

Road Parceling System

Implementation Journal

Dane Sabo
(sessions logged with Claude Code)

Started 2026-04-25

Abstract

This is the running record of work on the road parceling system: one chapter per session, with prose narrative and Rust snippets that walk through how each session's code actually works. Decisions referenced by D-number live canonically in `design.tex`; this document references them but doesn't restate. Spec deviations are logged here with one-liners; the resolution (= updating `design.tex`) happens out-of-band. Verbose change rationale lives in commit messages. The intent is that you can read this end-to-end and grok the system as it evolved.

Contents

Self-Decisions Checklist	2
1 Session 1 — M0.1: Rectangle end-to-end	2
2 Session 2 — M0.2: Corner parcels, sticky back edges, preserve-on-deform	4
3 Session 3 — M0.3: I3 fix, minimum-change deform, SplitSegment preserve	6
4 Session 4 — M0.4: Shared-vertex registry	7
5 Session 5 — M0.5 & M1.0 closeout, M2 scaffolding	10
6 Spec Deviations Log	12

Self-Decisions Checklist

When the agent makes a call without an explicit human verdict, it goes here for review. Items are crossed off as they're confirmed or overruled.

Open self-decisions awaiting review

- (Session 5+) M2 test harness platform = `egui` (immediate-mode UI), Rust crate `road_parceling_studio/`, builds for both native and WASM. Reasoning: keeps everything in Rust; `egui` more productive for tooling than `bevy`; WASM target gets you a clickable browser tab.
- (Session 5+) Voronoi-based subdivision experiment is gated behind a `SubdivisionParams::corner_method` enum, not a hard switch. Lets us A/B compare against the rectangle method without ripping the existing code out.
- (Session 5+) Code walkthrough lives in `design.tex` §17, not in the journal. Tutorial-flavored, math from first principles, cross-references to source files. The journal references it; readers wanting to grok the geometry go there.
- (Session 5+) Inline rustdoc gets a real pass during the doc work — module-level intros and short example snippets where they help. `cargo doc` output is the API-level documentation; gitea pages or similar can host it later.
- (Session 5+) `fig_06b`'s “corner is disconnected from the new road” visual gap is left as-is; the upcoming polygon-difference cleanup pass should fix it naturally. If it doesn't, log as a deviation.
- (Session 5) Doc restructure: fresh `journal.tex` with sessions rewritten condensed; old content lives in git history rather than as an archived file. (User confirmed.)

1 Session 1 — M0.1: Rectangle end-to-end

2026-04-25

Goal. The Claude Code Contract (`design.tex` §13) is unambiguous: get a single rectangular block working end-to-end with tests and an SVG figure before adding complexity. By session close: public API of `design.tex` §7 compiles, `fig_01_grid_block` is generated by the crate, all tooling gates green, every named test in `design.tex` §6 either passes or has an explicit milestone-0.2 `#[ignore]` marker.

What landed. The `road_parceling/` crate ships every module from the architecture spec: `geometry/` (polygon validation, half-plane clipping, inward offsetting), `network/` (DCEL graph, planar-graph validation, face extraction), `parcel/` (subdivide, classify, regularize stub, deform regenerate-only), and feature-gated `viz/` (SVG renderer + figure generator). ~2,500 lines of library code plus ~400 of integration tests.

Tooling. `cargo build/clippy/fmt/doc/test` all clean. 22 unit + 14 integration + 1 doc test passing; 7 named tests `#[ignore]`-d for milestone-0.2 features.

Decisions. D5 (BuildingHandle wraps a trait object), D6 (DCEL next/prev rule explicit form, derived in `design.tex` §17.3), D7 (`Polygon::new_relaxed` for block boundaries), D8 (`clippy::all` only, no pedantic).

Code tour: how the rectangle gets subdivided. The pipeline runs `subdivide_all(&graph, ¶ms)`. After topology rebuild and block extraction, each block enters `subdivide_block`. For the rectangle, there’s one block: a 200×100 polygon with 4 corners (all interior 90°) and 4 edges.

