# Cube, Hull-Free, Factors-Only Construction

Deriving all renderable coordinates and invariants from factors alone

Sir Robert Edward Grant

## Purpose (what this document proves)

We construct the regular cube in $\mathbb{R}^3$ *without* convex hulls, face inference, triangulation heuristics, symmetry-group lookup, or geometric "guessing".

Everything required to render the cube—a complete vertex list and a declared face list—is derived from a minimal factor set and explicit axioms, then verified computationally.

## Axiom 0: What counts as a "hull-free" construction

A construction is *hull-free* if:

- Vertices are produced by a closed algebraic rule.
- Faces are declared combinatorially (indices), not inferred from geometry.
- All numerical invariants (edge length, radii, angles) are derived and then verified.

## 1 Axiom 1: Primitive factor set for the cube

The cube is the Platonic solid with square faces. The only irrational factor forced by a square is

$$\boxed{\sqrt{2}}$$

because the diagonal of a unit square has length $\sqrt{2}$.

A cube additionally contains the space diagonal of a unit cube, which forces

$$\boxed{\sqrt{3}}$$

since the space diagonal of a unit cube is $\sqrt{3}$.

Thus the cube lives in the factor field generated by

$$\boxed{\mathcal{F}_{\text{cube}} = \{1, \sqrt{2}, \sqrt{3}\}.}$$

No $\varphi$ is required.

# 2 Axiom 2: Topological counts (purely combinatorial)

A cube has:

$$\boxed{V = 8, \quad E = 12, \quad F = 6}$$

and satisfies Euler's theorem:

$$\chi = V - E + F = 8 - 12 + 6 = 2.$$

# 3 Axiom 3: Local vertex structure forces orthogonality

At each vertex of a cube, exactly three edges meet, and each face angle is 90°. Therefore the three edge directions at a vertex must be mutually orthogonal.

## Derivation (no geometry inference)

Let $u, v, w \in \mathbb{R}^3$ be the direction vectors of the three edges incident to a vertex, each with the same length $\ell$:

$$\|u\| = \|v\| = \|w\| = \ell.$$

Because each pair of edges lies in a square face, the angle between any two incident edges is 90°, hence

$$u \cdot v = v \cdot w = w \cdot u = 0.$$

So $\{u, v, w\}$ forms an orthogonal triad.

# 4 Axiom 4: Coordinate system from orthogonal triad

Since $u, v, w$ are orthogonal and equal-length, we may choose coordinates so that

$$u = \ell\, e_1, \quad v = \ell\, e_2, \quad w = \ell\, e_3$$

where $e_1, e_2, e_3$ are the standard orthonormal basis vectors.

Now the cube's vertices are the endpoints of all combinations of half-steps along these axes.

# 5 Deriving the full vertex list (all steps shown)

## 5.1 Step 1: Put the cube center at the origin

Let the cube be centered at $0 \in \mathbb{R}^3$. Then vertices occur in opposite pairs $\pm p$.

### 5.2 Step 2: Let the half-edge offset be $a$

If the cube has edge length $\ell$, then from the center to any face is distance $\ell/2$. Let

$$\boxed{a = \frac{\ell}{2}.}$$

Each vertex is reached by choosing $\pm a$ along each of the three orthogonal axes:

$$(\pm a, \pm a, \pm a).$$

### 5.3 Step 3: Enumerate all $2^3 = 8$ sign combinations

The vertex set is therefore exactly

$$\boxed{\mathcal{V} = \{(a,a,a),\ (a,a,-a),\ (a,-a,a),\ (a,-a,-a),\ (-a,a,a),\ (-a,a,-a),\ (-a,-a,a),\ (-a,-a,-a)\}\,.}$$

This is a complete vertex list. Nothing else is needed to render the cube.

### 5.4 Step 4: Specialize to the canonical unit-edge cube

For $\ell = 2$, we have $a = 1$, giving the canonical coordinate generator

$$\boxed{\mathcal{V} = \{(\pm 1, \pm 1, \pm 1)\}.}$$

This is the minimal coordinate alphabet: only the factor 1 appears explicitly in the vertices.

