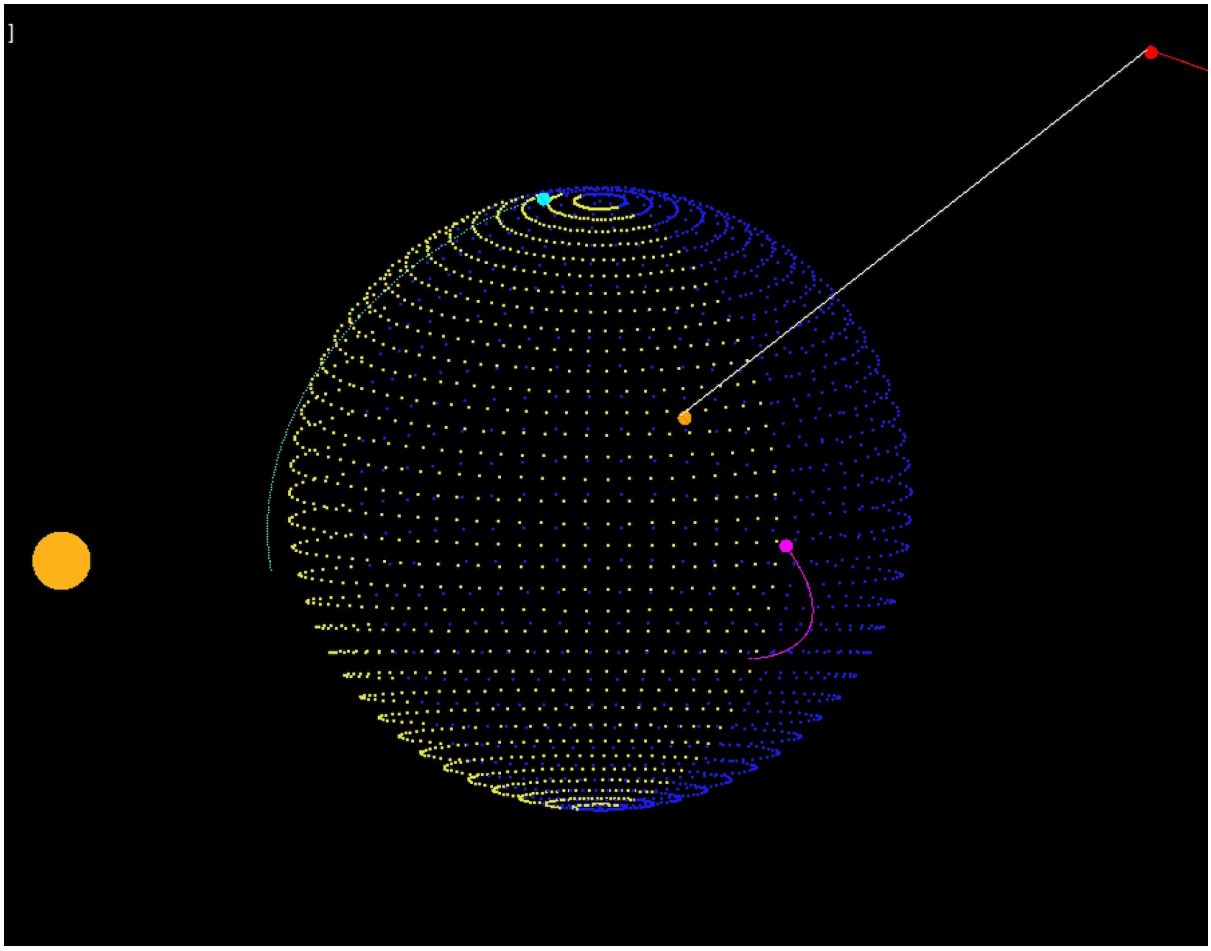
OrbitView: Building a Universe

A Computational Approach to Orbital Mechanics & 3D Visualization

Vivekjit Das
Department of Computer Science & Engineering
IIT Madras
Date: December 19 2025



A technical journey from Newton's Laws to Real-Time Satellite Tracking.

