

# Study Plan — Computational Geometry in C

## User Story Template & Examples

### Contents

### How to use this template

Each chapter becomes a *story card*. Fill the meta fields, keep the story in *As a <persona>, I want <capability> so that <benefit>* form, and list verifiable *Given/When/Then* acceptance criteria. Keep tasks small (10–40 minutes). Tags under **Non-Functional** are hints; add or remove as needed.

## CGC-1 — Triangulation Fundamentals

<b>Epic / Feature</b>	Core Geometry Primitives
<b>Business Value</b>	Establish robust predicates and triangulation to unlock downstream algorithms (decomposition, shortest paths, meshing).
<b>Priority / Estimate</b>	Priority: Must SP: 3
<b>Persona</b>	Developer learning computational geometry in C
<b>Dependencies</b>	C toolchain, unit test harness, simple SVG/PPM plotter
<b>Assumptions / Risks</b>	Floating-point robustness; near-collinear points
<b>Story</b>	<i>As a developer, I want to triangulate simple polygons so that I can decompose shapes for further processing.</i>

**Non-Functional** Performance Security Reliability Accessibility Privacy i18n

### Acceptance Criteria (BDD)

<b>Scenario</b>	Happy path
<b>Given</b>	a valid simple polygon in CCW order
<b>When</b>	I run <code>triangulate_monotone()</code> on y-monotone input
<b>Given</b>	adversarial inputs with collinear triples
<b>When</b>	I run the intersection predicates
<b>Then</b>	orientation and segment-intersection return correct results across fuzz tests

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

### Tasks

- ☐ Initialize repo; set up `clang`, `make`, and unit tests.
- ☐ Implement `orient()`, `on_segment()`, `segments_intersect()`.
- ☐ Implement `triangulate_monotone()` and validator.
- ☐ Add fuzz tests with random polygons; export SVG for visual checks.

## CGC-2 — Polygon Partitioning

**Epic / Feature** Decomposition

**Business Value** Partitioning enables linear-time triangulation per part and simpler downstream logic.

**Priority / Estimate** Priority: Must SP: 3

**Persona** Developer extending polygon ops

**Dependencies** CGC-1 predicates; sweep-line event queue

**Assumptions / Risks** Handling holes; event ordering ties

**Story** *As a developer, I want to partition polygons into y-monotone pieces so that triangulation becomes straightforward.*

**Non-Functional** Performance Reliability Testability

### Acceptance Criteria (BDD)

**Scenario** Happy path

**Given** a simple polygon (possibly with holes)

**When** I run `partition_to_monotone()`

**Then** the union of parts equals the original polygon and all parts are y-monotone

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

### Tasks

- ☐ Implement vertical trapezoidalization via sweep-line.
- ☐ Emit monotone parts; triangulate each and stitch.
- ☐ Build regression tests with random polygons and known fixtures.

## CGC-3 — Convex Hulls (2D)

**Epic / Feature** Extremal Geometry

**Business Value** Hulls support collision, diameter/width, and fast search.

**Priority / Estimate** Priority: Must SP: 3

**Persona** Algorithm engineer

**Dependencies** CGC-1 predicates

**Assumptions / Risks** Many duplicate or collinear points

**Story** *As an engineer, I want a robust 2D convex hull so that extremal queries are reliable.*

**Non-Functional** Performance Reliability Determinism

### Acceptance Criteria (BDD)

**Scenario** Happy path

**Given**  $n$  points in the plane

**When** I run `convex_hull()` (monotone chain)

**Then** output is CCW hull with no collinear duplicates on edges and  $O(n \log n)$  time observed

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

### Tasks

- ☐ Implement Graham scan and monotone chain; compare results.
- ☐ Add rotating-calipers: diameter, width, and support function.
- ☐ Benchmarks on random & clustered inputs.

## CGC-4 — Convex Hulls (3D)

**Epic / Feature** 3D Structures

**Business Value** Enables mesh generation and 3D collision.

**Priority /** Priority: Should SP: 5

**Estimate**

**Persona** 3D developer

**Dependencies** CGC-3 (2D hull), half-edge/face structure

**Assumptions /** General-position assumption; numerical tolerance

**Risks**

**Story** *As a 3D developer, I want a 3D convex hull so that I can generate manifold meshes.*

**Non-Functional** Reliability Testability

**Acceptance Criteria (BDD)**

**Scenario** Happy path

**Given** a set of 3D points

**When** I run `convex_hull_3d()` (incremental)

**Then** faces, edges, and vertices form a closed 2-manifold and satisfy Euler's formula

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

## Tasks

- ☐ Implement conflict graph and visibility checks.
- ☐ Export PLY/OBJ; add small viewer.
- ☐ Randomized incremental insertion; regression tests.

