BMAGS Ballistic Trajectory Calculator

Generated by Doxygen 1.9.8

Chapter 1

BBTC: BMAGS' Ballistic Trajectory Calculator

1.1 Introduction

Welcome to the BBTC project—BMAGS' Ballistic Trajectory Calculator. This is a cross-platform, scientific simulation engine designed to model the external ballistics of projectiles with high fidelity.

BBTC is written entirely in C, and is designed with deep educational value in mind, offering:

- Advanced 6DOF modeling with quaternion rotation
- Support for all major drag models (G1, G2, G5, G6, G7, G8, GL, GS, GI, RA4)
- Realistic atmosphere and environmental effects
- Detailed .csv (spreadsheet format—comma-separated values) logging and simulation output
- Fully static memory model with no dynamic allocation (no `malloc`, `realloc`, `calloc`, etc.)
- Doxygen-formatted documentation

1.2 License

1.3 How to Cite

If you use BBTC in research, coursework, or documentation, please cite it as:

```
BMAGS. "BBTC: BMAGS' Ballistic Trajectory Calculator." https://github.com/BMAGS6/BBTC, 2025.
```

1.4 Documentation Structure

This PDF is structured in the following order:

1. 6DOF Physics Core
2. Command-Line Interface
3. Drag Models
4. Environmental Modeling
5. Quaternion Rotation
6. Source File Reference
7. Structure and Field Diagrams

1.5 Build & Compilation

- BBTC builds via a portable Makefile
- `make` builds the simulator
- `make doc-pdf` generates this PDF documentation
- `drag_model_chart_combined.png` is automatically embedded if placed at `BBTC/docs/drag_model_chart_combined.png`

1.6 Documentation Style Notes

- LaTeX rendering of math formulas is enabled
- DOT and SVG diagrams are embedded inline for clarity
- Field-level units are included for all major structures
- CLI options are fully documented and indexed
- PDF output uses pdfLaTeX and standard vector graphics
- No Unicode or emoji is used to ensure cross-platform rendering

1.7 Acknowledgments

- Ballistic model side profiles image inspired by Fr. Frog's drag model illustrations and classic ballistic modeling references;
- JBM Ballistics , for the immense amount of invaluable data regarding external ballistics;

Chapter 2

Six Degrees of Freedom (6DOF) Physics Engine

2.1 Introduction

The 6DOF (Six Degrees of Freedom) solver in BBTC models a bullet's full spatial motion: translation in X, Y, Z and rotation around pitch, yaw, and roll axes. It is a high-precision simulation that treats the projectile as a rigid body subject to aerodynamic, gravitational, and rotational forces.

This is not a simplistic point-mass solver. It tracks:

- 3D position and linear velocity
- 3D angular velocity and orientation using quaternions
- Environmental variables (air density, wind, humidity, etc.)
- Variable drag coefficients from Mach-dependent tables (G1, G7)

The system runs at fixed time steps and integrates motion using RK4 (Runge-Kutta 4th order) for both translational and rotational dynamics, ensuring accurate trajectories even under extreme velocities and long times of flight.

2.2 Translational Dynamics

Newton's Second Law governs linear acceleration:

$$\mathbf{F}_{\text{net}} = m \cdot \mathbf{a} \Rightarrow \vec{a} = \frac{\mathbf{F}_{\text{net}}}{m}$$

Forces computed per step:

- Gravity: $\mathbf{F}_g = m \cdot \mathbf{g}$
- Drag: $\mathbf{F}_d = -\frac{1}{2}\rho C_d A \|\mathbf{v}\| \mathbf{v}$
- Wind correction (if enabled)

Drag coefficient C_d is determined dynamically using Mach number and interpolated from the drag table selected by the user that coresponds to their desired drag model.

RK4 integration is applied:

$$\begin{aligned}
k_1 &= f(t, \mathbf{v}) \\
k_2 &= f\left(t + \tfrac{\Delta t}{2}, \mathbf{v} + \tfrac{\Delta t}{2} k_1\right) \\
k_3 &= f\left(t + \tfrac{\Delta t}{2}, \mathbf{v} + \tfrac{\Delta t}{2} k_2\right) \\
k_4 &= f(t + \Delta t, \mathbf{v} + \Delta t \cdot k_3) \\
\mathbf{v}_{t+\Delta t} &= \mathbf{v}_t + \tfrac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

2.3 Rotational Dynamics

Angular velocity ω is integrated via quaternion differential equations. The quaternion q representing orientation is updated as:

$$\frac{dq}{dt} = \frac{1}{2} q \otimes \omega_q$$

Where:

- $\omega_q = (0, \omega)$ is the angular velocity as a pure quaternion
- $\otimes$ is quaternion multiplication

The rotational integration also uses RK4 and includes normalization at every step to prevent drift from accumulating.

2.4 Forces Modeled

Current aerodynamic and environmental forces modeled:

- Gravity
- Mach-dependent drag
- Static wind (constant direction and magnitude)
- Air density variation via altitude and humidity
- Speed of sound adjustment via temperature and humidity
- Bullet spin decay due to air resistance
- Lift via yaw-induced and pitch-induced terms
- Coriolis force (Earth's rotation)
- Eötvös effect (horizontal velocity interacting with Earth rotation)
- Latitude-based gravity and curvature approximation

Wind is modeled as a fixed 3D vector for now. The idea of sinusoidal fluctuation (e.g., gusting wind patterns modeled by sine waves) is **not yet implemented** but may be considered in future releases.

Spin decay is modeled by reducing angular velocity over time with air resistance:

$$\omega_{\text{new}} = \omega_{\text{old}} \cdot (1 - \alpha \Delta t)$$

where α is an empirical decay constant.

2.5 Numerical Stability and Performance

RK4 offers excellent tradeoff between stability and speed. Time steps must remain small enough to capture high-speed changes, especially in supersonic regimes.

Units are strictly **SI internally**.

No dynamic memory allocation is used anywhere in the solver. It's completely stack-based and deterministic for maximum speed, portability, and safety.

2.6 Output Logging

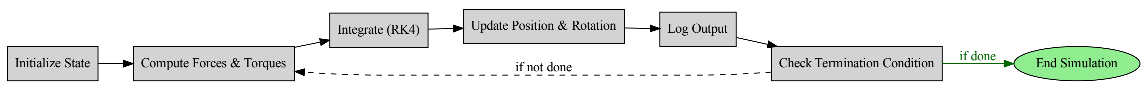
Output is written to a CSV file for every time step, including:

- Time, position, and velocity
- Orientation (quaternion and Euler angles)
- Angular velocity
- Mach number and drag
- Air density, temperature, speed of sound

This allows for visualization in gnuplot or post-hoc analysis tools.

2.7 6DOF Simulation Loop

This diagram outlines the core loop of the 6-degree-of-freedom simulation engine. It shows how the state of the bullet is advanced over time through force computation and integration. Each box represents a high-level operation performed per timestep.



2.7.1 Theory vs. Implementation Table

This table maps physical models used in the 6DOF simulation to their mathematical formulations and code-level implementations based on the actual contents of `6dof.c` , `environment.c` , `quat.c` , and `load_drag_tables.c`.

2.7.2 Theory vs. Implementation Table

Physical Concept	**Mathematical Formulation**	**Implemented In**
Translational Motion	$F = ma$	compute_6dof_derivatives()
Rotational Motion	$\tau = I\alpha + \omega \times (I\omega)$	compute_6dof_derivatives()
Drag Force	$F_d = -\frac{1}{2}\rho C_d A \|v\| v$	compute_6dof_derivatives(), interpolate_cd_from_table()
Magnus Drift	$F_m \propto \omega \times v$	compute_6dof_derivatives()
Coriolis Force	$a_c = -2\omega_E \times v$	compute_6dof_derivatives()
Eötvös Effect	$a_e = 2\omega_E v \cos\phi$	compute_6dof_derivatives()
Gravitational Force	$F_g = mg(h, \phi)$	compute_6dof_derivatives()
Spin Decay / Gyroscopic Drift	$\frac{d\omega}{dt} = \mathrm{precession}$	compute_6dof_derivatives()
Quaternion Update	$\dot{q} = \frac{1}{2} q \otimes \omega$	quaternion_derivative(), quaternion_update()
Numerical Integration	4th-Order Runge-Kutta (RK4)	rk4_integrate()
Wind Effects	$v_{\mathrm{rel}} = v_{\mathrm{bullet}} - v_{\mathrm{wind}}$	compute_6dof_derivatives(), Environment struct

2.8 Future Work

Planned and Potential additions:

- An optional second CSV in U.S. Customary units, if enabled
- Gusting wind (sinusoidal or Perlin-based)
- Live weather and location gathering for user to apply weather parameters based on their location
- Real-time zeroing correction
- Surface impact modeling (angle, penetration, ricochet logic)
- User-defined targets and hit/miss logging
- Integration with graphical front-end (OpenGL or SDL)

BBTC's 6DOF engine serves as both an educational reference and a highly accurate simulator for short- and long-range, real-world inspired ballistics.

Chapter 3

Command-Line Interface (CLI)

3.1 Overview

This page documents the command-line interface (CLI) for BBTC, the Ballistic Trajectory Calculator. All user input is passed via command-line arguments and parsed into the `CLI_inputs` structure defined in cli_opts.h.

Internally, argument parsing is handled with `getopt_long()`, supporting both short `-x` and long `--option` style flags. Invalid options trigger a usage printout and immediate exit.

3.2 Usage Syntax

```
./bbtc [OPTIONS...]
```

3.2.1 Supported Options

The following options are implemented and parsed directly by `parse_options()`:

Short	Long Option	Argument	Description
`-m`	`--magnus`	∗(none)∗	Enable Magnus effect modeling
`-c`	`--coriolis`	∗(none)∗	Enable Coriolis force modeling
`-e`	`--eotvos`	∗(none)∗	Enable Eötvös effect modeling
`-y`	`--yawRepose`	∗(none)∗	Enable yaw-of-repose modeling
`-t`	`--tilt`	∗(none)∗	Enable tilt modeling
`-Y`	`--yawFactor`	`<float>`	Set custom yaw scaling factor
`-T`	`--tiltFactor`	`<float>`	Set custom tilt scaling factor
`-M`	`--mass`	`<kg>`	Set bullet mass in kilograms
`-v`	`--velocity`	`<m/s>`	Set muzzle velocity in meters per second
`-d`	`--diameter`	`<m>`	Set bullet diameter in meters
`-a`	`--angle`	`<deg>`	Set elevation angle above horizontal
`-D`	`--direction`	`<deg>`	Set azimuth angle in degrees (0 = North)
`-w`	`--twistRate`	`<in>`	Set barrel twist rate in inches per turn
`-s`	`--timeStep`	`<sec>`	Set simulation time step (seconds)
`-x`	`--shotTemp`	`<K>`	Set ambient temperature at muzzle in Kelvin
`-G`	`--dragModel`	`<string>`	Select drag model: G1, G2, G5, G6, G7, G8, GI, GL, RA4
`-L`	`--latitude`	`<deg>`	Set firing latitude in degrees
`-A`	`--altitude`	`<m>`	Set muzzle altitude above sea level in meters
`-h`	`--help`	∗(none)∗	Show usage information and exit

3.3 CLI_inputs Structure

All parsed values are stored in a `CLI_inputs` structure:

```
typedef struct {
    SolverOptions options;     // Solver feature flags (Magnus, etc.)
    double muzzleSpeed;        // m/s
    double muzzleAngleDeg;     // degrees
    double tiltFactor;         // unitless
    double yawFactor;          // unitless
    double diam;               // meters
    double shotAngle;          // unused
    double twistInches;        // inches per rotation
    double timeStep;           // seconds
    double azimuthDeg;         // degrees
    double ambientTemp;        // Kelvin
    double mass;               // kg
    double latitude;           // degrees
    double altitude;           // meters
    DragModel dragModelChoice;// Enum (G1, G7, etc.)
} CLI_inputs;
```

3.4 Default Values

These are initialized by `initialize_CLI_inputs()`:

Field	Default
muzzleSpeed	`800.0`
muzzleAngleDeg	`1.0`
azimuthDeg	`90.0`
timeStep	`0.001`
twistInches	`7.0`
diam	`0.00556`
ambientTemp	`288.15`
mass	`0.0065`
dragModelChoice	`G1`

3.5 Example Command

```
./bbtc -mct -v 820 -a 1.5 -d 0.006 -M 0.007 -G G7 -L 45.0 -A 1000
```

This enables Magnus, Coriolis, and tilt, sets bullet parameters and drag model to G7, and fires from 1000m elevation at latitude 45°.

Chapter 4

Drag Coefficient Models

4.1 Overview

BBTC supports multiple empirical drag models that reflect various projectile shapes and flight characteristics. These drag models are used to look up the drag coefficient (C_d) as a function of Mach number, which directly influences aerodynamic resistance in the simulation.
Drag models available:

- G1: Flatbase with a blunt nose, which is typical of older bullets. Most commonly used drag model used in the market today.
- G2: Aberdeen J projectile (tangential ogive, sharp point). Not very compatible with small arms ammunition, and more so for certain artillery rounds
- G5: Short 7.5-degree boat tail with a 6.19 caliber long tangent ogive
- G6: For flat based "spire point" type bullets - 4.81 calibers long with a 2.53 caliber nose and a 6.99 caliber secant nose ogive
- G7: Long boat tail with a secant ogive, used for modern VLD bullets. Second most common model in use today—behind G1—and is suited for most modern intermediate and high power rounds. Most modern US military boat tailed bullets match this model.
- G8: Flat base with similar nose design to G7 - 3.64 calibers long with a 2.18 caliber long nose and a 10 caliber secant nose ogive. The US M2 152 gr .30 cal bullet matches this drag model. Close to the G6 model.
- GI: Converted from the original Ingalls tables, essentially G1. Not recommended for serious use other than illustration or simulation.
- GL: Traditional model used for flat-based, round nosed, exposed-lead bullets.
- GS: Model based on a 9/16" sphere. This model is best suited for spherical projectiles (cannon and musket balls, etc.) and is practically identical for all sizes of sphere.
- RA4: Used for heeled bullets that are most commonly fired from small rimfire rounds (e.g. .22LR) and airguns.

These drag models are implemented using CSV lookup tables that provide C_d values at discrete Mach intervals. During the simulation, the solver determines the projectile's Mach number, locates the appropriate table, and interpolates the drag coefficient accordingly.

