Odin CAD — High-Level Design

Goal: a focused Part-Design style CAD kernel and 2D technical drawing system implemented in **Odin**. Target features: parametric sketching, basic 3D features (pad/extrude, pocket/cut, revolve, fillet/chamfer basic), a parametric feature tree, robust B-rep/CSG hybrid kernel for mechanical part design, and orthographic drawing (hidden lines, dimensions, PDF/SVG export).

---

1 — Project Overview

Scope (MVP): - 2D sketcher with constraints (lines, arcs, circles, dimensions). 2D solver for constraints. - Extrude (pad) and pocket (cut) from sketches onto solids. - Revolve, basic fillet/chamfer limited to simple topologies. - A parametric feature tree with replay/regen. - Boundary representation (B-rep) for final solids; initial internal representation may be CSG for speed of prototyping. - Technical drawing: orthographic views, hidden line, simple dimensioning, export to SVG/PDF.

Non-MVP (later): NURBS surfaces, advanced boolean robustness, complex fillet algorithm, assemblies, STEP full schema coverage.

Why Odin: explicit memory layout, no GC pauses, easy C interop for reusing small C libraries (e.g., constraint solvers), simple syntax for systems programming.

---

2 — High-Level Architecture

```
app/
  cmd/ (cli / gui entry)
  core/
    math/
    geometry/
    topology/
    kernel_csg/
    kernel_brep/
  features/
    sketch/
    extrude/
    revolve/
    fillet/
    param_tree/
    constraints/
  io/
    stl/
    svg/
    step/ (later)
```

```
  ui/
    viewer/
    drawing/
  tests/
  examples/
```

Layers: 1. **Core Math** — vectors, matrices, robust utilities. 2. **Geometry** — analytic curves/surfaces and 2D primitives. 3. **Topology** — data structures (half-edge or directed edge) mapping geometry ↔ topology. 4. **Kernel** — CSG for rapid prototyping, B-rep for final operations and drawing. 5. **Features** — parametric operations built on kernel primitives. 6. **Solver** — 2D constraint solver for sketcher. 7. **IO/UI** — rendering, file export.

---

3 — Key Data Types (Odin pseudocode)

> Note: these are concise reference types to guide initial implementation. Use `#packed` or explicit alignment only when interfacing with C.

```
package cad

// --- math ---
Vec2 :: struct { x, y: f64 }
Vec3 :: struct { x, y, z: f64 }
Quat :: struct { x, y, z, w: f64 }
Mat4 :: [4][4]f64

// --- geometry primitives ---
Line2 :: struct { p0, p1: Vec2 }
Circle2 :: struct { center: Vec2, r: f64 }
Arc2 :: struct { center: Vec2, r: f64, t0, t1: f64 }

Plane :: struct { origin: Vec3, normal: Vec3 }
Sphere :: struct { center: Vec3, r: f64 }
Cylinder :: struct { axis_origin: Vec3, axis_dir: Vec3, r: f64 }

// Parametric curve example
Bezier3 :: struct { pts: [4]Vec3 }

// --- topology handles ---
Handle<T> :: struct { id: int }

Vertex :: struct { pos: Vec3 }
Edge :: struct { v0, v1: Handle(Vertex); curve_id: int }
Face :: struct { surface_id: int; boundary_edges: []Handle(Edge) }
Shell :: struct { faces: []Handle(Face) }
Solid :: struct { shells: []Handle(Shell) }
```

```
// --- B-rep container ---
BRep :: struct {
  vertices: []Vertex
  edges: []Edge
  faces: []Face
  solids: []Solid
}

// --- Feature node in parametric tree ---
FeatureNode :: struct {
  id: int
  name: string
  params: map[string]f64
  inputs: []int // references to previous features
}

Sketch2D :: struct {
  verts: []Vec2
  edges: []Line2 | Arc2 | Circle2
  constraints: []Constraint
}

Constraint :: union {
  Coincident { a: int; b: int }
  Distance { a: int; b: int; value: f64 }
  Angle { a: int; b: int; value: f64 }
  Tangent { edge_a: int; edge_b: int }
}
```

---

4 — Core Modules & Responsibilities

4.1 `core.math`

- Vector, matrix, quaternion operations, numeric epsilons, robust comparisons.
- Utilities: project point on plane, line-line intersection (2D/3D), plane-plane intersection.
- Tolerance management: global `EPS = 1e-9` with configurable epsilon per model.

4.2 `core.geometry`

- Analytic primitives and evaluation routines (point on curve, normal on surface).
- Curve/surface trimming helpers.
- 2D polygon boolean helpers (for extrusion profiles). Start with Greiner–Hormann or clipper port.

4.3 `core.topology`

- Basic handle allocator (stable integer handles referencing arrays).
- HalfEdge or directed edge structure supporting Euler checks.
- Functions for sewing edges into loops and faces.

4.4 `kernel_csg` (early prototypes)

- Fast boolean operations on solids using BSP or mesh-based booleans.
- Use mesh booleans for prototyping: triangulate surfaces and perform boolean using robust mesh boolean lib (port or call C).

4.5 `kernel_brep` (target)

- B-rep construction and boolean by face splitting + stitching.
- Maintain edge/face topology and provide queries: get_faces_on_solid(), get_edges_on_face().

