3I/ATLAS 3D Flight Tracker - Comprehensive Technical Specification

Analysis Date: October 20, 2025
Project: 3iatlas Interactive 3D Visualization Enhancement
Repository: github.com/kjfsoul/3iatlas
Website: 3iatlas.mysticarcana.com

Table of Contents

---

1. Project Overview

1.1 Mission Statement

Build an immersive 3D visualization for 3I/ATLAS (C/2025 N1), the third confirmed interstellar object, providing educational, scientifically accurate flight tracking with engaging camera perspectives and interactive controls.

1.2 Key Facts

- **Discovery:** July 1, 2025 by ATLAS telescope, Chile
- **Perihelion:** October 29, 2025 at 1.356 AU
- **Eccentricity:** 6.14 (hyperbolic - will leave solar system)
- **SPK-ID:** 1004083
- **Age:** >7 billion years (from Milky Way thick disk)
- **Visibility:** Magnitude ~15 (telescope only)

1.3 Technology Stack

- **Framework:** Next.js 15.5.4 with TypeScript 5.6.3
- **3D Engine:** Three.js 0.180.0
- **React Integration:** React Three Fiber 8.15.19 + Drei 9.88.13
- **Data Source:** NASA JPL Horizons API (public, no authentication)
- **Styling:** Tailwind CSS 3.4.13
- **Animation:** Framer Motion 11.2.10
- **Astronomy Calculations:** astronomy-engine 2.1.19

---

2. Current Architecture Analysis

2.1 Project Structure

```
3iatlas/
├── app/
│   ├── api/
│   │   └── horizons/
│   │       ├── ephemeris/route.ts    # Proxy for NASA API
│   │       └── lookup/route.ts       # Object lookup
│   ├── layout.tsx                     # Root layout
│   └── page.tsx                       # Landing page
├── components/
│   ├── ui/
│   │   ├── CelestialLabel.tsx        # 3D object labels
│   │   └── TelemetryHUD.tsx          # Real-time metrics overlay
│   ├── views/
│   │   ├── HistoricalFlightView.tsx  # React Three Fiber scene
│   │   ├── AtlasViewsContainer.tsx   # View container/router
│   │   ├── ControlPanel.tsx          # Playback controls
│   │   └── ViewSelector.tsx          # View switching
│   ├── Atlas3DTracker.tsx            # Legacy vanilla Three.js
│   ├── Atlas3DTrackerEnhanced.tsx    # Enhanced vanilla version
│   ├── Atlas3DTrackerWrapper.tsx     # Wrapper component
│   ├── ClientOnly3DTracker.tsx       # Client-side wrapper
│   └── ThreeJSErrorBoundary.tsx      # Error handling
├── hooks/
│   └── useAdaptiveQuality.ts         # Performance optimization
├── lib/
│   ├── horizons-api.ts               # NASA API integration
│   ├── horizons-cache.ts             # API caching layer
│   ├── solar-system-data.ts          # Multi-object data fetching
│   ├── atlas-orbital-data.ts         # Fallback calculations
│   ├── trajectory-calculator.ts      # Trajectory math
│   └── view-manager.ts               # View state management
├── public/
│   └── trajectory.json               # Pre-calculated planet data
├── docs/                              # Extensive documentation
└── package.json
```

2.2 Key Files Analysis

2.2.1 Atlas3DTrackerEnhanced.tsx (1,339 lines)

Purpose: Comprehensive 3D tracker with vanilla Three.js
Key Features: - Real-time NASA Horizons data fetching - Multiple camera views (default, follow, topDown, closeup, marsApproach, rideComet) - Motion amplification system (configurable multiplier) - Particle trail system for comet - CSS overlay labels for planets - Playback controls (play/pause, speed, timeline scrubber) - Smooth camera following with lerp - Motion interpolation between data points - E2E test support

Camera Modes: 1. **default:** Solar system overview (8, 5, 8) 2. **followComet:** Dynamic following with configurable distance/height 3. **topDown:** Bird's eye view (0, 25, 0) 4. **closeup:** Close-up of perihelion approach 5. **marsApproach:** Mars + comet perspective 6. **rideComet:** First-person "riding" the comet (CURRENT BEST!)

State Management:

```
const [solarSystemData, setSolarSystemData] =
        useState<Record<string, VectorData[]>>({});
const [currentIndex, setCurrentIndex] = useState(0);
const [isPlaying, setIsPlaying] = useState(autoPlay);
const [speed, setSpeed] = useState(playbackSpeed);
const [cameraView, setCameraView] = useState<ViewType>("default");
const [showLabels, setShowLabels] = useState(true);
```

Animation Loop Architecture: - Uses `requestAnimationFrame` for smooth 60fps - Ref-based state for performance (avoids re-renders) - Interpolation between frames for smooth motion - Motion amplification specifically for 3I/ATLAS visibility

Issues: - Some camera views still need fine-tuning - Trail system could use enhancement for greenish coma effect - Labels sometimes overlap with objects

2.2.2 HistoricalFlightView.tsx (557 lines)

Purpose: React Three Fiber alternative implementation
Key Features: - Uses `@react-three/fiber` Canvas API - Declarative 3D scene composition - Integration with `@react-three/drei` helpers - OrbitControls for user interaction - Adaptive quality with `useAdaptiveQuality` hook - TelemetryHUD overlay - Playback controls in parent component

Architecture:

```
<Canvas camera={{ position: [7, 4, 7], fov: 75 }}>
  <Scene>
    <Sun />
    <Planet trajectory={earth} color="skyblue" />
    <Planet trajectory={mars} color="#ff6666" />
    <Comet trajectory={atlas} targetRef={cometRef} />
    <TrajectoryTrail trajectory={atlas} />
    <FollowCamera targetRef={cometRef} />
    <OrbitControls />
  </Scene>
</Canvas>
```

Issues: - Animation not as smooth as vanilla Three.js version - FollowCamera implementation needs improvement - Comet visual representation basic (sphere + cone) - No multiple camera view system

2.2.3 lib/horizons-api.ts (371 lines)

Purpose: NASA JPL Horizons API integration
Endpoints: - `/api/horizons/lookup` - Find object by name/designation - `/api/horizons/ephemeris` - Get ephemeris data

Data Structure:

```typescript
export interface VectorData {
  jd: number;           // Julian Date
  date: string;         // ISO 8601
  position: {
    x: number;          // AU
    y: number;          // AU
    z: number;          // AU
  };
  velocity: {
    vx: number;         // AU/day
    vy: number;         // AU/day
    vz: number;         // AU/day
  };
}
```

Functions: - `lookupObject()` - Search Horizons database - `getEphemerisVectors()` - Fetch position/velocity data - `parseVectorData()` - Parse Horizons text output - `get3IAtlasVectors()` - Complete workflow for 3I/ATLAS

Known Limitations: - No caching in API layer (done in separate file) - Text parsing brittle for non-standard formats - No retry logic for failed requests

2.2.4 lib/solar-system-data.ts (281 lines)

Purpose: Multi-object data fetching orchestration
Supported Objects:

```typescript
const SOLAR_SYSTEM_OBJECTS = {
  mercury: { command: '199', color: 0x8c7853, size: 0.025 },
  venus: { command: '299', color: 0xffc649, size: 0.038 },
  earth: { command: '399', color: 0x2266ff, size: 0.04 },
  mars: { command: '499', color: 0xff6644, size: 0.034 },
  jupiter: { command: '599', color: 0xd4a574, size: 0.12 },
  saturn: { command: '699', color: 0xfad5a5, size: 0.10 },
  uranus: { command: '799', color: 0x4fd0e7, size: 0.07 },
  neptune: { command: '899', color: 0x4166f5, size: 0.07 },
  pluto: { command: '999', color: 0xccaa88, size: 0.02 },
  voyager1: { command: '-31', color: 0xffff00, size: 0.03 },
  voyager2: { command: '-32', color: 0xffff00, size: 0.03 },
  atlas: { command: '1004083', color: 0x00ff88, size: 0.06 },
};
```

Fallback System: - Primary: NASA Horizons API - Secondary: `generate3IAtlasVectors()` from `atlas-orbital-data.ts` - Tertiary: Simple circular orbits for planets

2.2.5 hooks/useAdaptiveQuality.ts (111 lines)

Purpose: Dynamic quality adjustment based on FPS
Adjustable Parameters:

```typescript
interface AdaptiveQualityState {
  starCount: number;       // 2,000 - 20,000
  geometryDetail: number;  // 16 - 128 segments
  shadowMapSize: number;   // 512 - 4,096
  pixelRatio: number;      // 0.5 - 2.0
}
```

Quality Scaling Logic: - Monitors FPS over 2-second window (120 frames) - Degrades quality if FPS < 45 for 2+ seconds - Restores quality if FPS > 55 consistently - Exposes `__AE_HEALTH__` global for monitoring

---

3. Component Inventory

3.1 Core Components

Component	Type	Purpose	Status	Issu
Atlas3DTrackerEnhanced	Vanilla Three.js	Main 3D visualization	✅ Working	Came views need tuning
HistoricalFlightView	React Three Fiber	Alternative 3D view	⚠️ Partial	Anima issues
TelemetryHUD	UI Overlay	Real-time metrics	✅ Working	None
CelestialLabel	3D Text	Object labels	✅ Working	Overl issues
ControlPanel	UI Controls	Playback controls	⚠️ Partial	Speed contr broke
AtlasViewsContainer	Container	View routing	✅ Working	None
ViewSelector	UI Controls	View switching	✅ Working	None
ThreeJSErrorBoundary	Error Handling	3D error catching	✅ Working	None

3.2 Component Hierarchy

```
app/page.tsx
└── ClientOnly3DTracker
    └── Atlas3DTrackerWrapper
        └── Atlas3DTrackerEnhanced (MAIN COMPONENT)
            ├── Three.js Scene
            │   ├── Sun (static)
            │   ├── Planets (dynamic from Horizons)
            │   ├── 3I/ATLAS (dynamic with trail)
            │   ├── Orbital paths (lines)
            │   └── Starfield (background)
            ├── CSS Labels Container
            │   ├── Earth label
            │   └── Mars label
            └── Playback Controls
                ├── Play/Pause button
                ├── Reset button
                ├── Speed selector
                ├── Timeline scrubber
                └── Camera view selector
```

```
Alternative Path:
app/page.tsx
└── AtlasViewsContainer
    └── HistoricalFlightView (React Three Fiber)
        ├── Canvas
        │   └── Scene
        │       ├── Sun
        │       ├── Planet (Earth)
        │       ├── Planet (Mars)
        │       ├── Comet
        │       ├── TrajectoryTrail
        │       └── FollowCamera
        └── TelemetryHUD
```

3.3 Props Interfaces

Atlas3DTrackerEnhanced Props

```typescript
interface Props {
  startDate?: string;         // Default: "2025-10-01"
  endDate?: string;           // Default: "2025-10-31"
  stepSize?: "1h" | "6h" | "12h" | "1d"; // Default: "6h"
  autoPlay?: boolean;         // Default: true
  playbackSpeed?: number;     // Default: 2
  className?: string;
  animationKey?: 'warning' | 'artifact' | 'discovery' | null;
}
```

HistoricalFlightView Props

```typescript
interface HistoricalFlightViewProps {
  atlasData: VectorData[];
  isPlaying: boolean;
  speed: number;
  currentIndex: number;
  onPlayPause: () => void;
  onReset: () => void;
  onIndexChange: (index: number) => void;
  onSpeedChange: (speed: number) => void;
}
```

---

4. Data Flow Architecture

4.1 Data Sources

Primary: NASA Horizons API (Real-time)

```
User Request
    ↓
fetchSolarSystemData(['atlas', 'earth', 'mars'])
    ↓
For each object:
    ↓
/api/horizons/ephemeris?COMMAND=<spkid>&START_TIME=...
    ↓
NASA JPL Horizons System
```

```
    ↓
Text response ($$SOE...$$EOE)
    ↓
parseVectorData()
    ↓
VectorData[] arrays
```

Secondary: Pre-calculated Fallback

```
/public/trajectory.json
    ↓
Import at build time
    ↓
Static planet positions (July-Oct 2025)
```

Tertiary: Orbital Calculations

```
lib/atlas-orbital-data.ts
    ↓
generate3IAtlasVectors()
    ↓
Mathematical orbital propagation
    ↓
Fallback VectorData[]
```

4.2 State Flow

```
Atlas3DTrackerEnhanced
├── Fetch Phase
│   ├── fetchSolarSystemData() → solarSystemData state
│   ├── loading state → true/false
│   └── error state → string | null
│
├── Playback Phase
│   ├── isPlaying ref → animation loop control
│   ├── speedRef → animation speed multiplier
│   ├── currentIndexRef → current frame (mutable)
│   └── currentIndex state → UI updates only
│
├── Scene Phase
│   ├── sceneRef → { scene, camera, renderer, objects, controls }
│   ├── objectEntriesRef → cached data entries
│   ├── maxFrameCountRef → total frames
│   └── sharedVectorsRef → reusable Three.Vector3 instances
│
└── Render Phase
    ├── Animation loop updates positions
    ├── Camera system (based on cameraView)
    ├── Labels projection to 2D
    └── Controls update (if free-look mode)
```

