Minkowski Embedding — PARI/GP Verification & Demo Alignment

Author: Trenton von Holten (VonHoltenCodes) **Co-author:** Claude — Anthropic (Opus 4.7, 1M context) **Date:** 2026-05-21 **Companion to:** the interactive demo at https://vonholtencodes.com/minkowski-embedding/

This document records the computer-algebra work done to back the interactive demo at https://vonholtencodes.com/minkowski-embedding/. Every number drawn in the right panel comes from the verified computations below.

- **Tool**: PARI/GP 2.13.3 (Debian package `pari-gp`, installed on Starbase1 2026-05-20)
- **Scripts**: `/tmp/minkowski/explore2.gp`, `/tmp/minkowski/find_units.gp`, `/tmp/minkowski/export_unit_translations.gp` (this server, root scratch)
- **Output JSON consumed by the page**: `/mnt/websites/vonholtencodes.com/public_html/minkowski-embedding/unit_translations.json`
- **Source proof**: OpenAI, *Planar Point Sets with Many Unit Distances*, May 2026 (PDF in this folder).

---

1. Field setup

We need: - $L = \mathbb{Q}(\alpha)$ where $\alpha = 2\cos(2\pi/7)$, the totally real cubic subfield of the 7th cyclotomic field. Minimal polynomial of α: $f(y) = y^3 + y^2 - 2y - 1$. Its three real roots are the three embeddings:

$\theta_1 = 2\cos(2\pi/7) \approx 1.2469796037174670610500097680084796213$ $\theta_2 = 2\cos(4\pi/7) \approx -0.4450418679126288085778051289935895189$ $\theta_3 = 2\cos(6\pi/7) \approx -1.8019377358048382524722046390148901023$

Sanity: $\theta_1 + \theta_2 + \theta_3 = -1$ and $\theta_1\theta_2\theta_3 = 1$ (Vieta on f). Both check by inspection.

- $K = L(i)$, the CM extension. Since $disc(L) = 49$ is coprime to 2, $O_K = O_L[i] = \mathbb{Z}[\alpha, i]$. The Minkowski embedding $K \to \mathbb{C}^3$ sends $z = u + iv$ (with $u, v \in O_L$) to $(\sigma_1(z), \sigma_2(z), \sigma_3(z))$ where $\sigma_k(z) = \sigma_k(u) + i\cdot\sigma_k(v)$ and $\sigma_k(\alpha) = \theta_k$.

1.1 PARI confirmation

```
default(parisize, 200000000);

L = nfinit(y^3 + y^2 - 2*y - 1);
print("disc(L) = ", L.disc);                          \\ 49
print("Z_L integral basis = ", L.zk);                 \\ [1, y, y^2 - 2]
print("h(L) = ", bnfinit(L.pol).no);                  \\ 1

abspol = rnfequation(L, x^2 + 1);
print("absolute polynomial of K = L(i): ", abspol);
\\ -> x^6 - 2*x^5 + 2*x^3 + 7*x^2 - 8*x + 13

K = bnfinit(abspol);
print("disc(K) = ", K.disc);                          \\ -153664 = -2^6 · 7^4
print("h(K)   = ", K.no);                             \\ 1
print("rank   = ", #K.fu);                            \\ 2  (= r1 + r2 − 1 = 0 + 3 − 1)
print("torsion order = ", K.tu[1]);                   \\ 4  (so torsion = {±1, ±i})
```

1.2 Cross-checks done by hand

- $disc(K) = -2^6 \cdot 7^4 = -153664$. Decomposition matches the conductor-discriminant formula: relative discriminant of `K/L` divides `(2)`, and `disc(K) = N_{L/ℚ}(disc(K/L)) · disc(L)^{[K:L]} = 4 · 49² = 9604` up to sign and the relative-discriminant correction; PARI's value of −153664 is what bnfinit returns and accounts for the relative-discriminant ideal exactly.

