DOMINIC WEBER

21.06.2000	Born in Erlangen (GER)
2000 - 2005	Early years in Nürnberg (GER) and Strasbourg (FRA)
2005 - 2011	Primary school in Lubumbashi (DRC)
2011 - 2018	Middle- and highschool in Ferrette (FRA) and St.Louis (FRA)
2018 - 2019	BHS Stages 1-4 at the Talland School of Equitation, Cirencester (GBR)
2019 - 2020	Level 1 Bespoke Tailoring at Newham College, London (GBR)
2020 - 2023	BSc Architecture at the EPFL, Lausanne (CH)
2023 - 2024	Internship at Burckhardt Architecture, Basel (CH)
2024 -	MSc Architecture at the ETHZ, Zürich (CH)
2025 -	Teaching Assistant at Gramazio Kohler Research, ETHZ (CH)

C2	German, French, English
A2	Italian, Japanese

+41 77 289 4770
dominictsweber@gmail.com

1
stacked plates
digital design and fabrication
python, fusion3d

2
compasstudioonline.com
software design
javascript, python, html

3
reciprocal frames
digital design and fabrication
c#, python

4
generative wall
generative design
python

5
lit vacuum
fabrication and product design
rhino

6
scar-city
design and project management
python, photoshop, twinmotion

stacked plates

digital design and fabrication
written in python using RhinoCommon
fabricated using a 6-axis CNC with Fusion3D

brief

Create a modular floor slab made of CLT plates.
Maximise efficient material usage.
Dry wood-wood joints for radical disassembly.

PROBLEMS

1. Providing the structural solution.
2. Solving plate intersections in situations of negative curvature.
3. Simulate the CNC-machining and generate G-Code.

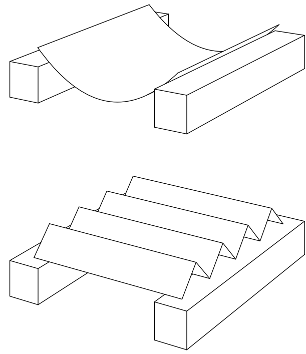
1 — STRUCTURAL MORPHOLOGY

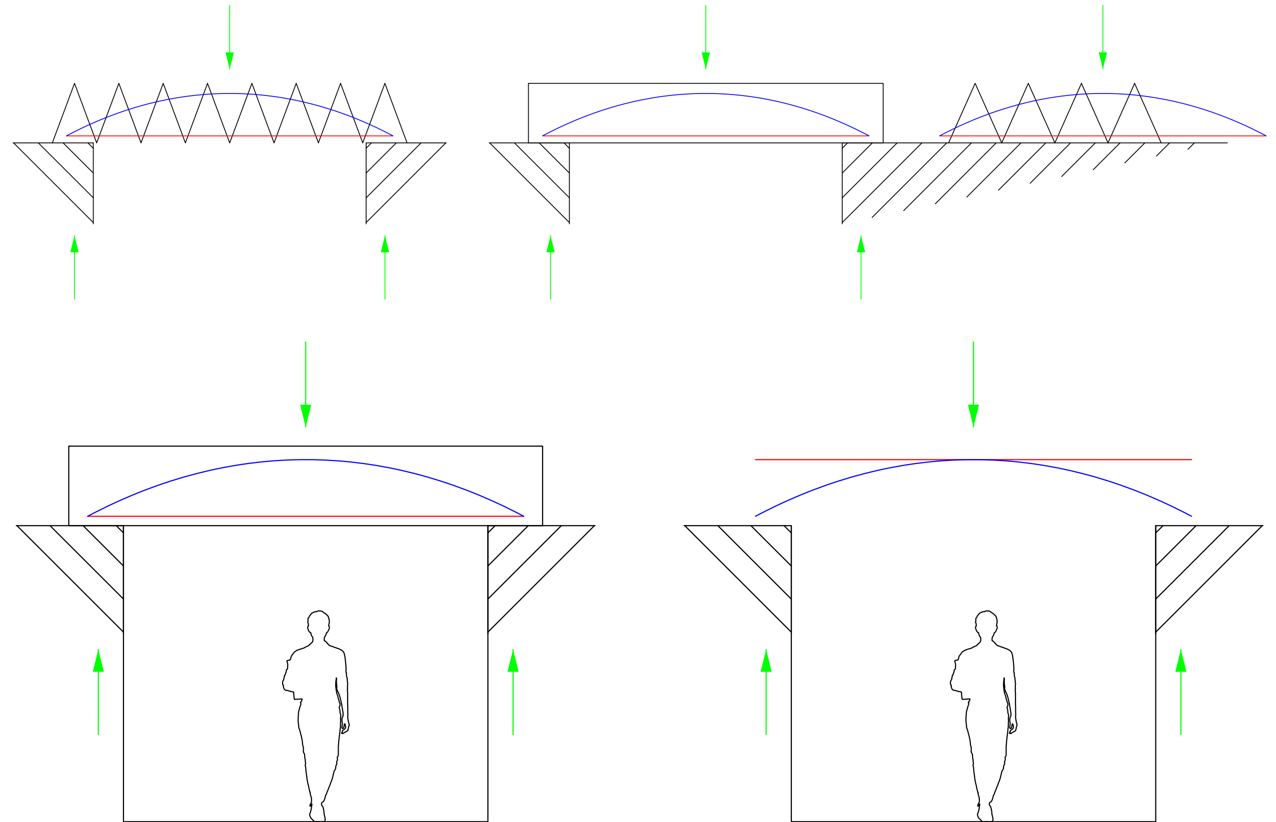
➡ **outset**

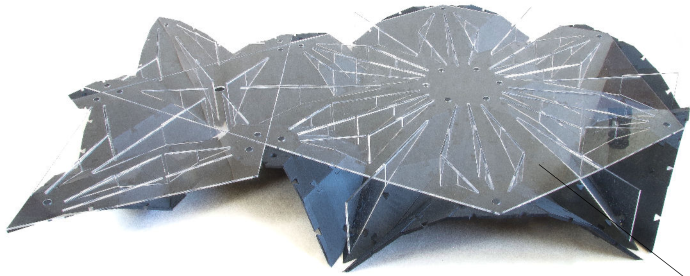
A modular floor slab made of small, handy CLT plates.
Only wood-wood connections to assure disassembly.

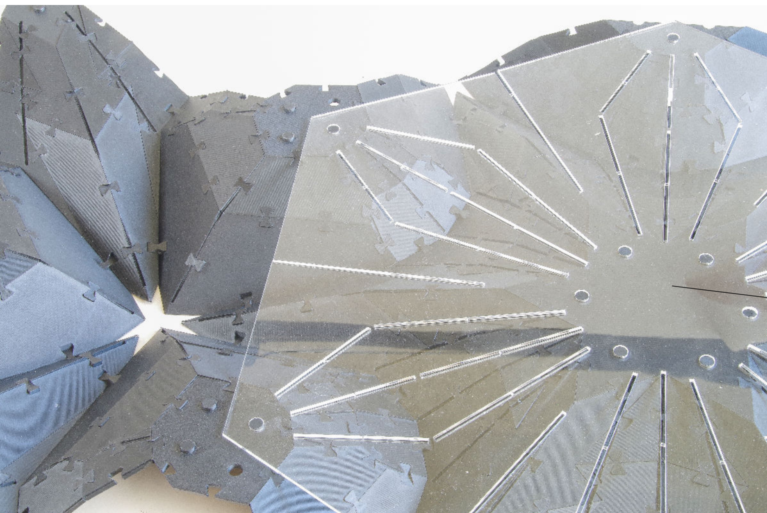
➡ **PROBLEM**

Specify a morphology that optimizes structural performance in synergy with its use as a floor.



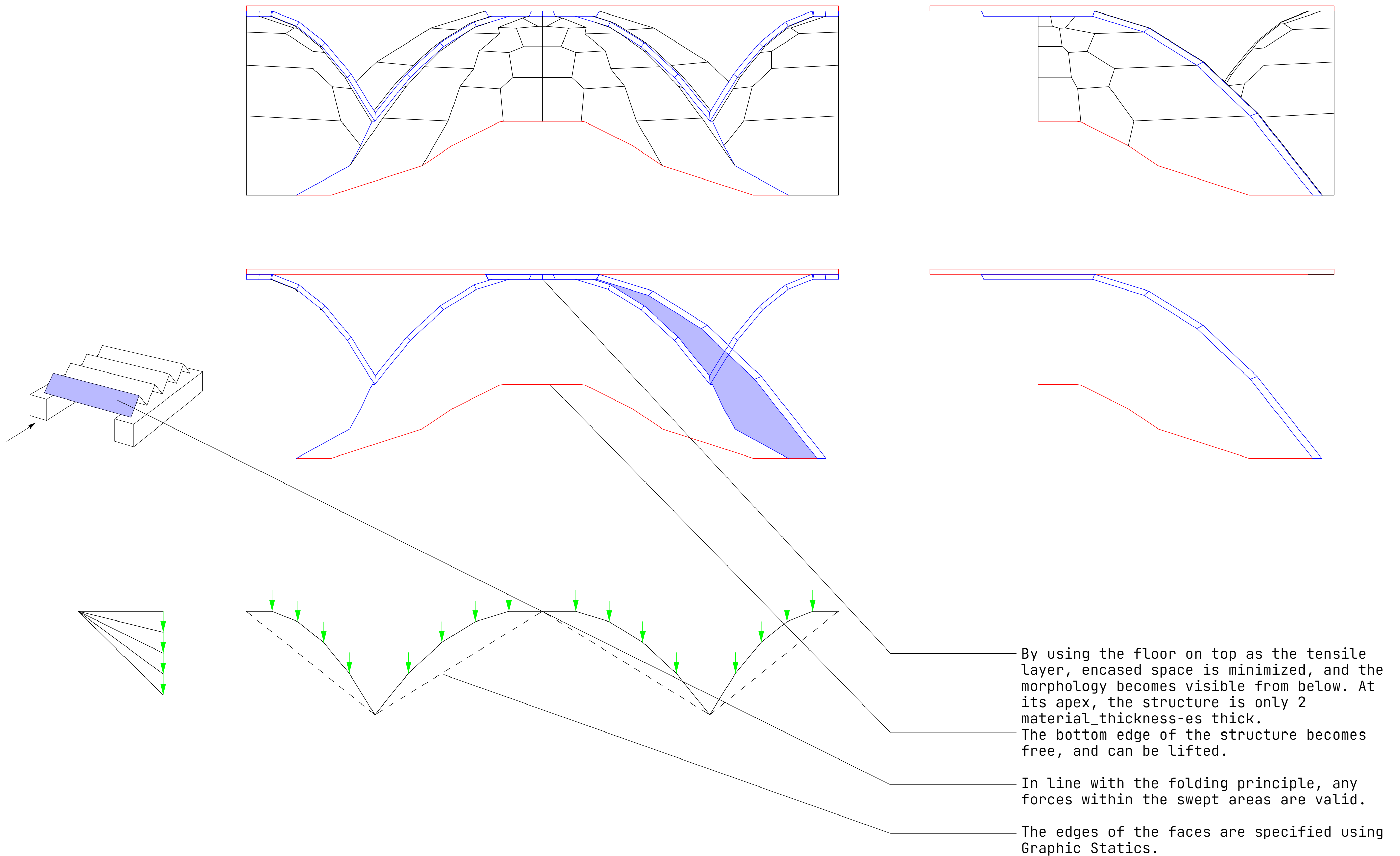




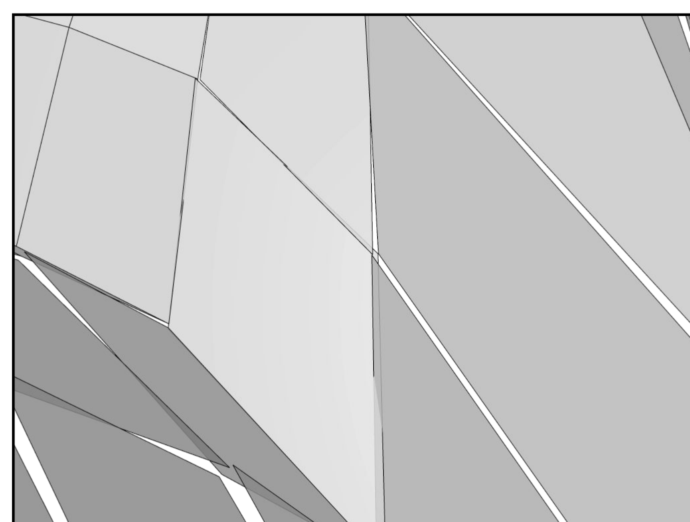
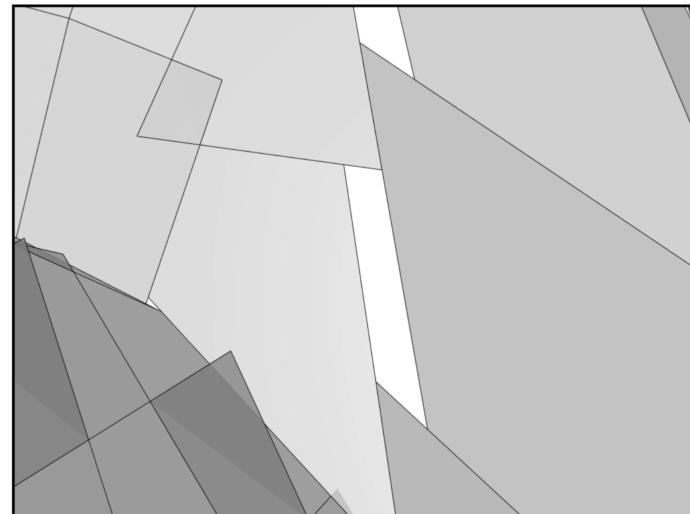
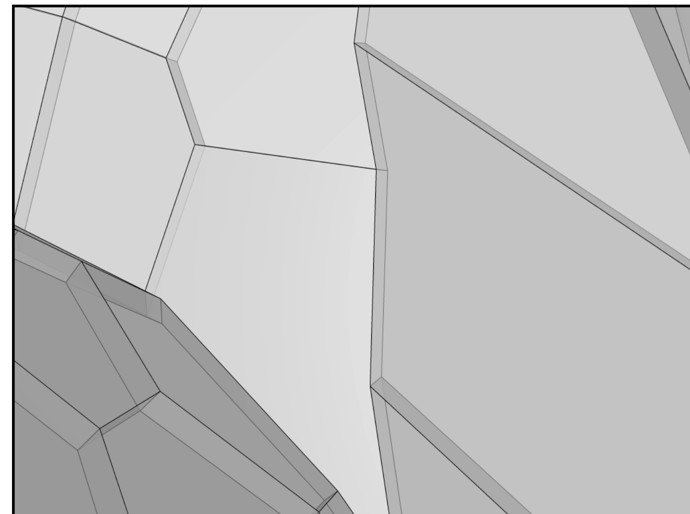
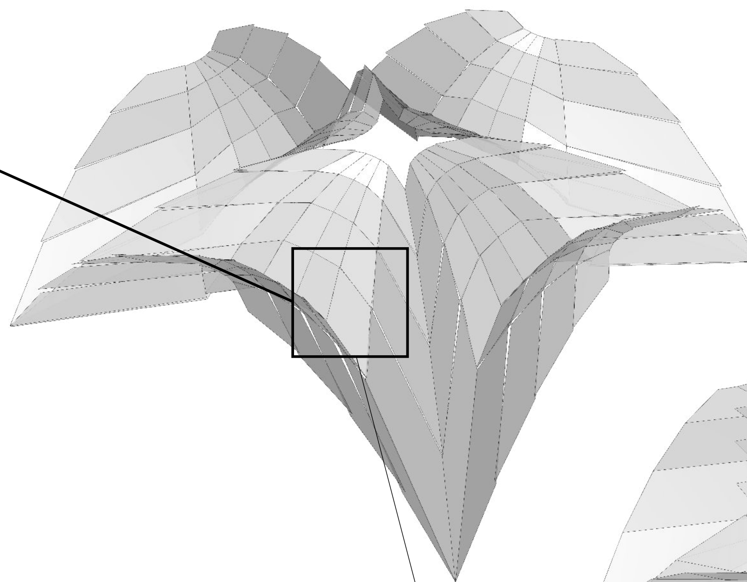
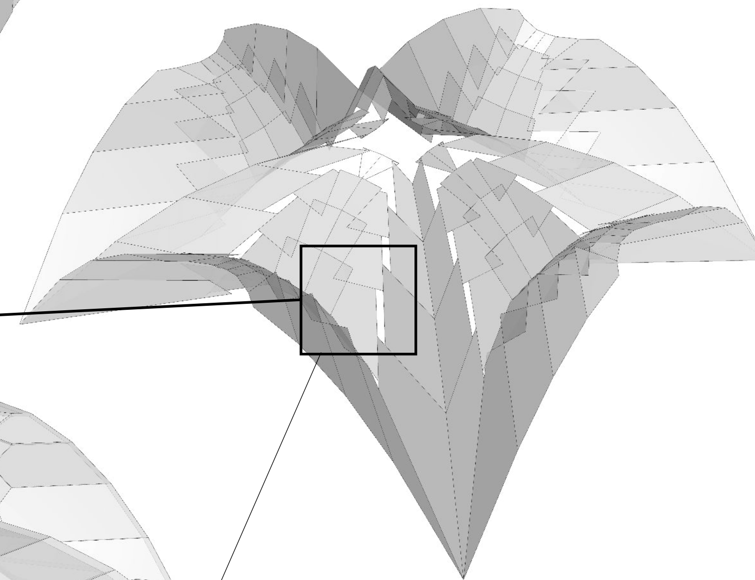
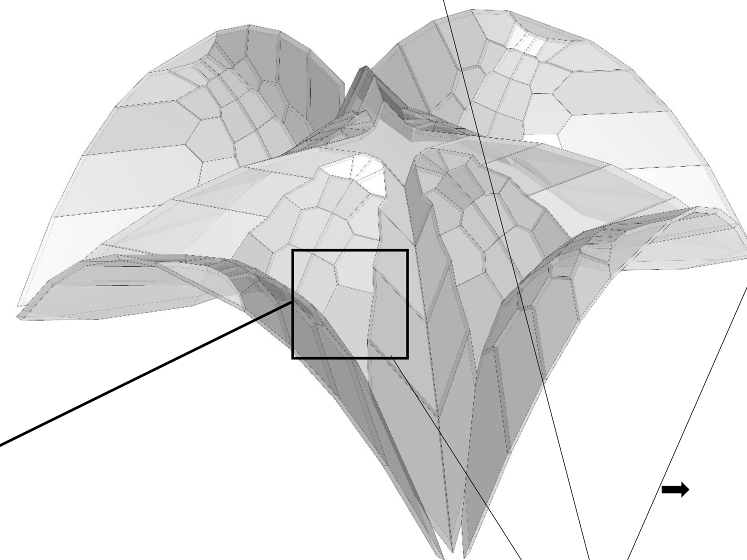


The project relies on the structural principle of folding to attain static height.
→ this states that as long as the forces are contained within the projection of material, the forces are valid.

