SGUABA

TYPE-SAFE SPATIAL MATH IN RUST

Jon Gjengset

Amsterdam Rust meetup, July 2025

WHERE IS “THAT WAY”?

Describing position and direction in 3D space is... tricky.

Two primary considerations:

1. what is your frame of reference?
2. how does that frame of reference relate to others?

WHERE IS THE BALLOON?

Pilot spots a weather balloon out the window 30° up at 2 o'clock, at a distance of 150m.

Plane is flying north-northeast, and climbing at a 10° angle.

Plane is located at 59°54'17"N 10°41'05"E, 1km above sea-level.

WHERE IS THE BALLOON?

FRD

Pilot spots a weather balloon out the window 30° up at 2 o'clock, at a distance of 150m.

NED

Plane is flying north-northeast, and climbing at a 10° angle.

WGS84

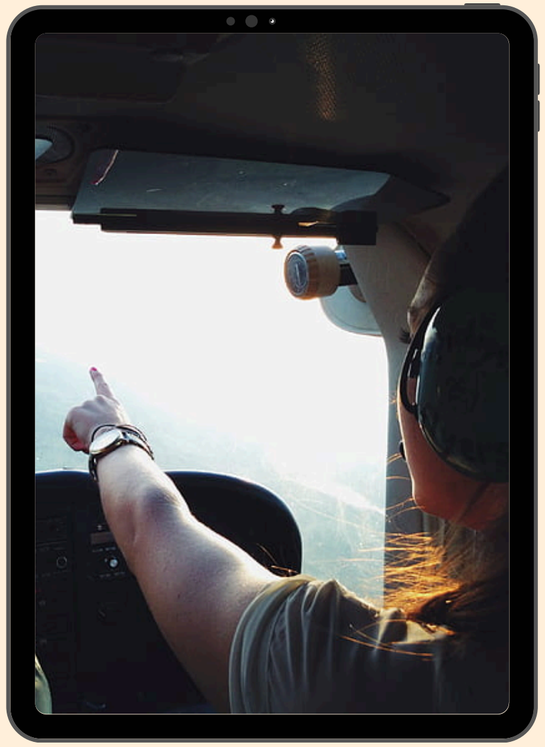
Plane is located at 59°54'17"N 10°41'05"E, 1km above sea-level.

LOCAL COORDINATE SYSTEMS

Describes coordinates relative to the body axes of some object (like a plane).

Front-Right-Down (FRD).
Common in aerospace.

Front-Right-Up (FRU).
Used in Unity.



SPATIAL REFERENCE SYSTEMS

Describes points relative to Earth.

Geographic or geodetic. Latitude + longitude + elevation (e.g., WGS84).

Geocentric. Cartesian coordinate system based on Earth's axes (e.g., ECEF).



LOCAL TANGENT PLANES

Describes coordinates relative to a tangent plane of a spatial reference system **at a particular point.**

North-East-Down (NED).
Common in aerospace.

East-North-Up (ENU).
Common in ground vehicles.



PUTTING THEM TOGETHER

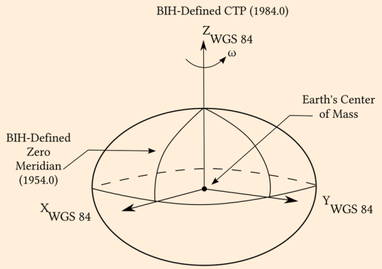
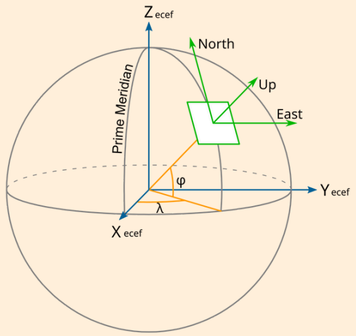
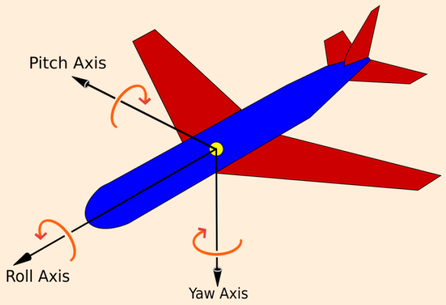


Figure 1.1 WGS 84 Reference Frame

Spatial reference



Tangent plane



Body reference

ROTATION + TRANSLATION = QUATERNION

IN CARTESIAN SYSTEMS

WGS84 IS NOT CARTESIAN

MUST USE ECEF FOR TRANSFORMATIONS

AXES CONVENTIONS

Left-handed. FRU.
Thanks Unity.

