Technical Specification: The "Trinity" Refactor

Project: Chronomorphic Polytopal Engine (CPE)
Target Architecture: Decomposed 24-Cell (Three Interlocking 16-Cells)
Date: 2026-01-11

---

1. Architectural Overview

The **Trinity Refactor** transitions the CPE from a monolithic geometric model (treating the 24-cell as a single surface) to a **Multi-State Superposition Model**. This allows the system to distinguish between three harmonic "universes" or axes that coexist within the same 4D space.

The Three Axes (Sub-Polytopes)

Based on the research findings, we define three orthogonal 16-cell sub-structures:

Axis	Label	Keys (Musical Function)	Vertex Indices
Alpha	alpha	**Natural** (C, G, D, A Maj & Rel. Min)	0, 1, 2, 3, 20, 21, 22, 23
Beta	beta	**Sharp** (E, B, F#, Db Maj & Rel. Min)	4, 5, 6, 7, 16, 17, 18, 19
Gamma	gamma	**Flat** (Ab, Eb, Bb, F Maj & Rel. Min)	8, 9, 10, 11, 12, 13, 14, 15

Core Logic Shifts

1. **State Definition:** currentVertex (int) $\rightarrow$ axisState (Vector3 representing α, β, γ weights).
2. **Navigation:** move(from, to) $\rightarrow$ navigate(from, to) which now returns a TransitionType (Standard Move vs. Phase Shift).
3. **Tension:** Now includes InterAxisTension (dissonance caused by conflicting axis activation).

2. Implementation Roadmap

Phase 1: Foundation (Type Definitions & Topology)

Objective: Codify the decomposition map and helper classes without breaking existing logic.

2.1 New File: lib/topology/PolytopeDecomposition.ts

This is the source of truth for the Trinity architecture. It maps every vertex of the 24-cell to its corresponding Axis.

TypeScript

```typescript
/**
 * PolytopeDecomposition.ts
 * Defines the structural decomposition of the 24-Cell into three disjoint 16-Cells.
 */

// 1. Strict Type Definitions
export type AxisLabel = 'alpha' | 'beta' | 'gamma';

export interface AxisDefinition {
  label: AxisLabel;
  description: string;
  indices: number; // The vertex IDs belonging to this axis
  color: string;    // For debug/visualization
}

// 2. The Decomposition Map (Hardcoded Truth)
export const TRINITY_AXES: Record<AxisLabel, AxisDefinition> = {
  alpha: {
    label: 'alpha',
    description: 'Natural Axis (C, G, D, A)',
    indices: ,
    color: '#FF0000' // Red
  },
  beta: {
    label: 'beta',
    description: 'Sharp Axis (E, B, F#, Db)',
    indices: [1, 2, 3, 4, 5, 6, 7, 8],
```

```
        color: '#00FF00' // Green
    },
    gamma: {
        label: 'gamma',
        description: 'Flat Axis (Ab, Eb, Bb, F)',
        indices: [9, 10, 11, 12, 13, 14, 15, 16],
        color: '#0000FF' // Blue
    }
};

// 3. Reverse Lookup Map (Vertex ID -> AxisLabel)
// Generated once for O(1) lookups
export const VERTEX_TO_AXIS: Record<number, AxisLabel> = (() => {
    const map: Record<number, AxisLabel> = {};

    // Helper to fill map
    const fill = (axis: AxisLabel) => {
        TRINITY_AXES[axis].indices.forEach(idx => map[idx] = axis);
    };

    fill('alpha');
    fill('beta');
    fill('gamma');

    // Validation: Ensure all 24 vertices are mapped
    if (Object.keys(map).length!== 24) {
        throw new Error("Critical Topology Error: Decomposition map does not cover all 24 vertices.");
    }

    return map;
})();

/**
 * Helper Class for Navigation Logic
 */
export class DecompositionRegistry {

    /**
     * Returns the axis label for a given vertex index.
     */
    static getAxis(vertexIndex: number): AxisLabel {
        const axis = VERTEX_TO_AXIS[vertexIndex];
        if (!axis) throw new Error(`Invalid Vertex Index: ${vertexIndex}`);
```

```
    return axis;
  }

  /**
   * Determines if a move between two vertices involves a "Phase Shift"
   * (Jumping from one 16-cell universe to another).
   */
  static getPhaseShift(fromVertex: number, toVertex: number): boolean {
    const axisA = this.getAxis(fromVertex);
    const axisB = this.getAxis(toVertex);
    return axisA!== axisB;
  }

  /**
   * Returns the "Spin" (distance) between two axes.
   * 0 = Same Axis, 1 = Adjacent Axis (Phase Shift)
   */
  static getAxisDistance(fromVertex: number, toVertex: number): number {
    return this.getPhaseShift(fromVertex, toVertex)? 1 : 0;
  }
}
```

2.2 Integration Test: scripts/test-decomposition.ts

This script verifies that the logic correctly identifies standard moves vs. phase shifts.