4.3 Animation Loop Logic

```
animate() {
  requestAnimationFrame(animate);

  // 1. Update time index
  if (isPlaying) {
```

```
    localIndex += deltaTime * speed * 3;
    if (localIndex >= maxFrames) localIndex = 0;
  }

  // 2. Update all object positions
  for (const [key, vectors] of objectEntries) {
    const frameIndex = Math.floor(localIndex);
    const currentVec = vectors[frameIndex];

    // Interpolate for smooth motion
    if (interpolationEnabled && frameIndex < vectors.length - 1) {
      const nextVec = vectors[frameIndex + 1];
      const t = localIndex - frameIndex;
      finalPosition.lerpVectors(currentPos, nextPos, t);

      // Amplify comet motion for visibility
      if (key === 'atlas') {
        offset.subVectors(nextPos, currentPos)
              .multiplyScalar(cometMotionMultiplier - 1);
        finalPosition.add(offset);
      }
    }

    mesh.position.copy(finalPosition);
  }

  // 3. Update comet trail
  // Shift trail positions, add new head position

  // 4. Update labels
  // Project 3D positions to 2D screen coordinates

  // 5. Update camera
  switch (cameraView) {
    case 'followComet': /* smooth following logic */
    case 'rideComet': /* first-person perspective */
    case 'topDown': /* fixed overhead */
    // ... etc
  }

  // 6. Render
  if (freeLookMode) controls.update();
  renderer.render(scene, camera);
}
```

---

5. Implementation Gap Analysis

5.1 Playback Controls Issues

Problem: Speed Control Not Fully Responsive

Current State: - Speed dropdown exists with values: 0.5x, 1x, 2x, 5x, 10x - State updates correctly (`setSpeed()` called) - Ref updates correctly (`speedRef.current = speed`)

Root Cause: - Animation loop uses `speedRef.current * 3` as multiplier - The `* 3` is hardcoded, making actual speeds: - 0.5x → 1.5x - 1x → 3x - 2x → 6x - 5x → 15x - 10x → 30x

Fix Required:

```
// Current (line 679):
localIndex += dt * speedRef.current * 3;

// Should be:
localIndex += dt * speedRef.current;

// Then adjust default speed options to account for frame rate:
<option value={1}>Slow (1x)</option>
<option value={3}>Normal (3x)</option>
<option value={6}>Fast (6x)</option>
<option value={15}>Very Fast (15x)</option>
<option value={30}>Ultra Fast (30x)</option>
```

Problem: Timeline Scrubber Doesn't Pause

Current State: - User can drag timeline slider - Sets `currentIndex` and `currentIndexRef` - Calls `setIsPlaying(false)` (line 1104)

Issue: - `isPlaying` state updates, but may not stop animation loop immediately - Race condition between state update and next animation frame

Fix Required:

```
// Add immediate ref update:
onChange={(e) => {
  const newIndex = Number(e.target.value);
  setCurrentIndex(newIndex);
  currentIndexRef.current = newIndex;
  setIsPlaying(false);
  isPlayingRef.current = false; // ← ADD THIS
}}
```

Problem: Reset Button Not Always Returning to Start

Current State: - Reset button calls: `setCurrentIndex(0); setIsPlaying(false);` - Works most of the time

Potential Issue: - Doesn't reset `currentIndexRef` - Doesn't stop animation loop ref

Fix Required:

```
// lines 1126-1129:
<button onClick={() => {
  setCurrentIndex(0);
  currentIndexRef.current = 0; // ← ADD THIS
  setIsPlaying(false);
  isPlayingRef.current = false; // ← ADD THIS
}}>
```

5.2 Camera System Gaps

Current Camera Views

View	Status	Issues
default	✅ Working	Fixed position, basic
followComet	⚠ Partial	Too much smoothing, loses comet sometimes
topDown	✅ Working	Good for overview
closeup	⚠ Partial	Sometimes too close or too far
marsApproach	⚠ Partial	Mars scaling not consistent
rideComet	✅ Working	BEST VIEW! Needs more exposure

Missing Capabilities

1. FollowCamera Improvements Needed:

```
// Current (lines 834-849):
case 'followComet': {
  const followDistance = 5;
  const height = 2;
  const smoothing = 0.02;

  // Issues:
  // - Fixed distance doesn't account for comet speed
  // - Height is constant (should be relative to orbital plane)
  // - Smoothing too aggressive when comet accelerates
}
```

Better Implementation:

```
case 'followComet': {
  // Dynamic distance based on velocity
  const velocity = currentVec.velocity;
  const speed = Math.sqrt(velocity.vx**2 + velocity.vy**2 +
        velocity.vz**2);
  const dynamicDistance = 5 + speed * 10; // Further back when
        faster

  // Orbital plane normal (perpendicular to velocity)
  const velVector = new THREE.Vector3(velocity.vx, velocity.vz, -
        velocity.vy);
  const sunVector = cometPosition.clone().normalize();
  const orbitalNormal = new THREE.Vector3().crossVectors(velVector,
        sunVector);

  // Position camera on normal (above orbital plane)
  const idealPos = cometPosition.clone()
    .add(velVector.clone().normalize().multiplyScalar(-
        dynamicDistance))
    .add(orbitalNormal.clone().normalize().multiplyScalar(height));

  // Adaptive smoothing (less when fast, more when slow)
  const adaptiveSmoothing = 0.05 / (1 + speed);
  camera.position.lerp(idealPos, adaptiveSmoothing);
  camera.lookAt(cometPosition);
}
```

2. New Camera View Needed: "Mars Flyby"

```
case 'marsFlyby': {
  // Show 3I/ATLAS approaching Mars from Mars's perspective
  const marsData = solarSystemData["mars"];
  if (!marsData || !atlasData) break;

  const marsPos = marsData[currentIdx].position;
  const atlasPos = atlasData[currentIdx].position;

  const marsPosVec = new THREE.Vector3(marsPos.x, marsPos.z, -
        marsPos.y);
  const atlasPosVec = new THREE.Vector3(atlasPos.x, atlasPos.z, -
        atlasPos.y);

  // Camera positioned at Mars, looking toward comet
  camera.position.copy(marsPosVec);
  camera.lookAt(atlasPosVec);

  // Adjust FOV based on distance
  const distance = marsPosVec.distanceTo(atlasPosVec);
  camera.fov = THREE.MathUtils.clamp(75 / distance, 30, 120);
  camera.updateProjectionMatrix();
}
```

3. Needed: Cinematic Camera Sequences

```
interface CameraSequence {
  name: string;
  duration: number;  // seconds
  keyframes: Array<{
    time: number;    // 0-1
    view: ViewType;
    position?: THREE.Vector3;
    target?: THREE.Vector3;
    fov?: number;
  }>;
}

const PERIHELION_SEQUENCE: CameraSequence = {
  name: "Perihelion Approach",
  duration: 30,
  keyframes: [
    { time: 0.0, view: 'topDown' },
    { time: 0.2, view: 'followComet' },
    { time: 0.5, view: 'closeup' },
    { time: 0.8, view: 'rideComet' },
    { time: 1.0, view: 'topDown' }
  ]
};
```

5.3 Comet Visual Representation Gaps

Current Implementation (Atlas3DTrackerEnhanced)

```
// Nucleus (lines 395-426)
const geometry = new THREE.SphereGeometry(obj.size, 32, 32);
const material = new THREE.MeshBasicMaterial({ color: 0x00ff88 });
        // Green

// Glow (lines 430-438)
```

```
const glowGeo = new THREE.SphereGeometry(obj.size * 2.5, 32, 32);
const glowMat = new THREE.MeshBasicMaterial({
  color: 0x00ff88,
  transparent: true,
  opacity: 0.7
});

// Bloom (lines 441-449)
const bloomGeo = new THREE.SphereGeometry(obj.size * 4.0, 24, 24);
const bloomMat = new THREE.MeshBasicMaterial({
  color: 0x00ff88,
  transparent: true,
  opacity: 0.3
});

// Trail (lines 452-487)
const trailGeometry = new THREE.BufferGeometry();
const trailMaterial = new THREE.PointsMaterial({
  size: 0.05,
  vertexColors: true,
  transparent: true,
  opacity: 1.0,
  blending: THREE.AdditiveBlending
});
```

Issues with Current Representation

1. **Too simple** - Just sphere + point trail
2. **Wrong color** - Should be greenish (coma) + blue/white (tail)
3. **No directionality** - Tail doesn't point away from Sun
4. **Static appearance** - No dynamic activity
5. **Scale issues** - Trail too short, nucleus too uniform

Required Enhancements

1. Realistic Coma (Gas Cloud)

```
// Replace simple glow with particle system
class ComaParticleSystem {
  constructor(nucleusRadius: number) {
    this.geometry = new THREE.BufferGeometry();
    this.particleCount = 5000;

    const positions = new Float32Array(this.particleCount * 3);
    const colors = new Float32Array(this.particleCount * 3);
    const sizes = new Float32Array(this.particleCount);

    for (let i = 0; i < this.particleCount; i++) {
      // Spherical distribution around nucleus
      const radius = nucleusRadius + Math.random() * nucleusRadius *
        5;
      const theta = Math.random() * Math.PI * 2;
      const phi = Math.acos(2 * Math.random() - 1);

      positions[i * 3] = radius * Math.sin(phi) * Math.cos(theta);
      positions[i * 3 + 1] = radius * Math.sin(phi) *
        Math.sin(theta);
      positions[i * 3 + 2] = radius * Math.cos(phi);
```

```
      // Greenish color (CN, C2 molecules)
      const green = 0.7 + Math.random() * 0.3;
      colors[i * 3] = 0.2 * green;     // R
      colors[i * 3 + 1] = green;       // G
      colors[i * 3 + 2] = 0.3 * green; // B

      sizes[i] = 0.03 + Math.random() * 0.05;
    }

    this.geometry.setAttribute('position', new
        THREE.BufferAttribute(positions, 3));
    this.geometry.setAttribute('color', new
        THREE.BufferAttribute(colors, 3));
    this.geometry.setAttribute('size', new
        THREE.BufferAttribute(sizes, 1));

    this.material = new THREE.PointsMaterial({
      size: 0.05,
      vertexColors: true,
      transparent: true,
      opacity: 0.6,
      blending: THREE.AdditiveBlending,
      sizeAttenuation: true
    });

    this.particles = new THREE.Points(this.geometry, this.material);
  }

  update(sunPosition: THREE.Vector3, cometPosition: THREE.Vector3) {
    // Animate particles away from sun
    // Simulate solar wind pressure
  }
}
```

2. Realistic Tail (Dust and Ion)

```
class CometTail {
  constructor() {
    this.dustTail = this.createDustTail();
    this.ionTail = this.createIonTail();
  }

  createDustTail() {
    // Yellowish-white, curved, follows orbit
    const geometry = new THREE.BufferGeometry();
    const material = new THREE.PointsMaterial({
      color: 0xffeeaa,
      size: 0.03,
      transparent: true,
      opacity: 0.5,
      blending: THREE.AdditiveBlending
    });

    // Create curved tail (dust particles lag behind comet)
    const tailLength = 50; // points
    const positions = new Float32Array(tailLength * 3);
    // ... populate with curved path ...

    return new THREE.Points(geometry, material);
```

```
  }

  createIonTail() {
    // Bluish, straight, points directly away from sun
    const geometry = new THREE.BufferGeometry();
    const material = new THREE.LineBasicMaterial({
      color: 0x8888ff,
      transparent: true,
      opacity: 0.7,
      blending: THREE.AdditiveBlending
    });

    // Create straight tail (ion tail follows magnetic field)
    const tailPoints = [];
    for (let i = 0; i < 100; i++) {
      tailPoints.push(new THREE.Vector3(0, 0, i * 0.1));
    }

    geometry.setFromPoints(tailPoints);
    return new THREE.Line(geometry, material);
  }

  update(sunPos: THREE.Vector3, cometPos: THREE.Vector3, cometVel:
        THREE.Vector3) {
    // Update tail direction (away from sun)
    const sunToComet = cometPos.clone().sub(sunPos).normalize();

    // Ion tail: straight away from sun
    this.ionTail.position.copy(cometPos);
    this.ionTail.lookAt(cometPos.clone().add(sunToComet));

    // Dust tail: curved based on velocity
    // ... calculate curved path ...
  }
}
```

3. Dynamic Activity Based on Distance

```
function updateCometActivity(distanceFromSun: number) {
  // More activity when closer to sun
  const activityLevel = Math.max(0, 1 - (distanceFromSun / 3)); //
        Peak at <1 AU

  // Scale coma size
  coma.scale.setScalar(1 + activityLevel * 2);

  // Increase tail length
  tail.scale.setScalar(1 + activityLevel * 5);

  // Increase particle emission rate
  comaParticles.material.opacity = 0.3 + activityLevel * 0.5;

  // Change color temperature (warmer when closer)
  const heatColor = new THREE.Color().lerpColors(
    new THREE.Color(0x00ff88), // Green when far
    new THREE.Color(0xffee88), // Yellow-white when close
    activityLevel
  );
  coma.material.color.copy(heatColor);
```

```
}
```

