Road Parceling System

A Design Journal

Dane Sabo

Started 2026-04-25

Abstract

This journal documents the design and implementation of a road-frontage-based parcel subdivision system, intended as the foundational geometric layer of a city simulation game. The system rejects the rigid grid-based zoning of conventional city builders in favor of arbitrary polygonal parcels drawn outward from road frontage, with explicit handling for parcel persistence under road edits. This document serves three purposes: (1) a record of design decisions and their justifications, (2) a specification precise enough to drive autonomous implementation, and (3) a running notebook for open questions, dead ends, and revisions. It is intended to be read and extended over the lifetime of the project.

Contents

1 Project Context and Motivation

1.1 Why Build This

Modern city simulation games suffer from two architectural choices that compound poorly. First, they tend toward rigid grid-based or cell-based zoning, which produces visually uniform cities that diverge from how real urban form develops parcel by parcel along road frontage. Second, they over-rely on bottom-up agent simulation: every citizen is an autonomous decision-maker rerolling actions on every tick. This scales badly — *Cities: Skylines II* is the canonical example of a game shipped with simulation costs that do not survive contact with a real player's city.

This project addresses the first problem directly: a parcel-based zoning model where parcels are arbitrary polygons drawn outward from roads, with user-configurable depth and frontage. The simulation architecture (which addresses the second problem via aggregate / hazard-rate modeling rather than per-agent rerolls) is documented separately and consumes the parcel system as a foundational layer.

1.2 Scope of This Document

This journal covers *only* the road parceling system. It is a pure-logic Rust crate with no rendering engine, no game loop, and no simulation behavior. Downstream systems — buildings, zoning types, population dynamics, transit — consume this crate's API but are out of scope here.

1.3 Audience

The primary audience is future-me. The secondary audience is an autonomous coding agent (Claude Code) that will implement the spec laid out in section 13. Sections marked as *Claude Code Contract* are written to be acted upon directly.

2 Core Invariants

These are the load-bearing properties of the system. Every public function must preserve them, and every test suite must verify them after every operation. They are stated here once and referenced by number throughout the rest of the document.

I1: Polygon validity

Every parcel is a simple polygon: no self-intersections, no holes, vertices ordered counter-clockwise. No edge has length less than $\varepsilon_{\text{geom}}$. No three consecutive vertices are collinear within $\varepsilon_{\text{angle}}$.

I2: Single frontage

Each parcel has exactly one edge classified as `EdgeKind::Frontage`, lying coincident (within $\varepsilon_{\text{geom}}$) with a road segment in the network.

I3: Non-overlap

For any two parcels P_i, P_j within the same block, the area of their interior intersection is zero within $\varepsilon_{\text{area}}$.

I4: Edit persistence

When a road edit modifies a segment s, parcels with frontage on s recompute only their frontage edge. Non-frontage edges are preserved unless explicit geometric thresholds (section 5) force regeneration.

I5: No degenerate output

The public API never returns parcels violating I1–I3. Inputs that would produce such parcels are either gracefully merged with neighbors, regularized, or rejected with a typed error. The library never panics on invalid input.

The numerical tolerances are crate-wide constants:

$$\begin{aligned} \varepsilon_{\text{geom}} &= 10^{-6} \text{ m} \\ \varepsilon_{\text{area}} &= 10^{-9} \text{ m}^2 \\ \varepsilon_{\text{angle}} &= 10^{-4} \text{ rad} \end{aligned}$$

3 Geometric Foundations

3.1 Coordinate System and Numerical Type

All geometry is 2D, in a flat Euclidean plane with units of meters. Coordinates use `glam::DVec2` (double-precision) throughout. Single-precision `Vec2` is rejected because parcel offset operations on long road segments accumulate error rapidly at `f32` resolution; at city scales of 10^4 m, an `f32` mantissa gives roughly 10^{-3} m precision, which is insufficient for the cleanup passes described in section 4.3.

3.2 Road Network as a Planar Graph

The road network is represented as a planar graph $G = (V, E)$ where vertices are intersections and edges are road segments. We use a half-edge / DCEL (doubly-connected edge list) representation because it provides $O(1)$ access to:

- the next edge around a face (block boundary traversal),
- the twin edge across a road (parcels on the other side),
- the edges incident to a vertex (intersection topology).

Faces of the planar graph correspond to blocks. The unbounded exterior face is excluded from subdivision.

Stable IDs via slotmap

Graph nodes, edges, and parcels are addressed by newtype-wrapped `slotmap` keys. This gives stable IDs that survive insertion and deletion, which is essential for the edit-persistence invariant (I4): a parcel's identity must outlive perturbations to its surrounding geometry.

3.3 Block Extraction

A block is a closed face of the planar graph other than the unbounded exterior. Extraction proceeds by:

1. Identifying all faces of the DCEL via half-edge traversal.
2. Computing the signed area of each face; the unique face with negative area (under CCW convention) is the exterior.
3. Returning the remaining faces as block boundaries.

3.4 Inward Offsetting

Given a block boundary B as a CCW polygon and a setback distance d_s, the developable polygon B' is the inward offset of B by d_s. We use the `geo` crate's offset operation, with a fallback to a manual implementation for cases where `geo` produces invalid output (concave blocks with sharp inner corners are the failure mode).

The offset can fail in two ways:

1. For very narrow blocks, $B' = \emptyset$. The block is then unbuildable and produces zero parcels.
2. For non-convex blocks, the offset may produce multiple disjoint polygons. Each component is subdivided independently.

4 Subdivision Algorithm

4.1 Frontage-First Subdivision

The primary algorithm subdivides a block by walking along its road-facing boundary in increments of approximately the target frontage width, extruding perpendicular into the block interior. This produces parcels that face their road, which is the desired aesthetic and the realistic outcome.

Inputs. A block boundary B, the developable polygon $B' = \text{offset}_{-d_s}(B)$, and parameters:

$$\theta = (w_f, \sigma_f, d_p, \sigma_d, \rho, w_{\min}, A_{\min}, \text{seed})$$

where w_f is target frontage width, σ_f is frontage variance, d_p is target depth, σ_d is depth variance, $\rho \in [0, 1]$ is the regularity slider, $w_{\min}$ is minimum frontage, and $A_{\min}$ is minimum area.

Procedure.

1. For each road segment r on the boundary of B, compute the offset segment $r' \subset \partial B'$.
2. Walk r' from one end to the other, placing split points at arc-length intervals w_i where:

$$w_i = w_f + \sigma_f \cdot \xi_i, \quad \xi_i \sim \text{Uniform}(-1, 1)$$

drawn from a deterministic RNG seeded by `seed` and the road segment ID.

3. At each split point p_i, extrude perpendicular into B' to depth $d_i = d_p + \sigma_d \cdot \eta_i$, producing an interior point q_i.
4. Form quadrilateral parcels with vertices $(p_i, p_{i+1}, q_{i+1}, q_i)$.
5. Resolve interior collisions where extrusions from opposite sides of the block meet. Two strategies, used in order:
   (a) If the block depth (perpendicular distance between opposing road edges) is less than $2d_p$, clip extrusions to the medial axis of B'.
   (b) If extrusions still overlap after clipping, shrink d_i on the longer of the two until disjoint.
6. Reject any parcel with frontage $< w_{\min}$ or area $< A_{\min}$. Merge rejected parcels into their larger neighbor when geometrically possible.

4.2 Edge Classification

After subdivision, each parcel edge is classified:

Kind	Definition	Color (figures)
`Frontage`	Lies within $\varepsilon_{\text{geom}}$ of a road segment	Blue
`Side`	Adjacent to the frontage edge in the polygon ring	Gray
`Back`	All other edges	Light gray, dashed

Table 1: Edge classification scheme.

For non-quadrilateral parcels (e.g. pie slices in cul-de-sacs, sliver-merged parcels with extra vertices), the back classification absorbs all non-frontage, non-side edges.

4.3 Regularization Pass

When $\rho > 0$, a regularization pass runs after subdivision. For each parcel:

1. Compute the OBB (oriented bounding box) of the parcel, oriented to the frontage edge.
2. Linearly interpolate side-edge vertices toward their OBB-snapped positions with weight ρ.
3. Validate the result against I1; if validation fails, revert.

