Below is a high-level sketch of an entirely new mathematical framework—let’s call it Acoustic Algebra—in which the fundamental primitives and operations are built not around time or space, but around sound relations: frequency, timbre, phase, and spectral shape. This is purely theoretical but designed to give a coherent structure you can develop further.

## **1. Primitive Objects: “Sonons”**

Rather than points in ℝ or vectors in ℝⁿ, our basic objects are Sonons—abstract “atoms” of sound characterized by:

* f: central frequency (Hz)
* Φ: phase offset (radians)
* T(ω): timbral spectral envelope function over angular frequency ω
* A: amplitude

A Sonon S is written

S = (f, \Phi, T(\omega), A)

We think of Sonons as basis functions—generalizing sinusoids to carry arbitrary envelope shapes.

## **2. Acoustic Vector Spaces**

Collections of Sonons span an Acoustic Vector Space 𝒜:

* Addition (superposition):  
    
   S\_1 \oplus S\_2 = (f\_, \Phi\_, T\_(ω), A\_)  
    
   where spectral envelopes and phases combine via a spectral mixing operator (e.g. weighted sum of T₁, T₂) and peak frequency f\_\* is a function of the envelopes’ centroids.
* Scalar multiplication (gain control):  
    
   \alpha \odot S = (f,\; \Phi,\; T(\omega),\; \alpha\,A)

This gives linear structure for mixing and amplitude scaling.

## **3. Acoustic Metric & Distance**

Define a sonic distance between two Sonons:

d(S\_1,\,S\_2) = \sqrt{ w\_f (f\_1–f\_2)^2 \;+\; w\_Φ \,\DeltaΦ^2 \;+\; w\_T \int (T\_1(ω)–T\_2(ω))^2\,dω}

* w\_f, w\_Φ, w\_T are weighting factors
* \DeltaΦ is minimal circular phase difference

This metric measures perceptual difference in pitch, phase, and timbre.

## **4. Operators & Algebraic Structures**

### **4.1 Spectral Convolution (⊛)**

Analogous to function convolution, we define spectral convolution of Sonons to model additive synthesis morphing:

\[

S\_1 \;\boxast\; S\_2 \;=\; \bigl(f\_1+f\_2,\;\Phi\_1+\Phi\_2,\;T\_1(ω)\;\*\;T\_2(ω),\;A\_1\,A\_2\bigr)

\]

where T\_1\*T\_2 is the usual convolution of their envelopes.

### **4.2 Phase Group**

Phase offsets form a compact group under addition modulo 2\pi:

Φ\_1 \oplus Φ\_2 = (Φ\_1 + Φ\_2)\bmod 2\pi

We can lift this to Sonons by holding other parameters constant.

## **5. Acoustic Calculus**

Define a differential operator over frequency:

\frac{∂}{∂\!f} \;S(f,\Phi,T,A) = \lim\_{Δf→0}\frac{S(f+Δf,…)−S(f,…)}{Δf}

This models the sensitivity of a Sonon to pitch bends or vibrato. One can similarly differentiate with respect to timbral parameters:

\frac{δ}{δ T(ω\_0)} S = \lim\_{\epsilon→0}\frac{S(f,\Phi, T+ε\,δ(ω−ω\_0), A) − S}{\epsilon}

capturing “spectral gradient” at a particular harmonic.

## **6. Sound-Algebraic Equations**

You can pose “Acoustic PDEs”:

\frac{∂S}{∂t} + v\_f\,\frac{∂S}{∂f} = 0

describing frequency drift (like glissando) propagating at rate v\_f. Or spectral diffusion:

\frac{∂T}{∂t} = D\,\frac{∂^2 T}{∂ω^2}

modeling timbre smoothing over time.

## **7. Harmonic Topology**

Since Sonons have frequency and phase, you can build a topological space of sound:

* Open sets correspond to ranges of frequencies and phase neighborhoods
* Continuous maps preserve perceptual continuity

This underpins “spectral continuity”: morphing one timbral shape to another along a path in this topology.

## **8. Examples & Applications**

1. Additive Synthesis Algebra: Modeling blending of two instruments as spectral convolution of their Sonon bases.
2. Filter Design: Represent an EQ as an operator on Sonons, shifting their envelopes T(ω).
3. Phase-Space Analysis: Plot trajectories of Sonons under modulation as curves in (f,Φ) space.

## **9. Next Steps**

* Formalize vector space axioms for 𝒜 (closure under ⊕, ⊛, scalar ⊗).
* Prove properties of the sonic metric (triangle inequality, symmetry).
* Develop a spectral differential geometry: define manifolds of timbre.
* Explore connections to wavelet and Fourier analyses as coordinate transforms.