Right-handed. NED, ENU, FRD.
Thumb is X + roll.
Index is Y + pitch.
Middle is Z + yaw.

THEN THERE'S BEARING...

Also known as a **horizontal coordinate system.**
Bearing is azimuth (e.g., 2 o'clock) and elevation (e.g., 10° up), but **relative to what?**

... WHICH FOLLOWS NO RULE

In FRD, azimuth is on FR plane, clockwise from F.
In NED, azimuth is on NE plane, clockwise from N.
In ENU, azimuth is on EN plane, **clockwise from N.**
In FRU, azimuth is on FR plane, clockwise from F.

In **all four**, positive elevation is “up” from the plane.

MY HEAD HURTS.

EVEN IF YOU GET THE QUATERNIONS RIGHT,
HOW CONFIDENT ARE YOU IN REVIEW?

LET'S USE TYPES

SGUABA

From Celtic mythology; Sguaba Tuinne ("Wave-sweeper") was the self-navigating boat of Manannán mac Lir that was navigated solely by the thoughts of its pilot.



SGUABA: EVERY SYSTEM IS A TYPE

```
// the pilot observes things in FRD of the plane
system!(struct PlaneFrd using FRD);

// the pilot's instruments indicate the plane's orientation in NED
system!(struct PlaneNed using NED);

// note that there can be *multiple* FRDs and NEDs!
```

SYSTEM-APPROPRIATE BEARINGS

```
// what the pilot saw:
let balloon = Coordinate::<PlaneFrd>::from_bearing(
    Bearing::builder()
      // clockwise from forward
      .azimuth(Angle::new::<degree>(60.))
      // upwards from straight-ahead
      .elevation(Angle::new::<degree>(30.))
      .expect("elevation is in [-90°, 90°]")
      .build(),
    Length::new::<meter>(150.), // at this range
);
```

ORIENTATION IN YAW-PITCH-ROLL

```
// where the plane was facing at the time (eg, from instrument panel);
// expressed in yaw, pitch, roll relative to North-East-Down:
let orientation_in_ned = Orientation::<PlaneNed>::from_tait_bryan_angles(
    Angle::new::<degree>(22.5),  // yaw; north-northeast
    Angle::new::<degree>(10.),   // pitch; 10° climb
    Angle::new::<degree>(0.),    // roll
);
```

DIRECT SUPPORT FOR WGS84

```
// where the plane was at the time (e.g., from GPS):
let wgs84 = Wgs84::builder()
    .latitude(Angle::new::<degree>(59.5417))
    .expect("latitude is in [-90°, 90°]")
    .longitude(Angle::new::<degree>(10.4105))
    .altitude(Length::new::<meter>(1000.))
    .build();
```

THE WHOLE THING TYPE-CHECKS

```
// SAFETY: `wgs84` is the location of `PlaneNed`'s origin.
// we must go via ECEF as WGS84 isn't Cartesian and can't be directly transformed
let ecef_to_plane_ned = unsafe { RigidBodyTransform::ecef_to_ned_at(&wgs84) };

// SAFETY: the given NED orientation makes up the axes of `PlaneFrd`.
let plane_ned_to_plane_frd = unsafe { orientation_in_ned.map_as_zero_in::<PlaneFrd>() };

// this chaining would fail to compile if you got the arguments wrong!
let ecef_to_plane_frd = ecef_to_plane_ned.and_then(plane_ned_to_plane_frd);

// take our `Coordinate<PlaneFrd>` and produce a `Coordinate<Ecef>`:
let balloon_in_ecef = ecef_to_plane_frd.inverse_transform(balloon);

// we can then turn that into WGS84 lat/lon/altitude!
println!("{:?}", balloon_in_ecef.to_wgs84());
```

SHOULD IT BE UNSAFE?

Unsafe is traditionally for **memory** safety.
In Sguaba, unsafe operations can cause invalid transformations, which will violate **type** safety.

Thus, Sguaba's use of unsafe is non-idiomatic, but **extremely helpful** – it highlights the brittle code.
Other Sguaba code is unlikely to contain errors.

SGUABA

https://docs.rs/sguaba

Also includes:

- extensive docs, including primers
- an extra, “math focused” API
- macros for convenient (and safe) coordinate and vector construction
- extensive tests