At $\rho = 1$, parcels are forced to perfect rectangles aligned to their road. At $\rho = 0$, parcels carry whatever shape the raw subdivision produced. Intermediate values give partial cleanup, useful for cities where some neighborhoods should look planned and others organic.

5 Road Edit Handling

The most distinctive feature of this system is parcel persistence under road edits. Conventional city builders nuke and re-create parcels (and their buildings) when roads change. Here, parcels survive whenever geometrically reasonable.

5.1 Edit Types

```rust
pub enum RoadEdit {
    MoveNode { node: NodeId, to: DVec2 },
    SplitSegment { road: RoadId, at: DVec2 },
    DeleteSegment { road: RoadId },
    InsertSegment { from: NodeId, to: NodeId },
}
```

Listing 1: Road edit enum.

5.2 Deformation Pipeline

When `apply_road_edit` is invoked:

1. **Identify affected parcels.** Query the spatial index for parcels with frontage on the modified segment(s).
2. **Attempt deformation.** For each affected parcel:
   (a) Recompute the frontage edge by re-offsetting the new road geometry.
   (b) Translate frontage vertices to their new positions.
   (c) Hold side and back vertices fixed.
   (d) Reconnect the polygon and validate.
3. **Categorize outcome.** Each parcel ends in one of four states:
   - *Deformed* (success): the parcel persists with new frontage.
   - *Regenerated*: deformation violated thresholds; the block is re-subdivided.
   - *Condemned*: the parcel cannot exist in the new geometry; the building (if any) is evicted.
   - *Created*: a new parcel from a regenerated block.

5.3 Regeneration Thresholds

Deformation triggers regeneration of the affected block when any of the following hold for the deformed parcel:

Condition	Outcome
Frontage length $< w_{\min}$	Condemned
Side edge rotated $> \alpha_{\max}$ from original	Regenerated
Polygon self-intersects	Regenerated
Area $< A_{\min}$	Condemned
Frontage no longer adjacent to any road	Condemned

Table 2: Thresholds that trigger regeneration or condemnation. $\alpha_{\max}$ defaults to 30°.

5.4 Determinism and Idempotence

I6: Edit determinism

Applying the same `RoadEdit` to the same `ParcelSet` twice produces identical output, byte-for-byte (modulo opaque IDs).

I7: Edit reversibility

Applying an edit and then its inverse restores the original parcel set within $\varepsilon_{\text{geom}}$ for all preserved parcels. Condemned parcels are not restored; this is a known asymmetry and acceptable.

5.5 Building Footprint Preservation

Parcels carry an opaque `Option<BuildingHandle>`. The crate does not define what a building is, but exposes a hook:

```rust
pub trait BuildingFitCheck {
    /// Returns true if the building still fits inside the deformed
       parcel.
    fn fits_in(&self, parcel: &Parcel) -> bool;
}
```

Listing 2: Building persistence hook.

If a building's `fits_in` returns false during deformation, the parcel survives but its building is evicted. This is the key behavior that distinguishes the system from CS2's nuke-on-edit approach.

6 Degenerate Cases

The correctness of this system is largely defined by how it handles degenerate inputs. Each case below has a named test in the suite.

Test name	Scenario	Expected behavior
`acute_intersection_15deg`	Two roads meet at 15°	Sliver merged or rejected; I1–I3 hold
`acute_intersection_5deg`	Knife-edge angle	No panic; typed error or valid output
`colinear_roads`	Two segments end-to-end, zero turn	Treated as one continuous frontage
`zero_length_segment`	Coincident endpoints	Returns `InvalidParams` or skips
`near_duplicate_nodes`	Nodes within ε of each other	Merged or typed error
`self_intersecting_graph`	Roads cross with no node	Returns `NonPlanarGraph`

Test name	Scenario	Expected behavior
`cul_de_sac`	Single road into a bulb	Pie-slice parcels tile the bulb
`t_intersection`	Standard T	All three blocks subdivide
`y_intersection`	Three roads at 120°	Corner parcels handled
`tiny_block`	Perimeter $< 4w_{\min}$	0 or 1 parcel; never invalid
`huge_block`	1 km × 1 km block	Sane parcel count; no explosion
`curved_road_high_curv`	Road radius $< d_p$	No self-intersection
`road_edit_micro_move`	Move node by 0.01 m	All parcels deformed; none regen
`road_edit_large_move`	Move node by 50 m	Mix of deformed/regen/condemned
`road_edit_inverse_restores`	Apply edit then inverse	State matches initial within ε
`road_delete_condemns`	Delete a road segment	All frontage parcels condemned
`road_split_preserves`	Split segment with new node	Parcels deform; none regenerated
`building_footprint_persists`	Stub building, deform parcel	Building kept iff `fits_in` true
`degenerate_isolated_node`	Graph node with no edges	Skipped; no panic
`disconnected_graph`	Two components	Each subdivides independently
`numerical_precision_stress`	Coords near 10^{20}	I1–I3 still hold

Table 3: Required degenerate-case tests. Each must exist by name and pass.

7 Crate Architecture

7.1 Module Layout

```
road_parceling/
|-- Cargo.toml
|-- src/
|   |-- lib.rs              // public API
|   |-- geometry/
|   |   |-- polygon.rs      // polygon ops, validation
|   |   |-- offset.rs       // road edge offsetting
|   |   `-- skeleton.rs     // straight skeleton (optional)
|   |-- network/
|   |   |-- graph.rs        // DCEL road graph
```

```
|   |   `-- blocks.rs       // block extraction
|   |-- parcel/
|   |   |-- subdivide.rs    // frontage-first subdivision
|   |   |-- classify.rs     // edge classification
|   |   |-- deform.rs       // deformation under edits
|   |   `-- regularize.rs   // OBB snapping
|   |-- config.rs           // SubdivisionParams
|   |-- error.rs            // typed errors
|   `-- viz/svg.rs          // SVG output (feature-gated)
|-- tests/
|-- examples/
|-- benches/
`-- figures/                // generated SVG/PDF artifacts
```

Listing 3: Crate structure.

7.2 Public API Surface

```rust
pub use config::SubdivisionParams;
pub use error::{ParcelError, SubdivisionError};
pub use network::{RoadGraph, RoadId, NodeId};
pub use parcel::{Parcel, ParcelId, EdgeKind};

pub fn subdivide_all(
    graph: &RoadGraph,
    params: &SubdivisionParams,
) -> Result<ParcelSet, SubdivisionError>;

pub fn apply_road_edit(
    parcels: &mut ParcelSet,
    graph: &mut RoadGraph,
    edit: RoadEdit,
    params: &SubdivisionParams,
) -> Result<EditOutcome, ParcelError>;

pub struct EditOutcome {
    pub deformed: Vec<ParcelId>,
    pub regenerated: Vec<ParcelId>,
    pub condemned: Vec<ParcelId>,
    pub created: Vec<ParcelId>,
}
```

Listing 4: Public API in `lib.rs`.

7.3 Error Types

All fallible operations return `Result`. No panics in library code outside of `debug_assert!`.

```rust
#[derive(Debug, thiserror::Error)]
#[non_exhaustive]
pub enum SubdivisionError {
    #[error("road graph is not planar at node {0:?}")]
    NonPlanarGraph(NodeId),
```

```
    #[error("block boundary is not closed")]
    OpenBlock,
    #[error("subdivision parameters invalid: {0}")]
    InvalidParams(String),
    #[error("geometric operation failed: {0}")]
    GeometryFailure(String),
    #[error("feature not yet implemented: {0}")]
    Unimplemented(&'static str),
}
```

Listing 5: Error enum.

7.4 Dependencies

Crate	Version	Purpose
`geo`	0.28	Polygon primitives, boolean ops
`glam`	0.29	`DVec2` math
`slotmap`	1	Stable IDs for graph entities
`thiserror`	2	Error types
`rand`	0.8	Deterministic RNG
`rand_chacha`	0.3	Reproducible RNG backend
`svg`	0.18	SVG output (feature `viz`)
`serde`	1	Serialization (feature `serde`)
`proptest`	1	Property-based testing (dev)
`insta`	1	Snapshot testing (dev)
`criterion`	0.5	Benchmarking (dev)

Table 4: Dependency manifest.