## 6 Deriving all metric invariants from factors

### 6.1 Edge length from coordinates

Choose adjacent vertices differing in exactly one coordinate sign, e.g.

$$p_1 = (a,a,a), \quad p_2 = (-a,a,a).$$

Then

$$p_2 - p_1 = (-2a, 0, 0), \qquad \|p_2 - p_1\| = 2a = \ell.$$

So the coordinate system yields the desired edge length *by construction.*

### 6.2 Face diagonal (forces $\sqrt{2}$)

Across one square face:

$$p_1 = (a,a,a), \quad p_3 = (-a,-a,a).$$

Then

$$p_3 - p_1 = (-2a, -2a, 0), \quad \|p_3 - p_1\| = \sqrt{(2a)^2 + (2a)^2} = 2a\sqrt{2} = \ell\sqrt{2}.$$

So $\sqrt{2}$ is forced.

### 6.3 Space diagonal (forces $\sqrt{3}$)

Opposite vertices:

$$p_1 = (a, a, a), \quad p_4 = (-a, -a, -a).$$

Then

$$p_4 - p_1 = (-2a, -2a, -2a), \quad \|p_4 - p_1\| = \sqrt{3(2a)^2} = 2a\sqrt{3} = \ell\sqrt{3}.$$

So $\sqrt{3}$ is forced.

### 6.4 Circumradius and inradius

Circumradius $R$ is the distance from center to a vertex:

$$R = \|(a, a, a)\| = \sqrt{a^2 + a^2 + a^2} = a\sqrt{3} = \frac{\ell\sqrt{3}}{2}.$$

Inradius $r$ (center to a face) is:

$$r = a = \frac{\ell}{2}.$$

Thus

$$\boxed{R : r = \sqrt{3} : 1}$$

and the only irrationals required are $\sqrt{2}$ and $\sqrt{3}$.

## 7 Faces (declared, not inferred)

### Index the vertices

Let the eight vertices be ordered as:

$$\begin{array}{ll} 0 : (-a, -a, -a) & 1 : (-a, -a, a) \\ 2 : (-a, a, -a) & 3 : (-a, a, a) \\ 4 : (a, -a, -a) & 5 : (a, -a, a) \\ 6 : (a, a, -a) & 7 : (a, a, a) \end{array}$$

### Declare the 6 square faces

Each face is a 4-cycle of vertex indices (CCW as viewed from outside):

$$\boxed{\begin{array}{ll} F_0 = [0, 4, 6, 2] & (z = -a) \\ F_1 = [1, 3, 7, 5] & (z = +a) \\ F_2 = [0, 1, 5, 4] & (y = -a) \\ F_3 = [2, 6, 7, 3] & (y = +a) \\ F_4 = [0, 2, 3, 1] & (x = -a) \\ F_5 = [4, 5, 7, 6] & (x = +a) \end{array}}$$

No convex hull is used. If a face is not listed, it does not exist.

# 8 Comparison to classical known values (unit edge)

For a cube with $\ell = 1$:

$$a = \frac{1}{2}, \qquad R = \frac{\sqrt{3}}{2}, \qquad r = \frac{1}{2}, \qquad \text{face diagonal} = \sqrt{2}, \qquad \text{space diagonal} = \sqrt{3}.$$

| Quantity | Derived (factors-only) | Matches classical? |
|---|---|---|
| Edge length $\ell$ | 1 | ✓ |
| Half-edge $a$ | $1/2$ | ✓ |
| Face diagonal | $\sqrt{2}$ | ✓ |
| Space diagonal | $\sqrt{3}$ | ✓ |
| Circumradius $R$ | $\sqrt{3}/2$ | ✓ |
| Inradius $r$ | $1/2$ | ✓ |
| Euler $\chi$ | 2 | ✓ |

# 9 Python: factors-only cube construction, verification, and render

The code below:

- constructs vertices from the derived formula $(\pm a, \pm a, \pm a)$,
- declares faces exactly as above,
- verifies edge lengths and radii numerically,
- renders the cube *without* any hull.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from itertools import product

