (which can color the tone and prolong the sustain of certain frequencies).

In a modal synthesis model of a piano string, one would define a set of mode frequencies ${f_k}$ using a stiff-string formula or measured spectrum for that note (similar to the inharmonic partials above). Each mode is implemented as a damped oscillator (often as a biquad filter or a simple equation $y_k(t) = e^{-R_k t}\sin(2\pi f_k t)$ for the impulse response). To generate a tone, we apply an **excitation** that represents the hammer hit. A simple choice is a short pulse or an impulse with appropriate filtering. For example, striking a piano string can be modeled as applying an impulse *force* at time $t=0$ with a certain spectral character (the hammer felt hardness will make the excitation more or less bright). Modal synthesis allows us to feed that **excitation signal** into all the modes in parallel [8]. The output is the sum of each mode's response to the excitation. Because each mode is like a ringing filter, they will produce decaying sinusoidal outputs at their frequencies. The initial amplitude of each mode is determined by the excitation's spectrum: modes whose frequency content overlaps with the hammer impulse get strongly excited. In practice, one can derive the mode amplitudes from an assumed hammer force curve. For instance, a harder hammer strike can be modeled as a *sharper* impulse (with more high-frequency energy), thus exciting higher-frequency modes more. A softer strike could be a more rounded impulse (lacking extreme highs), resulting in weaker excitation of the highest modes. This gives a natural way to simulate **dynamic timbre variation** (forte notes are brighter) in a parametric way.

Modal synthesis is powerful because it ties the sound's evolution to physical parameters. **String coupling** and **soundboard resonance** can be incorporated by additional modes or by connecting the modes. For example, a piano note with three strings can be modeled with three sets of string modes that are coupled: when one string is struck, the others will vibrate lightly through bridge coupling. A simple way to approximate this is to include a *detuned copy* of loud modes as described earlier (this is effectively what happens if you simulate three strings – each string's modes will be at slightly different frequencies, yielding beating) [11] [12]. More advanced modal models explicitly simulate coupling by mixing energy between modes (e.g., two modes at nearly the same frequency exchange energy, causing beatings). For soundboard and cabinet resonance, one can add a set of low-frequency modes that ring longer (the **secondary resonators** in physical models [13]). These account for the **resonant decay**: after the string vibrations pass into the soundboard, certain frequencies (like around 100 Hz or other resonances of the wooden body) may linger even after the string partial has decayed. In practice, this can be approximated by adding a gentle **reverb or filter** that boosts the sustain of certain frequencies. (For example, one might include a very low amplitude, long-decay mode at the fundamental or at common formant frequencies of the piano body.)

From a mathematical perspective, modal synthesis often uses parallel second-order filter sections, each defined by:

[H_k(s) = \frac{\omega_k^2}{s^2 + 2 \zeta_k \omega_k s + \omega_k^2},]

where $\omega_k = 2\pi f_k$ and ζ_k is a damping factor related to the decay time ($\tau_k = 1/(\zeta_k \omega_k)$). In a digital implementation, each mode can be a digital biquad filter with a pair of complex conjugate poles. Feeding an impulse into this filter yields a decaying sinusoid at f_k with decay τ_k. The **modal model is agnostic to pitch** – you can transpose the whole note by scaling all f_k simultaneously [14] [15]. This is useful for generating the full 88 (or 128) notes: one could design a generic set of modes and then scale them for each MIDI note, though in practice piano notes vary in spectrum, so it's common to have note-specific parameters.

Compared to purely additive synthesis, modal synthesis is more **structured** – parameters like B (inharmonicity) or damping factors can often be estimated from physics or measurements, and changing something like hammer hardness can be done by altering the excitation signal rather than manually tweaking dozens of partial envelopes. Modern research synthesizers (e.g. Pianoteq) use sophisticated modal or **digital waveguide** models to achieve very realistic piano tones through physics-based synthesis. A digital waveguide is another physical modeling approach where the string is modeled as a delay loop with filters (accounting for wave propagation and dispersion). Waveguide and modal methods are closely related; in fact, the *modal solution* of the string wave equation yields the same set of modes that a waveguide would produce [16] [17]. The advantage of modal synthesis in a **pure offline generation** context is that it's naturally a sum of sinusoids (like additive), which is easy to implement in Python (just summing exponential sines), and it is highly parallel (each mode computed independently) [18]. The parameters can be tuned via analytical formulas or fitted to data.