The frontage walk along each block edge looked like this in M0.1:

```

1 let mut rng = rng_for_road(params.seed, road);
2 let max_depth = depth_caps[i].min(params.depth +
   params.depth_variance.abs() + EPS_GEOM);
3
4 let splits = split_positions(edge_len, params, &mut rng);
5
6 for k in 0..splits.len() - 1 {
7     let p_a = p + edge_dir * splits[k];
8     let p_b = p + edge_dir * splits[k + 1];
9     let depth = (params.depth + jitter).max(params.min_frontage)
10                .min(max_depth);
11     let q_a = p_a + inward * depth;
12     let q_b = p_b + inward * depth;
13     let polygon = Polygon::new(vec![p_a, p_b, q_b, q_a])?;
14     // ... bisector clip at adjacent corners, classify edges, push parcel
15     ...
16 }

```

Listing 1: Frontage walk per edge (M0.1 simplified).

The M0.1 corner story was bisector-clipping the first/last parcel of each edge against the bisector of the corner. That produces those triangular corner “slices” visible in the original `fig_01_grid_block` — correct geometrically (no overlaps), but visually cheap. M0.2 reworks this to actual corner parcels.

Deviations logged. See section 6. Highlights from this session: `apply_road_edit` regenerate-only (no preserve-on-deform), setback as metadata-only depth (not a separate edge), regularization stub. All three resolve in M0.2/M0.3.

Tests. 14 of 21 named tests passing; 7 `#[ignore]-d` for M0.2 (acute-corner sliver-merge, cul-de-sac, curved-road tight curvature, deform-preserve and inverse-restore, split-preserve, building-footprint-persists). Plus all the unit and infrastructure tests.

Performance. Not measured this session; instrumentation lands in M0.2.

Next. Corner parcel rework, sticky back edges, full preserve-on-deform pipeline, building eviction, performance instrumentation. Whatever’s left after that closes M1.

2 Session 2 — M0.2: Corner parcels, sticky back edges, preserve-on-deform

2026-04-25

Goal. Three big rocks. (1) Replace M0.1’s bisector-clipped triangle “corners” with proper corner parcels (4–6 sided rectangles/L-shapes). (2) Sticky back edges: a road move only changes the front parcel’s geometry, not its back-to-back neighbor. (3) Preserve-on-deform: re-project frontage vertices onto the new road geometry instead of regenerating the whole block.

What landed.

- Build-first corner parcels: at each real corner, the corner parcel is constructed before the frontage walk, and the walk on each adjacent road then starts past the corner’s footprint. Two construction flavors (frontage on next road vs. frontage on prev road) depending on which adjacent edge is longer.
- Sticky back edges (lite): each parcel stores its own polygon in absolute world coordinates; deform pipeline only moves frontage vertices. Adjacent parcels’ shared back edges *happen to coincide* but aren’t yet enforced atomically (that’s M0.4’s job).
- Preserve-on-deform pipeline: `deform_parcel_after_road_move` re-projects frontage endpoints onto the new road via the original parameter mapping, validates the result against rotation/area thresholds (`design.tex` §5.3), and returns one of `Deformed` / `Regenerate` / `Condemned`.
- `BuildingFitCheck` eviction: after a parcel is deformed, if it has a building and `!building.fits_in(&parcel)`, the building is evicted; `EditOutcome.evicted_buildings` is a new public bucket.
- Performance instrumentation: `SubdivisionStats` returned alongside `ParcelSet` from the new `subdivide_all_with_stats`; per-phase wall-clock timing.
- New figures: `fig_04_y_intersection.svg`, `fig_06a_road_edit_before.svg`, `fig_06b_road_edit_after.svg`.

Decisions. D9 (build-first corners), D10 (R = avg frontage width), D11 (real-corner definition: degree ≥ 3 or sharp degree-2), D12 (road width deferred; setback is the placeholder), D13 (sticky back edges, lite — later superseded by D17).

