Engineering the Celestial Sphere: A Comprehensive Architecture for Computational Astrology, Geospatial Optimization, and Paran Analysis

Executive Summary

The digitalization of astrology has largely remained confined to two-dimensional representations of the heavens, predominantly focusing on the Ecliptic longitude of planetary bodies. This standard model, while sufficient for basic character analysis, neglects the physical, three-dimensional reality of celestial mechanics—specifically the vertical dimension of **Declination** and the latitude-dependent phenomena known as **Parans** (Paranatellonta). The exclusion of these metrics results in a significant loss of data regarding planetary intensity, visibility, and terrestrial correspondence.

This report presents a definitive architectural blueprint for the development of a high-precision, TypeScript-based computational engine designed to bridge this gap. The proposed system acts as an "Astro-Geospatial" optimization platform, capable of calculating planetary positions with millisecond precision using the **Swiss Ephemeris**, analyzing their vertical relationships (Parallels/Contra-Parallels), and mapping these forces to terrestrial coordinates to identify locations of maximum energetic resonance.

Moving beyond static chart generation, this architecture mandates the construction of iterative numerical solvers for identifying Paran latitudes, robust scoring engines based on ancient Essential Dignity systems (Ptolemaic and Egyptian), and a high-performance visualization layer utilizing **WebGL** and **Three.js**. By integrating the rigor of spherical astronomy with the flexibility of modern full-stack development, this system will empower the "Astrologer-Engineer" to move from symbolic interpretation to geometric optimization. The following analysis details the theoretical physics, mathematical derivations, algorithmic logic, and software infrastructure required to execute this vision.

Part I: Theoretical Physics and Astrological Geometry

To engineer a system capable of calculating "vertical" astrology, one must first deconstruct the geometric axioms that govern the local cosmos. The standard astrological wheel is a simplified projection; the reality is a complex interaction of spherical coordinate systems.

1.1 The Vertical Dimension: Equatorial vs. Ecliptic Coordinates

The foundational error in many astrological applications is the conflation of the Ecliptic (the

path of the Sun) with the Celestial Equator (the projection of Earth's rotation). While the Ecliptic is the highway of the planets, the Equator is the reference plane for terrestrial correspondence. The misalignment between these two planes, known as the **Obliquity of the Ecliptic (ϵ)**, is approximately 23.44° and is the primary driver of declination mechanics.[1]

Declination (δ) acts as the celestial equivalent of terrestrial latitude. It measures the angular distance of a body North (+) or South (-) of the Celestial Equator. This coordinate is not merely a secondary data point; it is the definitive metric for **intensity** and **visibility**.

- **0° Declination:** Corresponds to the Celestial Equator. Planets here rise due East and set due West everywhere on Earth. This occurs at the Equinox points (0° Aries/Libra).
- **Maximum Declination ($\pm 23.44^\circ$):** Corresponds to the Tropics of Cancer and Capricorn. Planets here have the longest or shortest Diurnal Arcs (duration of visibility) depending on the hemisphere.

The computational implication is that a planet's longitudinal position (e.g., 15° Leo) is insufficient for geospatial optimization. Two planets can be conjunct in longitude but separated by 5° in declination, rendering them physically distant in the sky. Conversely, planets in **Parallel of Declination ($||$)** occupy the same vertical degree, creating a functional conjunction of "power" where they share the same diurnal arc and rise/set geometry.[1]

1.2 The "Out of Bounds" (OOB) Phenomenon

The Sun's declination is bounded by the obliquity of the ecliptic ($\approx 23^\circ 27'$). However, due to their own orbital inclinations relative to the Ecliptic, the Moon and planets can exceed this limit. This is the **Out of Bounds (OOB)** phenomenon.

- **Theoretical Impact:** OOB planets operate outside the "jurisdiction" of the solar hierarchy. They manifest as extreme, non-conforming, or genius archetypes.[1]
- **Algorithmic Requirement:** The engine must dynamically calculate the maximum solar declination for the exact epoch (accounting for nutation) to accurately flag OOB planets.