In summary, a modal synthesis approach to piano involves defining each note's **modes (partials)** and their decay, then exciting them with a model hammer impulse. It inherently produces the correct **time evolution** (sharp attack as the impulse excites all modes, then modes decay at their own rates). By including sufficient modes (dozens per note) and possibly some coupling, modal synthesis can produce extremely realistic piano tones. For implementation, one can either pre-compute the sum of damped sinusoids (similar to additive) or simulate the system in the time-domain (e.g. send a digital impulse through a bank of resonant filter sections). Both yield the same result – the **exponentially decaying sinusoidal components** that make up a piano sound [9].

Simulating Attack, Coupling, and Resonant Decay

Regardless of additive or modal implementation, capturing the **attack, sustain, and decay** behavior of a piano note is essential. A real piano note's **attack** (~5–50 ms) is complex and crucial to realism. When the hammer strikes the string, it excites a broad range of frequencies. High-frequency components give the piano its percussive "bite" or brightness at the onset. To synthesize this, one can introduce a brief burst of noise or very short-lived partials. For example, adding a **transient noise component** at the attack can simulate the sound of the hammer hitting the string and the slight *buzz* of string excitation [7]. This could be white noise filtered to a high frequency range, decaying within a few tens of milliseconds. Alternatively, one can use a short broadband pulse (an impulse convolved with a shaping filter) as the excitation in a modal model. *Sound on Sound* notes on piano synthesis emphasize adding these "missing" pieces: the **high-frequency transient** from the hammer-string contact (depends on hammer hardness) and even the **low-frequency "thump"** of the key hitting the keybed [7]. Indeed, real pianos produce a faint thump in the low frequencies at the moment of attack (especially for staccato or forte playing) due to the key mechanism – this can be mimicked by adding a low-frequency damped thud sound (e.g. a sine burst around 30–60 Hz) with a very short decay [7]. While these mechanical noises are subtle, including them makes the synthesized tone feel more "acoustic" and less electronic.

During the **sustain** portion (the 1–5 seconds the note rings), the sound should gradually change rather than remaining static. In a piano, two main things are happening in this phase: (1) higher partials are fading out, shifting the timbre, and (2) if multiple strings were struck, they interfere and exchange energy (producing beats as discussed). Also, **sympathetic resonance** might occur (if the sustain pedal is down, other strings and the soundboard resonate along, but for a single note without pedal this effect is mostly limited to the struck strings and soundboard). To simulate string coupling without a full multi-string physical model, the earlier technique of detuned partials suffices: those partials will naturally produce interference beats over the sustain, a kind of slow **amplitude modulation** that adds richness. It's important that this interference not be so strong as to cause the volume to *increase* at any point – in a real piano the sound is always decaying in the long run. If one sets all partials to start in phase at the attack, the overall waveform will have a strong initial peak (lots of constructive interference giving a punchy attack), and afterwards, as phases drift, there may be slight ebbs and flows, but not a net growth. **Avoiding warbling artifacts** means choosing detune values that yield beat frequencies that are not overly fast or loud. Extremely close tuning (e.g. 0.1 Hz apart) yields a very slow beat (10-second cycle) which is almost imperceptible; a slightly larger detune (0.5–1 Hz) yields a gentle wah-wah every second or two [5]. One should also ensure the partial amplitude envelopes don't all peak at exactly the same time if using multi-stage envelopes, to prevent a sudden hump. In practice, if each partial simply decays from its initial value, the only way to get a "volume growth" is if two partials were initially canceling each other and later become in phase. Careful initial phase assignment (or using the physical model's natural phase from an impulse) will prevent any unnatural dip at the start that could lead to a later rise.