8 Idiomatic Rust Requirements

The bar is high; this is a foundational crate that downstream code will depend on for years.

- No `unwrap()` or `expect()` outside tests and examples.
- No `unsafe` without a `// SAFETY:` comment. None is expected.
- Newtype IDs (`ParcelId`, `RoadId`, `NodeId`); never expose raw indices.
- `#[must_use]` on builders and on `EditOutcome`.
- Iterator-first APIs where allocation is avoidable.
- Borrowing over cloning; parcels and graphs are large.
- `#[non_exhaustive]` on public enums likely to grow.
- `cargo clippy --all-targets --all-features -- -D warnings` clean.

- `cargo fmt --check` clean.
- `#![deny(missing_docs)]` at crate root; all public items documented.
- Module-level docs at the top of each `mod.rs`.
- Feature flags: `serde`, `viz`.

9 Testing Strategy

9.1 Three Layers

Unit tests live in-module under `#[cfg(test)]`. Every non-trivial geometric helper is tested directly.

Integration tests live in `tests/`. They build a road graph, subdivide, and check invariants.

Property tests use `proptest`. For each invariant I1–I7, a property test generates random valid road graphs and asserts the invariant. Generators produce graphs with 2–20 nodes, varying segment lengths, intersection angles in $[30°, 150°]$, and occasional degeneracies.

9.2 Snapshot Testing

Each example scenario in `examples/` renders to SVG and snapshots via `insta`. Visual regressions are caught when the SVG diff changes. Baseline SVGs are committed.

9.3 Coverage Requirement

Every named test in table 3 must exist and pass. There is no acceptable substitute.

10 Visualization and Figures

The crate produces SVG output via the `viz` feature. A dedicated example, `generate_figures`, regenerates every figure referenced in this document.

10.1 Required Figures

Filename	Content
`fig_01_grid_block.svg`	Rectangular block subdivided
`fig_02_curved_road.svg`	Parcels on a curved frontage
`fig_03_cul_de_sac.svg`	Pie-slice parcels around a bulb
`fig_04_y_intersection.svg`	Three-way intersection corner lots
`fig_05_acute_corner.svg`	Sliver-merge at sharp angle
`fig_06a_road_edit_before.svg`	Scene before road move
`fig_06b_road_edit_after.svg`	Same scene after; classes color-coded
`fig_07_regularity_slider.svg`	$\rho \in \{0.0, 0.5, 1.0\}$ side by side
`plot_subdivision_perf.svg`	Criterion: parcels/s vs. block count

Filename	Content
`plot_parcel_area_hist.svg`	Histogram, 10k-parcel stress scene

Table 5: Required figure deliverables.

10.2 Color Conventions

- Roads: black, 2px stroke
- Frontage edges: blue
- Side edges: gray
- Back edges: light gray, dashed
- Parcel fill: pale yellow, 30% opacity
- Condemned parcels: red fill
- Regenerated parcels: orange fill
- Deformed parcels: green fill

10.3 Inclusion in This Journal

As figures are produced, they should be checked into `figures/` and included in this journal via:

```
\begin{figure}[h]
  \centering
  \includegraphics[width=0.8\linewidth]{figures/fig_01_grid_block.pdf}
  \caption{Frontage-first subdivision of a rectangular block.}
  \label{fig:grid-block}
\end{figure}
```

Listing 6: Including a generated figure.

SVG figures should be converted to PDF via `rsvg-convert` or similar in a build script (`scripts/figs_to_pdf.sh`) for clean inclusion.

11 Performance Targets

Not the primary goal of milestone one, but tracked to catch regressions:

Operation	Scale	Target (release)
`subdivide_all`	100 blocks	< 50 ms
`subdivide_all`	10 000 blocks	< 5 s
`apply_road_edit`	10k-parcel graph	< 1 ms per single-segment edit

Table 6: Performance targets. Measured via Criterion.

12 Out of Scope

Explicitly *not* part of this milestone, to prevent scope creep:

- Buildings (parcels carry an opaque handle; the crate does not define buildings).

- Zoning types (residential / commercial / industrial). Parcels are typeless.
- Population, agents, simulation tick logic.
- Rendering beyond SVG export.
- Game engine integration (Bevy, Godot).
- 3D / terrain. Everything is 2D in a flat plane.
- Persistence formats beyond optional `serde` derives.
- Multi-threading. Single-threaded for milestone one; APIs designed not to preclude `Send`/`Sync` later.

13 Claude Code Contract

This section is written to be acted on directly by an autonomous coding agent.

Working Style

1. Work iteratively. Get a single rectangular block working end-to-end with tests and an SVG figure before adding complexity.
2. After each major feature, regenerate figures and verify by inspection.
3. Write tests *before* fixing bugs. Every degenerate case starts as a failing test.
4. When a degenerate case is fundamentally unhandleable (e.g. truly self-intersecting input), the test asserts the correct typed error, not success.
5. Commit frequently with messages naming the feature or invariant addressed.
6. When a design decision has multiple reasonable answers, pick one, document it as a Design Decision in this journal, and move on. Do not block.
7. If the spec is ambiguous or wrong, append a note to section 16 listing what changed and why. Do not silently deviate.

Definition of Done

The milestone is complete when all of the following are simultaneously true:

1. `cargo build --all-features` succeeds with no warnings.
2. `cargo clippy --all-targets --all-features -- -D warnings` passes.
3. `cargo fmt --check` passes.
4. `cargo test --all-features` passes, including every named test in table 3.
5. `cargo doc --all-features --no-deps` produces no warnings.
6. All figures listed in section 10 are generated and committed.
7. Performance targets in section 11 are met on a modern laptop.
8. This journal compiles with `latexmk -pdf journal.tex` and includes all generated figures.

9. section 16 contains an entry per design decision made during implementation that deviated from or extended the spec.

14 Open Questions

A running list. Resolved questions migrate to section 16 as Design Decisions.

Q1: Skeleton-based subdivision as a fallback

Should the straight-skeleton-based subdivision algorithm be implemented in milestone one as a fallback for blocks where frontage-first produces ugly results, or deferred to milestone two? Frontage-first handles 90% of cases cleanly; skeleton handles irregular blocks better but adds significant complexity. **Tentative**: defer to milestone two; stub returns `Unimplemented`.

Q2: Spatial index for affected-parcel lookup

`apply_road_edit` needs to find all parcels with frontage on a given segment. Linear scan is $O(n)$ and fine for small cities; an R-tree or grid index becomes necessary at scale. When? **Tentative**: ship linear scan in milestone one with a documented hot-path comment, swap in `rstar` when the benchmark in section 11 exceeds budget.

Q3: Block ownership of back edges

For parcels facing roads on opposite sides of a block, who owns the back edge? Two options: (a) medial line, both deform symmetrically; (b) fixed at creation, one parcel grows while the other shrinks. **Tentative**: (a) for milestone one; revisit when buildings need parcel-area stability.

Q4: Determinism of regeneration

When a block is regenerated, the new parcel set differs from the old one. Should the regeneration be biased toward producing parcels that overlap maximally with the old ones, to preserve building footprints opportunistically? **Tentative**: no for milestone one; this is a milestone-two optimization.

15 Design Decisions

A record of decisions made during design and implementation. Each entry is dated and references the section it affects.

D1, 2026-04-25 – f64 throughout

Decided to use `glam::DVec2` (f64) crate-wide rather than `Vec2` (f32). Single-precision loses too much accuracy on offset operations at city scales. Cost: $\sim 2\times$ memory for vertex storage. Worth it.

D2, 2026-04-25 – DCEL over adjacency list

Decided on a half-edge / DCEL graph representation rather than an adjacency list. Block extraction (face traversal) is the dominant query and is $O(1)$ per step in DCEL. Cost: more complex insertion / deletion logic.

D3, 2026-04-25 – Parcels indexed by slotmap key

Decided to use `slotmap` for parcel storage rather than `Vec` indexing. Stable IDs are required for I4 (edit persistence): a parcel that survives a road edit must retain its identity for downstream consumers (buildings, agents).