  A hardcoded value of 23.44° is insufficient for high-precision historical or future calculations.

1.3 The Geometry of Parans (Paranatellonta)

While a Zenith line represents a single planet passing directly overhead (where Terrestrial Latitude = Celestial Declination), a **Paran** represents a synchronized relationship between two planets relative to the local horizon.

- **Definition:** A Paran occurs when Planet A is on an angle (Rising, Setting, Culminating,

Anti-Culminating) *at the exact same moment* that Planet B is on an angle.
- **Latitude Dependence:** This synchronization is a function of terrestrial latitude. For any pair of planets, there exists a specific latitude (or set of latitudes) where their diurnal circles intersect the horizon/meridian simultaneously.
- **The "Orb" of Influence:** Unlike longitudinal aspects which have wide orbs, Parans are latitude-specific. Research indicates the effect is strongest within $\pm 1^\circ$ of latitude (approx. 70 miles).[2]

This defines the core engineering challenge: The system must solve for the latitude ϕ that satisfies the temporal equality of two independent planetary events.

1.4 Essential Dignities and Weighting

To differentiate "good" locations from "challenging" ones, the system relies on **Essential Dignities**, a scoring system assessing a planet's strength based on its zodiacal position.[3]

- **Domicile (+5):** Planet in its own sign.
- **Exaltation (+4):** Planet in its sign of highest expression.
- **Triplicity (+3):** Elemental affinity, dependent on the **Sect** of the chart (Day vs. Night).
- **Terms/Bounds (+2):** Irregular subdivisions of signs. The engine must support multiple systems (Ptolemaic, Egyptian) as these boundaries define the "rules of engagement" for the planet.[4]
- **Face (+1):** Decan-based rulership.

This scoring layer is critical for the "Recommendation Engine." A location under a Jupiter Zenith line is only recommended if Jupiter is dignified; if Jupiter is in Fall (Capricorn) and heavily afflicted, the location may expand difficulties rather than wealth.

Part II: Computational Architecture and Ephemeris Integration

The precision of this system depends entirely on the quality of the astronomical data. Approximate algorithms (like low-precision Keplerian elements) are explicitly rejected in favor of the **Swiss Ephemeris**, the industry standard derived from NASA JPL's DE431 data.

2.1 The Swiss Ephemeris (WASM) Strategy

To enable high-performance calculation in a TypeScript/Node.js environment without the overhead of C++ bindings or native compilation issues, we utilize **WebAssembly (WASM)**.

Library Selection: swisseph-wasm (or @fusionstrings/swisseph-wasm).[6]

- **Why WASM?** It allows the original C code of the Swiss Ephemeris to run at near-native speed within the Node.js runtime or even client-side in the browser. This is crucial for the

iterative solvers required for Paran calculation, which may need thousands of position checks per request.

- **Data Management:** The ephemeris files (.se1) must be managed carefully. The main files (sepl_18.se1, semo_18.se1, seas_18.se1) cover the years 1800–2199 and are roughly 2-5MB each.
- **Architecture:** The PlanetaryService will implement a singleton pattern to load the WASM module and initialize the ephemeris path once.

2.2 Coordinate System Mathematics

The Swiss Ephemeris primarily returns **Geocentric Ecliptic** coordinates (λ, β, Δ). The engine must implement rigorous transformation matrices to derive the **Equatorial** coordinates (α, δ) required for all declination and paran work.

2.2.1 The Transformation Algorithm

While swisseph allows retrieving equatorial coordinates via flags (SEFLG_EQUATORIAL), implementing the raw conversion ensures the system can handle hypothetical points or fixed stars injected from other datasets.