The **decay** or release of the note is when the sound finally dies out. On a real piano with the key held (no damper yet), the strings and soundboard eventually dissipate all energy. In synthesis, we ensure all partial envelopes (or modal dampings) are set so that by ~5 seconds, the amplitudes are near zero (for most notes). One consideration for **pitch-dependent decay**: low notes actually ring longer than high notes in a piano [3]. If we are generating the *full MIDI range*, a one-size 5-second envelope may not fit all. High notes (toward MIDI 108 and above) decay very fast – they might be almost silent after 1–2 seconds, whereas the lowest bass notes (MIDI ~21) can audibly ring well beyond 5 seconds (sometimes 10+ seconds). For practical sample generation, you might choose to truncate the low notes at 5 seconds (fading out any remaining sound toward the end to avoid an abrupt stop), and for high notes you might end up with a lot of silence if you allocate a full 5 seconds. The key is that each note has an appropriate decay rate. In additive terms, $ \tau_k$ and initial amplitudes would be set higher for low notes to ensure a longer sustain (and more energy in low-frequency modes which naturally decay slower due to lower damping in a large string). In modal terms, the damping factors ζ_k for low note modes would be smaller (longer ringing). This **variation of decay time with pitch** is an important realism factor often missed in simplistic synth patches [3].

Finally, **resonant decay**: as the note decays, the influence of the **soundboard** and environment might cause some frequencies to linger. A straightforward way to simulate this is to apply a low-level **reverberation or resonant filter** tuned to piano body resonances. For example, one could convolve the synthesized note with a short piano soundboard impulse response (which might add a subtle reverb tail). An even simpler approach is to include extra modal components that ring very quietly but long – e.g. a mode at ~110 Hz (the wood resonance) with a 5+ second decay, even for notes that don't have a strong 110 Hz partial initially. This can fill out the tail with a realistic ambience. The *differentiable piano model* by Berendes *et al.* (2023) uses a **soundboard filter** (obtained from impulse measurements) to shape the string sound, indicating that adding a linear filter after the string modes greatly improves realism [19] [20]. In offline synthesis, one could design a simple FIR filter that imparts a bit of that body resonance to each note. However, care must be

taken with an 8-bit output: adding reverb or long tails that fade into the noise floor might be lost or might accentuate quantization noise. With 8-bit PCM, the dynamic range is limited (~48 dB), so once the signal decays past a certain point, it will quantize to zero. **Dithering** can be used when quantizing to 8-bit to avoid abrupt cutoff in the tail, but given the application (ESP32 playback), it might be acceptable if very quiet sounds drop out.

Implementation in Python (Offline Generation)

Generating 128 piano tones offline in Python is quite feasible with these models. For additive synthesis, one can use NumPy to efficiently sum partials. A recommended approach is: for each MIDI note (e.g. 0–127 corresponding to frequencies from ~8.18 Hz up to ~12.5 kHz), determine the fundamental frequency using the MIDI tuning formula (e.g. A4 = 440 Hz, so $f_{\text{note}} = 440 \times 2^{(\text{note}-69)/12}$). Then decide on a set of partial frequencies. This could be done by a small loop up to a certain N (maybe 20–30 partials) or until frequencies exceed the Nyquist frequency (22,050 Hz at 44.1 kHz sample rate). For each partial k, compute $f_k = k \cdot f_1 \sqrt{1 + B k^2}$ – here B can be estimated or set per note. In lieu of a precise physical B, you might choose a heuristic: for higher notes, use a larger B (more inharmonicity) and for low notes, a smaller B. Some studies provide measured B values per piano note [2]. As an example, a mid-range note might have B on the order of 10^{-4}, while extreme bass strings might be 10^{-5} or less. Once f_k is set, assign an initial amplitude. You could use a **spectral envelope** approach – e.g. piano tones have a roughly bell-shaped spectral envelope (strong middle harmonics, slightly less fundamental for very hard strikes). Simpler: many implementations start with $A_k(0)$ proportional to $1/k$ or similar (fundamental strongest, decreasing for higher harmonics), then adjust to taste or data. For more accuracy, one can use reference data: the UIowa instrument sample set (free) contains piano notes that can be analyzed via FFT to get relative partial strengths [4]. In fact, the **AMY Synthesizer** project on GitHub did exactly this – analyzing real piano samples to get harmonic amplitudes over time [4]. Such data can be used to directly drive the envelopes in your code. If using those, you essentially load arrays of $A_{k}(t)$ values (perhaps at fixed time breakpoints) and interpolate them for your 5-second duration.