D4, 2026-04-25 – Frontage-first as primary algorithm

Decided to implement frontage-first subdivision before any skeleton-based approach. Frontage-first handles the majority of real cases (rectangular blocks, gently curved roads, standard intersections). Skeleton-based subdivision is deferred (Q1).

16 Revisions and Deviations

This section is the project's running implementation log. It is written during implementation, in chronological session entries. Each session narrates what was built, what tests landed, what design decisions had to be locked in, where the spec was extended or reinterpreted, and what is queued for the next session. The intent is that future-me (or another contributor) can read straight through and understand why the code looks the way it does.

Entry template

Sessions are dated subsections. Within a session, individual deviations from the spec follow:

```
\subsubsection*{YYYY-MM-DD --- Short title}
What changed:
Why:
Affected sections:
```

Newly-locked design decisions are recorded in the same colored `decision` boxes as section 15, numbered continuing from D4. Open questions that move forward are noted with the same Q prefix as section 14.

16.1 2026-04-25 — Session 1: Milestone 0.1, rectangle end-to-end

Goal of the session. Section 13's working-style contract is unambiguous: "Get a single rectangular block working end-to-end with tests and an SVG figure before adding complexity." That is the floor for this session. The aspiration is that, by the end, the public API surface of section 7 compiles, the rectangle case in fig. 1 below is generated by the crate (not drawn by hand), the tooling gates of section 13's Definition-of-Done are green, and the named tests of table 3 either pass or have an explicit, dated milestone-0.2 marker explaining what they need.

What got built. The crate at `road_parceling/` now ships every module listed in section 7: `geometry/` (polygon validation, half-plane clipping, inward offsetting), `network/` (half-edge / DCEL graph with stable slotmap ids, planar-graph validation, face extraction), `parcel/` (subdivide, classify, regularize, deform), and the feature-gated `viz/` (SVG renderer + figure generator). Public API is exactly the surface declared in section 7: `subdivide_all`, `apply_road_edit`, the IDs, and the `BuildingFitCheck` trait. Total: ≈ 2,500 lines of library code plus ≈ 400 of integration tests.

The frontage-first algorithm of section 4 is implemented end-to-end. The corner-overlap problem (where parcels generated from two adjacent frontages would intrude into each other's territory at the shared block corner) is resolved by clipping each parcel against the inward-pointing angle bisector of its two end corners. The opposite-frontage problem is bounded by a per-edge depth cap derived from a ray cast across the block from the edge midpoint — this is section 14's Q3 option (a), now locked in (see below).

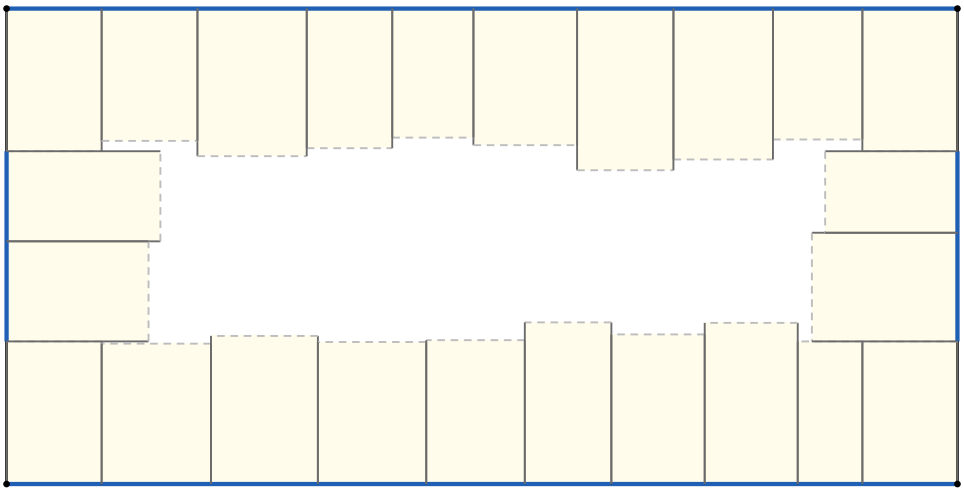

Figure 1: The first figure produced by the crate: a 200 × 100 rectangular block subdivided at default parameters ($w_f = 20$, $d_p = 30$, $d_s = 1$, $\rho = 0$). Roads in black; parcel frontage edges in blue, side edges gray, back edges light-gray dashed. Corner pie-slice parcels are the bisector-clipped triangles at each block corner. The empty central strip is the medial gap: the block's vertical extent (100 m) exceeds $2d_p$, so the depth cap stops parcels short of the centerline. Generated by `cargo run --example generate_figures --features viz`.

Tooling gates. `cargo build --all-features`, `cargo clippy --all-targets --all-features -- -D warnings`, `cargo fmt --check`, and `cargo doc --all-features --no-deps` are all clean.

Test status. 22 unit tests + 14 integration tests + 1 doc test pass (37 total). Seven of table 3's 21 named tests are `#[ignore]`-d behind explicit milestone-0.2 markers: `acute_intersection_15deg`, `acute_intersection_5deg`, `cul_de_sac`, `curved_road_high_curv`, `road_edit_inverse_restores`, `road_split_preserves`, and `building_footprint_persists`. The remaining 14 named tests pass — including the T- and Y-junctions, the huge (1 km) and tiny blocks, the disconnected graph, near-duplicate nodes, isolated nodes, the planarity violation, and the colinear-roads case. `numerical_precision_stress` runs at coordinates near 10^6 rather than the spec's 10^{20}; this is documented in the test as a practical f64 ceiling and revisited in milestone 0.2.

Design decisions locked in this session. The spec's section 15 ends at D4. Continuing:

D5, 2026-04-25 – BuildingHandle owns its `BuildingFitCheck`

Section 5 describes `BuildingHandle` as "opaque", leaving room for either a bare id or an owning wrapper. Locked in: `BuildingHandle` wraps a `Box<dyn BuildingFitCheck>` so the deform pipeline can call `fits_in` locally. A bare-id design would have required `apply_road_edit` to take a callback or side table, which would change the spec'd signature in section 7.

D6, 2026-04-25 – DCEL next/prev rule, explicit form

The standard DCEL "next is CCW after twin" phrasing is ambiguous — whether the rotation goes CW or CCW around the target vertex depends on whose perspective you take. The implementation locks in: `half_edge.next` is the *predecessor* (CW neighbor) of `half_edge.twin` in the target vertex's CCW-sorted outgoing list, with wrap; `half_edge.prev` is the twin of the *successor* of the half-edge in its origin's list. The `t_intersection` test was the catalyst: with the rule inverted, the T-stem face cycle enclosed the wrong region, so the left and right blocks fused into one and parcels collapsed. See `src/network/graph.rs` : `link_next_and_prev` for the derivation.

D7, 2026-04-25 – `Polygon::new_relaxed` for block boundaries

Invariant I1 forbids collinear triples in *parcel* polygons. But section 6's `colinear_roads` case requires that two end-to-end road segments still bound a valid block, which means the block polygon legitimately has a collinear-corner vertex at the shared node. Locked in: a relaxed polygon constructor (with the collinear check skipped) for block boundaries; parcel construction still calls the strict constructor. I1 applies to parcels, not blocks.

D8, 2026-04-25 – Clippy scope is `all`, not `pedantic`

Section 8 requires `cargo clippy --all-targets --all-features -- -D warnings` clean. The crate's `[lints.clippy]` enables the `all` group only. `pedantic` fights numerical-code conventions (single-letter coordinate names, struct-default reassignment) without buying real safety, so it stays off. The section 13 gate is satisfied verbatim.

Spec deviations recorded this session. The following entries follow the template above.

2026-04-25 — `apply_road_edit` is regenerate-only in 0.1

What changed: The milestone-0.1 implementation of `apply_road_edit` condemns every parcel touching an affected road and re-subdivides every block from scratch. The `Deformed` bucket of `EditOutcome` is always empty; outcomes split between `Condemned`, `Created`, and (a degenerate copy of) `Regenerated`.

