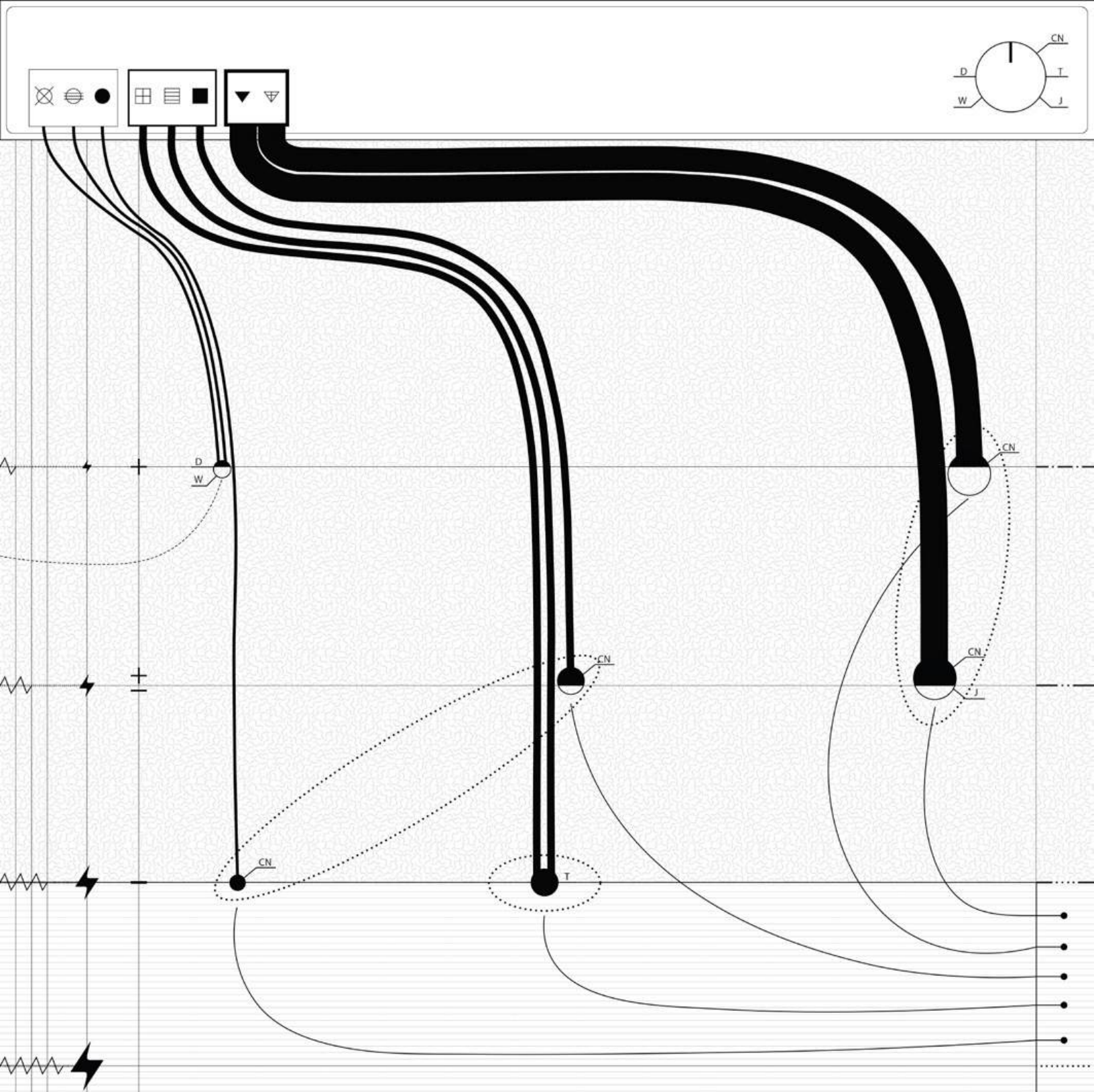




Virginia Italia

Portfolio



On research, interrogation, mediation, temporality, movement, documentation, territory, systems, nature, structure,

and **the specificity of conditions.**

May 2026
by Virginia Italia
Columbia GSAPP
Master of Architecture 2026

A sincere thank you to my professors, my friends, and especially my family.

Studio Projects			
01	The “Guilt-Free” Luxury Resort	ADV VI Studio <i>No Vacancy</i>	04-21
02	Straw Housing	Core III Studio <i>Housing</i>	22-47
03	A Sport or a Spectacle?	Core II Studio <i>Damage Control</i>	48-77
04	The Lichen Archive	ADV V Studio <i>Melting Archives</i>	78-95
05	A Deterritorialized Sanctuary	ADV IV Studio <i>Sanctuaries Territories</i>	96-107
06	The Blighted Pier	Core I Studio <i>An—Other Public</i>	108-117
Other Projects			
07	Doubly Ruled	Tensile/Compression Surfaces	118-119
08	Stair Generator	Generative Design	120-121
09	The Balcony Is	GAP I	122-129

01 The 'Guilt-Free' Luxury Resort

Spring 2026
ADV VI Studio
Instructor: Mireia Luzárraga
Individual Work

Site: Temozon Sur, Yucatan, Mexico

In Yucatán, the presence of the hacienda hotel, exemplified by Hacienda Temozón Sur, is embedded within a system of **cultural tourism** that transforms histories of exploitation into consumable experiences. Emerging from the legacy of the **henequen economy**, it operates within a regional network of elite tourism tied to routes connecting nearby Mayan ruins, cenotes, and curated "authenticity", where heritage is aestheticized and selectively performed. Within this framework, value is extracted not only from land and labor but from the controlled representation of culture itself. Positioned against this condition, **milpa-claimed territories** redistribute local agency in Temozón Sur by rendering the domestic backyard as a site of governance. Through Indigenous Mayan practices, milpa cultivation, melipona beekeeping, and craft economies, boundaries are reworked from instruments of exclusion into productive interfaces, **shifting power** from exclusive ownership toward collective, land-based autonomy.



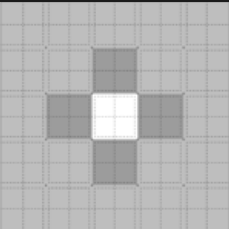


The **"hacienda"** as a tool for luxury tourism operates within a broader network of cultural tourism linked to the extraction of heritage, exploitation of labor, and erasure of territory that has impacted Indigenous communities in Yucatán, Mexico for centuries. Hacienda Temozón Sur functions through subtle forms of spatial and social isolation: controlled access, curated landscapes, and staged "authentic" experiences that provide connectivity and exclusivity for tourists while reinforcing dependency and exclusion for local communities. Extending across a regional network of boutique hotels south of Mérida, these systems perpetuate **historical models of dependency** in which local populations increasingly lose access to land, resources, and territorial agency.

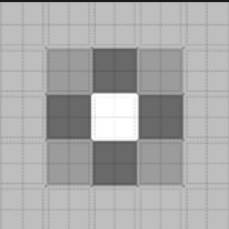


The Milpa Cycle

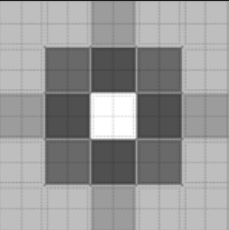
Year 01



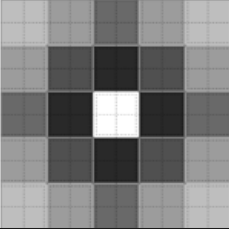
Year 02



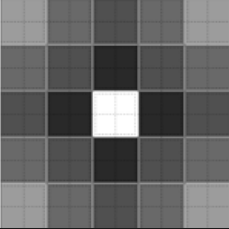
Year 03



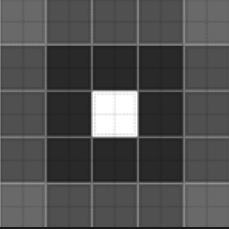
Year 04



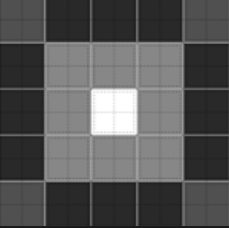
Year 05



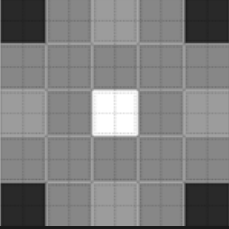
Year 06-08



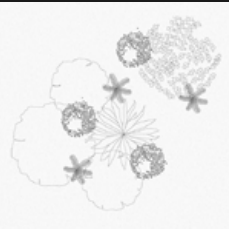
Year 10+



Year 20/ cycle
restars



Phase 00



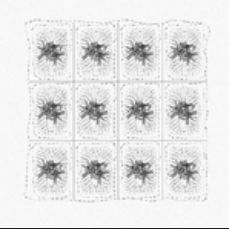
Phase 01



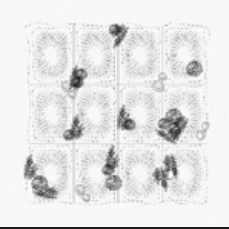
Phase 02



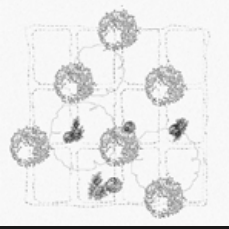
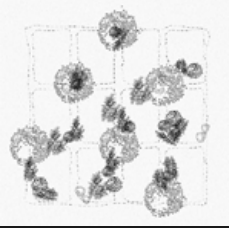
Phase 03



Phase 04



Phase 05



Dry forest

Slash and burn

Maize planting

Maize, Beans,
Squash harvest

Harvest

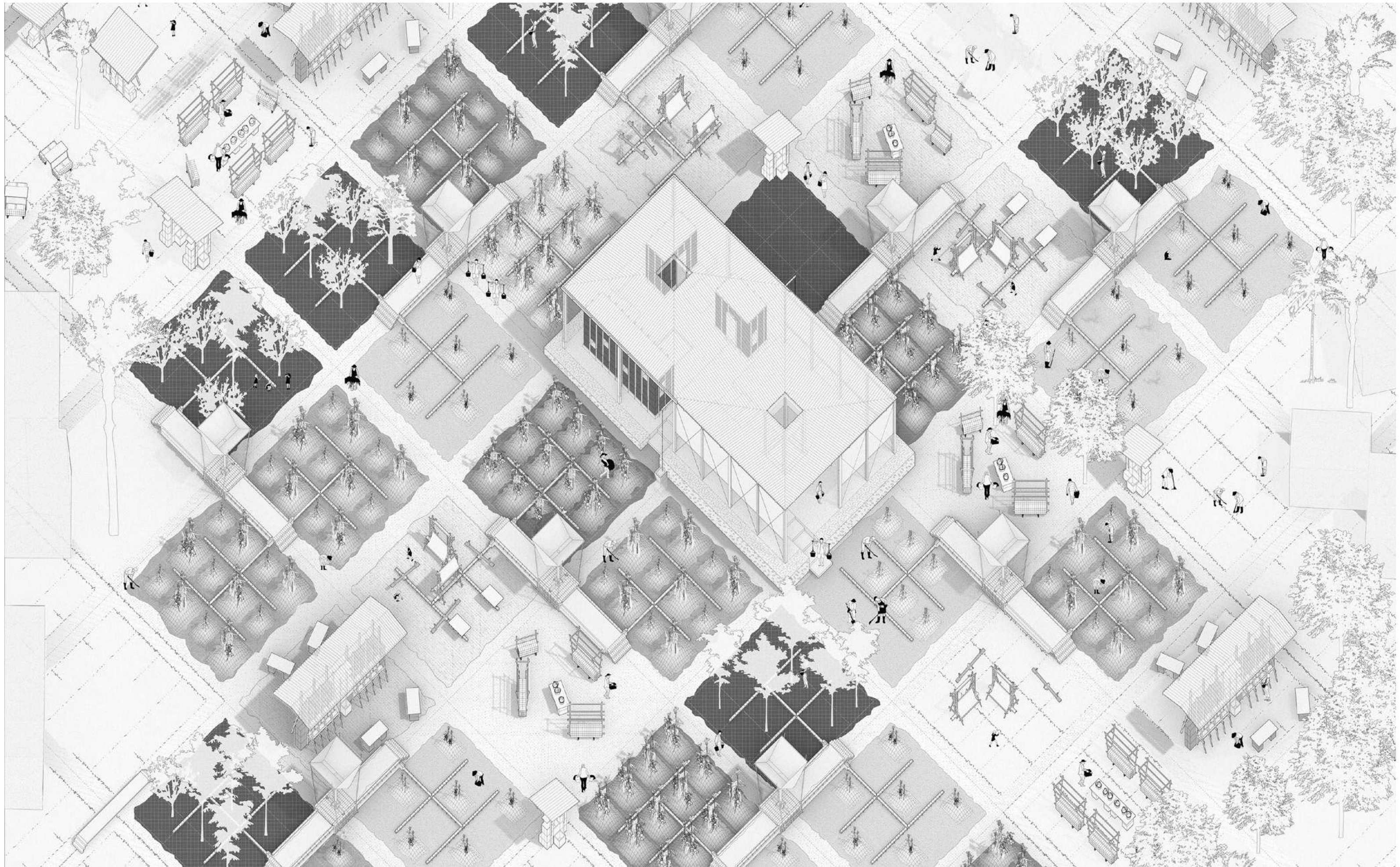
Weeds and fruit
tree planting

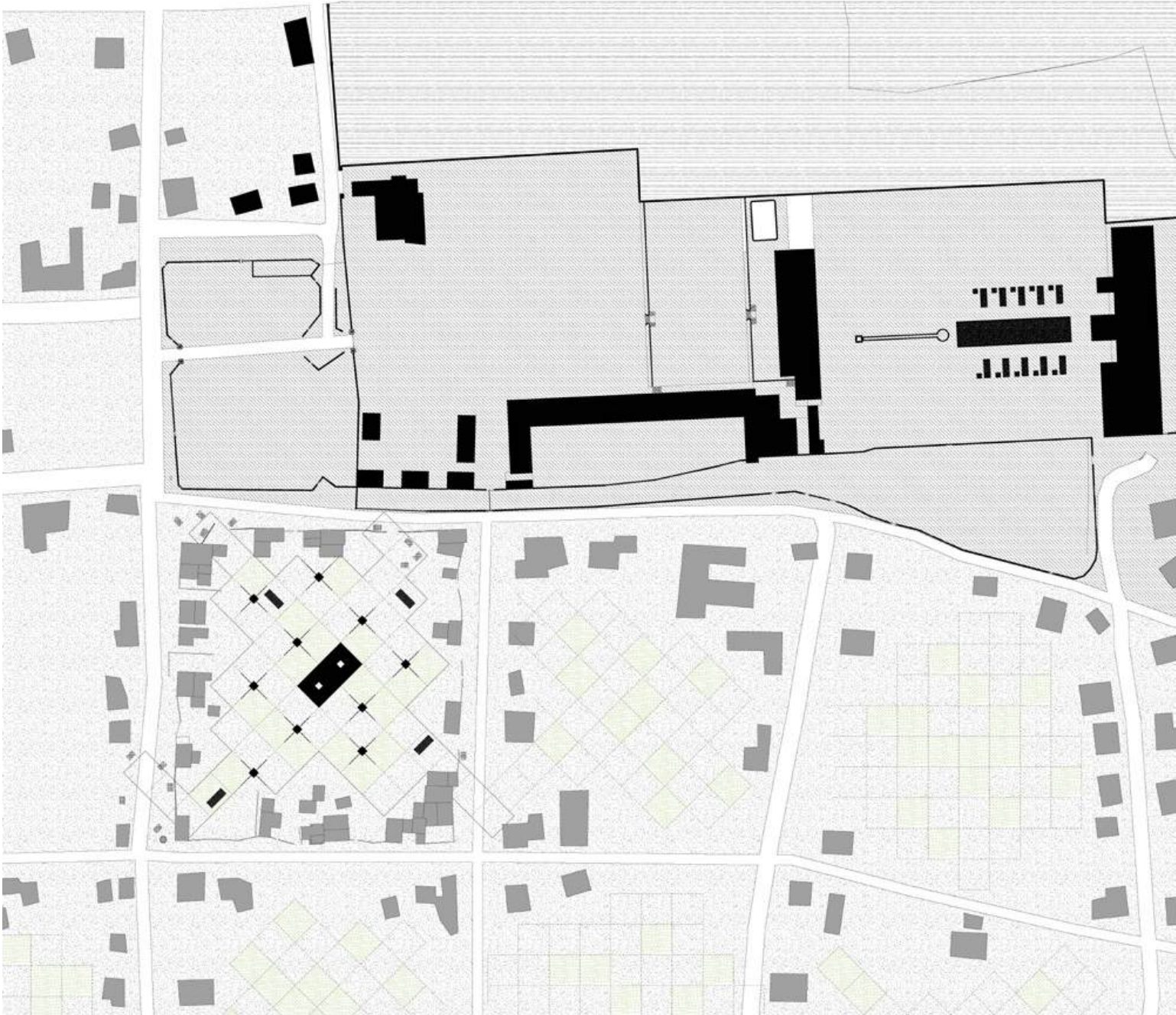
Fruit tree forest

Mature forest

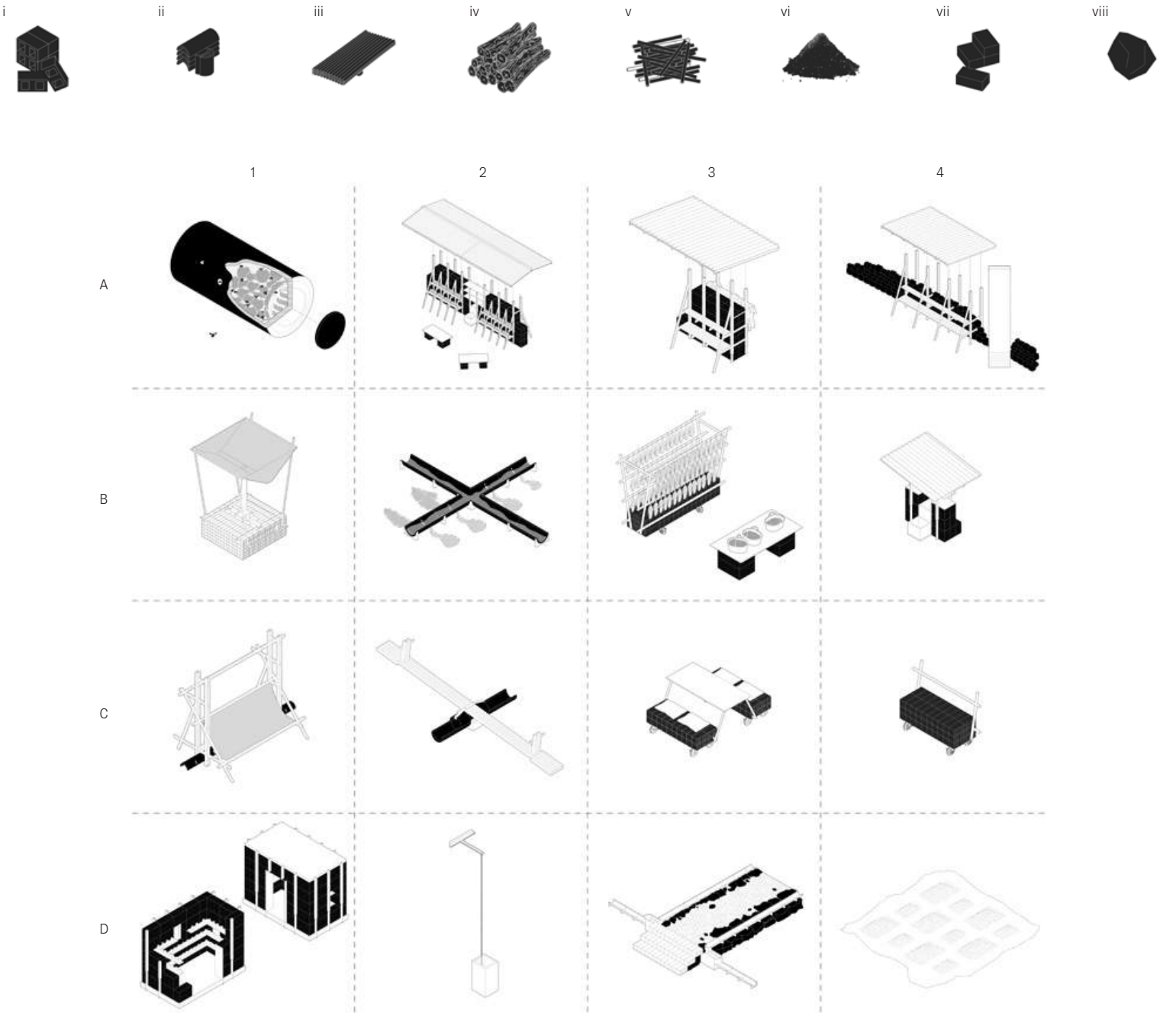


Plots and Harvests
In response, redistributing agency through the **knowledge systems** already embedded within the community. Practices such as **milpa farming** and **melipona beekeeping** construct localized networks of food production, education, water collection, and exchange. Milpa farming, a traditional Mayan polycultural system in which maize, beans, and squash grow interdependently, becomes both an agricultural and spatial framework organized through cycles of cultivation and regeneration. Alongside this, meliponarios support the cultivation of medicinal honey. Through rainwater **collection systems**, **seed banks**, **communal infrastructures**, and **market spaces**, the project transforms underutilized plots into productive landscapes that support local autonomy, collective learning, and long-term territorial stewardship.