5.4 Milestone Markers Gap

Not Implemented - No clickable 3D milestone markers exist

Requirements

1. **Key Events to Mark:**
   - Discovery (July 1, 2025)
   - First JWST observation (August 6, 2025)
   - Perihelion (October 29, 2025)
   - Mars closest approach
   - Exit solar system (future)
2. **Visual Representation:**
   - 3D icon/sphere at event location
   - Label with event name
   - Glow/pulse effect to attract attention
   - Clickable to show info panel
3. **Information Panel:**
   - Event date/time
   - Scientific significance
   - Distance measurements
   - Link to more details

Implementation Specification

```
interface Milestone {
  id: string;
  date: string;
  position: THREE.Vector3;
  title: string;
  description: string;
  icon: string;  // emoji or icon name
  color: number; // hex color
}

const MILESTONES: Milestone[] = [
  {
    id: 'discovery',
    date: '2025-07-01T00:00:00.000Z',
    position: new THREE.Vector3(-4.413, 0.074, -0.846), // from
        trajectory data
    title: 'Discovery',
    description: 'First detected by ATLAS telescope in Chile',
    icon: '🔭',
    color: 0xffff00
  },
  {
    id: 'perihelion',
    date: '2025-10-29T00:00:00.000Z',
    position: new THREE.Vector3(/* calculate from data */),
    title: 'Perihelion',
    description: 'Closest approach to Sun at 1.356 AU',
    icon: '🔥',
    color: 0xff4400
  },
  // ... more milestones ...
```

```
];

class MilestoneMarker extends THREE.Group {
  constructor(milestone: Milestone) {
    super();

    // Create marker sphere
    const markerGeo = new THREE.SphereGeometry(0.1, 16, 16);
    const markerMat = new THREE.MeshBasicMaterial({
      color: milestone.color,
      transparent: true,
      opacity: 0.8
    });
    const marker = new THREE.Mesh(markerGeo, markerMat);
    this.add(marker);

    // Create glow effect
    const glowGeo = new THREE.SphereGeometry(0.15, 16, 16);
    const glowMat = new THREE.MeshBasicMaterial({
      color: milestone.color,
      transparent: true,
      opacity: 0.3
    });
    const glow = new THREE.Mesh(glowGeo, glowMat);
    this.add(glow);

    // Pulse animation
    this.userData.pulsePhase = Math.random() * Math.PI * 2;

    // Click detection
    this.userData.milestone = milestone;
    this.userData.clickable = true;
  }

  update(time: number) {
    // Pulse animation
    const scale = 1 + Math.sin(time * 2 + this.userData.pulsePhase)
        * 0.1;
    this.scale.setScalar(scale);
  }
}
```

Interaction System:

```
// Raycasting for click detection
const raycaster = new THREE.Raycaster();
const mouse = new THREE.Vector2();

function onCanvasClick(event: MouseEvent) {
  // Calculate mouse position in normalized device coordinates
  mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
  mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;

  // Update picking ray
  raycaster.setFromCamera(mouse, camera);

  // Find intersections with clickable objects
  const intersects = raycaster.intersectObjects(scene.children,
        true);
```

```
  for (const intersect of intersects) {
    if (intersect.object.parent?.userData.clickable) {
      const milestone = intersect.object.parent.userData.milestone;
      showMilestonePanel(milestone);
      break;
    }
  }
}

function showMilestonePanel(milestone: Milestone) {
  // Show modal/panel with milestone info
  // Could use Framer Motion for animation
  // Include: title, date, description, images, links
}
```

5.5 Educational Content Gap

Not Implemented - No off-canvas educational panels

Requirements

1. **Content Types:**
   - Scientific facts about 3I/ATLAS
   - Interstellar object primer
   - Orbital mechanics explanation
   - How to observe (if visible)
   - Fun facts & comparisons
2. **UI Pattern:**
   - Slide-out panel (right side)
   - Tab system for different topics
   - Images, diagrams, videos
   - Links to NASA resources
3. **Triggers:**
   - Info button in UI
   - Clicking on objects
   - Reaching milestones
   - First-time visitor tutorial

Implementation Specification

```
interface EducationalContent {
  id: string;
  title: string;
  sections: Array<{
    heading: string;
    body: string;
    image?: string;
    video?: string;
    links?: Array<{ text: string; url: string }>;
  }>;
}

const CONTENT: EducationalContent[] = [
  {
    id: 'about-3iAtlas',
    title: 'About 3I/ATLAS',
```

```
    sections: [
      {
        heading: 'What is 3I/ATLAS?',
        body: '3I/ATLAS is the third confirmed interstellar object
        to visit our solar system...',
        image: '/images/3iatlas-discovery.jpg'
      },
      {
        heading: 'Why is it Special?',
        body: 'At over 7 billion years old, 3I/ATLAS predates our
        solar system...',
        links: [
          { text: 'NASA Science', url:
        'https://science.nasa.gov/...' },
          { text: 'JPL Horizons', url:
        'https://ssd.jpl.nasa.gov/...' }
        ]
      }
    ]
  },
  {
    id: 'interstellar-objects',
    title: 'Interstellar Visitors',
    sections: [
      {
        heading: '1I/'Oumuamua (2017)',
        body: 'The first known interstellar object, cigar-shaped,
        mysterious...',
        image: '/images/oumuamua.jpg'
      },
      {
        heading: '2I/Borisov (2019)',
        body: 'The first confirmed interstellar comet, showed
        familiar comet activity...',
        image: '/images/borisov.jpg'
      },
      {
        heading: '3I/ATLAS (2025)',
        body: 'YOU ARE HERE! The third visitor and oldest object
        ever observed...',
      }
    ]
  },
  {
    id: 'orbital-mechanics',
    title: 'How Orbits Work',
    sections: [
      {
        heading: 'What is a Hyperbolic Orbit?',
        body: 'Unlike elliptical orbits that loop forever,
        hyperbolic orbits are one-way trips...',
        image: '/images/orbit-types.png'
      }
    ]
  }
];

// React component for educational panel
function EducationalPanel({ isOpen, onClose }: { isOpen: boolean;
        onClose: () => void }) {
  const [activeTab, setActiveTab] = useState('about-3iAtlas');
```

```
  return (
    <motion.div
      initial={{ x: '100%' }}
      animate={{ x: isOpen ? 0 : '100%' }}
      transition={{ type: 'spring', damping: 20 }}
      className="fixed right-0 top-0 h-full w-96 bg-black/90
        backdrop-blur-lg border-l border-white/20 z-50"
    >
      <div className="p-6">
        <button onClick={onClose} className="absolute top-4 right-
        4">x</button>

        {/* Tab navigation */}
        <div className="flex gap-2 mb-6">
          {CONTENT.map(content => (
            <button
              key={content.id}
              onClick={() => setActiveTab(content.id)}
              className={`px-3 py-2 rounded ${activeTab ===
      content.id ? 'bg-green-600' : 'bg-white/10'}`}
            >
              {content.title}
            </button>
          ))}
        </div>

        {/* Content display */}
        <div className="overflow-y-auto h-[calc(100vh-120px)]">
          {CONTENT.find(c => c.id ===
      activeTab)?.sections.map((section, i) => (
            <div key={i} className="mb-6">
              <h3 className="text-xl font-bold text-white mb-2">
      {section.heading}</h3>
              <p className="text-white/80 mb-4">{section.body}</p>
              {section.image && <img src={section.image} alt=
      {section.heading} className="rounded mb-4" />}
              {section.links && (
                <div className="flex flex-col gap-2">
                  {section.links.map((link, j) => (
                    <a key={j} href={link.url} target="_blank"
      className="text-blue-400 hover:underline">
                      {link.text} →
                    </a>
                  ))}
                </div>
              )}
            </div>
          ))}
        </div>
      </div>
    </motion.div>
  );
}
```

5.6 Mobile Responsiveness Gaps

Current State

- Layout optimized for desktop (1920x1080+)

- Three.js canvas adapts to container size
- Controls can overflow on small screens
- Text sizes not optimized for mobile

Issues on Mobile

1. **Canvas Performance:** Lower FPS on mobile GPUs
2. **Touch Controls:** OrbitControls work but not optimized
3. **UI Density:** Too many controls cramped on small screen
4. **Text Readability:** Small fonts hard to read
5. **Network:** Larger data payloads on cellular

Required Enhancements

1. Responsive Layout

```
// Breakpoints
const MOBILE_BREAKPOINT = 768;
const TABLET_BREAKPOINT = 1024;

function useResponsive() {
  const [deviceType, setDeviceType] = useState<'mobile' | 'tablet' |
        'desktop'>('desktop');

  useEffect(() => {
    function checkDevice() {
      const width = window.innerWidth;
      if (width < MOBILE_BREAKPOINT) setDeviceType('mobile');
      else if (width < TABLET_BREAKPOINT) setDeviceType('tablet');
      else setDeviceType('desktop');
    }

    checkDevice();
    window.addEventListener('resize', checkDevice);
    return () => window.removeEventListener('resize', checkDevice);
  }, []);

  return deviceType;
}

// Adaptive quality based on device
function useDeviceQuality(deviceType: string) {
  return useMemo(() => {
    if (deviceType === 'mobile') {
      return {
        starCount: 1000,
        geometryDetail: 16,
        shadowMapSize: 512,
        pixelRatio: 1,
        trailLength: 10
      };
    } else if (deviceType === 'tablet') {
      return {
        starCount: 5000,
        geometryDetail: 32,
        shadowMapSize: 1024,
        pixelRatio: 1.5,
        trailLength: 20
```

```
      };
    } else {
      return {
        starCount: 10000,
        geometryDetail: 64,
        shadowMapSize: 2048,
        pixelRatio: 2,
        trailLength: 30
      };
    }
  }, [deviceType]);
}
```

2. Touch-Optimized Controls

```
// Simplified mobile UI
function MobileControls({ isPlaying, onPlayPause, speed,
        onSpeedChange }: ControlProps) {
  return (
    <div className="fixed bottom-0 left-0 right-0 bg-black/90 p-4">
      {/* Big touch-friendly play button */}
      <button
        onClick={onPlayPause}
        className="w-full mb-3 py-4 bg-green-600 rounded-xl text-
        2xl"
      >
        {isPlaying ? '⏸' : '▶'} {isPlaying ? 'Pause' : 'Play'}
      </button>

      {/* Simplified speed selector */}
      <div className="flex gap-2">
        {[1, 2, 5, 10].map(s => (
          <button
            key={s}
            onClick={() => onSpeedChange(s)}
            className={`flex-1 py-3 rounded-lg ${speed === s ? 'bg-
        green-600' : 'bg-white/20'}`}
          >
            {s}x
          </button>
        ))}
      </div>

      {/* Collapsible advanced controls */}
      <details className="mt-3">
        <summary className="text-white/70 text-sm">Advanced
        Settings</summary>
        {/* Camera views, etc */}
      </details>
    </div>
  );
}
```

3. Network Optimization

```
// Reduce data resolution on mobile
async function fetchTrajectoryData(deviceType: string) {
  const stepSize = deviceType === 'mobile' ? '12h' : '6h';  // Half
        the data points

  // Check if device prefers reduced data
```

```
  const connection = (navigator as any).connection;
  const slowConnection = connection?.effectiveType === '2g' ||
        connection?.effectiveType === '3g';

  if (slowConnection) {
    // Use even coarser data
    stepSize = '24h';
  }

  return fetchSolarSystemData(['atlas', 'earth', 'mars'], startDate,
        endDate, stepSize);
}
```

---

6. Enhancement Specifications

6.1 FollowCamera Enhancement

Goal: Create a cinematic camera system that smoothly follows 3I/ATLAS with intelligent positioning