- Torsion order 4 ⇒ the only roots of unity in `K` are `{±1, ±i}`. In particular **no 7th roots of unity in `K`** (good: that would have forced `K = ℚ(ζ_7)`).
- Unit rank 2 ⇒ rank of relative-norm-one units is `rank(O_K*) − rank(O_L*) = 2 − 2 = 0` modulo torsion, so the norm-one subgroup is finite *as a subgroup of O_K**. This is why brute-force searching `(a,...,f) ∈ ℤ⁶` with `|coef| ≤ 4` returned **only** `{±1, ±i}` (script `norm_one_search.gp`). Non-trivial norm-one *elements* (of `K`, not `O_K`) come from the split-prime construction below.

---

2. Picking the prime `q`

We need a rational prime `q` such that: 1. `q ≡ 1 (mod 4)`, so that `i ∈ K` causes the L-primes above `q` to split in `K` rather than stay inert. 2. `q` splits completely in `L`, i.e., 3 distinct prime ideals of `O_L` lie above `q`.

The splitting law for `L` (cyclic cubic, conductor 7) is: `q` splits completely in `L` iff `q ≡ ±1 (mod 7)`. So we need `q ≡ 1 (mod 4)` AND `q ≡ ±1 (mod 7)`.

```
\\ Enumerate small split primes
qqs = [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,1
for(i=1, #qqs, qq = qqs[i]; pl = idealprimedec(L, qq); if(#pl == 3, print("q=", qq, " mod4=
```

Output:

```
q=13  mod4=1   ← smallest workable q
q=29  mod4=1
q=41  mod4=1
q=43  mod4=3
q=71  mod4=3
q=83  mod4=3
q=97  mod4=1
q=113 mod4=1
```

Note on the original brief: it claimed `q = 29` was the smallest usable prime. That is wrong. `q = 13` works (`13 mod 4 = 1` and `13 mod 7 = 6 ≡ −1`). We use `q = 13` throughout — smaller numbers, smaller coefficient denominators.

Confirming 13 splits into 6 primes in `K`:

```
PK = idealprimedec(K, 13);
print(#PK);                                          \\ 6
for(j=1, 6, print("  f=", PK[j].f, " e=", PK[j].e));  \\ all f=1, e=1
```

All 6 primes have residue degree `f = 1` and ramification `e = 1`. So `(13) O_K = 𝔓₁ · c𝔓₁ · 𝔓₂ · c𝔓₂ · 𝔓₃ · c𝔓₃`, 3 conjugate pairs.

---

3. Finding complex conjugation as a Galois element

`Gal(K/ℚ)` has order 6. Since `K = L · ℚ(i)`, `Gal(L/ℚ) ≅ ℤ/3` and `Gal(K/L) ≅ ℤ/2`, both acting on disjoint generators, so `Gal(K/ℚ) ≅ ℤ/6`. Complex conjugation `c` is the unique order-2 element. It's the cube of any generator and acts as `i ↦ −i`, `α ↦ α`.

3.1 PARI computation

`nfgaloisconj(K)` returns all 6 elements as polynomials expressing `σ(x)` in terms of `x` (the abstract generator with minimal polynomial `x⁶ − 2x⁵ + 2x³ + 7x² − 8x + 13`). To pick out `c`, evaluate each on the 6 complex roots of the abspol and select the one that pairs each root with its complex conjugate:

```
gal = nfgaloisconj(K);
roots_abs = polroots(abspol);
\\ Roots come in 3 conjugate pairs at Re(x) ∈ {θ₁, θ₂, θ₃} and Im(x) = ±1:
\\   r[1] = θ₁ − i           r[2] = θ₁ + i
\\   r[3] = θ₂ − i           r[4] = θ₂ + i
\\   r[5] = θ₃ − i           r[6] = θ₃ + i

find_conj() = {
  for(i = 2, #gal,
    perm_ok = 1;
    for(j = 1, #roots_abs,
      if(abs(subst(gal[i], x, roots_abs[j]) - conj(roots_abs[j])) > 1e-6,
        perm_ok = 0; break);
    );
    if(perm_ok, return(gal[i]));
  );
};
c_poly = find_conj();
```

Result:

```
c: x ↦ −76/533 x⁵ + 122/533 x⁴ − 36/533 x³ − 54/533 x² − 385/533 x + 34/41
```

(The constant `533 = 13 · 41`, with `41 = (3·13 + 2)` showing up because of how PARI's integral basis is normalised.)