EPS = 1e-9

def cube_vertices(edge_length=1.0):
    # Derived: a = ell/2
    a = edge_length / 2.0
    # Derived: vertices are all sign combinations of (\pm a, \pm a, \pm a)
    # Order chosen to match the face declaration in the paper:
    # 0..7 correspond to (-,-,-), (-,-,+), (-,+,-), (-,+,+), (+,-,-), (+,-,+), (+,+,-), (+,+,+)
```

```
    V = np.array([
        [-a, -a, -a],
        [-a, -a,  a],
        [-a,  a, -a],
        [-a,  a,  a],
        [ a, -a, -a],
        [ a, -a,  a],
        [ a,  a, -a],
        [ a,  a,  a],
    ], dtype=float)
    return V

def cube_faces():
    # Declared faces (no inference)
    return [
        [0,4,6,2],  # z = -a
        [1,3,7,5],  # z = +a
        [0,1,5,4],  # y = -a
        [2,6,7,3],  # y = +a
        [0,2,3,1],  # x = -a
        [4,5,7,6],  # x = +a
    ]

def unique_edges_from_faces(F):
    edges = set()
    for face in F:
        m = len(face)
        for i in range(m):
            a = int(face[i])
            b = int(face[(i+1) % m])
            e = (a,b) if a < b else (b,a)
            edges.add(e)
    return sorted(edges)

def verify_cube(V, F, edge_length=1.0, verbose=True):
    # Check Euler
    Eset = unique_edges_from_faces(F)
    chi = V.shape[0] - len(Eset) + len(F)

    # Check edges
    lens = []
    for (i,j) in Eset:
        lens.append(np.linalg.norm(V[j] - V[i]))
    lens = np.array(lens)

    # Check radii
    radii = np.linalg.norm(V, axis=1)
```

```
    R_num = radii.max()
    r_num = np.min(np.abs(V[:,0]))  # distance to x=0 faces is |x| = a

    # Expected
    a = edge_length / 2.0
    R_exp = np.sqrt(3) * a
    r_exp = a

    ok_edges = np.all(np.abs(lens - edge_length) < 1e-9)
    ok_chi = (chi == 2)
    ok_R = abs(R_num - R_exp) < 1e-9
    ok_r = abs(r_num - r_exp) < 1e-9

    if verbose:
        print("=== CUBE VERIFICATION (HULL-FREE) ===")
        print("V,E,F:", V.shape[0], len(Eset), len(F), "Euler chi =", chi)
        print("Edge length min/max:", lens.min(), lens.max(), "expected", edge_length)
        print("Circumradius R:", R_num, "expected", R_exp)
        print("Inradius r:", r_num, "expected", r_exp)
        print("STATUS:",
              "OK OK" if (ok_edges and ok_chi and ok_R and ok_r) else "FAIL FAIL")

    return ok_edges and ok_chi and ok_R and ok_r

def render(V, F, title="Cube (declared faces, no hull)", outfile=None):
    fig = plt.figure(figsize=(6,6), dpi=200)
    ax = fig.add_subplot(111, projection="3d")

    polys = [V[np.array(face, dtype=int)] for face in F]
    coll = Poly3DCollection(polys, alpha=0.35, edgecolor="k", linewidths=1.0)
    ax.add_collection3d(coll)

    m = np.linalg.norm(V, axis=1).max() * 1.2
    ax.set_xlim([-m, m]); ax.set_ylim([-m, m]); ax.set_zlim([-m, m])
    ax.set_box_aspect((1,1,1))
    ax.set_axis_off()
    ax.set_title(title, pad=10)

    plt.tight_layout()
    if outfile:
        plt.savefig(outfile, bbox_inches="tight")
```

```
        print("Saved:", outfile)
        plt.close(fig)
    else:
        plt.show()

if __name__ == "__main__":
    ell = 1.0
    V = cube_vertices(edge_length=ell)
    F = cube_faces()
    assert verify_cube(V, F, edge_length=ell, verbose=True)
    render(V, F, outfile="cube_declared.png")
```

# Conclusion

The cube is fully renderable from the factor-derived coordinate rule $(\pm a, \pm a, \pm a)$ with $a = \ell/2$ and a declared 6-face index list. All invariants reduce to $\{1, \sqrt{2}, \sqrt{3}\}$ and match classical values exactly. No convex hull is required at any stage.