PURSUIT GUIDANCE LOGIC MISSILE SIMULATION

1. System Overview

This application simulates a missile interception scenario in 3D space. It integrates kinematic trajectory generation, pure pursuit guidance logic, and real-time 3D rendering of STL (stereolithography) models. The core objective was to visualize the spatial relationship between an evading fighter jet (F-15) and an interceptor missile.

2. Mathematical Framework

- Guidance Law (Pure Pursuit): The missile's velocity vector is updated every time step (dt) to point directly at the target's current position.

$$V_{missile} = V_{mag} \times \frac{P_{target} - P_{missile}}{\left|\left|P_{target} - P_{missile}\right|\right|}$$

Theory: This is the simplest guidance form. Unlike "Proportional Navigation" (which leads the target), Pure Pursuit results in a tail-chase scenario.

- 3D Rotation (Euler Angles):

To visualize the aircraft correctly, we cannot just place a dot; we must orient the 3D mesh to align with its velocity vector. We construct a rotation matrix R by combining Yaw (R_z) and Pitch (R_y):

$$R = R_z(\psi) \cdot R_y(\theta) \quad \psi = \arctan 2\left(v_y, v_x\right) \; \theta = \arctan 2\left(v_z, \sqrt{v_x^2 + v_y^2}\right)$$

This allows the static STL mesh (defined in a local coordinate system) to be transformed into the global "World" coordinate system.

- Collision Detection:

We utilize a discrete Euclidean distance check at every simulation step. A "kill" is registered if:

$$\left|\left|P_t - P_m\right|\right| < R_{kill}$$

Where R_{kill} is the proximity fuse radius (set to 50m).

3. Software Architecture & Optimization

- Pre-Calculation Strategy:

Instead of calculating physics during the animation loop (which can cause lag), we pre-calculate the entire state space$(x,y,z) for both entities into NumPy arrays. The animation loop acts purely as a "renderer" for this pre-computed data.

Benefit: Ensures deterministic physics and smooth 60 FPS playback.

- Mesh Simplification (Decimation): Raw STL files often contain 100k+ triangles, which is too heavy for Matplotlib's CPU-based renderer.
  - *Technique:* We implemented a "strided slicing" technique (vectors[::simplify_factor]) on the mesh data arrays.
  - *Result:* Reduces vertex count by ~ 80-90% while preserving the visual hull, allowing for real-time rotation without GPU acceleration.

- Orientation Normalization:

STL models from CAD often use different coordinate standards (e.g., Z-up vs Y-up). The code implements a basis transformation matrix to map the input mesh (Nose = -Z) to the simulation standard (Nose = +X).

4. Visual Logic (The "Splash" Effect)

To handle the intercept event, the system uses Frame Clamping. When the animation frame counter exceeds the pre-calculated intercept index:

1. The position data index is clamped to the intercept moment.
2. The alpha channel of the 3D polygon collection is set to 0 (simulating destruction).
3. The calculation of rotation matrices is suspended to conserve resources.

5. Future Improvements

- **Guidance:** Upgrade from Pure Pursuit to Proportional Navigation (PropNav) for more efficient intercepts.
- **Rendering:** Port the visualization to GPU-based libraries like **PyVista** or **Open3D** for handling higher-fidelity meshes without lag.