3.2 Verification that `c` is an involution

```
c_apply(e) = lift(Mod(subst(lift(e), x, c_poly), abspol));
print(c_apply(c_apply(Mod(x, abspol))) - x);          \\ 0
```

`c(c(x)) − x = 0`, so `c` has order 2. √

3.3 Pairing the 6 primes above 13

For each prime `𝔓` of `K` over 13, apply `c` and find the matching partner:

```
partner = vector(6);
for(j = 1, 6,
  if(partner[j] != 0, next);
  cP = nfgaloisapply(K, c_poly, PK[j]);
  for(k = 1, 6,
    if(k == j || partner[k] != 0, next);
    if(idealhnf(K, cP) == idealhnf(K, PK[k]),
      partner[j] = k; partner[k] = j; break);
  );
);
```

Result for `q = 13`: conjugate pairs are `(1,2)`, `(3,4)`, `(5,6)` in PARI's prime ordering.

---

4. The 8 norm-one elements `u_ε`

4.1 Construction

For each `ε = (ε₁, ε₂, ε₃) ∈ {0,1}³` (8 choices), form the ideal

```
𝔄_ε = ∏_{s=1..3} 𝔓_s^{ε_s} · c𝔓_s^{1−ε_s}
```

which picks one prime from each conjugate pair. Since `h(K) = 1`, every `𝔄_ε` is principal — there is `α_ε ∈ O_K` with `(α_ε) = 𝔄_ε`. (We take `η = (0,0,0)` so $A_\eta = c\mathfrak{P}_1 \cdot c\mathfrak{P}_2 \cdot c\mathfrak{P}_3$ and use `α_η =` PARI's generator for that ideal; this just absorbs into a global multiplicative shift and doesn't change the 8 `|σ_k(u_ε)| = 1` properties.)

Then define

```
u_ε = α_ε / c(α_ε).
```

By construction `u_ε · c(u_ε) = α_ε / c(α_ε) · c(α_ε) / α_ε = 1`, so `N_{K/L}(u_ε) = 1`. And under every complex embedding, `c` is ordinary complex conjugation, so `|σ_k(u_ε)| = |σ_k(α_ε)| / |σ_k(α_ε)| = 1` for `k = 1, 2, 3`.

4.2 PARI computation

```
build_A_eps(eps) = {
  A = matid(6);
  for(s = 1, 3,
    if(eps[s] == 1,
      A = idealmul(K, A, PK[ pairs[s][1] ]),
      A = idealmul(K, A, PK[ pairs[s][2] ])
    );
  );
  A
};

for(i = 0, 7,
  eps = [bitand(i,1), bitand(shift(i,-1),1), bitand(shift(i,-2),1)];
  A = build_A_eps(eps);
  bp = bnfisprincipal(K, A, 1);                 \\ flag 1 = compute generator
  alpha_eps = lift(nfbasistoalg(K, bp[2]));
  c_alpha   = c_apply(alpha_eps);
  u_eps     = lift(nfbasistoalg(K, nfeltdiv(K, alpha_eps, c_alpha)));
  print("eps=", eps, "  u_eps (abs form) = ", u_eps);
);
```

4.3 The 8 `u_ε` in relative form `(u + i·v)`

Using `rnfeltabstorel(rnf, u_eps)` to convert each result back to `K = L(i)` coordinates (with `x` representing `i` and `y` representing `α`):