4.2 Drag Force Equation

The drag force applied to the bullet is given by:

$$\mathbf{F}_d = -\frac{1}{2}\rho C_d A \|\mathbf{v}_{\text{rel}}\| \mathbf{v}_{\text{rel}}$$

Where:

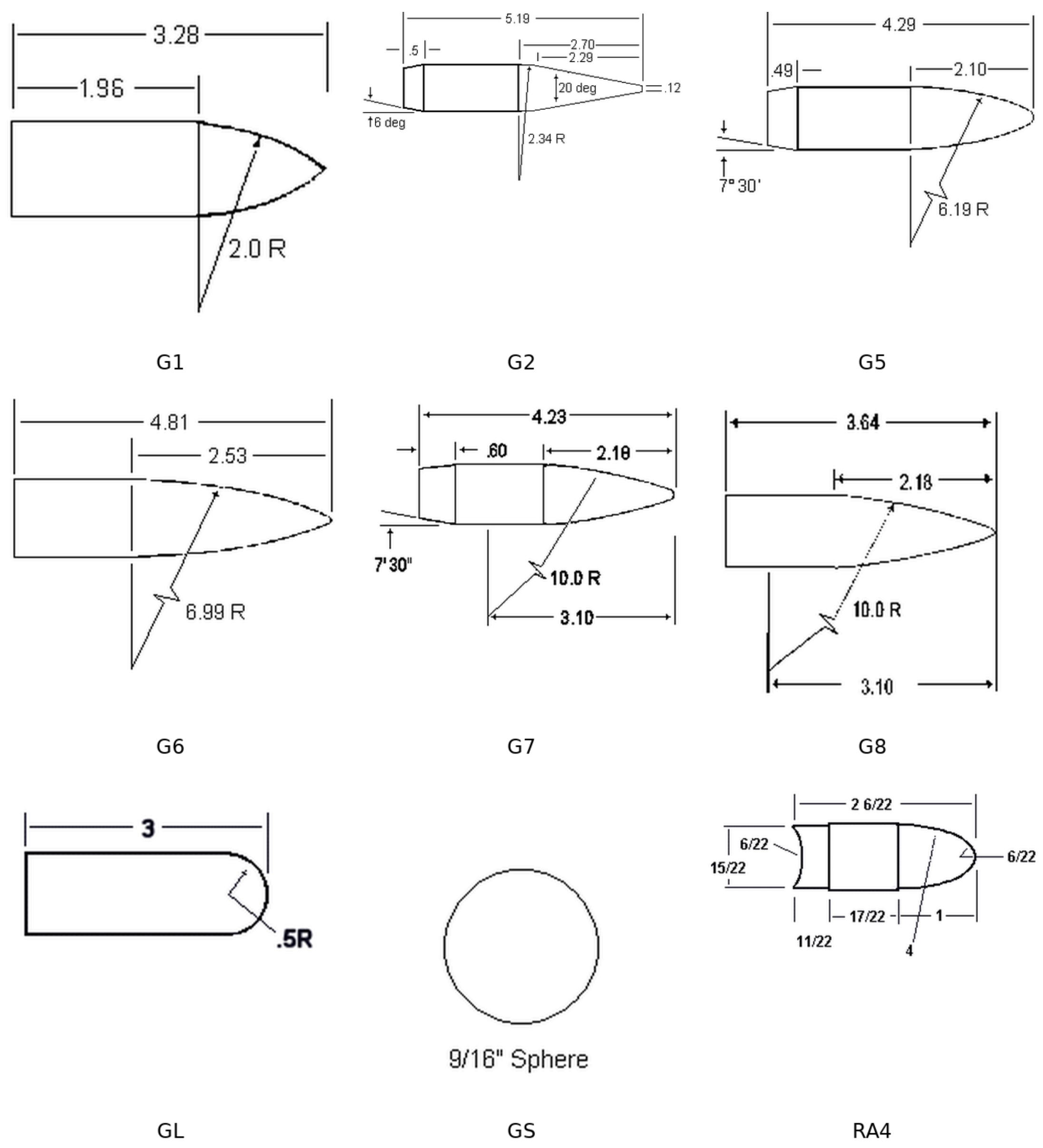


Figure 4.1 Standard Drag Model Shapes

- ρ is the air density
- C_d is the drag coefficient from the selected model
- A is the bullet's cross-sectional area
- $\mathbf{v}_{\text{rel}}$ is the velocity of the bullet relative to the air

The direction of the drag force is opposite to the motion relative to the surrounding air. This includes wind if enabled.

4.3 Interpolation Logic

Since the tables contain discrete entries, C_d must be interpolated for smooth physics. BBTC uses binary search followed by linear interpolation for performance and accuracy.
Steps:

1. Determine the Mach number from the bullet's speed and speed of sound
2. Use binary search to find the nearest bracketing values in the drag table

3. Apply linear interpolation between those two points to compute an accurate C_d

This ensures consistent drag calculation even during rapid transonic transitions where C_d changes dramatically.

4.4 Units and Internal Format

Drag model tables use dimensionless coefficients and are stored in plain-text CSV format. Internally, all physics are computed using SI units:

- Mach number (unitless)
- Cross-sectional area (m^2)
- Velocity (m/s)
- Density (kg/m^3)

4.5 Future Enhancements

Planned upgrades to drag modeling include:

- Temperature-adjusted Reynolds number influence (viscosity effects)
- Automatic curve smoothing or polynomial fitting
- User-defined drag tables
  - Validation routines to test imported tables for monotonicity and resolution

BBTC's drag model system is extensible and central to its accurate aerodynamics engine. The system ensures physically realistic bullet trajectories across a wide range of speeds and shapes.

Chapter 5

Environmental Modeling System

5.1 Overview

BBTC's environmental modeling system dynamically computes atmospheric variables that influence projectile behavior during flight. These variables include air temperature, pressure, density, humidity, and the speed of sound, each of which contributes to realistic drag, lift, and sound barrier effects.
This system ensures that long-range simulations remain accurate across altitude bands, weather conditions, and geographic locations.

5.2 Core Atmospheric Variables

The following variables are computed or adjusted during each simulation step:

- **Temperature** (T) — Measured in kelvin (K), derived from user input or standard conditions.
- **Pressure** (P) — Calculated from altitude using the barometric formula:

  $$P = P_0 \cdot \left(1 - \frac{Lh}{T_0}\right)^{\frac{gM}{RL}}$$

  where:

  - P_0: sea level pressure (101325 Pa)
  - L: lapse rate (0.0065 K/m)
  - h: altitude above sea level (m)
  - T_0: reference temperature (K)
  - g: gravitational acceleration (9.80665 m/s^2)
  - M: molar mass of Earth's air (0.0289644 kg/mol)
  - R: universal gas constant (8.31447 J/(mol·K))
- **Humidity** (H) — Impacts the effective air density and speed of sound.
- **Air Density** ρ — Adjusted based on altitude, temperature, and humidity:

  $$\rho = \frac{P}{R_d T} + \frac{e}{R_v T}$$

  where:

  - e: vapor pressure from humidity
  - R_d: specific gas constant for dry air
  - R_v: specific gas constant for water vapor
- **Speed of Sound** (a) — Influences Mach number and drag coefficient lookup:

  $$a = \sqrt{\gamma R T}$$

  γ and R are adjusted for humidity using weighted mixing rules.

5.3 Wind Modeling

Wind is currently modeled as a fixed 3D vector, representing a constant wind condition throughout the simulation. It is subtracted from the projectile's velocity to obtain a relative airspeed used in drag and lift calculations:

$$\mathbf{v}_{\text{rel}} = \mathbf{v}_{\text{bullet}} - \mathbf{v}_{\text{wind}}$$

The wind vector is defined in world coordinates (e.g., east/north/up), and its effect is consistent over time.
Future versions of BBTC may implement time-varying wind patterns, terrain-influenced wind behavior, or stochastic gust models.

5.4 Altitude Effects

All environmental variables dynamically update based on the bullet's instantaneous altitude. As the bullet ascends or descends, pressure and temperature change according to the International Standard Atmosphere (ISA) model, and density is recalculated accordingly.
These changes allow BBTC to realistically model supersonic transition behavior and long-range arcing flight paths.

5.5 Units and Internal Format

Internally, the environmental model uses strict SI units:

- Temperature: K (Kelvin)
- Pressure: Pa (Pascals = 0.000145038 psi = 0.00750062 mmHg)
- Humidity: % (0.0 to 1.0, where 1.0 = 100%)
- Density: kg/m^3 (kilograms per cubic meter = 0.062428 lb/ft^3)
- Speed of sound: m/s (meters per second = 3.28084 ft/s = 2.23694 mph)

5.6 Output Logging

The environment model contributes the following to each output step:

- Air density
- Temperature
- Pressure
- Speed of sound
- Relative velocity (for drag and Mach)

These values are logged alongside core trajectory information to allow deep post-analysis and visualization.

5.7 Future Enhancements

Planned improvements to the environmental system include:

- Input able to be entered in U.S. customary units and to be converted to SI on ingest. The simulation could optionally output a secondary CSV log in U.S. units for convenience.
- Dynamic wind profiles
- Weather layer ingestion (e.g., METAR or grib2 feeds)
- Pressure wave modeling

- Humidity-based fog effects or index of refraction simulation
- Layered atmospheric models with inversion zones
- Custom weather and location data uploaded by the user or downloaded by the user to the program through a weather API

BBTC's environmental engine ensures that projectile motion is grounded in physical reality and can adapt to a wide range of shooting scenarios.

Chapter 6

Quaternion Mathematics

6.1 Overview

This section documents the quaternion math used by B.B.T.C. to handle 3D orientation and rotation. Quaternions offer a compact, non-singular, and numerically stable method of representing orientations in three dimensions, making them a superior choice over Euler angles or rotation matrices for physics simulations that require smooth, continuous rotational updates.
This ballistic simulator uses quaternions to represent projectile orientation in a rotation-aware, drift-free way. This is essential in modeling spin-stabilized projectiles with gyroscopic precession and nutation effects.

6.2 Why Quaternions?

Rotation in 3D space can be represented in multiple ways:

- **Euler angles** (yaw, pitch, roll) are intuitive but suffer from *gimbal lock* .
- **Rotation matrices** are expressive but large (9 values) and can become numerically unstable.
- **Axis-angle pairs** are compact but harder to interpolate or apply repeatedly.
- **Quaternions** combine the best of all worlds:
  - Compact: only 4 values.
  - Interpolatable.
  - No gimbal lock.
  - Fast composition and inverse operations.
  - Easily convertible to/from axis-angle and rotation matrix forms.

In mathematical terms, a unit quaternion can be expressed as:

$$q = w + xi + yj + zk$$

where:

- $w \in \mathbb{R}$ is the scalar part,
- $(x, y, z) \in \mathbb{R}^3$ forms the vector part.

For unit quaternions (used in rotation), the constraint is:

$$w^2 + x^2 + y^2 + z^2 = 1$$

6.3 Application in Code

In this codebase, quaternions are used to:

- Represent projectile orientation in 3D space.
- Compute orientation updates during simulation.
- Apply compound rotations via quaternion multiplication.
- Normalize after integration to maintain unit length.

You provide the following functionality in `quat.c` and `quat.h`:

- `quat_multiply()` — multiplies two quaternions.
- `quat_normalize()` — ensures unit quaternion normalization.
- `quat_from_axis_angle()` — converts axis-angle pairs to quaternions.
- `quat_rotate_vec3()` — applies quaternion rotation to a 3D vector.
- `quat_to_rotation_matrix()` — converts quaternion to 3×3 matrix (optional).
- Other utility functions for quaternion conjugates and inverses.

These are the backbone of rotational motion in the solver.

6.4 Quaternion Update Step

Orientation is updated every time step using the angular velocity vector (\omega), integrated into quaternion space. Given a quaternion q and angular velocity $\omega = (\omega_x, \omega_y, \omega_z)$, we construct a pure quaternion:

$$\Omega = 0 + \omega_x i + \omega_y j + \omega_z k$$

Then, the derivative of orientation is:

$$\frac{dq}{dt} = \frac{1}{2} q \cdot \Omega$$

This differential equation is integrated numerically (e.g., via RK4), and the result is normalized to maintain unit length.

6.5 Optional: SLERP (Not Implemented)

SLERP, or *Spherical Linear intERPolation*, is a method for smoothly interpolating between two orientations represented by unit quaternions.
It has the following properties:

- Constant angular velocity interpolation.
- Always takes the shortest arc between two orientations.
- Avoids artifacts from linear interpolation or Euler angle blending.

Although **SLERP is not currently implemented** in this program, its mathematical definition is provided here for reference:
Given two unit quaternions q_0 and q_1, and interpolation parameter $t \in [0, 1]$, the SLERP function is:

$$\text{SLERP}(q_0, q_1, t) = \frac{\sin((1-t)\theta)}{\sin(\theta)} q_0 + \frac{\sin(t\theta)}{\sin(\theta)} q_1$$

where θ is the angle between q_0 and q_1:

$$\cos(\theta) = q_0 \cdot q_1$$

I am considering implementing SLERP in future revisions to enable smooth interpolation of projectile orientation (e.g., for cinematic visualization, camera tracking, or guided munitions).

6.6 Summary

Quaternions are used in this simulator because they are:

- **Robust** against gimbal lock.
- **Lightweight** and fast.
- **Ideal** for compound 3D rotations.
- **Naturally suited** to Runge-Kutta integration.

This implementation is already capable of full quaternion-based motion integration, and this provides a powerful foundation for realistic rotational dynamics. The current state of the program leaves the door open for interpolation methods like SLERP, but currently sticks to what's implemented.
For more detail, see:

- `quat.h` — quaternion API definition.
- `quat.c` — implementation of rotation math.
- `core_6dof.dox` — overview of how orientation is integrated.

Chapter 7

Topic Index

7.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 8

Data Structure Index

8.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 9

File Index

9.1 File List

Here is a list of all files with brief descriptions:

Chapter 10

Topic Documentation

10.1 Main Program

Macros

- #define BULLET_IS_AIRBORNE 1

  A macro for controlling the main simulation loop logic. Pure syntactic sugar.
- #define SEA_LVL 0.0

  Value for sea level in meters.
- #define MAX_TIME 60.0

  1 minute simulation time-limit
- #define GROUND_LEVEL 0.0

  y=0 means ground

Functions

- static const char ∗ get_drag_model_string (DragModel model)

  Returns a string name corresponding to the user's selected drag model.
- int main (int argc, char ∗argv[ ])

  The main entry point for the 6-DOF ballistic solver.

10.1.1 Detailed Description

10.1.2 Macro Definition Documentation

10.1.2.1 BULLET_IS_AIRBORNE

```
#define BULLET_IS_AIRBORNE 1
```
A macro for controlling the main simulation loop logic. Pure syntactic sugar.

10.1.2.2 GROUND_LEVEL

```
#define GROUND_LEVEL 0.0
```
y=0 means ground

10.1.2.3 MAX_TIME

```
#define MAX_TIME 60.0
```
1 minute simulation time-limit

10.1.2.4 SEA_LVL

```
#define SEA_LVL 0.0
```
Value for sea level in meters.

10.1.3 Function Documentation

10.1.3.1 get_drag_model_string()

```
static const char * get_drag_model_string (
            DragModel model )  [static]
```

Returns a string name corresponding to the user's selected drag model.

Parameters

model	The chosen DragModel enum

Returns

A readable string such as "G1", "G7", etc.

10.1.3.2 main()

```
int main (
            int argc,
            char * argv[] )
```

The main entry point for the 6-DOF ballistic solver.

Parameters

in	*argc*	The number of command-line arguments
in	*argv*	The array of command-line arguments

Returns

0 on success, non-zero if error or user requested help

10.2 6-DOF Solver

Data Structures

- struct Inertia

  Moments of inertia for the projectile in principal axes.
- struct AeroCoeffs

  Aerodynamic coefficients for advanced torque/force modeling.
- struct SixDOFSpecs

  Extended projectile specifications for 6-DOF.

Macros

- #define STANDARD_EARTH_ROTATION 7.292115e-5
- #define M_PI 3.14159265358979323846
- #define PI_OVER_180 (M_PI / 180.0)

Functions

- static void cross3 (double a[3], double b[3], double out[3])

  Cross product: $\mathbf{out} = \mathbf{a} \times \mathbf{b}$.
- static double clamp_magnus (double x, double min, double max)

  Clamps Magnus force to a safe maximum based on bullet weight.

- static void compute_bullet_axes_world (const Quat ∗ori, double forwardW[3], double upW[3], double rightW[3])

    Computes bullet's forward, up, and right unit vectors in world coordinates.