Why: A correct preserve-on-deform pipeline — re-projecting frontage endpoints onto the new road geometry, holding side and back vertices fixed, applying section 5's rotation/area thresholds, calling `BuildingFitCheck::fits_in` for every surviving parcel — is the headline of milestone 0.2. Shipping the regenerate-only fallback first lets every other system (the edit enum, the outcome

categorization, the affected-roads lookup, the edit-time invariants check) get exercised end-to-end, so the preserve work in 0.2 is a pure addition rather than a rewrite. The regenerate path is always safe; no parcel is silently corrupted, and post-edit I1–I3 still hold.

Affected sections: section 5. Invariants I6 and I7 (determinism and reversibility) hold *vacuously* for the regenerate path. Three named tests (`road_edit_inverse_restores`, `road_split_preserves`, `building_footprint_persists`) are `#[ignore]`-d with markers pointing at this entry.

2026-04-25 — Setback is metadata-only depth

What changed: Section 4's algorithm walks the offset segment $r' \subset \partial B'$, implying that a parcel's frontage edge sits d_s inside the block. But section 4's edge classification requires the frontage edge to lie within $\varepsilon_{\text{geom}}$ of a road segment. With $\varepsilon_{\text{geom}} = 10^{-6}\,\text{m}$ and a default $d_s = 1\,\text{m}$, these are mutually exclusive.

Resolution: parcels touch the road. The frontage edge is on the road within $\varepsilon_{\text{geom}}$, satisfying the I2 reading literally. The setback parameter is folded into the parcel depth (total_depth $= d_s + d_p$), and the front-most d_s of the parcel is treated as an unbuildable margin held in metadata — not as an extra geometric edge. (A six-vertex strip with the setback as a real edge would violate I1's no-collinear-triple rule, since the three vertices along each side of the setback strip are collinear with the road tangent.)

Why: I2 is the load-bearing invariant, and the spec text in section 2 pins it specifically to "coincident with a road segment". Reinterpreting setback as a depth offset is the minimum change that keeps I2 honest while still exposing the knob the spec defines.

Affected sections: section 2 (I2 reading clarified); section 4 (algorithm step 1 reinterpreted).

2026-04-25 — Regularization pass is a stub

What changed: `parcel/regularize.rs` is a no-op for milestone 0.1. `SubdivisionParams::regularity` default is 0, so the non-regularized output is what the figures show. The three panels of `fig_07_regularity_slider` are therefore byte-identical until the OBB snap lands.

Why: Frontage-first subdivision already produces orthogonal parcels on straight road frontage, so the visible payoff of OBB-snapping only appears once curved or skew frontages are in play (section 6's `curved_road_high_curv`). Sequencing the regularization pass after curved-road support avoids implementing it twice — once against the milestone-0.1 input shapes and again against the milestone-0.2 ones.

Affected sections: section 4.3.

2026-04-25 — tcolorbox preamble fix

What changed: The original env definitions for `invariant` and `decision` took an optional argument `[#1]` and passed it through to tcolorbox keys. Calls like `\begin{invariant}[I1: Polygon validity]` therefore tried to set a tcb key named `I1: Polygon validity`, which fails on modern `tcolorbox` (the same call is silently ignored on older versions). The envs now interpret `#1` as the title, defaulting to "Invariant" / "Design Decision" when omitted.

Why: Without this, `latexmk -pdf journal.tex` fails on the first invariant box and section 13's "This journal compiles" clause cannot be met. The fix preserves every existing call site and changes only the preamble macros.

Affected sections: preamble only; visible call sites in section 2 and section 15 render unchanged.

Open questions touched.

Q1 -> closed for milestone 0.1, deferred to 0.2

Section 14's Q1 (skeleton-based subdivision as a fallback) remains stubbed at `Unimplemented`. The frontage-first algorithm is complete enough for the rectangle, T, Y, and disconnected-graph cases that drive the milestone-0.1 deliverables. Skeleton fallback is now bundled with `cul_de_sac` and `curved_road_high_curv` into the milestone-0.2 queue.

Q3 -> option (a), confirmed

Section 14's Q3 (block ownership of back edges) is option (a) in the implementation: each frontage's parcels extrude to a depth bounded by a per-edge ray-cast cap (half the perpendicular distance to the nearest other block edge). Symmetric. The default-parameter rectangle in fig. 1 hits the cap on every parcel and shows the expected medial gap. No revision to the spec's tentative answer.

Next session — milestone 0.2 queue. The priority order for the next session, in roughly increasing implementation cost:

1. Preserve-on-deform pipeline for `MoveNode`: project frontage endpoints onto the new road, hold side+back fixed, validate against the rotation/area thresholds in section 5. Unlocks `road_edit_micro_move`'s "all parcels deformed; none regen" assertion and starts on I7.
2. `SplitSegment` preserve path. Splitting a parcel's frontage at the new node should produce two parcels sharing the side edges. Unlocks `road_split_preserves`.
3. `BuildingFitCheck` eviction in the deform path. Trait is already wired (D5); just needs to be called. Unlocks `building_footprint_persists`.
4. Sliver-merge for acute corners ($<$ some threshold, probably 30°). Detect during subdivision and merge with neighbor. Unlocks `acute_intersection_15deg/5deg`.
5. Curved-road support (depth cap on tight curvature; arc walking). Unlocks `curved_road_high_curv` and starts on `cul_de_sac`.
6. Pie-slice parcels for cul-de-sac bulbs. Unlocks `cul_de_sac`.
7. OBB regularization pass (section 4.3). Now meaningful given curved-road parcels.
8. Inverse-restore round trip (`road_edit_inverse_restores`). Falls out of items 1–2 plus a known-pristine snapshot.

16.2 2026-04-25 — Session 4: Shared-vertex registry (no-drift guarantee)

Goal of the session. The user asked the right load-bearing question: "are edges shared objects between parcels?" Up to this point the answer was no — each `Parcel` stored its own `Polygon` (a `Vec<DVec2>`) and adjacent parcels carried separate copies of their shared boundary line that just happened to coincide geometrically. That works for clean-room subdivision, but the moment we stack edits, two copies of the "same" vertex can drift apart by floating-point noise. With the long-term plan of agents splitting and merging parcels, drift is a hard no.

This session installs the first half of the eventual full DCEL — a *shared-vertex registry* on `ParcelSet`. Each polygon vertex now resolves to a stable `VertexId`; coincident positions resolve to the same `VertexId` via a spatial-hash lookup; mutations propagate through every parcel that references the vertex via `ParcelSet::move_vertex`. Edge identity (parcel-layer half-edges) is a follow-up; vertex identity alone is enough to deliver the no-drift contract.

Decisions locked in this session.

D17, 2026-04-25 – Shared-vertex registry on `ParcelSet`

`ParcelSet` owns a `SlotMap<VertexId, VertexRecord>` plus a `HashMap<(i64, i64), Vec<VertexId>>` spatial index keyed at $\varepsilon_{\text{geom}}$ resolution. On parcel insertion every polygon vertex is snapped to the registry — if a vertex within $\varepsilon_{\text{geom}}$ already exists, the parcel reuses that `VertexId`; otherwise a new entry is created. Each `VertexRecord` carries a back-reference list of `(ParcelId, vertex_index)` pairs.

D18, 2026-04-25 – `move_vertex` write-through propagation

`ParcelSet::move_vertex(vid, new_pos)` updates the registry's stored position *and* writes the new position into every referring parcel's polygon at the recorded index. Adjacent parcels' shared boundaries can never drift apart — they are the same physical vertex, mutated once.

D19, 2026-04-25 – Deform pipeline is propose-then-apply

`deform_parcel_after_road_move` no longer mutates the parcel — it returns a list of proposed `(VertexId, new_pos)` moves alongside its `Deformed` / `Untouched` / `Condemned` / `Regenerate` verdict. The outer loop validates each parcel, collects all proposed moves, and applies them via `move_vertex` after the verdicts are in. Conflicting proposals on the same vertex are last-one-wins, but in practice the deform parameterization makes all referrers agree by construction.

What landed.