To allow for ceiling height, we activate the flat floor surface on top as the tensile layer.


By using the floor on top as the tensile layer, encased space is minimized, and the morphology becomes visible from below. At its apex, the structure is only 2 material_thickness-es thick.
The bottom edge of the structure becomes free, and can be lifted.
In line with the folding principle, any forces within the swept areas are valid.
The edges of the faces are specified using Graphic Statics.

Small pieces make assembly by one or two people possible.
The X-fix (c) connectors allow for dry wood-wood connections that work even at shallow or almost parallel plate meetings.

2 — PLATE DEFINITION

outset

A network of faces created through graphic statics.
A plane for each face.

PROBLEMS

Compute the edges of each face.
Solve situations of negative curvature.

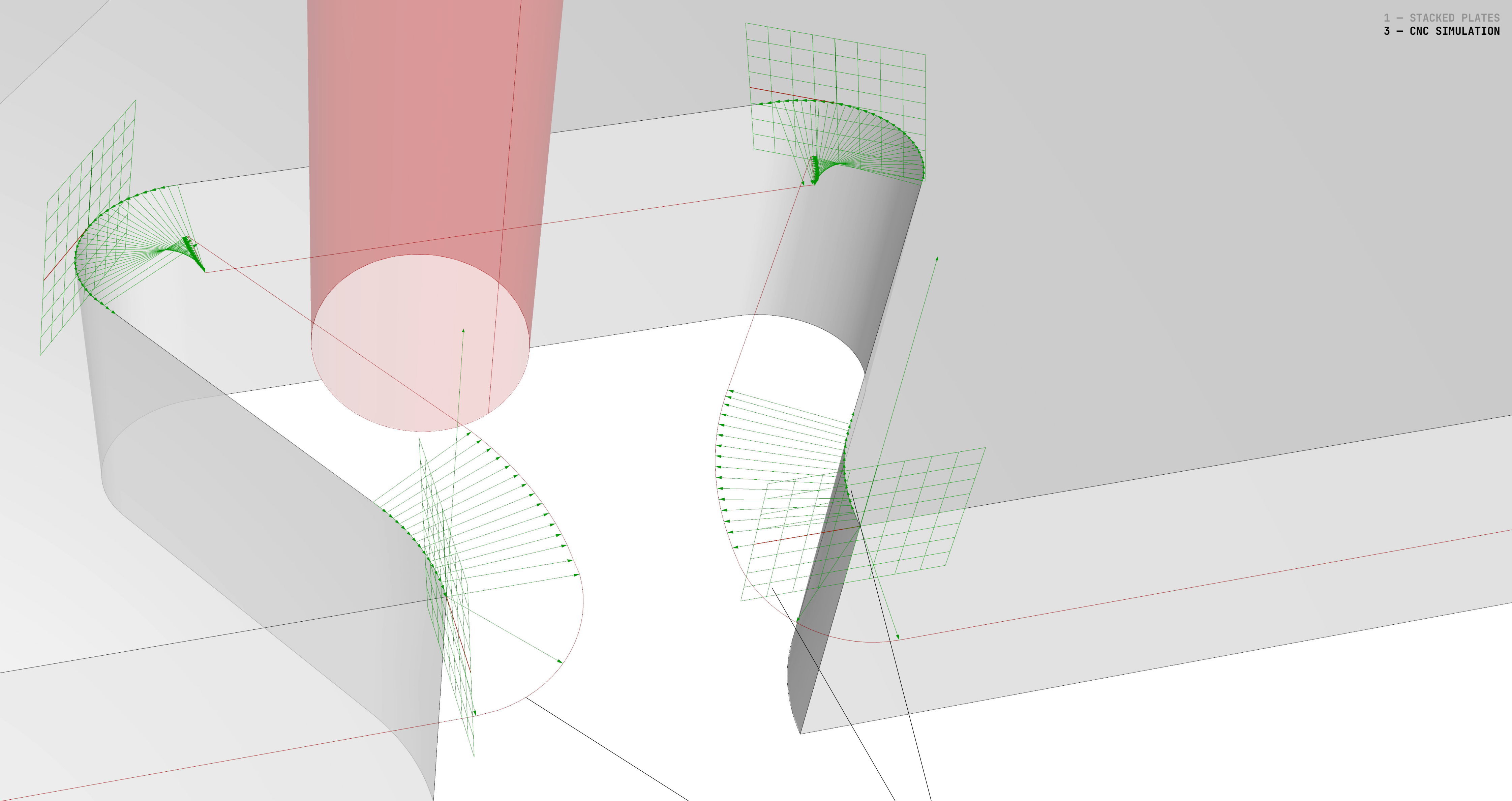
1. harvest the origin, normal and four planes of the corner

2. define the two diagonals of the corner

3. compare their distance to the origin of the corner

4. both diagonals are valid. however, the closest diagonal is the preferable one

```
elif len(filtered_planes) == 4:
    line1 = planeplane_intersection(p1, p3)
    line2 = planeplane_intersection(p2, p4)
    int_point1 = lineplane_intersection(line1, p2)
    int_point2 = lineplane_intersection(line2, p1)

    plane1 = rg.Plane(int_point1, normal)
    plane2 = rg.Plane(int_point2, normal)

    point1 = lineplane_intersection(resultant, plane1)
    point2 = lineplane_intersection(resultant, plane2)

    lcurve1 = rg.LineCurve(origin, point1)
    lcurve2 = rg.LineCurve(origin, point2)
    length1 = lcurve1.GetLength()
    length2 = lcurve2.GetLength()

    tolerance = 0.1
    difference = abs(length1 - length2)

    if length1 < length2 and difference > tolerance:
        name = "left"
        upperpt = lineplane_intersection(line1, p2)
        if not upperpt:
            upperpt = point1
        lowerpt = lineplane_intersection(line1, p4)
        if not lowerpt:
            lowerpt = point1

        if upperpt.DistanceTo(lowerpt) < tolerance:
            name = "unique"
            corners[f][i][j] = Corner(name, normal, origin, resultant)
            corners[f][i][j].upperpoint = upperpt

        else:
            corners[f][i][j] = Corner(name, normal, origin, resultant)
            corners[f][i][j].upperpoint = upperpt
            corners[f][i][j].lowerpoint = lowerpt

    elif length1 > length2 and difference > tolerance:
        name = "right"
        …

    else:
        name = "unique"
        upperpt = point1
        corners[f][i][j] = Corner(name, normal, origin, resultant)
        corners[f][i][j].upperpoint = upperpt
```



3 — CNC SIMULATION

➡ **outset**
Plate contours discretized into vertices.
A normal vector to the side face on each vertice.

➡ **PROBLEMS**
Generate an offset path for the CNC tool.
Generate G-Code for the CNC.

1. extract start, end segment normals

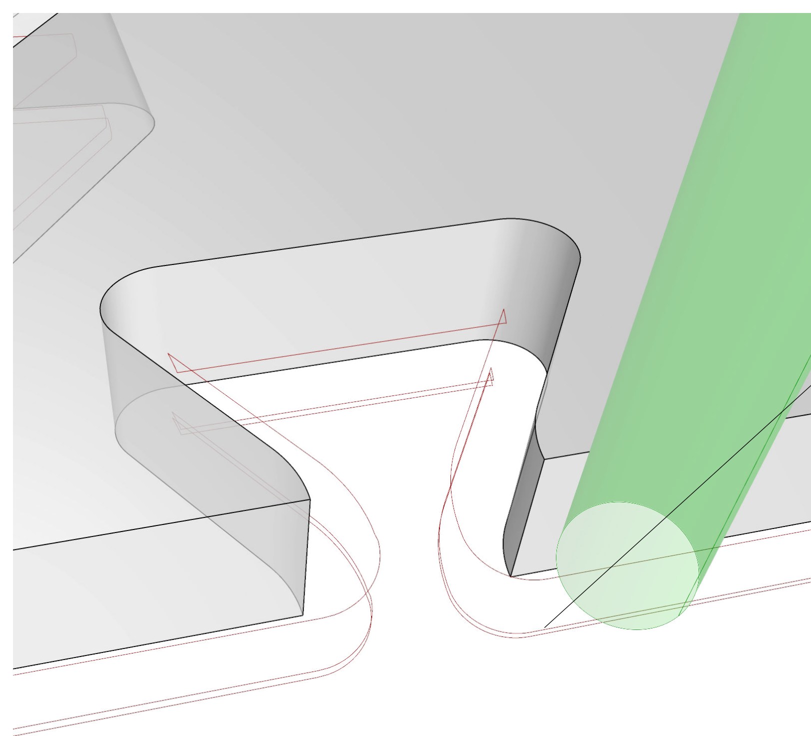
2. compute average corner normals

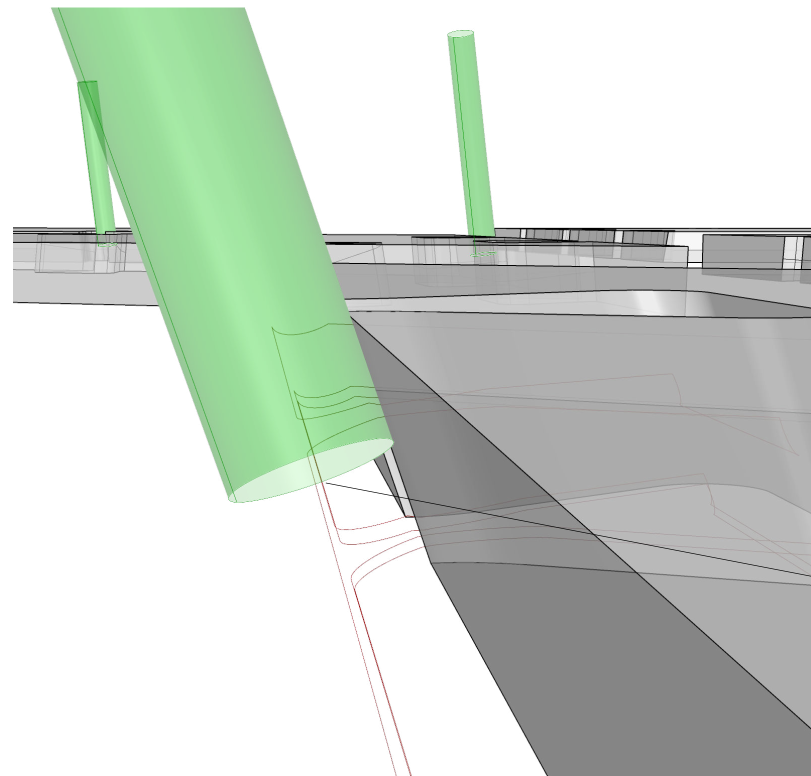
4. solve concave angles

5. offset rest of the segment vertices

6. solve convex angles by adding an arc

```
def make_offset(lastseg, curseg, nextseg):

    # we begin by the start and end normal vectors of our current segment
    currentstartnormal = curseg.normals[0]
    currentendnormal = curseg.normals[-1]

    # before and after our current segment
    nextnormal = nextseg.normals[0]
    lastnormal = lastseg.normals[-1]

    # averages at the ends to make the connection with the next segment
    naveragenormal = (currentendnormal + nextnormal)
    laveragenormal = (currentstartnormal + lastnormal)

    unitizednav = naveragenormal.Clone()
    unitizedlav = laveragenormal.Clone()
    unitizednav.Unitize()
    unitizedlav.Unitize()
    unitizednav *= curseg.currenttooldiameter/2
    unitizedlav *= curseg.currenttooldiameter/2

    nextpt = curseg.curve.PointAtEnd + unitizednav
    lastpt = curseg.curve.PointAtStart + unitizedlav

    polylinepts = []

    # updating the basic start and end points of the segment
    movedstartpt = curseg.pointlist[0] + curseg.normals[0]
    movedendpt = curseg.pointlist[-1] + curseg.normals[-1]

    nextstartpt = nextseg.pointlist[0] + nextseg.normals[0]
    lastendpt = lastseg.pointlist[-1] + lastseg.normals[-1]

    # but the average might be behind a start or end, so we have to
prioritize it
    # offsetting the start
    if lastpt.DistanceTo(nextpt) < movedstartpt.DistanceTo(nextpt):
        polylinepts.append(lastpt)
    else:
        polylinepts.append(movedstartpt)

    # offsetting the middle points
    for p, pt in enumerate(curseg.pointlist[1:-1]):
        movedpt = pt + curseg.normals[p]
        polylinepts.append(movedpt)

    # offsetting the end point, maybe with arc if necessary
    if nextpt.DistanceTo(lastpt) < movedendpt.DistanceTo(lastpt):
        polylinepts.append(nextpt)
    else:
        polylinepts.append(movedendpt)
        arc = rg.Arc(movedendpt, nextpt, nextstartpt)

        arcpoints = discretizecurve(arc.ToNurbsCurve())
        for pt in arcpoints[1:]:
            polylinepts.append(pt)

    curseg.movedpointlist = polylinepts
    curseg.relativetoolpath = rg.Polyline(polylinepts)
```





7. calculate the cumulative length of all passes, incl. a finishing pass

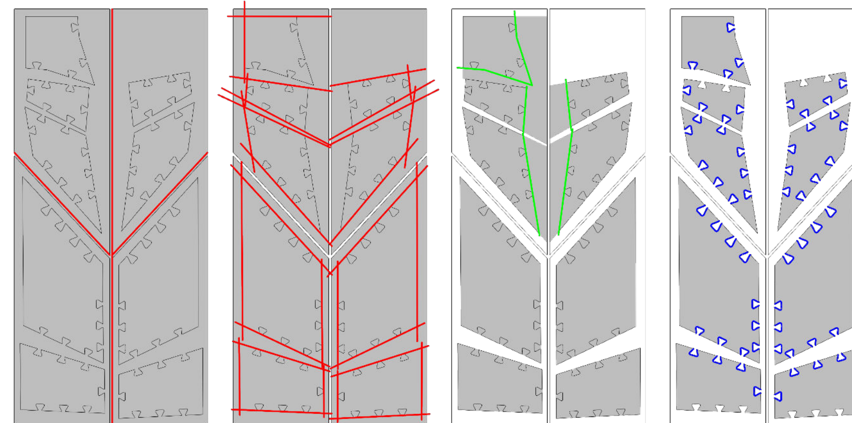
8. calculate the length of the spiral up to and excl. the current segment

9. add the length of the segment up to the current vertice

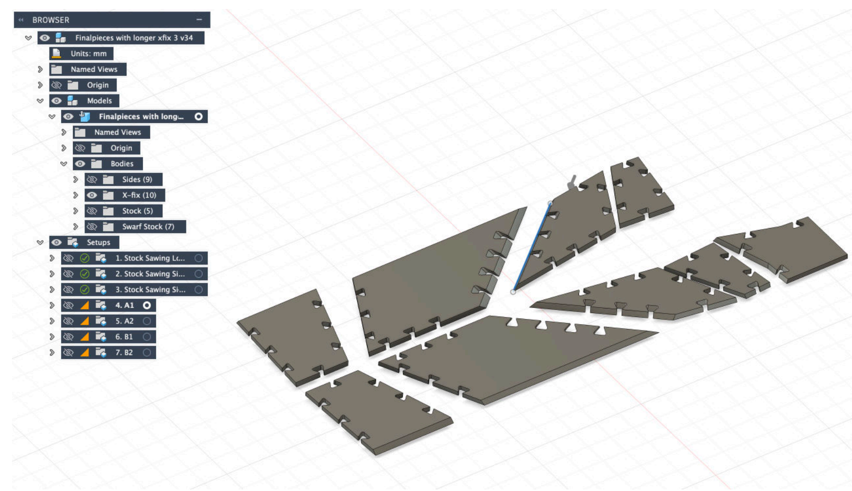
10. deduce the height of the tool at the given vertice