Variables

- double Inertia::Ixx
- double Inertia::Iyy
- double Inertia::Izz
- double AeroCoeffs::Cm

    Magnus coefficient (lift from spin)
- double AeroCoeffs::CspinDamp

    Damping term proportional to spin rate.
- double AeroCoeffs::CyawRepose

    Drift force due to yaw-of-repose.
- double AeroCoeffs::Ctilt

    Torque from bullet tilt during flight.
- double SixDOFSpecs::diam

    Bullet diameter (caliber) in meters.
- double SixDOFSpecs::area

    Cross-sectional area in m^2.
- double SixDOFSpecs::areaOverMass

    Precomputed area / mass ratio.
- Inertia SixDOFSpecs::inertia

    Diagonal inertia tensor.
- AeroCoeffs SixDOFSpecs::aero

    Aerodynamic coefficients.
- double SixDOFSpecs::mass

    Mass in kilograms.
- double SixDOFSpecs::rAC_local [3]

    Aerodynamic center offset vector in local bullet coordinates.

10.2.1 Detailed Description

10.2.2 Macro Definition Documentation

10.2.2.1 M_PI

```
#define M_PI 3.14159265358979323846
```

10.2.2.2 PI_OVER_180

```
#define PI_OVER_180 (M_PI / 180.0)
```

10.2.2.3 STANDARD_EARTH_ROTATION

```
#define STANDARD_EARTH_ROTATION 7.292115e-5
```

10.2.3 Function Documentation

10.2.3.1 clamp_magnus()

```
static double clamp_magnus (
             double x,
             double min,
             double max )  [inline], [static]
```

Clamps Magnus force to a safe maximum based on bullet weight.

Parameters

x	Input force value to clamp
min	Minimum allowable value
max	Maximum allowable value

Returns

Clamped value in the range [min, max]

10.2.3.2 compute_bullet_axes_world()

```
static void compute_bullet_axes_world (
            const Quat * ori,
            double forwardW[3],
            double upW[3],
            double rightW[3] )  [inline], [static]
```

Computes bullet's forward, up, and right unit vectors in world coordinates.
Useful for determining spin-axis orientation, side forces, and yaw-of-repose effects.

Parameters

ori	Bullet orientation quaternion
forwardW	Output forward vector in world space
upW	Output up vector in world space
rightW	Output right vector in world space

10.2.3.3 cross3()

```
static void cross3 (
            double a[3],
            double b[3],
            double out[3] )  [inline], [static]
```

Cross product: $\mathbf{out} = \mathbf{a} \times \mathbf{b}$.

Parameters

a	Left-hand 3D vector
b	Right-hand 3D vector
out	Output vector ($\mathbf{a} \times \mathbf{b}$)

10.2.4 Variable Documentation

10.2.4.1 aero

```
AeroCoeffs SixDOFSpecs::aero
```

Aerodynamic coefficients.

10.2.4.2 area

```
double SixDOFSpecs::area
```

Cross-sectional area in m^2.

10.2.4.3 areaOverMass

`double SixDOFSpecs::areaOverMass`

Precomputed area / mass ratio.

10.2.4.4 Cm

`double AeroCoeffs::Cm`

Magnus coefficient (lift from spin)

10.2.4.5 CspinDamp

`double AeroCoeffs::CspinDamp`

Damping term proportional to spin rate.

10.2.4.6 Ctilt

`double AeroCoeffs::Ctilt`

Torque from bullet tilt during flight.

10.2.4.7 CyawRepose

`double AeroCoeffs::CyawRepose`

Drift force due to yaw-of-repose.

10.2.4.8 diam

`double SixDOFSpecs::diam`

Bullet diameter (caliber) in meters.

10.2.4.9 inertia

`Inertia SixDOFSpecs::inertia`

Diagonal inertia tensor.

10.2.4.10 Ixx

`double Inertia::Ixx`

10.2.4.11 Iyy

`double Inertia::Iyy`

10.2.4.12 Izz

`double Inertia::Izz`

10.2.4.13 mass

`double SixDOFSpecs::mass`

Mass in kilograms.

10.2.4.14 rAC_local

`double SixDOFSpecs::rAC_local[3]`

Aerodynamic center offset vector in local bullet coordinates.

10.3 CLI Option Parsing

Data Structures

- struct CLI_inputs

  Composite type that holds the command-line input parameters specified by the user.

Functions

- void initialize_CLI_inputs (CLI_inputs ∗cli)

  Initializes defaults for CLI_Inputs, so we have a baseline.
- int parse_options (CLI_inputs ∗cli, int argc, char ∗argv[ ])

  Parses the command-line arguments for the solver.

- struct option longOpts [ ]

  The global array describing our long options.
- static const char ∗ shortOpts = "mceyM:tY:T:v:d:a:D:w:s:x:G:L:A:h"

  Character array of possible short options.
- COLD void print_usage_instructions (const char ∗progName)

  Prints usage instructions for the user, listing all recognized options.
- static int str_cmp_ignoring_case (const char ∗str1, const char ∗str2)

  Case-insensitive string compare.

10.3.1 Detailed Description

10.3.2 Function Documentation

10.3.2.1 initialize_CLI_inputs()

```
void initialize_CLI_inputs (
            CLI_inputs * cli )
```

Initializes defaults for CLI_Inputs, so we have a baseline.

Parameters

cli	pointer to CLI_Inputs struct

This function zeroes out the struct and then sets default values such as muzzleSpeed = 800.0, muzzleAngleDeg = 1.0, mass = 0.0065, etc.
Initializes defaults for CLI_Inputs, so we have a baseline.

10.3.2.2 parse_options()

```
int parse_options (
            CLI_inputs * cli,
            int argc,
            char * argv[] )
```

Parses the command-line arguments for the solver.

Parameters

in	*cli*	Pointer to CLI_Inputs to store the user choices
in	*argc*	Argument count
in	*argv*	Argument vector

Returns

0 on success, non-zero on error or if user requested help

This function uses `getopt_long` to handle short and long options. The recognized options and their effect are documented in print_usage_instructions().

10.3.2.3 print_usage_instructions()

```
COLD void print_usage_instructions (
             const char * progName )
```

Prints usage instructions for the user, listing all recognized options.

Parameters

progName	The name of the executable (argv[0])

10.3.2.4 str_cmp_ignoring_case()

```
static int str_cmp_ignoring_case (
             const char * str1,
             const char * str2 )  [static]
```

Case-insensitive string compare.

Parameters

str1	First string
str2	Second string

Returns

1 if equal ignoring case, 0 if mismatch, -1 if either is NULL

10.3.3 Variable Documentation

10.3.3.1 longOpts

```
struct option longOpts[]
```

Initial value:

```
=
{
    {"magnus",      no_argument,        0, 'm'},
    {"coriolis",    no_argument,        0, 'c'},
    {"eotvos",      no_argument,        0, 'e'},
    {"yawRepose",   no_argument,        0, 'y'},
    {"mass",        required_argument,  0, 'M'},
    {"tilt",        no_argument,        0, 't'},
    {"yawFactor",   required_argument,  0, 'Y'},
    {"tiltFactor",  required_argument,  0, 'T'},
    {"velocity",    required_argument,  0, 'v'},
    {"diameter",    required_argument,  0, 'd'},
    {"angle",       required_argument,  0, 'a'},
    {"direction",   required_argument,  0, 'D'},
    {"twistRate",   required_argument,  0, 'w'},
    {"timeStep",    required_argument,  0, 's'},
    {"shotTemp",    required_argument,  0, 'x'},
    {"dragModel",   required_argument,  0, 'G'},
    {"latitude",    required_argument,  0, 'L'},
    {"altitude",    required_argument,  0, 'A'},
    {"help",        no_argument,        0, 'h'},
    {0, 0, 0, 0}
}
```

The global array describing our long options.

10.3.3.2 shortOpts

```
const char* shortOpts = "mceyM:tY:T:v:d:a:D:w:s:x:G:L:A:h"  [static]
```

Character array of possible short options.

10.4 Compiler Detection Macros

Macros that detect the compiler being used.

Macros

- #define COMPILER_CLANG 0

  Set to 1 if compiled with Clang, otherwise 0.
- #define COMPILER_GCC 0

  Set to 1 if compiled with GCC, otherwise 0.
- #define COMPILER_MSVC 0

  Set to 1 if compiled with Microsoft Visual C++, otherwise 0.
- #define COMPILER_UNKNOWN 0

  Set to 1 if the compiler is unknown or unsupported, otherwise 0.
- #define COMPILER_UNKNOWN 1

  Set to 1 if the compiler is unknown or unsupported, otherwise 0.

10.4.1 Detailed Description

Macros that detect the compiler being used.
These macros allow conditional compilation based on the detected compiler. They normally do not have to be manually set by the user.

10.4.2 Macro Definition Documentation

10.4.2.1 COMPILER_CLANG

```
#define COMPILER_CLANG 0
```
Set to 1 if compiled with Clang, otherwise 0.

10.4.2.2 COMPILER_GCC

```
#define COMPILER_GCC 0
```
Set to 1 if compiled with GCC, otherwise 0.

10.4.2.3 COMPILER_MSVC

```
#define COMPILER_MSVC 0
```
Set to 1 if compiled with Microsoft Visual C++, otherwise 0.

10.4.2.4 COMPILER_UNKNOWN [1/2]

```
#define COMPILER_UNKNOWN 0
```
Set to 1 if the compiler is unknown or unsupported, otherwise 0.

10.4.2.5 COMPILER_UNKNOWN [2/2]

```
#define COMPILER_UNKNOWN 1
```
Set to 1 if the compiler is unknown or unsupported, otherwise 0.

10.5 C Standard Version Macros

Macros for detecting the C standard version used.
Macros for detecting the C standard version used.
These macros set C_VERSION to the detected C standard:

- 23 for C23
- 17 for C17
- 11 for C11
- 99 for C99
- 90 for C90

10.6 Optimization and Performance Macros

Macros for compiler optimizations and function visibility.

Macros

- #define ALIGN(n)

  Aligns a variable or struct to `n` bytes.
- #define HAS_BUILTIN(x) 0

  Checks if the compiler supports a particular builtin.
- #define PREFETCH(addr)

  Hints to the compiler to prefetch memory at the given address.
- #define LIKELY(x) (x)

  Optimizes branch prediction by marking an expression as likely true.
- #define HOT

  Marks a function as frequently called, hinting for better cache locality.
- #define EXPORT_SYMBOL

  Controls symbol visibility in shared libraries.

10.6.1 Detailed Description

Macros for compiler optimizations and function visibility.

10.6.2 Macro Definition Documentation

10.6.2.1 ALIGN

```
#define ALIGN(
                n )
```

Aligns a variable or struct to `n` bytes.
Example usage:

```
typedef struct ALIGN(64) {
    int a;
    double b;
} AlignedStruct;
```

10.6.2.2 EXPORT_SYMBOL

```
#define EXPORT_SYMBOL
```

Controls symbol visibility in shared libraries.

10.6.2.3 HAS_BUILTIN

```
#define HAS_BUILTIN(
              x ) 0
```

Checks if the compiler supports a particular builtin.

10.6.2.4 HOT

```
#define HOT
```

Marks a function as frequently called, hinting for better cache locality.

10.6.2.5 LIKELY

```
#define LIKELY(
              x ) (x)
```

Optimizes branch prediction by marking an expression as likely true.
Compilers may use this to optimize hot paths.

10.6.2.6 PREFETCH

```
#define PREFETCH(
              addr )
```

Hints to the compiler to prefetch memory at the given address.

10.7 Environment & Weather

Data Structures

- struct Environment

  Holds environment data such as temperature, pressure, wind, etc.

Macros

- #define TEMP_SEA_LVL_K 288.15
- #define TEMP_SEA_LVL_C 15.0065
- #define LAPSE_RATE 0.0065
- #define STANDARD_PRESSURE 101325.0065
- #define GRAVITY 9.807
- #define MOL_M_AIR 0.0289644
- #define R_DRY_AIR 287.052874
- #define R_H2O_VAPOR 461.52
- #define R_UNIV 8.3144598
- #define GAMMA_DRY_AIR 1.461
- #define GAMMA_H2O_VAPOR 1.32

Functions

- double saturation_vapor_pressure (double tK)

  Computes the saturation vapor pressure at temperature tK (Kelvin).
- void update_environment (Environment ∗restrict env, PState ∗restrict state)

  Updates the environment based on projectile state (e.g., altitude).
- void compute_speed_of_sound (Environment ∗restrict env)

  Computes the speed of sound in the environment.
- static double compute_gamma (double mass_frac_h2o)

  Computes an effective ratio of specific heats $\gamma_{\mathrm{effective}}$ given the fraction of water vapor in air.
- void update_environment (Environment ∗env, PState ∗state)

Updates the environment based on projectile state (e.g., altitude).

- #define PRECOMPUTED_EXPONENT (GRAVITY ∗ MOL_M_AIR) / (R_UNIV ∗ LAPSE_RATE)

  A constant factor used for exponent calculation in some atmospheric models.

10.7.1 Detailed Description

10.7.2 Macro Definition Documentation

10.7.2.1 GAMMA_DRY_AIR

```
#define GAMMA_DRY_AIR 1.461
```

10.7.2.2 GAMMA_H2O_VAPOR

```
#define GAMMA_H2O_VAPOR 1.32
```

10.7.2.3 GRAVITY

```
#define GRAVITY 9.807
```

10.7.2.4 LAPSE_RATE

```
#define LAPSE_RATE 0.0065
```

10.7.2.5 MOL_M_AIR

```
#define MOL_M_AIR 0.0289644
```

10.7.2.6 PRECOMPUTED_EXPONENT

```
#define PRECOMPUTED_EXPONENT (GRAVITY * MOL_M_AIR) / (R_UNIV * LAPSE_RATE)
```

A constant factor used for exponent calculation in some atmospheric models.

Defined as:

$$\text{PRECOMPUTED\_EXPONENT} = \frac{GRAVITY \times MOL\_M\_AIR}{R\_UNIV \times LAPSE\_RATE}.$$

10.7.2.7 R_DRY_AIR

```
#define R_DRY_AIR 287.052874
```

10.7.2.8 R_H2O_VAPOR

```
#define R_H2O_VAPOR 461.52
```

10.7.2.9 R_UNIV

```
#define R_UNIV 8.3144598
```

10.7.2.10 STANDARD_PRESSURE

```
#define STANDARD_PRESSURE 101325.0065
```

10.7.2.11 TEMP_SEA_LVL_C

```
#define TEMP_SEA_LVL_C 15.0065
```

10.7.2.12 TEMP_SEA_LVL_K

```
#define TEMP_SEA_LVL_K 288.15
```

10.7.3 Function Documentation

10.7.3.1 compute_gamma()

```
static double compute_gamma (
            double mass_frac_h2o )  [inline], [static]
```

Computes an effective ratio of specific heats $\gamma_{\text{effective}}$ given the fraction of water vapor in air.
Weighted average:

$$\gamma_{\text{effective}} = 1.4 \times (1 - mass\_frac\_h2o) + 1.33 \times mass\_frac\_h2o.$$

Parameters

in	*mass_frac_h2o*	Fraction of water vapor by pressure

Returns

Effective gamma

10.7.3.2 compute_speed_of_sound()

```
void compute_speed_of_sound (
            Environment *restrict env )
```

Computes the speed of sound in the environment.

Parameters

env	The environment

10.7.3.3 saturation_vapor_pressure()

```
double saturation_vapor_pressure (
            double tK )  [inline]
```

Computes the saturation vapor pressure at temperature tK (Kelvin).