## CGC-5 — Voronoi & Delaunay (2 weeks)

<b>Epic / Feature</b>	Proximity Structures
<b>Business Value</b>	Nearest-neighbor, meshing, interpolation, and path planning.
<b>Priority / Estimate</b>	Priority: Must SP: 8
<b>Persona</b>	Geometry practitioner
<b>Dependencies</b>	CGC-3 (2D hull); robust circumcircle predicate
<b>Assumptions / Risks</b>	Degenerate cocircular sets; numeric stability
<b>Story</b>	<i>As a practitioner, I want Delaunay triangulations and Voronoi diagrams so that I can solve proximity problems robustly.</i>

**Non-Functional** Performance Reliability Visualization

### Acceptance Criteria (BDD)

<b>Scenario</b>	Happy path
<b>Given</b>	a set of planar points
<b>When</b>	I run <code>delaunay()</code> and derive Voronoi cells
<b>Then</b>	duality holds; no triangle has an interior point inside its circumcircle; cells partition the plane

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

### Tasks

- ☐ Implement Bowyer–Watson; add edge-flip validator.
- ☐ Compute circumcenters; export Voronoi edges to SVG.
- ☐ Fuzz tests including grids and cocircular inputs.

## CGC-6 — Arrangements of Lines/Segments (2 weeks)

**Epic / Feature** Combinatorial Plane Subdivision  
**Business Value** Enables overlay, point location, and complex planar reasoning.  
**Priority / Estimate** Priority: Should SP: 8  
**Persona** Algorithms engineer  
**Dependencies** DCEL structure; robust intersection  
**Assumptions / Risks** Handling coincident and overlapping segments  
**Story** *As an engineer, I want an arrangement data structure so that I can query faces and overlay datasets.*

**Non-Functional** Reliability Visualization Testability

### Acceptance Criteria (BDD)

**Scenario** Happy path  
**Given** a set of lines/segments  
**When** I insert them incrementally  
**Then** the DCEL remains consistent and point location answers face queries correctly

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

### Tasks

- ☐ Implement DCEL with split/merge operations.
- ☐ Incremental overlay; face-walk verification.
- ☐ Build point-location using trapezoidal map or DAG.

## CGC-7 — Search & Intersection

**Epic / Feature** Query Primitives

**Business Value** Core for collision, CAD, GIS, and planning.

**Priority / Estimate** Priority: Must SP: 5

**Persona** Library author

**Dependencies** CGC-1 predicates; CGC-6 point location

**Assumptions / Risks** Degeneracies; performance on large inputs

**Story** *As a library author, I want robust intersection and point-in-\* tests so that downstream code is correct.*

**Non-Functional** Performance Reliability Testability

### Acceptance Criteria (BDD)

**Scenario** Happy path

**Given** convex polygons  $A, B$

**When** I call `intersect_convex(A,B)`

**Then** returns the exact intersection polygon or empty set and passes property tests

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

### Tasks

- ☐ Implement point-in-polygon (winding and ray-cast).
- ☐ Segment-segment and segment-triangle intersection.
- ☐ Convex polygon intersection via separating axis/rotating calipers.

## CGC-8 — Motion Planning Basics

**Epic / Feature** Paths & Clearance

**Business Value** Pathfinding for robots and games.

**Priority / Estimate** Priority: Could SP: 5

**Persona** Robotics/game developer

**Dependencies** CGC-7 intersections; visibility graph

**Assumptions / Risks** Narrow passages; numeric robustness

**Story** *As a developer, I want shortest paths in polygonal environments so that agents can navigate safely.*

**Non-Functional** Performance Reliability Visualization

### Acceptance Criteria (BDD)

**Scenario** Happy path

**Given** start/goal inside a simple polygon

**When** I run the visibility-graph planner

**Then** it returns a collision-free shortest polyline path with vertex waypoints

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

### Tasks

- ☐ Build visibility graph from reflex vertices + endpoints.
- ☐ Shortest path via Dijkstra on the visibility graph.
- ☐ Optional: PRM with segment-collision queries.

## CGC-9 — Sources, Libraries & Capstone

**Epic / Feature** Synthesis  
**Business Value** Tie implementations to literature and production libraries.  
**Priority / Estimate** Priority: Should SP: 3  
**Persona** Research-minded engineer  
**Dependencies** All previous chapters  
**Assumptions / Risks** Overfitting benchmarks; scope creep  
**Story** *As an engineer, I want a small geometry toolkit and reference notes so that I can apply methods correctly.*

**Non-Functional** Documentation Reproducibility Maintainability

### Acceptance Criteria (BDD)

**Scenario** Happy path  
**Given** the chapter modules  
**When** I run the `cg` CLI on sample datasets  
**Then** I obtain correct outputs with documented trade-offs and performance numbers

*Definition of Ready:* Persona clear; AC drafted; Dependencies known; Estimate set. • *Definition of Done:* All ACs pass; Tests green; Security/all checks; Docs updated; Deployed/flagged.

### Tasks

- ☐ Package modules into a small `cg` CLI with subcommands.
- ☐ Add benchmarks and a gallery of SVG outputs.
- ☐ Write a one-page “When to use” guide per algorithm.