Abstract

This report documents the design and implementation of *OrbitView*, a high-fidelity orbital mechanics simulator built from scratch in C++. The project evolves through two distinct phases: a 2D Ground Track visualizer demonstrating the relative motion of satellites over a rotating Earth, and a full 3D Graphics Engine capable of rendering wireframe celestial bodies, calculating dynamic lighting, and parsing real-world NASA TLE (Two-Line Element) data.

Key technical achievements include the implementation of a custom physics engine using Newtonian gravitation, coordinate transformations between Earth-Centered Inertial (ECI) and Earth-Centered Earth-Fixed (ECEF) frames, and the derivation of state vectors from Keplerian orbital elements. The final system successfully tracks the International Space Station (ISS), Hubble Telescope, and GPS satellites in real-time, providing visibility analysis for a specific ground station in Agartala, India.

Contents

The Mission

1.1 Introduction

Space exploration relies heavily on the ability to predict and visualize the trajectories of objects in orbit. While analytical solutions exist for simple two-body problems, real-world applications require robust computational models that can handle dynamic updates and provide intuitive visual feedback.

This project was conceived to bridge the gap between abstract physics equations and visual reality. I didn't want to use a pre-made library. I wanted to build the engine myself—starting from the raw laws of physics. The goal was to build **OrbitView**, answering practical questions like: *"Where is the ISS right now?"* and *"Can I see it from my home in Agartala?"*

1.2 Project Scope

The project was executed in progressive stages:

1. **The Physics Core:** Developing a stable numerical integrator.
2. **2D Visualization:** Mapping 3D coordinates to a Mercator projection.
3. **3D Engine Construction:** Building a custom rendering pipeline.
4. **Real-Data Integration:** Parsing NORAD/NASA TLE data.

Theoretical Framework: The Physics

2.1 Newtonian Gravitation

The foundation of the engine is Newton's Law of Universal Gravitation. For a satellite of mass m and Earth of mass M, the force vector $\vec{F}$ is given by:

$$\vec{F} = -G\frac{Mm}{|\vec{r}|^3}\vec{r} \tag{2.1}$$

Since $\vec{F} = m\vec{a}$, the mass of the satellite cancels out, leaving us with the acceleration vector:

$$\vec{a}(t) = -\frac{GM}{|\vec{r}(t)|^3}\vec{r}(t) \tag{2.2}$$

Where:

- $G = 6.674 \times 10^{-11}\,\mathrm{m^3 kg^{-1} s^{-2}}$ (Gravitational Constant)
- $\vec{r}$ is the position vector from the Earth's center $(0, 0, 0)$.

2.2 Numerical Integration

To simulate this over time, we cannot simply use 'distance = speed * time' because speed is constantly changing due to gravity. We use **Numerical Integration** to step forward in time by Δt.

We utilized the **Semi-Implicit Euler Method** for its symplectic stability (it preserves energy better than standard Euler):

$$\vec{v}_{new} = \vec{v}_{old} + \vec{a}(\vec{r}_{old}) \cdot \Delta t \tag{2.3}$$

$$\vec{r}_{new} = \vec{r}_{old} + \vec{v}_{new} \cdot \Delta t \tag{2.4}$$

Mathematical Principle: Why Semi-Implicit?

In standard integration, energy errors accumulate, causing the satellite to spiral away from Earth. The Semi-Implicit method is "Symplectic," meaning it conserves the orbital geometry over long durations.

2.3 Keplerian Elements to Cartesian Vectors

Real-world satellite data comes in "Keplerian Elements" (Angles and Shapes), not vectors. To simulate the ISS, we must convert these 6 elements into $\vec{r}$ and $\vec{v}$.