Specification

Location: New component `components/CameraController.tsx`

```
import { useFrame, useThree } from '@react-three/fiber';
import { useRef } from 'react';
import * as THREE from 'three';

interface FollowCameraProps {
  targetRef: React.RefObject<THREE.Object3D>;
  mode: 'follow' | 'chase' | 'orbit' | 'static';
  distance?: number;
  height?: number;
  lookAhead?: number;  // How far ahead of comet to look
  smoothness?: number; // 0-1, higher = smoother but more lag
}

export function FollowCamera({
  targetRef,
  mode,
  distance = 5,
  height = 2,
  lookAhead = 2,
  smoothness = 0.1
}: FollowCameraProps) {
  const { camera } = useThree();
  const prevPositionRef = useRef(new THREE.Vector3());
  const velocityRef = useRef(new THREE.Vector3());
  const targetOffsetRef = useRef(new THREE.Vector3());

  useFrame((state, delta) => {
    if (!targetRef.current) return;

    const target = targetRef.current;
    const targetPos = target.position.clone();

    // Calculate velocity
```

```
    velocityRef.current.copy(targetPos).sub(prevPositionRef.current).divideScalar(delt

    prevPositionRef.current.copy(targetPos);

    // Calculate ideal camera position based on mode
    let idealPosition = new THREE.Vector3();
    let lookAtPosition = targetPos.clone();

    switch (mode) {
      case 'follow':
        // Camera behind and above, looking at comet
        const behindDir =
        velocityRef.current.clone().normalize().multiplyScalar(-
        distance);


        const aboveOffset = new THREE.Vector3(0, height, 0);
        idealPosition.copy(targetPos).add(behindDir).add(aboveOffset);


        break;

      case 'chase':
        // Camera behind, looking ahead of comet
        const chaseDir =
        velocityRef.current.clone().normalize().multiplyScalar(-
        distance * 1.5);
        idealPosition.copy(targetPos).add(chaseDir);
        lookAtPosition.add(velocityRef.current.clone().normalize().multiplyScalar(look


        break;

      case 'orbit':
        // Camera orbits around comet
        const time = state.clock.getElapsedTime();
        const angle = time * 0.2;
        idealPosition.set(
          Math.cos(angle) * distance,
          height,
          Math.sin(angle) * distance
        ).add(targetPos);
        break;

      case 'static':
        // Camera fixed in space, tracks comet
        idealPosition.set(distance, height, distance);
        break;
    }

    // Smoothly interpolate camera position
    camera.position.lerp(idealPosition, smoothness);

    // Smoothly interpolate look-at target
    targetOffsetRef.current.lerp(lookAtPosition, smoothness);
    camera.lookAt(targetOffsetRef.current);

    // Update camera matrix
    camera.updateMatrixWorld();
  });

  return null;
```

```
}
```

Integration:

```
// In HistoricalFlightView.tsx
import { FollowCamera } from '../CameraController';

// Inside Scene component:
<FollowCamera
  targetRef={cometRef}
  mode="follow"
  distance={5}
  height={3}
  lookAhead={1}
  smoothness={0.1}
/>
```

Test Plan

1. Load scene with comet visible
2. Switch to follow mode
3. Verify camera stays behind comet as it moves
4. Test different speeds (1x, 10x, 30x)
5. Verify no jerky motion or camera jumping
6. Test all four modes (follow, chase, orbit, static)

6.2 Camera View System Enhancement

Goal: Provide multiple preset camera perspectives with smooth transitions

Specification

Location: Enhanced in `Atlas3DTrackerEnhanced.tsx`

```
interface CameraPreset {
  name: string;
  description: string;
  icon: string;
  position: THREE.Vector3 | ((comet: THREE.Vector3) =>
        THREE.Vector3);
  target: THREE.Vector3 | ((comet: THREE.Vector3) => THREE.Vector3);
  fov?: number;
  transition?: 'instant' | 'smooth' | 'cinematic';
}

const CAMERA_PRESETS: Record<string, CameraPreset> = {
  overview: {
    name: 'Solar System Overview',
    description: 'See the entire inner solar system',
    icon: '🌍',
    position: new THREE.Vector3(0, 20, 20),
    target: new THREE.Vector3(0, 0, 0),
    fov: 60,
    transition: 'smooth'
  },

  followComet: {
```

```
    name: 'Follow Mode',
    description: 'Camera tracks behind the comet',
    icon: '📹',
    position: (comet) => {
      // Calculate position behind comet
      return comet.clone().add(new THREE.Vector3(-5, 3, -5));
    },
    target: (comet) => comet.clone(),
    transition: 'smooth'
  },

  rideComet: {
    name: 'Ride the Comet',
    description: 'Experience the journey first-hand',
    icon: '🚀',
    position: (comet) => {
      // Very close to comet, slightly offset
      return comet.clone().add(new THREE.Vector3(-0.2, 0.1, -0.2));
    },
    target: (comet) => {
      // Look ahead of comet
      return comet.clone().add(new THREE.Vector3(5, 0, 5));
    },
    fov: 90,
    transition: 'instant'
  },

  perihelion: {
    name: 'Perihelion Approach',
    description: 'Watch the closest pass to the Sun',
    icon: '☀',
    position: new THREE.Vector3(3, 1, 3),
    target: new THREE.Vector3(0, 0, 0),  // Look at Sun
    fov: 75,
    transition: 'cinematic'
  },

  marsFlyby: {
    name: 'Mars Flyby',
    description: 'View from Mars as comet passes',
    icon: '🔴',
    position: (comet) => {
      // Would need mars position from data
      // For now, approximate
      return new THREE.Vector3(1.5, 0, 0);
    },
    target: (comet) => comet.clone(),
    transition: 'smooth'
  },

  topDown: {
    name: 'Top-Down View',
    description: 'Birds-eye view of orbital plane',
    icon: '⬇',
    position: new THREE.Vector3(0, 30, 0),
    target: new THREE.Vector3(0, 0, 0),
    fov: 90,
```

```
    transition: 'smooth'
  }
};

// Camera transition system
function transitionCamera(
  camera: THREE.Camera,
  targetPos: THREE.Vector3,
  targetLookAt: THREE.Vector3,
  preset: CameraPreset,
  onComplete?: () => void
) {
  if (preset.transition === 'instant') {
    camera.position.copy(targetPos);
    camera.lookAt(targetLookAt);
    if (preset.fov) {
      (camera as THREE.PerspectiveCamera).fov = preset.fov;
      (camera as THREE.PerspectiveCamera).updateProjectionMatrix();
    }
    onComplete?.();
  } else if (preset.transition === 'smooth') {
    // Smooth lerp over ~1 second
    const startPos = camera.position.clone();
    const startLookAt = new THREE.Vector3(0, 0,
        -1).applyQuaternion(camera.quaternion);
    const startFov = (camera as THREE.PerspectiveCamera).fov;
    const targetFov = preset.fov || startFov;

    let progress = 0;
    const duration = 1000; // ms
    const startTime = performance.now();

    function animate() {
      const elapsed = performance.now() - startTime;
      progress = Math.min(elapsed / duration, 1);

      // Smooth ease-in-out
      const t = progress < 0.5
        ? 2 * progress * progress
        : 1 - Math.pow(-2 * progress + 2, 2) / 2;

      // Interpolate position
      camera.position.lerpVectors(startPos, targetPos, t);

      // Interpolate look-at
      const currentLookAt = new
        THREE.Vector3().lerpVectors(startLookAt, targetLookAt, t);
      camera.lookAt(currentLookAt);

      // Interpolate FOV
      (camera as THREE.PerspectiveCamera).fov = startFov +
        (targetFov - startFov) * t;
      (camera as THREE.PerspectiveCamera).updateProjectionMatrix();

      if (progress < 1) {
        requestAnimationFrame(animate);
      } else {
        onComplete?.();
      }
```

```
    }

    animate();
  } else if (preset.transition === 'cinematic') {
    // Dramatic curved path transition
    // TODO: Implement bezier curve camera path
  }
}
```

UI Component:

```
function CameraViewSelector({ currentView, onViewChange,
        cometPosition }: Props) {
  return (
    <div className="grid grid-cols-3 gap-2">
      {Object.entries(CAMERA_PRESETS).map(([key, preset]) => (
        <button
          key={key}
          onClick={() => onViewChange(key)}
          className={`
            flex flex-col items-center p-3 rounded-lg border-2
        transition-all
            ${currentView === key
              ? 'border-green-500 bg-green-500/20'
              : 'border-white/20 bg-white/5 hover:bg-white/10'}
          `}
          title={preset.description}
        >
          <span className="text-2xl mb-1">{preset.icon}</span>
          <span className="text-xs text-white/90">{preset.name}
        </span>
        </button>
      ))}
    </div>
  );
}
```

Test Plan

1. Load scene with comet at different positions
2. Click each camera preset button
3. Verify smooth transition (no jumping)
4. Verify correct positioning relative to comet
5. Test FOV changes
6. Test at different playback speeds

6.3 Playback Controls Fix

Goal: Fix broken speed control and ensure all playback controls work reliably

Issues & Fixes

Issue 1: Speed multiplier hardcoded

```
// BEFORE (line 679 in Atlas3DTrackerEnhanced.tsx):
localIndex += dt * speedRef.current * 3;  // ← Hardcoded multiplier

// AFTER:
```

```
localIndex += dt * speedRef.current;
```

Issue 2: Speed selector values don't match

```
// BEFORE:
<option value={0.5}>Slow (0.5x)</option>    // Actually 1.5x
<option value={1}>Normal (1x)</option>       // Actually 3x
<option value={2}>Fast (2x)</option>         // Actually 6x

// AFTER (account for frame rate):
<option value={1}>Slow</option>      // 1 frame/sec
<option value={3}>Normal</option>    // 3 frames/sec
<option value={6}>Fast</option>      // 6 frames/sec
<option value={15}>Very Fast</option> // 15 frames/sec
<option value={30}>Ultra</option>    // 30 frames/sec
```

Issue 3: Timeline scrubber doesn't pause

```
// BEFORE (line 1100-1105):
onChange={(e) => {
  const newIndex = Number(e.target.value);
  setCurrentIndex(newIndex);
  currentIndexRef.current = newIndex;
  setIsPlaying(false); // State update may not be immediate
}}

// AFTER:
onChange={(e) => {
  const newIndex = Number(e.target.value);
  setCurrentIndex(newIndex);
  currentIndexRef.current = newIndex;
  setIsPlaying(false);
  isPlayingRef.current = false; // ← Immediate ref update
}}
```

Issue 4: Reset doesn't always work

```
// BEFORE (line 1126-1129):
<button onClick={() => {
  setCurrentIndex(0);
  setIsPlaying(false);
}}>

// AFTER:
<button onClick={() => {
  setCurrentIndex(0);
  currentIndexRef.current = 0; // ← Reset ref
  setIsPlaying(false);
  isPlayingRef.current = false; // ← Stop animation immediately
}}>
```

Test Plan

1. **Speed Control:**
   - Set speed to each option (Slow, Normal, Fast, Very Fast, Ultra)
   - Verify comet moves at expected speed
   - Verify UI label matches actual speed
2. **Timeline Scrubber:**
   - Play animation

- Drag scrubber to middle
- Verify animation pauses immediately
- Verify comet jumps to correct position
- Resume playing - should continue from scrubbed position

3. **Reset Button:**
   - Let animation play for 10+ seconds
   - Click reset
   - Verify animation stops
   - Verify comet returns to start position
   - Verify timeline is at 0
4. **Play/Pause:**
   - Click play - animation starts
   - Click pause - animation stops immediately (no lag)
   - Verify button icon changes correctly
   - Verify state persists across camera view changes

6.4 Enhanced Comet Visuals

Goal: Create scientifically accurate, visually stunning comet representation with coma, tail, and dynamic activity