Masterplan
Taking over the **'backyard'** of each block in Temozon Sur, the plots sit detached from the hacienda itself. The distribution of the elements on the site encourages produce, honey, and craft **exchange** between the neighborhoods while front-facing the hacienda to sell locals' products. Intended to expand, the introduction of milpa farms and learning spaces fosters **local agency and sovereignty**, all the while generating **microeconomies**.



Materials: found in 'backyards'

- | | |
|-----------------------|-----------------|
| i. cmu bricks | v. metal pipes |
| ii. clay tiles | vi. soil |
| iii. corrugated metal | vii. mud blocks |
| iv. wooden logs | viii. stones |

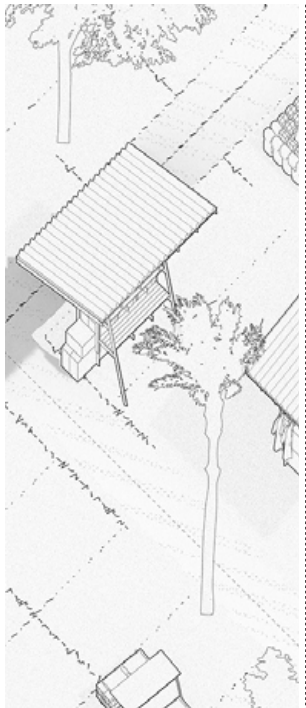
Elements

- | | | | |
|-----------------|-----------------------|-------------------|----------------|
| A1. jubones | B1. rainwater catcher | C1. lounge | D1. seed bank |
| A2. meliponario | B2. drip irrigation | C2. seesaw | D2. gutter |
| A3. market A | B3. corn drying rack | C3. movable table | D3. sacbe |
| A4. market B | B4. tool storage | C4. movable seat | D4. milpa plot |

Hacienda Temozon Sur
gate

Hacienda Temozon Sur
curb

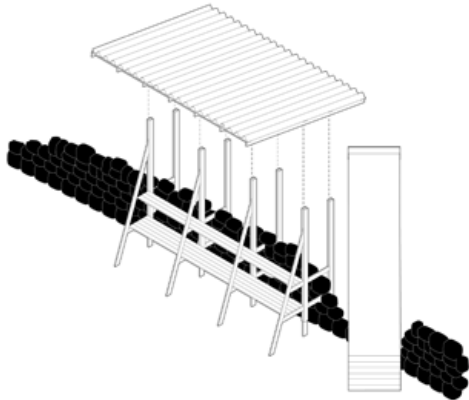
the 'Backyard'



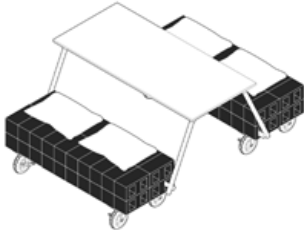
A3



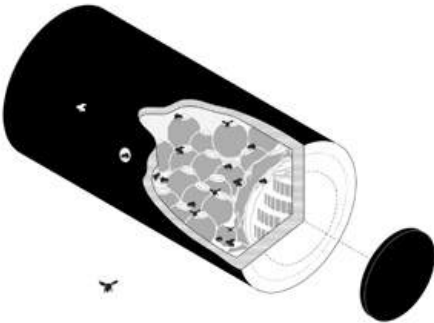
A4



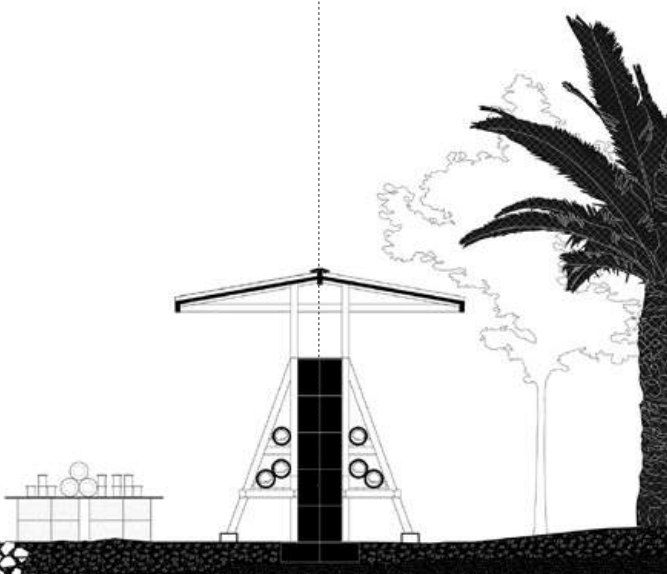
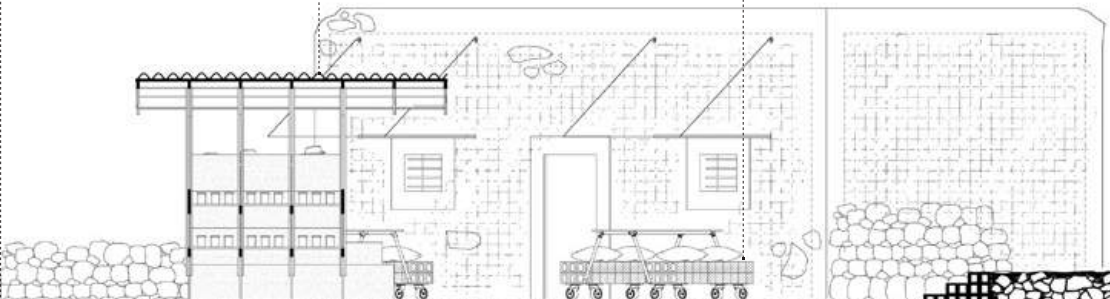
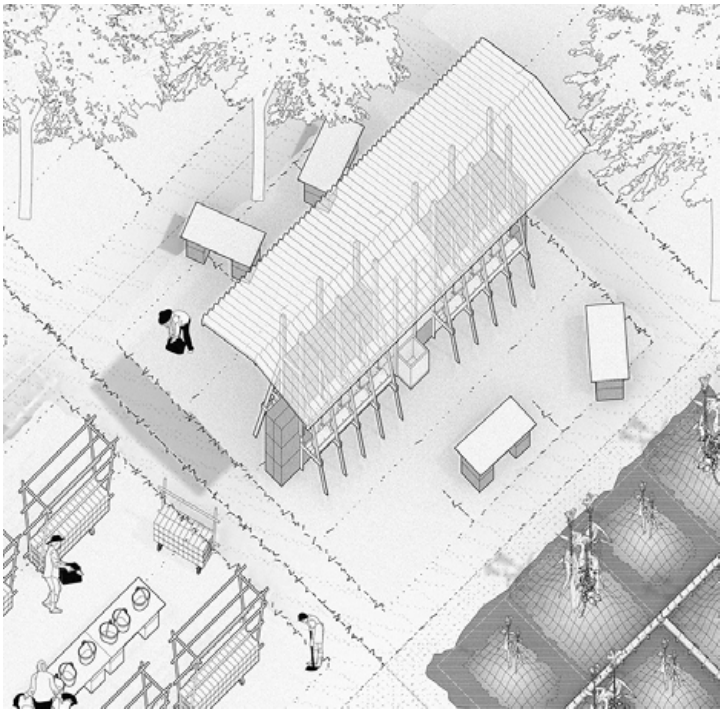
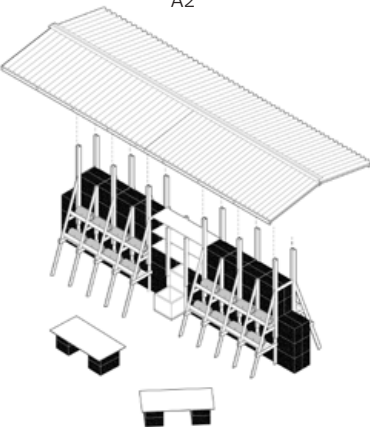
C3

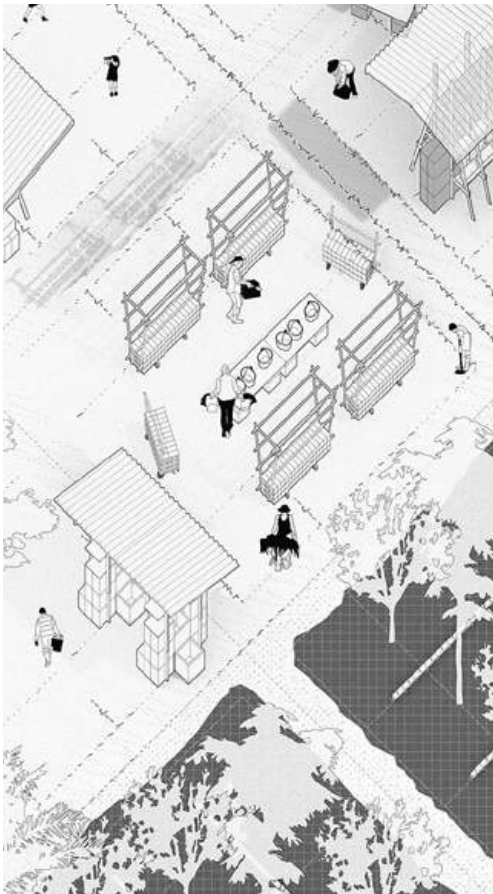


A1



A2

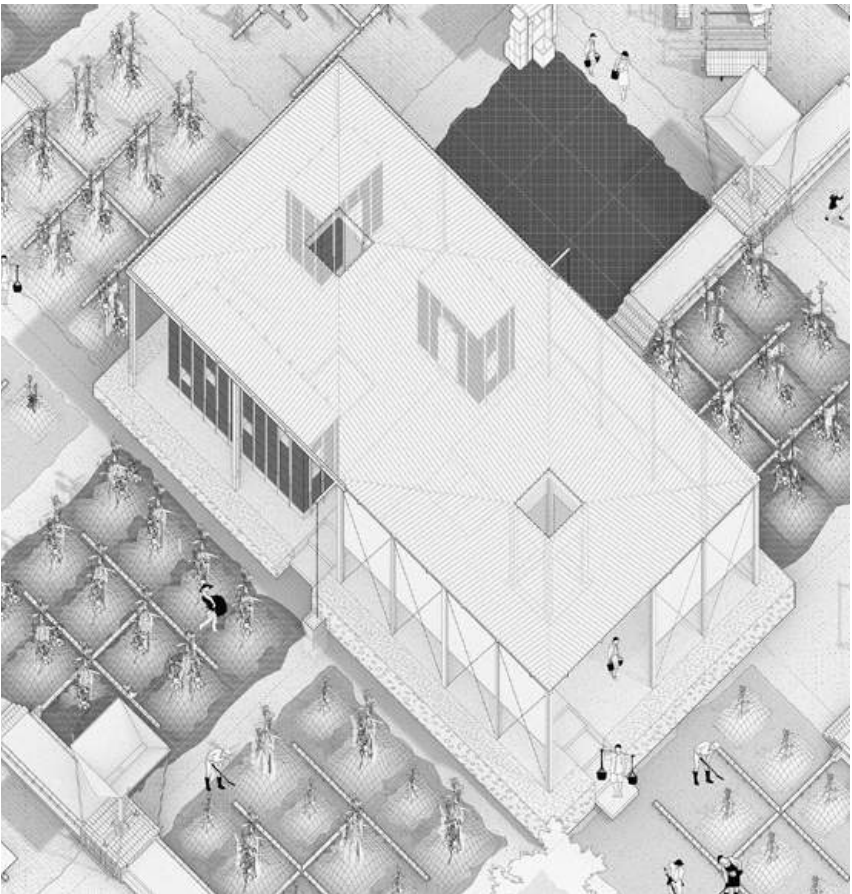
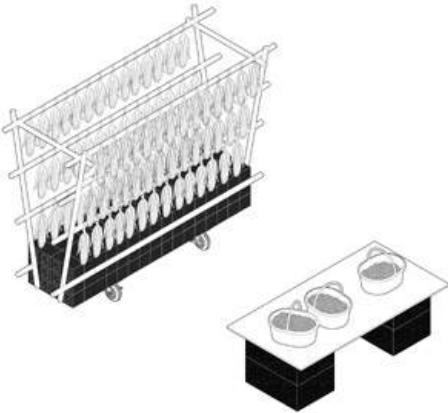




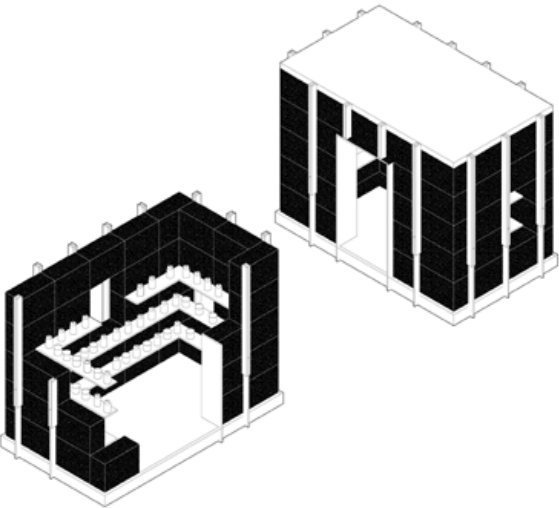
C4



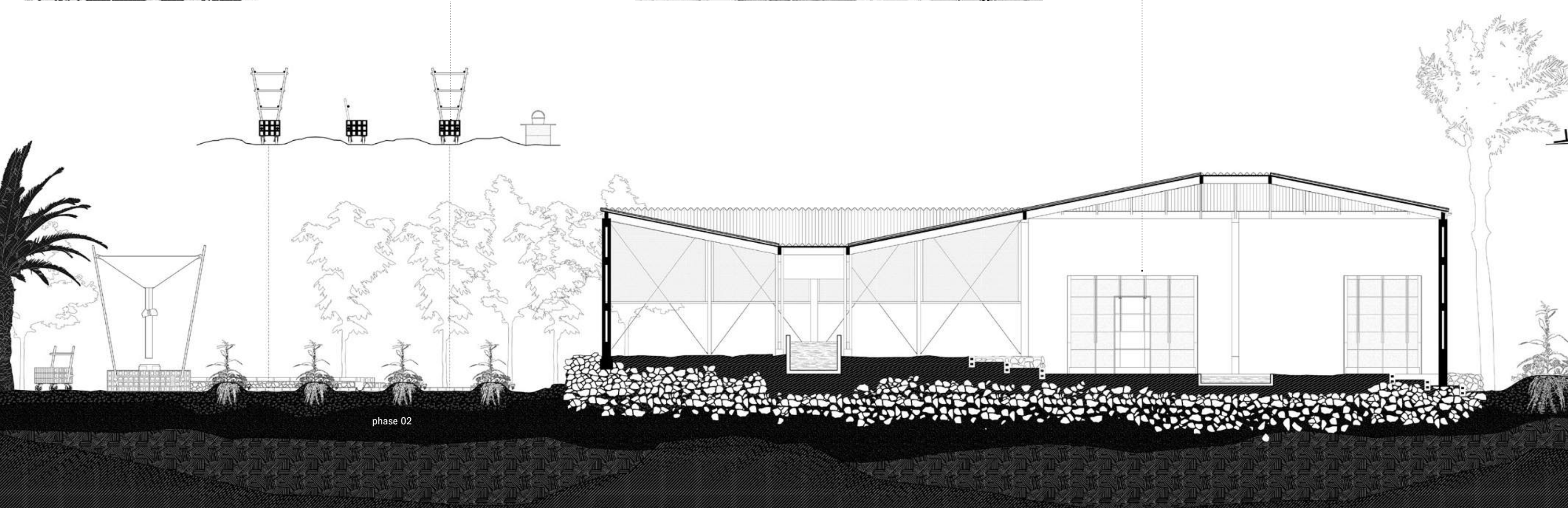
B3

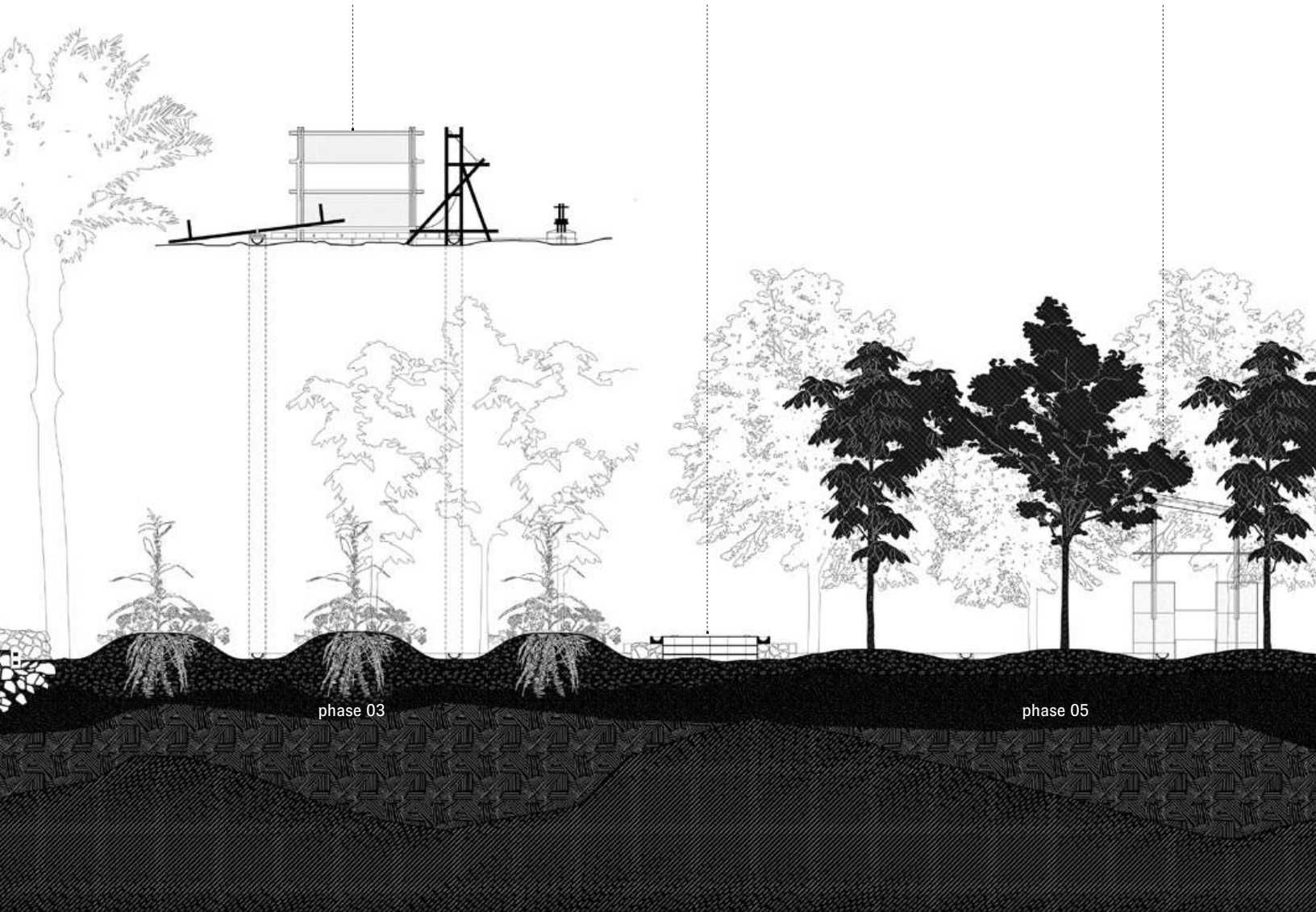
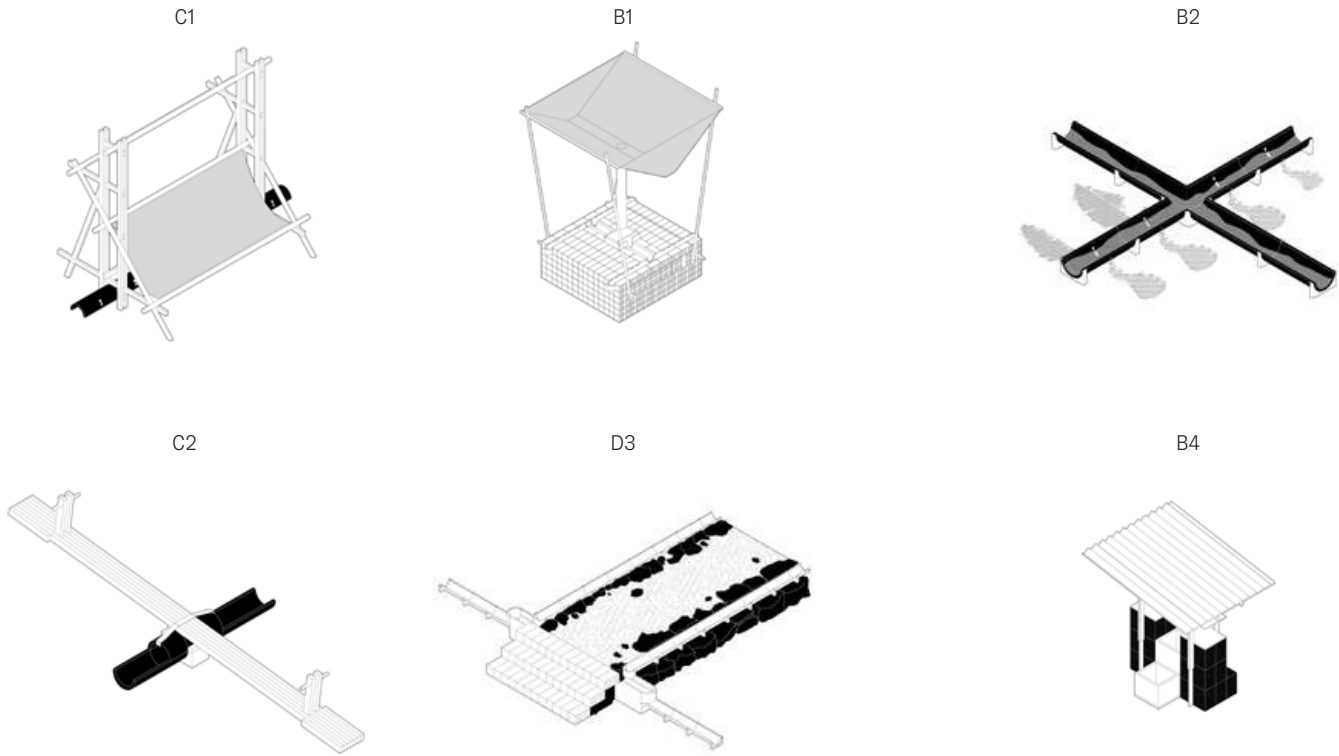