The conversion requires solving **Kepler's Equation** iteratively:

$$M = E - e\sin(E) \tag{2.5}$$

Where M is Mean Anomaly and E is Eccentric Anomaly. We solve for E using the Newton-Raphson method. Once E is found, we calculate position in the orbital plane (P, Q):

$$P = a(\cos E - e) \tag{2.6}$$

$$Q = a\sqrt{1 - e^2}\sin E \tag{2.7}$$

Finally, we apply rotation matrices for the Longitude of Ascending Node (Ω), Inclination (i), and Argument of Perigee (ω) to transform this into 3D space.

Part 2: The Map (2D Phase)

3.1 The Coordinate Transform

A fundamental challenge in satellite tracking is the difference between reference frames.

- **ECI (Inertial):** The physics frame. The stars are fixed, Earth spins.
- **ECEF (Fixed):** The map frame. The Earth is static, stars spin.

To plot the satellite on a 2D map, we must subtract Earth's rotation:

$$\lambda_{map} = \text{atan2}(y, x) - (\omega_{Earth} \times t) \tag{3.1}$$

3.2 The Sine Wave Mystery

You've probably seen those "sine wave" curves in Mission Control movies. Why does a satellite moving in a straight circle look like a wave on a map?

- The satellite moves **East**.
- The Earth spins **East** underneath it.

This creates a relative motion. As the satellite goes over the poles, the Earth has rotated beneath it, causing the path to shift westward on every orbit.