Parameters

tK	Temperature in Kelvin

Returns

Saturation vapor pressure in Pa

Computes the saturation vapor pressure at temperature tK (Kelvin).
Internally converts to Celsius ($t_{\text{C}} = t_{\text{K}} - 273.15$), then uses an approximation:

$$p_{\text{sat}} = 6.1078 \times \exp\left(\frac{17.08085 \times t_{\text{C}}}{234.175 + t_{\text{C}}}\right) \times 100.0\text{Pa}.$$

Parameters

in	*tK*	Temperature in Kelvin

Returns

Saturation vapor pressure in Pascals

10.7.3.4 update_environment() [1/2]

```
void update_environment (
            Environment * env,
            PState * state )
```

Updates the environment based on projectile state (e.g., altitude).
Modifies `env->localTempK`, `env->localPressure`, `env->localDensity`, and `env->local↩ SpdOfSnd` according to various atmospheric layers:

- Troposphere
- Stratosphere (lower, mid, upper)
- Mesosphere (lower, mid, upper)
- Thermosphere
- Exosphere

Each layer has a distinct temperature/pressure model:

- Some layers have a constant temperature, with exponential decay in pressure.
- Others have linear or variable lapse rates in temperature.

After computing local temperature and pressure, the function calculates partial pressures of water vapor, total density, and speed of sound.

Parameters

env	[in,out] The environment to update
state	[in] Current projectile state for altitude, `state->y`

10.7.3.5 update_environment() [2/2]

```
void update_environment (
            Environment *restrict env,
            PState *restrict state )
```

Updates the environment based on projectile state (e.g., altitude).

Parameters

env	[in,out] The environment to update
state	[in] Current projectile state

10.8 Drag Table Infrastructure

Data Structures

- struct DragEntry

    One entry in a drag function table (Mach vs. Cd)

Enumerations

- enum DragModel {
  G1 , G2 , G5 , G6 ,
  G7 , G8 , GL , GS ,
  GI , RA4 , G_UNKNOWN }

  Standard drag function types used in external ballistics.

Functions

- size_t load_drag_table_CSV (const char ∗filename)

  Loads a drag table from a CSV file into the global lookup table.
- double interpolate_cd_from_table (double mach)

  Linearly interpolates a drag coefficient from the global lookup table.
- double fast_interpolate_cd (const double ∗restrict lookupTable, double mach)

  Quickly retrieves a drag coefficient using a precomputed table.
- void precompute_drag_lookup_table (double ∗restrict lookupTable)

  Precomputes a lookup table of Cd values for fast runtime access.
- const char ∗ get_drag_model_file (DragModel model)

  Gets the file name for a given drag model enum.

Variables

- double lookupTable [ ]

  Global drag coefficient lookup table.

- static DragEntry g_dragTable [MAX_TABLE_SIZE]
- double lookupTable [LOOKUP_SIZE]

  The precomputed finer array.
- static size_t g_tableSize = 0
- static int compare_drag_entries (const void ∗a, const void ∗b)

  Compare function for qsort, sorting DragEntry by Mach.
- #define MAX_TABLE_SIZE 500

  Max # of entries in the CSV table.
- #define LOOKUP_STEP 0.001

  Step for the finer resolution array.
- #define LOOKUP_MAX_MACH 5.0

  Maximum Mach # in the finer resolution array.
- #define LOOKUP_SIZE 5001

  $\frac{\mathrm{LOOKUP\_MAX\_MACH}}{\mathrm{LOOKUP\_STEP}} + 1$

10.8.1 Detailed Description

This module handles:

- Defining standard drag models (G1, G7, etc.),
- Parsing CSV tables of Mach/Cd values,
- Interpolating drag coefficients at arbitrary Mach numbers,
- Precomputing fast lookup tables for runtime efficiency.

10.8.2 Macro Definition Documentation

10.8.2.1 LOOKUP_MAX_MACH

```
#define LOOKUP_MAX_MACH 5.0
```
Maximum Mach # in the finer resolution array.

10.8.2.2 LOOKUP_SIZE

```
#define LOOKUP_SIZE 5001
```
$\frac{\text{LOOKUP\_MAX\_MACH}}{\text{LOOKUP\_STEP}} + 1$

10.8.2.3 LOOKUP_STEP

```
#define LOOKUP_STEP 0.001
```
Step for the finer resolution array.

10.8.2.4 MAX_TABLE_SIZE

```
#define MAX_TABLE_SIZE 500
```
Max # of entries in the CSV table.

10.8.3 Enumeration Type Documentation

10.8.3.1 DragModel

```
enum DragModel
```
Standard drag function types used in external ballistics.
These refer to common empirical drag models such as G1, G7, etc. Each is based on different reference projectile shapes:

- G1: Flat-base, blunt nose (classic bullet)
- G7: Boat-tail projectile (modern long-range)
- G5, G6, etc.: Specialized shapes
- RA4, GS, GL, GI: Less commonly used or experimental

These are used to select which drag function file to load.

Enumerator

G1	G1 Drag Model (Most used drag model today)
G2	G2 Drag Model.
G5	G5 Drag Model.
G6	G6 Drag Model.
G7	G7 Drag Model (Second-most used drag model today; most accurate for modern, boat-tailed bullets)
G8	G8 Drag Model.
GL	GL Drag Model.
GS	
GI	GS Drag Model (used for spheres, e.g., cannon and musket balls) GI Drag Model
RA4	RA4 Drag Model (used rarely and only for "heeled" bullets, only seen in air guns and small rimfire rounds)
G_UNKNOWN	Unknown or unsupported drag model.

10.8.4 Function Documentation

10.8.4.1 compare_drag_entries()

```
static int compare_drag_entries (
            const void * a,
            const void * b )  [inline], [static]
```
Compare function for qsort, sorting DragEntry by Mach.

Parameters

a	Pointer to first DragEntry
b	Pointer to second DragEntry

Returns

negative/zero/positive if a $<$ b, a == b, a $>$ b

10.8.4.2 fast_interpolate_cd()

```
double fast_interpolate_cd (
             const double *restrict lookupTable,
             double mach )
```

Quickly retrieves a drag coefficient using a precomputed table.
Similar to `interpolate_cd_from_table()`, but optimized using fixed-resolution sampling of the Mach domain and avoiding runtime search.

Parameters

lookupTable	Pointer to the precomputed Cd lookup array.
mach	Mach number.

Returns

Drag coefficient approximated from the precomputed table.

Quickly retrieves a drag coefficient using a precomputed table.

Parameters

in	*lookupTable*	A pointer to the precomputed array
in	*mach*	The Mach number to sample

Returns

Drag coefficient

We clamp mach to $[0, \mathrm{LOOKUP\_MAX\_MACH}]$, then compute:

$$\mathrm{index} = \left\lfloor \frac{\mathrm{mach}}{\mathrm{LOOKUP\_STEP}} \right\rfloor$$

and linearly interpolate between index and index+1.

10.8.4.3 get_drag_model_file()

```
const char * get_drag_model_file (
             DragModel model )
```

Gets the file name for a given drag model enum.
Maps a `DragModel` enum to a corresponding CSV filename string.

Parameters

model	Drag model to query.

Returns

Pointer to string containing the file path, or NULL if unknown.

Gets the file name for a given drag model enum.

Parameters

`in`	*model*	The selected drag model

Returns

The corresponding CSV file path. e.g. "CSV_Files/G1.csv" for G1

10.8.4.4 interpolate_cd_from_table()

```
double interpolate_cd_from_table (
            double mach )
```

Linearly interpolates a drag coefficient from the global lookup table.
Given a Mach number, this function searches for two nearby entries in the `lookupTable[]` and performs linear interpolation. If `mach` is outside the table range, it clamps to the first or last available value.

Parameters

`in`	*mach*	Mach number to query.

Returns

The interpolated or clamped drag coefficient.

Linearly interpolates a drag coefficient from the global lookup table.
Internally does a binary search to find the bracketed region. Then performs linear interpolation:

$$\mathrm{Cd}(mach) = \mathrm{Cd}(m_1) + \frac{mach - m_1}{m_2 - m_1}\big(\mathrm{Cd}(m_2) - \mathrm{Cd}(m_1)\big).$$

Complexity

$\mathcal{O}(\log n)$ due to binary search.

Parameters

`in`	*mach*	Current Mach number.
	mach_table	Sorted array of Mach numbers.
	cd_table	Corresponding array of drag coefficients.
	table_size	Number of elements in the tables.

Returns

Interpolated drag coefficient for the given Mach number.

10.8.4.5 load_drag_table_CSV()

```
size_t load_drag_table_CSV (
            const char * filename )
```

Loads a drag table from a CSV file into the global lookup table.

Parses a file formatted as "Mach,Cd" on each line, stores values in a global table.

Parameters

filename	Path to the CSV file to load.

Returns

The number of entries successfully loaded.

Note

This function allocates and populates the global `lookupTable[]`. The file must be correctly formatted as a .csv file with no spaces, where mach is the first column and Cd is the right column and there is no header, or the program will not work.

Loads a drag table from a CSV file into the global lookup table.

Parameters

`in`	*filename*	Path to the CSV file

Returns

The number of entries loaded. Returns 0 if the file could not be opened.

The CSV is expected to have lines of the form:

```
0.1,0.23
0.2,0.30
...
```

where `mach` is the left column, and `Cd` is the right.

10.8.4.6 precompute_drag_lookup_table()

```
void precompute_drag_lookup_table (
            double *restrict lookupTable )
```

Precomputes a lookup table of Cd values for fast runtime access.
This function builds a uniformly sampled array of Cd values across a typical Mach number range (e.g., 0.0 to 5.0). Used by `fast_interpolate_cd()`.

Parameters

lookupTable	Pointer to the buffer to fill with precomputed Cd values.

Precomputes a lookup table of Cd values for fast runtime access.

Parameters

`in,out`	*lookupTable*	A pointer to the array of size LOOKUP_SIZE

For each index i, we map:

$$i \cdot \text{LOOKUP\_STEP} \mapsto \text{interpolate\_cd\_from\_table}$$

10.8.5 Variable Documentation

10.8.5.1 g_dragTable

```
DragEntry g_dragTable[MAX_TABLE_SIZE]  [static]
```

10.8.5.2 g_tableSize

```
size_t g_tableSize = 0  [static]
```

10.8.5.3 lookupTable `[1/2]`

```
double lookupTable[ ]  [extern]
```

Global drag coefficient lookup table.
This global buffer is filled by `load_drag_table_CSV()` or `precompute_drag_lookup_table()` and used by the other functions for interpolation.
Global drag coefficient lookup table.

10.8.5.4 lookupTable `[2/2]`

```
double lookupTable[LOOKUP_SIZE]
```

The precomputed finer array.
Global drag coefficient lookup table.

10.9 Projectile State Tracking

Data Structures

- struct PState

    Represents the full projectile state in 3D space.
- struct StateDeriv3D

    Derivatives used in 3D integration steps (RK4, etc.).

10.9.1 Detailed Description

10.10 Quaternion Utilities

Data Structures

- struct Quat

    A quaternion representing orientation in 3D, with fields w, x, y, z.

Functions

- static Quat quat_derivatives (const Quat $*$q, const double w[3])

    Computes $\dot{q} = \frac{1}{2}\mathbf{w}_q \cdot q$ for quaternion integration.
- void quat_normalize (Quat $*$q)

    Normalize the quaternion in-place.
- Quat quat_multiply (const Quat $*$q1, const Quat $*$q2)

    Multiply two quaternions: result = q1 $*$ q2.
- Quat quat_conjugate (const Quat $*$q)

    Conjugate of a quaternion: $q^* = (w, -x, -y, -z)$.
- void quat_rotate_vector (const Quat $*$q, const double vect[3], double outVect[3])

    Rotates a 3D vector using the quaternion q.
- Quat quat_from_axis_angle (double axis_x, double axis_y, double axis_z, double angleRad)

    Create a pure rotation quaternion from an axis (x,y,z) and an angle (radians).
- void quat_integrate_angular_velocity (Quat $*$q, const double w[3], double dt)

    Update orientation by angular velocity w (in rad/s) over time dt.
- Quat quat_add_scaled (const Quat $*$q, const Quat $*$dq, double scale)

    Add a scaled quaternion dq to a base quaternion q: result = q + scale$*$dq.

- void quat_normalize (Quat $*$restrict q)

Normalizes a quaternion to unit length.

- Quat quat_conjugate (const Quat ∗restrict q)

  Computes the conjugate of a quaternion.

- void quat_rotate_vector (const Quat ∗restrict q, const double vect[3], double outVect[3])

  Rotates a 3D vector using the quaternion q.

- void quat_integrate_angular_velocity (Quat ∗restrict q, const double w[3], double dt)

  Integrates quaternion orientation by angular velocity over time dt.

10.10.1 Detailed Description

10.10.2 Function Documentation

10.10.2.1 quat_add_scaled()

```
Quat quat_add_scaled (
            const Quat * q,
            const Quat * dq,
            double scale )
```

Add a scaled quaternion dq to a base quaternion q: result = q + scale∗dq.

Parameters

q	Base quaternion
dq	Delta quaternion
scale	Scale factor

Returns

The resulting quaternion

This effectively does: $q \leftarrow q + 0.5\,(0,\omega)\,q\,dt$, then normalizes the quaternion to unit length. Typically used in RK4 steps, e.g.:

```
midState.ori = quat_add_scaled(&state->ori, &k1.dOri, 0.5 * dt);
```

Add a scaled quaternion dq to a base quaternion q: result = q + scale∗dq.

10.10.2.2 quat_conjugate() [1/2]

```
Quat quat_conjugate (
            const Quat * q )
```

Conjugate of a quaternion: $q^* = (w, -x, -y, -z)$.

Parameters

q	The input quaternion

Returns

The conjugate

10.10.2.3 quat_conjugate() [2/2]

```
Quat quat_conjugate (
            const Quat *restrict q )
```

Computes the conjugate of a quaternion.

Parameters

q	Input quaternion.

Returns

Quaternion conjugate q^* such that $q^* = (w, -x, -y, -z)$

10.10.2.4 quat_derivatives()

```
static Quat quat_derivatives (
             const Quat * q,
             const double w[3] )  [inline], [static]
```

Computes $\dot{q} = \frac{1}{2}\mathbf{w}_q \cdot q$ for quaternion integration.

Parameters

q	Input orientation quaternion
w	Angular velocity vector in radians per second

Returns

Quaternion derivative $\dot{q}$

10.10.2.5 quat_from_axis_angle()

```
Quat quat_from_axis_angle (
             double axis_x,
             double axis_y,
             double axis_z,
             double angleRad )
```

Create a pure rotation quaternion from an axis (x,y,z) and an angle (radians).

Parameters

axis_x	X component of axis
axis_y	Y component of axis
axis_z	Z component of axis
angleRad	Rotation angle in radians

Returns

The resulting quaternion

Create a pure rotation quaternion from an axis (x,y,z) and an angle (radians).

Parameters

axis	Rotation axis (unit vector).
angle	Rotation angle in radians.

Returns

Quaternion representing the rotation.

10.10.2.6 quat_integrate_angular_velocity() [1/2]

```
void quat_integrate_angular_velocity (
             Quat * q,
             const double w[3],
             double dt )
```

Update orientation by angular velocity w (in rad/s) over time dt.
This uses dq/dt = 0.5 ∗ (0, w) ∗ q, then normalizes q.

Parameters

q	Pointer to the quaternion
w	The angular velocity (3D)
dt	The small time step

10.10.2.7 quat_integrate_angular_velocity() [2/2]

```
void quat_integrate_angular_velocity (
            Quat *restrict q,
            const double w[3],
            double dt )
```

Integrates quaternion orientation by angular velocity over time dt.

10.10.2.8 quat_multiply()

```
Quat quat_multiply (
            const Quat * q1,
            const Quat * q2 )
```

Multiply two quaternions: result = q1 ∗ q2.