Specification

Location: New file `components/CometVisuals.tsx`

```
import * as THREE from 'three';
import { useFrame } from '@react-three/fiber';
import { useRef, useMemo } from 'react';

interface CometProps {
  position: THREE.Vector3;
  velocity: THREE.Vector3;
  sunPosition: THREE.Vector3;
  distanceFromSun: number;
}

export function CometVisuals({ position, velocity, sunPosition,
        distanceFromSun }: CometProps) {
  // Calculate activity level based on distance from sun
  const activityLevel = useMemo(() => {
    // More active when closer to sun
    // Peak activity at perihelion (1.356 AU)
    return Math.max(0, Math.min(1, (3 - distanceFromSun) / 2));
  }, [distanceFromSun]);

  return (
    <group position={position}>
      <Nucleus size={0.06} />
      <Coma
        size={0.3 * (1 + activityLevel * 3)}
        opacity={0.3 + activityLevel * 0.4}
      />
      <DustTail
        length={2 + activityLevel * 10}
        sunDirection={position.clone().sub(sunPosition).normalize()}
        velocity={velocity}
      />
```

```
      <IonTail
        length={5 + activityLevel * 20}
        sunDirection={position.clone().sub(sunPosition).normalize()}
        opacity={0.5 + activityLevel * 0.3}
      />
    </group>
  );
}

// Solid nucleus
function Nucleus({ size }: { size: number }) {
  return (
    <mesh>
      <sphereGeometry args={[size, 32, 32]} />
      <meshStandardMaterial
        color={0x334444}  // Dark gray-green
        roughness={0.9}
        metalness={0.1}
      />
    </mesh>
  );
}

// Greenish coma (gas cloud)
function Coma({ size, opacity }: { size: number; opacity: number })
        {
  const particlesRef = useRef<THREE.Points>(null);

  const [positions, colors, sizes] = useMemo(() => {
    const count = 5000;
    const positions = new Float32Array(count * 3);
    const colors = new Float32Array(count * 3);
    const sizes = new Float32Array(count);

    for (let i = 0; i < count; i++) {
      // Random position in sphere
      const r = Math.random() * size;
      const theta = Math.random() * Math.PI * 2;
      const phi = Math.acos(2 * Math.random() - 1);

      positions[i * 3] = r * Math.sin(phi) * Math.cos(theta);
      positions[i * 3 + 1] = r * Math.sin(phi) * Math.sin(theta);
      positions[i * 3 + 2] = r * Math.cos(phi);

      // Greenish color (CN, C2 molecules)
      const brightness = 0.7 + Math.random() * 0.3;
      colors[i * 3] = 0.1 * brightness;      // R
      colors[i * 3 + 1] = brightness;        // G
      colors[i * 3 + 2] = 0.3 * brightness;  // B

      sizes[i] = 0.02 + Math.random() * 0.04;
    }

    return [positions, colors, sizes];
  }, [size]);

  useFrame((state) => {
    if (particlesRef.current) {
```

```
      // Gentle rotation/pulsing
      particlesRef.current.rotation.y += 0.001;
      const scale = 1 + Math.sin(state.clock.elapsedTime * 2) *
        0.05;
      particlesRef.current.scale.setScalar(scale);
    }
  });

  return (
    <points ref={particlesRef}>
      <bufferGeometry>
        <bufferAttribute
          attach="attributes-position"
          count={positions.length / 3}
          array={positions}
          itemSize={3}
        />
        <bufferAttribute
          attach="attributes-color"
          count={colors.length / 3}
          array={colors}
          itemSize={3}
        />
        <bufferAttribute
          attach="attributes-size"
          count={sizes.length}
          array={sizes}
          itemSize={1}
        />
      </bufferGeometry>
      <pointsMaterial
        size={0.05}
        vertexColors
        transparent
        opacity={opacity}
        blending={THREE.AdditiveBlending}
        sizeAttenuation
        depthWrite={false}
      />
    </points>
  );
}

// Yellowish dust tail (curves, follows orbit)
function DustTail({
  length,
  sunDirection,
  velocity
}: {
  length: number;
  sunDirection: THREE.Vector3;
  velocity: THREE.Vector3;
}) {
  const points = useMemo(() => {
    const pts: THREE.Vector3[] = [];
    const segments = 50;
```

```
    // Dust tail curves due to solar radiation pressure + orbital
        motion
    // Lags behind ion tail
    for (let i = 0; i < segments; i++) {
      const t = i / segments;

      // Base direction: away from sun
      const dir = sunDirection.clone();

      // Add orbital lag (dust particles slower than comet)
      const lag = velocity.clone().multiplyScalar(-t * 0.5);
      dir.add(lag);

      // Calculate position along tail
      const point = dir.normalize().multiplyScalar(t * length);

      // Add some randomness/turbulence
      point.x += (Math.random() - 0.5) * t * 0.2;
      point.y += (Math.random() - 0.5) * t * 0.2;
      point.z += (Math.random() - 0.5) * t * 0.2;

      pts.push(point);
    }

    return pts;
  }, [length, sunDirection, velocity]);

  return (
    <line>
      <bufferGeometry>
        <bufferAttribute
          attach="attributes-position"
          count={points.length}
          array={new Float32Array(points.flatMap(p => [p.x, p.y,
        p.z]))}
          itemSize={3}
        />
      </bufferGeometry>
      <lineBasicMaterial
        color={0xffdd88}  // Yellowish
        transparent
        opacity={0.6}
        blending={THREE.AdditiveBlending}
        linewidth={2}
      />
    </line>
  );
}

// Bluish ion tail (straight, points directly away from sun)
function IonTail({
  length,
  sunDirection,
  opacity
}: {
  length: number;
  sunDirection: THREE.Vector3;
  opacity: number;
```

```
}) {
  const trailRef = useRef<THREE.Line>(null);

  const points = useMemo(() => {
    const pts: THREE.Vector3[] = [];
    const segments = 100;

    // Ion tail is straight (solar wind pushes ions directly away)
    for (let i = 0; i < segments; i++) {
      const t = i / segments;
      const point = sunDirection.clone().multiplyScalar(t * length);

      // Slight randomness for realism
      point.x += (Math.random() - 0.5) * 0.05;
      point.y += (Math.random() - 0.5) * 0.05;
      point.z += (Math.random() - 0.5) * 0.05;

      pts.push(point);
    }

    return pts;
  }, [length, sunDirection]);

  useFrame((state) => {
    if (trailRef.current) {
      // Subtle shimmer effect
      const shimmer = Math.sin(state.clock.elapsedTime * 5) * 0.1 +
        0.9;
      (trailRef.current.material as THREE.LineBasicMaterial).opacity
        = opacity * shimmer;
    }
  });

  return (
    <line ref={trailRef}>
      <bufferGeometry>
        <bufferAttribute
          attach="attributes-position"
          count={points.length}
          array={new Float32Array(points.flatMap(p => [p.x, p.y,
        p.z]))}
          itemSize={3}
        />
      </bufferGeometry>
      <lineBasicMaterial
        color={0x6699ff}  // Bluish
        transparent
        opacity={opacity}
        blending={THREE.AdditiveBlending}
        linewidth={1}
      />
    </line>
  );
}
```

Integration into Atlas3DTrackerEnhanced:

```
// Replace simple sphere (lines 395-488) with:
import { CometVisuals } from './CometVisuals';
```

```
// In scene setup:
const cometGroup = new THREE.Group();
scene.add(cometGroup);
objectMeshes.set('atlas', cometGroup);

// In animation loop (replace lines 746-790):
if (key === 'atlas') {
  // Update comet position
  cometGroup.position.copy(final);

  // Update visual components
  // (This would require converting to React Three Fiber
  //  OR manually managing Three.js objects)

  // For vanilla Three.js, create equivalent classes:
  // - CometNucleus (sphere)
  // - ComaParticleSystem (Points)
  // - DustTailGeometry (Line)
  // - IonTailGeometry (Line)
}
```

Test Plan

1. **Visual Quality:**
   - Comet should have visible green coma
   - Dust tail should be yellowish and curved
   - Ion tail should be bluish and straight
   - All should point away from Sun
2. **Dynamic Activity:**
   - At 3+ AU from Sun: minimal tail, small coma
   - At 2 AU: moderate activity
   - At perihelion (1.356 AU): maximum activity, long tails
   - Test at different distances
3. **Performance:**
   - Check FPS with new particle systems
   - Verify no memory leaks
   - Test on lower-end hardware
4. **Accuracy:**
   - Verify tail direction relative to Sun
   - Verify dust tail lags behind ion tail
   - Compare to reference images of real comets

6.5 Milestone Markers Implementation

Goal: Add interactive 3D markers for key events with clickable info panels

Specification

Location: New file `components/MilestoneMarkers.tsx`

```
import * as THREE from 'three';
import { Html } from '@react-three/drei';
import { useState } from 'react';
import { useFrame } from '@react-three/fiber';
```

```typescript
interface Milestone {
  id: string;
  date: string;
  julianDate: number;
  position: THREE.Vector3;
  title: string;
  shortDesc: string;
  fullDescription: string;
  icon: string;
  color: string;
  image?: string;
  links?: Array<{ text: string; url: string }>;
}

const MILESTONES: Milestone[] = [
  {
    id: 'discovery',
    date: '2025-07-01',
    julianDate: 2460857.5,
    position: new THREE.Vector3(-4.413, 0.074, -0.846),
    title: 'Discovery',
    shortDesc: 'First detected by ATLAS',
    fullDescription: 'On July 1, 2025, the ATLAS telescope in Chile
        detected an unusual object moving through the constellation
        Sagittarius. Initial observations revealed hyperbolic
        trajectory characteristics, indicating an origin outside our
        solar system.',
    icon: '🔭',
    color: '#ffff00',
    image: '/images/atlas-telescope.jpg',
    links: [
      { text: 'ATLAS Survey', url: 'https://atlas.fallingstar.com/'
        },
      { text: 'Discovery Paper', url: 'https://arxiv.org/...' }
    ]
  },
  {
    id: 'jwst-observation',
    date: '2025-08-06',
    julianDate: 2460893.5,
    position: new THREE.Vector3(/* calculate from trajectory data
        */),
    title: 'JWST Observation',
    shortDesc: 'James Webb Space Telescope',
    fullDescription: 'NASA\'s James Webb Space Telescope observed
        3I/ATLAS using its NIRSpec and MIRI instruments, capturing
        detailed spectroscopic data revealing the comet\'s chemical
        composition and temperature distribution.',
    icon: '🔬',
    color: '#ff8800',
    image: '/images/jwst-observation.jpg',
    links: [
      { text: 'JWST Science', url:
        'https://science.nasa.gov/mission/webb/' }
    ]
  },
  {
    id: 'mars-approach',
    date: '2025-10-15',
    julianDate: 2460963.5,
    position: new THREE.Vector3(/* Mars orbit intersection */),
```

```
    title: 'Mars Close Approach',
    shortDesc: 'Passes near Mars orbit',
    fullDescription: '3I/ATLAS crosses Mars\'s orbital path,
        providing a unique opportunity to compare its trajectory
        with inner solar system dynamics. Distance to Mars:
        approximately 0.5 AU.',
    icon: '🔴',
    color: '#ff4444',
  },
  {
    id: 'perihelion',
    date: '2025-10-29',
    julianDate: 2460977.5,
    position: new THREE.Vector3(/* perihelion position */),
    title: 'Perihelion',
    shortDesc: 'Closest approach to Sun',
    fullDescription: 'At perihelion, 3I/ATLAS reaches its closest
        distance to the Sun at 1.356 AU (just inside Mars\'s orbit).
        This is the moment of maximum activity, with peak coma
        brightness and tail development.',
    icon: '☀',
    color: '#ff2200',
    image: '/images/perihelion.jpg',
  },
  {
    id: 'exit',
    date: '2025-12-01',
    julianDate: 2461010.5,
    position: new THREE.Vector3(/* outbound position */),
    title: 'Departure',
    shortDesc: 'Exiting inner solar system',
    fullDescription: 'After passing perihelion, 3I/ATLAS accelerates
        back toward interstellar space, never to return. Its
        hyperbolic trajectory ensures this is our only chance to
        study this ancient visitor.',
    icon: '🚀',
    color: '#00ffff',
  }
];

export function MilestoneMarkers({
  currentJulianDate,
  onMilestoneClick
}: {
  currentJulianDate: number;
  onMilestoneClick: (milestone: Milestone) => void;
}) {
  // Only show milestones that have occurred or are close in time
  const visibleMilestones = MILESTONES.filter(m => {
    const daysDiff = m.julianDate - currentJulianDate;
    return daysDiff < 10 && daysDiff > -30; // Show if within 10
        days future or 30 days past
  });

  return (
    <>
      {visibleMilestones.map(milestone => (
        <MilestoneMarker
          key={milestone.id}
          milestone={milestone}
          currentDate={currentJulianDate}
```

```
          onClick={() => onMilestoneClick(milestone)}
        />
      ))}
    </>
  );
}

function MilestoneMarker({
  milestone,
  currentDate,
  onClick
}: {
  milestone: Milestone;
  currentDate: number;
  onClick: () => void;
}) {
  const markerRef = useRef<THREE.Group>(null);
  const [hovered, setHovered] = useState(false);

  // Check if milestone has occurred
  const occurred = currentDate >= milestone.julianDate;

  // Pulse animation
  useFrame((state) => {
    if (markerRef.current) {
      const time = state.clock.elapsedTime;
      const pulse = 1 + Math.sin(time * 3) * 0.1;
      markerRef.current.scale.setScalar(pulse);
    }
  });

  return (
    <group
      ref={markerRef}
      position={milestone.position}
      onClick={onClick}
      onPointerOver={() => setHovered(true)}
      onPointerOut={() => setHovered(false)}
    >
      {/* Marker sphere */}
      <mesh>
        <sphereGeometry args={[0.15, 16, 16]} />
        <meshBasicMaterial
          color={milestone.color}
          transparent
          opacity={occurred ? 0.9 : 0.5}
          emissive={milestone.color}
          emissiveIntensity={hovered ? 0.5 : 0.2}
        />
      </mesh>

      {/* Outer glow */}
      <mesh>
        <sphereGeometry args={[0.25, 16, 16]} />
        <meshBasicMaterial
          color={milestone.color}
          transparent
```

```
            opacity={occurred ? 0.3 : 0.1}
            side={THREE.BackSide}
          />
        </mesh>

        {/* Label (always visible) */}
        <Html
          distanceFactor={10}
          position={[0, 0.5, 0]}
          center
          style={{
            pointerEvents: 'none',
            transition: 'all 0.2s',
            opacity: hovered ? 1 : 0.8,
            transform: `scale(${hovered ? 1.1 : 1})`
          }}
        >
          <div className="bg-black/70 backdrop-blur-sm px-3 py-1
          rounded-full border border-white/30">
            <span className="text-xl mr-2">{milestone.icon}</span>
            <span className="text-white text-sm font-medium">
          {milestone.shortDesc}</span>
          </div>
        </Html>

        {/* Hover tooltip */}
        {hovered && (
          <Html
            distanceFactor={10}
            position={[0, 1.5, 0]}
            center
          >
            <div className="bg-black/90 backdrop-blur-md px-4 py-3
          rounded-lg border border-white/40 max-w-xs">
              <div className="text-white font-bold text-lg mb-1">
          {milestone.title}</div>
              <div className="text-white/70 text-xs mb-2">
          {milestone.date}</div>
              <div className="text-white/90 text-sm">
          {milestone.fullDescription.slice(0, 100)}...</div>
              <div className="text-green-400 text-xs mt-2">Click for
          more info</div>
            </div>
          </Html>
        )}
      </group>
    );
}

// Info panel component (shown when milestone clicked)
export function MilestoneInfoPanel({
  milestone,
  onClose
}: {
  milestone: Milestone | null;
  onClose: () => void;
}) {
  if (!milestone) return null;

  return (
```

```
    <div className="fixed inset-0 bg-black/70 backdrop-blur-sm z-50
        flex items-center justify-center">
      <div className="bg-gradient-to-br from-gray-900 to-black
        border border-white/30 rounded-xl p-6 max-w-2xl max-h-[80vh]
        overflow-y-auto">
        <button
          onClick={onClose}
          className="absolute top-4 right-4 text-white/70
        hover:text-white text-2xl"
        >
          ×
        </button>

        <div className="flex items-start gap-4 mb-4">
          <span className="text-6xl">{milestone.icon}</span>
          <div>
            <h2 className="text-3xl font-bold text-white mb-2">
        {milestone.title}</h2>
            <div className="text-white/60 text-sm">{milestone.date}
        </div>
          </div>
        </div>

        {milestone.image && (
          <img
            src={milestone.image}
            alt={milestone.title}
            className="w-full rounded-lg mb-4"
          />
        )}

        <p className="text-white/90 text-lg leading-relaxed mb-6">
          {milestone.fullDescription}
        </p>

        {milestone.links && (
          <div className="space-y-2">
            <div className="text-white/60 text-sm font-semibold mb-
        2">Learn More:</div>
            {milestone.links.map((link, i) => (
              <a
                key={i}
                href={link.url}
                target="_blank"
                rel="noopener noreferrer"
                className="block text-blue-400 hover:text-blue-300
        underline"
              >
                {link.text} →
              </a>
            ))}
          </div>
        )}
      </div>
    </div>
  );
}
```