11. calculate the height of the tool at the requested position on the spiral

```
def make_CNCspiral(currentplate, spiralfactor):

    CNCpoints = []
    CNCnormals = []

    unitglobalcurvelength = sum(c.relativetoolpath.Length for c in
currentplate.lowerkeylist)
    globalcurvelength = unitglobalcurvelength * (spiralfactor+1) # this
is the lower edge

    for s in range((len(currentplate.upperkeylist) * (spiralfactor+1))):
# we do it (spiralfactor+1) times to include a finishing pass

        currentindex = s % len(currentplate.upperkeylist)
        spiralindex = s // len(currentplate.upperkeylist)

        addition = (spiralindex * unitglobalcurvelength) +
sum(c.relativetoolpath.Length for c in
currentplate.lowerkeylist[:currentindex])

        csegu = currentplate.upperkeylist[currentindex]
        csegl = currentplate.lowerkeylist[currentindex]

        if len(csegu.movedpointlist) ≠ len(csegl.movedpointlist):
            print("points don't match")

        for p, upperpt in enumerate(csegu.movedpointlist[1:], start = 1):
            lowerpt = csegl.movedpointlist[p]
            vector = upperpt - lowerpt

            nurbs_curve = csegl.relativetoolpath.ToNurbsCurve()
            domain = nurbs_curve.Domain
            success, lowerparam = nurbs_curve.ClosestPoint(lowerpt)
            locallowerparam = lowerparam / nurbs_curve.GetLength()
            globallowerlength = (locallowerparam *
nurbs_curve.GetLength()) + addition
            globalllowerparam = globallowerlength/globalcurvelength

            thresholdv = spiralfactor / (spiralfactor +1)
            depthparam = (globalllowerparam/thresholdv) if
globalllowerparam ⩽ thresholdv else 1
            finalpt = lowerpt + (vector * (1-depthparam))

            CNCpoints.append(finalpt)
            CNCnormals.append(vector)
            a.append(finalpt)

    return CNCpoints, CNCnormals


# example using the center plate
CNCPoints, CNCNormals = make_CNCspiral(centerplate, spfactor)
centerplate.CNCpoints = CNCPoints
centerplate.CNCnormals = CNCNormals

# formula for the tool depth in a simulation
depthparam = (pparam/thresholdv) if pparam ⩽ thresholdv else 1
basepoint = lowerpoint + (vector * (1-depthparam))
```



Machining Workflow
Red: Saw blade
Green: 20mm finger
Blue: 12mm finger



Fusion3D setup

```
gcode = []

# Header
gcode.append("%")
gcode.append("O1000 (FOCUSWORK DOMINIC)")
gcode.append("G90 G21 G17")   # Absolute, metric, XY plane
gcode.append("S10000 M3")     # Spindle on

# Approach safely
gcode.append("G0 Z10")        # Rapid to safe height
gcode.append("G0 X0 Y0")      # Rapid to start XY
gcode.append("G0 Z1")         # Rapid to approach height

for point, vector in zip(centerpoints, centervectors):
    x, y, z = point
    i, j, k = vector
    gcode.append(f"G1 X{x} Y{y} Z{z} A{i} B{j} C{k} F1000")

# Retract safely
gcode.append("G0 Z10")        # Rapid to safe height
gcode.append("M5")            # Spindle off
gcode.append("M30")           # Program end
gcode.append("%")
```

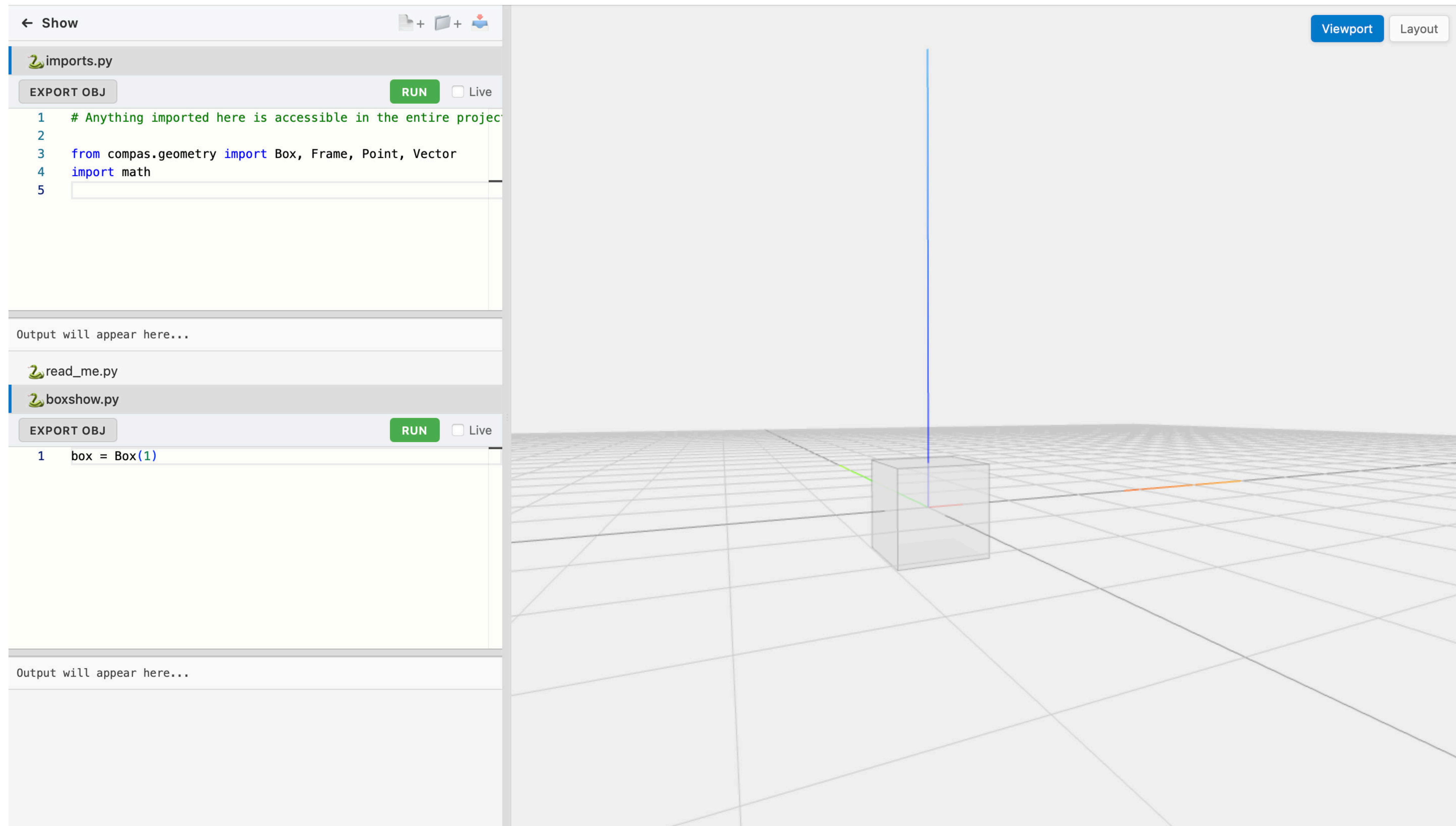
Custom G-Code exporter

compasstudioonline.com
software design
written in javascript, python and html/css

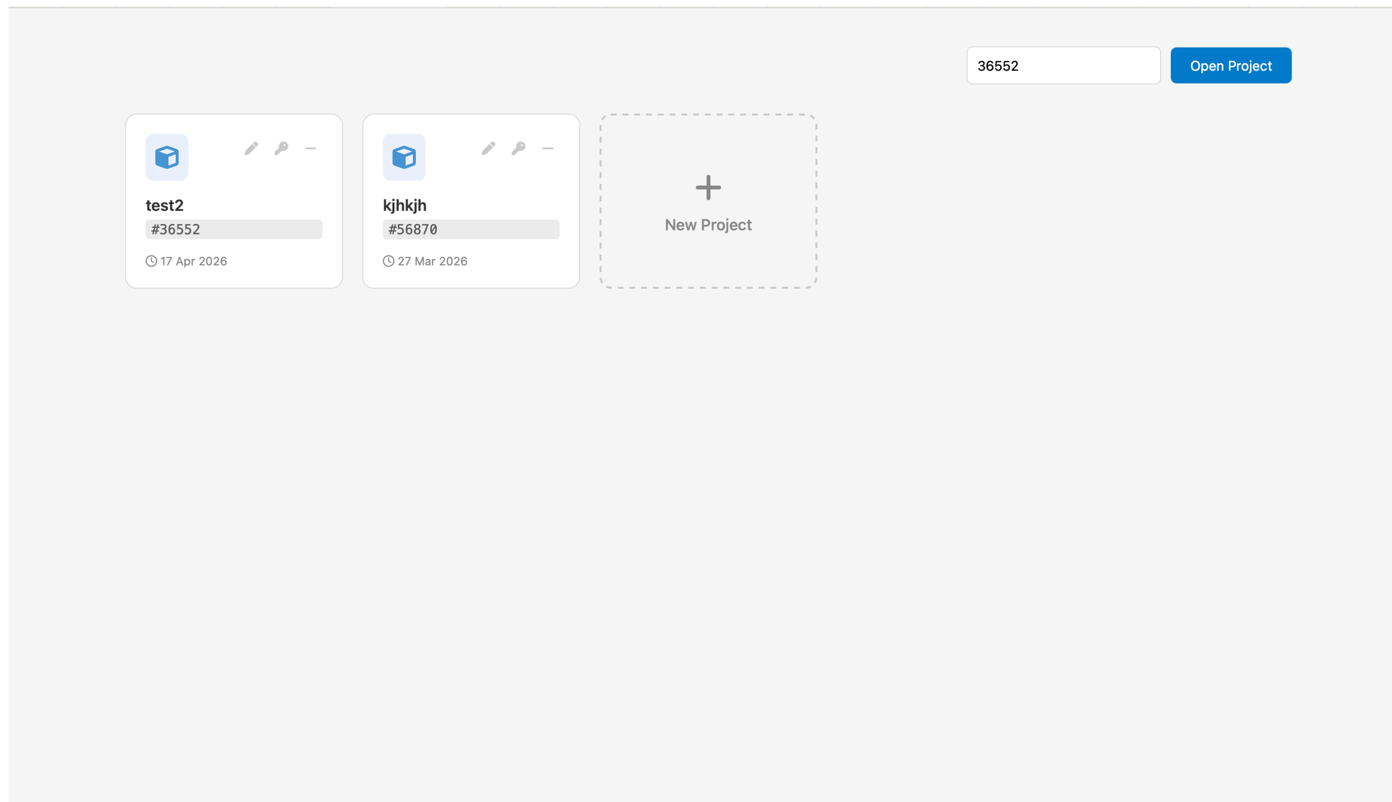
➡ **brief**
Create a web-based design application for designers who code. Bringing together aspects from CAD, BIM, and Graphic interfaces in an excl. written user experience.

➡ **PROBLEMS**
1. Create a functioning terminal using Monaco that successfully visualises geometry.
2. Allow for several users in the same terminal.
3. Integrate viewport-code pipelines.

2



This project consists in a web-based coding terminal for OOC COMPAS users, using Monaco.




```
// --- State ---
let isCreating = false;

// --- Projects ---
async function loadProjects() {
    const container = document.getElementById('project-list');

    try {
        const response = await fetch('/projects');
        if (!response.ok) throw new Error('Failed to fetch');

        const projects = await response.json();
        container.innerHTML = '';

        // 1. Render Existing Projects
        projects.forEach(p => {
            const card = document.createElement('div'); // Changed to
div for click handler
            card.className = 'project-card';
            // card.href = `/${p.key}`; // No link, use click
            card.onclick = () => openEditor(p.key);

            const date = new Date(p.created *
1000).toLocaleDateString(undefined, {
                year: 'numeric', month: 'short', day: 'numeric'
            });

            // Prevent navigating when clicking actions
            const deleteHandler = `event.preventDefault();
event.stopPropagation(); deleteProject('${p.key}')`;
            const copyHandler = `event.preventDefault();
event.stopPropagation(); copyKey('${p.key}')`;
            // Escape single quotes for JS string
            const safeName = p.name ? p.name.replace(/'/g, "\\'") : '';
            const renameHandler = `event.preventDefault();
event.stopPropagation(); renameProject('${p.key}', '${safeName}')`;

            card.innerHTML = `
                <div class="card-header">
                    <div class="project-icon"><i class="fas fa-cube"></
i></div>
                    <div class="card-actions">
                        <button class="action-btn btn-rename"
onclick="${renameHandler}" title="Rename Project">
                            <i class="fas fa-pen"></i>
                        </button>
                        <button class="action-btn btn-copy" onclick="$
{copyHandler}" title="Copy Key">
                            <i class="fas fa-key"></i>
                        </button>
```

The interface includes a project page. Each project is a Juypter Kernel, which allows for several users to simultaneously code in the same project, using Monaco Collab.

← test2

brickwall.py

EXPORT OBJ RUN Live

```
1  print("ok")
2  # box = Box(1,1,1)
3
4  col = 17 # range(1,100)
5  row = 10 # range(1,100)
6  glb_boxes = []
7
8  zaxis = Vector(0,0,1)
9
10 for i in range(col):
11     for j in range(row):
```

ok

dslkd.py

imports.py

read_me.py

test.py

Viewport Layout

```
def _get_mesh_data(obj):
    # Quick exit for basic primitives
    if isinstance(obj, (int, float, str, bool, bytes, dict, set)):
        return None

    # Try standard mesh/shape method
    if hasattr(obj, 'to_vertices_and_faces'):
        try:
            _v, _f = obj.to_vertices_and_faces()
            return {'vertices': [list(pt) for pt in _v], 'faces': _f}
        except: pass

    # Try converting Primitive/Shape to Mesh
    try:
        mod = getattr(type(obj), '__module__', '')
        if mod and mod.startswith('compas'):
            from compas.datastructures import Mesh
            _m = Mesh.from_shape(obj)
            _v, _f = _m.to_vertices_and_faces()
            return {'vertices': [list(pt) for pt in _v], 'faces': _f}
    except: pass
    return None
```

The viewport uses ThreeJS. COMPAS geometry is meshed and represented in the viewport. Clicking on an object in the viewport will add its variable name in the terminal at the current cursor.

reciprocal frames

digital design and fabrication
written in c# and python using RhinoCommon
fabricated using a laser-cutter

brief

Create free shape roof canopies using the principle of reciprocal framing, allowing for short pieces and a material-efficient structural performance.

PROBLEMS

1. Creating the structural morphology.
2. Solving free form best case scenarios.
3. Solving wood-wood joints.
4. Producing optimised fabrication files.

3

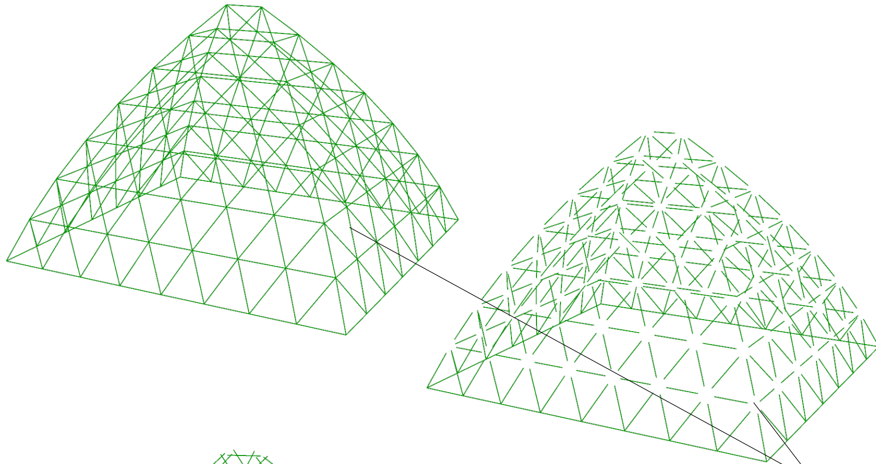
1 — STRUCTURAL MORPHOLOGY

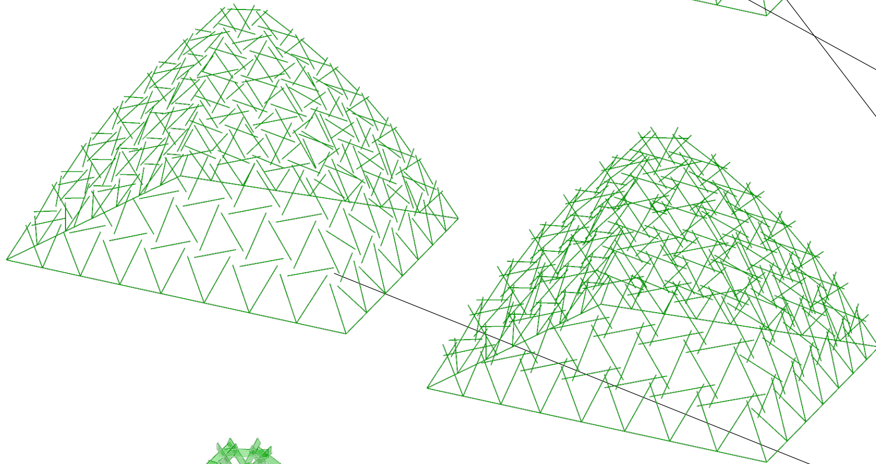
outset

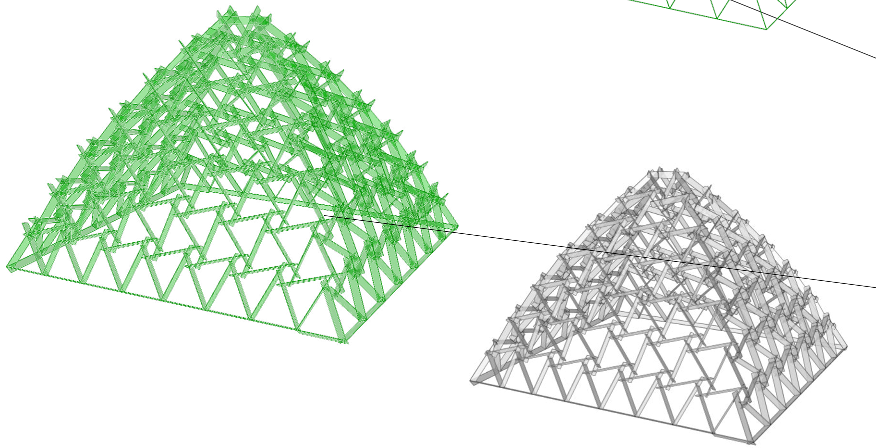
Using the principle of reciprocal framing, create roofs allowing for short pieces and a material-efficient structural performance.