D1



phase 02







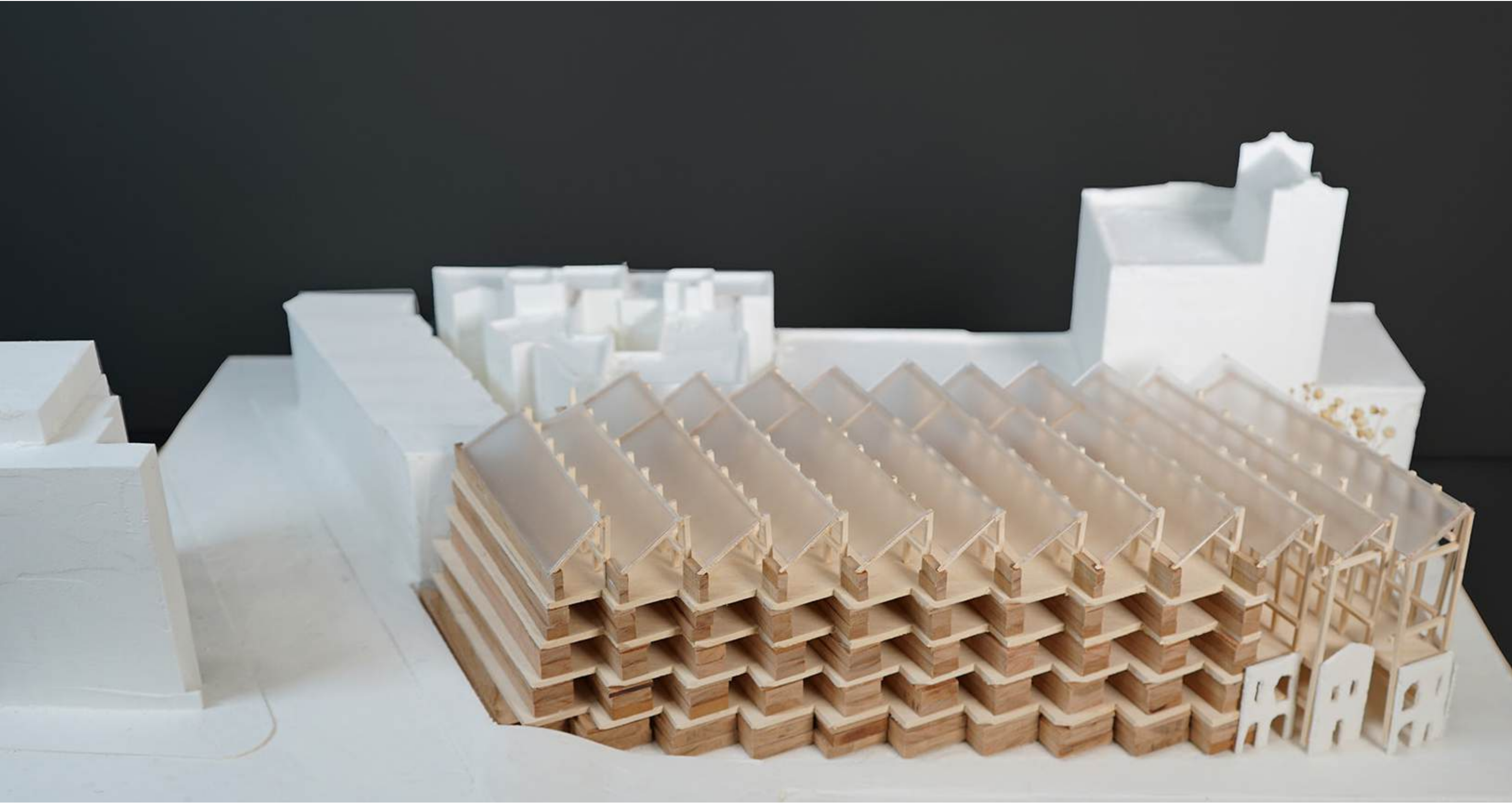
Together, the interventions form a landscape of recycled and locally sourced materials whose use and occupation shift alongside the cycles of cultivation and regeneration. Dependent on milpa rotations and collective maintenance, the project establishes an independent presence alongside the hacienda while reinforcing histories of agriculture, education, craft, and communal living that continue to be obscured by the image of the so called **"guilt-free" luxury resort**.

02 Straw Housing

Fall 2024
Core III Studio
Instructor: Marc Tsurumaki
Collaborative Work with Hayden St John

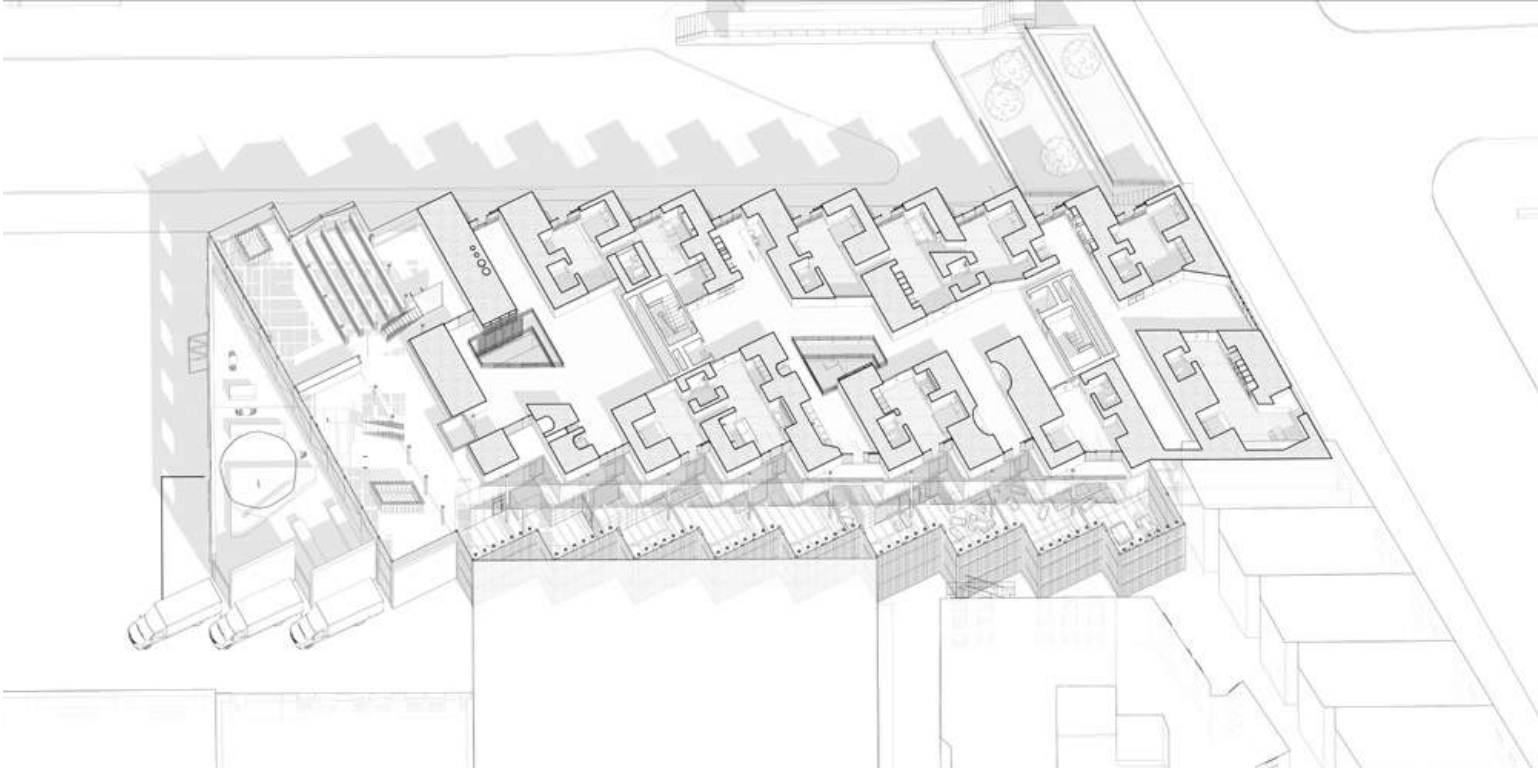
Site: Harlem, New York, NY, USA

This project explores the potential of **straw as a primary building material** to address sustainability and affordability in urban housing. By leveraging the thickness and density of straw bales, the design incorporates massive wall assemblies that taper as the building ascends, integrating passive solar strategies and rooftop growing spaces to enhance environmental performance. The ground floor features **community-focused programs** such as a daycare, educational center, food bank, and market, housed within **carved spaces** formed by the thick straw bale walls. Residences above transition from wall-embedded living areas to lighter layouts on higher floors, with shared kitchens and communal spaces fostering interaction and efficiency. **Visible systems** for water harvesting, composting, and waste recycling connect the building to broader agricultural networks, demonstrating how urban housing can embrace regenerative design and biogenic materials to promote a more sustainable future.





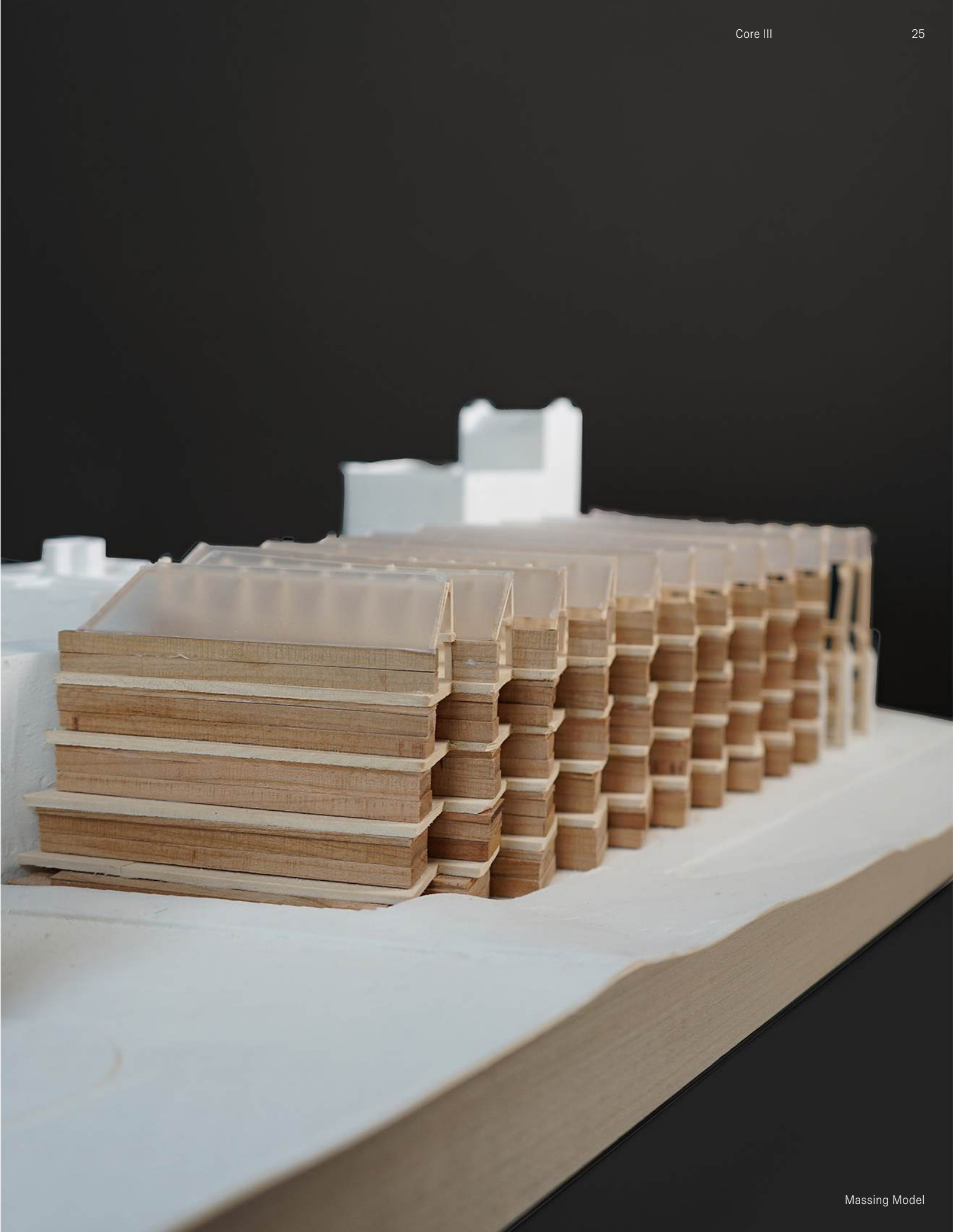
1



2

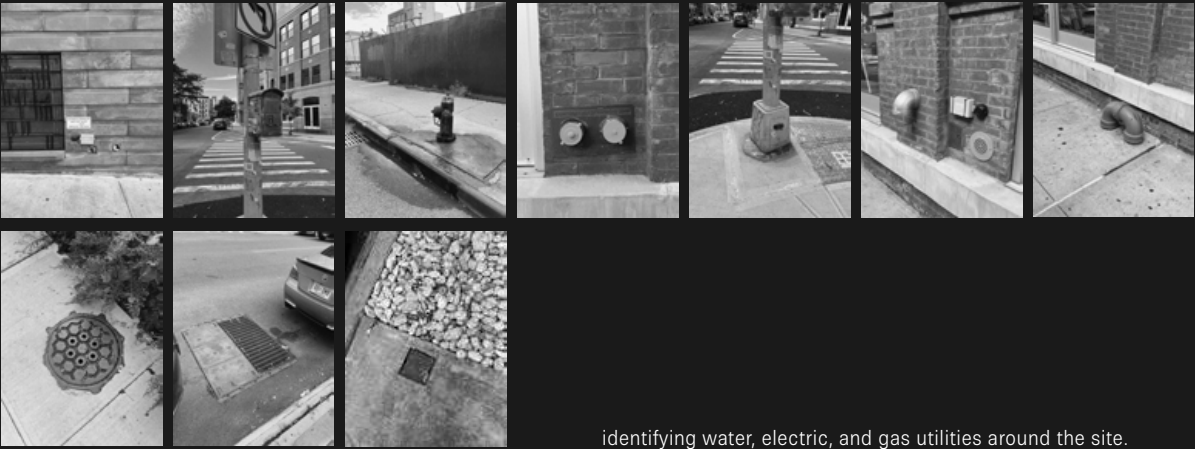
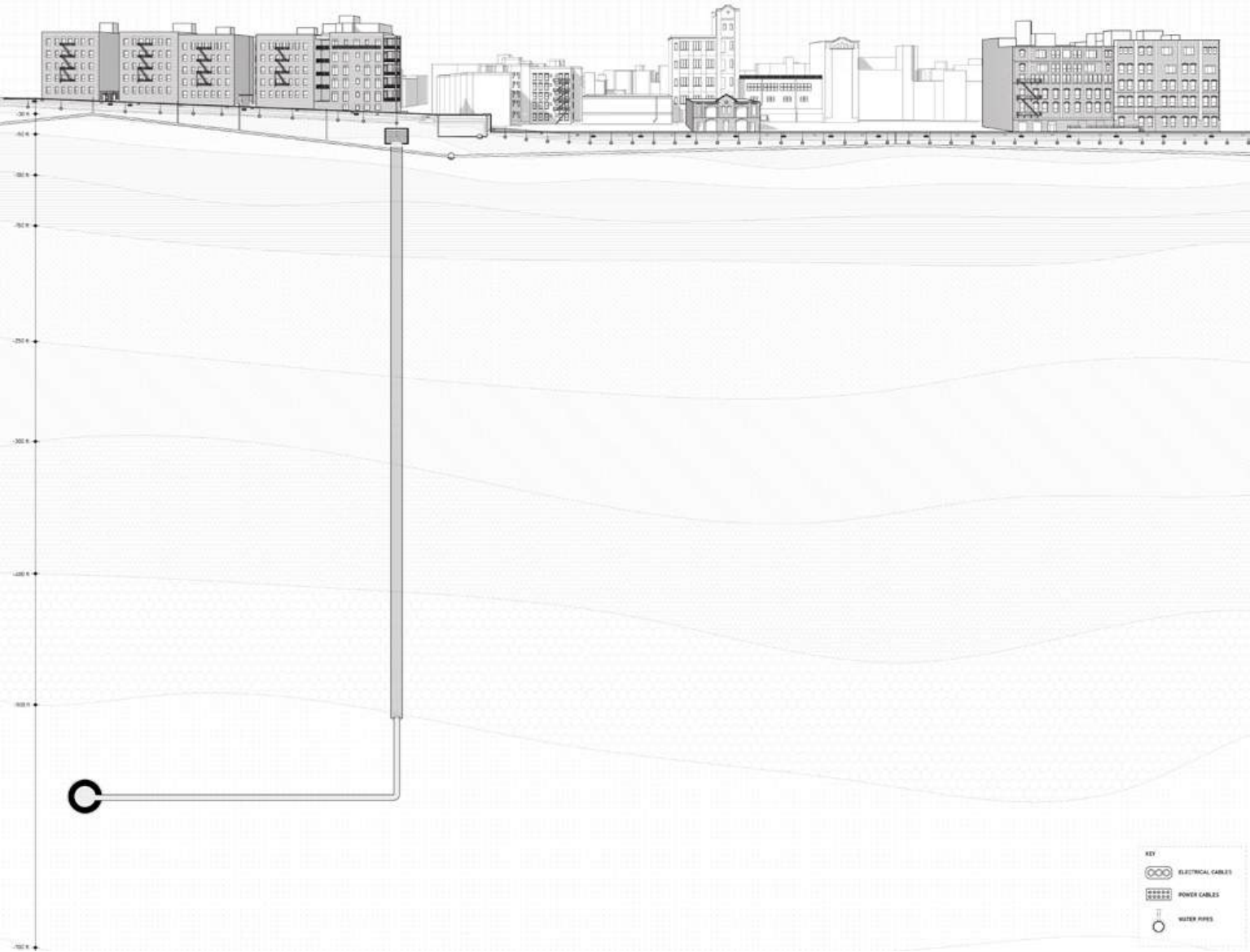
1 Third Floor Plan