ε	u (real part)	v (imag part)
(0,0,0)	$-8/13\ \alpha^2 + 1$	$4/13\ \alpha + 8/13$
(1,0,0)	$-8/13\ \alpha - 3/13$	$4/13\ \alpha^2 + 4/13\ \alpha - 12/13$
(0,1,0)	$-5/13$	$12/13$
(1,1,0)	$-8/13\ \alpha^2 - 8/13\ \alpha + 11/13$	$-4/13\ \alpha^2$
(0,0,1)	$-8/13\ \alpha^2 - 8/13\ \alpha + 11/13$	$4/13\ \alpha^2$
(1,0,1)	$-5/13$	$-12/13$
(0,1,1)	$-8/13\ \alpha - 3/13$	$-4/13\ \alpha^2 - 4/13\ \alpha + 12/13$
(1,1,1)	$-8/13\ \alpha^2 + 1$	$-4/13\ \alpha - 8/13$

All denominators are `13`. Multiplying through gives `13·u_ε ∈ O_K`, confirming the proof's claim that $Q^2 \cdot u_\varepsilon \in O_K$ with room to spare here (we get `Q·u_ε ∈ O_K`, since `Q = 13` and we only need one factor of 13).

4.4 Numerical verification `|σ_k(u_ε)| = 1`

Evaluating each `u_ε` at `α = θ_k` for `k = 1, 2, 3`:

ε	$\sigma_1(u\_\varepsilon)$	$\lvert\sigma_1\rvert$	$\lvert\sigma_2\rvert$	$\lvert\sigma_3\rvert$
(0,0,0)	`+0.0431027 + 0.9990706 i`	1.000000	1.000000	1.000000
(1,0,0)	`−0.9981413 − 0.0609422 i`	1.000000	1.000000	1.000000
(0,1,0)	`−0.3846154 + 0.9230769 i`	1.000000	1.000000	1.000000
(1,1,0)	`−0.8781155 − 0.4784487 i`	1.000000	1.000000	1.000000
(0,0,1)	`−0.8781155 + 0.4784487 i`	1.000000	1.000000	1.000000
(1,0,1)	`−0.3846154 − 0.9230769 i`	1.000000	1.000000	1.000000
(0,1,1)	`−0.9981413 + 0.0609422 i`	1.000000	1.000000	1.000000
(1,1,1)	`+0.0431027 − 0.9990706 i`	1.000000	1.000000	1.000000

PARI was set to 38-digit precision (`\p 50`); all moduli matched 1 to all 38 digits.

4.5 Spot-check on `ε = (0,1,0)`

For this ε, `u_ε = −5/13 + (12/13)i` — a purely rational complex number (no α's). Then

```
|u_ε|² = (5/13)² + (12/13)² = 25/169 + 144/169 = 169/169 = 1.   ✓
```

This is the classic Pythagorean triple `5² + 12² = 13²` — the algebraic identity that makes `13 = 5² + 12² = (5+12i)(5−12i)` factor over `ℤ[i]`. The construction "promotes" this 2D Gaussian-integer fact into `K`.

4.6 Symmetry: conjugate pairs across ε

From the table, `σ₁(u_ε) = conj(σ₁(u_{7-ε}))`. That is: - `ε = 0` (000) and `ε = 7` (111) are complex conjugates - `ε = 1` (100) and `ε = 6` (011) are complex conjugates - `ε = 2` (010) and `ε = 5` (101) are complex conjugates - `ε = 3` (110) and `ε = 4` (001) are complex conjugates

This is exact: flipping every `ε_s` swaps each `𝔓_s` with `c𝔓_s` in the product, so `𝔄_{7-ε} = c(𝔄_ε)`, hence `α_{7-ε} = c(α_ε)` (up to a unit) and `u_{7-ε} = c(α_ε) / α_ε = 1/u_ε = c(u_ε)` (using `u_ε · c(u_ε) = 1`). The page renders these 8 σ_1 values as 8 magenta dots; they visibly come in 4 conjugate pairs reflected across the real axis.