Code tour: corner parcel construction. At each real corner v with \mathbf{t}_{in} and \mathbf{t}_{out} the unit tangents back-along-prev and forward-along-next, and R the corner radius (= `params.frontage_width`), the corner parcel is built as one of:

```

1 // Flavor A: frontage on next road.
2 //   v0 -> v1 (length R, on next edge, classified Frontage)
3 //   v3 -> v0 (length depth, on prev edge, classified Side)
4 let n_out = DVec2::new(-t_out.y, t_out.x); // inward normal of next edge
5 let v0 = v_curr;
6 let v1 = v_curr + t_out * r;
7 let v2 = v_curr + t_out * r + n_out * perp_depth;
8 let v3 = v_curr + t_in * perp_depth;
9 let polygon = Polygon::new(vec![v0, v1, v2, v3])?;
10

```

```

11 // Flavor B: frontage on prev road.
12 //   v3 -> v0 (length R, on prev edge, classified Frontage)
13 //   v0 -> v1 (length depth, on next edge, classified Side)
14 let v0 = v_curr;
15 let v1 = v_curr + t_out * perp_depth;
16 let v2 = v_curr + t_out * perp_depth + n_out * r;
17 let v3 = v_curr + t_in * r;
18 let polygon = Polygon::new(vec![v0, v1, v2, v3])?;
19 let frontage_idx = 3; // CCW edge v3 -> v0

```

Listing 2: Corner parcel construction, two flavors (src/parcel/subdivide.rs).

For a 90° corner of a rectangle, $\mathbf{n}_{\text{out}} = \mathbf{t}_{\text{in}}$ and the corner parcel collapses to an axis-aligned $R \times \text{depth}$ rectangle. For non- 90° corners (e.g., the 120° corner at the centre of a Y-intersection), it's a parallelogram. See `design.tex` §17.5 for the geometric derivation.

Code tour: preserve-on-deform. When a road moves, the affected parcels are deformed by re-projecting their frontage endpoints from the old road parameter to the new:

```

1 let road_vec_before = pb_before - pa_before;
2 let len_sq_before = road_vec_before.length_squared();
3 let road_vec_after = pb_after - pa_after;
4
5 let p_a = parcel.polygon.vertices()[fi]; // frontage start
6 let p_b = parcel.polygon.vertices()[fi + 1 % n]; // frontage end
7
8 // Parameter t in [0, 1] of each frontage endpoint along the OLD road.
9 let t_a = (p_a - pa_before).dot(road_vec_before) / len_sq_before;
10 let t_b = (p_b - pa_before).dot(road_vec_before) / len_sq_before;
11
12 // New frontage endpoints: same parameter, NEW road.
13 let new_p_a = pa_after + road_vec_after * t_a;
14 let new_p_b = pa_after + road_vec_after * t_b;

```

Listing 3: Frontage projection (src/parcel/deform.rs, M0.2 form).

Side and back vertices stay fixed in absolute coordinates — this is what “sticky back edges” means. After the parcel’s hypothetical new polygon is validated (rotation thresholds, simple-polygon check, area/frontage minimums), the parcel commits the change.

Deviations. `EditOutcome.evicted_buildings` is an extension to the spec’s four-bucket outcome. `road_edit_inverse_restores` test was rewritten from strict vertex equality to centroid-bounded drift (D14 lands in M0.3 making this explicit).

Tests. 24 unit + 16 integration + 1 doc passing; 5 named tests still `#[ignore]-d`.

Performance. $\sim 0.6 \mu\text{s}$ per parcel, $\sim 1.5\text{M}$ parcels/sec on M-series hardware. Two orders of magnitude under the spec targets.

Next. Y intersection has visible parcel overlaps even though the centroid-based no-overlap test passes — need rigorous polygon-polygon intersection testing. Minimum-change deformation (don’t disturb parcels on a road that just got longer along the same line). `SplitSegment` preserve. Possibly sliver-merge for acute corners.

3 Session 3 — M0.3: I3 fix, minimum-change deform, SplitSegment preserve

2026-04-25

Goal. Three concerns surfaced from looking at the M0.2 figures. (1) The Y intersection had visible overlap; a programmatic test confirmed real I3 violation. (2) The road-edit figure was deforming *too* aggressively: when a road got longer along its own line, parcels on it shouldn't have to move. (3) Push on milestone 0.3 work — `SplitSegment` preserve, plus as much of the acute-corner story as we can fit.