PROBLEM

Specify a principle-coherent morphology.







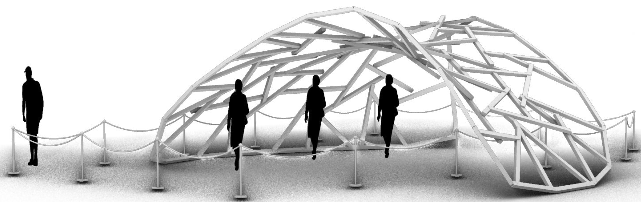
1. define the grid of nodes

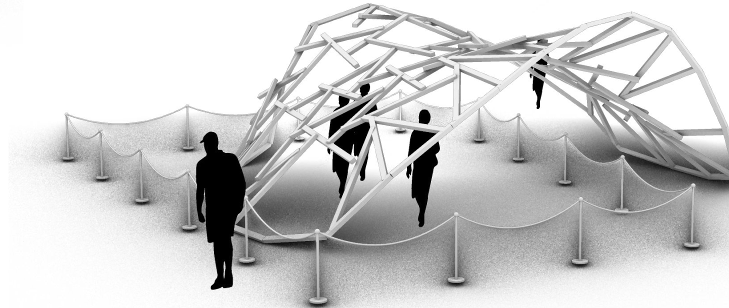
2. define a node excentricity

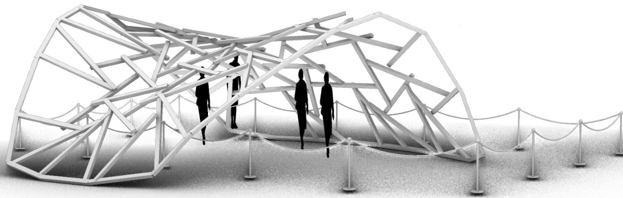
3. twist each node by replacing the start and end points of each segment

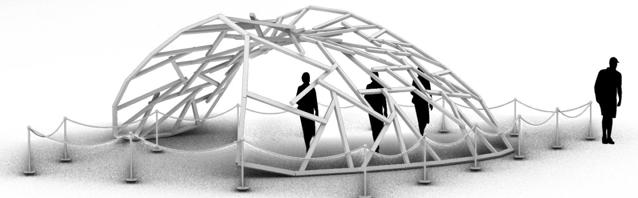
4. lengthen the lines to create intersecting boards

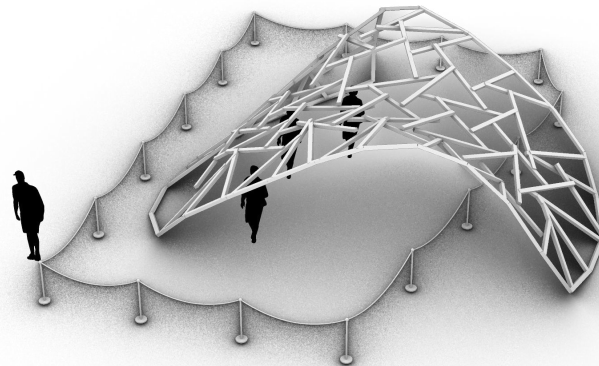
```
for s, side in enumerate(sides):
    for l, level in enumerate(side):
        for p, pt in enumerate(level):
            firstcurve = flatlines[pt.curves[0]]

            for c, crv in enumerate(pt.curves):

                curve = flatlines[crv]
                crvvec = curve.PointAtEnd - curve.PointAtStart
                crvvec.Unitize()

                if c == len(pt.curves) -1:
                    nextcurve = firstcurve
                else:
                    nextcurve = flatlines[pt.curves[(c+1)]]

                if pt.centre.DistanceTo(nextcurve.PointAtStart) <
pt.centre.DistanceTo(nextcurve.PointAtEnd):
                    newpt = nextcurve.PointAtStart
                else:
                    newpt = nextcurve.PointAtEnd

                if pt.centre.DistanceTo(curve.PointAtStart) <
pt.centre.DistanceTo(curve.PointAtEnd):
                    oldpt = curve.PointAtStart
                    newpt1 = newpt
                    newpt2 = curve.PointAtEnd

                else:
                    oldpt = curve.PointAtEnd
                    newpt1 = curve.PointAtStart
                    newpt2 = newpt

                newcurve = rg.LineCurve(newpt1, newpt2)
                flatlines[crv] = newcurve
```







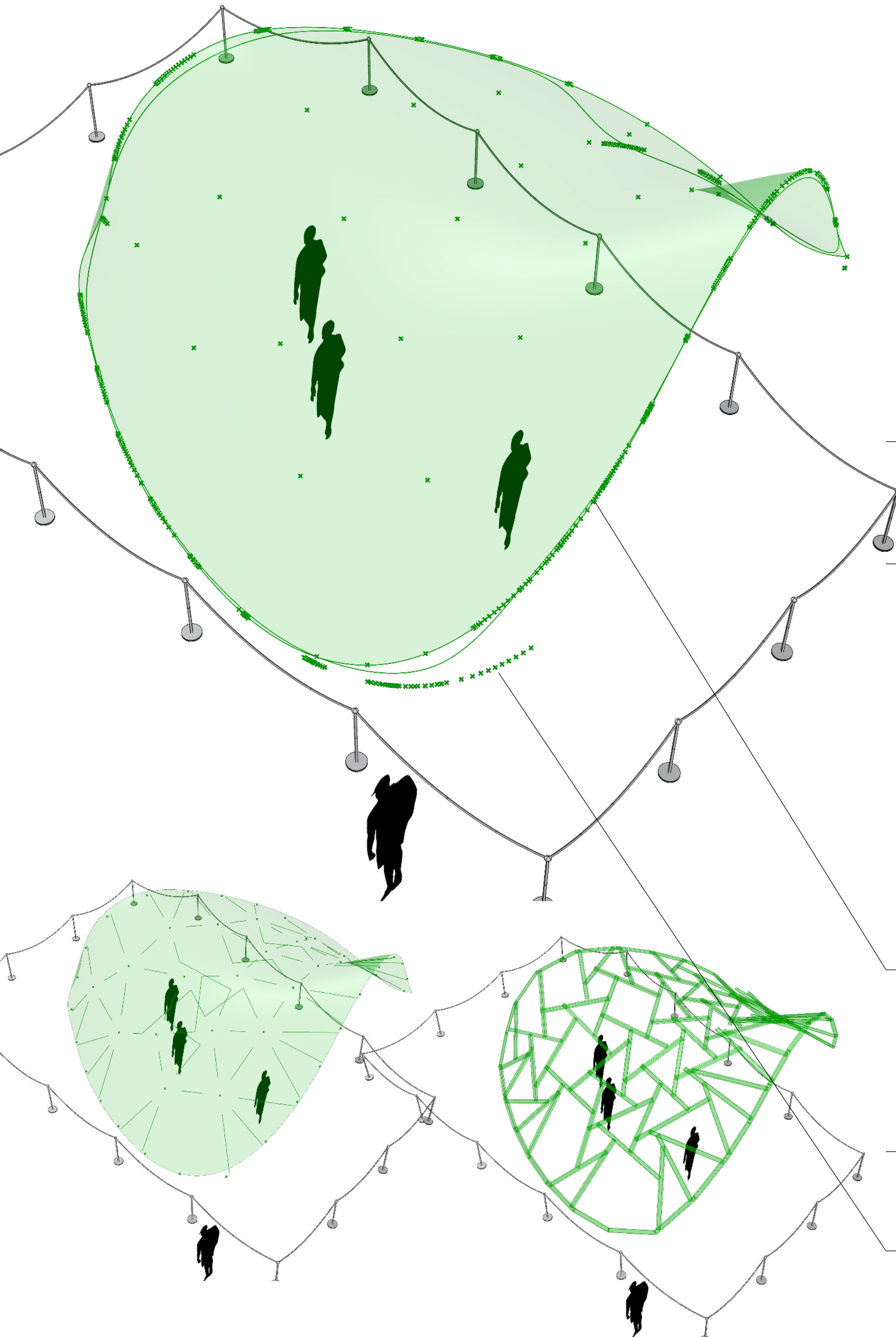




2 — OPTIMISING FREE FORM

outset
Using the principle of reciprocal framing, create roofs allowing for short pieces and a material-efficient structural performance.

PROBLEM
In a free form, find the best possible position of the reciprocal nodes through iterative optimisation.



1. from the neighbors, derive an average repulsionvector

2. check if the moved point is on the boundary or a foundation node

3. in this case of a boundary node, the node gets moved along the boundary

4. if the node moves too far away from its original position, it is culled

5. one can thus follow the positions of the nodes as they are optimized for n iterations

```
for (int i = 0; i < repulsionIterations; i++)
{
    for (int j = 0; j < count; j++)
    {
        if (uniqueCentroids[j] != Point3d.Unset)
        {
            Vector3d force = Vector3d.Zero;
            for (int k = 0; k < count; k++)
            {
                if (j == k) continue;
                if (uniqueCentroids[k] != Point3d.Unset)
                {
                    double dist =
uniqueCentroids[j].DistanceTo(uniqueCentroids[k]);
                    if (dist < repulsionRadius)
                    {
                        Vector3d dir = uniqueCentroids[j] -
uniqueCentroids[k];
                        dir.Unitize();
                        force += dir * (repulsionRadius - dist);
                    }
                }
            }

            // Apply movement constraints
            if (isBoundaryPoint[j])
            {
                if (isFloorPoint[j])
                {
                    Curve closestCrv = null;
                    double closestT = 0;
                    foreach (Curve crv in floorCurves)
                    {
                        crv.ClosestPoint(uniqueCentroids[j], out double t);
                        if (closestCrv == null ||
                            uniqueCentroids[j].DistanceTo(crv.PointAt(t)) <

uniqueCentroids[j].DistanceTo(closestCrv.PointAt(closestT)))
                        {
                            closestCrv = crv;
                            closestT = t;
                        }
                    }

                    Vector3d tangent = closestCrv.TangentAt(closestT);
                    double dot = force * tangent;
                    Vector3d boundaryForce = tangent * dot;

                    Point3d newPt = uniqueCentroids[j] + boundaryForce * 0.1;
                    closestCrv.ClosestPoint(newPt, out closestT);

                    double distToFirst =
FloorPts[j].DistanceTo(closestCrv.PointAt(closestT));

                    if (distToFirst > maxOrigDist){
                        uniqueCentroids[j] = Point3d.Unset;
                    }
                    else {
                        uniqueCentroids[j] = closestCrv.PointAt(closestT);
                        afterRepFloor.Add(closestCrv.PointAt(closestT));
                    }
                }
                else
                    …
```

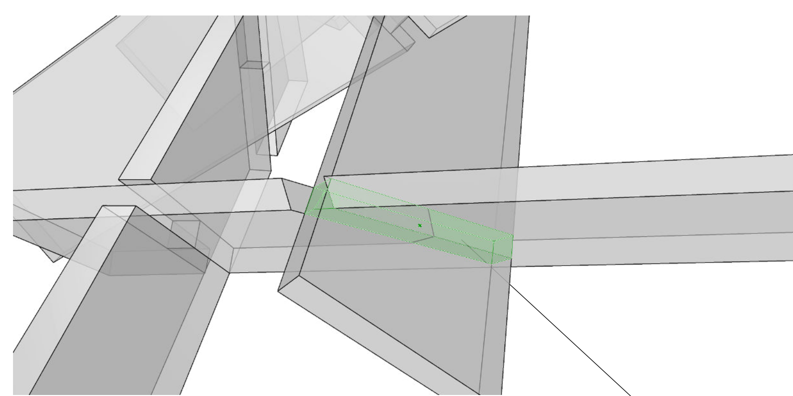
3 — JOINTS

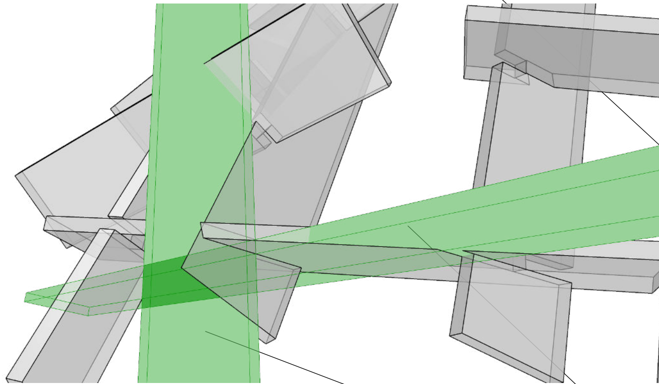
outset

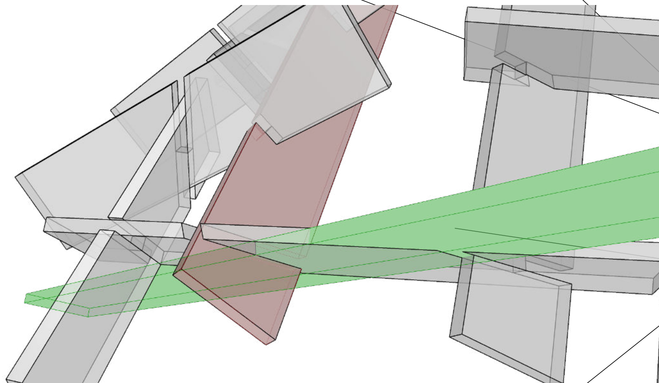
Using the principle of reciprocal framing, create roofs allowing for short pieces and a material-efficient structural performance.

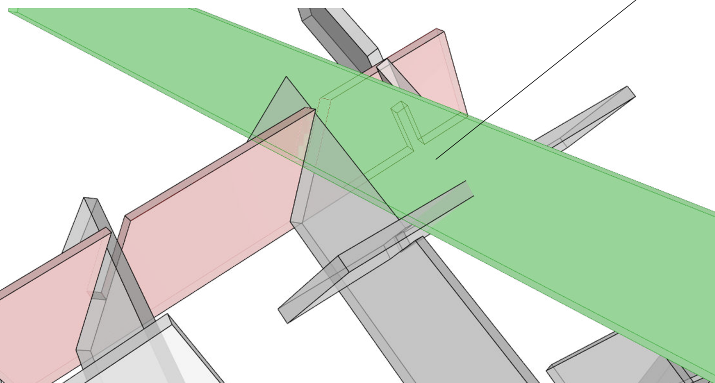
PROBLEM

Specify wood-wood joints.