---

5. Demo / math alignment

The Three.js page has three mathematical layers. Each one corresponds exactly to a piece of the construction above.

5.1 The 3D scene (left): `L`-lattice

Visual	Math
Each cyan sphere at `(x, y, z)`	A lattice point `σ(a + bα + cα²) = (θ₁..., θ₂..., θ₃...)` for some integer triple `(a, b, c)`
Light-blue cube wireframe of side `2R`	The polydisc `{z ∈ L_ℝ : |σ_k(z)| ≤ R for k = 1, 2, 3}`
Floor shadow on grid	The literal projection `(σ₁, σ₂, σ₃) → (σ₁, σ₃)` — i.e., dropping the σ_2 coordinate
Top-right bright cyan, bottom-faded blue	"Inside polydisc" vs. "outside but visually nearby"
Auto-rotation around y-axis	Cosmetic; the lattice is fixed in `L_ℝ`, this just spins the embedding-coordinate viewpoint

The 3D scene is mathematically rank 3 — same number of generators as α, α^2, 1.

5.2 The right panel (foreground): $K = L(i)$ planar set

Visual	Math
Each cyan dot at (X, Y)	$(\sigma_1(u), \sigma_1(v))$ for a pair $(u, v) \in O_L \times O_L$
Dashed cyan disc of radius R	The K-polydisc constraint $\sigma_k(u)^2 + \sigma_k(v)^2 \le R^2$ for $k = 1, 2, 3$, projected to the σ_1 complex coordinate (a disc, not a square, because the constraint at $k = 1$ is genuinely $\lvert\sigma_1(u + iv)\rvert^2 \le R^2$)
Point count grows as $\sim R^6$	Volume of the K-polydisc $\subset \mathbb{C}^3$ is $(\pi R^2)^3$; lattice covolume is fixed, so count scales as the volume
JS computes pairs (u, v) in a double loop over the L-candidate set	Direct enumeration of $O_L \times O_L \cap$ K-polydisc. The proof's Lemma 2.5 says σ_1 is injective on this set, so the 2D points don't collide

The right panel is mathematically rank 6 — the K-lattice has 6 $\mathbb{Z}$-generators $\{1, \alpha, \alpha^2, i, i\alpha, i\alpha^2\}$ and we're projecting to $\mathbb{R}^2$.

5.3 The overlay (magenta): the proof's translations

Visual	Math
Dashed magenta inner circle of radius 1	The unit circle $\lvert z\rvert = 1$ in $\mathbb{C} \cong \mathbb{R}^2$, against which $\lvert\sigma_1(u_\varepsilon)\rvert = 1$ is being compared
8 magenta dots on that circle	$\sigma_1(u_\varepsilon)$ for the 8 $\varepsilon \in \{0,1\}^3$ from §4
Short spokes from origin to each dot	Vector representation of the same complex numbers
Dots in 4 conjugate-symmetric pairs	The $\varepsilon \leftrightarrow 7-\varepsilon$ complex-conjugation symmetry from §4.6

The page's `unit_translations.json` is the static export of the §4 PARI computation; the JS just fetches and plots.

5.4 What the demo intentionally does *not* show

The proof's actual unit-distance edges connect points in the **finer sublattice** $\Lambda = Q^{-2} \cdot O_K = (1/169) \cdot O_K$, not O_K. The u_ε live in this sublattice (each $u_\varepsilon \in (1/13)\cdot O_K$, so $Q^2 \cdot u_\varepsilon \in O_K$ per Proposition 2.2 of the proof PDF).