This Acoustic Algebra framework replaces “time” as the primary axis with spectral and timbral relations, giving you a toolkit to reason about sound transformations, morphing, and synthesis in purely algebraic and geometric terms.

💥 Let’s go. This’ll be the SononKit — a lightweight DSP + math framework where sounds are treated as spectral objects, not waveforms.

GOAL: Create a Node.js + TypeScript foundation that defines Sonons, operations over them, and prepares us for future real-time audio tools.

### **🧱 PART 1: CORE MATH –**

### **sonon.ts**

We’ll start with:

* Sonon object
* Distance function
* Convolution
* Differentiation (spectral gradient)

// sonon.ts

export type SpectralEnvelope = (omega: number) => number;

export interface Sonon {

freq: number; // Hz

phase: number; // radians

amp: number; // linear

envelope: SpectralEnvelope;

}

export function createSonon(

freq: number,

phase: number,

amp: number,

envelope: SpectralEnvelope

): Sonon {

return { freq, phase, amp, envelope };

}

export function sononDistance(s1: Sonon, s2: Sonon, weights = { f: 1, p: 1, t: 1 }): number {

const deltaF = s1.freq - s2.freq;

const deltaP = Math.atan2(Math.sin(s1.phase - s2.phase), Math.cos(s1.phase - s2.phase));

const deltaT = integrateSpectralDiff(s1.envelope, s2.envelope);

return Math.sqrt(weights.f \* deltaF \*\* 2 + weights.p \* deltaP \*\* 2 + weights.t \* deltaT);

}

function integrateSpectralDiff(t1: SpectralEnvelope, t2: SpectralEnvelope): number {

const steps = 128;

const ωmin = 0;

const ωmax = 2 \* Math.PI \* 20000;

const dω = (ωmax - ωmin) / steps;

let sum = 0;

for (let i = 0; i <= steps; i++) {

const ω = ωmin + i \* dω;

const diff = t1(ω) - t2(ω);

sum += diff \* diff;

}

return sum \* dω;

}

export function convolveSonons(s1: Sonon, s2: Sonon): Sonon {

const freq = s1.freq + s2.freq;

const phase = (s1.phase + s2.phase) % (2 \* Math.PI);

const amp = s1.amp \* s2.amp;

const envelope = (ω: number) => s1.envelope(ω) \* s2.envelope(ω);

return { freq, phase, amp, envelope };

}

### **🧪 PART 2: TEST BENCH –**

### **test.ts**

Let’s give it a test run with basic bell curves as envelopes.

// test.ts

import { createSonon, convolveSonons, sononDistance } from "./sonon";

const gaussian = (center: number, width: number) =>

(ω: number) => Math.exp(-((ω - center) \*\* 2) / (2 \* width \*\* 2));

const s1 = createSonon(440, 0, 1.0, gaussian(2 \* Math.PI \* 440, 2000));

const s2 = createSonon(880, Math.PI / 4, 0.5, gaussian(2 \* Math.PI \* 880, 2000));

const s3 = convolveSonons(s1, s2);

const dist = sononDistance(s1, s2);

console.log("Convolved Sonon freq:", s3.freq);

console.log("Distance between s1 and s2:", dist.toFixed(4));

🧠 Next Moves:

* Export Sonons as JSON
* Visualize a Sonon spectrum in D3.js or p5.js
* Define scalar ops and calculus ops (df/df, δ/δT)
* Eventually: hook into WebAudio or DustEngine

Want this bundled into a GitHub folder starter pack? Or should we jump straight to visualizing some Sonon curves?

# **SononKit — Node.js + TypeScript Starter Pack**

A lightweight DSP & math framework treating sounds as spectral objects (Sonons).

## **Project Structure**

SononKit/

├── package.json

├── tsconfig.json

├── README.md

└── src/

├── sonon.ts # Core Sonon definitions and operations

└── test.ts # Test bench for Sonon operations

## **1. package.json**

{

"name": "sononkit",

"version": "0.1.0",

"description": "Acoustic Algebra library for Sonon objects",

"main": "dist/test.js",

"scripts": {

"build": "tsc",

"test": "node dist/test.js"

},

"keywords": ["DSP", "spectral", "Sonon", "audio"],

"author": "",

"license": "MIT",

"devDependencies": {

"typescript": "^4.9.5"

}

}

## **2. tsconfig.json**

{

"compilerOptions": {

"target": "ES2020",

"module": "commonjs",

"outDir": "dist",

"strict": true,

"esModuleInterop": true,

"forceConsistentCasingInFileNames": true

},

"include": ["src"]

}

## **3. src/sonon.ts**

export type SpectralEnvelope = (omega: number) => number;

export interface Sonon {

freq: number; // Hz

phase: number; // radians

amp: number; // linear amplitude

envelope: SpectralEnvelope;