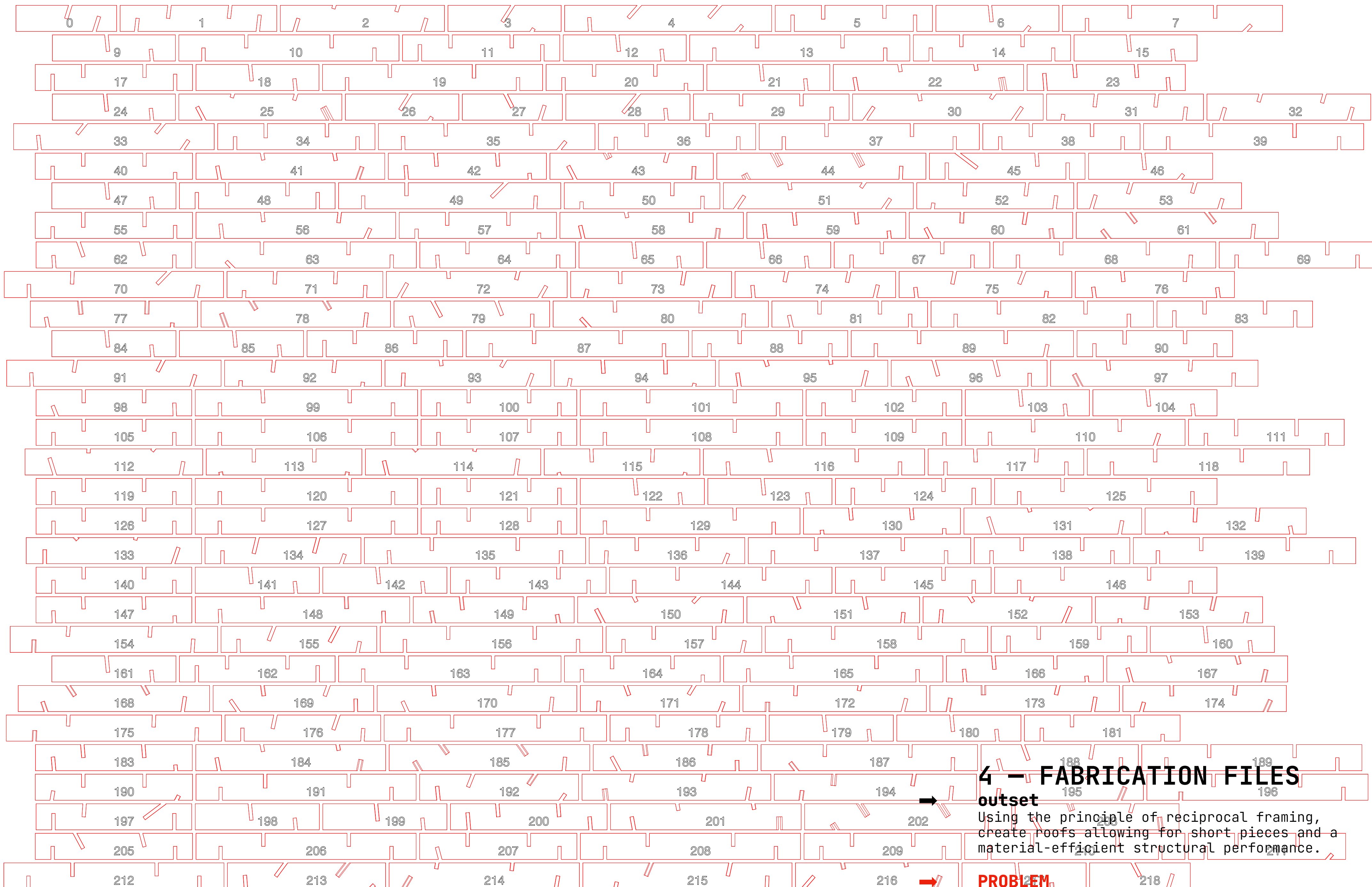


1. compute the key intersection geometry and derive all necessary data for the joint

2. create cutting geometries

3. produce the geometric difference

```
def XJoint(untenbeam, obenbeam, bias, normal):

    intersection = rg.Brep.CreateBooleanIntersection(
        untenbeam.brep,
        obenbeam.brep,
        0.1
    )

    keybrep = intersection[0]
    centroid = rg.AreaMassProperties.Compute(keybrep).Centroid

    obencentroid = centroid + (normal * (obenbeam.height /2))
    untencentroid = centroid - (normal * (untenbeam.height/2))

    obencrossp = rg.Vector3d.CrossProduct(obenbeam.plane.YAxis, normal)
    obencrossp.Unitize()
    untencrossp = rg.Vector3d.CrossProduct(untenbeam.plane.YAxis, normal)
    untencrossp.Unitize()

    obenhalfplane = rg.Plane(obencentroid, obencrossp, obenbeam.plane.YAxis)
    untenhalfplane = rg.Plane(untencentroid, untencrossp,
untenbeam.plane.YAxis)

    cuttingunten = planetrans(untenbeam, untenhalfplane)
    cuttingoben = planetrans(obenbeam, obenhalfplane)

    cuttingunten = singlescaling(cuttingunten, untenhalfplane)
    cuttingoben = singlescaling(cuttingoben, obenhalfplane)

    unten = booldiff(untenbeam.brep, cuttingoben)
    oben = booldiff(obenbeam.brep, cuttingunten)

    return unten, oben
```



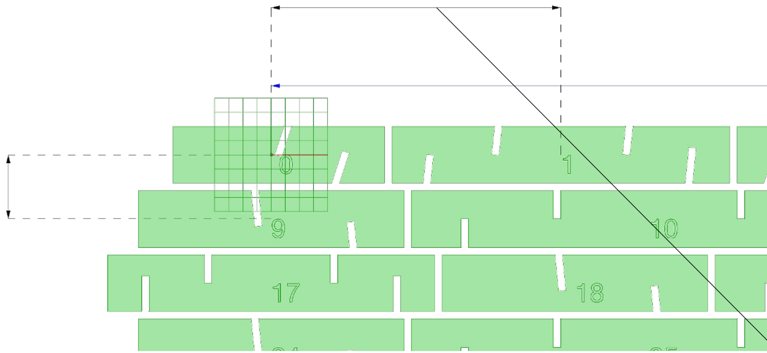
4 — FABRICATION FILES

outset

Using the principle of reciprocal framing, create roofs allowing for short pieces and a material-efficient structural performance.

PROBLEM

Provide an optimized fabrication file pipeline.

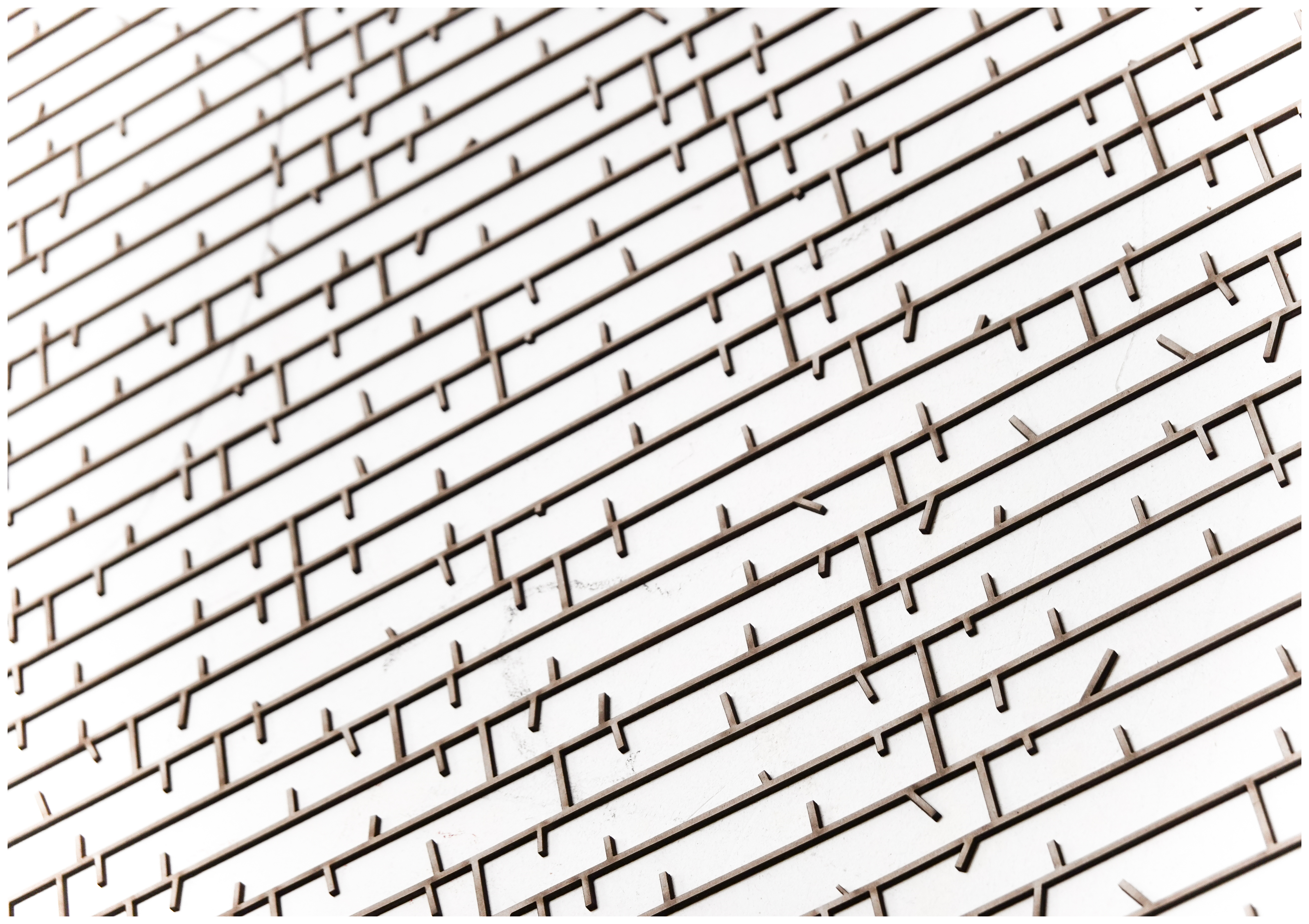


1. determine the distance between two centres

2. check if we fit in the page

3. scale and place the pieces

```
lasthalf = -beams[0].length/2
for b, beam in enumerate(beams):

    currenthalf = beam.length/2
    currentlength = lasthalf + currenthalf

    if limit > maxlimit:
        limit = 0
        nextorigin = xorigin - (rg.Vector3d.YAxis * ((beam.height/20) +
tolerance))
        xorigin = nextorigin
        nextplane = rg.Plane(nextorigin, rg.Vector3d.XAxis,
rg.Vector3d.ZAxis)
        limit += ((currentlength/20) + tolerance)
    else:
        nextorigin = nextorigin + (rg.Vector3d.XAxis * ((currentlength/20) +
tolerance))
        nextplane = rg.Plane(nextorigin, rg.Vector3d.XAxis,
rg.Vector3d.ZAxis)
        limit += ((currentlength/20) + tolerance)

    currentbeam = beam.brep
    currentplane = beam.plane

    scaledbrep = scale(currentplane.Origin, currentbeam)
    scaledbrep = planetrans(scaledbrep, currentplane, nextplane)
    lasthalf = currenthalf
```



generative wall

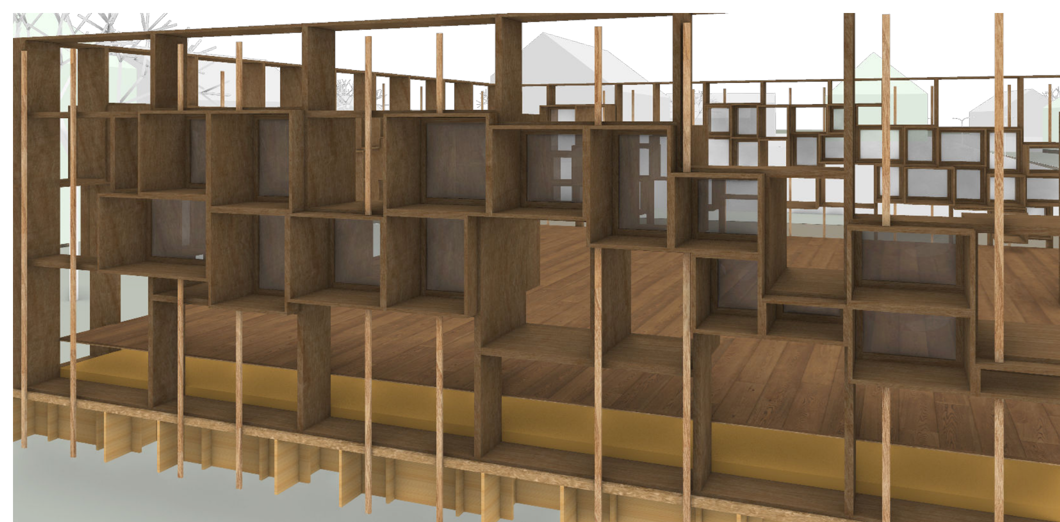
generative design
written in python using RhinoCommon

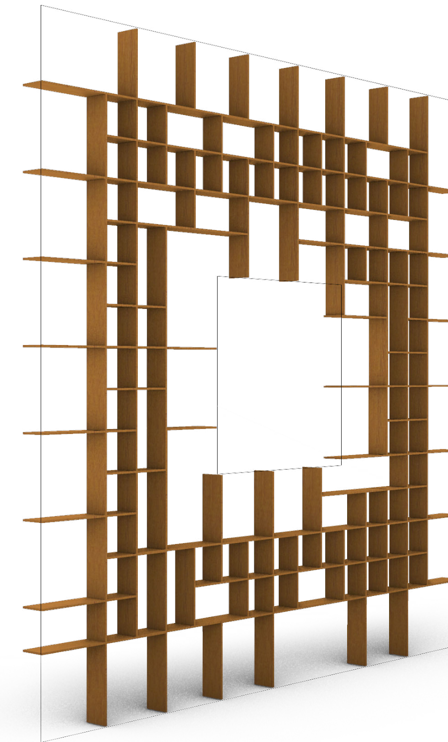
➡ brief

Create a wall made of a set of randomly dimensioned timber boards. The wall adapts itself to any requested opening and dimension.

➡ PROBLEMS

1. Identify a logic that can solve this problem.
2. Write an algorithm that satisfies the brief.
3. Curate the generated solutions for the desired outcome.





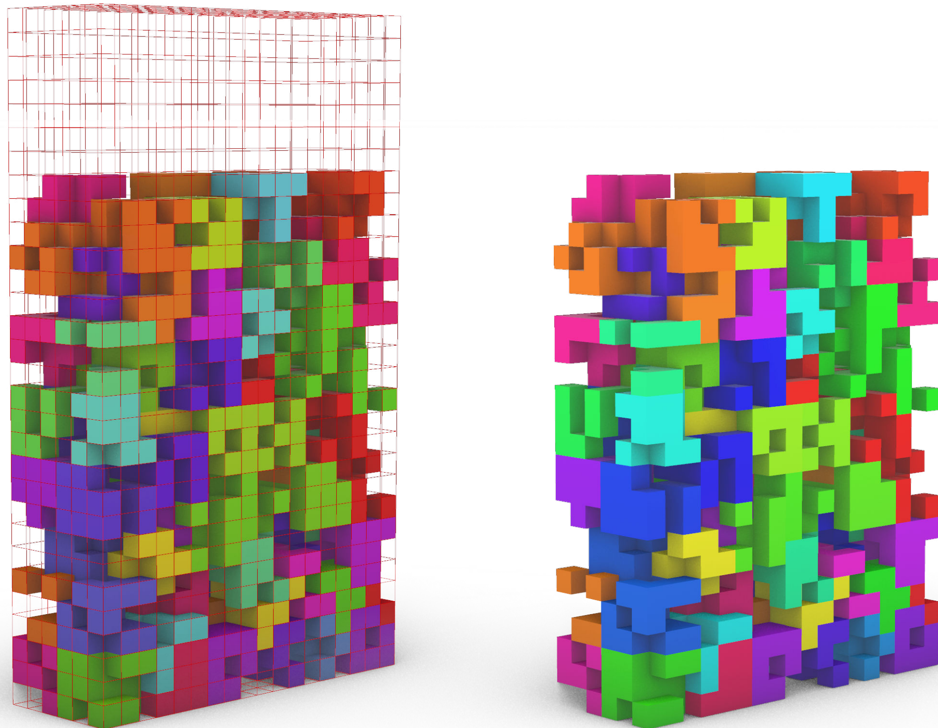
1 — GENERATIVE DESIGN

outset

We have a random set of timber boards, which we want to assemble into a wall of given dimensions. The created cells are filled with straw or windows.

PROBLEM

Identify a logic that can solve this wicked problem.



Here an example for a generative design problem. We have the wicked problem of populating a given space with a random set of random objects. It is solved by iterations of placing objects in the space, checking if a given voxel is already populated, placing an object if not, otherwise continuing to the next iteration, thus slowly filling the space.

The code is only rule; there are many solutions, which the designer then curates, adapting the rules depending on design requirements.

Generative design

dʒenərətɪv dɪzaɪn adjective noun

is **a process that can solve problems with a large or unknown solution set**.[1]

It's an iterative design process, generating outputs for evolving design requirements, which get adjusted iteratively by the designer.
It works because computing power can evaluate more design permutations than a human.

→ examples: cellular automata, shape grammar, genetic algorithm, space syntax, artificial neural network

Architecture is known as a **wicked problem**, meaning a problem that is difficult or impossible to solve because of incomplete, contradictory or changing requirements that are often difficult to recognize.[2]

This project is no different: **the set of material to be used is unknown or random, yet we still want to build a wall that has defined dimensions and openings.**

Further projects where these logics could be and are applied include sustainable design, facade design, structural optimisation and other similar topics.

[1] Meintjes, Keith. ""Generative Design" – What's That? - CIMdata". Retrieved 2018-06-15.
[2] Johnston, Jane; Gulliver, Robyn (2022). "What are wicked problems?". Commons Social Change Library.

2 — ALGORITHM

outset

We have a random set of timber boards, which we want to assemble into a wall of given dimensions. The created cells are filled with straw.

PROBLEM

Write an algorithm that takes as argument a random set of random boards and assembles them within a given shape.