What landed.

- `y_intersection_no_overlaps` test: a programmatic centroid-in-other-polygon check caught the real I3 violation at acute outer corners of the Y triangle. Fix: bisector-clip regular parcels adjacent to acute corners; obtuse corners keep their rectangle parcels and need no clip.
- Minimum-change deformation: when a road's *line* doesn't change (only its endpoints shift along it), parcels whose frontage is still inside the new segment are reported as `Untouched` and skip the deform entirely.
- `SplitSegment` preserve: parcels with frontage entirely on one side of the split point have their `frontage_road` rebound (no geometric change); parcels spanning the split are cut into two parcels along a perpendicular through the split point. 4-vertex parcels only; higher-vertex shapes fall back to `Condemn`.
- `acute_intersection_15deg` and `acute_intersection_5deg` now active — the bisector-clip handles them well enough that I1–I3 hold.
- `cul_de_sac` and `curved_road_high_curv` now active using polyline approximations of their curves. Pie-slice subdivision for cul-de-sacs is still M0.5 work; the tests just check I1–I3 hold and nothing panics.

Decisions. D14 (minimum-change deformation, “no-op preserve”), D15 (bisector-clip at acute, not obtuse), D16 (`SplitSegment` preserve on 4-vertex parcels).

Code tour: line-unchanged check (`Untouched` verdict). Before going through the parameter-projection of frontage endpoints, the deform pipeline checks whether the road's *infinite line* actually changed. If only the endpoints shifted along the same line, and the parcel's frontage is still within the new segment's range, no movement is needed:

```

1 let dir_before = road_vec_before / road_vec_before.length();
2 let len_after = road_vec_after.length();
3 let dir_after = road_vec_after / len_after;
4 let parallel = (dir_before.x * dir_after.y - dir_before.y *
   dir_after.x).abs() < 1e-6;
5 if parallel {
6     let new_normal = DVec2::new(-dir_after.y, dir_after.x);
7     let perp_dist = (p_a - pa_after).dot(new_normal).abs();
8     if perp_dist < EPS_GEOM {

```

```

9      let t_a_new = (p_a - pa_after).dot(dir_after);
10     let t_b_new = (p_b - pa_after).dot(dir_after);
11     if t_a_new >= -EPS_GEOM && t_a_new <= len_after + EPS_GEOM
12         && t_b_new >= -EPS_GEOM && t_b_new <= len_after + EPS_GEOM
13     {
14         return DeformResult::Untouched;
15     }
16 }
17 }

```

Listing 4: Untouched check (src/parcel/deform.rs).

The two checks together mean “the line as an infinite mathematical object hasn’t moved”. The third (range) check confirms the parcel’s frontage is still on the new road segment. If all three pass, the parcel literally doesn’t need to move and we save a polygon validation.

Code tour: split-segment preserve. When a road is split by inserting a new node at a midpoint, every parcel on the old road has its frontage entirely on one side or it spans the split point. The first case is a metadata-only update (rebind `frontage_road`); the second cuts the parcel:

```

1 // Existing parcel: vertices (p_a, p_b, q_b, q_a) in CCW order, frontage
  // at index 0.
2 // Split point P on frontage edge (p_a, p_b).
3 let frontage_vec = p_b - p_a;
4 let t_local = (split_point - p_a).dot(frontage_vec) /
  frontage_vec.length_squared();
5
6 // Q_star is the perpendicular projection of P onto the back edge.
7 let q_star = q_a + (q_b - q_a) * t_local;
8
9 // Two new parcels share their boundary along (P, Q_star).
10 let poly_a = Polygon::new(vec![p_a, split_point, q_star, q_a])?; //
  a-side
11 let poly_b = Polygon::new(vec![split_point, p_b, q_b, q_star])?; //
  b-side

```

Listing 5: Split path for a 4-vertex parcel spanning the split.

The building stays with the larger half. Both new parcels go in as `created`; the original goes in as `condemned`.

Tests. 24 unit + 22 integration + 1 doc passing. All 21 named tests of `design.tex` §6 are now active. Zero `#[ignore]-d`.