4.6 `features`

- Each feature is a deterministic function: given prior model state and parameters → new model state + history.
- Common features: Sketch (creates 2D profile), Extrude, Cut, Revolve, Fillet, Chamfer.
- Parametric references: features reference sketches by id and edges by handles.

4.7 `constraints` (sketch solver)

- 2D solver implementing: coincidence, distance, angle, equality, perpendicular, tangent, dimension constraints.
- Numerical approach: build system of nonlinear equations and solve with Levenberg–Marquardt.
- Consider porting SolveSpace solver or using C library via FFI.

4.8 `io` and `ui`

- IO: STL for quick export, SVG/PDF for drawings, STEP/IGES later.
- UI: minimal OpenGL/Vulkan viewer; use existing C libs for windowing (GLFW) with Odin bindings.

---

5 — APIs & Example Workflows

Parametric Flow (user perspective): 1. Create Sketch on plane. 2. Add geometry to sketch and add constraints/dimensions. 3. Solve constraints → finalize closed profile. 4. Create Extrude feature referencing Sketch id and depth param. 5. Create Cut feature referencing second sketch on face. 6. Change a sketch dimension → param tree recomputes, model updates.

Simplified API sketch (Odin pseudo):

```
m := cad.NewModel()
sk := m.NewSketch(plane = Plane{origin, normal})
sk.AddLine(p0, p1)
sk.AddCircle(center, r)
sk.AddConstraint(Constraint.Distance{a: v0, b: v1, value: 20.0})
sk.Solve()
feat := m.FeatureExtrude(sketch_id = sk.id, depth = 10.0)
```

---

6 — Robustness & Numerical Stability

- Use double precision everywhere.
- Implement consistent tolerance policy and adjustable model tolerance.
- Avoid relying only on floating equality checks; provide `is_near(a,b, eps)` helpers.
- For booleans, implement face classification with conservative epsilon and robust point-in-solid tests.
- Start with *mesh boolean* fallback for complex cases; improve robustness over time.

---

7 — Interop & Reuse Opportunities

- Use C interop to leverage small, proven projects: e.g., **libtess** for tessellation, **clipper** for 2D polygon booleans (C++ but small), or **GLFW** for windowing.
- Consider compiling an existing small C/C++ constraint solver or porting the SolveSpace solver into Odin.
- For STEP I/O, full STEP is large — consider using the STEPcode (C++). Initially support only simple STEP write of solids or rely on export to STEP using external tools.

---

8 — Testing Strategy

- Unit tests for all math primitives and geometric predicates.
- Property tests for boolean ops (commutativity where expected, edge cases with coplanar faces).
- Regression tests saved as small models (load, perform operations, compare hashes of topology/geometry).
- Visual golden tests: render to SVG and compare bitwise (or perceptual threshold).

---

9 — Roadmap & Milestones (example)

- **Month 0–2:** Core math, Vec/Mat library, basic 3D primitives and viewer.
- **Month 2–4:** Sketcher + 2D constraint solver (basic constraints), extrude, export STL.
- **Month 4–8:** B-rep basics and simple booleans, parametric feature tree, UI basics.
- **Month 8–12:** Improve boolean robustness, fillets/chamfers, drawing export to SVG/PDF.
- **12+ months:** STEP import/export, NURBS, advanced fillet engine.

10 — Example File/Module API Signatures (starter)

```
package cad/core/math
  func Vec3_Add(a,b: Vec3) -> Vec3
  func TransformPoint(m: Mat4, p: Vec3) -> Vec3

package cad/geometry
  func Line2_Intersect(l1, l2: Line2) -> (bool, Vec2)
  func PolygonArea2D(poly: []Vec2) -> f64

package cad/topology
  func NewHandleAllocator() -> HandleAllocator
  func SewEdgesIntoLoop(edges: []Handle(Edge)) -> Handle(Face)

package cad/features
  func NewSketch(plane: Plane) -> Handle(Sketch2D)
  func Extrude(sketch: Handle(Sketch2D), depth: f64) -> Handle(Solid)

package cad/constraints
  func SolveSketch(sk: Handle(Sketch2D)) -> bool
```

11 — Recommended Starting Libraries & References

- SolveSpace — open small constraint-based modeler (study/port solver).
- BRL-CAD — source of ideas for C APIs and CSG concepts.
- Clipper (Vatti/Weiler–Atherton alternatives) for 2D polygon booleans.
- libtess2 or libtess for tessellation for rendering/drawing.

12 — Development Tips (Odin specifics)

- Use slices + stable handle indices for topology rather than raw pointers; simplifies serialization.
- Create small `unsafe` blocks only where necessary for performance or C interop.
- Keep math functions `inline` where beneficial.
- Provide `#[test]` driven examples in Odin to validate geometry.

13 — Deliverables I can prepare next (pick one)

1. Odin module skeleton (file layout + empty functions) you can clone and start coding.
2. Constraint solver plan + small prototype (2D solver equations and LM implementation sketch).
3. Minimal example: build a sketch → extrude → export STL in Odin (working code).

If you want, I can **generate the Odin module skeleton** now (file structure + function signatures and minimal implementations) so you can clone it and begin coding. Which of the three deliverables above should I produce first?