

# 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

<b>1</b>	<b>Project Context and Motivation</b>	<b>3</b>
1.1	Why Build This . . . . .	3
1.2	Scope of This Document . . . . .	3
1.3	Audience . . . . .	3
<b>2</b>	<b>Core Invariants</b>	<b>3</b>
<b>3</b>	<b>Geometric Foundations</b>	<b>4</b>
3.1	Coordinate System and Numerical Type . . . . .	4
3.2	Road Network as a Planar Graph . . . . .	4
3.3	Block Extraction . . . . .	5
3.4	Inward Offsetting . . . . .	5
<b>4</b>	<b>Subdivision Algorithm</b>	<b>5</b>
4.1	Frontage-First Subdivision . . . . .	5
4.2	Edge Classification . . . . .	6
4.3	Regularization Pass . . . . .	6
<b>5</b>	<b>Road Edit Handling</b>	<b>6</b>
5.1	Edit Types . . . . .	6
5.2	Deformation Pipeline . . . . .	7
5.3	Regeneration Thresholds . . . . .	7
5.4	Determinism and Idempotence . . . . .	7
5.5	Building Footprint Preservation . . . . .	8

<b>6</b>	<b>Degenerate Cases</b>	<b>8</b>
<b>7</b>	<b>Crate Architecture</b>	<b>9</b>
7.1	Module Layout . . . . .	9
7.2	Public API Surface . . . . .	10
7.3	Error Types . . . . .	10
7.4	Dependencies . . . . .	11
<b>8</b>	<b>Idiomatic Rust Requirements</b>	<b>11</b>
<b>9</b>	<b>Testing Strategy</b>	<b>12</b>
9.1	Three Layers . . . . .	12
9.2	Snapshot Testing . . . . .	12
9.3	Coverage Requirement . . . . .	12
<b>10</b>	<b>Visualization and Figures</b>	<b>12</b>
10.1	Required Figures . . . . .	12
10.2	Color Conventions . . . . .	13
10.3	Inclusion in This Journal . . . . .	13
<b>11</b>	<b>Performance Targets</b>	<b>13</b>
<b>12</b>	<b>Out of Scope</b>	<b>13</b>
<b>13</b>	<b>Claude Code Contract</b>	<b>14</b>
<b>14</b>	<b>Open Questions</b>	<b>15</b>
<b>15</b>	<b>Design Decisions</b>	<b>15</b>
<b>16</b>	<b>Revisions and Deviations</b>	<b>16</b>
<b>A</b>	<b>Notation Reference</b>	<b>16</b>

# 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.

### Invariant

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  $\varepsilon_{\text{angle}}$ .

### Invariant

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

### Invariant

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

**Invariant**

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.

**Invariant**

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.

**Design Decision**

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,  $\sigma_f$  is frontage variance,  $d_p$  is target depth,  $\sigma_d$  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  $\rho$ .
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

```

1 pub enum RoadEdit {
2   MoveNode { node: NodeId, to: DVec2 },
3   SplitSegment { road: RoadId, at: DVec2 },
4   DeleteSegment { road: RoadId },
5   InsertSegment { from: NodeId, to: NodeId },

```

6 }

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  $30^\circ$ .

## 5.4 Determinism and Idempotence

### Invariant

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

**Invariant**

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:

```
1 pub trait BuildingFitCheck {
2     /// Returns true if the building still fits inside the deformed
3         parcel.
4     fn fits_in(&self, parcel: &Parcel) -> bool;
5 }
```

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
<code>acute_intersection_15deg</code>	Two roads meet at 15°	Sliver merged or rejected; I1–I3 hold
<code>acute_intersection_5deg</code>	Knife-edge angle	No panic; typed error or valid output
<code>colinear_roads</code>	Two segments end-to-end, zero turn	Treated as one continuous frontage
<code>zero_length_segment</code>	Coincident endpoints	Returns <code>InvalidParams</code> or skips
<code>near_duplicate_nodes</code>	Nodes within $\varepsilon$ of each other	Merged or typed error
<code>self_intersecting_graph</code>	Roads cross with no node	Returns <code>NonPlanarGraph</code>
<code>cul_de_sac</code>	Single road into a bulb	Pie-slice parcels tile the bulb
<code>t_intersection</code>	Standard T	All three blocks subdivide
<code>y_intersection</code>	Three roads at 120°	Corner parcels handled