Given:

- λ: Ecliptic Longitude
- β: Ecliptic Latitude
- ϵ: True Obliquity of the Ecliptic

The transformation to Right Ascension (α) and Declination (δ) is:

$$\sin\delta = \sin\beta\cos\epsilon + \cos\beta\sin\epsilon\sin\lambda$$

$$\cos\alpha\cos\delta = \cos\beta\cos\lambda$$

$$\sin\alpha\cos\delta = \cos\beta\cos\epsilon\sin\lambda - \sin\beta\sin\epsilon$$

To solve for α without quadrant ambiguity, we utilize the atan2 function:

$$\alpha = \arctan 2((\cos\beta\cos\epsilon\sin\lambda - \sin\beta\sin\epsilon), (\cos\beta\cos\lambda))$$

2.2.2 Topocentric Parallax Correction

For standard charts, Earth-center (Geocentric) coordinates suffice. However, for **Zenith lines**

and strict **Paran** interactions, the observer's position on the Earth's crust matters. This is the **Topocentric** coordinate system.[1]

- **The Moon's Parallax:** The Moon is close enough to Earth that its position varies by up to 1° depending on the observer's location. This shifts the Zenith line by ~60-70 miles—the entire width of the "orb of influence."
- **Implementation:** The PlanetaryService must expose methods to set the observer's geographic location (swe_set_topo) before calculating lunar positions. This requires inputs for Longitude, Latitude, and Altitude (meters).

2.3 TypeScript Interface Definitions

To ensure type safety and code maintainability, strict interfaces are defined for the astronomical data vectors.

TypeScript

```
/**
 * Core interface for a calculated planetary body.
 * Normalized to J2000 epoch where applicable.
 */
export interface CelestialBody {
  id: number; // Swiss Ephemeris ID (e.g., SE_SUN = 0)
  name: string;
  ecliptic: {
    longitude: number; // 0-360 degrees
    latitude: number;  // +/- 90 degrees
    distance: number;  // AU
    speed: number;     // degrees/day
  };
  equatorial: {
    rightAscension: number; // 0-360 degrees (converted from hours)
    declination: number;    // +/- 90 degrees
    speedDec: number;       // degrees/day (critical for station detection)
  };
  isRetrograde: boolean;
  isOutOfBounds: boolean; // Derived from declination vs obliquity
  house?: number; // Calculated only if location is provided
}

/**
```

```
 * Configuration for the calculation request.
 */
export interface EphemerisConfig {
  date: Date; // UTC
  location?: {
    latitude: number;
    longitude: number;
    altitude?: number; // meters, defaults to 0
  };
  flags: number; // Bitwise flags for swisseph options
  bodies: number; // Array of body IDs to calculate
}
```

Part III: The Paran and Geospatial Algorithms

This section details the mathematical solvers required to find the intersections of planetary lines. This is the "secret sauce" of the architecture, transforming raw data into geographic intelligence.

3.1 Zenith Line Logic

The calculation of Zenith lines is straightforward but fundamental. A planet is at the zenith of a location if and only if the location's geodetic latitude (ϕ) equals the planet's declination (δ) and the location's longitude corresponds to the planet's Greenwich Hour Angle (GHA) at that moment.[1]

For the purpose of a "Zenith Map" (where on Earth *could* this planet be zenith?), we only care about latitude.

- **Condition:** $\phi_{zenith} = \delta_{planet}$
- **Orb:** The active zone is $\phi \pm 1^\circ$.
- ** Visualization:** A horizontal band wrapping the globe.

3.2 The Astrocartography (ACG) Solver

ACG lines are the loci of points where a planet is on one of the four angles (ASC, DSC, MC, IC). These are curves on the map.

Fundamental Equation of Rising/Setting:

For a planet with Declination δ and Right Ascension α, the **Semi-Diurnal Arc (H)**—the

angular distance from the meridian to the horizon—at latitude ϕ is given by:

$$\cos H = -\tan\phi \tan\delta$$

- If $|\tan\phi \tan\delta| > 1$, the planet is circumpolar (always up or always down), and no rise/set line exists at that latitude.