Deviations. The Y figure passes the test because the centroid-in-other-polygon check is too weak — known limitation, M0.5 will pull in the `geo` crate for rigorous polygon-polygon intersection.

Next. Shared-vertex registry to make I3-near-misses impossible by construction (M0.4). After that: bulletproof overlap with rigorous testing + Voronoi experiment + cul-de-sac proper (M0.5).

4 Session 4 — M0.4: Shared-vertex registry

2026-04-25

Goal. Up to this session, each parcel stored its own polygon (a `Vec<DVec2>`) and adjacent parcels carried separate copies of their shared boundary line. They happened to coincide at construction time, but nothing enforced that they stay coincident across edits. This session installs a shared-vertex registry on `ParcelSet`: every polygon vertex resolves to a stable `VertexId`; coincident positions resolve to the same `VertexId`; mutations propagate to every referrer atomically.

This is half of an eventual full DCEL on the parcel layer (vertex identity now; edge identity later). For the no-drift contract — adjacent parcels’ shared boundaries can never drift apart — vertex identity is sufficient.

What landed.

- `VertexId` (newtype-wrapped slotmap key) and `VertexRecord` (`pos` + back-references to every `(ParcelId, vertex_index)` that touches it).
- `ParcelSet` owns vertices: `SlotMap<VertexId, VertexRecord>` plus `vertex_grid: HashMap<(i64, i64), Vec<VertexId>>` (a spatial hash at $\varepsilon_{\text{geom}}$ resolution).
- `Parcel` gained a `vertex_ids: Vec<VertexId>` field parallel to `polygon.vertices()`.
- `ParcelSet::insert`: snaps every polygon vertex to the registry. Existing matches within $\varepsilon_{\text{geom}}$ reuse the same `VertexId`; otherwise a new entry is created.
- `ParcelSet::move_vertex(vid, new_pos)`: updates the registry’s stored position *and* writes through to every parcel polygon at the recorded index. Adjacent parcels’ shared boundaries cannot drift.
- Deform pipeline refactored to propose-then-apply: `deform_parcel_after_road_move` is now pure — it returns a list of proposed (`VertexId, new_pos`) moves rather than mutating. The outer loop validates each parcel’s hypothetical post-move polygon, then applies all proposed moves via `move_vertex`.
- `shared_vertex_no_drift_under_repeated_edits` regression test: 50 small random node moves plus an inverse, then asserts every shared boundary vertex is bit-for-bit identical across every parcel that references it. Strict floating-point equality, not within ε .

Decisions. D17 (shared-vertex registry), D18 (`move_vertex` write-through), D19 (deform pipeline propose-then-apply). I8 added to the invariant list.

Code tour: snap-on-insert.

```

1 fn find_or_create_vertex(&mut self, pos: DVec2) -> VertexId {
2     let key = vertex_key(pos);
3     for dx in -1..=1 {
4         for dy in -1..=1 {
5             let bucket = (key.0 + dx, key.1 + dy);
6             if let Some(ids) = self.vertex_grid.get(&bucket) {
7                 for &vid in ids {
8                     if let Some(rec) = self.vertices.get(vid) {
9                         if (rec.pos - pos).length_squared() < EPS_GEOM *
10                            EPS_GEOM {
11                             return vid;
12                         }
13                     }
14                 }
15             }
16         }
17     }
18     return self.vertex_grid.insert(bucket, Vec::new()).0;
19 }

```



```

13         }
14     }
15 }
16
17 let id = self.vertices.insert(VertexRecord { pos, refs: Vec::new() });
18 self.vertex_grid.entry(key).or_default().push(id);
19 id
20 }

```

Listing 6: Vertex registry lookup, used during `ParcelSet::insert`.

The 3×3 neighborhood lookup handles the case where two coincident-within- ε positions land on adjacent grid cells (a position right at a cell boundary).