}

export function createSonon(

freq: number,

phase: number,

amp: number,

envelope: SpectralEnvelope

): Sonon {

return { freq, phase, amp, envelope };

}

export function sononDistance(

s1: Sonon,

s2: Sonon,

weights = { f: 1, p: 1, t: 1 }

): number {

const deltaF = s1.freq - s2.freq;

const deltaP = Math.atan2(

Math.sin(s1.phase - s2.phase),

Math.cos(s1.phase - s2.phase)

);

const deltaT = integrateSpectralDiff(s1.envelope, s2.envelope);

return Math.sqrt(

weights.f \* deltaF \*\* 2 +

weights.p \* deltaP \*\* 2 +

weights.t \* deltaT

);

}

function integrateSpectralDiff(

t1: SpectralEnvelope,

t2: SpectralEnvelope

): number {

const steps = 128;

const ωmin = 0;

const ωmax = 2 \* Math.PI \* 20000;

const dω = (ωmax - ωmin) / steps;

let sum = 0;

for (let i = 0; i <= steps; i++) {

const ω = ωmin + i \* dω;

const diff = t1(ω) - t2(ω);

sum += diff \* diff;

}

return sum \* dω;

}

export function convolveSonons(s1: Sonon, s2: Sonon): Sonon {

const freq = s1.freq + s2.freq;

const phase = (s1.phase + s2.phase) % (2 \* Math.PI);

const amp = s1.amp \* s2.amp;

const envelope = (ω: number) => s1.envelope(ω) \* s2.envelope(ω);

return { freq, phase, amp, envelope };

}

## **4. src/test.ts**

import { createSonon, convolveSonons, sononDistance } from './sonon';

// Gaussian bell curve envelope

const gaussian = (center: number, width: number) =>

(ω: number) => Math.exp(-((ω - center) \*\* 2) / (2 \* width \*\* 2));

// Create two Sonons

const s1 = createSonon(

440,

0,

1.0,

gaussian(2 \* Math.PI \* 440, 2000)

);

const s2 = createSonon(

880,

Math.PI / 4,

0.5,

gaussian(2 \* Math.PI \* 880, 2000)

);

// Convolution and distance

const s3 = convolveSonons(s1, s2);

const dist = sononDistance(s1, s2);

console.log('Convolved Sonon freq:', s3.freq);

console.log('Distance between s1 and s2:', dist.toFixed(4));

## **5. Getting Started**

1. Install dependencies:

npm install

2. Build the TypeScript:

npm run build

2. Run Tests:

npm test

This starter pack lays the foundation for SononKit. Next steps could include:

* JSON serialization of Sonons
* Visualization via D3.js or p5.js
* Advanced calculus ops (frequency derivatives, spectral gradients)
* Real-time WebAudio or DustEngine integration

Enjoy building with Acoustic Algebra!

Perfect. We’ll go modular and clean—each step functional, each part exportable.

## **🧱 Step 3: Project Structure**

We’ll evolve this like a micro-DSP framework:

sononkit/

├── src/

│ ├── core/ # math + definitions

│ │ └── sonon.ts

│ ├── viz/ # D3 or canvas-based visual tools

│ │ └── spectrumPlot.ts

│ ├── test/ # test harnesses

│ │ └── testSonons.ts

│ ├── index.ts # entry point

├── package.json

├── tsconfig.json

└── README.md

## **🎨 Step 4:**

## **spectrumPlot.ts**

## **– Visualizing a Sonon**

This is a minimal D3 sketch that plots a Sonon’s envelope.

// spectrumPlot.ts

import \* as d3 from "d3";

import { Sonon } from "../core/sonon";

export function plotSononEnvelope(svgId: string, sonon: Sonon) {

const svg = d3.select(`#${svgId}`);

svg.selectAll("\*").remove();

const width = +svg.attr("width");

const height = +svg.attr("height");

const freqs = d3.range(20, 20000, 50);

const envelope = freqs.map(f => ({

freq: f,

mag: sonon.envelope(2 \* Math.PI \* f) \* sonon.amp,

}));

const x = d3.scaleLog().domain([20, 20000]).range([0, width]);

const y = d3.scaleLinear().domain([0, d3.max(envelope, d => d.mag)!]).range([height, 0]);

const line = d3.line<{ freq: number; mag: number }>()

.x(d => x(d.freq))

.y(d => y(d.mag));

svg.append("path")

.datum(envelope)

.attr("fill", "none")

.attr("stroke", "cyan")

.attr("stroke-width", 2)

.attr("d", line);