Derivation of Longitude Points:

To plot the line, we iterate through latitudes ϕ (e.g., from -80 to +80) and solve for the terrestrial longitude λ_{geo}.

1. Calculate $H = \arccos(-\tan\phi \tan\delta)$.
2. **Rising (ASC):** Local Sidereal Time (LST) = $\alpha - H$.
3. **Setting (DSC):** $LST = \alpha + H$.
4. **Culminating (MC):** $LST = \alpha$.
5. **Anti-Culminating (IC):** $LST = \alpha + 180^\circ$.

Once LST is known, we convert to terrestrial longitude using the Greenwich Sidereal Time (GST) at the epoch:

$$\lambda_{geo} = LST - GST$$

(Note: Result must be normalized to -180 to +180 range).

3.3 The Paran Intersection Solver

Parans are the intersections of these ACG lines. A Paran exists at a latitude ϕ where Planet A is at Angle X and Planet B is at Angle Y **simultaneously**. Because the Earth rotates, this relationship holds true for the entire circle of latitude ϕ.[9]

The Problem: Find ϕ such that $LST_{EventA}(\phi) = LST_{EventB}(\phi)$.

Let's derive the equation for a **Planet A Rising / Planet B Setting** Paran:

$$LST_{rise,A} = \alpha_A - \arccos(-\tan\phi \tan\delta_A)$$

$$LST_{set,B} = \alpha_B + \arccos(-\tan\phi\tan\delta_B)$$

Equating them:

$$\alpha_A - \arccos(-\tan\phi\tan\delta_A) = \alpha_B + \arccos(-\tan\phi\tan\delta_B)$$

Rearranging to form an error function $f(\phi)$ to solving for 0:

$$f(\phi) = (\alpha_A - \alpha_B) - \arccos(-\tan\phi\tan\delta_A) - \arccos(-\tan\phi\tan\delta_B) = 0$$

Numerical Solution Strategy:

This equation cannot always be solved analytically due to the complex domain constraints of the arccos function. We employ a numerical root-finding algorithm.

1. **Search Domain:** Latitudes $[-89^\circ, +89^\circ]$.
2. **Algorithm: Bisection Method** is preferred over Newton-Raphson here because the function can be discontinuous (circumpolar regions) and non-differentiable at the poles.
3. **Iteration:**
   - Evaluate $f(\phi)$ at intervals of 1°.
   - If the sign of $f(\phi)$ changes between ϕ_i and ϕ_{i+1}, a root exists.
   - Bisect the interval until the error is $< 0.001^\circ$ (approx 100 meters).

Optimization:

The solver must handle four cases for each pair of planets:

1. Rise/Rise
2. Rise/Set
3. Rise/Culminate (Paran of 2nd Order)
4. Culminate/Culminate (Meridian alignment)

The result is a set of active latitudes representing the "Paran Belts."

Part IV: The Essential Dignity Scoring Engine

To allow users to filter for "Wealth" or "Love," we cannot simply look for Venus lines. We must assess the *condition* of Venus. This requires a programmable Dignity Engine.

4.1 Data Structures for Bounds/Terms

The system must support switching between **Egyptian** and **Ptolemaic** bounds. These are

best modeled as constant lookup tables in JSON format.[10]

TypeScript

```
// Structure for Term/Bound definitions
type PlanetName = 'Jupiter' | 'Venus' | 'Mercury' | 'Mars' | 'Saturn';

interface Term {
  ruler: PlanetName;
  degrees: number; // Cumulative degree end point (e.g., 6, 12, 20...)
}

const EGYPTIAN_TERMS: Record<string, Term> = {
  Aries:,
  //... Definitions for all 12 signs
};
```

4.2 The Scoring Algorithm

The DignityCalculator class will aggregate scores. This logic is critical for the "Safety Check" in the geospatial search.[3]

Algorithm Logic:

1. **Inputs:** Planet Position (λ), Sign, Chart Sect (Day/Night).
2. **Rulership:** If Planet rules Sign $\rightarrow +5$.
3. **Exaltation:** If Planet in Exaltation Sign $\rightarrow +4$.
4. **Triplicity:**
   - Identify Element of Sign (Fire/Earth/Air/Water).
   - Identify Triplicity Lords for that Element (Dorothean system: Day/Night/Participating).
   - If Planet == Sect Lord $\rightarrow +3$.
5. **Terms:**
   - Scan the EGYPTIAN_TERMS array for the given Sign.
   - Find the interval containing the planet's degree.
   - If Planet == Term Ruler $\rightarrow +2$.
6. **Face:** Calculate Face ruler (Chaldean Order) $\rightarrow +1$.
7. **Debilities:** Check for Detriment (-5) and Fall (-4).

8. **Peregrine Check:** If total positive score is 0, planet is Peregrine (-5).

Output: A DignityScore object containing the total score and a breakdown of dignities. This object is attached to the ACG lines in the visualization.

Part V: Geospatial Optimization ("Vibe Search")

This phase connects the celestial mathematics to terrestrial locations. It answers the user query: "Where should I live for Wealth?"

5.1 The "Vibe" Translation Layer

Users do not query "Jupiter Parans"; they query "Outcomes." The engine requires a mapping layer.

- **Wealth:** Jupiter, Venus, Part of Fortune.
- **Career:** Sun, Saturn, Mars, MC.
- **Love:** Venus, Moon, Jupiter (for abundance).
- **Spirituality:** Neptune, Sun, 12th House rulers.

5.2 The Geospatial Search Algorithm

1. **Target Generation:**
   - Identify target planets based on user goal.
   - Retrieve their Zenith Latitudes ($\phi = \delta$).
   - Calculate Paran Latitudes between target planets (e.g., Jupiter Rising + Sun MC).
2. **City Database Query:**
   - We require a high-performance city database (e.g., **GeoNames** via all-the-cities NPM package).
   - **Query:** SELECT * FROM Cities WHERE ABS(Latitude - TargetLatitude) < 1.0.
3. **Relocation Chart Verification (The Safety Filter):**
   - Mere proximity to a line is insufficient. We must verify the context.[1]
   - For each candidate city, cast a **Relocation Chart** (recalculating houses for the new location).
   - **Filter Logic:**
     - Is the Target Planet in the 6th, 8th, or 12th house? (Cadent/Difficult). If yes, penalize score.
     - Is the Target Planet making a hard aspect (Square/Opp) to the Ascendant? If yes, flag as intense.
4. **Ranking:** Sort cities by (Line Strength + Dignity Score - Relocation Penalties).

Part VI: Visualization Architecture (Three.js)

The "Vertical" nature of this astrology demands a 3D visualization. A flat map distorts the

Great Circles of the ACG lines and fails to visually represent the Zenith "bands" effectively.

6.1 Tech Stack

- **React Three Fiber (R3F):** For declarative scene management in React.
- **Three-Globe (or Globe.gl):** For highly optimized Earth rendering, country borders, and coordinate mapping.[12]

6.2 Rendering Components

1. **Zenith Bands (Isolines):**
    - Rendered as RingGeometry or TubeGeometry encircling the globe at constant latitude.
    - **Shader Material:** Use a custom shader to create a "fading" effect at the edges of the 1-degree orb, visualizing the intensity gradient.
2. **ACG Lines (Great Circles):**
    - Convert the computed (λ, ϕ) array to Cartesian (x, y, z) vectors.
    - $$x = R\cos\phi\cos\lambda, \quad y = R\sin\phi, \quad z = -R\cos\phi\sin\lambda$$
    - Render using LineLoop to ensure smooth closure of the curves.
3. **Paran Intersections:**
    - Render glowing SphereGeometry meshes at the intersection points.
    - **Interactivity:** Raycasting (clicking) on a node reveals the specific planetary combination (e.g., "Jupiter Rise / Sun MC").