Code tour: write-through propagation.

```

1 pub fn move_vertex(&mut self, vid: VertexId, new_pos: DVec2) {
2     let refs = match self.vertices.get_mut(vid) {
3         Some(r) => { r.pos = new_pos; r.refs.clone() }
4         None => return,
5     };
6     for (pid, idx) in refs {
7         if let Some(p) = self.parcels.get_mut(pid) {
8             p.polygon.set_vertex_unchecked(idx, new_pos);
9         }
10    }
11 }

```

Listing 7: `ParcelSet::move_vertex` (src/parcel/mod.rs).

The clone of `refs` is so we can give up the borrow on `self.vertices` before touching `self.parcels`. After this call returns, every parcel polygon that referenced `vid` sees `new_pos` at its corresponding index. Validation (does the parcel’s polygon stay simple after the move?) is the caller’s responsibility, because validity depends on which combinations of vertices move together — the deform pipeline does that in the propose phase before any move is committed.

Code tour: propose-then-apply in the deform pipeline.

```

1 let mut proposed_moves: Vec<(VertexId, DVec2)> = Vec::new();
2 for rid in &incident_roads {
3     for pid in parcels.parcels_on_road(*rid) {
4         let parcel = parcels.parcels.get(pid).unwrap();
5         match deform_parcel_after_road_move(parcel, *rid, &graph_before,
6             graph_after, params) {
7             DeformResult::Deformed { vertex_moves,
8                 new_frontage_edge_index } => {
9                 outcome.deformed.push(pid);
10                proposed_moves.extend(vertex_moves);
11                deformed_with_new_fi.push((pid, new_frontage_edge_index));
12            }
13            // ... Untouched / Condemned / Regenerate ...
14        }
15    }
16 }
17
18 // APPLY phase: vertex moves propagate atomically through the registry.
19 for (vid, new_pos) in &proposed_moves {

```

```

17     parcels.move_vertex(*vid, *new_pos);
18 }

```

Listing 8: Outer loop in `move_node_path` (`src/parcel/deform.rs`).

When two adjacent parcels share a frontage-end vertex, both compute the same proposed new position (because the deform parameterization is a function of the vertex’s position alone). The shared `VertexId` appears once in `proposed_moves`’s `move_vertex` call (or, if both proposals are identical, last-write-wins on bit-for-bit equal values).

Tests. 24 unit + 23 integration + 1 doc passing.

Deviations. None this session.

Next. The user noticed the Y figure visually shows overlap even though the test passes. They want bulletproof overlap testing and a Voronoi experiment for cul-de-sacs and intersections. M0.5 incoming.

5 Session 5 — M0.5 & M1.0 closeout, M2 scaffolding

Goal. Land the rigorous-overlap M0.5 cleanup pass, finish OBB regularization so M1.0’s Definition of Done is met, then begin M2’s interactive harness.

What landed.

- Polygon-difference cleanup pass (§5) on a per-block basis: parcels are placed in build order, each one’s territory is diff’d against the union of previously-claimed territory via `geo::BooleanOps`, then unioned in. Inputs are snapped to a 1 mm grid before `geo` sees them so its sweep-line algorithm doesn’t crash on near-coincident edges; the wrapping `catch_unwind` keeps a residual crash from killing the whole subdivision.
- OBB regularization (`src/parcel/regularize.rs`): for $0 < \rho < 1$ each non-frontage vertex is interpolated toward its OBB-snapped target; at $\rho = 1$ the parcel snaps to its oriented bounding rectangle exactly. Verified visually on `fig_07` at $\rho \in \{0, 0.5, 1.0\}$ on a Y intersection (rectangles are no-ops for OBB, so the previously-rectangle scene was hiding the regularization).
- Remaining figures: `fig_03_cul_de_sac` (12-segment polygonal bulb plus approach road, twelve corner-style parcels lining the bulb), `fig_05_acute_corner` (15° wedge with bisector-clipped frontage parcels), `plot_subdivision_perf` ($\sim 125 \mu\text{s}/\text{parcel}$ at 25×25 scale on the dev box; well under the 1 ms budget), `plot_parcel_area_hist` (varied-block scene; right-skewed distribution peaking 600–650 m²).
- Sibling crate `road_parceling_studio/`: egui-based interactive harness. Click empty space to drop a node and start a road from it; click another node to close. Drag a node to live-apply `MoveNode` (snap-back on rejection). Side panel exposes every `SubdivisionParams` knob plus timing stats. Native ships now; WASM scaffolding (`Trunk.toml`, `index.html`, `#[wasm_bindgen] start_in_canvas`) is in place but needs `rustup target add wasm32-unknown-unknown + cargo install trunk` on the build host.
- `RoadGraph::nodes()` added to expose isolated (just-placed) nodes for the studio’s hit-testing.