Parameters

q1	First quaternion
q2	Second quaternion

Returns

The product quaternion

Multiply two quaternions: result = q1 ∗ q2.
This performs Hamilton product: $q = q_1 \cdot q_2$

Parameters

a	First quaternion (left-hand operand).
b	Second quaternion (right-hand operand).

Returns

Result of quaternion multiplication.

10.10.2.9 quat_normalize() [1/2]

```
void quat_normalize (
            Quat * q )
```

Normalize the quaternion in-place.
Ensures $q.w^2 + q.x^2 + q.y^2 + q.z^2 = 1$.

Parameters

q	Pointer to the quaternion.

10.10.2.10 quat_normalize() [2/2]

```
void quat_normalize (
            Quat *restrict q )
```

Normalizes a quaternion to unit length.

Parameters

q	Quaternion to normalize (modified in place).

10.10.2.11 quat_rotate_vector() [1/2]

```
void quat_rotate_vector (
            const Quat * q,
            const double vect[3],
            double outVect[3] )
```

Rotates a 3D vector using the quaternion `q`.

Parameters

q	Rotation quaternion
vect	The input vector (x,y,z)
outVect	The rotated vector (x,y,z)

This uses an optimized formula for quaternion $*$ vect multiplication:

$$\mathbf{v}_{out} = \mathbf{v} + 2.0\Big(q_w(\mathbf{q}_v \times \mathbf{v}) + \mathbf{q}_v \times (\mathbf{q}_v \times \mathbf{v})\Big).$$

10.10.2.12 quat_rotate_vector() [2/2]

```
void quat_rotate_vector (
            const Quat *restrict q,
            const double vect[3],
            double outVect[3] )
```

Rotates a 3D vector using the quaternion `q`.

10.11 Solver Options

Data Structures

- struct SolverOptions

  Defines which advanced physics effects are accounted for in the simulation.

Enumerations

- enum SolverEffects {
  SOLVER_ENABLE_NONE = 0x00 , SOLVER_ENABLE_MAGNUS = 0x01 , SOLVER_ENABLE_CORIOLIS = 0x02 , SOLVER_ENABLE_EOTVOS = 0x04 ,
  SOLVER_ENABLE_YAW_REPOSE = 0x08 , SOLVER_ENABLE_BULLET_TILT = 0x10 }

  Bitmask flags for advanced physics effects in the 6-DOF solver.

10.11.1 Detailed Description

10.11.2 Enumeration Type Documentation

10.11.2.1 SolverEffects

`enum SolverEffects`

Bitmask flags for advanced physics effects in the 6-DOF solver.

Enumerator

SOLVER_ENABLE_NONE	
SOLVER_ENABLE_MAGNUS	
SOLVER_ENABLE_CORIOLIS	
SOLVER_ENABLE_EOTVOS	
SOLVER_ENABLE_YAW_REPOSE	
SOLVER_ENABLE_BULLET_TILT	

Chapter 11

Data Structure Documentation

11.1 AeroCoeffs Struct Reference

Aerodynamic coefficients for advanced torque/force modeling.

```
#include <6dof.h>
```

Data Fields

- double Cm

  Magnus coefficient (lift from spin)
- double CspinDamp

  Damping term proportional to spin rate.
- double CyawRepose

  Drift force due to yaw-of-repose.
- double Ctilt

  Torque from bullet tilt during flight.

11.1.1 Detailed Description

Aerodynamic coefficients for advanced torque/force modeling.
Includes empirical coefficients for effects like Magnus, spin damping, and more.
The documentation for this struct was generated from the following file:

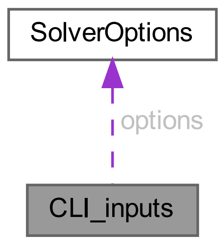
- 6dof.h

11.2 CLI_inputs Struct Reference

Composite type that holds the command-line input parameters specified by the user.

```
#include <cli_opts.h>
```

Collaboration diagram for CLI_inputs:



Data Fields

- SolverOptions options
- double muzzleSpeed
- double muzzleAngleDeg
- double tiltFactor
- double yawFactor
- double diam
- double shotAngle
- double twistInches
- double timeStep
- double azimuthDeg
- double ambientTemp
- double mass
- double latitude
- double altitude
- DragModel dragModelChoice

11.2.1 Detailed Description

Composite type that holds the command-line input parameters specified by the user.
This structure aggregates ballistic parameters such as muzzle velocity, angle, bullet mass, solver toggles, drag model selection, etc.

11.2.2 Field Documentation

11.2.2.1 altitude

`double CLI_inputs::altitude`
Muzzle altitude above sea level in meters

11.2.2.2 ambientTemp

`double CLI_inputs::ambientTemp`
Ambient temperature at muzzle in Kelvin

11.2.2.3 azimuthDeg

`double CLI_inputs::azimuthDeg`
Heading in degrees (0 = North)

11.2.2.4 diam

`double CLI_inputs::diam`
Bullet diameter in meters

11.2.2.5 dragModelChoice

`DragModel CLI_inputs::dragModelChoice`
Drag model enum specifying which ballistic drag curve is used

11.2.2.6 latitude

`double CLI_inputs::latitude`
Firing latitude in degrees

11.2.2.7 mass

`double CLI_inputs::mass`
Bullet mass in kg

11.2.2.8 muzzleAngleDeg

`double CLI_inputs::muzzleAngleDeg`
Muzzle angle in degrees above horizontal

11.2.2.9 muzzleSpeed

`double CLI_inputs::muzzleSpeed`
Muzzle velocity in m/s

11.2.2.10 options

`SolverOptions CLI_inputs::options`

11.2.2.11 shotAngle

`double CLI_inputs::shotAngle`
Not used in the code snippet, but could represent shot angle in degrees?

11.2.2.12 tiltFactor

`double CLI_inputs::tiltFactor`
Bullet tilt factor (advanced)

11.2.2.13 timeStep

`double CLI_inputs::timeStep`
Integration time step in seconds

11.2.2.14 twistInches

`double CLI_inputs::twistInches`
Barrel twist rate (inches per rotation)

11.2.2.15 yawFactor

`double CLI_inputs::yawFactor`
Yaw-of-repose factor (advanced)
The documentation for this struct was generated from the following file:

- cli_opts.h

11.3 DragEntry Struct Reference

One entry in a drag function table (Mach vs. Cd)
`#include <load_drag_tables.h>`

Data Fields

- double mach

    Mach number (ratio of bullet speed to local speed of sound)
- double Cd

    Corresponding drag coefficient.

11.3.1 Detailed Description

One entry in a drag function table (Mach vs. Cd)
This structure holds a single pair:

- `mach`: The Mach number
- `Cd`: The corresponding drag coefficient

Used for interpolation in drag functions.

11.3.2 Field Documentation

11.3.2.1 Cd

`double DragEntry::Cd`
Corresponding drag coefficient.

11.3.2.2 mach

`double DragEntry::mach`
Mach number (ratio of bullet speed to local speed of sound)
The documentation for this struct was generated from the following file:

- load_drag_tables.h

11.4 Environment Struct Reference

Holds environment data such as temperature, pressure, wind, etc.
`#include <environment.h>`

Data Fields

- double muzzleTempK

    Temperature of the air at the muzzle in K.
- double muzzlePressure

    Air pressure at the muzzle.
- double muzzleAlt

    Altitude of the muzzle.
- double latitude

    Latitude in meters.
- double relHumidity

    Humidity of the air at projectile.
- double windX

    Speed of wind, east-west, in m/s.
- double windY

Speed of wind, vertically, in m/s.
- double localSpdOfSnd

    Speed of sound at projectile.
- double windZ

    Speed of wind, north-south, in m/s.
- double localDensity

    Density of air at projectile.
- double localTempK

    Temperature of air at projectile in K.
- double spdOfSndInv

    Inverse of the local speed of sound.
- double localPressure

    Air pressure at projectile.
- double windDirDeg

    Direction of horizontal wind from 0 - 359 degrees.
- double groundLvl

    Elevation of the ground from sea level.
- double massFrac

    Mass fraction of water vapor.
- double p_vap

    Partial pressure of water vapor.
- double p_sat

    Saturation vapor pressure.
- double p_dry

    Partial pressure of dry air.

11.4.1 Detailed Description

Holds environment data such as temperature, pressure, wind, etc.

11.4.2 Field Documentation

11.4.2.1 groundLvl

```
double Environment::groundLvl
```
Elevation of the ground from sea level.

11.4.2.2 latitude

```
double Environment::latitude
```
Latitude in meters.

11.4.2.3 localDensity

```
double Environment::localDensity
```
Density of air at projectile.

11.4.2.4 localPressure

```
double Environment::localPressure
```
Air pressure at projectile.

11.4.2.5 localSpdOfSnd

```
double Environment::localSpdOfSnd
```
Speed of sound at projectile.

11.4.2.6 localTempK

`double Environment::localTempK`
Temperature of air at projectile in K.

11.4.2.7 massFrac

`double Environment::massFrac`
Mass fraction of water vapor.

11.4.2.8 muzzleAlt

`double Environment::muzzleAlt`
Altitude of the muzzle.

11.4.2.9 muzzlePressure

`double Environment::muzzlePressure`
Air pressure at the muzzle.

11.4.2.10 muzzleTempK

`double Environment::muzzleTempK`
Temperature of the air at the muzzle in K.

11.4.2.11 p_dry

`double Environment::p_dry`
Partial pressure of dry air.

11.4.2.12 p_sat

`double Environment::p_sat`
Saturation vapor pressure.

11.4.2.13 p_vap

`double Environment::p_vap`
Partial pressure of water vapor.

11.4.2.14 relHumidity

`double Environment::relHumidity`
Humidity of the air at projectile.

11.4.2.15 spdOfSndInv

`double Environment::spdOfSndInv`
Inverse of the local speed of sound.

11.4.2.16 windDirDeg

`double Environment::windDirDeg`
Direction of horizontal wind from 0 - 359 degrees.

11.4.2.17 windX

`double Environment::windX`
Speed of wind, east-west, in m/s.

11.4.2.18 windY

`double Environment::windY`

Speed of wind, vertically, in m/s.

11.4.2.19 windZ

`double Environment::windZ`

Speed of wind, north-south, in m/s.

The documentation for this struct was generated from the following file:

- environment.h

11.5 Inertia Struct Reference

Moments of inertia for the projectile in principal axes.

`#include <6dof.h>`

Data Fields

- double Ixx
- double Iyy
- double Izz

11.5.1 Detailed Description

Moments of inertia for the projectile in principal axes.

Assumes diagonal inertia tensor:

- $I = \mathrm{diag}(I_{xx}, I_{yy}, I_{zz})$

The documentation for this struct was generated from the following file:

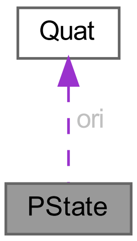
- 6dof.h

11.6 PState Struct Reference

Represents the full projectile state in 3D space.

`#include <pstate.h>`

Collaboration diagram for PState:



Data Fields

- double x

  position X
- double y

  position Y
- double z

  position Z
- double vx

  velocity in X
- double vy

  velocity in Y
- double vz

  velocity in Z
- double wx

  angular vel about local X
- double wy

  angular vel about local Y
- double wz

  angular vel about local Z
- Quat ori

  Orientation in 3D, as a quaternion.

11.6.1 Detailed Description

Represents the full projectile state in 3D space.
Fields:

- `x`, `y`, `z` for position (meters)
- `vx`, `vy`, `vz` for velocity (m/s)
- `wx`, `wy`, `wz` for angular velocity (rad/s)
- `ori` for quaternion orientation

11.6.2 Field Documentation

11.6.2.1 ori

`Quat PState::ori`
Orientation in 3D, as a quaternion.

11.6.2.2 vx

`double PState::vx`
velocity in X

11.6.2.3 vy

`double PState::vy`
velocity in Y

11.6.2.4 vz

`double PState::vz`
velocity in Z

11.6.2.5 wx

```
double PState::wx
```

angular vel about local X

11.6.2.6 wy

```
double PState::wy
```

angular vel about local Y

11.6.2.7 wz

```
double PState::wz
```

angular vel about local Z

11.6.2.8 x

```
double PState::x
```

position X

11.6.2.9 y

```
double PState::y
```

position Y

11.6.2.10 z

```
double PState::z
```

position Z

The documentation for this struct was generated from the following file:

- pstate.h

11.7 Quat Struct Reference

A quaternion representing orientation in 3D, with fields w, x, y, z.

```
#include <quat.h>
```

Data Fields

- double w

  Real part.
- double x

  i component
- double y

  j component
- double z

  k component

11.7.1 Detailed Description

A quaternion representing orientation in 3D, with fields w, x, y, z.

Typically normalized so that $w^2 + x^2 + y^2 + z^2 = 1$.

11.7.2 Field Documentation

11.7.2.1 w

```
double Quat::w
```

Real part.

11.7.2.2 x

```
double Quat::x
```

i component

11.7.2.3 y

```
double Quat::y
```

j component

11.7.2.4 z

```
double Quat::z
```

k component

The documentation for this struct was generated from the following file:

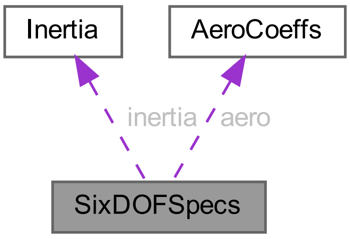
- quat.h

11.8 SixDOFSpecs Struct Reference

Extended projectile specifications for 6-DOF.

```
#include <6dof.h>
```

Collaboration diagram for SixDOFSpecs:



Data Fields

- double diam

  Bullet diameter (caliber) in meters.
- double area

  Cross-sectional area in m^2.
- double areaOverMass

  Precomputed area / mass ratio.
- Inertia inertia

  Diagonal inertia tensor.
- AeroCoeffs aero

  Aerodynamic coefficients.
- double mass

  Mass in kilograms.
- double rAC_local [3]

  Aerodynamic center offset vector in local bullet coordinates.

11.8.1 Detailed Description

Extended projectile specifications for 6-DOF.
This structure defines the bullet's physical properties needed for accurate 6DOF simulation.
The documentation for this struct was generated from the following file:

- 6dof.h

11.9 SolverOptions Struct Reference

Defines which advanced physics effects are accounted for in the simulation.
`#include <solver_options.h>`

Data Fields

- int effects

    bitmask from SolverEffects

11.9.1 Detailed Description

Defines which advanced physics effects are accounted for in the simulation.