2 Second Floor Plan



Massing Model

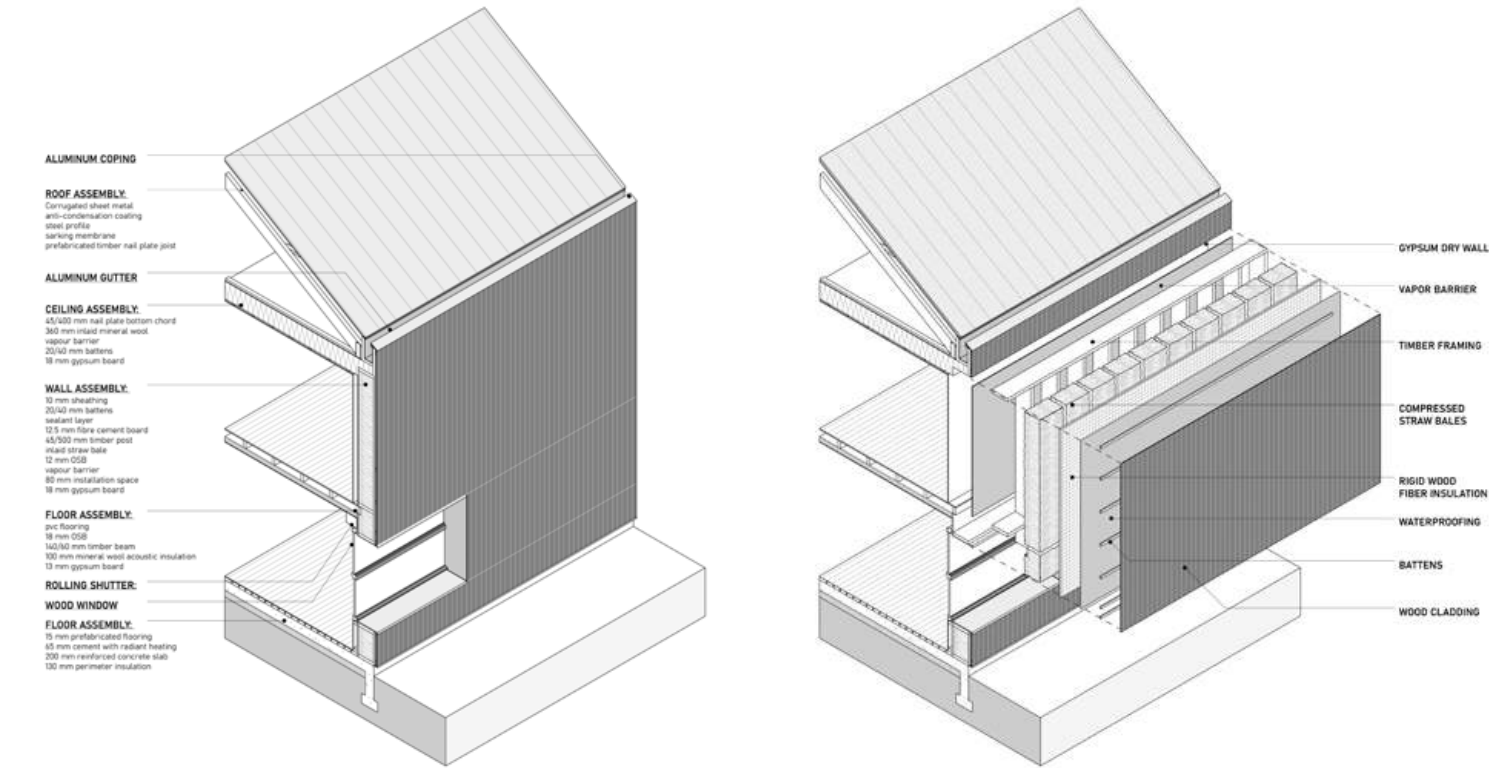
Site Section
Identifying the scalar relationship between house, building, site, and utilities.



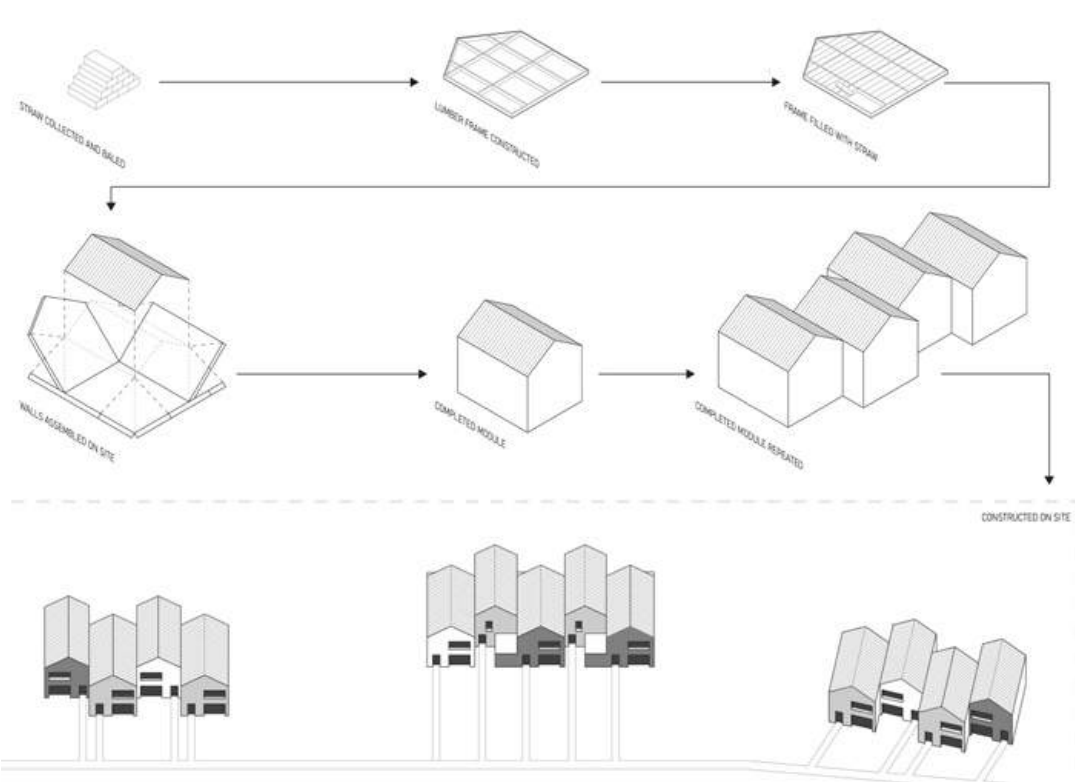
identifying water, electric, and gas utilities around the site.



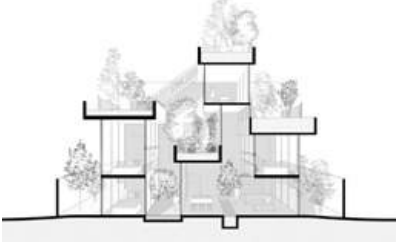
New York State's agricultural density, driven by its position as one of the most agriculturally productive states, generates **vast amounts of agricultural byproducts** like straw. With New York City consuming 19 billion pounds of food annually, straw—a common byproduct—offers **an untapped resource** for sustainable construction, sequestering carbon while reducing waste.



1

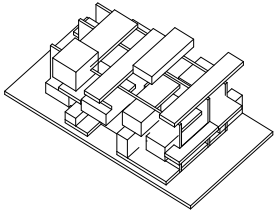
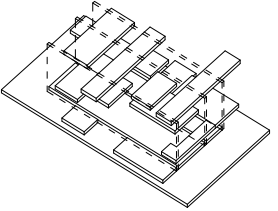
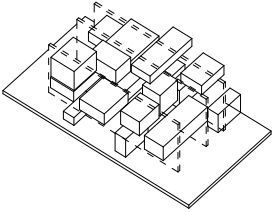
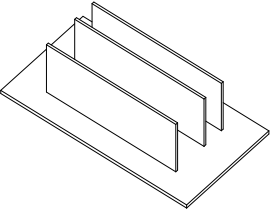
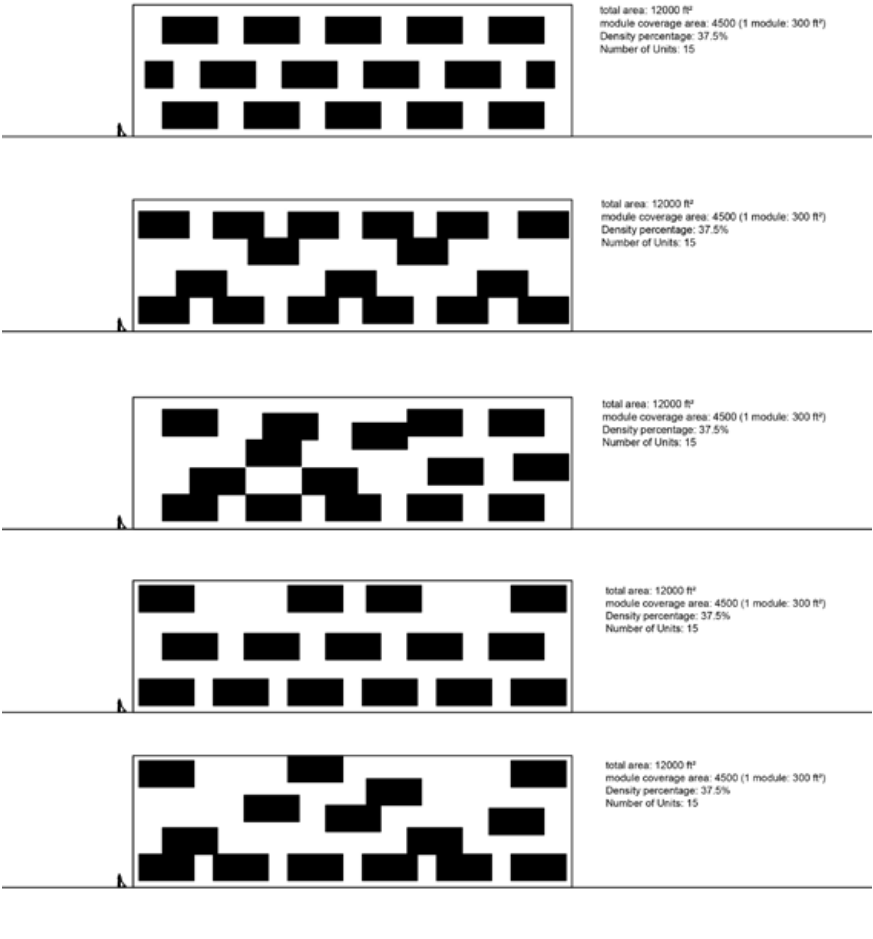
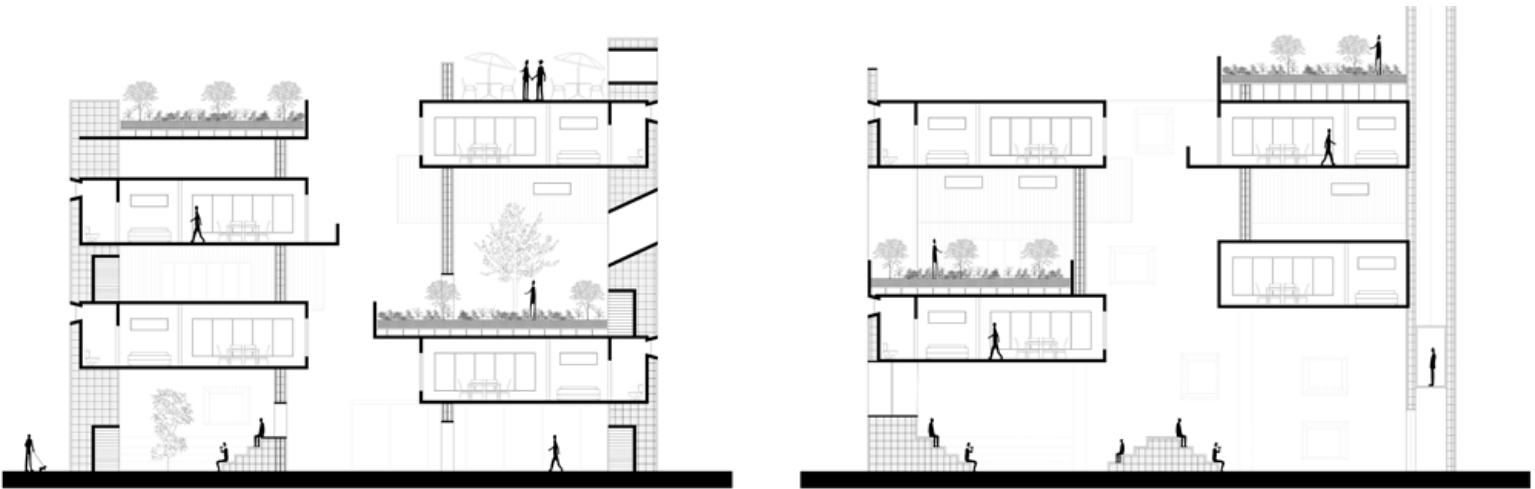


2



1 Wood and Straw Housing
NZI Architects

2 Binh House
VTN Architects



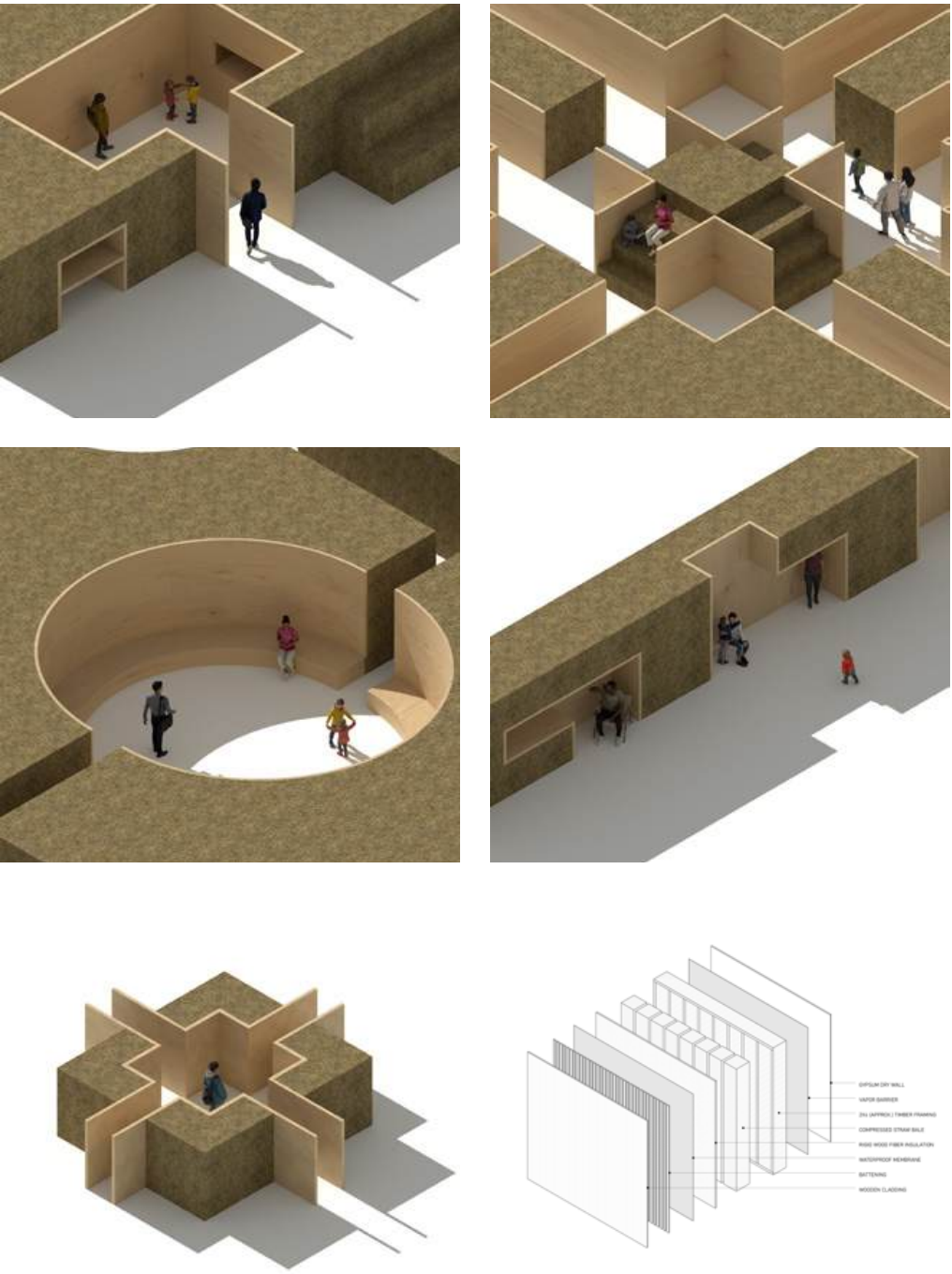
Massing and Concept: The massive straw bale wall



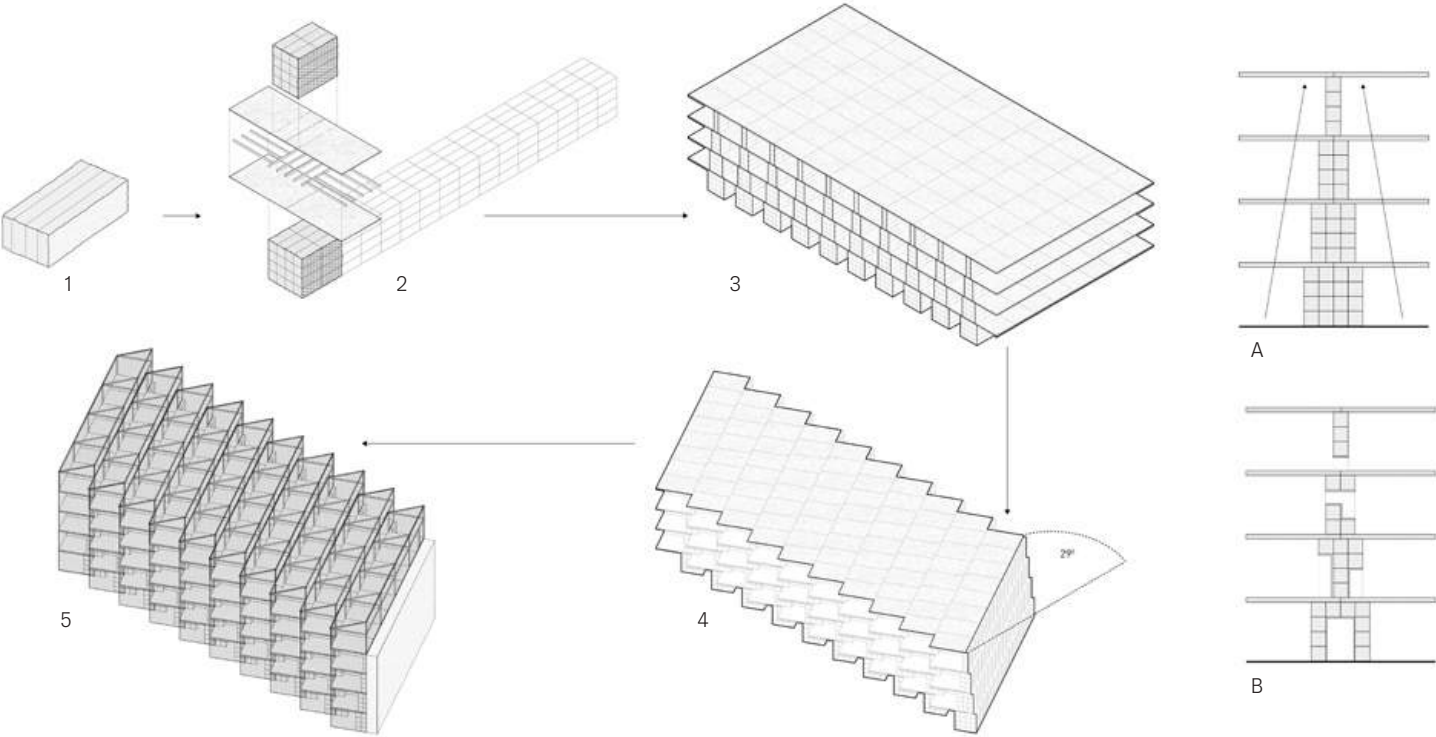
1 water pipe cassette

2 exposed jumbo straw bales

3 compressed straw bales CLT panels



Utilizing the straw bale's thickness—often seen as an inefficiency—as an opportunity to create space.