TypeScript

```typescript
import { DecompositionRegistry } from '../lib/topology/PolytopeDecomposition';

function runTest() {
  console.log("=== Testing Trinity Decomposition Logic ===");

  // Case 1: Simple Dominant resolution (C Maj -> G Maj)
  // C Maj is index 0 (Alpha), G Maj is index 1 (Alpha)
  const cMaj = 0;
  const gMaj = 1;
  const isPhaseShift1 = DecompositionRegistry.getPhaseShift(cMaj, gMaj);
  console.log(`C -> G (Alpha -> Alpha): Phase Shift? ${isPhaseShift1} (Expected: false)`);

  // Case 2: Modulation to E Major (C Maj -> E Maj)
```

```
  // C Maj is index 0 (Alpha), E Maj is index 4 (Beta)
  const eMaj = 4;
  const isPhaseShift2 = DecompositionRegistry.getPhaseShift(cMaj, eMaj);
  console.log(`C -> E (Alpha -> Beta): Phase Shift? ${isPhaseShift2} (Expected: true)`);

  // Case 3: Trone Substitution (C Maj -> F# Maj)
  // C Maj is index 0 (Alpha), F# Maj is index 6 (Beta)
  const fSharpMaj = 6;
  const isPhaseShift3 = DecompositionRegistry.getPhaseShift(cMaj, fSharpMaj);
  console.log(`C -> F# (Alpha -> Beta): Phase Shift? ${isPhaseShift3} (Expected: true)`);

  if (!isPhaseShift1 && isPhaseShift2 && isPhaseShift3) {
    console.log("SUCCESS: Topology logic valid.");
  } else {
    console.error("FAILURE: Topology logic incorrect.");
  }
}

runTest();
```

Phase 2: Axis-Aware State Engine (Core Logic)

Objective: Update the main engine to track the "Quantum State" of the music.

Refactoring SonicGeometryEngine.ts

The engine currently tracks this.currentVertex. We will change this to track this.axisState.

TypeScript

```typescript
// Proposed interface update
interface AxisState {
  activeAxis: AxisLabel;     // The dominant universe
  previousAxis: AxisLabel;   // For history tracking
  confidence: number;        // 0.0 - 1.0 (How firmly are we in this key?)
  tension: number;           // Inter-axis tension
}

class SonicGeometryEngine {
```

```
  //... existing properties
  private axisState: AxisState;

  constructor() {
    this.axisState = {
      activeAxis: 'alpha', // Start in C Major / Natural
      previousAxis: 'alpha',
      confidence: 1.0,
      tension: 0
    };
  }

  /**
   * Updates the geometric state based on a new chord/key input.
   */
  updateState(newVertexIndex: number) {
    const newAxis = DecompositionRegistry.getAxis(newVertexIndex);
    const isPhaseShift = newAxis!== this.axisState.activeAxis;

    if (isPhaseShift) {
      console.log(` Moving from ${this.axisState.activeAxis} to ${newAxis}`);
      // Logic to trigger visual effects or tension spike
      this.axisState.tension = 1.0; // Max tension on shift
    } else {
      // Decay tension if we stay in the same axis
      this.axisState.tension *= 0.9;
    }

    this.axisState.previousAxis = this.axisState.activeAxis;
    this.axisState.activeAxis = newAxis;

    //... existing update logic
  }
}
```

Phase 3: The "Tension-Between-Worlds" Metric

Objective: Quantify the 'rub' of polytonality.

We introduce calculateInterAxisTension(chordIndices: number).

1. **Monolithic Chord:** If all notes in a chord belong to **Alpha**, tension is 0.

2. **Bitonal Chord:** If notes are split between **Alpha** and **Beta** (e.g., C Major + E Major polychord), tension is high (0.8).
3. **Diminished Chord:** A fully diminished 7th has one note in Alpha, one in Beta, one in Gamma, and one repeating. This is "Omni-axial" tension.

TypeScript

```typescript
function calculateInterAxisTension(notes: number): number {
  const axesPresent = new Set<AxisLabel>();

  notes.forEach(n => {
    axesPresent.add(DecompositionRegistry.getAxis(n));
  });

  // 1 axis = 0 tension
  // 2 axes = 0.5 tension
  // 3 axes = 1.0 tension (Maximal ambiguity)
  return (axesPresent.size - 1) / 2.0;
}
```

Phase 4: Visualization (The Heads-Up Display)

Objective: Visual feedback for the Phase Shifts.

Update HypercubeRenderer to use the TRINITY_AXES colors.

1. **Vertex Coloring:** Instead of a single color, vertices are colored Red, Green, or Blue based on VERTEX_TO_AXIS.
2. **Edge Coloring:**
   - Alpha-Alpha edges: **Red** (Stable)
   - Beta-Beta edges: **Green** (Stable)
   - Alpha-Beta edges: **Yellow** (Tension/Bridge)

This allows the user to *see* the modulation happen. When the music modulates from C to E, the visual activity shifts from the Red sub-lattice to the Green sub-lattice.

Execution Instructions for the Agent

1. **Copy** the code from **Phase 1 (2.1)** into lib/topology/PolytopeDecomposition.ts.
2. **Copy** the test script into scripts/test-decomposition.ts.
3. **Run** the test script to verify the logic.
4. Once verified, proceed to **Phase 2** refactoring of the Engine class.