Executive Summary

TL;DR

- **`CoordIJK`** is a **local coordinate** system in a hexagonal grid, relative to an origin.
- **`FaceIJK`** represents a point in the icosahedron-based grid system: it combines a **face number** (which of the 20 icosahedron faces you're on) with a `CoordIJK` that defines your position within that face.
- Together, these are used to map lat/lng positions to discrete hexagon indices in the H3 grid hierarchy.

Background: H3 Grid Structure

H3 maps Earth using a spherical icosahedron (20 triangular faces). Each face is projected onto a 2D plane and subdivided into hexagons.

H3 then builds hexagons hierarchically — resolution 0 is a coarse grid, and each higher resolution subdivides each hex into smaller hexes.

What is CoordIJK?

`CoordIJK` is a **local 2D coordinate system** for hexagons, based on a triple `(i, j, k)` such that:

i + j + k = some constant (typically, they represent relative steps)

It's similar to cube coordinates in hex grids:

- `i`, `j`, and `k` are **axes separated by 120°** in 2D.
- They can be converted to axial or offset coordinates for easier visualization.
- They allow **precise navigation** within the hex grid: moving from one hex to another is a simple vector addition.

In practice:

- When you're on a face, `CoordIJK(0, 0, 0)` is the center.
- `CoordIJK(1, 0, 0)` is one step in a certain direction, say "East".

- And so on for the other directions.

Think of `CoordIJK` as a local coordinate system that says "how far and in what direction from the center of this face am I?"

What is FaceIJK?

`FaceIJK` is:

- The **face number** (0 to 19, for each face of the icosahedron)
- A `CoordIJK` coordinate on that face

So it tells you:

> "I'm on face X, and at this IJK coordinate on that face."

This is the **intermediate representation** that helps convert:

1. `lat/lng → H3 index` (via FaceIJK first)
2. `H3 index → lat/lng` (also via FaceIJK)

How It's Used in H3 Internally

Almost every conversion in H3 (lat/lng ↔ H3 index) passes through `FaceIJK`:

- `geoToH3()`:
  - uses `geoToFaceIjk()` → returns `FaceIJK`
  - then encodes into an H3 index
- `h3ToGeo()`:
  - decodes H3 index → `FaceIJK`
  - then projects back to lat/lng

It's the **bridge between sphere and hex grid.**

Border Handling

When cells lie near the edge of a face:

- `CoordIJK` may land just *outside* the current face
- H3 has to **adjust**:
  - wrap across face edges
  - possibly switch to a neighboring face
  - reproject CoordIJK accordingly

So `FaceIJK` also lets H3 **track transitions** across face edges.

The 20 Faces Are Not the Final Grid — They're Just the Base

Think of the **icosahedron** as the **first level of approximation** of the sphere. Then:

- Each face gets **subdivided recursively** into finer hexagons.
- So, while there are only 20 starting faces, the hex grid can get very fine — H3 supports up to **resolution 15**, which gives ~1m^2 hexagons.

> The face is just a **container for local projection math** — not a limitation on spatial precision.

Why Only 20 Faces?

1. **Icosahedron** = **best tradeoff** between:
   - Low distortion
   - Manageable complexity
   - Uniform face size (almost equal-area triangles)
2. It supports **gnomonic projection** of Earth to 2D planes with reasonable angular distortion (better than cube or octahedron)
3. The small number of faces lets H3:
   - Use fast math for coordinate transforms
   - Store the face index in just **5 bits** (plenty for 0–19)

Resolution Scaling (How Finer Hexes Are Made)

Each time you go up a resolution:

- Each hex gets **subdivided into 7 children**
- So:
  - Res 0 = ~1100 km wide hexes
  - Res 5 = ~3 km
  - Res 10 = ~100 m
  - Res 15 = ~1 m

Thus, even though you're only working with 20 face planes, you can pinpoint locations on Earth to **fine-grained precision**, because everything happens in the **hex hierarchy built on top of the faces**.

Gnomonic Projection Distortion

A **gnomonic projection** projects the surface of a sphere onto a plane from the **center of the sphere**.

- This preserves **straight lines** (great circles become straight lines)
- But **distorts distances and areas**, increasingly so **farther from the face center**

What Happens in H3?

Each **icosahedron face** is:

- Projected from 3D (curved Earth) to 2D (flat triangle)
- A **hex grid is laid out in that 2D space** using uniform-sized hexagons

Then:

- When **projecting back to the sphere**, the hexagons near the **edges or corners of a face** get stretched
- But because the 2D grid assumes uniform size, this means:

On the Earth's surface, **hexagons near the face edges cover *less* or *more* real area** compared to those near the center

---

Hexes near face edges may:

- **Appear the same size in CoordIJK**
- But **cover slightly less or more surface area on Earth**
- Especially noticeable at **lower resolutions** (large hexes)

How H3 Handles It

1. **Accepts a small amount of distortion** as a trade-off
    - The distortion is bounded and predictable
2. **Edge-crossing logic** exists to handle:
    - Hexes that straddle face edges
    - Hexes that map to multiple possible locations (ambiguity resolution)
3. **High resolutions** (small hexes) reduce distortion effect in practice

---

Real-World Impact?