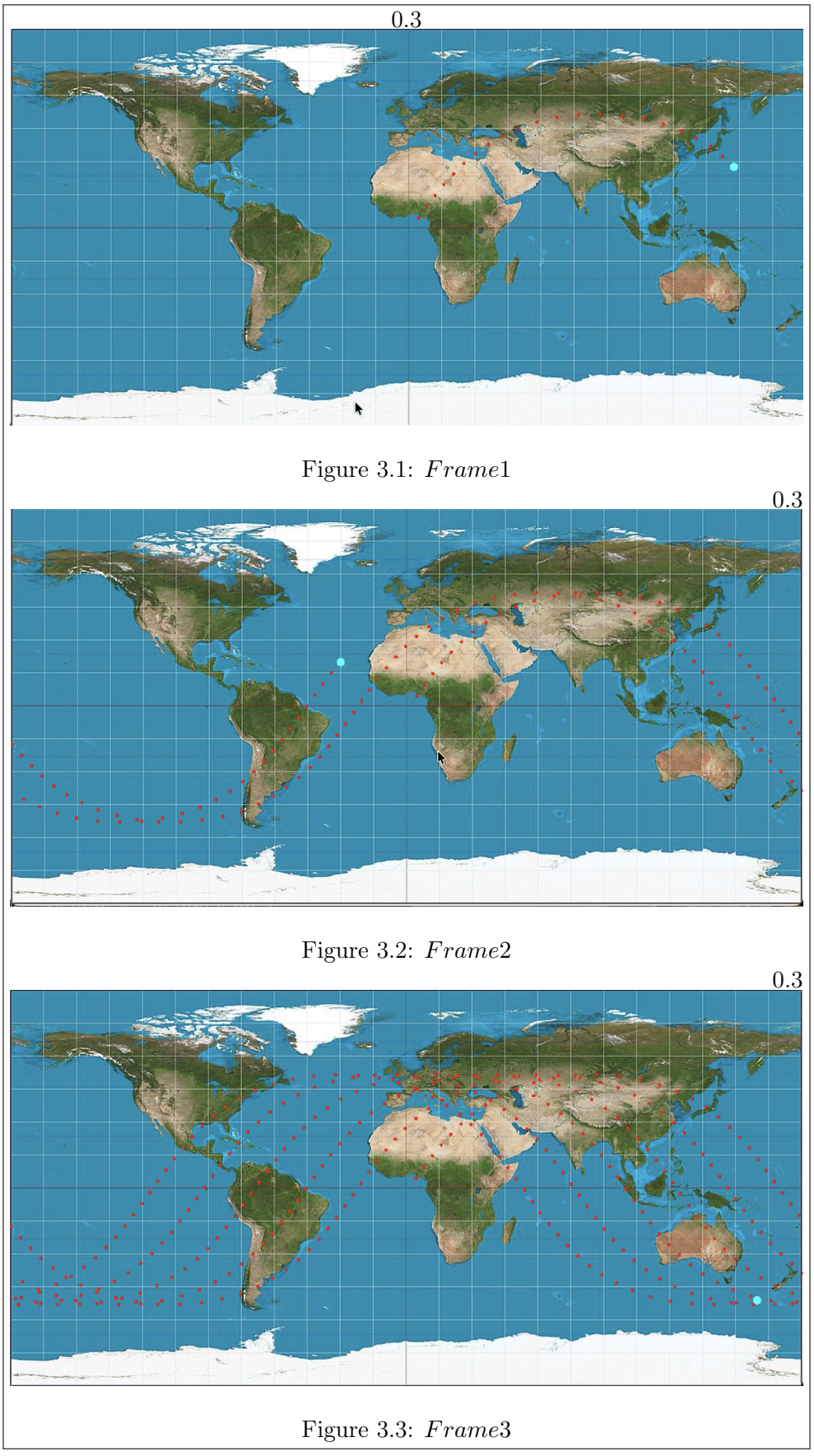


Figure 3.1: *Frame*1

Figure 3.2: *Frame*2

Figure 3.3: *Frame*3

Figure 3.4: Sinusoidal Trajectory Observation

Part 3: Going 3D (The Engine)

4.1 Building a Graphics Pipeline

Moving from 2D to 3D requires a way to project a 3D world onto a 2D monitor. I implemented a **Perspective Projection** algorithm manually.

The core formula divides the coordinate by its depth (z):

$$x_{screen} = \frac{x_{world}}{z_{dist} - z_{world}} \times \text{Scale} + \text{Center}_x \tag{4.1}$$

This creates the sensation of depth—objects moving further away shrink in size.

4.2 Wireframe Earth Lighting

I generated a mesh of the Earth using loops of Latitude and Longitude. To visualize the Sun, I implemented a directional lighting model using the **Dot Product**.

- **Sun Vector ($\vec{S}$):** Fixed at $(-1, 0, 0)$.
- **Normal Vector ($\vec{N}$):** The surface direction.

$$\text{Brightness} = \max(0, \vec{N} \cdot \vec{S}) \tag{4.2}$$

Points facing the sun are rendered Bright yellow ; points in shadow are Blue.

Part 4: Real Data Agartala Tracking

5.1 NASA's TLE Data

I wrote a custom parser to read Two-Line Elements (TLEs) from NASA. This allowed me to simulate the real **ISS**, **Hubble Telescope**, and **GPS Constellation**.