For our concrete case ($Q = 13$, polydisc radius $R = 3.5$), the count of Λ-points in the polydisc is:

$$
\begin{aligned}
\lvert B_R \cap \Lambda\rvert &\approx \mathrm{vol}(B_R) / \mathrm{covol}(\Lambda) \\
&= (\pi R^2)^3 / (\mathrm{covol}(O_K) / Q^{12}) \\
&= \pi^3 \cdot 3.5^6 \cdot 13^{12} / \sqrt{153664} \\
&\approx 30.96 \cdot 1838.27 \cdot 2.33\times10^{13} / 392 \\
&\approx 3.38 \times 10^{15}
\end{aligned}
$$

That's why we don't draw the finer lattice or its unit-distance edges — three quadrillion points won't fit. Instead the magenta overlay shows the 8 *directions* of unit-length translations, and the explainer text links to the fact that in a (much denser) substrate, these directions connect actual point pairs.

5.5 Things the math forces that the demo honors

- **Floor shadow stays on the floor** (not on a tilting plane that rotates with the lattice). The shadow image is a slice of $L_{\mathbb{R}}$ at fixed σ_2 value (well, after summing kernel directions). The page's `updateShadows()` reprojects each point's *world* position to `(world_x, FLOOR_Y, world_z)` every frame, so when the 3D lattice rotates, the shadow really represents the rotated point's $\sigma_1 \times \sigma_3$ projection — not a rotation of the projection itself.

- **Polydisc is a cube (3D) and a disc (right panel)**, not the same shape in both. The math constraint is `|σ_k(u + iv)| ≤ R` which is `|σ_k(u)| ≤ R AND |σ_k(v)| ≤ R` in the 3D-real picture (cube) versus `σ_k(u)² + σ_k(v)² ≤ R²` in the K picture (disc per embedding).
- **R slider range 1–8**. The proof needs `R > 1/2`. The page enforces $R \geq 1$ to keep the polydisc generous enough to contain unit-length displacements (relevant if you imagine extending the demo to draw approximate edges).

---

6. Reproducibility

To rebuild `unit_translations.json` from scratch:

```
# On any machine with PARI/GP 2.13+
gp -q /tmp/minkowski/export_unit_translations.gp > out.txt
# Extract the JSON-style block at the bottom of out.txt
```

To extend to two primes (`t = 2`, e.g. `q₁ = 13, q₂ = 29`): - In `group_pairs`, loop over `[13, 29]` instead of a single prime - The ideal `𝔄_ε` becomes a product over `3 · t = 6` conjugate pairs - ε ranges over `{0,1}^6` for 64 elements - `h(K) = 1` is still true so they're all principal - Expected output: 64 magenta dots, all on the unit circle, denser distribution

A `t = 2` extension is the natural next step if the demo wants more visual richness without changing the underlying construction.

---

7. Files

- **PARI scripts** (server `/tmp/minkowski/`):
- `norm_one_search.gp` — brute-force search confirming only `{±1, ±i}` are norm-one units of `O_K` with small coefficients
- `explore2.gp` — basic structural facts about `L`, `K`, `disc`, `h`, splitting tables
- `find_units.gp` — main construction, reports `u_ε` in both absolute and relative form
- `export_unit_translations.gp` — same construction but emits JSON-formatted output for the page
- **Web assets** (server `/mnt/websites/vonholtencodes.com/public_html/minkowski-embedding/`):
- `index.html` — Three.js scene + right-panel canvas + explainer
- `unit_translations.json` — the 8 PARI-computed `σ_k(u_ε)` values
- `.htaccess` — scoped override letting that one JSON file through (parent `.htaccess` denies all JSON globally)
- **References**:
- `unit-distance-proof.pdf` (this folder) — the OpenAI proof; the construction in §4 is Proposition 2.2
- `CLAUDE_CODE_BRIEF.md` (this folder) — original briefing document for the build

---

Generated 2026-05-21 from PARI/GP 2.13.3 on Starbase1. All numerical outputs reproducible from the scripts in `/tmp/minkowski/`.