- 1

base structural unit:
3'x3'x3' jumbo bale.
- 2

bales assembled into wall:
prefab CLT placed between.
- 3

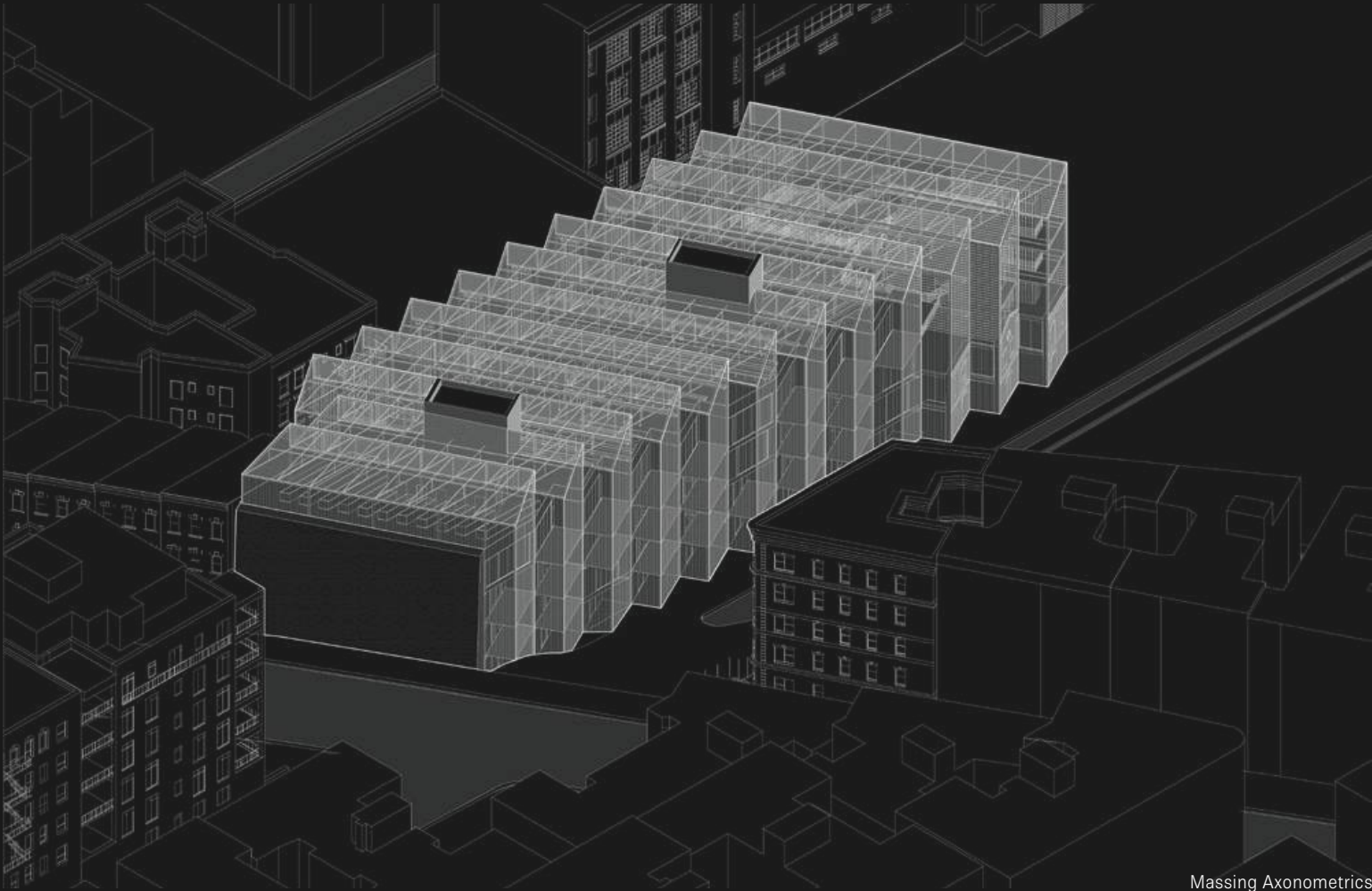
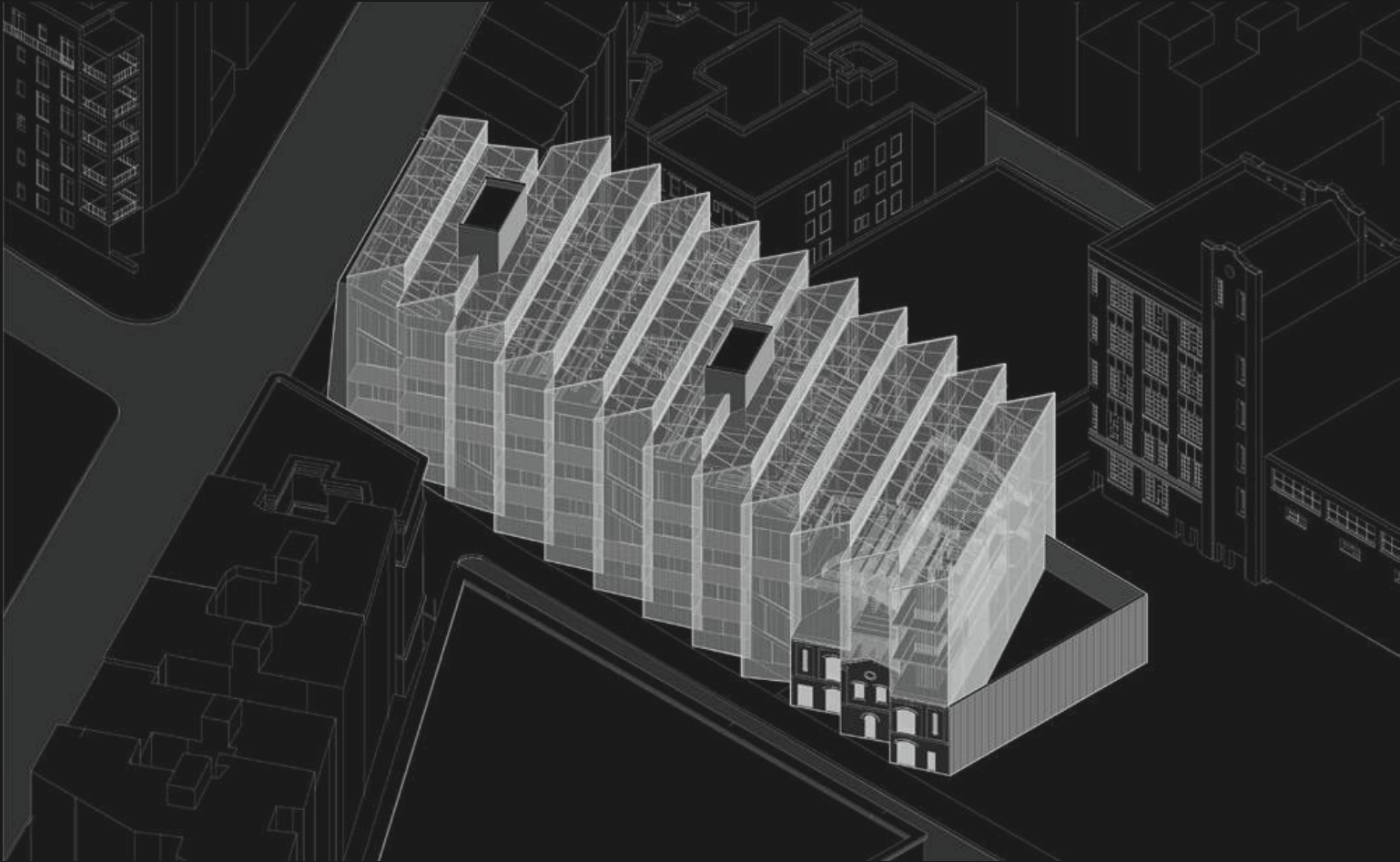
wall repeated 10x to fill
site.
- 4

modules rotated 29° to
orient to true north.
- 5

mass concealed in polycarb
shell to create varied climatic
conditions and optimize
agricultural production site.
- A

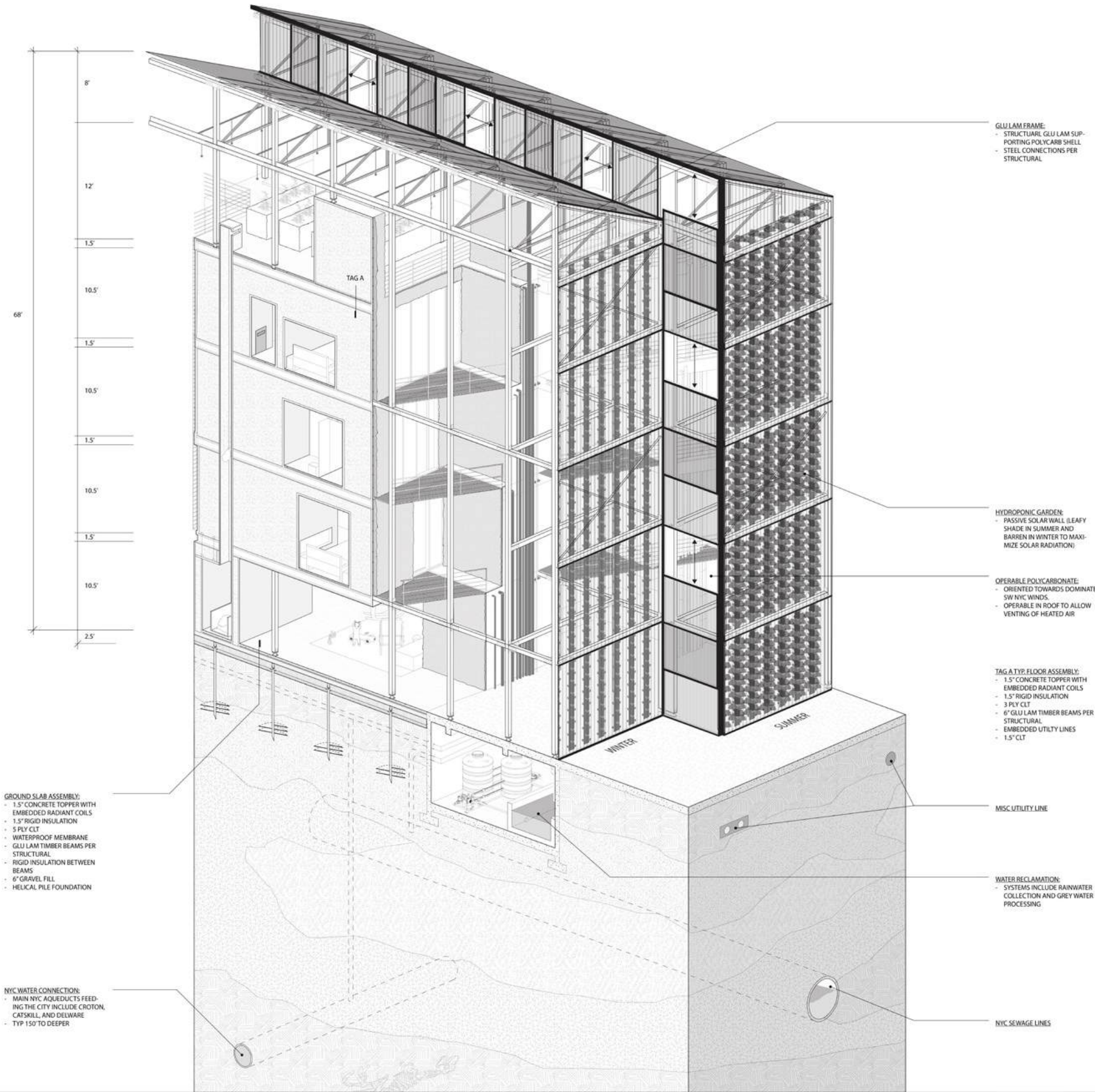
profile of wall tapers
to account for material
stability.
- B

CLT panels hold
excavations for
inhabitation of straw.