The Scale of Space

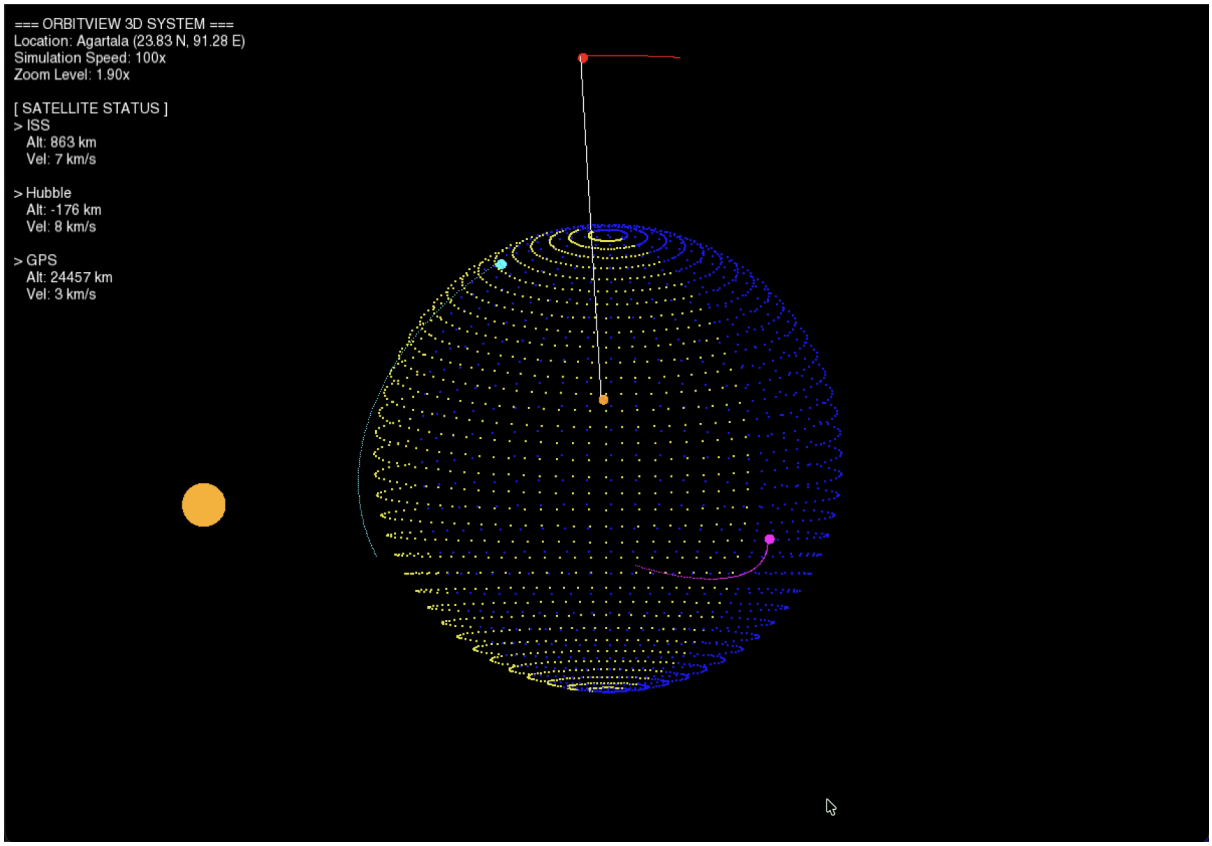
Visualizing the data revealed a shock: The ISS (400km) hugs the Earth tightly, while GPS satellites (20,000km) are incredibly far away. The scale difference is massive!

5.2 Visibility from Agartala

The final feature was personal. I added my home coordinates: **Agartala (23.83 N, 91.28 E)**. The system checks if a satellite is visible using vector math:

$$\text{Visible} \iff (\vec{r}_{sat} - \vec{r}_{city}) \cdot \vec{r}_{city} > 0 \tag{5.1}$$

When visible, a **White Laser Line** connects the city to the satellite in the 3D view.


=== ORBITVIEW 3D SYSTEM ===
Location: Agartala (23.83 N, 91.28 E)
Simulation Speed: 100x
Zoom Level: 1.90x
[SATELLITE STATUS]
> ISS
Alt: 863 km
Vel: 7 km/s
> Hubble
Alt: -176 km
Vel: 8 km/s
> GPS
Alt: 24457 km
Vel: 3 km/s

Conclusion

The OrbitView project successfully demonstrated the application of computational physics to solve complex spatial problems. By building the system from the ground up—starting with raw Newtonian equations and ending with a polished 3D renderer—I gained deep insights into both orbital mechanics and computer graphics.

The system now accurately:

- Propagates orbits using symplectic integration.
- Renders 3D geometry with perspective and lighting.
- Calculates real-time visibility for ground stations.

This project proves that with C++ and curiosity, we can hold the mechanics of the universe in our hands.