Integration:

```
// In Atlas3DTrackerEnhanced.tsx or HistoricalFlightView.tsx:
```

```
const [selectedMilestone, setSelectedMilestone] = useState<Milestone
        | null>(null);

// In scene:
<MilestoneMarkers
  currentJulianDate={atlasData[currentIndex]?.jd || 0}
  onMilestoneClick={setSelectedMilestone}
/>

// Above canvas:
<MilestoneInfoPanel
  milestone={selectedMilestone}
  onClose={() => setSelectedMilestone(null)}
/>
```

Test Plan

1. **Visibility:**
   - Milestones appear at correct 3D positions
   - Icons/labels visible from different camera angles
   - Pulse animation smooth and not distracting
2. **Interaction:**
   - Hover shows expanded tooltip
   - Click opens full info panel
   - Close button dismisses panel
   - Multiple milestones don't overlap
3. **Timeline Integration:**
   - Milestones appear/disappear based on current date
   - Past milestones shown but dimmed
   - Future milestones hidden until close

6.6 Educational Content Integration

Goal: Provide comprehensive educational content via off-canvas panels

Specification

Location: New file `components/EducationalPanel.tsx`

```
import { motion, AnimatePresence } from 'framer-motion';
import { useState } from 'react';

interface EducationalSection {
  heading: string;
  body: string;
  image?: string;
  video?: string;
  links?: Array<{ text: string; url: string }>;
}

interface EducationalTopic {
  id: string;
  title: string;
  icon: string;
  sections: EducationalSection[];
}
```

```
const EDUCATIONAL_CONTENT: EducationalTopic[] = [
  {
    id: 'about-3iatlas',
    title: 'About 3I/ATLAS',
    icon: '☄',
    sections: [
      {
        heading: 'What is 3I/ATLAS?',
        body: `3I/ATLAS (C/2025 N1) is the third confirmed
interstellar object to visit our solar system, discovered on
July 1, 2025, by the ATLAS telescope in Chile. This ancient
cosmic wanderer is over 7 billion years old—older than our
entire solar system!

Unlike asteroids or comets born around our Sun, 3I/ATLAS formed
around a different star in the Milky Way's thick disk and
has been traveling through interstellar space for billions
of years before passing through our neighborhood.`,
        image: '/images/3iatlas-artist.jpg',
        links: [
          { text: 'NASA Science: 3I/ATLAS', url:
'https://science.nasa.gov/solar-system/comets/3i-atlas/' },
          { text: 'Discovery Paper', url: 'https://arxiv.org/...' }
        ]
      },
      {
        heading: 'Key Facts',
        body: `• **Age:** Over 7 billion years old
• **Size:** 440 meters to 5.6 kilometers
• **Speed:** ~137,000 mph (221,000 km/h)
• **Origin:** Milky Way thick disk
• **Perihelion:** October 29, 2025 at 1.356 AU
• **Orbit Type:** Hyperbolic (will never return)
• **Discovery:** ATLAS telescope, Chile
• **Visibility:** Magnitude ~15 (telescope only)`,
      },
      {
        heading: 'Why is it Important?',
        body: `3I/ATLAS is like a cosmic time capsule! It carries
material from the early days of our galaxy, providing
scientists with a rare opportunity to study:

• **Ancient star systems:** Learn about stellar environments
billions of years ago
• **Interstellar chemistry:** Compare comet composition across star
systems
• **Galaxy evolution:** Understand how our Milky Way has changed
over time
• **Comet formation:** Test theories about how comets form
universally

This is only the third time we've had a chance to study material
from outside our solar system, making 3I/ATLAS incredibly
valuable for science.`,
        image: '/images/galaxy-thick-disk.jpg'
      }
    ]
  },

  {
    id: 'interstellar-visitors',
    title: 'Interstellar Visitors',
```

```
    icon: '☄',
    sections: [
      {
        heading: '1I/‘Oumuamua (2017)',
        body: `The first confirmed interstellar object discovered in
        our solar system.

**Key Facts:**
• **Shape:** Elongated (cigar or pancake)
• **Size:** ~100-400 meters
• **Unique:** No visible coma or tail
• **Mystery:** Accelerated away from Sun (non-gravitational force)

‘Oumuamua revolutionized our understanding of interstellar objects
        and sparked intense scientific debate about its composition
        and origin.`,
        image: '/images/oumuamua.jpg',
        links: [
          { text: '‘Oumuamua Facts', url:
        'https://science.nasa.gov/solar-system/asteroids/oumuamua/'
        }
        ]
      },
      {
        heading: '2I/Borisov (2019)',
        body: `The first confirmed interstellar comet—and it looked
        remarkably like comets from our own solar system!

**Key Facts:**
• **Type:** Active comet with visible coma and tail
• **Size:** ~400-meter nucleus
• **Activity:** Strong gas and dust emission
• **Composition:** Similar to solar system comets
• **Color:** Reddish (organic-rich surface)

Borisov proved that comets form similarly across different star
        systems, validating theories of comet formation.`,
        image: '/images/borisov.jpg',
        links: [
          { text: 'Borisov Mission', url:
        'https://science.nasa.gov/solar-system/comets/2i-borisov/' }
        ]
      },
      {
        heading: '3I/ATLAS (2025) - YOU ARE HERE!',
        body: `The third interstellar visitor and the OLDEST object
        ever directly observed!

**What Makes It Special:**
• **Age:** >7 billion years (older than the Sun!)
• **Size:** Largest interstellar object discovered
• **Activity:** Strong coma and tail development
• **Origin:** Milky Way thick disk (ancient stellar population)
• **Best studied:** Early detection + modern instruments

3I/ATLAS represents our best opportunity yet to study interstellar
        material in detail, thanks to powerful telescopes like JWST
        being ready when it arrived.`,
      },
      {
        heading: 'How Many Are Out There?',
```

```
        body: `Astronomers estimate that **dozens** of interstellar
        objects pass through the solar system every year! But most
        are too faint and fast to detect.

**Detection Challenges:**
• Moving fast (hyperbolic trajectories)
• Usually far from Earth
• Often small and dim
• Short observation windows

**The Future:**
With next-generation survey telescopes like the Vera Rubin
        Observatory (Legacy Survey of Space and Time), we expect to
        discover many more interstellar visitors in the coming
        years.`,
      }
    ]
  },

  {
    id: 'orbital-mechanics',
    title: 'How Orbits Work',
    icon: '🔄',
    sections: [
      {
        heading: 'Types of Orbits',
        body: `Objects in space follow different types of paths
        depending on their speed and distance from the Sun:

**1. Circular Orbit (e = 0)**
• Perfect circle around the Sun
• Rare in nature (most orbits are slightly elliptical)

**2. Elliptical Orbit (0 < e < 1)**
• Oval-shaped path
• Most planets and comets in our solar system
• Bound to the Sun forever

**3. Parabolic Orbit (e = 1)**
• Exactly at escape velocity
• Theoretical boundary case

**4. Hyperbolic Orbit (e > 1)** ← 3I/ATLAS IS HERE!
• Open-ended curve
• Moving faster than escape velocity
• Will leave the solar system
• Indicates origin outside our solar system`,
        image: '/images/orbit-types.png'
      },
      {
        heading: 'What is Eccentricity?',
        body: `**Eccentricity (e)** measures how "stretched" an
        orbit is:

• **e = 0:** Perfect circle
• **e = 0.1:** Nearly circular (Earth is 0.017)
• **e = 0.5:** Moderately elliptical
• **e = 0.9:** Very elongated (some comets)
• **e = 1.0:** Parabola (escape velocity)
• **e > 1.0:** Hyperbola (interstellar!)
```

```
**3I/ATLAS has e = 6.14!**
This extremely high eccentricity mathematically proves it came from
        interstellar space and will return there.`,
      },
      {
        heading: 'Perihelion and Aphelion',
        body: `**Perihelion:** Closest point to the Sun
• For 3I/ATLAS: **1.356 AU** (October 29, 2025)
• Just inside Mars's orbit (1.52 AU)
• Maximum solar heating and comet activity

**Aphelion:** Farthest point from the Sun
• For elliptical orbits only
• 3I/ATLAS has no aphelion—it never loops back!

**Why Does This Matter?**
At perihelion, the Sun's heat vaporizes ice on the comet's surface,
        creating the spectacular coma and tail. This is when
        3I/ATLAS is most active and brightest!`,
        image: '/images/perihelion-diagram.png'
      },
      {
        heading: 'Astronomical Units (AU)',
        body: `An **Astronomical Unit (AU)** is the average distance
        between Earth and the Sun:

**1 AU = 149,597,870.7 kilometers (93 million miles)**

**Quick Reference:**
• Mercury: 0.39 AU
• Venus: 0.72 AU
• Earth: 1.0 AU (by definition)
• Mars: 1.52 AU
• Jupiter: 5.2 AU
• Neptune: 30 AU
• Pluto: 39.5 AU (average)

**3I/ATLAS perihelion: 1.356 AU**
Just slightly closer to the Sun than Mars!

Using AU makes it easier to think about distances in the solar
        system without dealing with huge numbers in kilometers or
        miles.`,
      }
    ]
  },

  {
    id: 'observing',
    title: 'How to Observe',
    icon: '🔭',
    sections: [
      {
        heading: 'Can I See 3I/ATLAS?',
        body: `**Short Answer:** Only with a good telescope.

At its brightest (perihelion), 3I/ATLAS reaches magnitude **~15**,
        which is:
• Too faint for naked eye (need mag 6 or brighter)
```

```
• Too faint for binoculars (need mag 10 or brighter)
• **Visible with:** 8-inch or larger telescopes

**What You'll See:**
• Fuzzy greenish patch (coma)
• Possibly a faint tail in long-exposure photos
• Moves noticeably against background stars over hours

**Best Viewing Period:** October 15-31, 2025`,
      },
      {
        heading: 'Equipment Needed',
        body: `**Minimum Equipment:**
• 8-inch (200mm) aperture telescope
• Dark sky location (Bortle 4 or darker)
• Star charts or planetarium software
• Patience!

**Photography Equipment:**
• DSLR or astronomy camera
• Telescope or telephoto lens
• Tracking mount (to follow stars)
• Long exposures (30-60 seconds)

**Software:**
• Stellarium (free planetarium)
• SkySafari (mobile app)
• NASA Horizons (position data)`,
        links: [
          { text: 'Stellarium', url: 'https://stellarium.org/' },
          { text: 'NASA Horizons', url:
        'https://ssd.jpl.nasa.gov/horizons/' }
        ]
      },
      {
        heading: 'When and Where to Look',
        body: `**October 2025 Viewing Guide:**

**Constellation:** Moving through Sagittarius/Scorpius
**Best Time:** Late evening, looking south
**Peak Dates:** October 15-31, 2025

**Finding It:**
1. Use planetarium software to get current position
2. Locate the constellation Sagittarius (the Teapot)
3. Use star charts to navigate to 3I/ATLAS's position
4. Look for fuzzy patch that moves relative to stars

**Tips:**
• Allow 20-30 minutes for eyes to adapt to darkness
• Use averted vision (look slightly to the side)
• Take notes on position to confirm motion
• Share your observation with astronomy communities!`,
      }
    ]
  },

  {
```

```
    id: 'science',
    title: 'The Science',
    icon: '🔬',
    sections: [
      {
        heading: 'What Are We Learning?',
        body: `**Active Research Questions:**

**1. Chemical Composition**
• What molecules are present in the coma?
• How does it compare to solar system comets?
• Are there unusual isotopic ratios?

**2. Age and Origin**
• How long has it been traveling through space?
• What star system did it come from?
• How has interstellar radiation affected it?

**3. Structure**
• What is the nucleus shape and size?
• How fast is it rotating?
• What is the surface composition?

**4. Activity**
• How does outgassing change as it approaches the Sun?
• What triggers the coma and tail formation?
• How does it compare to "normal" comet activity?`,
      },
      {
        heading: 'Instruments Observing 3I/ATLAS',
        body: `**Space Telescopes:**
• **James Webb Space Telescope (JWST)**
  - Infrared spectroscopy
  - Chemical composition analysis
  - Temperature mapping

• **Hubble Space Telescope (HST)**
  - UV spectroscopy
  - High-resolution imaging
  - Coma structure

**Ground Observatories:**
• **Keck Observatory** (Hawaii): Near-infrared spectroscopy
• **Very Large Telescope** (Chile): Optical monitoring
• **ALMA** (Chile): Millimeter-wave gas emission
• **ATLAS** (Hawaii & Chile): Discovery and monitoring

**What They're Measuring:**
• Gas and dust production rates
• Molecular abundances (H₂O, CO, CO₂, etc.)
• Nucleus size, shape, rotation
• Orbit refinement`,
        image: '/images/telescopes-observing.jpg'
      },
      {
        heading: 'Significance for Astrobiology',
```

```
        body: `Interstellar objects like 3I/ATLAS are crucial for
        understanding the **distribution of organic molecules** in
        the galaxy:

**Why It Matters:**
• Comets may have delivered water and organics to early Earth
• Interstellar comets spread material between star systems
• Complex molecules (prebiotic chemistry) could be common

**What We're Looking For:**
• Amino acids (building blocks of proteins)
• Sugars (building blocks of DNA/RNA)
• Alcohols and aldehydes (organic chemistry)
• Water ice and its isotopes

**The Big Question:**
Are the chemical ingredients for life common throughout the galaxy?
        3I/ATLAS helps answer this!`,
      }
    ]
  },

  {
    id: 'fun-facts',
    title: 'Fun Facts',
    icon: '✨',
    sections: [
      {
        heading: 'Mind-Blowing Comparisons',
        body: `**Age:**
• 3I/ATLAS: >7 billion years old
• The Sun: 4.6 billion years old
• Earth: 4.5 billion years old
• Modern humans: 300,000 years old

**Speed:**
• 3I/ATLAS: 137,000 mph (221,000 km/h)
• Space Shuttle: 17,500 mph
• Fastest car: 284 mph
• Usain Bolt: 28 mph

**Distance:**
• 3I/ATLAS perihelion: 170 million miles from Earth
• Moon: 239,000 miles
• ISS: 250 miles
• Commercial flight: 7 miles altitude`,
      },
      {
        heading: 'Cosmic Journey',
        body: `**Where Has 3I/ATLAS Been?**

Imagine: This comet formed over 7 billion years ago, when our solar
        system didn't even exist yet!

• Formed around an ancient star in the Milky Way thick disk
• Ejected from its home system (maybe by a passing star or planet)
• Traveled through the galaxy for billions of years
• Survived countless cosmic ray impacts
• Finally encountered our solar system by chance
```

```
**After its visit:**
• Will accelerate back to interstellar space
• Will travel for millions/billions more years
• Might eventually encounter another star system
• Will never return to our solar system

We are witnessing a brief moment in its epic cosmic journey!`,
      },
      {
        heading: 'Name Origins',
        body: `**Why "3I"?**
• "I" stands for "Interstellar"
• "3" because it's the third confirmed interstellar object
• Previous: 1I/'Oumuamua and 2I/Borisov

**Why "ATLAS"?**
• Named after the telescope that discovered it
• ATLAS = **A**steroid **T**errestrial-impact **L**ast **A**lert
      **S**ystem
• Located in Hawaii and Chile
• Designed to detect potentially hazardous asteroids

**Full Designation: C/2025 N1**
• "C" = Comet
• "2025" = Year of discovery
• "N" = Second half of July (N = 14th letter → July 14-31)
• "1" = First object discovered in that half-month`,
      }
    ]
  }
];

export function EducationalPanel({
  isOpen,
  onClose
}: {
  isOpen: boolean;
  onClose: () => void;
}) {
  const [activeTopicId, setActiveTopicId] = useState('about-
      3iatlas');

  const activeTopic = EDUCATIONAL_CONTENT.find(t => t.id ===
      activeTopicId);

  return (
    <AnimatePresence>
      {isOpen && (
        <motion.div
          initial={{ x: '100%' }}
          animate={{ x: 0 }}
          exit={{ x: '100%' }}
          transition={{ type: 'spring', damping: 25, stiffness: 200
        }}
          className="fixed right-0 top-0 h-full w-[500px] bg-
        gradient-to-br from-gray-900 via-black to-gray-900 border-l
        border-white/20 z-50 shadow-2xl overflow-hidden"
        >
          {/* Header */}
```

```
  <div className="bg-black/50 backdrop-blur-md border-b
border-white/20 p-4 flex items-center justify-between">
    <h2 className="text-white text-xl font-bold">Learn About
3I/ATLAS</h2>
    <button
      onClick={onClose}
      className="text-white/70 hover:text-white text-2xl
transition-colors"
      aria-label="Close"
    >
      ×
    </button>
  </div>

  {/* Topic Tabs */}
  <div className="bg-black/30 border-b border-white/10 px-4
py-3 flex gap-2 overflow-x-auto">
    {EDUCATIONAL_CONTENT.map(topic => (
      <button
        key={topic.id}
        onClick={() => setActiveTopicId(topic.id)}
        className={`
          flex items-center gap-2 px-4 py-2 rounded-lg
whitespace-nowrap transition-all
          ${activeTopicId === topic.id
            ? 'bg-green-600 text-white shadow-lg'
            : 'bg-white/10 text-white/70 hover:bg-white/20
hover:text-white'}
        `}
      >
        <span className="text-xl">{topic.icon}</span>
        <span className="text-sm font-medium">{topic.title}
</span>
      </button>
    ))}
  </div>

  {/* Content */}
  <div className="h-[calc(100vh-140px)] overflow-y-auto p-6
space-y-6">
    {activeTopic?.sections.map((section, i) => (
      <motion.div
        key={i}
        initial={{ opacity: 0, y: 20 }}
        animate={{ opacity: 1, y: 0 }}
        transition={{ delay: i * 0.1 }}
        className="space-y-3"
      >
        <h3 className="text-2xl font-bold text-white border-
b border-white/20 pb-2">
          {section.heading}
        </h3>

        {section.image && (
          <img
            src={section.image}
            alt={section.heading}
            className="w-full rounded-lg shadow-xl"
          />
        )}
```

```
                {section.video && (
                  <video
                    src={section.video}
                    controls
                    className="w-full rounded-lg shadow-xl"
                  />
                )}

                <div className="text-white/90 leading-relaxed
        whitespace-pre-line">
                  {section.body}
                </div>

                {section.links && (
                  <div className="space-y-2 pt-2">
                    <div className="text-white/60 text-sm font-
        semibold">Learn More:</div>
                    {section.links.map((link, j) => (
                      <a
                        key={j}
                        href={link.url}
                        target="_blank"
                        rel="noopener noreferrer"
                        className="block text-blue-400 hover:text-
        blue-300 underline text-sm"
                      >
                        {link.text} →
                      </a>
                    ))}
                  </div>
                )}
              </motion.div>
            ))}
          </div>
        </motion.div>
      )}
    </AnimatePresence>
  );
}

// Trigger button for educational panel
export function EducationalPanelButton({ onClick }: { onClick: () =>
        void }) {
  return (
    <button
      onClick={onClick}
      className="fixed top-4 left-4 bg-blue-600 hover:bg-blue-700
        text-white px-4 py-2 rounded-lg shadow-lg flex items-center
        gap-2 transition-all hover:scale-105"
      aria-label="Open educational panel"
    >
      <span className="text-xl">📘</span>
      <span className="font-medium">Learn More</span>
    </button>
  );
}
```