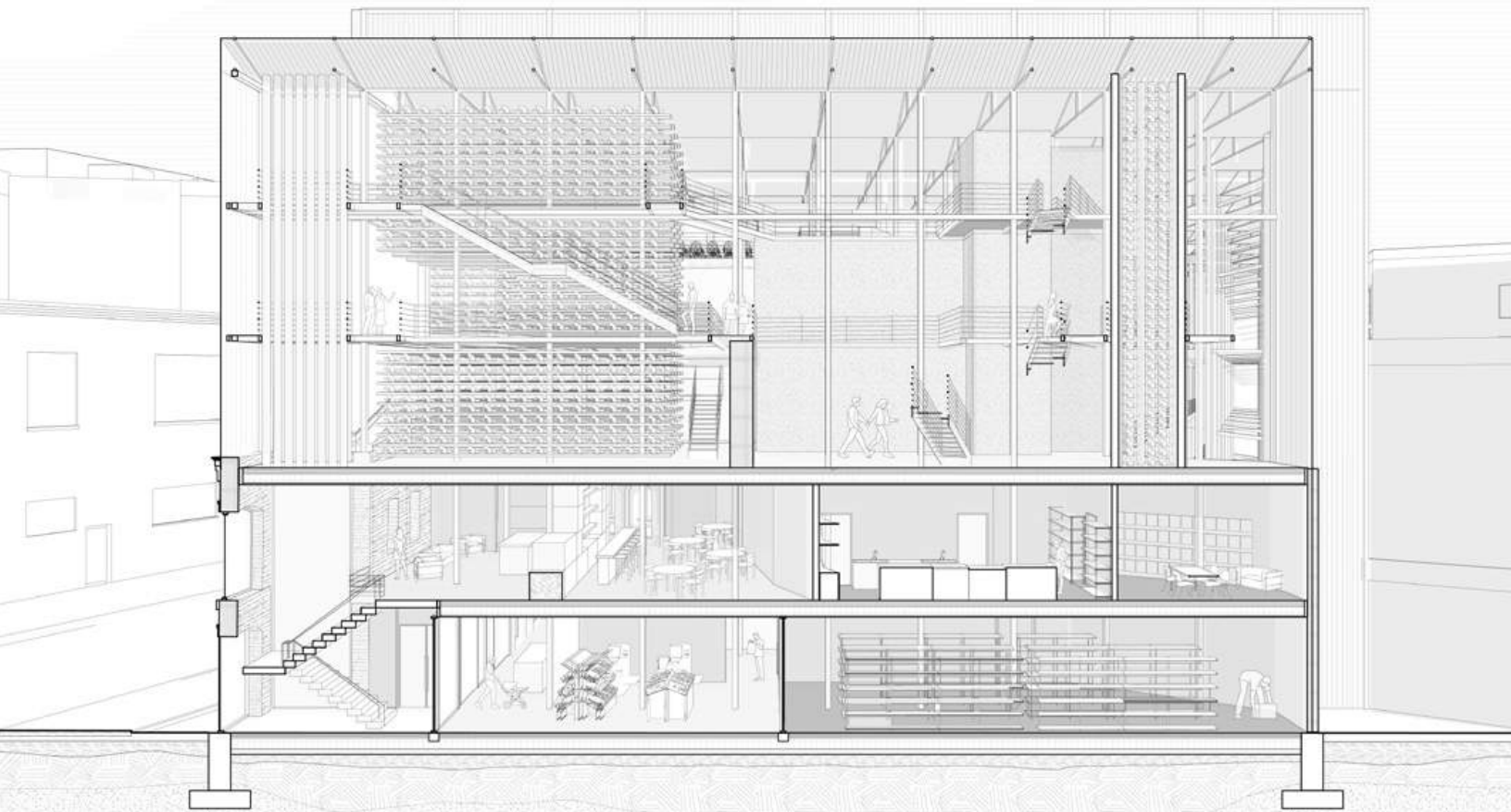




Detail Chunk Model



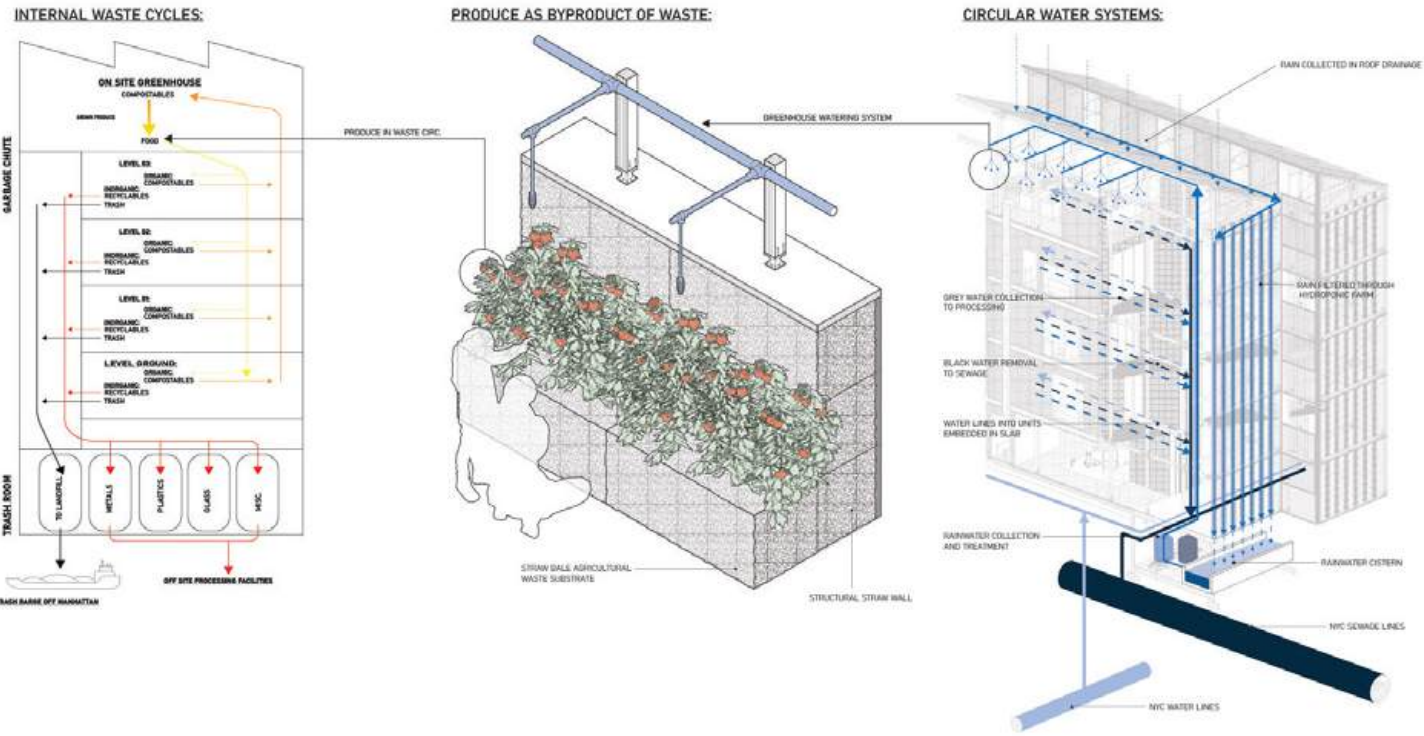




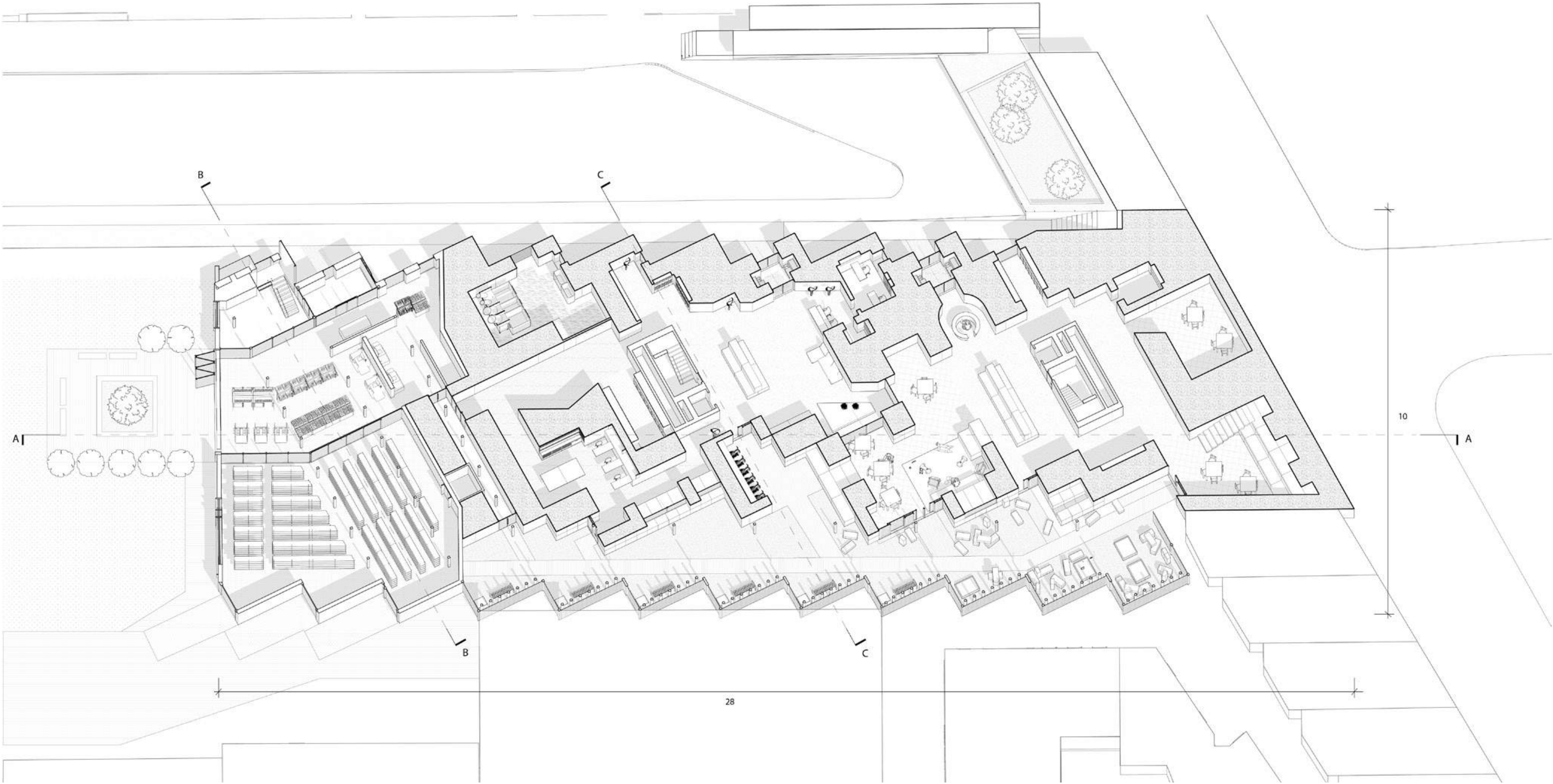
Section B-B



Roof Plan



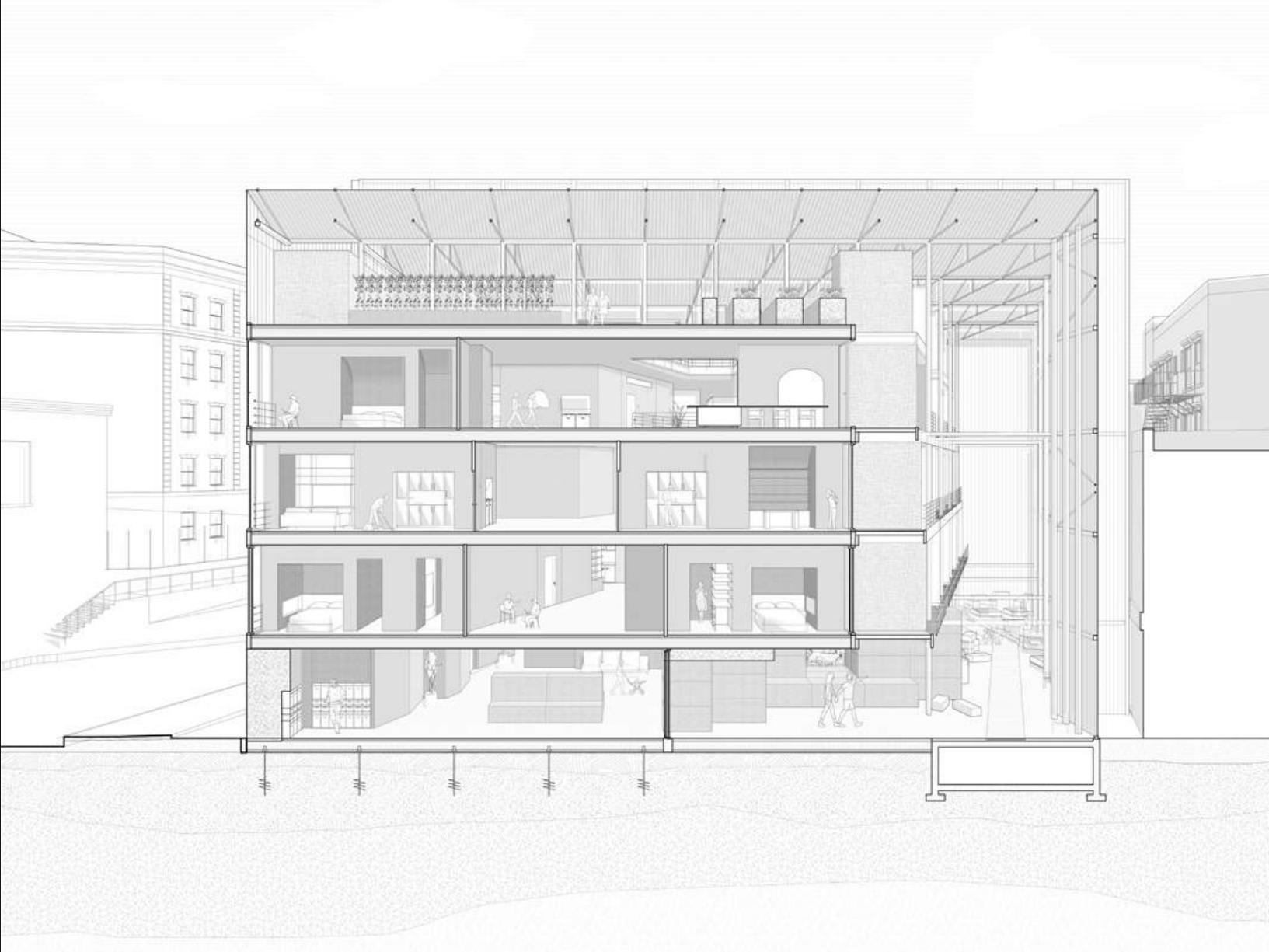
Utilities Cycles: Agricultural waste, rainwater, trash, recyclables, food, waste water.







The inhabitation of the massive, jumbo straw bale wall.



Section C-C

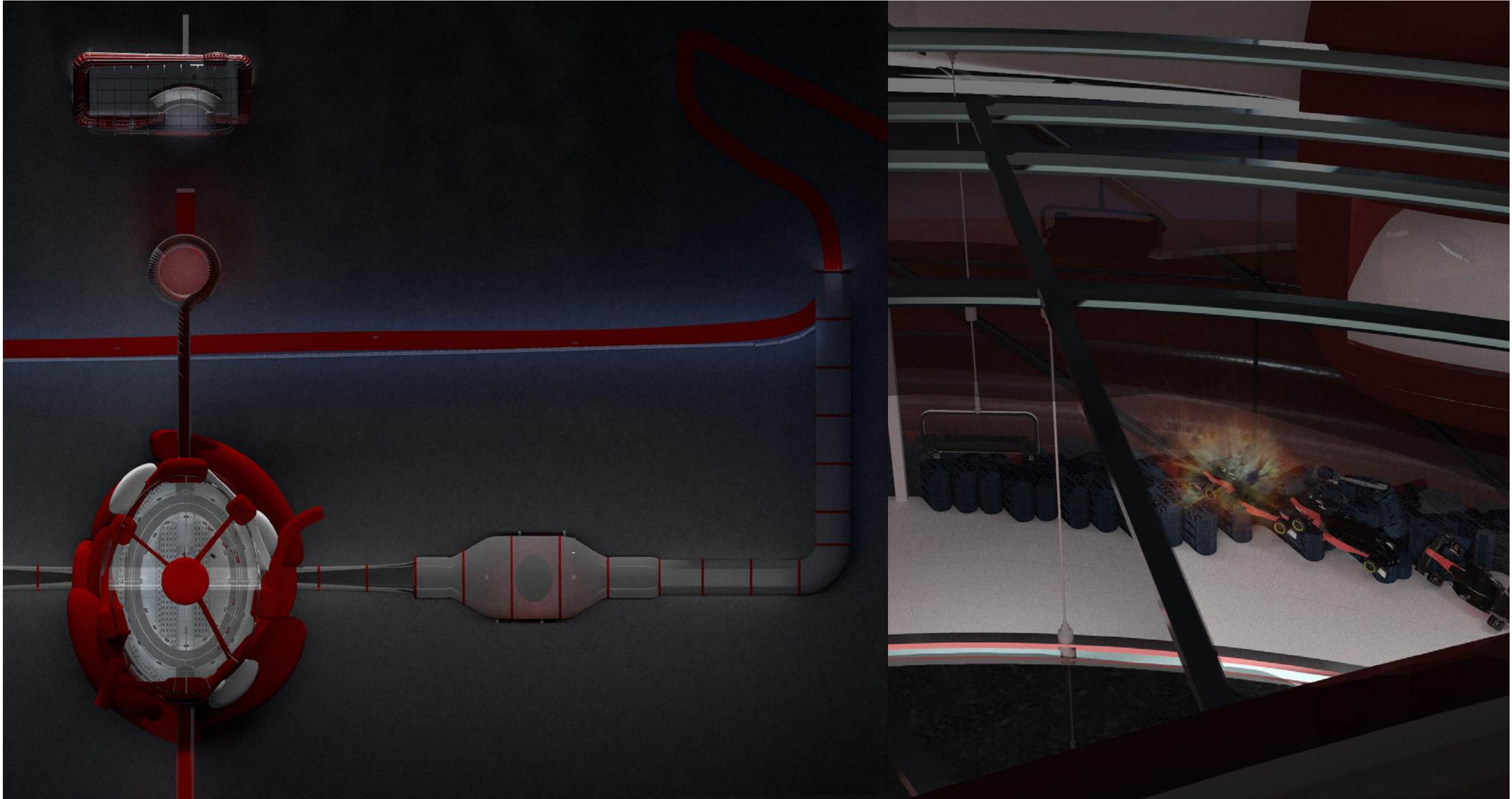


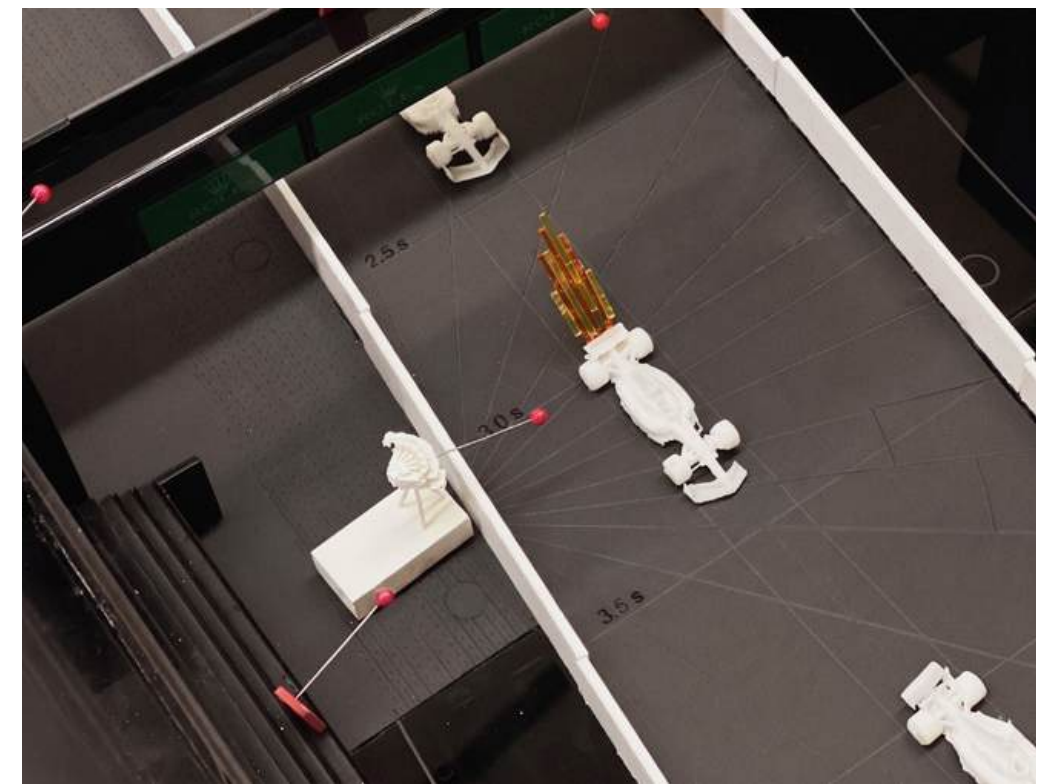
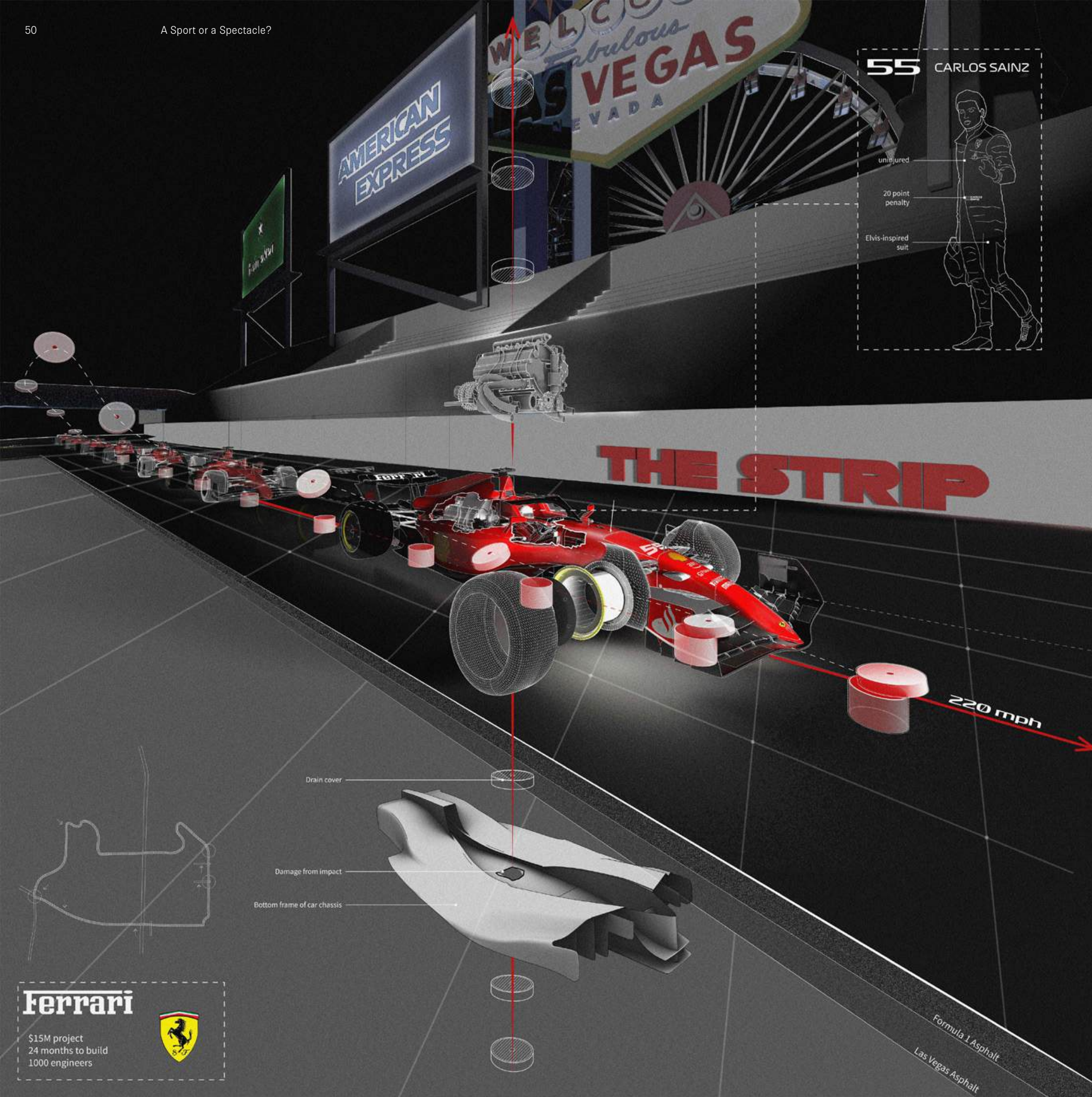
03 A Sport or a Spectacle?

Spring 2024
Core II Studio
Instructor: Jarrett Ley
Individual Work

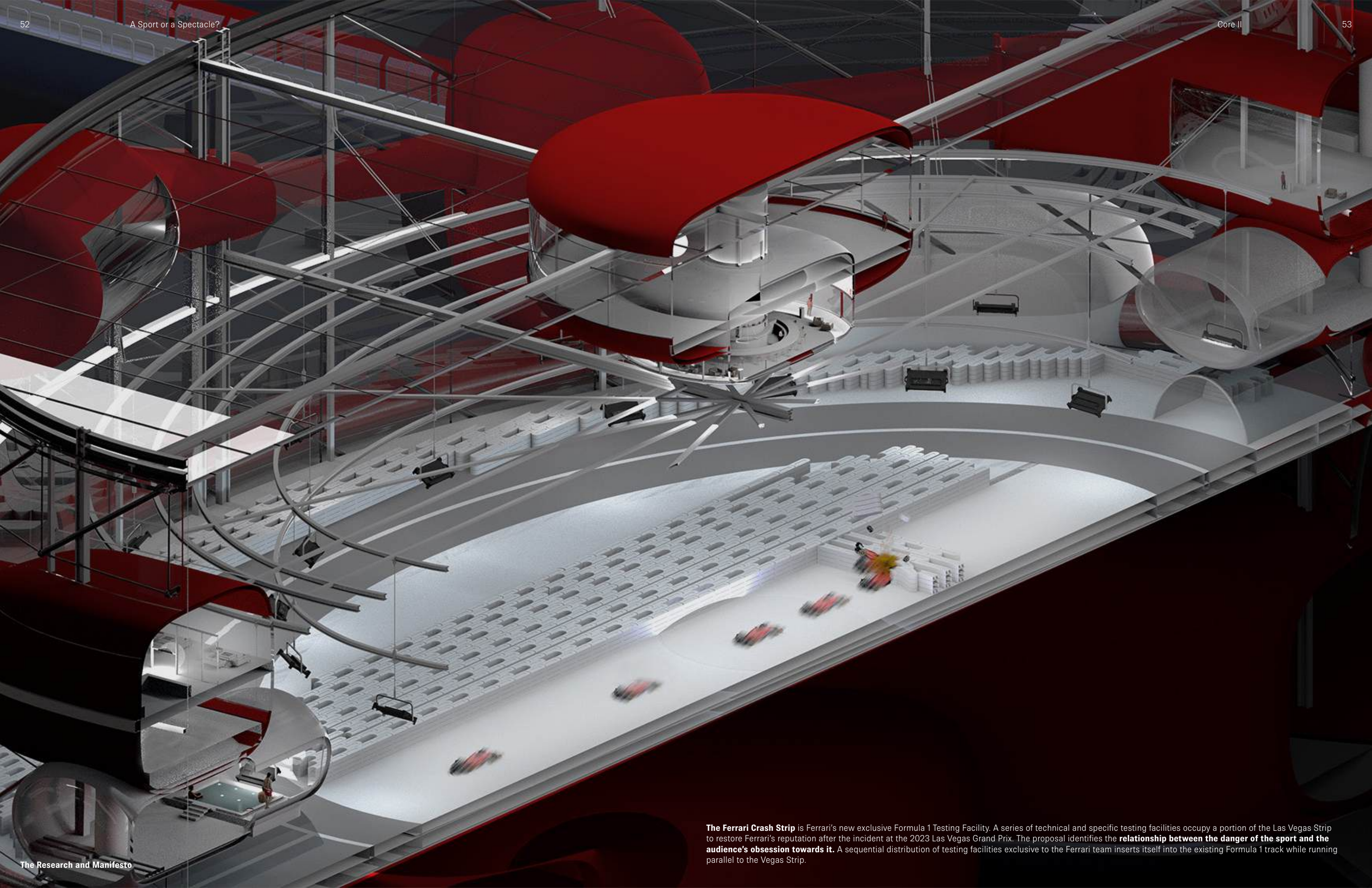
Site: Las Vegas, NV, USA

The **Formula 1 Grand Prix** in the center of Las Vegas was a highly anticipated event that after 2 years of planning and construction at a cost of over **520M dollars**, stopped in the first eight minutes of the first practice round. Carlos Sainz, a driver for Ferrari, drove over an unsealed manhole that popped open and **destroyed** the bottom of the 15M dollar car. A car that was designed by a team of 1000 engineers over the course of about 24 months. The site of damage becomes one that concerns multiple architectures, one which concerns **a sport of precise engineering** that is calibrated on risk taking, and the other which focuses on **a system of spectacle** that extracts those images of danger and thrill. The proposal attends to the **spectacularity and exclusivity** of the sport while re-purposing the nature of the Formula 1 racing track to that of **crash testing courses**, where the elements of safety and viewership allow for a new experience based on the live sensation of watching the sport.