1. the def **branching** is given two curves, a curve to branch and its mother

2. iterate through the available stock, excepting the mother board, meaning the current support board

3. we check whether the current curve intersects with any existent placed curve

4. if either side of it does, that side does not continue branching

5. the length of the line is adjusted to meet any existent lines correctly

6. to avoid very thin boxes, lines that create close parallels do not continue branching

7. new branches are added to the branch_queue to continue branching themselves

```
def branching(current_crv, mother):
    current_firstpt = current_crv.PointAtStart
    current_lastpt = current_crv.PointAtEnd

    current_vector = current_lastpt - current_firstpt

    abort_first_half = False
    abort_second_half = False

    for crv2 in stockcrv:
        if crv2 == mother:
            continue
        inter = rg.Intersect.Intersection.CurveCurve(current_crv, crv2,
0.1, 0.1)
        if inter.Count >0:
            for int_ind in range(inter.Count):
                sec_pt = inter[int_ind].PointA
                points.append(sec_pt)
                param = inter[int_ind].ParameterA
                domain = first_crv.Domain
                normalized_param = param / current_crv.GetLength()

                if normalized_param < 0.5:
                    abort_first_half = True
                    if sec_pt.DistanceTo(current_crv.PointAtStart) >
current_firstpt.DistanceTo(current_crv.PointAtStart):
                        current_firstpt = sec_pt
                elif normalized_param > 0.5:
                    abort_second_half = True
                    if sec_pt.DistanceTo(current_crv.PointAtEnd) >
current_lastpt.DistanceTo(current_crv.PointAtEnd):
                        current_lastpt = sec_pt

    revised_line = rg.LineCurve(current_firstpt, current_lastpt)

    start_too_close = False
    end_too_close = False

    if not abort_first_half:
        rayend = current_firstpt - current_vector * 1000000
        ray = rg.LineCurve(current_crv.PointAtMid, rayend)
        current_firstpt, start_too_close = IsTooClose(current_firstpt,
ray, current_vector, mother, stockcrv)
    if not abort_second_half:
        rayend = current_lastpt + current_vector * 1000000
        ray = rg.LineCurve(current_crv.PointAtMid, rayend)
        current_lastpt, end_too_close = IsTooClose(current_lastpt, ray,
current_vector, mother, stockcrv)

    revised_line = rg.LineCurve(current_firstpt, current_lastpt)

    if revised_line not in stockcrv:
        stockcrv.append(revised_line)
        crv_list.append(revised_line)

    if not abort_second_half and not end_too_close:
        line2 = orthline(revised_line.PointAtEnd, revised_line)
        branch_queue.append((line2, revised_line))  # RIGHT branch added
FIRST

    if not abort_first_half and not start_too_close:
        line1 = orthline(revised_line.PointAtStart, revised_line)
        branch_queue.append((line1, revised_line))
```

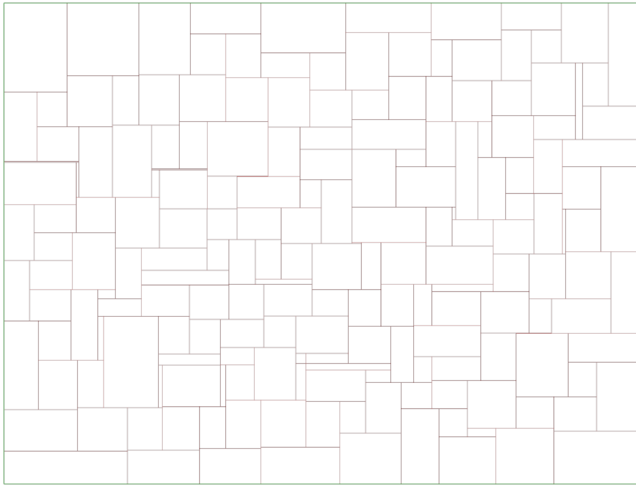
3 — CURATION

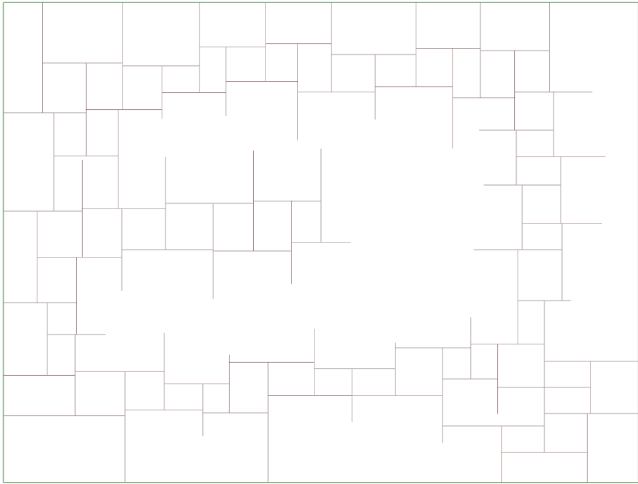
outset

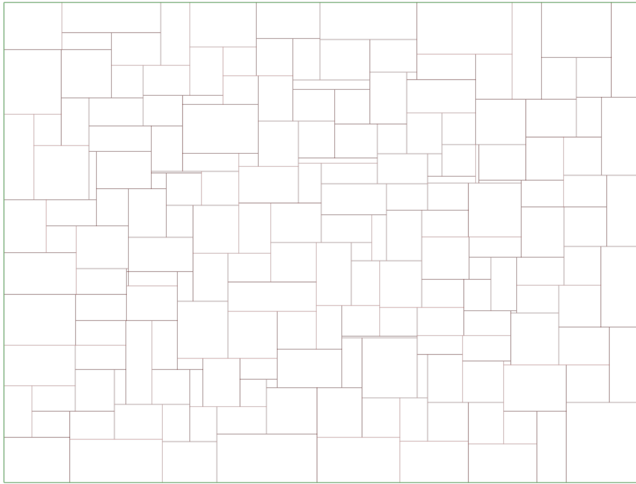
We have a random set of timber boards, which we want to assemble into a wall of given dimensions. The created cells are filled with straw.

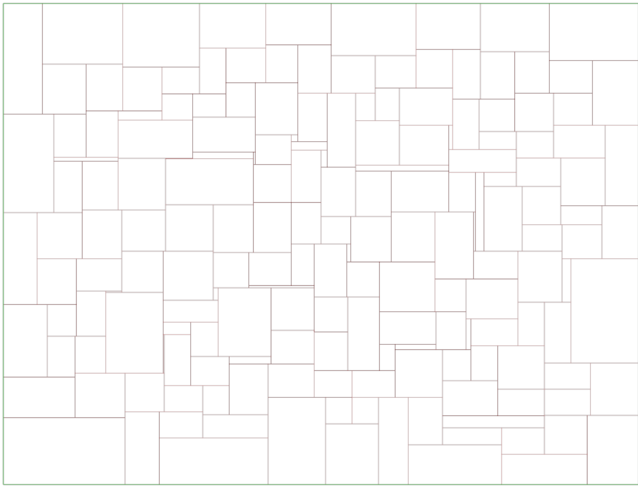
PROBLEM

Realize the factors available to the designer, and segregate the solutions intentionally.

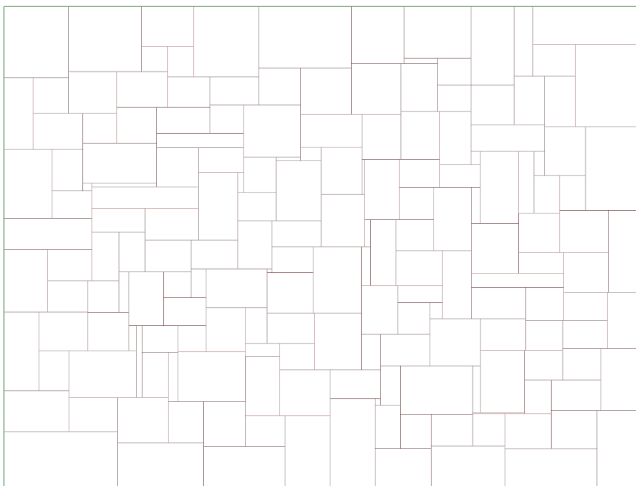




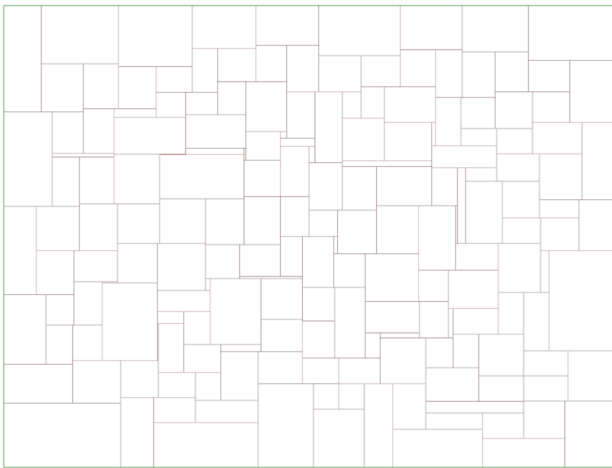


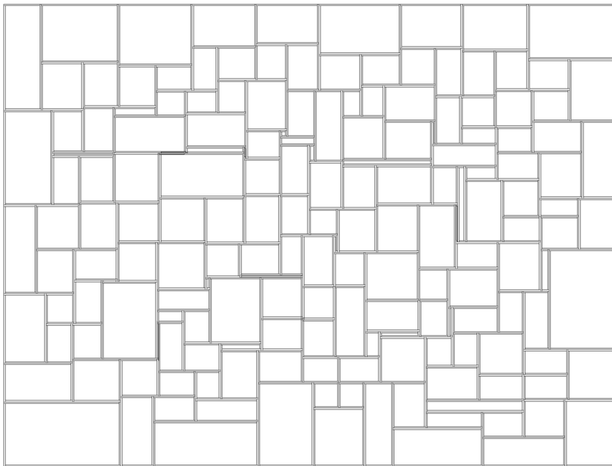


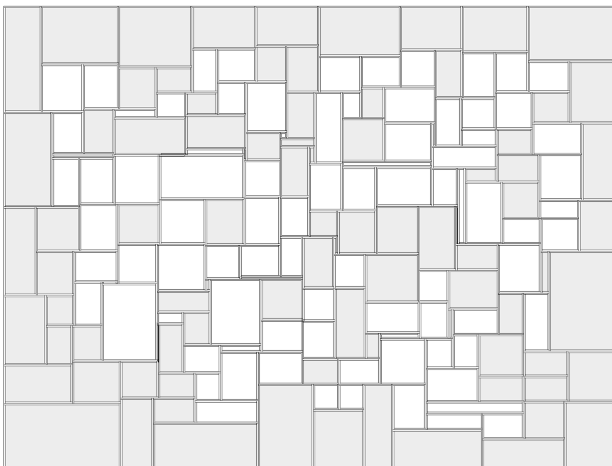
Following the branching until the desired shape is filled.



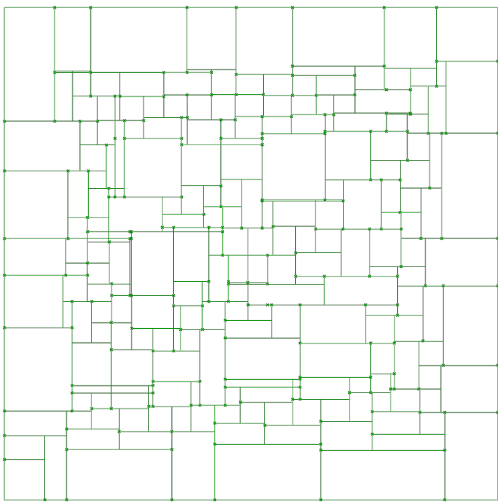
Many solutions for the same problem are generated, and then segregated to obtain the desired result.

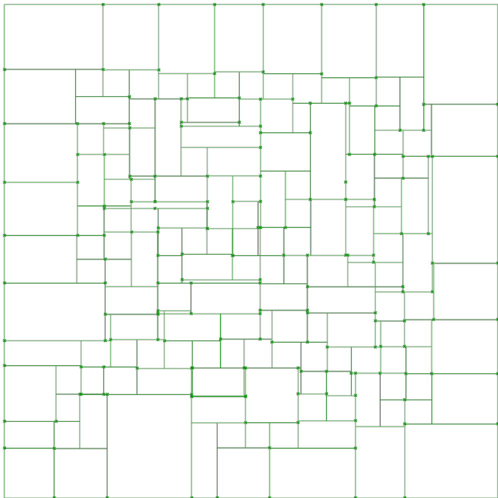


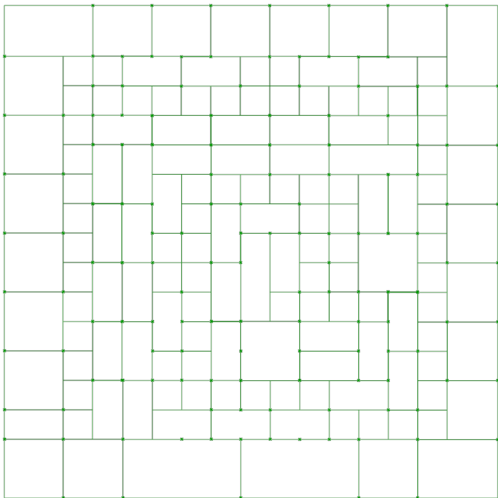




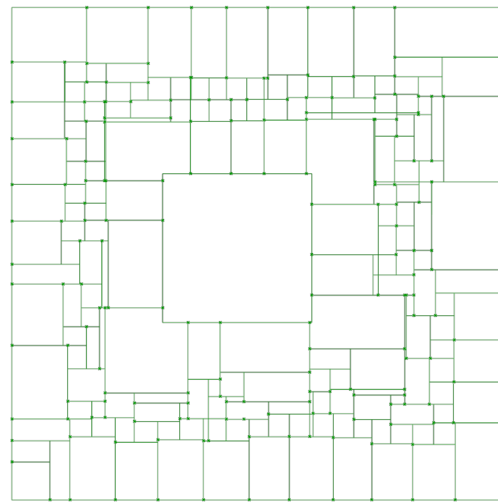
From the branching the actual geometry of the boards can then be deduced, and a given percentage of surface can be specified for windows.

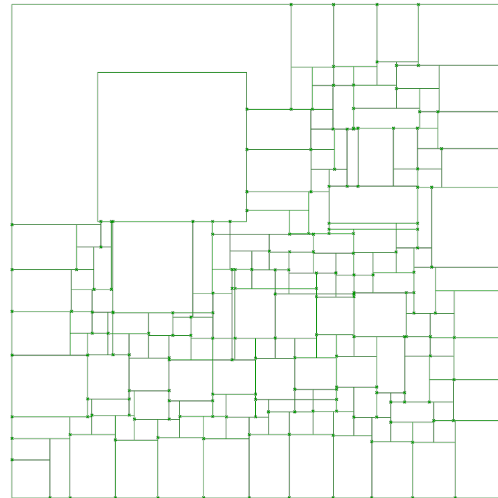


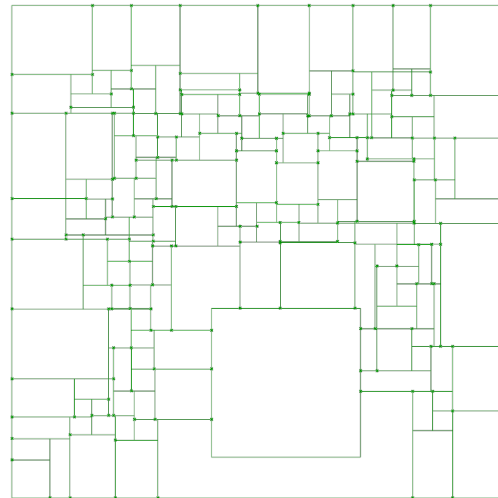




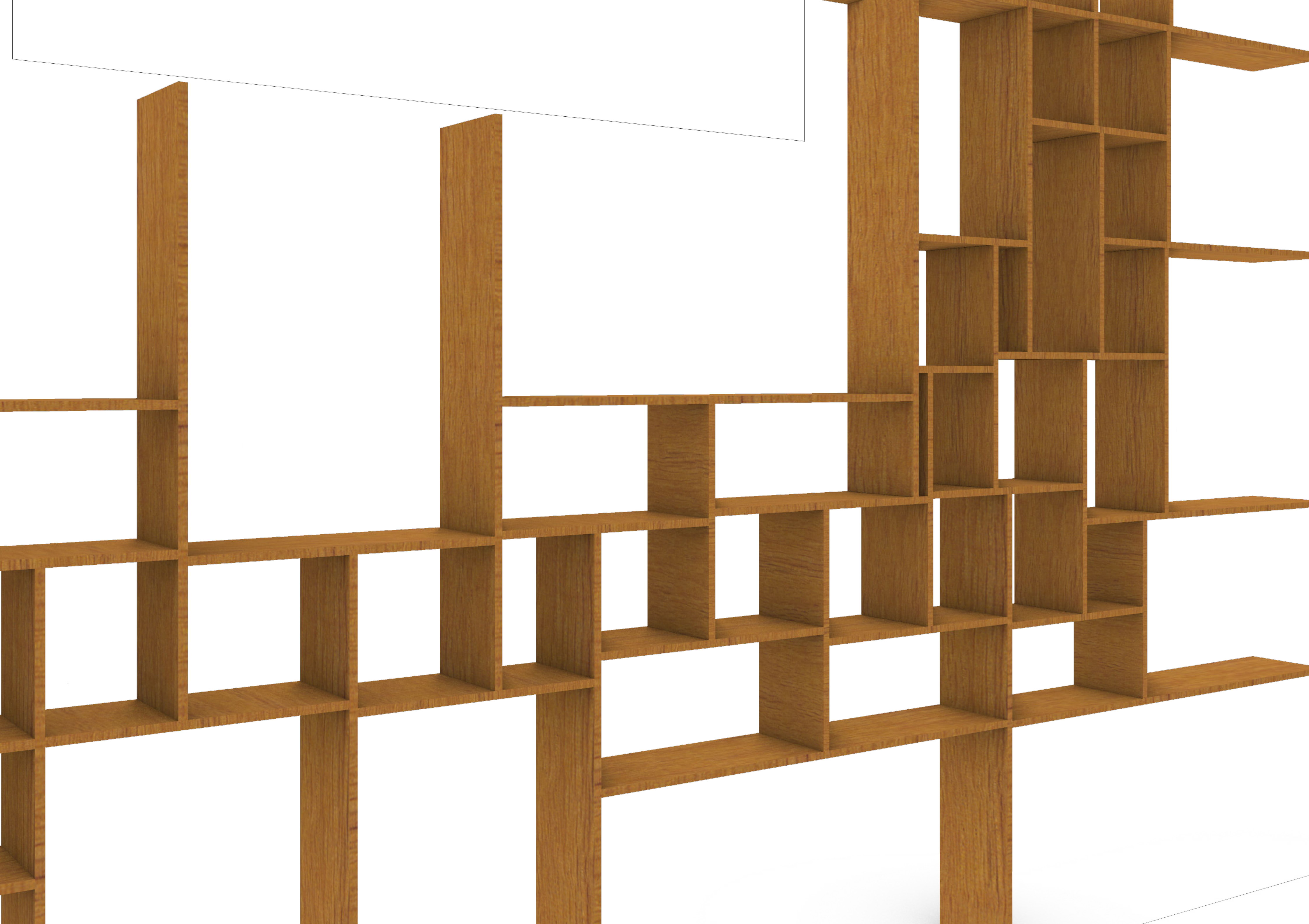
The design is influenced by the domain of dimensions of the given timber set. The larger the domain, the more the design becomes irregular.