11.9.2 Field Documentation

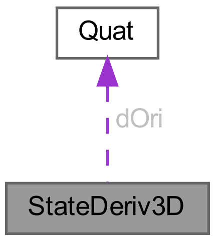
11.9.2.1 effects

`int SolverOptions::effects`
bitmask from SolverEffects
The documentation for this struct was generated from the following file:

- solver_options.h

11.10 StateDeriv3D Struct Reference

Derivatives used in 3D integration steps (RK4, etc.).
`#include <pstate.h>`
Collaboration diagram for StateDeriv3D:



Data Fields

- double dx
- double dy
- double dz

- double dvx
- double dvy
- double dvz
- double dwx
- double dwy
- double dwz
- Quat dOri

11.10.1 Detailed Description

Derivatives used in 3D integration steps (RK4, etc.).
Typically:

- `dx`, `dy`, `dz` = velocity,
- `dvx`, `dvy`, `dvz` = acceleration,
- `dwx`, `dwy`, `dwz` = angular acceleration,
- `dOri` = orientation derivative = 0.5 ∗ (0, w) ∗ q

11.10.2 Field Documentation

11.10.2.1 dOri

```
Quat StateDeriv3D::dOri
```

11.10.2.2 dvx

```
double StateDeriv3D::dvx
```

11.10.2.3 dvy

```
double StateDeriv3D::dvy
```

11.10.2.4 dvz

```
double StateDeriv3D::dvz
```

11.10.2.5 dwx

```
double StateDeriv3D::dwx
```

11.10.2.6 dwy

```
double StateDeriv3D::dwy
```

11.10.2.7 dwz

```
double StateDeriv3D::dwz
```

11.10.2.8 dx

```
double StateDeriv3D::dx
```

11.10.2.9 dy

```
double StateDeriv3D::dy
```

11.10.2.10 dz

```
double StateDeriv3D::dz
```

The documentation for this struct was generated from the following file:

- pstate.h

Chapter 12

File Documentation

12.1 a_core_6dof.dox File Reference

12.2 cli_opts.dox File Reference

12.3 drag_model.dox File Reference

12.4 environment.dox File Reference

12.5 mainpage.dox File Reference

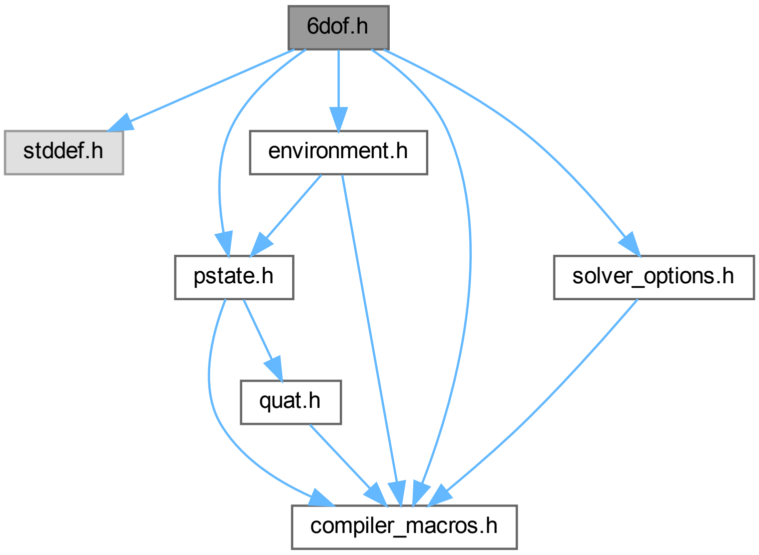
12.6 quat_math.dox File Reference

12.7 6dof.h File Reference

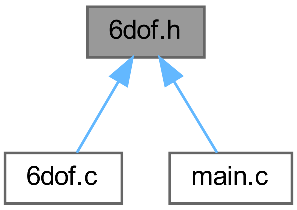
6 degrees-of-freedom (6-DOF) ballistic solver interface and data structures.

```
#include <stddef.h>
#include "pstate.h"
#include "environment.h"
#include "compiler_macros.h"
#include "solver_options.h"
```

Include dependency graph for 6dof.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Inertia

  Moments of inertia for the projectile in principal axes.
- struct AeroCoeffs

  Aerodynamic coefficients for advanced torque/force modeling.
- struct SixDOFSpecs

  Extended projectile specifications for 6-DOF.

Macros

- #define STANDARD_EARTH_ROTATION 7.292115e-5
- #define M_PI 3.14159265358979323846
- #define PI_OVER_180 (M_PI / 180.0)

12.7.1 Detailed Description

6 degrees-of-freedom (6-DOF) ballistic solver interface and data structures.
This header defines the data structures and function prototypes for simulating a full six-degrees-of-freedom projectile. It models translation (3 DOF) and rotation (3 DOF) using quaternion-based orientation, angular velocities, and an inertia tensor. Supports Magnus force, Coriolis effect, spin damping, yaw of repose, and other aerodynamic torques.
See core_6dof for how these operations are used in the simulator.

12.8 6dof.h

Go to the documentation of this file.

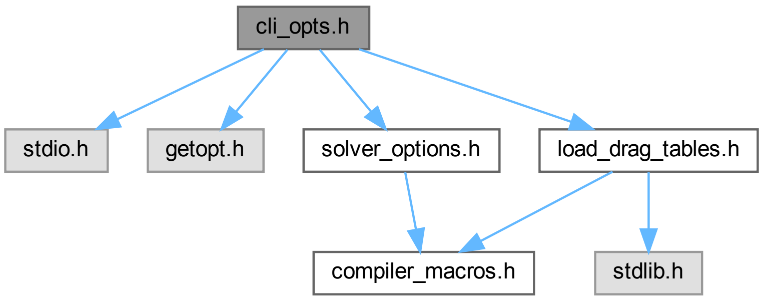
```

#ifndef SIX_DOF_H
#define SIX_DOF_H

#ifdef __cplusplus
    extern "C" {
#endif

#include <stddef.h>
#include "pstate.h"
#include "environment.h"
#include "compiler_macros.h"
#include "solver_options.h"

// fallback macros
#ifndef STANDARD_EARTH_ROTATION
    #define STANDARD_EARTH_ROTATION 7.292115e-5
#endif
#ifndef M_PI
    #define M_PI 3.14159265358979323846
#endif
#ifndef PI_OVER_180
    #define PI_OVER_180 (M_PI / 180.0)
#endif

typedef struct ALIGN(64)
{
    double Ixx; // Inertia about local X axis
    double Iyy; // Inertia about local Y axis
    double Izz; // Inertia about local Z axis (spin axis if axisymmetric)
} Inertia;

typedef struct ALIGN(64)
{
    double Cm;
    double CspinDamp;
    double CyawRepose;
    double Ctilt;
} AeroCoeffs;

typedef struct ALIGN(64)
{
    double diam;
    double area;
    double areaOverMass;
    Inertia inertia;
    AeroCoeffs aero;
    double mass;
    double rAC_local[3];
} SixDOFSpecs;

void compute_6dof_derivatives(
    const PState         *state,
    const Environment    *env,
    const SixDOFSpecs    *specs,
    StateDeriv3D         *dState,
    const SolverOptions *options
);

HOT void rk4_step_6dof(
    PState               *state,
    double                dt,
    const Environment     *env,
    const SixDOFSpecs     *specs,
    const SolverOptions   *options
);

// end of Doxygen Parsing group "SixDOF_Solver"

#ifdef __cplusplus
}
#endif

#endif /* SIX_DOF_H */

```

12.9 cli_opts.h File Reference

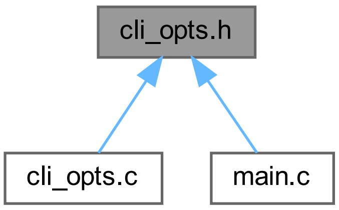
Function declarations and datatypes for user input through the command line.

```
#include <stdio.h>
#include <getopt.h>
#include "solver_options.h"
#include "load_drag_tables.h"
```

Include dependency graph for cli_opts.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct CLI_inputs

  Composite type that holds the command-line input parameters specified by the user.

Functions

- void initialize_CLI_inputs (CLI_inputs ∗cli)

  Initializes defaults for CLI_Inputs, so we have a baseline.
- int parse_options (CLI_inputs ∗cli, int argc, char ∗argv[ ])

Parses the command-line arguments for the solver.

12.9.1 Detailed Description

Function declarations and datatypes for user input through the command line.

12.10 cli_opts.h

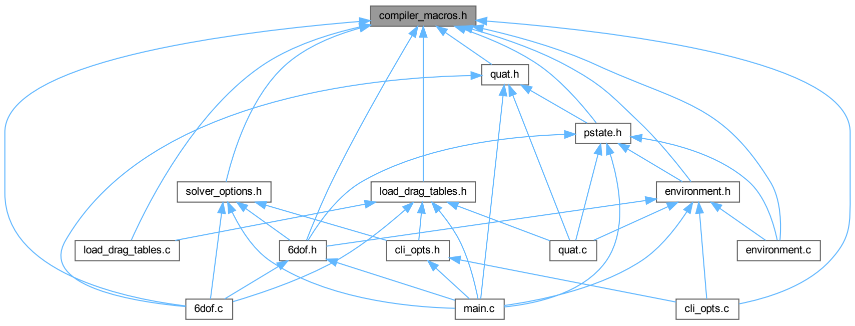
Go to the documentation of this file.

```
#ifndef CLI_OPTS_H
#define CLI_OPTS_H

#include <stdio.h>
#include <getopt.h>
#include "solver_options.h"
#include "load_drag_tables.h"

#ifdef __cplusplus
    extern "C"{
#endif

typedef struct
{
    SolverOptions options;   /* Toggles/flags (Magnus, Coriolis, etc.) from solver_options.h */

    // Ballistic Parameters:
    double muzzleSpeed;
    double muzzleAngleDeg;
    double tiltFactor;
    double yawFactor;
    double diam;
    double shotAngle;
    double twistInches;
    double timeStep;
    double azimuthDeg;
    double ambientTemp;
    double mass;
    double latitude;
    double altitude;
    DragModel dragModelChoice;
} CLI_inputs;

void initialize_CLI_inputs(CLI_inputs *cli);

int parse_options(CLI_inputs *cli, int argc, char *argv[]);

// end of CLI_Parsing group

#ifdef __cplusplus
}
#endif
#endif /* CLI_OPTS_H */
```

12.11 compiler_macros.h File Reference

Contains macros for cross-compatible compiler optimizations and compatibility.

This graph shows which files directly or indirectly include this file:



Macros

- #define COMPILER_CLANG 0

    Set to 1 if compiled with Clang, otherwise 0.
- #define COMPILER_GCC 0

    Set to 1 if compiled with GCC, otherwise 0.
- #define COMPILER_MSVC 0

    Set to 1 if compiled with Microsoft Visual C++, otherwise 0.
- #define COMPILER_UNKNOWN 0

    Set to 1 if the compiler is unknown or unsupported, otherwise 0.
- #define COMPILER_UNKNOWN 1

    Set to 1 if the compiler is unknown or unsupported, otherwise 0.
- #define C_VERSION 90
- #define ALIGN(n)

    Aligns a variable or struct to n bytes.
- #define HAS_BUILTIN(x) 0

    Checks if the compiler supports a particular builtin.
- #define PREFETCH(addr)

    Hints to the compiler to prefetch memory at the given address.
- #define LIKELY(x) (x)

    Optimizes branch prediction by marking an expression as likely true.
- #define UNLIKELY(x) (x)
- #define HOT

    Marks a function as frequently called, hinting for better cache locality.
- #define EXPORT_SYMBOL

    Controls symbol visibility in shared libraries.

12.11.1 Detailed Description

Contains macros for cross-compatible compiler optimizations and compatibility.

12.11.2 Macro Definition Documentation

12.11.2.1 C_VERSION

```
#define C_VERSION 90
```

12.11.2.2 UNLIKELY

```
#define UNLIKELY(
                x ) (x)
```

12.12 compiler_macros.h

Go to the documentation of this file.

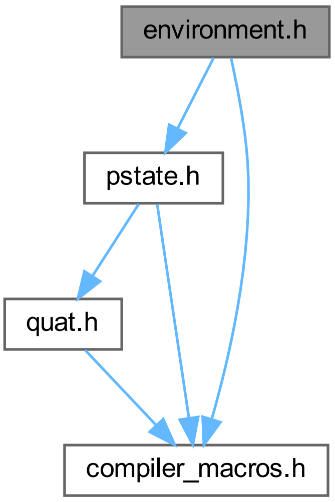
```

#ifndef COMPILER_MACROS_H
#define COMPILER_MACROS_H

#define COMPILER_CLANG 0

#define COMPILER_GCC 0

#define COMPILER_MSVC 0

#define COMPILER_UNKNOWN 0

#if defined (__clang__)
  #undef COMPILER_CLANG
  #define COMPILER_CLANG 1
#elif defined(__GNUC__) || defined(__GNUG__)
  #undef COMPILER_GCC
  #define COMPILER_GCC 1
#elif defined(_MSC_VER)
  #undef COMPILER_MSVC
  #define COMPILER_MSVC 1
#else
  #undef COMPILER_UNKNOWN
  #define COMPILER_UNKNOWN 1
#endif

#ifdef __STDC_VERSION__
  #if __STDC_VERSION__ >= 202311L
    #define C_VERSION 23
  #elif __STDC_VERSION__ >= 201710L
    #define C_VERSION 17
  #elif __STDC_VERSION__ >= 201112L
    #define C_VERSION 11
  #elif __STDC_VERSION__ >= 199901L
    #define C_VERSION 99
  #else
    #define C_VERSION 90
  #endif
#else
  #define C_VERSION 90
#endif

#ifdef __cplusplus
  #undef restrict
  #define restrict /* Disable restrict in C++ */
#endif

#ifdef DOXYGEN
  #define ALIGN(n)
#elif COMPILER_GCC || COMPILER_CLANG
  #define ALIGN(n) __attribute__((aligned(n)))
#elif COMPILER_MSVC
  #define ALIGN(n) __declspec(align(n))
#else
  #define ALIGN(n)
#endif

#ifdef DOXYGEN
  #define HAS_BUILTIN(x) 0
#elif defined(__has_builtin)
  #define HAS_BUILTIN(x) __has_builtin(x)
#else
  #define HAS_BUILTIN(x) 0
#endif

#ifdef DOXYGEN
  #define PREFETCH(addr)
#elif HAS_BUILTIN(__builtin_prefetch)
  #define PREFETCH(addr) __builtin_prefetch(addr)
#else
  #define PREFETCH(addr)
#endif

#ifdef DOXYGEN
  #define LIKELY(x) (x)
  #define UNLIKELY(x)  (x)
#elif COMPILER_GCC || COMPILER_CLANG
  #define LIKELY(x)    __builtin_expect(!!(x), 1)
  #define UNLIKELY(x) __builtin_expect(!!(x), 0)
#else
  #define LIKELY(x)    (x)
  #define UNLIKELY(x)  (x)
#endif

#ifdef DOXYGEN
  #define HOT
#elif COMPILER_GCC || COMPILER_CLANG
  #define HOT __attribute__((hot))
#else
  #define HOT
#endif

#ifdef DOXYGEN
  #define EXPORT_SYMBOL
#elif COMPILER_GCC || COMPILER_CLANG
  #define EXPORT_SYMBOL __attribute__((visibility("default")))
#elif COMPILER_MSVC
  #define EXPORT_SYMBOL __declspec(dllexport)
#else
  #define EXPORT_SYMBOL
#endif

#endif // COMPILER_MACROS_H
```

12.13 environment.h File Reference

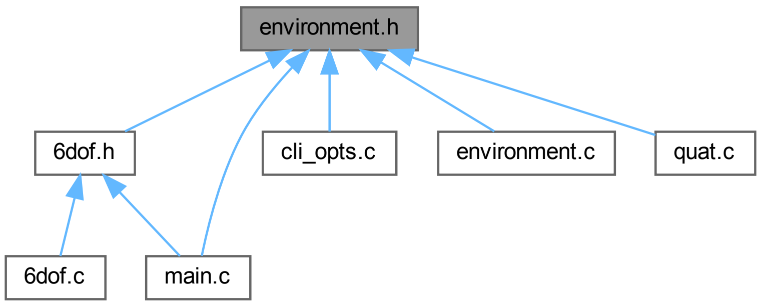
Declarations for environment & weather calculations.

```
#include "pstate.h"
#include "compiler_macros.h"
```

Include dependency graph for environment.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Environment

  Holds environment data such as temperature, pressure, wind, etc.