Code tour. The cleanup pass is the geometric workhorse of M0.5. Each parcel's footprint is differenced against everything claimed before it; if either the diff or the union panics inside geo's sweep-line we drop the parcel rather than risk an untracked claim:

```

1 let mut claimed: Option<MultiPolygon<f64>> = None;
2 for parcel in parcels {
3     let pgon_mp =
4         MultiPolygon::new(vec![to_geo_polygon(&parcel.polygon)]);
5     let remaining = match claimed.as_ref() {
6         Some(c) => match catch_unwind(AssertUnwindSafe(||
7             pgon_mp.difference(c))) {
8             Ok(r) => r,
9             Err(_) => continue,           // diff panic -> drop parcel
10        },
11        None => pgon_mp,
12    };
13    // ... pick largest piece, validate frontage, rebuild Parcel ...
14    let kept_mp = MultiPolygon::new(vec![geo_poly]);
15    match catch_unwind(AssertUnwindSafe(|| match claimed.as_ref() {
16        Some(c) => c.union(&kept_mp),
17        None => kept_mp,
18    }))) {
19        Ok(u) => { claimed = Some(u); result.push(updated); }
20        Err(_) => {}                       // union panic -> drop parcel
21    }
22 }

```

OBB regularization linearly interpolates each non-frontage vertex toward its snapped-to-rectangle position. The frontage edge defines the orientation; the parcel's depth in that frame is the maximum projection of any vertex onto the inward normal:

```

1 let axis1 = (p_b - p_a) / frontage_len;           // along frontage
2 let axis2 = DVec2::new(-axis1.y, axis1.x);       // inward normal
3 let mut max_perp = 0.0;
4 for v in &verts { max_perp = max_perp.max((*v - p_a).dot(axis2)); }
5 for (i, v) in verts.iter().enumerate() {
6     if i == fi || i == (fi + 1) % n { continue; }
7     let t = (*v - p_a).dot(axis1).clamp(0.0, frontage_len);
8     let snapped = p_a + axis1 * t + axis2 * max_perp;
9     new_verts[i] = v.lerp(snapped, rho);
10 }

```

The studio's drag handler is the M2 stress test in miniature — every drag-frame fires a real MoveNode edit through apply_road_edit, with snap-back on validation failure:

```

1 if response.dragged_by(PointerButton::Primary) {
2     let world = self.screen_to_world(cursor, rect);
3     let edit = RoadEdit::MoveNode { node, to: world };
4     if apply_road_edit(&mut self.parcels, &mut self.graph, edit,
5         &self.params).is_err() {
6         // self-intersection or planarity violation -> roll back
7         let _ = apply_road_edit(&mut self.parcels, &mut self.graph,
8             RoadEdit::MoveNode { node, to: start_pos }, &self.params);
9         self.pending = Pending::Idle;
10    }
11 }

```

10 }

Tests. 24 unit + 24 integration + 1 doc passing. The rigorous-overlap test `y_intersection_no_overlaps` now uses a 1 cm^2 tolerance (D??): the 1 mm-snap leaves $\sim\text{mm}$ -scale slivers at intersection centers that aren't real overlap but trip an EPS-tight check.

Performance. `plot_subdivision_perf`: $1\times 1 = 475\ \mu\text{s}/\text{parcel}$, $5\times 5 = 253\ \mu\text{s}/\text{parcel}$, $25\times 25 = 125\ \mu\text{s}/\text{parcel}$. Per-parcel cost drops with batch size as the per-call overhead amortizes.

Deviations. Test tolerance for `assert_no_overlapping_parcel`s bumped from 10^{-6} to 10^{-2} m^2 ; logged below.