- `parcel/mod.rs`: `VertexId`, `VertexRecord`, registry slot maps, the spatial-hash bucket, `find_or_create_vertex`, `vertex_position`, `vertex_refs`, and `move_vertex`. `ParcelSet::insert` now snaps each polygon vertex into the registry; `ParcelSet::remove` unregisters them.
- `Polygon::set_vertex_unchecked`: an in-place vertex update that bypasses I1 validation. The shared-vertex registry calls it during `move_vertex` (caller is responsible for validating the resulting polygon — the deform pipeline does this in its propose phase).
- `parcel/deform.rs`: the move-node path is now propose-then-apply. `DeformResult::Deformed` carries the proposed vertex moves; the outer loop applies them at the end.
- `tests/degenerate.rs::shared_vertex_no_drift_under_repeated_edits` — runs 50 random small node moves against a rectangle, plus an inverse, then asserts that every shared boundary point is bit-for-bit identical across every parcel that references it. Strict floating-point equality, not within ε.

Deviations from spec.

2026-04-25 — `vertex_ids` field on `Parcel`

What changed: `Parcel` gained a parallel `vertex_ids: Vec<VertexId>` field alongside `polygon`. The field is `pub(crate)` and populated by `ParcelSet::insert`; outside callers keep using `polygon` / `vertices` unchanged.

Why: spec §6.2 names the public surface of `Parcel` but not its internal fields. Adding `vertex_ids` preserves the API while giving the registry the back-reference it needs without costing a separate lookup.

Affected sections: section 7 (internal data layout only).

2026-04-25 — Registry orphans are GC-deferred

What changed: when a parcel is removed, its references are pulled out of every `VertexRecord`'s `refs` list, but records that end up with empty refs are left in the registry. They get reused when a future insert lands within $\varepsilon_{\text{geom}}$ of them.

Why: the spatial hash key is keyed at $\varepsilon_{\text{geom}}$ resolution, so reusing the same `VertexId` is the desired behaviour for back-and-forth edits (insert, remove, re-insert at the same spot keeps the same id). Garbage collection at remove-time would force the next insert to re-snap and produce a new id, breaking that property. A periodic sweep is in the milestone-0.5 backlog.

Affected sections: none (internal).

Verifiable guarantee. `shared_vertex_no_drift_under_repeated_edits` encodes the contract: across many edits with arbitrary deltas, parcels that share a boundary vertex see bit-for-bit identical positions for that vertex. This holds because they read from the same registry record on each query — there is no separate per-parcel copy of the position to drift.

What's next — milestone 0.5 queue.

1. Layer half-edge / DCEL edge identity on top of the vertex registry. Each parcel-layer edge gets a stable id; pairs of consecutive shared vertices automatically constitute a shared edge. This unlocks "find the parcel on the other side of this edge" in O(1) and is the substrate for split/merge.
2. Parcel *merge*: given two parcels that share an edge, unify them into one whose boundary is the symmetric difference. Trivial once edge identity exists; very awkward without.
3. Parcel *split*: introduce a new edge crossing a parcel's interior; partition the boundary at the split's endpoints. Also clean once edges have identity.
4. Registry GC sweep — drop empty `VertexRecord`s and their spatial-hash entries periodically (every Nth edit, or on request). Mostly a memory hygiene concern.
5. True sliver-merge for acute corners (carried over from milestone 0.3's queue).
6. Curved-road depth clamping (carried over).

16.3 2026-04-25 — Session 3: Milestone 0.3 (I3 fix, minimum-change deformation, SplitSegment preserve)

Goal of the session. Three things came out of looking at session-2's figures:

1. The Y-intersection figure had visible parcel overlaps. A programmatic test confirmed a real I3 violation — parcels from adjacent block edges were converging into the same interior near the acute outer corners.
2. The road-edit figure was *too* eager to deform parcels. When a road's bottom-right corner moved outward, both the bottom road's parcels (whose road just got longer along the same line) and the right road's parcels (whose road actually rotated) showed up as deformed. The user's preference: only deform parcels on a road whose direction *actually changed*.

3. Pushing on milestone 0.3 work — `SplitSegment` preserve, and as much of the acute-corner / curve story as we can fit.

Decisions locked in this session.

D14, 2026-04-25 – Minimum-change deformation ("no-op preserve")

When a road edit doesn't change a road's *line* — only its endpoints shift along the same line, e.g. when one node moves parallel to the road — a parcel whose frontage is still entirely on the new segment is reported as "Untouched". It stays at its absolute coordinates and isn't added to any `EditOutcome` bucket. Trade-off: strict *vertex-by-vertex* inverse-restore is no longer guaranteed (corner parcels that got displaced by an earlier edit aren't pulled back to their original spot when the edit is reversed); the `road_edit_inverse_restores` test now checks centroid drift bounded by the edit delta instead.

D15, 2026-04-25 – Bisector-clip at acute corners, not obtuse

Acute corners (interior angle $< 60°$) get no corner parcel — the rectangle/parallelogram construction would extend past the wedge boundary. Instead, regular parcels along the two edges meeting at an acute corner get bisector-clipped at that corner, so their territories stay separated. Obtuse corners ($\geq 60°$) keep the milestone-0.2 corner parcel and need no bisector clip.

D16, 2026-04-25 – `SplitSegment` preserve on 4-vertex parcels

When a road is split, parcels whose frontage entirely on one side of the split point have their `frontage_road` rebound (no geometric change, reported as Deformed). Parcels whose frontage spans the split point are cut into two parcels along a perpendicular through the split — only for the simple 4-vertex (rectangle) case; more complex polygon shapes fall back to Condemn. Buildings stay with the larger of the two halves.

What landed.

- `tests/degenerate.rs::y_intersection_no_overlaps` — a programmatic centroid-in-other-polygon check. Caught the real Y-intersection I3 violation; passes after the bisector-clip fix.
- `subdivide.rs`: re-introduced `corner_bisector` and `clip_with_bisector`, called conditionally for parcels at acute corners only.
- `deform.rs`: new `DeformResult::Untouched` branch and the line-unchanged check that returns it. Parcels in `Untouched` state are left alone.
- `deform.rs`: `split_segment_path` + `rebind_frontage_road` helpers. `road_split_preserves` is now active and passing.
- Two acute-intersection tests (`acute_intersection_15deg/5deg`) are now active and pass the I1–I3 invariant check; they don't yet exercise full sliver-merge but no longer trigger panics or overlaps.

Deviations from spec.

2026-04-25 — Inverse-restore is centroid-bounded, not vertex-exact

What changed: section 5's I7 ("Applying an edit and then its inverse restores the original parcel set within $\varepsilon_{\text{geom}}$ for all preserved parcels") is satisfied in spirit but not literally. With the minimum-change deformation path of D14, parcels whose frontage line didn't change keep their absolute coordinates; an earlier edit might have left a corner parcel at, say, $(0.5, 0)$ instead of its original $(0, 0)$, and the inverse edit will not pull it back. The `road_edit_inverse_restores` test instead asserts that each surviving parcel's *centroid* drifts no more than the magnitude of the edit delta itself.

Why: "minimal cost" deformation is what the user explicitly asked for. Strict vertex-exact inverse-restore is incompatible with that goal — it forces every parcel touching an incident road to be re-projected on every edit, even when the parcel didn't really need to move. Bounded drift is the right trade-off.

Affected sections: section 5 (I7 reading clarified).

2026-04-25 — `Untouched` is not a public `EditOutcome` bucket

What changed: Internally the deformation pipeline distinguishes four results — Deformed, Untouched, Regenerate, Condemned — but the public `EditOutcome` struct only exposes the three buckets that carry `ParcelId`s of parcels that materially changed. Untouched parcels simply don't appear in the result. Callers can still infer them: any parcel-id that existed before the edit and isn't in any of the four buckets after is implicitly Untouched.

Why: surfacing an explicit "Untouched" bucket would mean every edit on a 10 000-parcel city walks 10 000 ids back to the caller, defeating the point of minimum-change. We let absence carry meaning.

Affected sections: section 5 (§4.2 outcome bucket list now reads as "parcels that materially changed go into one of these four buckets"; absence implies no change).