The algorithm is able to respond to any opening in any position.



5

lit vacuum

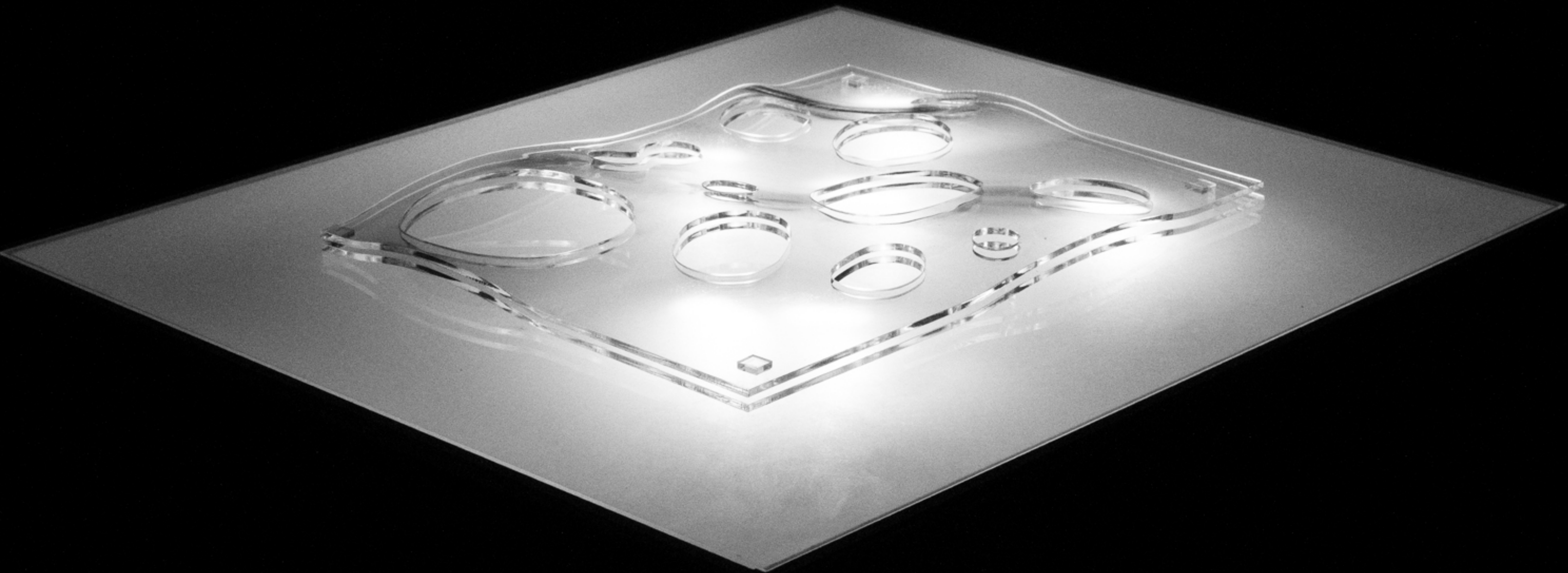
fabrication and product design
modelled in rhino
fabricated using a 3-axis CNC, a laser-cutter and a thermo-vacuum former

➡ brief

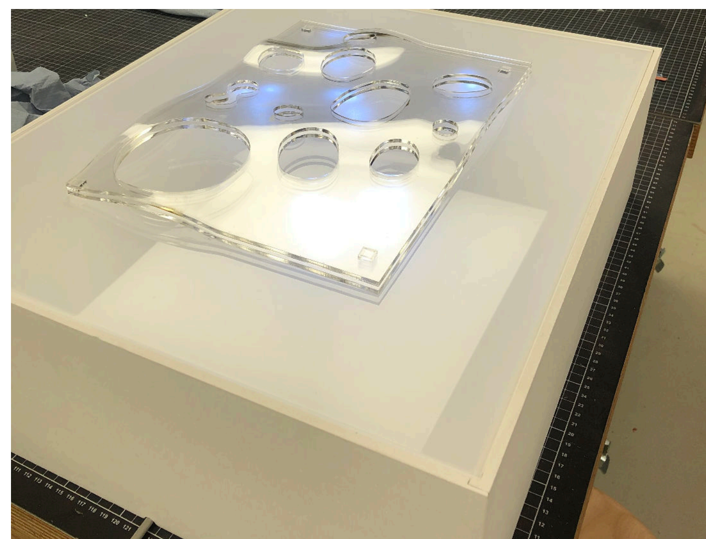
Fabricate a 1:500 physical model of the Rolex Learning Center in Lausanne, turning the problem of fabricating the geometry into a design strength.

➡ PROBLEMS

1. Providing a viable fabrication method.
2. Making efficient material, finish and fabrication choices.



A mold is cut by 3-Axis CNC. 3mm Acrylic is then vacuum-thermo molded on top of it.



The Acrylic sheets are lasercut into their final dimensions, and the holes are cut.
In the custom base, the lights are inserted at specific positions to highlight the shape optimally.

scar-city

design and project management
written in python using RhinoCommon
rendered in twinmotion, post-pro in photoshop

brief

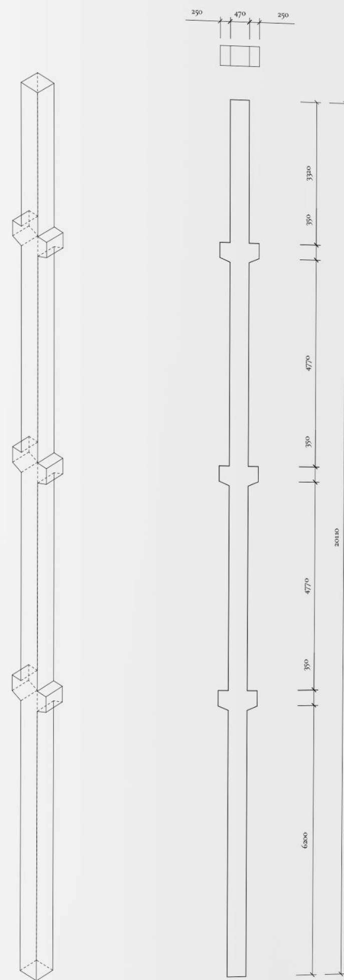
Reusing dissembled building components from a catalogue, design housing that is radically CO2-positive, pushes typologies and establishes a clear design language.

PROBLEMS

1. Providing a structural solution that can dynamically adapt to given building components to achieve desired dimensions.
2. Calculate and optimize the CO2 footprint of the design.
3. Establish a coherent design language throughout all levels and scales of the project.

C-001-PLB

14



ID	C-001-PLB	Year of construction	1970
Description	Reinforced concrete column	Distance to HB Zürich	90
Material	Concrete	Status	Dismanteled
Dimension (m)	20.11 x 0.47 x 0.57	Storage	Walkeweg, Basel
Amount	54	CO_2e [kg/m³]	440
Mine	Parkhaus Lysbüchel, Basel	Source	Zirkular

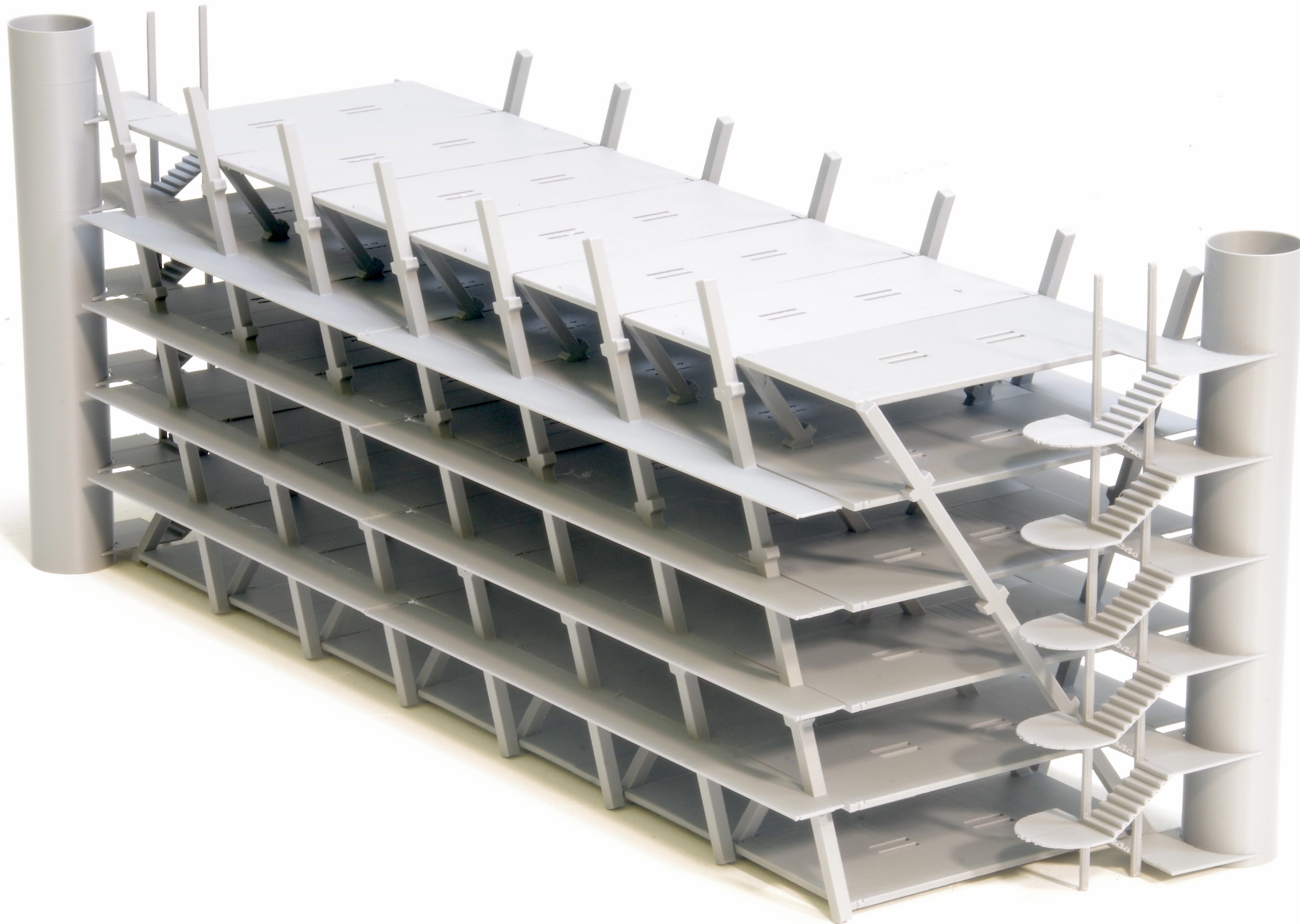
1 — STRUCTURAL MORPHOLOGY

➡ **outset**

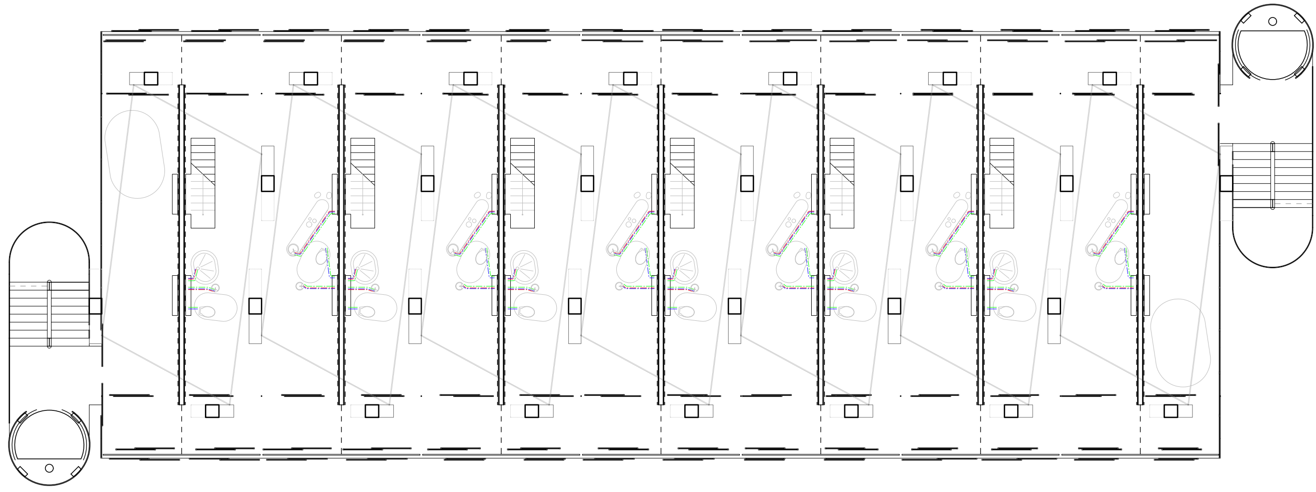
A catalogue of dissembled building components to be reused for housing.

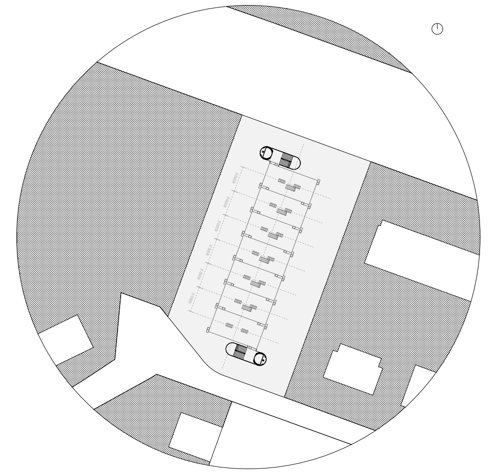
➡ **PROBLEM**

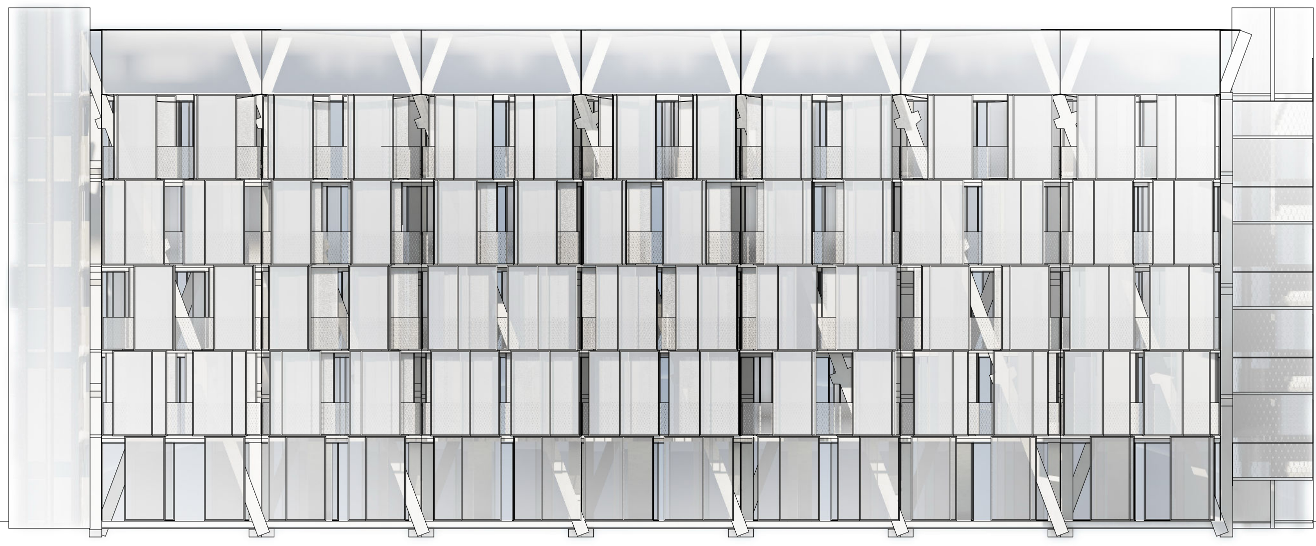
Providing a structural solution that can dynamically adapt to given building components to achieve desired dimensions.

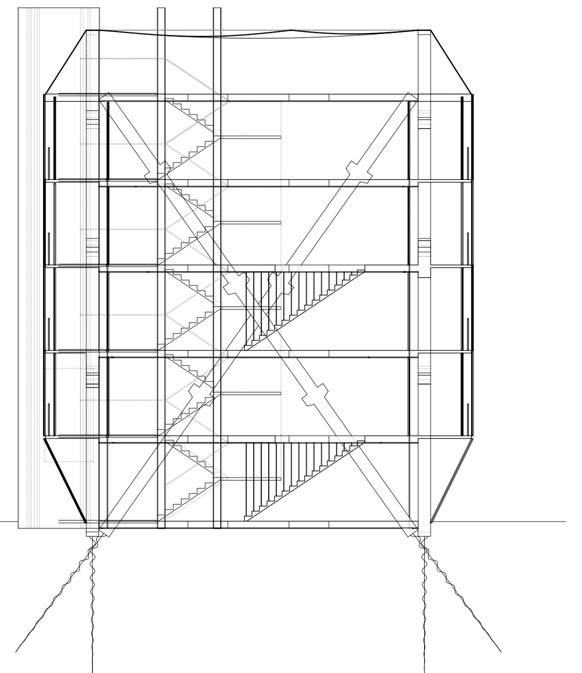


By harnessing tensegrity and thus slanting the columns, we can even achieve specific ceiling heights with reused columns.