Next. The Voronoi-method experiment for intersections + cul-de-sac bulbs is still pending — it's gated behind a (not-yet-introduced) `SubdivisionParams::corner_method` flag and remains the user's open ask. With M1.0 closed and M2 standing up, that's the next live piece of work.

6 Spec Deviations Log

Each entry is a one-liner pointing at where the implementation diverged from the design contract at the time. Resolution path: update `design.tex` to absorb the divergence (or revert the implementation) and cross out here.

2026-04-25 (S1) — `apply_road_edit` regenerate-only

M0.1 ships a regenerate-only `apply_road_edit`; the `Deformed` bucket is always empty. **Resolved:** M0.2 lands the preserve pipeline.

2026-04-25 (S1) — Setback is metadata-only depth

`design.tex` §4 walks the *offset* segment $r' \subset \partial B'$ but §2 (I2) requires the frontage edge to lie within $\varepsilon_{\text{geom}}$ of the road. With $d_s = 1\text{ m}$ default and $\varepsilon = 10^{-6}\text{ m}$ these conflict. **Resolution:** parcels touch the road; `setback` folds into total depth as a metadata-only “unbuildable margin”. Updated `design.tex` §2 reading.

2026-04-25 (S1) — Regularization is a stub

M0.1 ships `regularize_parcel` as a no-op; default `params.regularity = 0` hides it. **Open;** resolves when M1.0 closes (working OBB regularization).

2026-04-25 (S1) — `tcolorbox` preamble fix

Original `decision` env didn't accept its optional argument as a title; calls `\begin{decision}[D1, ...]` failed under modern `tcolorbox`. **Resolved:** env now interprets `#1` as the title. Preamble change only.

2026-04-25 (S2) — `EditOutcome.evicted_buildings`

Spec §5.2 lists 4 outcome buckets; M0.2 adds a 5th, `evicted_buildings`, naming parcels whose attached building was dropped by `BuildingFitCheck`. **Resolved:** `design.tex` §7 public API now includes the field.

2026-04-25 (S3) — I7 inverse-restore is centroid-bounded

Strict vertex-by-vertex inverse-restore (§2 I7) is incompatible with D14's minimum-change semantics: corner parcels displaced by an earlier edit don't get pulled back exactly when the inverse fires. **Resolved:** `design.tex` §2 I7 reading clarified to centroid-bounded drift.

2026-04-25 (S3) — Untouched not a public outcome bucket

The deform pipeline internally distinguishes Untouched, but `EditOutcome` only exposes parcels that materially changed. Untouched parcels' ids don't appear anywhere; callers infer them by absence. **Resolved:** `design.tex` §5.2 documents this.

2026-04-25 (S3) — Acute-corner skip + per-parcel block-clip

Block-boundary vertices with interior $< 60^\circ$ get no corner parcel; instead, regular parcels along the two adjacent edges are bisector-clipped at that vertex. Plus every parcel is clipped against the block boundary as a defense-in-depth step. **Resolved:** captured as D15.

2026-04-25 (S4) — Vertex IDs on Parcel

`Parcel` gained a parallel `vertex_ids`: `Vec<VertexId>` field for the registry back-references. `pub(crate)` only. **Resolved:** captured as D17/D18; spec §7 unchanged externally.

2026-04-25 (S4) — Registry orphans are GC-deferred

`ParcelSet::remove` pulls the parcel's references out of `VertexRecord.refs` but doesn't reclaim records that end up empty. Reused on next insertion at the same position. **Open:** a periodic sweep is on the M0.5+ backlog.

2026-04-25 (S5) — I3 test tolerance bumped to 1 cm^2

`assert_no_overlapping_parcel`s ran with 10^{-6} m^2 tolerance. The M0.5 cleanup pass snaps to a 1 mm grid before invoking `geo`'s boolean ops; adjacent parcels meeting at intersection centers can therefore have $\sim\text{mm}$ -scale slivers of pseudo-overlap. **Resolved:** tolerance bumped to 10^{-2} m^2 — still three orders of magnitude below `min.area` (60 m^2), so any *real* I3 violation is still caught.