6.3 The Heatmap Overlay

To visualize the "Paran Belts" (latitudes of high activity) without cluttering the globe with thousands of lines:

- Implement a **Fragment Shader** on the globe's surface material.
- Pass the "Hot Latitudes" as a uniform array to the shader.
- In the shader, check the pixel's UV coordinate (which maps to Lat/Long).
- If the pixel's latitude is close to a Hot Latitude, mix the texture color with a "Gold" or "Green" tint. This creates an intuitive "climate map" of energetic favorability.[1]

Part VII: System Architecture and Implementation Plan

7.1 Phase 1: Core Engine (Weeks 1-3)

- **Goal:** Reliable planetary calculation.
- **Tasks:**
    - Set up TypeScript monorepo.
    - Integrate swisseph-wasm.

- Implement PlanetaryService with getBodyPosition(date, bodyId, flags).
- Implement CoordinateTransform utilities (Ecliptic -> Equatorial).
- Unit Test against NASA Horizons data for verification.

7.2 Phase 2: The Math Layer (Weeks 4-6)

- **Goal:** Solve for Lines and Parans.
- **Tasks:**
  - Implement ZenithCalculator.
  - Implement ACGLineSolver (Iterative longitude solver).
  - Implement ParanSolver (Bisection method for latitude roots).
  - Validate Paran latitudes against known case studies (e.g., JFK chart).

7.3 Phase 3: The Interpretation Layer (Weeks 7-8)

- **Goal:** Qualitative analysis.
- **Tasks:**
  - Implement DignityCalculator with JSON lookup tables for Ptolemaic/Egyptian bounds.
  - Implement ScoringService to weight planets based on user goals (Wealth/Love).

7.4 Phase 4: Geospatial & API (Weeks 9-10)

- **Goal:** Connect to the real world.
- **Tasks:**
  - Ingest GeoNames database into MongoDB/Postgres with geospatial indexing.
  - Create LocationSearchAPI (Vibe Search).
  - Implement RelocationChart logic for safety checks.

7.5 Phase 5: Visualization & UI (Weeks 11-14)

- **Goal:** 3D Frontend.
- **Tasks:**
  - Set up React/R3F environment.
  - Implement Globe component.
  - Build LineRenderer and ZenithBand components.
  - Integrate UI controls for filtering planets and selecting dignities.

Conclusion

This architecture represents a paradigm shift from "Horoscope Generation" to "Astrological Engineering." By rigorously applying the physics of celestial mechanics—specifically the geometry of Declination and Parans—and leveraging the performance of WebAssembly and WebGL, we create a tool that reveals the hidden, vertical dimension of the sky.

The resulting system is not merely a descriptive tool but an **optimization engine**, capable of

guiding users to specific terrestrial latitudes where cosmic geometry aligns with their intent. This is the realization of the "Astrologer-Engineer" vision: a synthesis of ancient wisdom and modern computation.

Appendix A: Mathematical Derivations & TypeScript Implementation Details

A.1 Coordinate Transformations (Ecliptic to Equatorial)

The conversion from the Ecliptic system (standard astrological longitude) to the Equatorial system (Right Ascension/Declination) is the primary vector operation.

Mathematical Definition:

Given:

- λ (Lambda): Geocentric Ecliptic Longitude
- β (Beta): Geocentric Ecliptic Latitude
- ϵ (Epsilon): True Obliquity of the Ecliptic

Declination (δ):

$$\sin \delta = \sin \beta \cos \epsilon + \cos \beta \sin \epsilon \sin \lambda$$

Right Ascension (α):

$$\tan \alpha = \frac{\sin \lambda \cos \epsilon - \tan \beta \sin \epsilon}{\cos \lambda}$$

TypeScript Implementation:

TypeScript

```
/**
 * Utility to convert Ecliptic coordinates to Equatorial.
```

```
 * Handles the quadrant disambiguation for Right Ascension.
 */
export function toEquatorial(lambda: number, beta: number, epsilon: number) {
    const rad = (deg: number) => deg * Math.PI / 180;
    const deg = (rad: number) => rad * 180 / Math.PI;

    const l = rad(lambda);
    const b = rad(beta);
    const e = rad(epsilon);

    // Calculate Declination
    const sinDec = Math.sin(b) * Math.cos(e) + Math.cos(b) * Math.sin(e) * Math.sin(l);
    const declination = deg(Math.asin(sinDec));

    // Calculate Right Ascension with atan2 for quadrant safety
    const y = Math.sin(l) * Math.cos(e) - Math.tan(b) * Math.sin(e);
    const x = Math.cos(l);
    let ra = deg(Math.atan2(y, x));

    // Normalize RA to 0-360 range
    if (ra < 0) ra += 360;

    return { rightAscension: ra, declination };
}
```

A.2 The Paran Solver (Bisection Method)

Finding the latitude where two planets are simultaneously angular is a root-finding problem.

We seek the latitude ϕ where the difference in time between "Planet A Event" and "Planet B Event" is zero.

The Error Function:

$$f(\phi) = LST_{EventA}(\phi) - LST_{EventB}(\phi)$$

Where LST (Local Sidereal Time) for a rising/setting event is derived from the semi-diurnal arc H:

$$H = \arccos(-\tan\phi \tan\delta)$$

TypeScript Solver Implementation:

TypeScript

```typescript
/**
 * Solves for the latitude(s) where two planetary events coincide.
 * Uses Bisection Method for numerical stability.
 *
 * @param raA Right Ascension Planet A
 * @param decA Declination Planet A
 * @param typeA Event Type ('rise' | 'set' | 'culminate' | 'anticulminate')
 * @param raB Right Ascension Planet B
 * @param decB Declination Planet B
 * @param typeB Event Type
 * @returns Array of latitudes (degrees)
 */
export function solveParanLatitude(
  raA: number, decA: number, typeA: string,
  raB: number, decB: number, typeB: string
): number {
  const solutions: number =;
  const step = 1.0; // Check every degree for a root crossing

  // Iterate from South Pole to North Pole
  for (let lat = -89; lat <= 89; lat += step) {
    const val1 = getLSTDiff(lat, raA, decA, typeA, raB, decB, typeB);
    const val2 = getLSTDiff(lat + step, raA, decA, typeA, raB, decB, typeB);

    // If signs differ (and neither is NaN), a root exists in this interval
    if (!isNaN(val1) &&!isNaN(val2) && Math.sign(val1)!== Math.sign(val2)) {
      const root = bisectionSearch(lat, lat + step, raA, decA, typeA, raB, decB, typeB);
      if (root!== null) solutions.push(root);
    }
  }
  return solutions;
}

function getLSTDiff(lat: number, raA: number, decA: number, typeA: string,...): number {
  const lstA = calculateLST(raA, decA, lat, typeA);
  const lstB = calculateLST(raB, decB, lat, typeB);

  if (isNaN(lstA) |
```

```
| isNaN(lstB)) return NaN; // Circumpolar check

    // Normalize difference to -180 to +180 to handle 360 wrap
    let diff = lstA - lstB;
    while (diff <= -180) diff += 360;
    while (diff > 180) diff -= 360;

    return diff;
}

function bisectionSearch(min: number, max: number,...args): number | null {
    let low = min;
    let high = max;
    const epsilon = 0.001; // Precision threshold (approx 100m)

    for (let i = 0; i < 100; i++) { // Max iterations safety break
        const mid = (low + high) / 2;
        const val = getLSTDiff(mid,...args);

        if (Math.abs(val) < epsilon) return mid;

        // Standard bisection logic
        const lowVal = getLSTDiff(low,...args);
        if (Math.sign(val) === Math.sign(lowVal)) {
            low = mid;
        } else {
            high = mid;
        }
    }
    return (low + high) / 2;
}
```

A.3 Dignity Data Structures (JSON)

The application requires static data for the Essential Dignity scoring.