Macros

- #define TEMP_SEA_LVL_K 288.15
- #define TEMP_SEA_LVL_C 15.0065
- #define LAPSE_RATE 0.0065
- #define STANDARD_PRESSURE 101325.0065
- #define GRAVITY 9.807
- #define MOL_M_AIR 0.0289644
- #define R_DRY_AIR 287.052874
- #define R_H2O_VAPOR 461.52
- #define R_UNIV 8.3144598
- #define GAMMA_DRY_AIR 1.461
- #define GAMMA_H2O_VAPOR 1.32

Functions

- double saturation_vapor_pressure (double tK)

  Computes the saturation vapor pressure at temperature tK (Kelvin).
- void update_environment (Environment ∗restrict env, PState ∗restrict state)

  Updates the environment based on projectile state (e.g., altitude).
- void compute_speed_of_sound (Environment ∗restrict env)

  Computes the speed of sound in the environment.

12.13.1 Detailed Description

Declarations for environment & weather calculations.

12.14 environment.h

Go to the documentation of this file.

```
00001
00007 #ifndef ENVIRONMENT_H
00008 #define ENVIRONMENT_H
00009
```

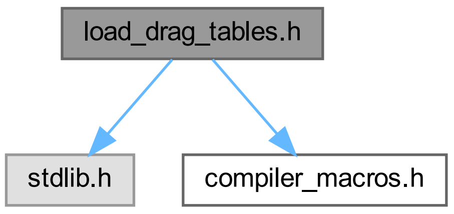
```
#ifdef __cplusplus
    extern "C"{
#endif

#include "pstate.h"
#include "compiler_macros.h"

// Macros for standard environment constants
#define TEMP_SEA_LVL_K    288.15        // Kelvin
#define TEMP_SEA_LVL_C    15.0065       // Celsius
#define LAPSE_RATE        0.0065        // K/m
#define STANDARD_PRESSURE 101325.0065   // Pa
#define GRAVITY           9.807         // m/s^2
#define MOL_M_AIR         0.0289644     // kg/mol
#define R_DRY_AIR         287.052874    // J/(kg*K)
#define R_H2O_VAPOR       461.52        // J/(kg*K)
#define R_UNIV            8.3144598     // J/(mol*K)
#define GAMMA_DRY_AIR     1.461
#define GAMMA_H2O_VAPOR   1.32

typedef struct ALIGN(64)
{
    double  muzzleTempK;
    double  muzzlePressure;
    double  muzzleAlt;
    double  latitude;
    double  relHumidity;
    double  windX;
    double  windY;
    double  localSpdOfSnd;
    double  windZ;
    double  localDensity;
    double  localTempK;
    double  spdOfSndInv;
    double  localPressure;
    double  windDirDeg;
    double  groundLvl;
    double  massFrac;
    double  p_vap;
    double  p_sat;
    double  p_dry;
} Environment;

double saturation_vapor_pressure(double tK);

void update_environment(Environment *restrict env, PState *restrict state);

void compute_speed_of_sound(Environment *restrict env);

// end of environment group
#ifdef __cplusplus
}
#endif
#endif /* ENVIRONMENT_H */
```

12.15 load_drag_tables.h File Reference

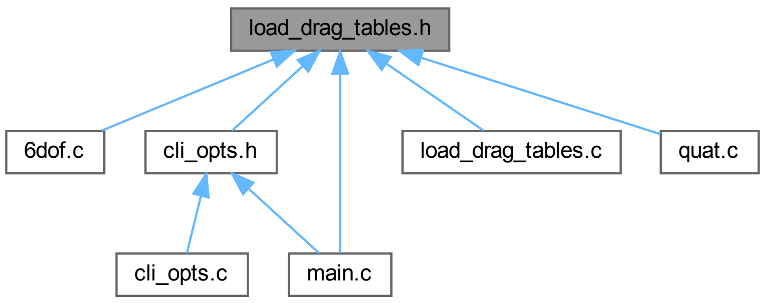
Drag model support and CSV loading for standard ballistic drag functions.

```
#include <stdlib.h>
#include "compiler_macros.h"
```

Include dependency graph for load_drag_tables.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct DragEntry

  One entry in a drag function table (Mach vs. Cd)

Enumerations

- enum DragModel {
  G1 , G2 , G5 , G6 ,
  G7 , G8 , GL , GS ,
  GI , RA4 , G_UNKNOWN }

  Standard drag function types used in external ballistics.

Functions

- size_t load_drag_table_CSV (const char ∗filename)

  Loads a drag table from a CSV file into the global lookup table.
- double interpolate_cd_from_table (double mach)

  Linearly interpolates a drag coefficient from the global lookup table.
- double fast_interpolate_cd (const double ∗restrict lookupTable, double mach)

  Quickly retrieves a drag coefficient using a precomputed table.
- void precompute_drag_lookup_table (double ∗restrict lookupTable)

Precomputes a lookup table of Cd values for fast runtime access.
- const char ∗ get_drag_model_file (DragModel model)

  Gets the file name for a given drag model enum.

Variables

- double lookupTable [ ]

  Global drag coefficient lookup table.

12.15.1 Detailed Description

Drag model support and CSV loading for standard ballistic drag functions.

12.16 load_drag_tables.h

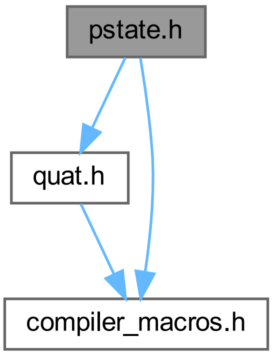
Go to the documentation of this file.
```

#ifndef LOAD_DRAG_TABLES_H
#define LOAD_DRAG_TABLES_H

#ifdef __cplusplus
    extern "C" {
#endif

#include <stdlib.h>
#include "compiler_macros.h"

typedef enum
{
    G1,
    G2,
    G5,
    G6,
    G7,
    G8,
    GL,
    GS,
    GI,
    RA4,
    G_UNKNOWN
} DragModel;

typedef struct ALIGN(64)
{
    double mach;
    double Cd;
} DragEntry;

extern double lookupTable[];

size_t load_drag_table_CSV(const char *filename);

double interpolate_cd_from_table(double mach);

double fast_interpolate_cd(const double *restrict lookupTable, double mach);

void precompute_drag_lookup_table(double *restrict lookupTable);

const char* get_drag_model_file(DragModel model);

// end of Doxygen parsing group "DragTableFunctions"

#ifdef __cplusplus
}
#endif
#endif /* LOAD_DRAG_TABLES_H */


```

12.17 pstate.h File Reference

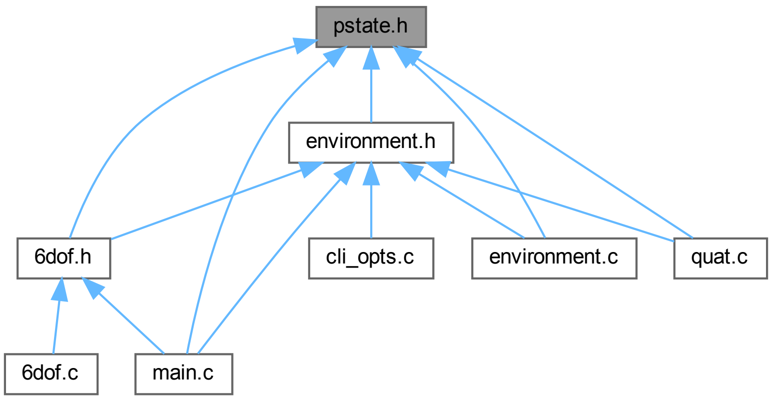
Defines the projectile state in 3D (position, velocity, orientation, etc.).

```
#include "quat.h"
#include "compiler_macros.h"
```

Include dependency graph for pstate.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct PState

  Represents the full projectile state in 3D space.
- struct StateDeriv3D

  Derivatives used in 3D integration steps (RK4, etc.).

12.17.1 Detailed Description

Defines the projectile state in 3D (position, velocity, orientation, etc.).

12.18 pstate.h

Go to the documentation of this file.

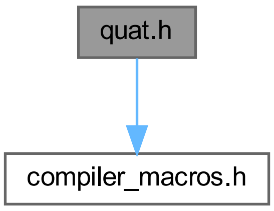
```
#ifndef PSTATE_H
#define PSTATE_H

#ifdef __cplusplus
    extern "C" {
#endif

#include "quat.h"
#include "compiler_macros.h"

typedef struct ALIGN(64)
{
    double x;
    double y;
    double z;
    double vx;
    double vy;
    double vz;
    double wx;
    double wy;
    double wz;

    Quat   ori;
} PState;

typedef struct ALIGN(64)
{
    double dx,  dy,  dz;
    double dvx, dvy, dvz;
    double dwx, dwy, dwz;

    Quat   dOri;
} StateDeriv3D;

// end of Doxygen Parsing group "ProjectileState"

#ifdef __cplusplus
}
#endif
#endif /* PSTATE_H */


```

12.19 quat.h File Reference

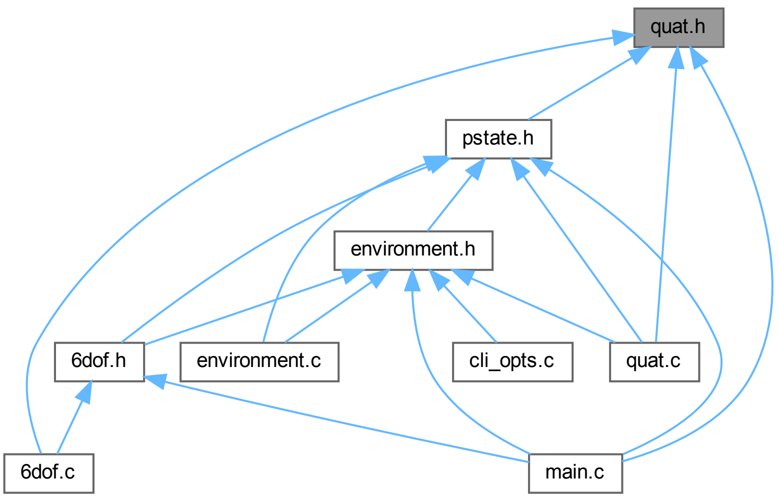
Declarations for quaternion-based rotation utilities.

`#include "compiler_macros.h"`

Include dependency graph for quat.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Quat

  A quaternion representing orientation in 3D, with fields w, x, y, z.

Functions

- void quat_normalize (Quat ∗q)

  Normalize the quaternion in-place.
- Quat quat_multiply (const Quat ∗q1, const Quat ∗q2)

  Multiply two quaternions: result = q1 ∗ q2.
- Quat quat_conjugate (const Quat ∗q)

  Conjugate of a quaternion: $q^* = (w, -x, -y, -z)$.
- void quat_rotate_vector (const Quat ∗q, const double vect[3], double outVect[3])

  Rotates a 3D vector using the quaternion q.
- Quat quat_from_axis_angle (double axis_x, double axis_y, double axis_z, double angleRad)

  Create a pure rotation quaternion from an axis (x,y,z) and an angle (radians).
- void quat_integrate_angular_velocity (Quat ∗q, const double w[3], double dt)

  Update orientation by angular velocity w (in rad/s) over time dt.
- Quat quat_add_scaled (const Quat ∗q, const Quat ∗dq, double scale)

  Add a scaled quaternion dq to a base quaternion q: result = q + scale∗dq.

12.19.1 Detailed Description

Declarations for quaternion-based rotation utilities.

12.20 quat.h

Go to the documentation of this file.

```
00001
00006 #ifndef QUAT_H
00007 #define QUAT_H
```

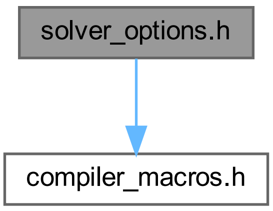
```

#ifdef __cplusplus
    extern "C"{
#endif

#include "compiler_macros.h"

typedef struct ALIGN(64)
{
    double w;
    double x;
    double y;
    double z;
} Quat;

void quat_normalize(Quat *q);

Quat quat_multiply(const Quat *q1, const Quat *q2);

Quat quat_conjugate(const Quat *q);

void quat_rotate_vector(const Quat *q, const double vect[3], double outVect[3]);

Quat quat_from_axis_angle(double axis_x, double axis_y, double axis_z, double angleRad);

void quat_integrate_angular_velocity(Quat *q, const double w[3], double dt);

Quat quat_add_scaled(const Quat *q, const Quat *dq, double scale);

// end of group "QuaternionUtilities"

#ifdef __cplusplus
}
#endif

#endif /* QUAT_H */


```

12.21 solver_options.h File Reference

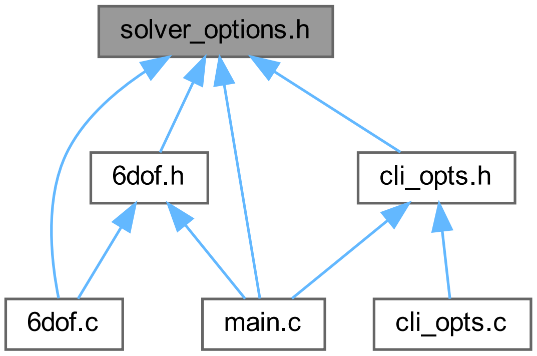
Defines toggles/flags controlling advanced solver effects.

`#include "compiler_macros.h"`

Include dependency graph for solver_options.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct SolverOptions

  Defines which advanced physics effects are accounted for in the simulation.

Enumerations

- enum SolverEffects {
  SOLVER_ENABLE_NONE = 0x00 , SOLVER_ENABLE_MAGNUS = 0x01 , SOLVER_ENABLE_CORIOLIS = 0x02 , SOLVER_ENABLE_EOTVOS = 0x04 ,
  SOLVER_ENABLE_YAW_REPOSE = 0x08 , SOLVER_ENABLE_BULLET_TILT = 0x10 }

  Bitmask flags for advanced physics effects in the 6-DOF solver.

12.21.1 Detailed Description

Defines toggles/flags controlling advanced solver effects.

12.22 solver_options.h

Go to the documentation of this file.