Without an existing dataset, you can still design a plausible envelope: e.g. $A_k(t) = A_k(0) \, e^{-t/\tau_k}$ with τ_k chosen such that by 5 s, $A_k(5\text{s})$ is maybe 1% of $A_k(0)$ for low k and far less (0.01%) for high k. This yields a faster drop-off for high harmonics. You might also implement a **two-stage decay**: for the first 0.5 s, use a shorter time constant (to simulate the rapid **initial decay** after the hammer strike), then switch to a longer time constant for the remaining sustain. This mimics the “double-slope” decay often observed (the sound drops quickly right after the attack, then decays slowly) – a phenomenon related to energy transfer from string to soundboard. In code, one could multiply two exponentials: for example $A_k(t) = A_k(0) e^{-t/T_{k1}} e^{-t^2/T_{k2}^2}$, where the second term in t^2 creates an initial steep drop and levels off. However, even a piecewise definition (if $t<t_1$ use $\tau_{k, \text{fast}}$, else use $\tau_{k,\text{slow}}$) can work.

To incorporate **detuned partials** for multiple strings, simply generate an extra sinusoid for some of the strongest partials at a frequency f_k' that is, say, $f_k \times (1 + \Delta)$ with Δ a tiny fraction (e.g. $\Delta = 0.00005$ for a 0.5 Hz detune at 440 Hz fundamental). You might do this only for the first few partials of each note – as higher partials might not need duplication (in real pianos, the *highest* notes actually have only one string, whereas notes in mid-range have 3 strings affecting the first several harmonics mainly) [5] [11]. A practical simplification is to just double all partials for notes that would have 2

strings, and triple for those with 3 strings, with slight detune differences. Randomizing the detune direction (some string slightly sharp, another slightly flat) around the target frequency can avoid bias.

When synthesizing, sum all partial contributions into a NumPy array representing the 5-second waveform at 44.1 kHz (so array length ~220,500 samples). It's wise to keep the amplitude moderate to avoid clipping when converting to 8-bit (since 8-bit PCM usually expects values 0–255 for unsigned or -128–127 for signed). You can normalize the waveform per note (e.g. peak at full scale 127/-128) or use a fixed scale across all notes (ensuring the loudest note fits in 8-bit). Keep in mind the ear's sensitivity and the limited dynamic range: you may want to use the full 8-bit range for each sample to maximize fidelity, but that means loud notes and soft notes both use full range individually. If that's acceptable (since these are individual samples to be triggered, not a continuous performance), it should be fine.

For **modal synthesis in Python**, one approach is to directly generate the modal responses as part of the additive sum (since a mode is just a sine with decay). For instance, you can define modes with frequencies f_k (including inharmonicity) and decay rates $R_k = 1/\tau_k$. If you have an idea of a hammer force function $h(t)$ (e.g. a short decay exponential or half-sine pulse of a few ms), you can analytically determine how each mode is excited (essentially the Fourier transform of $h(t)$ at frequency f_k gives the initial amplitude for mode k). A simpler way: just approximate the result by setting initial mode amplitude $A_k(0)$ in proportion to mode frequency (higher modes excited less for a soft hammer, more equally for a hard hammer). This is similar to setting initial partial amplitudes in additive. Thus, you may implement modal and additive in nearly identical code – the distinction is mostly conceptual (modal thinking ensures you include the right modes). An alternative implementation is using convolution: if you create an *impulse response* array for each note (as the sum of decaying sines over, say, 5 s), then convolving a short excitation signal with that IR will produce the note. But convolving a 5-s IR with a 5-ms excitation is effectively the same as just summing the modes excited by that impulse, so it's not adding efficiency here – it's easier to directly sum the sinusoids.