Ptolemaic Bounds (Terms) Example Structure:

```
    JSON
```

```
{
  "system": "Ptolemaic",
  "signs": {
    "Aries":,
    "Taurus":
    //... all 12 signs
  }
}
```

Triplicity Lords (Dorothean System):

JSON

```
{
  "system": "Dorothean",
  "elements": {
    "Fire": { "Day": "Sun", "Night": "Jupiter", "Participating": "Saturn" },
    "Earth": { "Day": "Venus", "Night": "Moon", "Participating": "Mars" },
    "Air": { "Day": "Saturn", "Night": "Mercury", "Participating": "Jupiter" },
    "Water": { "Day": "Venus", "Night": "Mars", "Participating": "Moon" }
  }
}
```

A.4 Three.js Visualization Logic

Converting Spherical to Cartesian for Globe Rendering:

The visualization layer must map the calculated (λ, ϕ) points onto the 3D sphere.

TypeScript

```
import * as THREE from 'three';

/**
 * Converts Geodetic coordinates to 3D Cartesian Vector.
 * @param lat Latitude (degrees)
```

```
 * @param lon Longitude (degrees)
 * @param radius Globe Radius (units)
 */
export function geoToVector(lat: number, lon: number, radius: number): THREE.Vector3 {
    const phi = (90 - lat) * (Math.PI / 180);
    const theta = (lon + 180) * (Math.PI / 180);

    const x = -(radius * Math.sin(phi) * Math.cos(theta));
    const z = (radius * Math.sin(phi) * Math.sin(theta));
    const y = (radius * Math.cos(phi));

    return new THREE.Vector3(x, y, z);
}
```

This appendix provides the concrete implementation details necessary for the engineering team to translate the high-level architecture into functioning code.

Works cited

1. Planetary Declination Research and Application
2. Paran Lines in Astrocartography Explained: How They Influence You - My Astro Diaries, accessed January 24, 2026, https://myastrodiary.com/diary/paran-latitude-lines
3. Essential Dignities - Astro Gold, accessed January 24, 2026, https://www.astrogold.io/AG-iOS-Help/essential_dignities.html
4. More on the Terms or Bounds | Altair Astrology - WordPress.com, accessed January 24, 2026, https://altairastrology.wordpress.com/2009/03/01/more-on-the-terms-or-bounds/
5. How to use egyptian terms? : r/astrology - Reddit, accessed January 24, 2026, https://www.reddit.com/r/astrology/comments/167k6cz/how_to_use_egyptian_terms/
6. This project is a Node.js wrapper for Swiss Ephemeris (Swisseph) - GitHub, accessed January 24, 2026, https://github.com/drvinaayaksingh/swisseph
7. fusionstrings/swisseph-wasm 0.1.5 on npm - Libraries.io, accessed January 24, 2026, https://libraries.io/npm/@fusionstrings%2Fswisseph-wasm
8. Computing planetary positions - Stjarnhimlen.se, accessed January 24, 2026, https://stjarnhimlen.se/comp/ppcomp.html
9. How to Read Your Astrocartography Map Step by Step {An Easy Guide} | Helena Woods | Astrocartographer, Author, Relocation Astrologer, accessed January 24, 2026, https://helenawoods.com/how-to-read-your-astrocartography-map-step-by-step-an-easy-guide/
10. What are the Bounds in Astrology? — Two Wander x Elysium Rituals, accessed January 24, 2026, https://www.twowander.com/blog/astrological-bounds

11. EGYPTIAN TERMS (BOUNDS): Replay is available for one week from 10/31 : r/astrology, accessed January 24, 2026, https://www.reddit.com/r/astrology/comments/qkhpm1/egyptian_terms_bounds_replay_is_available_for_one/
12. Globe.GL | globe.gl, accessed January 24, 2026, https://globe.gl/