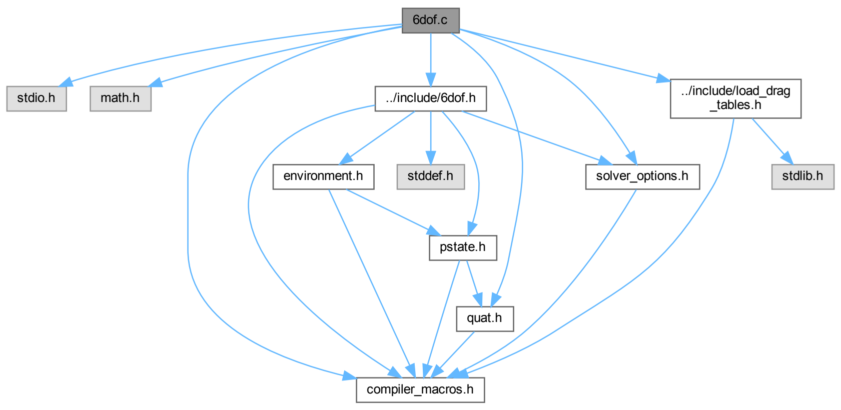
```

#ifndef SOLVER_OPTIONS_H
#define SOLVER_OPTIONS_H

#ifdef __cplusplus
    extern "C" {
#endif

#include "compiler_macros.h"

typedef enum
{
    SOLVER_ENABLE_NONE        = 0x00,
    SOLVER_ENABLE_MAGNUS      = 0x01,
    SOLVER_ENABLE_CORIOLIS    = 0x02,
    SOLVER_ENABLE_EOTVOS      = 0x04,
    SOLVER_ENABLE_YAW_REPOSE  = 0x08,
    SOLVER_ENABLE_BULLET_TILT = 0x10,
    // etc...
} SolverEffects;

typedef struct ALIGN(64)
{
    int effects;
} SolverOptions;

// end group
#ifdef __cplusplus
}
#endif
#endif /* SOLVER_OPTIONS_H */
```

12.23 6dof.c File Reference

6 degrees-of-freedom (6 D.O.F.) ballistic solver implementation.

```
#include <stdio.h>
#include <math.h>
#include "../include/6dof.h"
#include "../include/quat.h"
#include "../include/compiler_macros.h"
#include "../include/load_drag_tables.h"
#include "../include/solver_options.h"
```

Include dependency graph for 6dof.c:



Macros

- #define PI_TIMES_0_25 (M_PI ∗ 0.25)

Functions

- static void cross3 (double a[3], double b[3], double out[3])

  Cross product: $\mathbf{out} = \mathbf{a} \times \mathbf{b}$.
- static double clamp_magnus (double x, double min, double max)

  Clamps Magnus force to a safe maximum based on bullet weight.
- static Quat quat_derivatives (const Quat ∗q, const double w[3])

  Computes $\dot{q} = \frac{1}{2}\mathbf{w}_q \cdot q$ *for quaternion integration.*
- static void compute_bullet_axes_world (const Quat ∗ori, double forwardW[3], double upW[3], double rightW[3])

  Computes bullet's forward, up, and right unit vectors in world coordinates.
- void compute_6dof_derivatives (const PState ∗state, const Environment ∗env, const SixDOFSpecs ∗specs, StateDeriv3D ∗dState, const SolverOptions ∗options)

  Computes the full set of 6DOF derivatives for simulation.
- void rk4_step_6dof (PState ∗state, double dt, const Environment ∗env, const SixDOFSpecs ∗specs, const SolverOptions ∗options)

  Performs a single 6-DOF RK4 integration step on the projectile state.

12.23.1 Detailed Description

6 degrees-of-freedom (6 D.O.F.) ballistic solver implementation.
This file implements an advanced torque-based 6-DOF solver, building on the existing 3D ballistic framework. It uses an inertia tensor and aerodynamic coefficients to compute rotational accelerations, optionally including Coriolis, Eötvös, Magnus force & torque, and spin damping.
For a complete overview of the physics modeled here, refer to core_6dof.

12.23.2 Macro Definition Documentation

12.23.2.1 PI_TIMES_0_25

```
#define PI_TIMES_0_25 (M_PI * 0.25)
```

12.23.3 Function Documentation

12.23.3.1 compute_6dof_derivatives()

```
void compute_6dof_derivatives (
            const PState * state,
            const Environment * env,
            const SixDOFSpecs * specs,
            StateDeriv3D * dState,
            const SolverOptions * options )
```

Computes the full set of 6DOF derivatives for simulation.

Parameters

state	Current physical state of the bullet.
env	Environmental parameters.
options	Solver control options.
deriv	Output derivatives of the bullet's state.

12.23.3.2 rk4_step_6dof()

```
void rk4_step_6dof (
            PState * state,
            double dt,
            const Environment * env,
            const SixDOFSpecs * specs,
            const SolverOptions * options )
```

Performs a single 6-DOF RK4 integration step on the projectile state.

Parameters

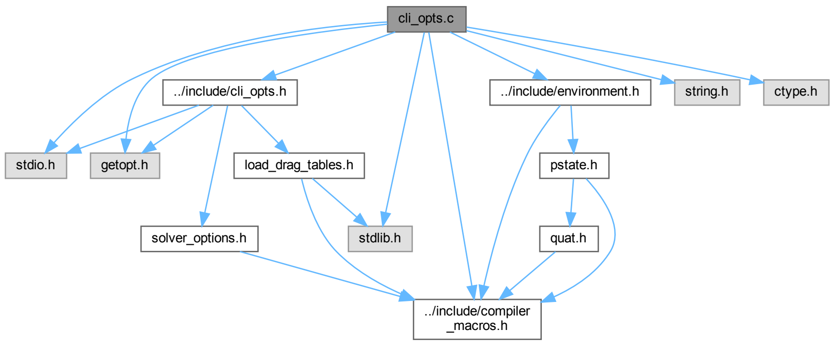
state	Mutable pointer to projectile state (position, velocity, orientation, etc.)
dt	Time step in seconds
env	Pointer to environmental conditions
specs	Pointer to 6DOF specifications (e.g. bullet parameters)
options	Pointer to simulation solver options

12.24 cli_opts.c File Reference

Definitions of functions for user input through the command line.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <getopt.h>
#include "../include/compiler_macros.h"
#include "../include/cli_opts.h"
#include "../include/environment.h"
```

Include dependency graph for cli_opts.c:



- struct option longOpts [ ]

  The global array describing our long options.
- static const char ∗ shortOpts = "mceyM:tY:T:v:d:a:D:w:s:x:G:L:A:h"

  Character array of possible short options.
- COLD void print_usage_instructions (const char ∗progName)

  Prints usage instructions for the user, listing all recognized options.
- static int str_cmp_ignoring_case (const char ∗str1, const char ∗str2)

  Case-insensitive string compare.
- void initialize_CLI_inputs (CLI_inputs ∗cli)

  Initializes CLI_Inputs with default toggles and ballistic parameters.
- int parse_options (CLI_inputs ∗cli, int argc, char ∗argv[ ])

  Parses the command-line arguments for the solver.

12.24.1 Detailed Description

Definitions of functions for user input through the command line.
Provides a command-line interface for specifying muzzle velocity, angle, bullet mass, and advanced solver toggles like Magnus effect, bullet tilt, etc.

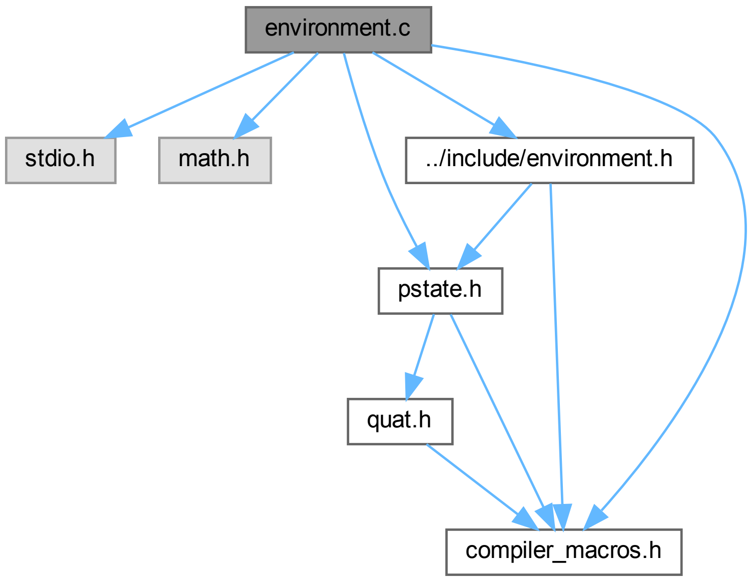
12.25 environment.c File Reference

Implements functions for atmospheric and environmental calculations.

```
#include <stdio.h>
#include <math.h>
#include "../include/environment.h"
#include "../include/pstate.h"
```

```
#include "../include/compiler_macros.h"
```

Include dependency graph for environment.c:



- #define PRECOMPUTED_EXPONENT (GRAVITY $*$ MOL_M_AIR) / (R_UNIV $*$ LAPSE_RATE)

  A constant factor used for exponent calculation in some atmospheric models.
- double saturation_vapor_pressure (double tK)

  Computes saturation vapor pressure (Pa) for a given temperature in Kelvin.
- static double compute_gamma (double mass_frac_h2o)

  Computes an effective ratio of specific heats $\gamma_{\text{effective}}$ given the fraction of water vapor in air.
- void update_environment (Environment $*$env, PState $*$state)

  Updates the environment based on projectile state (e.g., altitude).

12.25.1 Detailed Description

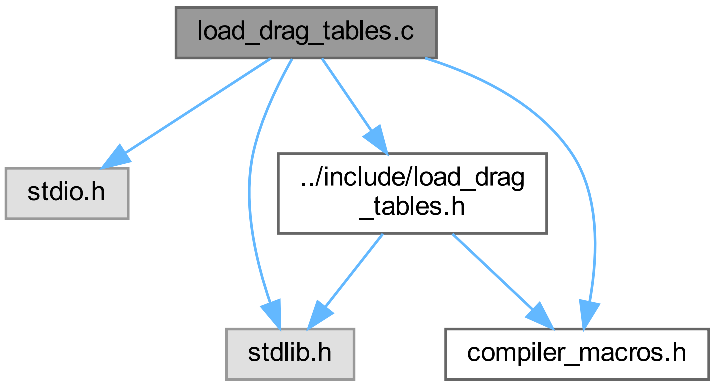
Implements functions for atmospheric and environmental calculations.
Integrated into the main loop described in core_6dof.

12.26 load_drag_tables.c File Reference

Contains functions to load and interpolate drag tables from CSV files.

```
#include <stdio.h>
#include <stdlib.h>
#include "../include/load_drag_tables.h"
#include "../include/compiler_macros.h"
```

Include dependency graph for load_drag_tables.c:



- #define MAX_TABLE_SIZE 500

  Max # of entries in the CSV table.
- #define LOOKUP_STEP 0.001

  Step for the finer resolution array.
- #define LOOKUP_MAX_MACH 5.0

  Maximum Mach # in the finer resolution array.
- #define LOOKUP_SIZE 5001

  $\frac{\text{LOOKUP\_MAX\_MACH}}{\text{LOOKUP\_STEP}} + 1$
- static DragEntry g_dragTable [MAX_TABLE_SIZE]
- double lookupTable [LOOKUP_SIZE]

  The precomputed finer array.
- static size_t g_tableSize = 0
- static int compare_drag_entries (const void ∗a, const void ∗b)

  Compare function for qsort, sorting DragEntry by Mach.
- size_t load_drag_table_CSV (const char ∗filename)

  Loads a CSV file of (Mach,Cd) pairs and sorts them by Mach.
- double interpolate_cd_from_table (double mach)

  Interpolates Cd for a given Mach # using linear search in g_dragTable[].
- COLD double fast_interpolate_cd (const double ∗restrict lookupTable, double mach)

  Fast interpolation from the finer resolution array for a given Mach.
- void precompute_drag_lookup_table (double ∗restrict lookupTable)

  Builds a finer-resolution array from the loaded table for fast lookup.
- const char ∗ get_drag_model_file (DragModel model)

  Returns the file path for each DragModel enumeration.

12.26.1 Detailed Description

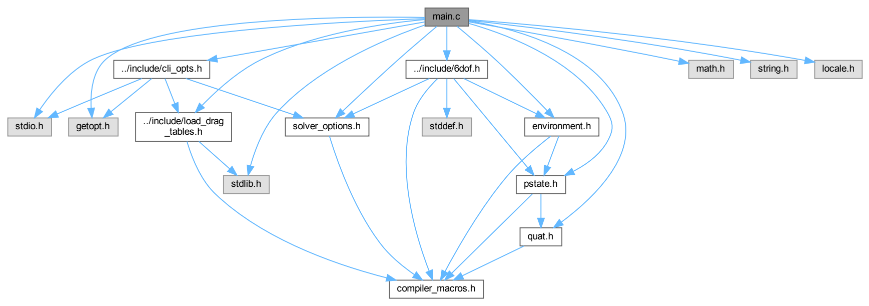
Contains functions to load and interpolate drag tables from CSV files.
These functions handle reading drag data from CSV, building a finer resolution array, and providing interpolation for Mach-based drag.

12.27 main.c File Reference

Prototype interface for the 6-DOF ballistic solver, now with actual drag table usage.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <getopt.h>
#include <locale.h>
#include "../include/6dof.h"
#include "../include/solver_options.h"
#include "../include/environment.h"
#include "../include/pstate.h"
#include "../include/quat.h"
#include "../include/load_drag_tables.h"
#include "../include/cli_opts.h"
```

Include dependency graph for main.c:



Macros

- #define BULLET_IS_AIRBORNE 1

  A macro for controlling the main simulation loop logic. Pure syntactic sugar.
- #define SEA_LVL 0.0

  Value for sea level in meters.
- #define MAX_TIME 60.0

  1 minute simulation time-limit
- #define GROUND_LEVEL 0.0

  y=0 means ground

Functions

- static const char ∗ get_drag_model_string (DragModel model)

  Returns a string name corresponding to the user's selected drag model.
- int main (int argc, char ∗argv[ ])

  The main entry point for the 6-DOF ballistic solver.

12.27.1 Detailed Description

Prototype interface for the 6-DOF ballistic solver, now with actual drag table usage.
This file:

- Loads a CSV drag table into a global array,
- Precomputes the finer-resolution lookup table,

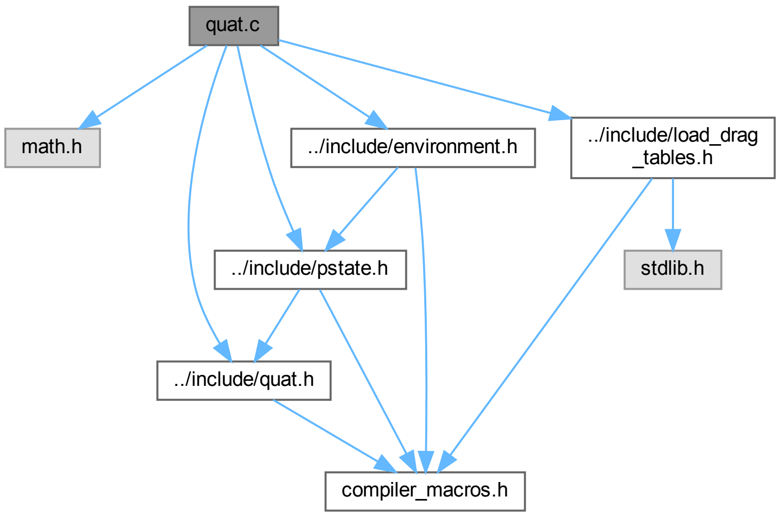
- Initializes the environment (weather, altitude, muzzle conditions),
- Initializes projectile specs (mass, inertia, aero, etc.),
- Runs the 6-DOF solver in a loop until the projectile hits the ground or times out.

12.28 quat.c File Reference

Implements quaternion-based rotation utilities.

```
#include <math.h>
#include "../include/quat.h"
#include "../include/pstate.h"
#include "../include/environment.h"
#include "../include/load_drag_tables.h"
```

Include dependency graph for quat.c:



Functions

- void quat_normalize (Quat ∗restrict q)

    Normalizes a quaternion to unit length.
- Quat quat_multiply (const Quat ∗q1, const Quat ∗q2)

    Multiplies two quaternions.
- Quat quat_conjugate (const Quat ∗restrict q)

    Computes the conjugate of a quaternion.
- void quat_rotate_vector (const Quat ∗restrict q, const double vect[3], double outVect[3])

    Rotates a 3D vector using the quaternion q.
- Quat quat_from_axis_angle (double axis_x, double axis_y, double axis_z, double angleRad)

    Creates a quaternion from axis-angle representation.
- Quat quat_add_scaled (const Quat ∗q, const Quat ∗dq, double scale)

    Adds `dq` scaled by `scale` to quaternion q.
- void quat_integrate_angular_velocity (Quat ∗restrict q, const double w[3], double dt)

    Integrates quaternion orientation by angular velocity over time dt.

12.28.1 Detailed Description

Implements quaternion-based rotation utilities.