Test name	Scenario	Expected behavior
tiny_block	Perimeter $< 4w_{\min}$	0 or 1 parcel; never invalid
huge_block	1 km $\times$ 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 $\varepsilon$
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	Sub 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

```

1 road_parceling/
2 |-- Cargo.toml
3 |-- src/
4 |   |-- lib.rs           // public API
5 |   |-- geometry/
6 |   |   |-- polygon.rs   // polygon ops, validation
7 |   |   |-- offset.rs    // road edge offsetting
8 |   |   |-- skeleton.rs  // straight skeleton (optional)
9 |   |-- network/
10 |   |   |-- graph.rs     // DCEL road graph
11 |   |   |-- blocks.rs    // block extraction
12 |   |-- parcel/
13 |   |   |-- subdivide.rs // frontage-first subdivision
14 |   |   |-- classify.rs  // edge classification
15 |   |   |-- deform.rs    // deformation under edits
16 |   |   |-- regularize.rs // OBB snapping
17 |   |-- config.rs        // SubdivisionParams

```

```

18 |   |-- error.rs           // typed errors
19 |   '-- viz/svg.rs        // SVG output (feature-gated)
20 |-- tests/
21 |-- examples/
22 |-- benches/
23 '-- figures/             // generated SVG/PDF artifacts

```

Listing 3: Crate structure.

## 7.2 Public API Surface

```

1 pub use config::SubdivisionParams;
2 pub use error::{ParcelError, SubdivisionError};
3 pub use network::{RoadGraph, RoadId, NodeId};
4 pub use parcel::{Parcel, ParcelId, EdgeKind};
5
6 pub fn subdivide_all(
7     graph: &RoadGraph,
8     params: &SubdivisionParams,
9 ) -> Result<ParcelSet, SubdivisionError>;
10
11 pub fn apply_road_edit(
12     parcels: &mut ParcelSet,
13     graph: &mut RoadGraph,
14     edit: RoadEdit,
15     params: &SubdivisionParams,
16 ) -> Result<EditOutcome, ParcelError>;
17
18 pub struct EditOutcome {
19     pub deformed: Vec<ParcelId>,
20     pub regenerated: Vec<ParcelId>,
21     pub condemned: Vec<ParcelId>,
22     pub created: Vec<ParcelId>,
23 }

```

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!`.

```

1 #[derive(Debug, thiserror::Error)]
2 #[non_exhaustive]
3 pub enum SubdivisionError {
4     #[error("road graph is not planar at node {0:?}")]
5     NonPlanarGraph(NodeId),
6     #[error("block boundary is not closed")]
7     OpenBlock,
8     #[error("subdivision parameters invalid: {0}")]
9     InvalidParams(String),
10    #[error("geometric operation failed: {0}")]
11    GeometryFailure(String),
12    #[error("feature not yet implemented: {0}")]

```

```

13     Unimplemented(&'static str),
14 }

```

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 <code>viz</code> )
serde	1	Serialization (feature <code>serde</code> )
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  $[30^\circ, 150^\circ]$ , 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
<code>fig_01_grid_block.svg</code>	Rectangular block subdivided
<code>fig_02_curved_road.svg</code>	Parcels on a curved frontage
<code>fig_03_cul_de_sac.svg</code>	Pie-slice parcels around a bulb
<code>fig_04_y_intersection.svg</code>	Three-way intersection corner lots
<code>fig_05_acute_corner.svg</code>	Sliver-merge at sharp angle
<code>fig_06a_road_edit.before.svg</code>	Scene before road move
<code>fig_06b_road_edit.after.svg</code>	Same scene after; classes color-coded
<code>fig_07_regularity_slider.svg</code>	$\rho \in \{0.0, 0.5, 1.0\}$ side by side
<code>plot_subdivision_perf.svg</code>	Criterion: parcels/s vs. block count
<code>plot_parcel_area_hist.svg</code>	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:

```

1 \begin{figure}[h]
2   \centering
3   \includegraphics[width=0.8\linewidth]{figures/fig_01_grid_block.pdf}
4   \caption{Frontage-first subdivision of a rectangular block.}
5   \label{fig:grid-block}
6 \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)
<code>subdivide_all</code>	100 blocks	< 50 ms
<code>subdivide_all</code>	10 000 blocks	< 5 s
<code>apply_road_edit</code>	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.

### Design Decision

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.

### Design Decision

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.

**Design Decision**

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).

**Design Decision**

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 written to during implementation. Every deviation from the spec, every late-breaking design decision, every reversal of an earlier choice goes here with a date and a reason. The intent is that future-me (or another contributor) can read this section and understand why the code looks the way it does.

### Template

`\subsubsection*{YYYY-MM-DD --- Short title}`

What changed:

Why:

Affected sections:

### Entries

*(none yet)*

## 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
$w_f, \sigma_f$	Target frontage width, variance
$d_p, \sigma_d$	Target parcel depth, variance
$\rho$	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 <sup>2</sup>
$\varepsilon_{\text{angle}}$	Angular tolerance, $10^{-4}$ rad