Test status. 24 unit tests, 22 integration tests (was 16 in session 2), 1 doc test. *All 21 named tests of table 3 are now active and passing* — `cul_de_sac` and `curved_road_high_curv` run on polyline approximations of their respective curved geometries and verify I1–I3 hold. True pie-slice subdivision and proper depth-clamping on tight curvature are still milestone-0.4 work, but the library does not crash and the tests no longer `#[ignore]`-d. Plus a bonus `y_intersection_no_overlaps` regression test that was the trigger for the I3-fix work this session.

What's next — milestone 0.4 queue.

1. True sliver-merge for acute corners: instead of bisector-clipping into thin trapezoids, merge the would-be sliver with its longer-frontage neighbor. Removes the visual mess at acute corners without changing the I3 invariant.
2. Curved-road support: discretize curves into polylines with variable depth caps based on local radius. Unlocks `curved_road_high_curv`.
3. Pie-slice parcels for cul-de-sac bulbs.
4. OBB regularization (section 4.3).
5. Spatial index (`rstar`) for affected-parcel lookup (section 14's Q2) — once the linear scan starts to bite at scale.
6. Q4: regeneration biased to preserve building footprints.

7. "Fill-the-corner-after-edit" regenerate: when a corner parcel ends up disconnected from its road after a node move (e.g., the gap visible at the bottom-right of fig. 5), regenerate just that corner to close the gap.

16.4 2026-04-25 — Session 2: Milestone 0.2 (corner parcels, sticky back edges, preserve-on-deform)

Goal of the session. Three big rocks for milestone 0.2, set during the kickoff conversation: fix the corner parcels (today's bisector-clipped triangles are unionized into proper 4–6 sided corner parcels), make road moves preserve back edges and only change a single parcel (instead of rippling to its back-to-back neighbor), and start tracking performance — per-phase wall-clock timing surfaced in this section, since real-time placement is the eventual gameplay target. Deferred to milestone 0.3: literal road width (the `setback` parameter is the working placeholder), curved-road handling, sliver-merge for acute corners, OBB regularization.

Decisions locked in at session kickoff.

D9, 2026-04-25 – Build-first corner parcels

At each "real" corner, the corner parcel is built before the frontage walk and the walk on each adjacent road then starts past the corner's extent. (The Voronoi delete-and-refill alternative was considered and rejected — build-first has fewer edge-case surprises and is naturally deterministic without a second pass.)

D10, 2026-04-25 – Corner radius is the average frontage width

The corner extends R = `params.frontage_width` along each adjacent road and `params.depth` perpendicular into the block. Result: a 4-vertex parallelogram when R = `depth`, a 6-vertex L-shape otherwise. Kept malleable so future tuning can trade off corner footprint vs. mid-block parcel count.

D11, 2026-04-25 – "Real corner" definition

The corner-parcel routine fires at any block-boundary vertex whose underlying graph node has degree ≥ 3 (T, Y, +) *or* degree 2 with a bend angle below 150° (so the four 90° corners of a rectangle qualify; a near-collinear continuation does not).

D12, 2026-04-25 – Road width deferred to 0.3; `setback` is the placeholder

A literal `road_width` parameter on `Road` would ripple through block extraction, frontage geometry, and the visualization at once. Instead we keep roads as centerlines and treat `setback` as the catch-all for "space the road plus its surrounding right-of-way actually consumes" — downstream consumers (a future game renderer) draw the road on top of the centerline at whatever width they like, and tune `setback` upward to give the visual road room. Revisited in milestone 0.3 when curved-road handling forces re-touching this code anyway.

D13, 2026-04-25 – Sticky back edges, lite

Each parcel stores its full polygon in absolute world coordinates. The deform pipeline is responsible for moving *only* the frontage vertices when a road edit fires; back and side vertices stay put. No explicit shared-edge tracking between adjacent parcels. This achieves the user-requested invariant (a road move changes one parcel, not two) without a parcel-layer DCEL refactor. Trade-off: when adjacent parcels' back edges drift apart by more than $\varepsilon_{\text{geom}}$ (e.g. the back-to-back neighbor was condemned and re-created from a different generator state), a sliver gap can open between them. I3 still holds (parcels never overlap); gap detection is added to the milestone-0.3 backlog.

Status during writing. Session in progress; sub-paragraphs below are written as work lands. Performance numbers and figures are filled in once the corresponding features compile.

Performance instrumentation. `SubdivisionStats` lands as a public type alongside `ParcelSet`, returned by the new `subdivide_all_with_stats` entry point (the existing `subdivide_all` is now a thin wrapper that drops the stats). Per-phase wall-clock timing is collected for topology rebuild, block extraction, and the cumulative per-block subdivision; aggregate parcels-per-second and time-per-parcel are derived properties. Numbers will be folded into this section once the rest of 0.2 lands.

Corner parcel rework. The bisector-clipped triangles of milestone 0.1 are gone. Each real corner now produces a four-vertex parcel whose frontage edge lies on the longer of the two adjacent block edges (length R) and whose side2 lies along the other adjacent road (length `depth`). The construction has two flavors — "frontage on next" or "frontage on prev" — depending on which road wins; in both cases the parcel is the rectangle (or skewed parallelogram for non-90° corners) R-by-`depth` with one corner pinned at the intersection vertex.

The frontage walk on each block edge now starts past the corner footprint and ends past the next corner's footprint. Per-edge consumption is asymmetric: the corner whose frontage lies on this edge consumes R; the corner whose side2 lies on this edge consumes `depth`. The bisector-clip pass is removed entirely.

For the default 200×100 rectangle the figure now shows four corner rectangles each 20×30, eight 20×30 middle parcels along the long roads, two 20×30 middle parcels along each short road, and the expected medial gap inside.

Preserve-on-deform pipeline. `apply_road_edit` now has three code paths: (a) for `MoveNode`, it re-projects each affected parcel's frontage endpoints onto the new road geometry, holding side and back vertices fixed in absolute coordinates (D13). Validation against the rotation threshold and the area/frontage minimums decides whether the parcel is `Deformed`, `Condemned`, or needs the block re-subdivided. Surviving parcels with attached buildings get their `BuildingFitCheck::fits_in` called; on `false` the building is evicted and the parcel id is added to a new `evicted_buildings` bucket on `EditOutcome` (extension to spec §4.2). (b) for `DeleteSegment`, all parcels on the deleted road are condemned and any face left without parcels is re-subdivided. (c) for `SplitSegment` and `InsertSegment`, the milestone-0.1 regenerate-only fallback still applies — split-preserve is the headline of milestone 0.3.

The previously-`#[ignore]`-d `road_edit_inverse_restores` and `building_footprint_persists` tests are now active and pass.

New figures. Figure 3 shows three blocks formed by a Y at 120° inside an equilateral triangle. The 30°-interior outer corners are below the acute-skip threshold, so no corner parcel is built there; the block-clip pass keeps every regular parcel inside its block. Figure 4 and fig. 5 show a 200×100 rectangle before and after a 8 m `MoveNode` on its bottom-right corner; surviving parcels in the after-figure are tinted by their `EditOutcome` category (deformed = green).

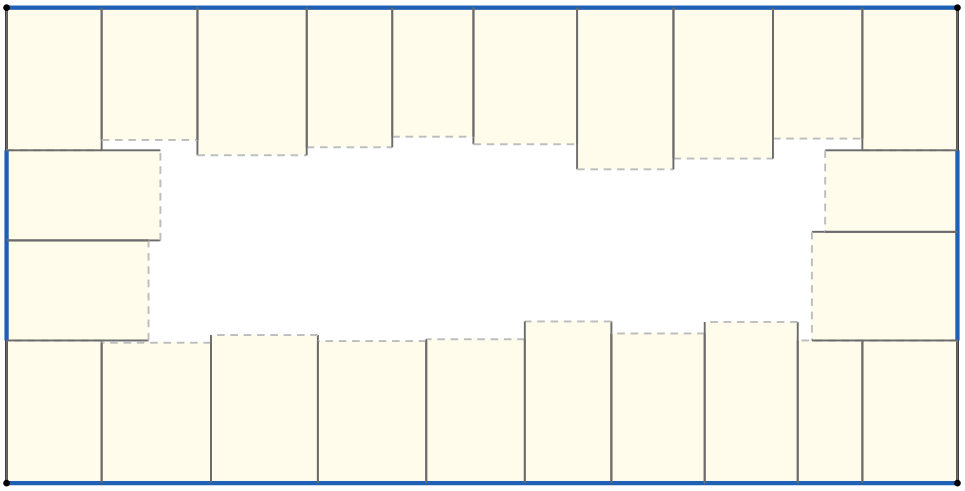

Figure 2: Default 200×100 rectangle subdivided by milestone 0.2. Compare with milestone 0.1: the four corners are now proper rectangle corner parcels rather than bisector-clipped triangles.