Integration:

```
// In main component:
const [eduPanelOpen, setEduPanelOpen] = useState(false);
```

```jsx
// In JSX:
<>
  <EducationalPanelButton onClick={() => setEduPanelOpen(true)} />
  <EducationalPanel
    isOpen={eduPanelOpen}
    onClose={() => setEduPanelOpen(false)}
  />
</>
```

Test Plan

1. **Functionality:**
   - Click “Learn More” button opens panel
   - Panel slides in from right smoothly
   - Tab switching works without lag
   - Close button dismisses panel
   - Panel scrolls smoothly
2. **Content:**
   - All text is readable and accurate
   - Images load correctly
   - Links open in new tabs
   - Formatting (bold, bullets) renders correctly
3. **Responsive:**
   - Panel width appropriate on different screens
   - Content doesn’t overflow
   - Touch scrolling works on mobile
4. **Accessibility:**
   - Keyboard navigation works
   - Screen readers can read content
   - Focus management when opening/closing

---

7. Integration Plan

7.1 New Trajectory Data Integration

Source: `/home/ubuntu/3iatlas_trajectory_data.json`

Data Structure:

```jsonc
{
  "atlas": [
    {
      "jd": 2460857.5,
      "date": "2025-07-01 00:00:00",
      "position": [-4.413, 0.074, -0.846],  // AU
      "velocity": [0.0349, 0.00047, -0.0054]  // AU/day
    }
  ]
}
```

Key Differences from Current Data: 1. **Arrays instead of objects:** Position/velocity are flat arrays, not `{x, y, z}` or `{vx, vy, vz}` 2. **More data points:** Hourly resolution (2900+ points) vs 6-hourly (121 points) 3. **Full trajectory:** July 1 - October 31, 2025 4. **Single object:** Only 3I/ATLAS data (no planets)

Integration Steps:

1. **Parse New Data Format:**

```
// lib/trajectory-parser.ts
export interface TrajectoryDataPoint {
  jd: number;
  date: string;
  position: [number, number, number];  // [x, y, z] in AU
  velocity: [number, number, number];  // [vx, vy, vz] in AU/day
}

export interface TrajectoryData {
  atlas: TrajectoryDataPoint[];
}

export function parseTrajectoryData(rawData: TrajectoryData):
        VectorData[] {
  return rawData.atlas.map(point => ({
    jd: point.jd,
    date: point.date,
    position: {
      x: point.position[0],
      y: point.position[1],
      z: point.position[2]
    },
    velocity: {
      vx: point.velocity[0],
      vy: point.velocity[1],
      vz: point.velocity[2]
    }
  }));
}
```

2. **Load New Data:**

```
// In Atlas3DTrackerEnhanced.tsx or similar:
import trajectoryData from
        '/home/ubuntu/3iatlas_trajectory_data.json';
import { parseTrajectoryData } from '@/lib/trajectory-parser';

// In component:
const atlasTrajectory = useMemo(() => {
  return parseTrajectoryData(trajectoryData);
}, []);

// Use atlasTrajectory instead of fetching from Horizons
setSolarSystemData({
  atlas: atlasTrajectory,
  earth: /* from existing trajectory.json */,
  mars: /* from existing trajectory.json */
});
```

3. **Fallback Strategy:**

```
async function loadAtlasData() {
  try {
    // Try to use pre-calculated high-resolution data
    const highResData = parseTrajectoryData(trajectoryData);
    if (highResData.length > 0) {
```

```
      console.log(`[Atlas] Using high-resolution trajectory data
        (${highResData.length} points)`);
      return highResData;
    }
  } catch (error) {
    console.warn('[Atlas] Failed to load high-res data, falling back
        to Horizons API');
  }

  // Fallback to Horizons API
  try {
    const horizonsData = await getCached3IAtlasVectors(startDate,
        endDate, stepSize);
    return horizonsData;
  } catch (error) {
    console.warn('[Atlas] Horizons API failed, using orbital
        calculations');
  }

  // Final fallback: calculate from orbital elements
  return generate3IAtlasVectors(startDate, endDate, stepHours);
}
```

4. **Performance Optimization:**

```
// With 2900+ data points, we need smart decimation for performance

function decimateTrajectory(data: VectorData[], targetPoints:
        number): VectorData[] {
  if (data.length <= targetPoints) return data;

  const step = Math.floor(data.length / targetPoints);
  const decimated: VectorData[] = [];

  for (let i = 0; i < data.length; i += step) {
    decimated.push(data[i]);
  }

  // Always include last point
  if (decimated[decimated.length - 1] !== data[data.length - 1]) {
    decimated.push(data[data.length - 1]);
  }

  return decimated;
}

// Usage:
const displayData = decimateTrajectory(atlasTrajectory, 500);  //
        500 points for smooth animation
```