Resolution	Distortion Noticeable?	Area Variation
Low (0–2)	Yes	Up to ~20%
Mid (5–8)	Minor	Few %
High (10+)	Minimal	Sub-percent

For applications like coverage estimation, routing, clustering — H3's precision is generally *good enough*, and much more consistent than lat/lng grids.

Practical Example

Let's say you want to convert a lat/lng point to an H3 cell.

1. **Determine the closest face** (which face of the icosahedron it lands on)
2. **Project the lat/lng to a 2D coordinate** on that face (using gnomonic projection)
3. **Convert the projected point to CoordIJK** (grid steps away from the face's center)
4. **Store this as FaceIJK**
5. **Encode into the compact H3 index**

The reverse works similarly: decode the H3 index into FaceIJK, then reverse project back to lat/lng.

Why This Complexity?

Because Earth is a sphere, and hexagons don't tile a sphere perfectly, H3:

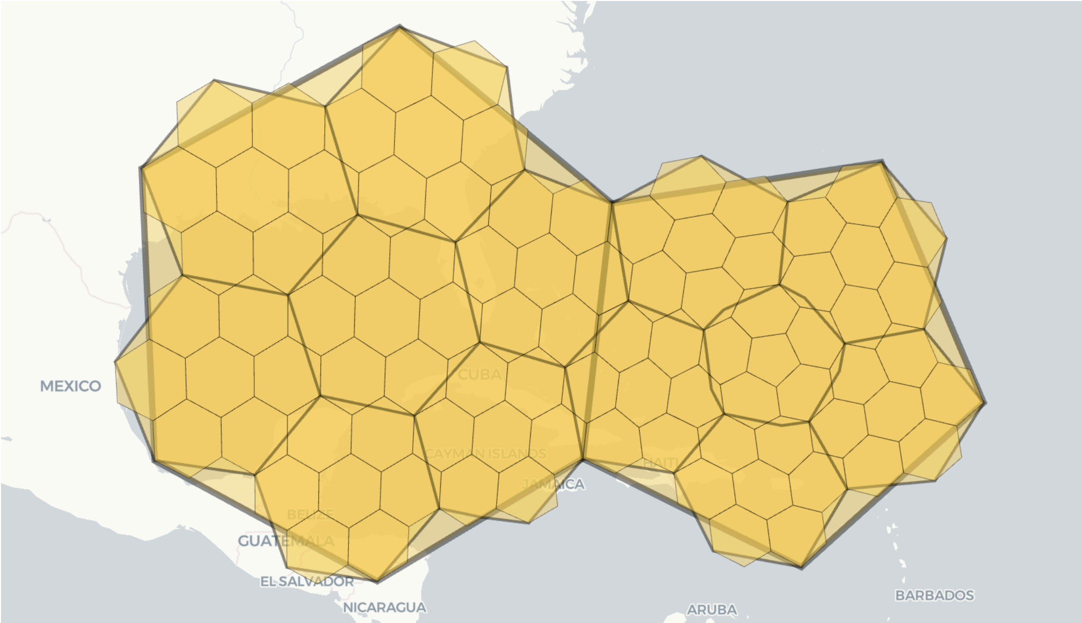
- Uses the icosahedron (20 triangular faces) to approximate the sphere
- Each face becomes a **flat 2D surface**
- Each surface gets a **hex grid**: local Euclidean math using `CoordIJK` within each face
- Encodes hierarchy and location compactly using these abstractions

---

Useful Links

- H3 Coordinate systems: https://h3geo.org/docs/core-library/coordsystems
- Red Blob Hexagonal Grids: https://www.redblobgames.com/grids/hexagons/

- Tables of Cell Statistics Across Resolutions: https://h3geo.org/docs/core-library/restable



The Red Blob bit

The **hex grid coordinate systems** that are conceptually the same as `CoordIJK`. Specifically:

- **Cube coordinates** (x, y, z) where `x + y + z = 0`
- This is equivalent to H3's `(i, j, k)` system
- H3 uses `(i, j, k)` instead of `(x, y, z)`, but it's the same 3-axis system with 120° between axes

So:

- **`CoordIJK(i, j, k)`** ≈ **`cube(x, y, z)`** from Red Blob
- The page is valuable to understand how directions, distances, and neighbors work in such a grid — even though it doesn't use H3's naming

---