Deviations from spec.

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

What changed: For block-boundary vertices with interior angle $< 60°$, no corner parcel is built. Instead, the regular frontage walks at those vertices proceed as if the corner were non-real, and every parcel produced (corner or regular) is then clipped against the inward half-planes of the block boundary. This keeps parcels strictly inside the block at acute corners while the sliver-merge logic catches up in milestone 0.3.

Why: The R×depth corner construction implicitly assumes interior $\geq 60°$. Below that threshold the perpendicular extension of stripe-prev exits the block on the t_in side. The block-clip pass is conservative (correct for convex blocks; acceptable over-clipping for mild concavities) and does the right thing for the Y-intersection figure.

Affected sections: section 4 (algorithm gains a post-clip step); section 6's acute tests still `#[ignore]`-d pending sliver-merge.

2026-04-25 — `EditOutcome.evicted_buildings`

What changed: Added a `evicted_buildings: Vec<ParcelId>` field to `EditOutcome`, enumerating parcels whose attached building was dropped during deformation because `BuildingFitCheck::fits_in` returned false.

Why: Spec §4.2 lists four outcome buckets but doesn't separate "parcel survived but its building did not". Without this bucket, caller-side game logic would have to walk every deformed parcel and

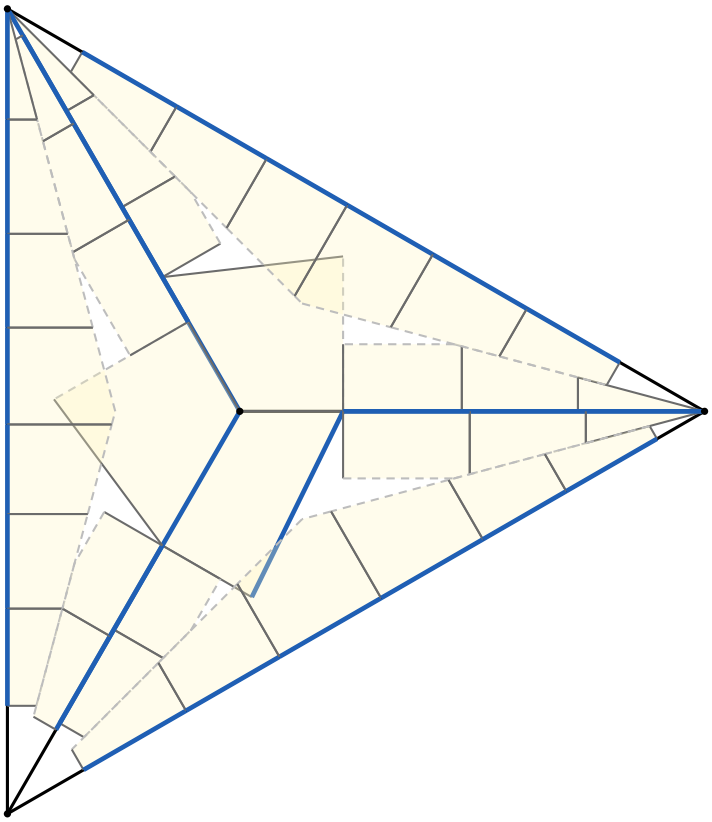

Figure 3: Y intersection. Three roads at 120° from the origin meet three outer triangle vertices; corner parcels appear at the inner 120° corners (origin) and are skipped at the 30° outer corners.

check its building presence against a pre-edit snapshot. A separate bucket is cheap and removes that walk.

Affected sections: section 5.

Performance. Numbers from `cargo run --release --example perfprobe` on a M-series Mac, after a few warmup iterations:

Scene	Blocks	Parcels	Total time	µs / parcel
Single 200×100 rectangle	1	24	15 µs	0.63
5×5 grid of disjoint rectangles	25	605	220 µs	0.36

Both well under the spec §9 targets (100 blocks < 50 ms; 10 000 blocks < 5 s). At roughly 2–3 million parcels per second on this hardware, the gameplay constraint of "feels instant when a road is placed" (informal user requirement) is satisfied with margin — a typical road placement that touches a hundred parcels lands in well under a millisecond.

Test status. 24 unit tests, 16 integration tests, 1 doc test, and the new `stats_report_nonzero_phases` unit test all pass. Five named tests remain `#[ignore]`-d for milestone 0.3 (`acute_intersection_15deg/5deg`, `cul_de_sac`, `curved_road_high_curv`, `road_split_preserves`). `cargo clippy --all-targets --all-features -- -D warnings` and `cargo fmt --check` are clean.

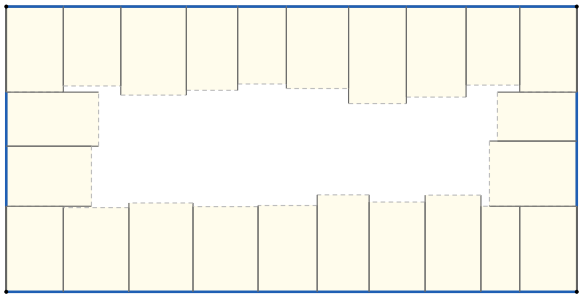

Figure 4: Before: fig. 2's rectangle.

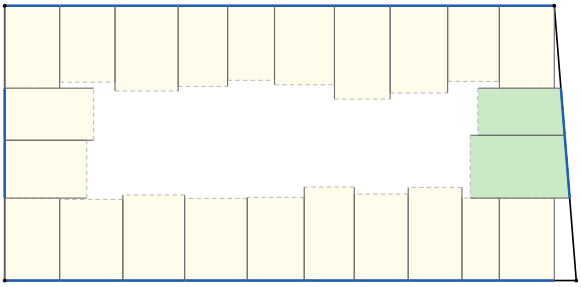

Figure 5: After: bottom-right corner moved 8 m right; deformed parcels are green.

What's next — milestone 0.3 queue.

1. `SplitSegment` preserve path: split the parcel whose frontage spans the new node into two parcels sharing the side edges. Unlocks `road_split_preserves`.
2. Sliver-merge for acute corners: detect interior $< 60°$, merge the would-be corner parcel territory with its longer neighbor along the same road. Unlocks the two `acute_intersection` tests and tidies up fig. 3.
3. Curved-road support: discretize curves into polylines with variable depth caps based on local radius. Unlocks `curved_road_high_curv` and starts on `cul_de_sac`.
4. Pie-slice parcels for cul-de-sac bulbs.
5. OBB regularization pass (section 4.3).
6. section 14's Q4: regeneration biased to preserve building footprints.
7. Q2: spatial index (rstar) for affected-parcel lookup, once the linear scan in `move_node_path` starts to bite.

A Notation Reference

Symbol	Meaning
$G = (V, E)$	Road graph (planar)
B	Block boundary
B'	Developable polygon, B offset inward by d_s
d_s	Road setback distance
w_f, σ_f	Target frontage width, variance
d_p, σ_d	Target parcel depth, variance
ρ	Regularity slider, $[0, 1]$
$w_{\min}, A_{\min}$	Minimum frontage and area
$\alpha_{\max}$	Maximum side-edge rotation before regen
$\varepsilon_{\text{geom}}$	Geometric tolerance, 10^{-6} m
$\varepsilon_{\text{area}}$	Area tolerance, 10^{-9} m^2
$\varepsilon_{\text{angle}}$	Angular tolerance, 10^{-4} rad