Preventing artifacts like looping or warbling is mostly about parameter choices. One should **avoid any sudden jumps or discontinuities** in the generated waveform or its slope, as those could create clicks. Since we are generating one long sample per note, that's straightforward: ensure the waveform smoothly decays to (almost) zero by the end of 5 seconds. If it hasn't fully died out (like a low note), you might apply a very gentle fade-out in the last 0.5 second to guarantee it ends at zero amplitude (preventing a DC offset or click when looping/concatenating samples on the ESP32). Warbling due to beats has been addressed by keeping detune minimal. If one hears a pronounced beating that sounds like a slow siren, the detune might actually be too *low* (very close frequencies can produce a noticeable tremolo). In that case, either reduce the amplitude of one of the beating components (so the modulation depth is shallow) or increase the frequency split so that the beat is faster than ~5 Hz, which the ear might integrate as a timbral difference (though too fast can start sounding like a chorus effect). Generally, piano unison beats are around 1–2 Hz or less, so aim for that regime [5].

Finally, after generating the waveforms in Python (using e.g. `numpy.sin` for the sine waves and multiplying by envelopes), you can use a library like `scipy.io.wavfile.write` or Python's built-in `wave` module to write each array as an 8-bit PCM WAV. Note that standard WAV expects 8-bit audio as unsigned bytes. That means you should offset the samples: e.g. if your waveform is in the range -1.0 to +1.0, convert to 0–255 by `(sample * 127.5 + 127.5)` and clip to [0,255] as integers. Alternatively, use 16-bit WAV and then post-process to 8-bit if needed.

Open-source resources can help in this process. The University of Iowa's Musical Instrument Samples (MIS) include single-note piano recordings which can serve as a reference or for analysis. The **AMY Synthesizer** project (open source on GitHub) provides a script that analyzed these piano samples to extract harmonic envelopes [4] – you might leverage their data or approach. There are also libraries like **CSound**, which has opcodes for modal synthesis and could be driven from Python (Csound has Python bindings). If interested in physical modeling, the **STK (Synthesis ToolKit)** has a piano model (using stretched tuning and Karplus-Strong with feedback). While STK is C++, one could examine its parameters for guidance.

In summary, the recommended approach for offline piano tone generation is to use **additive synthesis with inharmonic partials and tailored envelopes**, optionally incorporating aspects of **modal synthesis** (by choosing partial parameters informed by physics) and adding **transient/noise components** for attack. This gives fine control to achieve natural attack, realistic sustain (with gradually changing timbre), and smooth decay to silence. By following the guidelines on partial detuning and envelope shaping, you can avoid artifacts like warble or unnatural swells. The result will be a set of 128 wave files, each 5 seconds long, that capture the character of piano notes across the MIDI range, suitable for 8-bit playback on microcontroller hardware.

Sources:

- StackExchange discussion of piano synthesis: inharmonic (stretched) partial frequencies due to string stiffness [21], and importance of adding subtle detuning and duration to hear beats [5].
- Sound design notes from *Sound On Sound* and others: real pianos have non-uniform harmonic decay and pitch-dependent sustain times [3]; adding random pitch fluctuations can humanize the tone [6].
- *AMY Additive Piano* example: demonstrates using one oscillator per harmonic with an amplitude envelope derived from real piano recordings [4] (showing the efficacy of matching partial envelopes to achieve realism).
- Research on modal synthesis: describes modeling each string mode as an exponentially decaying sine (resonator) with frequency, amplitude, and decay time [9], which is well-suited to piano's percussive sound.
- StackExchange answer highlighting the need for transient noise (hammer noise, key thump) and multiple detuned strings for a convincing piano tone [22] [23].

---

[1] [2] [5] [7] [21] [22] [23] **piano - Why doesn't my synthesized note sound natural? - Music: Practice & Theory Stack Exchange**
https://music.stackexchange.com/questions/71531/why-doesnt-my-synthesized-note-sound-natural

[3] [4] **The AMY Additive Piano Voice**
https://shorepine.github.io/amy/piano.html

[6] **Synth Secrets Complete | PDF | Synthesizer | Harmonic**
https://www.scribd.com/doc/194386822/Synth-Secrets-Complete

[8] [9] [10] [11] [12] [14] [15] **Exploring Modal Synthesis | Nathan Ho**
https://nathan.ho.name/posts/exploring-modal-synthesis/

13 16 17 18 **ieee09.dvi**
https://home.mit.bme.hu/~bank/publist/taslp10.pdf

19 20 **AudioLabs - Towards Differentiable Piano Synthesis based on Physical Modeling**
https://audiolabs-erlangen.de/resources/MIR/2023-ISMIR-PianoSynth/