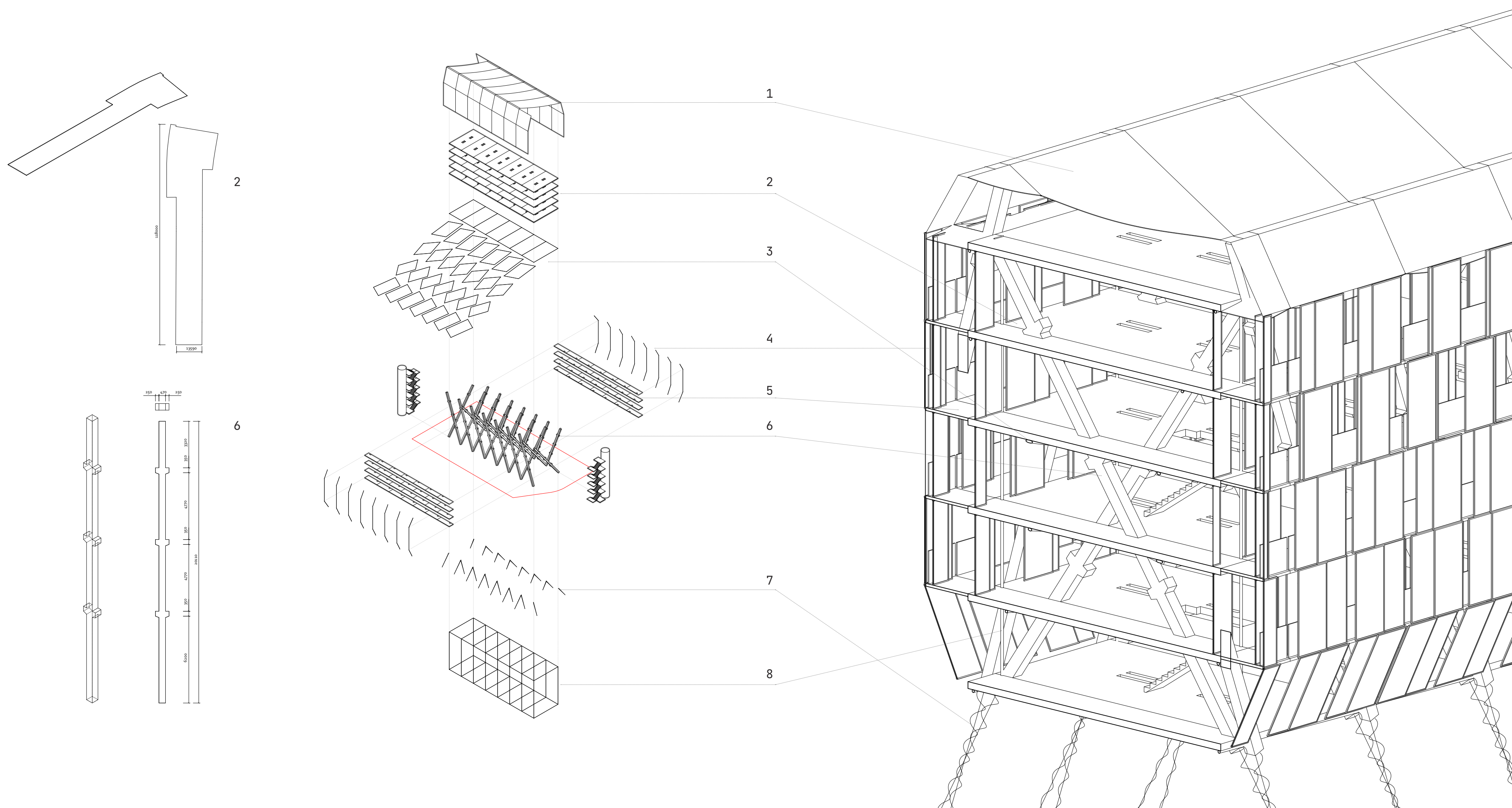








The slanted columns create exciting housing typologies throughout the building.



1	Facade	Hemp, ETFE, Mesh			
2	Floorslabs	In situ concrete	C-85-AWB	cut to 42 pieces of 12 * 6 * 0.25 m	+ 158,056.9 kgCO2-eq
3	Cables	Steel cables			- 4,600.91 kgCO2-eq
4	Cables	Steel cables			- 1,329.25 kgCO2-eq
5	Balconies	Garage walls		56 of 56 pieces	+ 15,393.08 kgCO2-eq
6	Columns	Reinforced concrete	C-001-PLB	28 of 54 pieces	+ 96,074.44 kgCO2-eq
7	Foundations	Steel geoscrews		28 pieces	- 6,994.17 kgCO2-eq
8	Cables	Steel cables			- 4,144.6 kgCO2-eq
				Total CO2	+ 252,366.4 kgCO2-eq
				Total weight	2,278,485 kg
				Total floor surface (garages, roof excl.)	2016 sqm
				CO2 per sqm	+ 125.2 kg kgCO2-eq

2 — CO2 FOOTPRINT

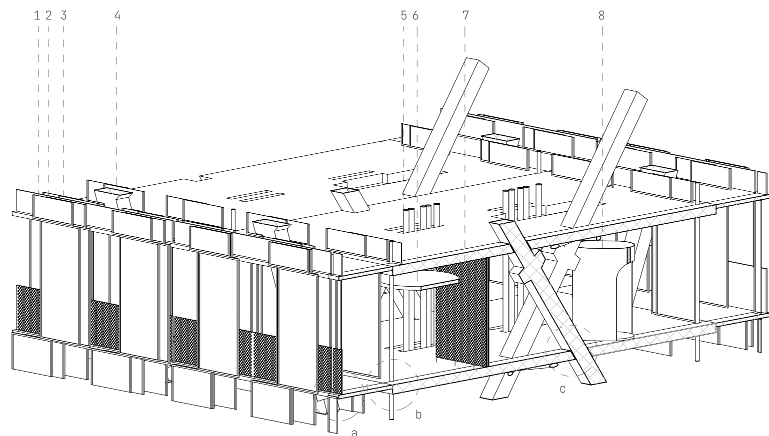
➡ **outset**

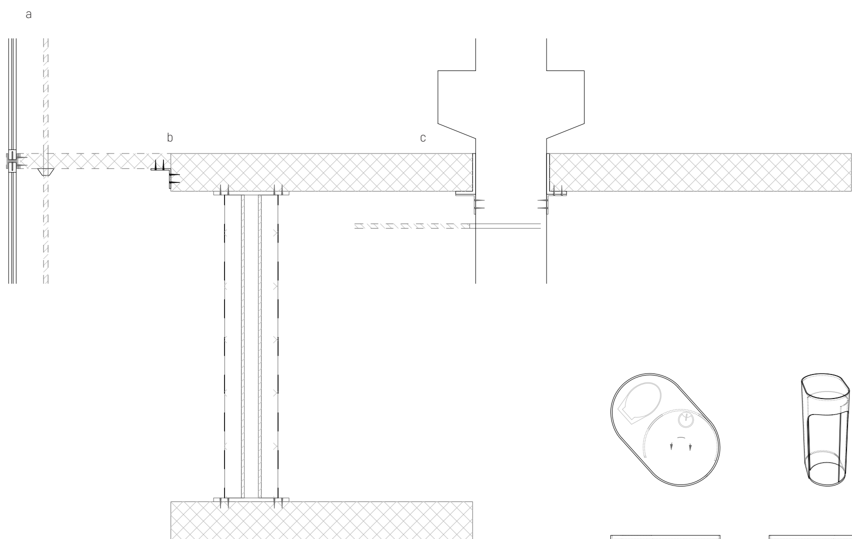
A catalogue of dissembled building components to be reused for housing.

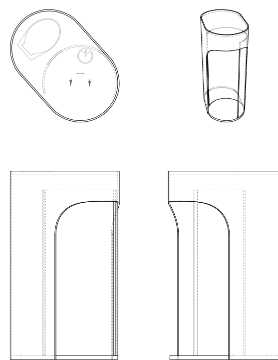
➡ **PROBLEM**

Calculate and optimize the CO2 footprint of the design.

1 Three layers of ETFE stretched over a metallic frame
2 One railing of metallic mesh stretched over a metallic frame
3 One layer of Hemp twill stretched over a metallic frame
4 Single-pane glass doors, fully movable
5 Hot and cold air, hot and cold water, wastewater pipes. Principle of a convective pump using the building skin
6 Acoustic felt ceiling
7 Acoustic walls made of one sheet of metallic mesh stretched between the floors, fiber insulation, hempclay board
8 Prefab wetroom made of corrugated metal, insulation and ceramics

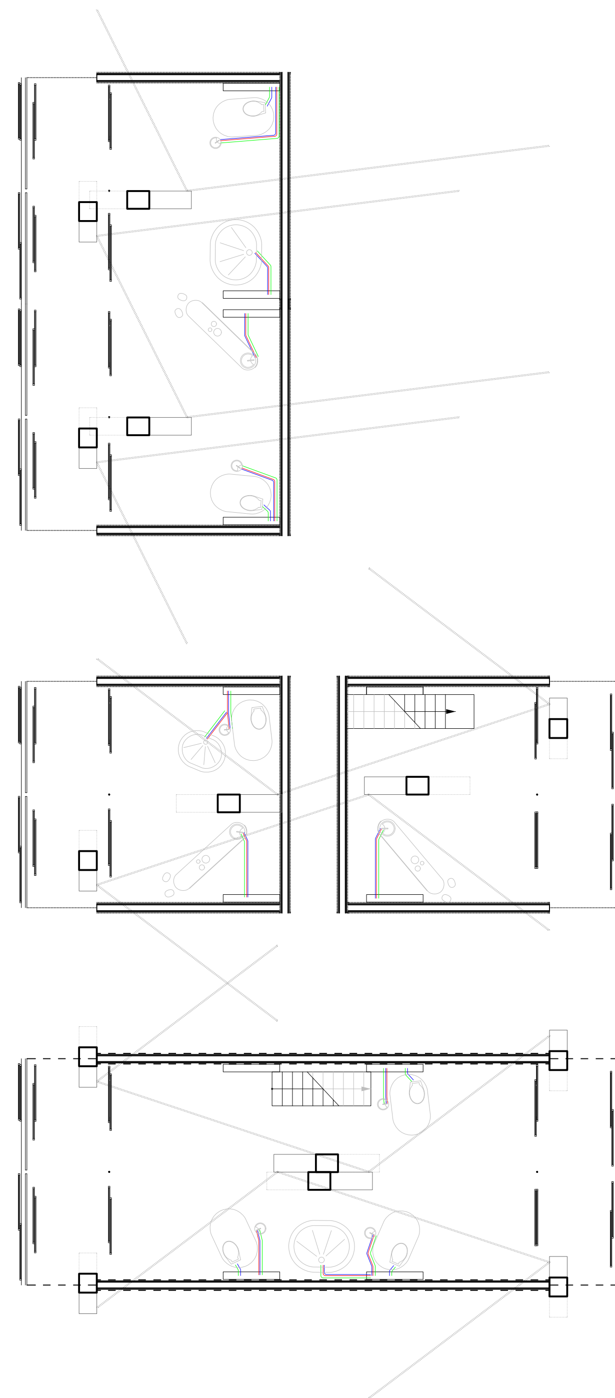


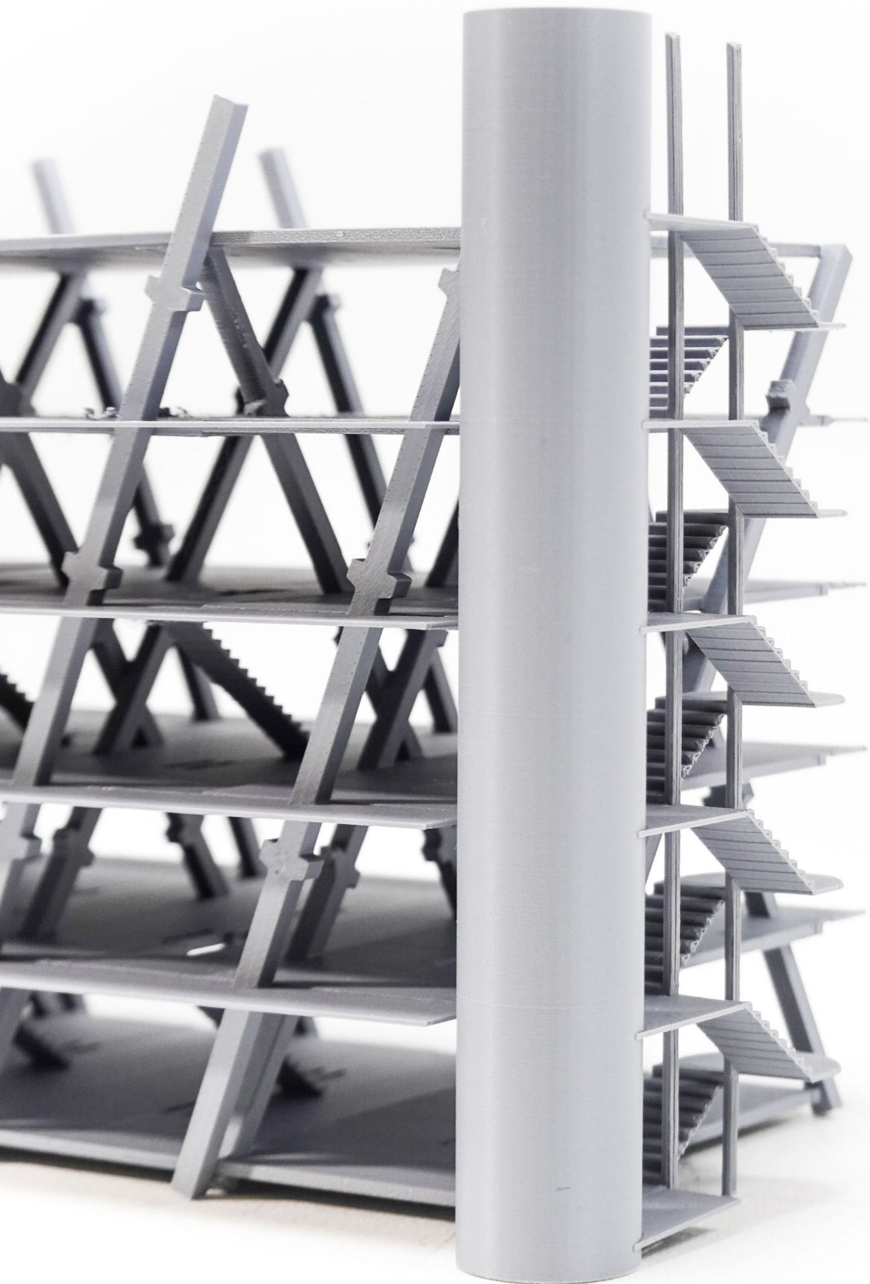




The wet room has a rotary door with a sink that optimizes space depending on the use.

8

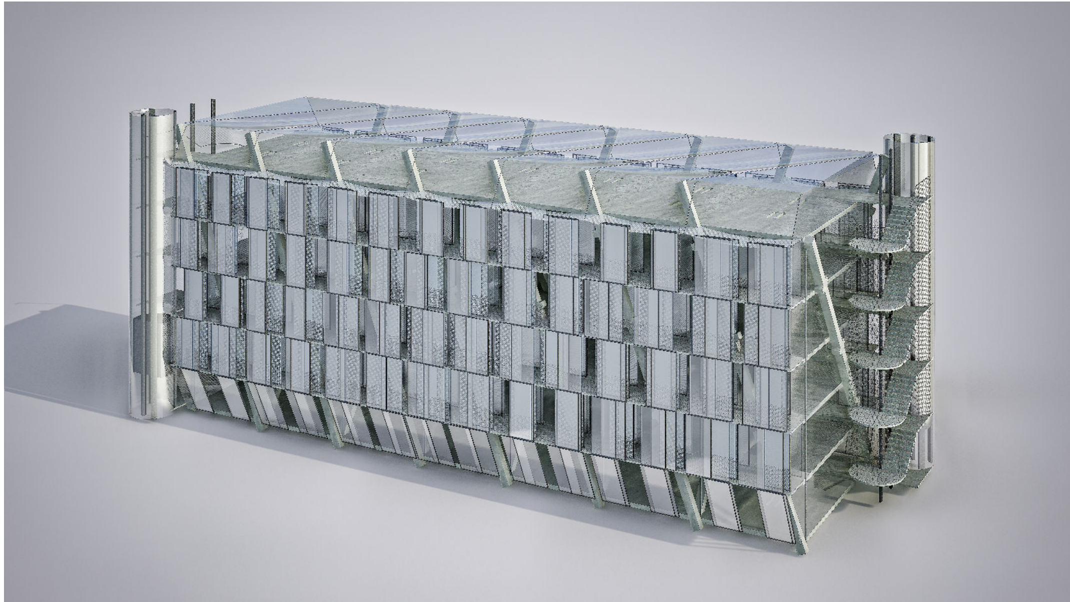




3 — DESIGN COHERENCE

outset
A catalogue of dissembled building components to be reused for housing.

PROBLEM
Establish a coherent design language around tension throughout all levels and scales of the project.



© Dominic Weber 2026