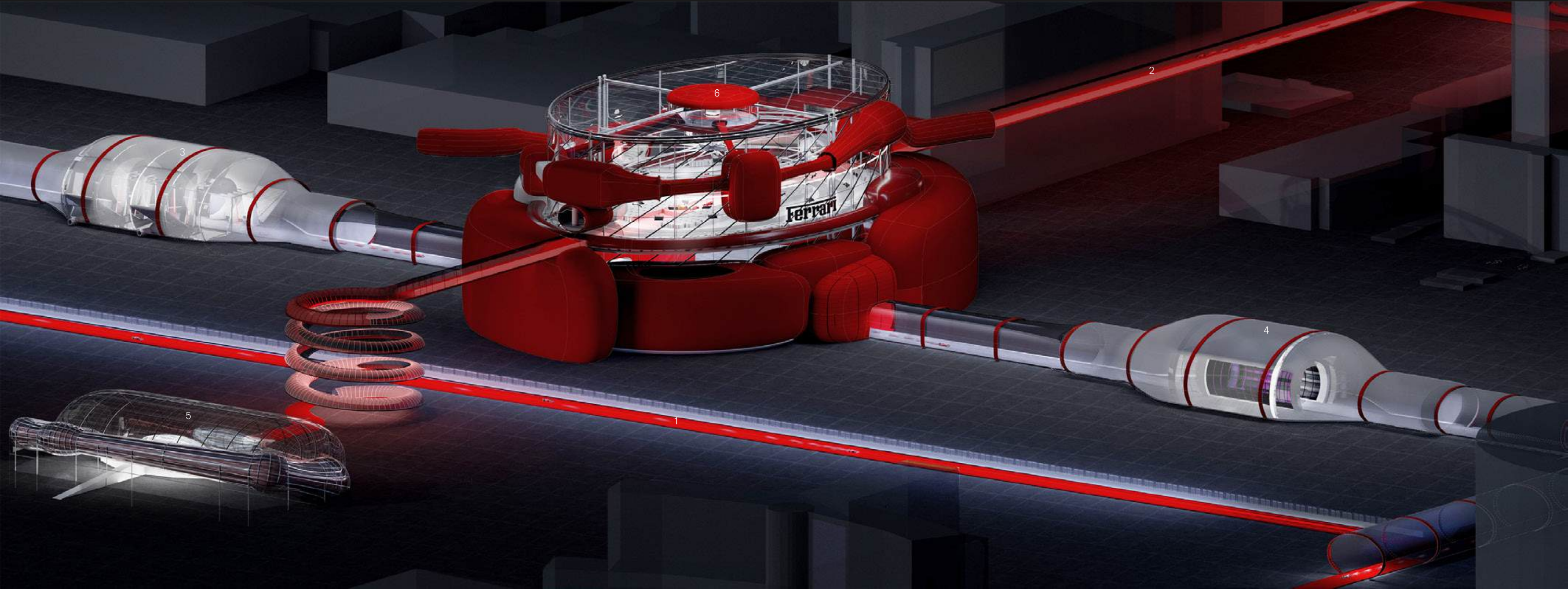




Analytical model of Carlos Sainz's crash, identifying the media's readiness in capturing the thrill.



The Ferrari Crash Strip is Ferrari's new exclusive Formula 1 Testing Facility. A series of technical and specific testing facilities occupy a portion of the Las Vegas Strip to restore Ferrari's reputation after the incident at the 2023 Las Vegas Grand Prix. The proposal identifies the **relationship between the danger of the sport and the audience's obsession towards it**. A sequential distribution of testing facilities exclusive to the Ferrari team inserts itself into the existing Formula 1 track while running parallel to the Vegas Strip.



1 the original F1 track

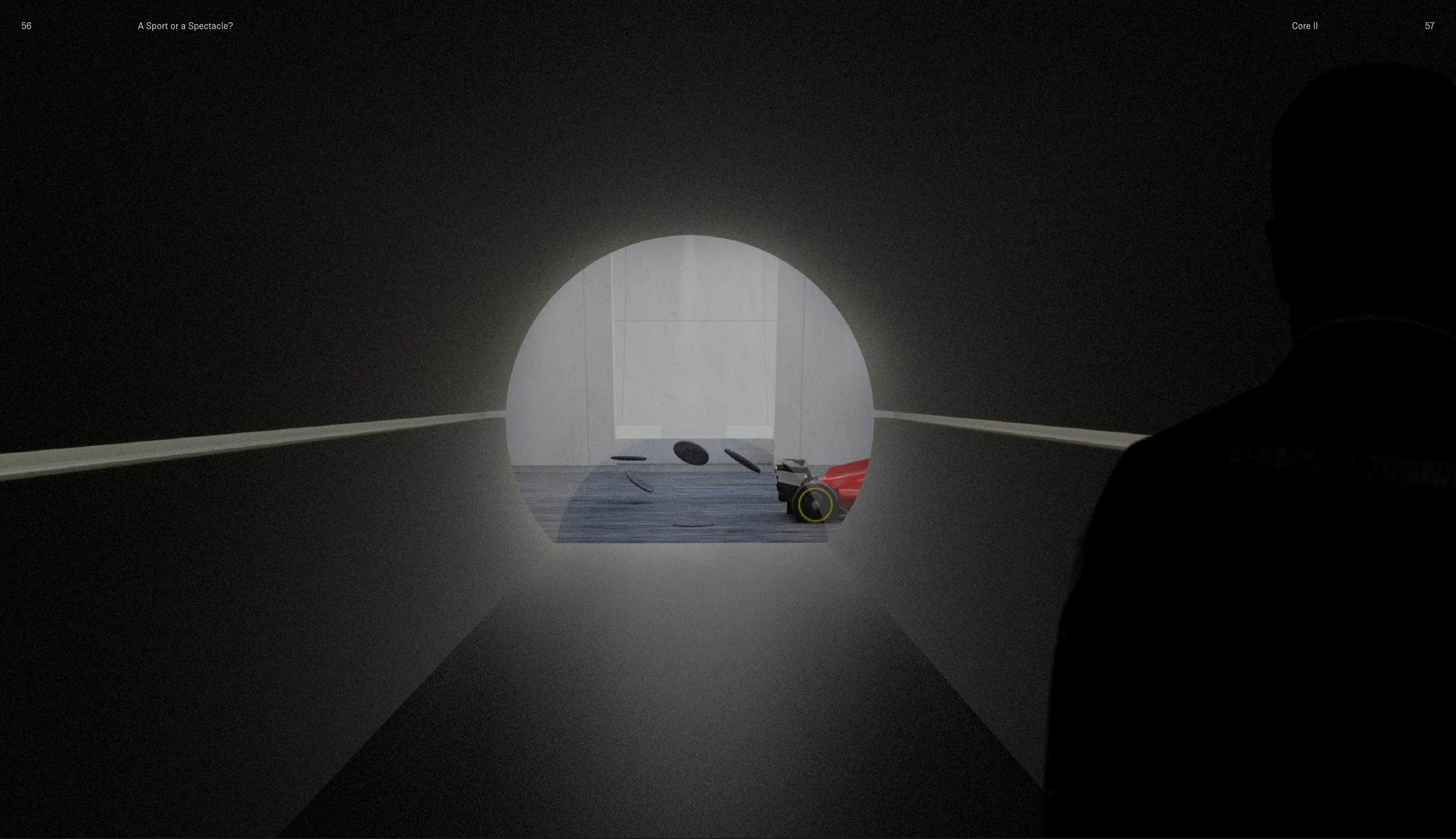
2 speed-generating track

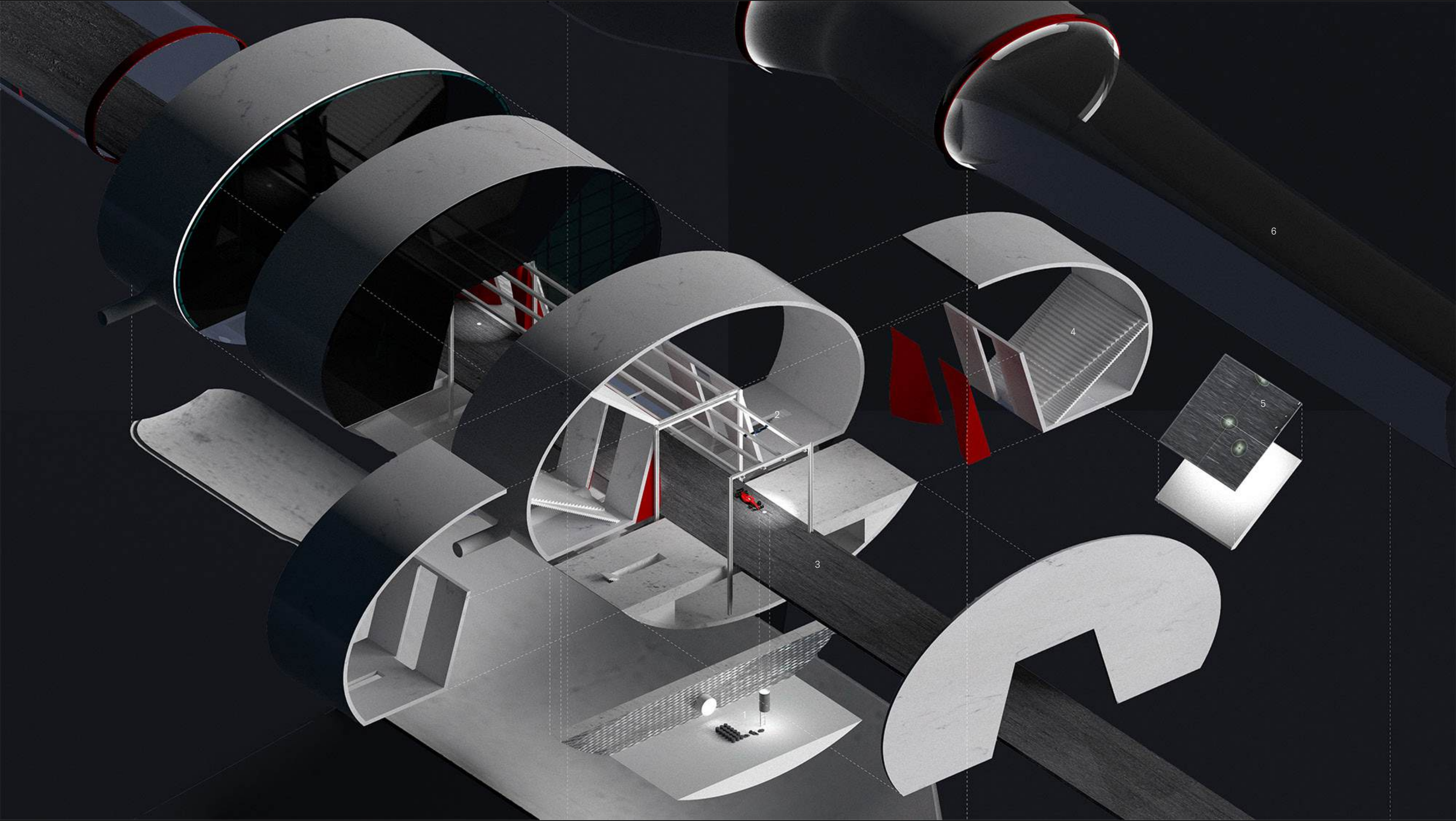
3 manhole sealing experiments

4 tyre temperature analysis

5 wind tunnel analysis

6 the crash test





1 underground testing facility
for access to manholes

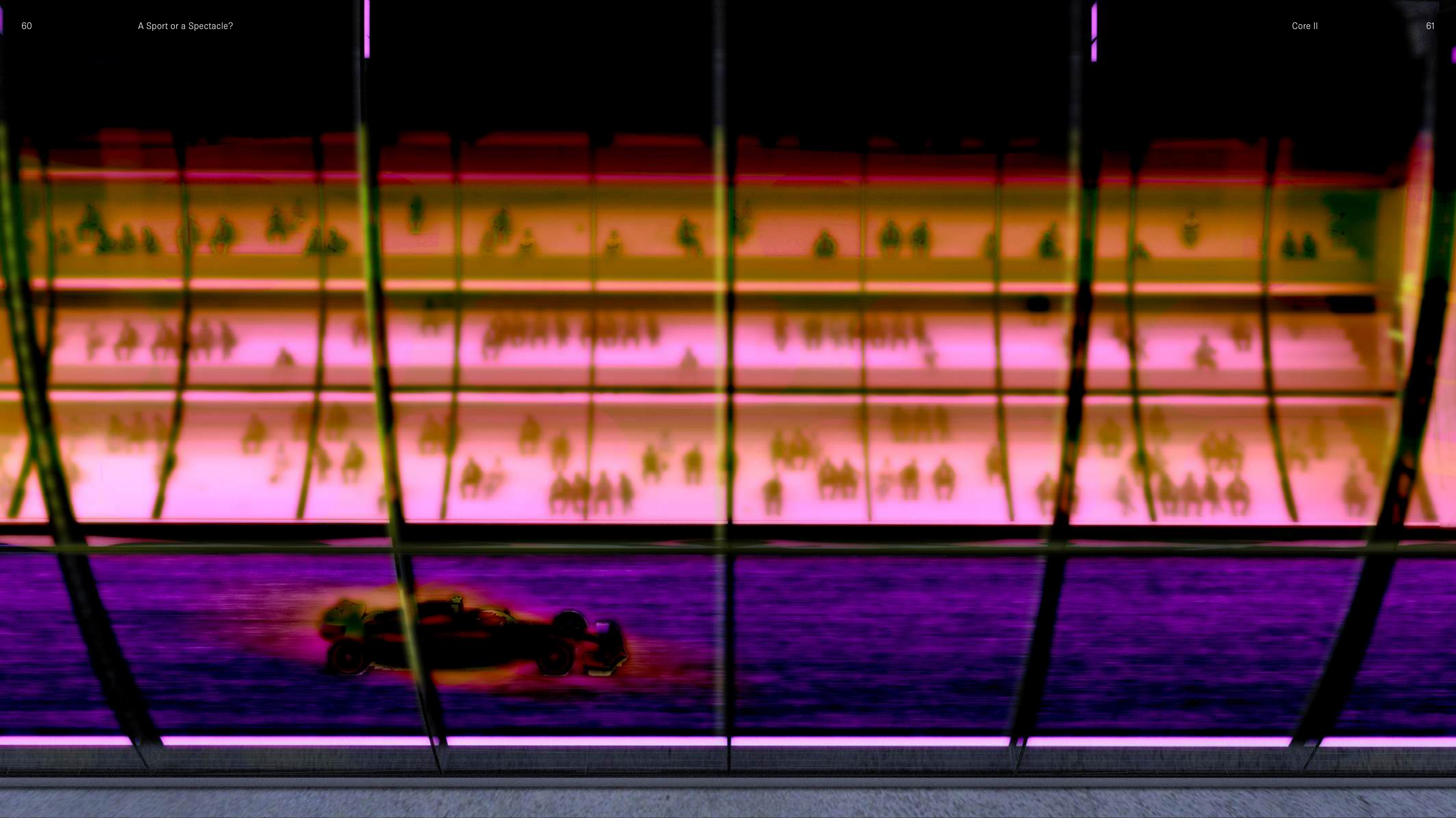
2 cable cameras

3 track

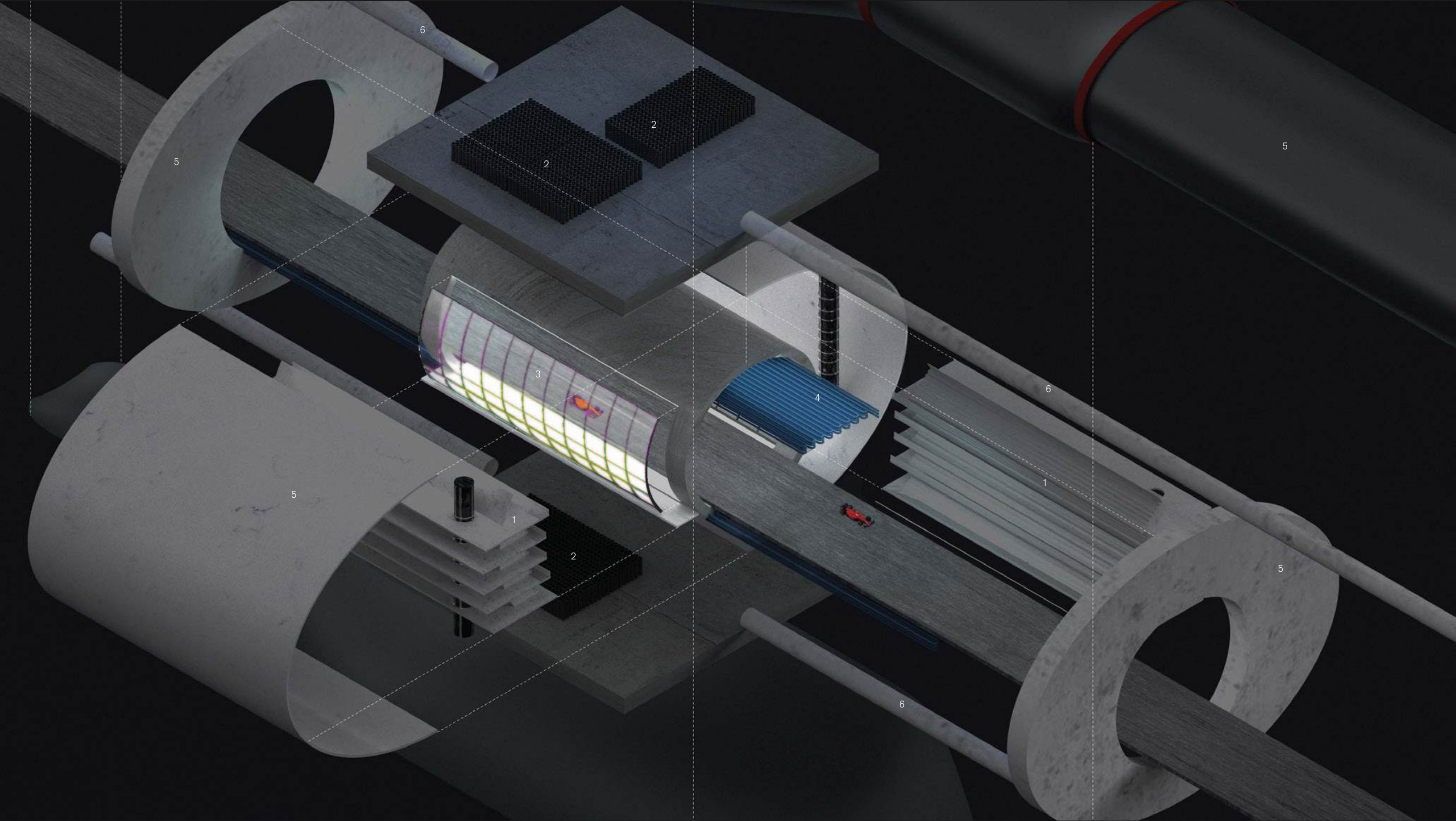
4 viewing bleachers

5 projection screen

6 track enclosure in
reinforced glass



Creating a new form of viewership of a Formula 1 race, this time from the extraction of the danger behind the temperature of the environment in which a car has to perform. The nature of the test comes from the speculation of Las Vegas' credentials as a new location and track for the whole sport. To test these environments, the proposal becomes an environmentally controlled chamber designed to test the car's reaction to the coldest temperatures of the different Formula 1 racetracks across the world.