7.2 Component Integration Order

Phase 1: Core Fixes (Week 1) 1. Fix playback controls (speed, reset, scrubber) 2. Integrate new trajectory data 3. Test existing camera views

Phase 2: Visual Enhancements (Week 2) 1. Implement enhanced comet visuals (coma, tail) 2. Improve camera follow system 3. Add new camera presets

Phase 3: Interactive Features (Week 3) 1. Add milestone markers 2. Implement educational panel 3. Mobile responsiveness improvements

Phase 4: Polish & Testing (Week 4) 1. Performance optimization 2. Cross-browser testing 3. Accessibility audit 4. User testing feedback

---

8. Performance Optimization Strategy

8.1 Current Performance Profile

Baseline Measurements: - FPS Target: 60 fps - Load Time: ~3-5 seconds (initial data fetch) - Memory Usage: ~100-150 MB - Animation Smoothness: Generally good with occasional drops

Bottlenecks Identified: 1. Large trajectory datasets (2900+ points) 2. Particle systems (coma = 5000 particles) 3. Trail rendering (30-50 points) 4. Multiple dynamic objects 5. Label projection calculations (every frame)

8.2 Optimization Techniques

8.2.1 Data Decimation

```
// Adaptive decimation based on device capability
function getOptimalDataResolution(deviceType: string, dataLength:
        number): number {
  const targets = {
    mobile: 250,   // ~250 points for smooth mobile experience
    tablet: 500,   // ~500 points for tablets
    desktop: 1000  // ~1000 points for desktops
  };

  const target = targets[deviceType as keyof typeof targets] ||
        targets.desktop;
  return Math.min(target, dataLength);
}

// Smart decimation preserving key features
function intelligentDecimation(data: VectorData[], targetCount:
        number): VectorData[] {
  if (data.length <= targetCount) return data;

  // Calculate importance score for each point
  const importance = data.map((point, i) => {
    if (i === 0 || i === data.length - 1) return Infinity; // Always
        keep endpoints

    let score = 0;

    // Importance based on velocity change (acceleration)
    if (i > 0 && i < data.length - 1) {
      const velBefore = data[i - 1].velocity;
      const velAfter = data[i + 1].velocity;
      const velChange = Math.abs(velAfter.vx - velBefore.vx) +
                        Math.abs(velAfter.vy - velBefore.vy) +
                        Math.abs(velAfter.vz - velBefore.vz);
      score += velChange * 100;
```

```
    }

    // Importance based on distance from sun (perihelion more
        important)
    const distFromSun = Math.sqrt(
      point.position.x ** 2 + point.position.y ** 2 +
        point.position.z ** 2
    );
    score += 1 / (distFromSun + 0.1);  // Higher score when closer

    return score;
  });

  // Sort indices by importance, keep top N
  const sortedIndices = importance
    .map((score, i) => ({ score, i }))
    .sort((a, b) => b.score - a.score)
    .slice(0, targetCount)
    .map(item => item.i)
    .sort((a, b) => a - b);  // Re-sort chronologically

  return sortedIndices.map(i => data[i]);
}
```

8.2.2 Level of Detail (LOD)

```
// Adjust visual quality based on camera distance
function getLODLevel(cameraDistance: number): 'high' | 'medium' |
        'low' {
  if (cameraDistance < 5) return 'high';
  if (cameraDistance < 15) return 'medium';
  return 'low';
}

function applyLOD(scene: THREE.Scene, camera: THREE.Camera) {
  const objects = scene.children;

  for (const obj of objects) {
    const distance = camera.position.distanceTo(obj.position);
    const lod = getLODLevel(distance);

    if (obj.userData.lodEnabled) {
      switch (lod) {
        case 'high':
          // Full detail
          if (obj instanceof THREE.Mesh) {
            obj.material.needsUpdate = true;
            obj.geometry = obj.userData.highResGeometry;
          }
          break;
        case 'medium':
          // Medium detail
          if (obj instanceof THREE.Mesh) {
            obj.geometry = obj.userData.mediumResGeometry;
          }
          break;
        case 'low':
          // Low detail or cull completely
          if (distance > 50) {
```

```
            obj.visible = false;
          } else if (obj instanceof THREE.Mesh) {
            obj.geometry = obj.userData.lowResGeometry;
          }
          break;
      }
    }
  }
}
```

8.2.3 Object Pooling

```
// Reuse Three.js objects instead of creating new ones
class ObjectPool<T> {
  private pool: T[] = [];
  private active: Set<T> = new Set();

  constructor(
    private factory: () => T,
    private reset: (obj: T) => void,
    initialSize: number = 10
  ) {
    for (let i = 0; i < initialSize; i++) {
      this.pool.push(factory());
    }
  }

  acquire(): T {
    let obj = this.pool.pop();
    if (!obj) {
      obj = this.factory();
    }
    this.active.add(obj);
    return obj;
  }

  release(obj: T) {
    if (this.active.has(obj)) {
      this.reset(obj);
      this.active.delete(obj);
      this.pool.push(obj);
    }
  }

  clear() {
    this.pool = [];
    this.active.clear();
  }
}

// Usage for particles:
const particlePool = new ObjectPool(
  () => new THREE.Vector3(),
  (vec) => vec.set(0, 0, 0),
  1000
);
```

8.2.4 Frustum Culling

```
// Don't render objects outside camera view
function frustumCull(scene: THREE.Scene, camera: THREE.Camera) {
  const frustum = new THREE.Frustum();
  const matrix = new THREE.Matrix4().multiplyMatrices(
    camera.projectionMatrix,
    camera.matrixWorldInverse
  );
  frustum.setFromProjectionMatrix(matrix);

  scene.traverse((obj) => {
    if (obj instanceof THREE.Mesh) {
      // Check if object's bounding sphere is in frustum
      obj.geometry.computeBoundingSphere();
      const sphere = obj.geometry.boundingSphere;

      if (sphere) {
        const worldSphere =
        sphere.clone().applyMatrix4(obj.matrixWorld);
        obj.visible = frustum.intersectsSphere(worldSphere);
      }
    }
  });
}
```

8.2.5 Web Workers for Heavy Calculations

```
// trajectory-worker.ts
self.addEventListener('message', (e) => {
  const { type, data } = e.data;

  switch (type) {
    case 'decimate':
      const decimated = intelligentDecimation(data.trajectory,
        data.targetCount);
      self.postMessage({ type: 'decimated', data: decimated });
      break;

    case 'interpolate':
      const interpolated = interpolateTrajectory(data.trajectory,
        data.frame, data.t);
      self.postMessage({ type: 'interpolated', data: interpolated
        });
      break;
  }
});

// Main thread usage:
const worker = new Worker(new URL('./trajectory-worker.ts',
        import.meta.url));

worker.postMessage({
  type: 'decimate',
  data: { trajectory: fullData, targetCount: 500 }
});

worker.addEventListener('message', (e) => {
  if (e.data.type === 'decimated') {
    setDecimatedData(e.data.data);
  }
});
```

8.2.6 Lazy Loading & Code Splitting

```
// Dynamic imports for heavy components
const CometVisuals = dynamic(() =>
        import('@/components/CometVisuals'), {
  ssr: false,
  loading: () => <LoadingPlaceholder />
});

const EducationalPanel = dynamic(() =>
        import('@/components/EducationalPanel'), {
  ssr: false
});

// Load only when needed:
const [showEducational, setShowEducational] = useState(false);

{showEducational && <EducationalPanel ... />}
```

8.3 Performance Monitoring

```
// Real-time performance monitor
class PerformanceMonitor {
  private frameCount = 0;
  private lastTime = performance.now();
  private fps = 60;
  private frameDrops = 0;

  update() {
    this.frameCount++;
    const now = performance.now();
    const delta = now - this.lastTime;

    if (delta >= 1000) {
      this.fps = (this.frameCount * 1000) / delta;

      if (this.fps < 45) {
        this.frameDrops++;
      }

      this.frameCount = 0;
      this.lastTime = now;

      // Report to analytics
      this.report();
    }
  }

  report() {
    console.log(`FPS: ${this.fps.toFixed(1)}, Drops:
        ${this.frameDrops}`);

    // Update global health object
    if (typeof window !== 'undefined') {
      (window as any).__AE_HEALTH__ = {
        fps: this.fps,
        frameDrops: this.frameDrops,
        memoryUsage: this.getMemoryUsage(),
        timestamp: Date.now()
```

```
      };
    }
  }

  getMemoryUsage() {
    if ('memory' in performance) {
      return (performance as any).memory.usedJSHeapSize / 1048576;
        // MB
    }
    return 0;
  }
}

// Usage in animation loop:
const monitor = new PerformanceMonitor();

function animate() {
  requestAnimationFrame(animate);

  monitor.update();

  // ... rest of animation ...
}
```

---

9. Mobile Responsiveness

9.1 Responsive Breakpoints

```
// config/responsive.ts
export const BREAKPOINTS = {
  mobile: 480,
  tablet: 768,
  laptop: 1024,
  desktop: 1440,
  wide: 1920
} as const;

export type DeviceType = 'mobile' | 'tablet' | 'laptop' | 'desktop'
        | 'wide';

export function getDeviceType(width: number): DeviceType {
  if (width < BREAKPOINTS.mobile) return 'mobile';
  if (width < BREAKPOINTS.tablet) return 'tablet';
  if (width < BREAKPOINTS.laptop) return 'laptop';
  if (width < BREAKPOINTS.desktop) return 'desktop';
  return 'wide';
}

export function useDeviceType(): DeviceType {
  const [deviceType, setDeviceType] = useState<DeviceType>
        ('desktop');

  useEffect(() => {
    function handleResize() {
      setDeviceType(getDeviceType(window.innerWidth));
    }
```

```
    handleResize();
    window.addEventListener('resize', handleResize);
    return () => window.removeEventListener('resize', handleResize);
  }, []);

  return deviceType;
}
```

9.2 Adaptive Quality Settings

```
// config/device-quality.ts
export interface QualitySettings {
  starCount: number;
  geometryDetail: number;
  shadowMapSize: number;
  pixelRatio: number;
  trailLength: number;
  particleCount: number;
  enablePostProcessing: boolean;
}

export const QUALITY_PRESETS: Record<DeviceType, QualitySettings> =
        {
  mobile: {
    starCount: 500,
    geometryDetail: 8,
    shadowMapSize: 256,
    pixelRatio: 1,
    trailLength: 10,
    particleCount: 500,
    enablePostProcessing: false
  },
  tablet: {
    starCount: 2000,
    geometryDetail: 16,
    shadowMapSize: 512,
    pixelRatio: 1.5,
    trailLength: 20,
    particleCount: 2000,
    enablePostProcessing: false
  },
  laptop: {
    starCount: 5000,
    geometryDetail: 32,
    shadowMapSize: 1024,
    pixelRatio: 1.5,
    trailLength: 30,
    particleCount: 5000,
    enablePostProcessing: true
  },
  desktop: {
    starCount: 10000,
    geometryDetail: 64,
    shadowMapSize: 2048,
    pixelRatio: 2,
    trailLength: 50,
    particleCount: 10000,
    enablePostProcessing: true
```

```
  },
  wide: {
    starCount: 20000,
    geometryDetail: 128,
    shadowMapSize: 4096,
    pixelRatio: 2,
    trailLength: 100,
    particleCount: 20000,
    enablePostProcessing: true
  }
};

export function getQualityForDevice(deviceType: DeviceType):
        QualitySettings {
  return QUALITY_PRESETS[deviceType];
}
```

9.3 Touch-Optimized Controls

```typescript // components/MobileControls.tsx export function MobileControls({ isPlaying, onPlayPause, speed, onSpeedChange, cameraView, onCameraChange }: MobileControlsProps) { const [showAdvanced, setShowAdvanced] = useState(false);

return (

```
  {/* Big play/pause button */}
  <button
    onClick={onPlayPause}
    className="w-full mb-3 py-4 bg-gradient-to-r from-green-600 to-
green-500 rounded-2xl text-white text-xl font-bold shadow-lg
active:scale-95 transition-transform flex items-center justify-
center gap-3"
  >
    {isPlaying ? '⏸' : '▶'}
    {isPlaying ? 'Pause' : 'Play'} Animation
  </button>

  {/* Speed selection */}

    Playback Speed

      {[1, 3, 6, 15].map(s => (
        <button
          key={s}
          onClick={() => onSpeedChange(s)}
          className={`
            py-3 rounded-xl font-medium transition-all
            ${speed === s
              ? 'bg-green-600 text-white shadow-lg scale-105'
              : 'bg-white/10 text-white/70 active:scale-95'}
          `}
        >
          {s}x
        </button>
      ))}

```
```

```
{/* Camera views */}
<div className="mb-3">
  <div className="text-white/60 text-xs mb-2">Camera View</div>
  <div className="grid grid-cols-3 gap-2">
    {[
      { id: 'default', icon: '⚽', label: 'Overview' },
      { id: 'followComet', icon: '☄', label
```