}

## **📦 Step 5:**

## **package.json**

## **&**

## **tsconfig.json**

Setting up basic build + typing:

package.json

{

"name": "sononkit",

"version": "0.1.0",

"main": "dist/index.js",

"type": "module",

"scripts": {

"start": "ts-node src/test/testSonons.ts",

"build": "tsc"

},

"dependencies": {

"d3": "^7.8.5"

},

"devDependencies": {

"ts-node": "^10.9.1",

"typescript": "^5.3.3"

}

}

tsconfig.json

{

"compilerOptions": {

"target": "ES2020",

"module": "ESNext",

"moduleResolution": "Node",

"outDir": "dist",

"rootDir": "src",

"esModuleInterop": true,

"strict": true

}

}

## **⚙️ Step 6: Entry & Launch**

index.ts (initial placeholder):

export \* from "./core/sonon";

export \* from "./viz/spectrumPlot";

testSonons.ts (launch test app + plot envelope):

import { createSonon } from "../core/sonon";

import { plotSononEnvelope } from "../viz/spectrumPlot";

const gaussian = (center: number, width: number) =>

(ω: number) => Math.exp(-((ω - center) \*\* 2) / (2 \* width \*\* 2));

const sonon = createSonon(440, 0, 1, gaussian(2 \* Math.PI \* 440, 1500));

// For DOM use only:

window.onload = () => plotSononEnvelope("sononViz", sonon);

Include an HTML scaffold to visualize. Want me to drop the index.html + build instructions next?

💥 Let’s go. This’ll be the SononKit — a lightweight DSP + math framework where sounds are treated as spectral objects, not waveforms.

GOAL: Create a Node.js + TypeScript foundation that defines Sonons, operations over them, and prepares us for future real-time audio tools.

### **🧱 PART 1: CORE MATH –**

### **sonon.ts**

We’ll start with:

* Sonon object
* Distance function
* Convolution
* Differentiation (spectral gradient)

// sonon.ts

export type SpectralEnvelope = (omega: number) => number;

export interface Sonon {

freq: number; // Hz

phase: number; // radians

amp: number; // linear

envelope: SpectralEnvelope;

}

export function createSonon(

freq: number,

phase: number,

amp: number,

envelope: SpectralEnvelope

): Sonon {

return { freq, phase, amp, envelope };

}

export function sononDistance(s1: Sonon, s2: Sonon, weights = { f: 1, p: 1, t: 1 }): number {

const deltaF = s1.freq - s2.freq;

const deltaP = Math.atan2(Math.sin(s1.phase - s2.phase), Math.cos(s1.phase - s2.phase));

const deltaT = integrateSpectralDiff(s1.envelope, s2.envelope);

return Math.sqrt(weights.f \* deltaF \*\* 2 + weights.p \* deltaP \*\* 2 + weights.t \* deltaT);

}

function integrateSpectralDiff(t1: SpectralEnvelope, t2: SpectralEnvelope): number {

const steps = 128;

const ωmin = 0;

const ωmax = 2 \* Math.PI \* 20000;

const dω = (ωmax - ωmin) / steps;

let sum = 0;

for (let i = 0; i <= steps; i++) {

const ω = ωmin + i \* dω;

const diff = t1(ω) - t2(ω);

sum += diff \* diff;

}

return sum \* dω;

}

export function convolveSonons(s1: Sonon, s2: Sonon): Sonon {

const freq = s1.freq + s2.freq;

const phase = (s1.phase + s2.phase) % (2 \* Math.PI);

const amp = s1.amp \* s2.amp;

const envelope = (ω: number) => s1.envelope(ω) \* s2.envelope(ω);

return { freq, phase, amp, envelope };

}

### **🧪 PART 2: TEST BENCH –**

### **test.ts**

Let’s give it a test run with basic bell curves as envelopes.

// test.ts

import { createSonon, convolveSonons, sononDistance } from "./sonon";

const gaussian = (center: number, width: number) =>

(ω: number) => Math.exp(-((ω - center) \*\* 2) / (2 \* width \*\* 2));

const s1 = createSonon(440, 0, 1.0, gaussian(2 \* Math.PI \* 440, 2000));

const s2 = createSonon(880, Math.PI / 4, 0.5, gaussian(2 \* Math.PI \* 880, 2000));

const s3 = convolveSonons(s1, s2);

const dist = sononDistance(s1, s2);

console.log("Convolved Sonon freq:", s3.freq);

console.log("Distance between s1 and s2:", dist.toFixed(4));

🧠 Next Moves:

* Export Sonons as JSON
* Visualize a Sonon spectrum in D3.js or p5.js
* Define scalar ops and calculus ops (df/df, δ/δT)
* Eventually: hook into WebAudio or DustEngine

Want this bundled into a GitHub folder starter pack? Or should we jump straight to visualizing some Sonon curves?