1 viewing bleachers

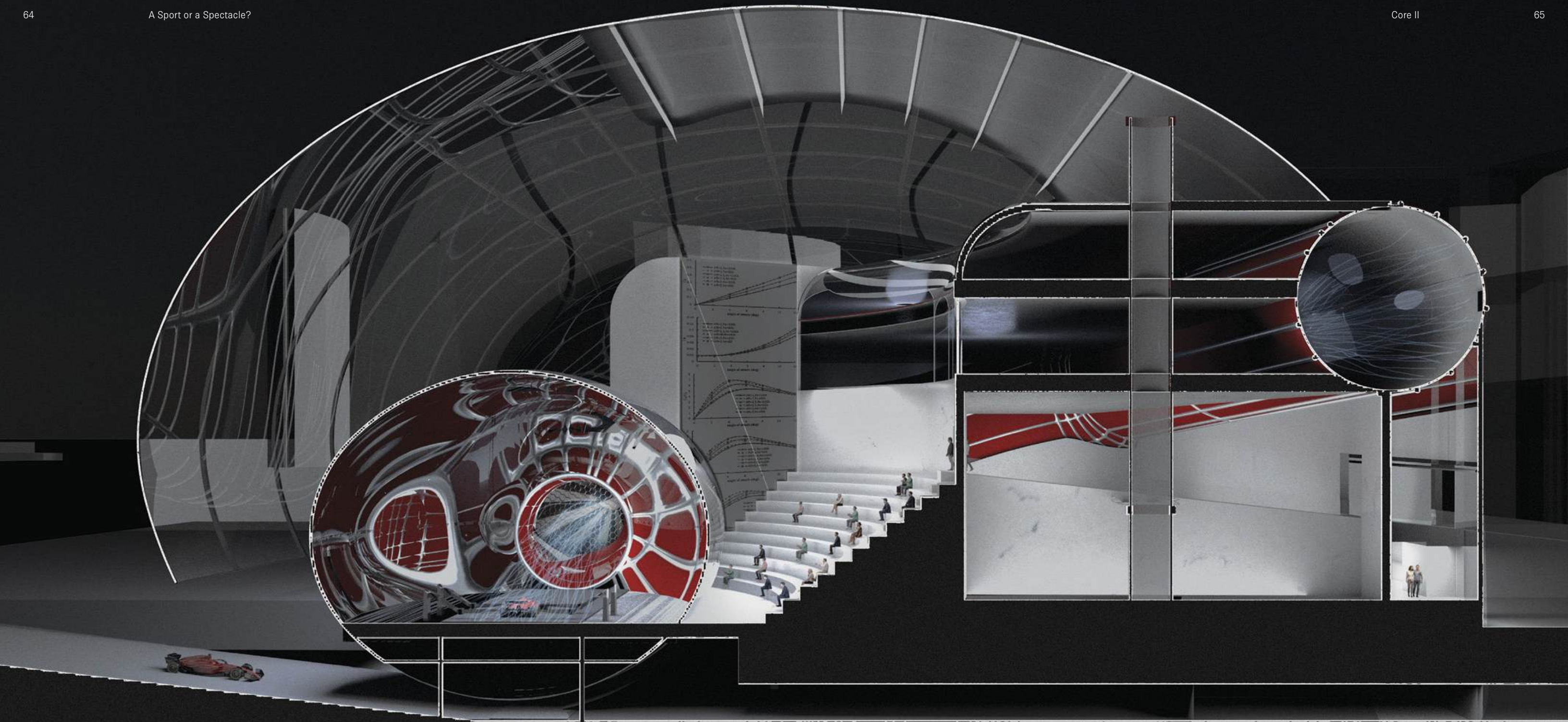
2 extra tyre storage units

3 thermal imaging screens for infrared analysis

4 cooling pipes

5 chamber sealing for temperature control

6 tunnel circulation for engineers





1 wind tunnel

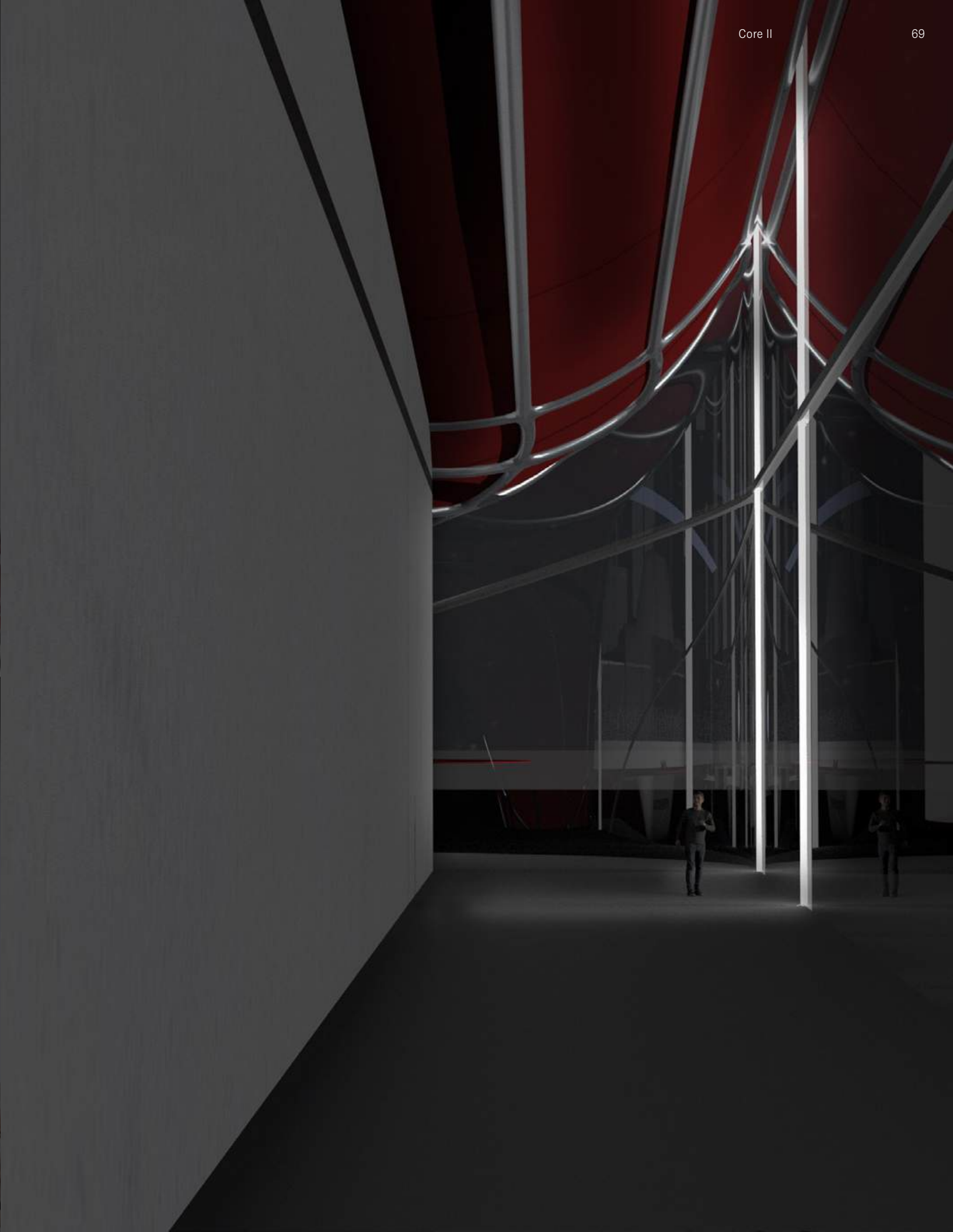
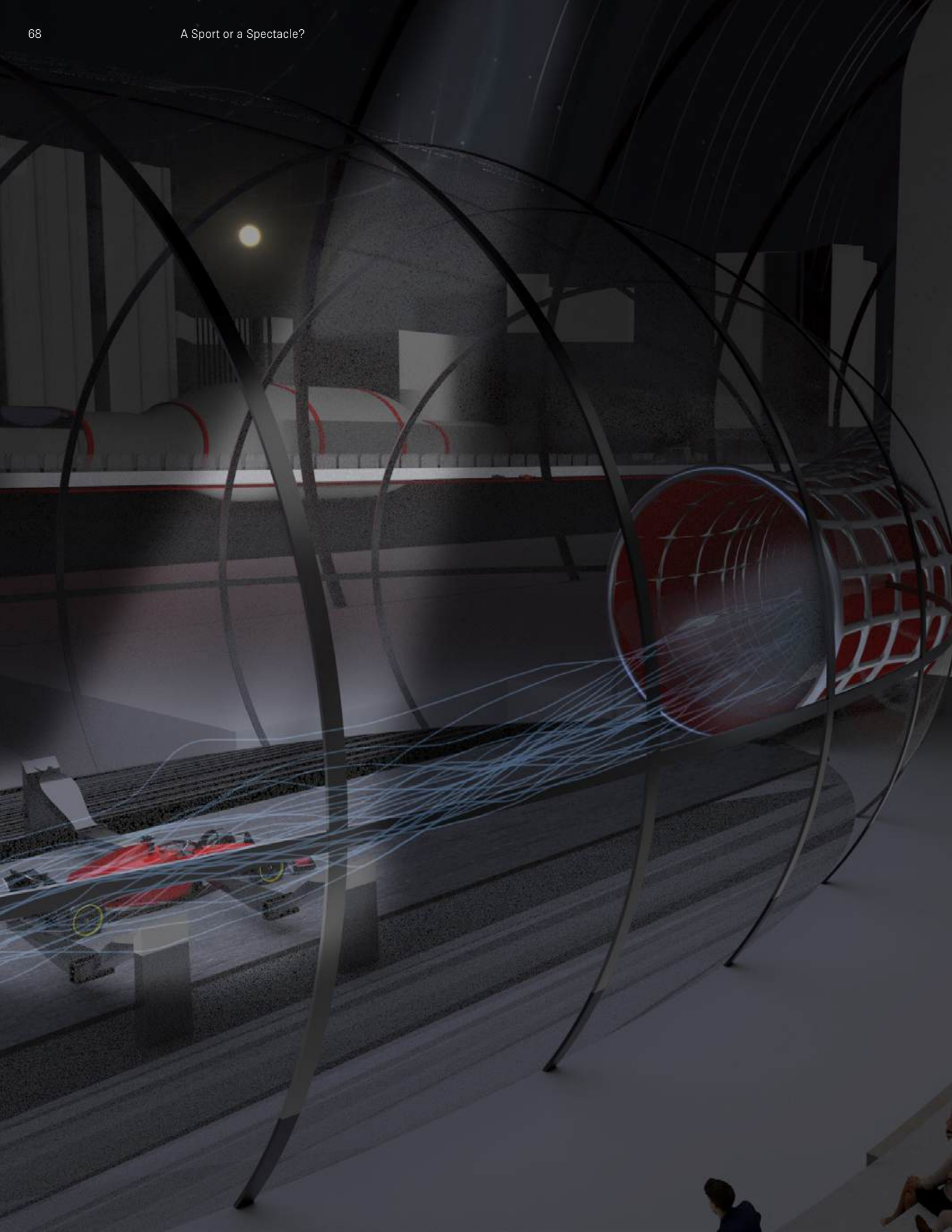
2 curved glass lab for car

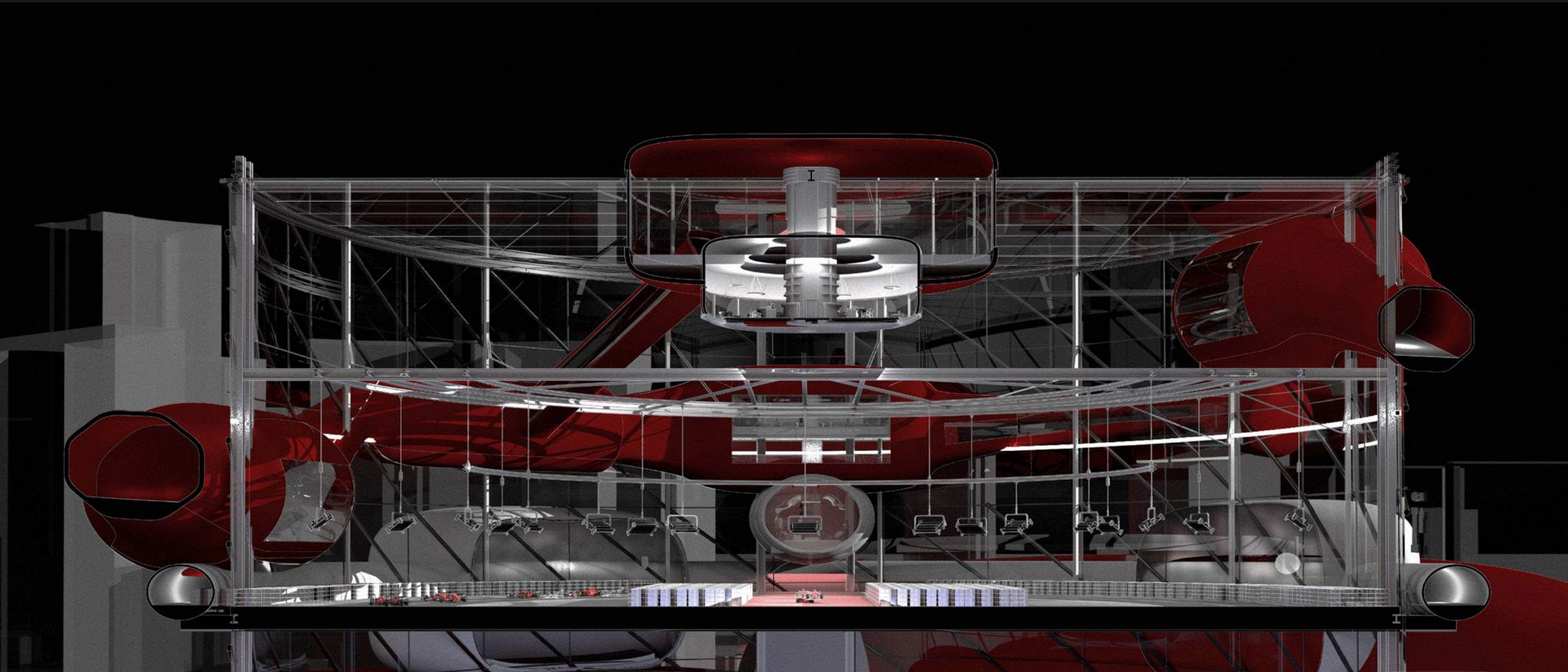
3 viewing bleachers

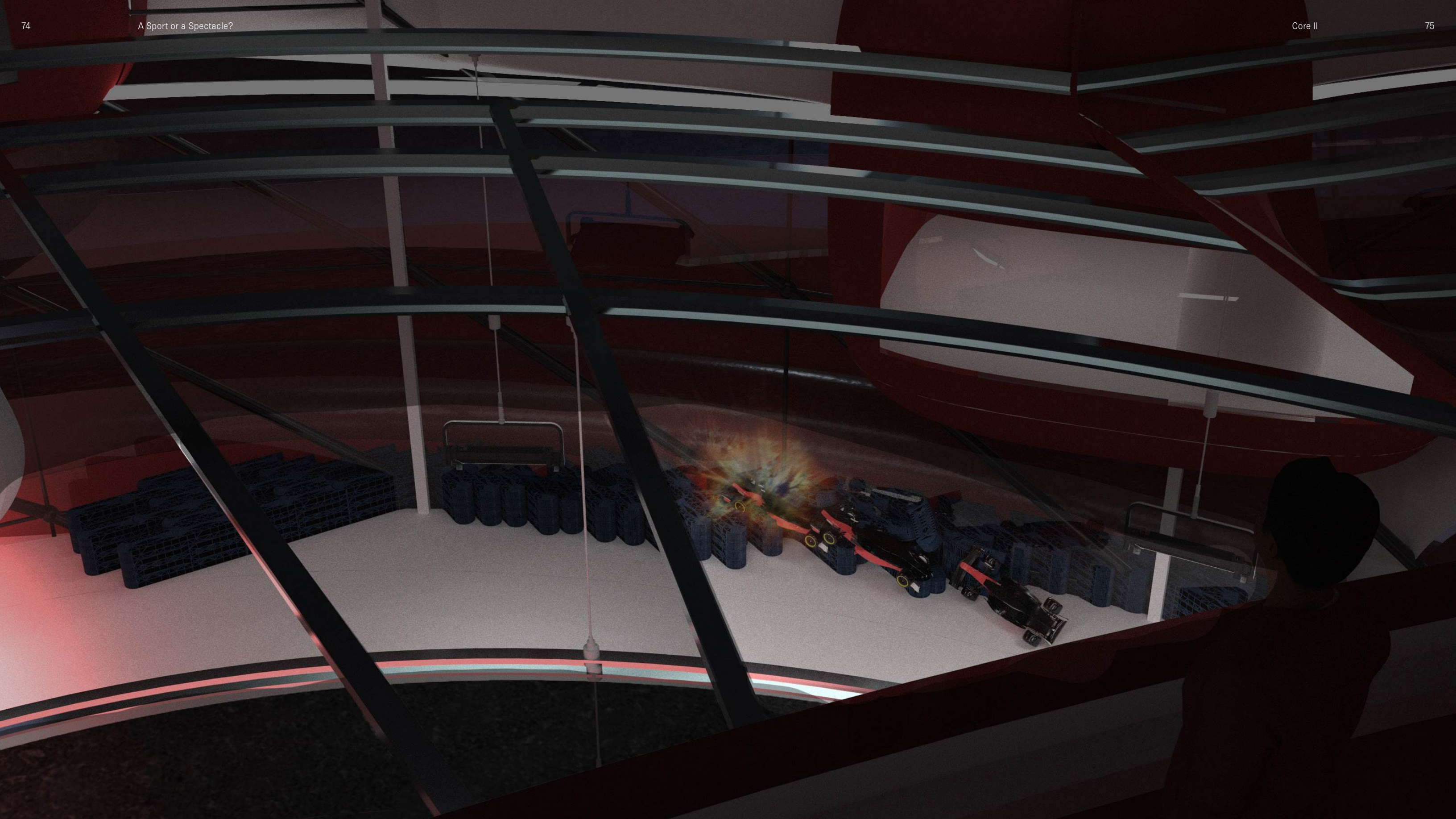
4 projection mapping

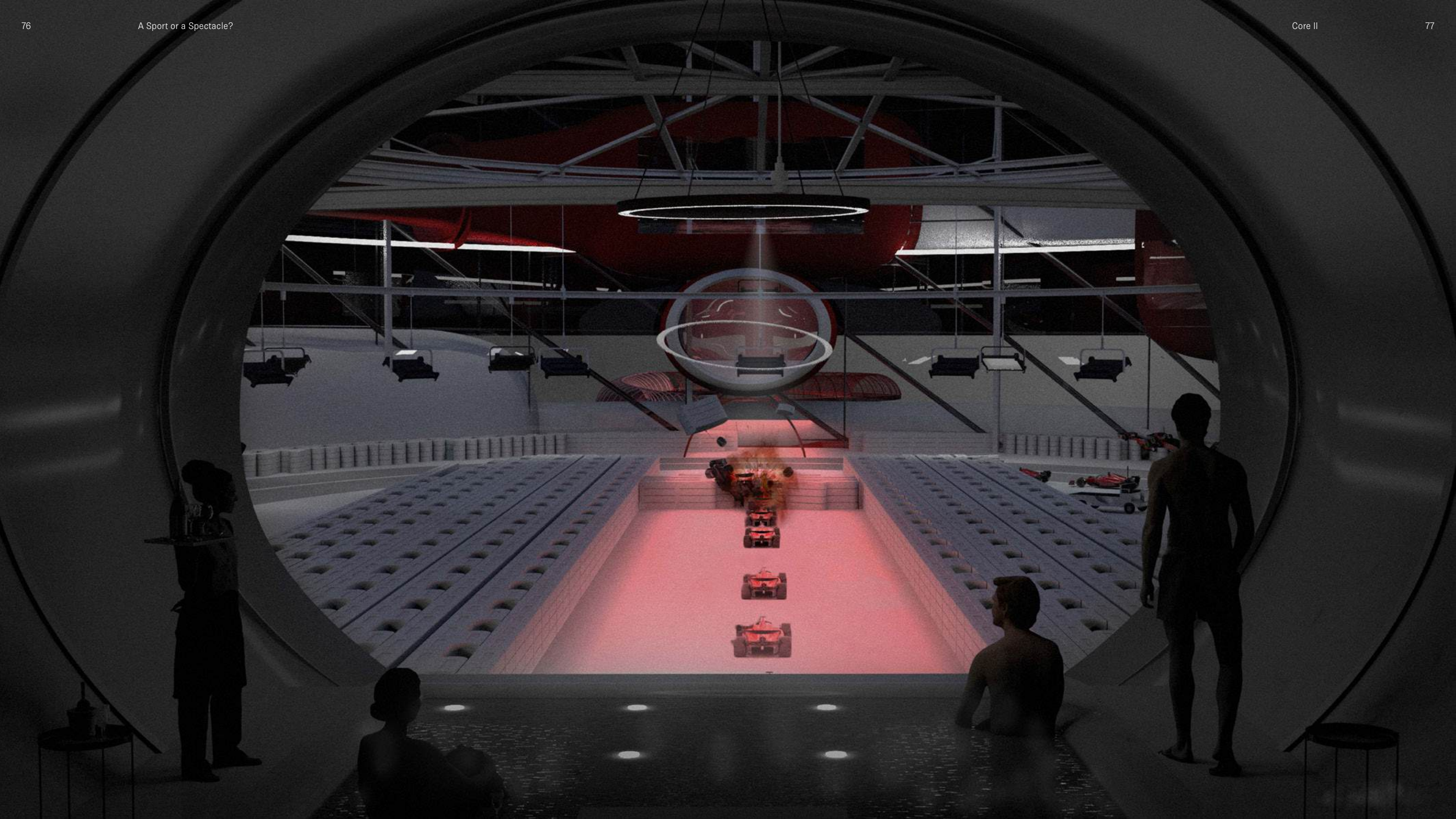
5 pathway for viewers

6 entry from main track







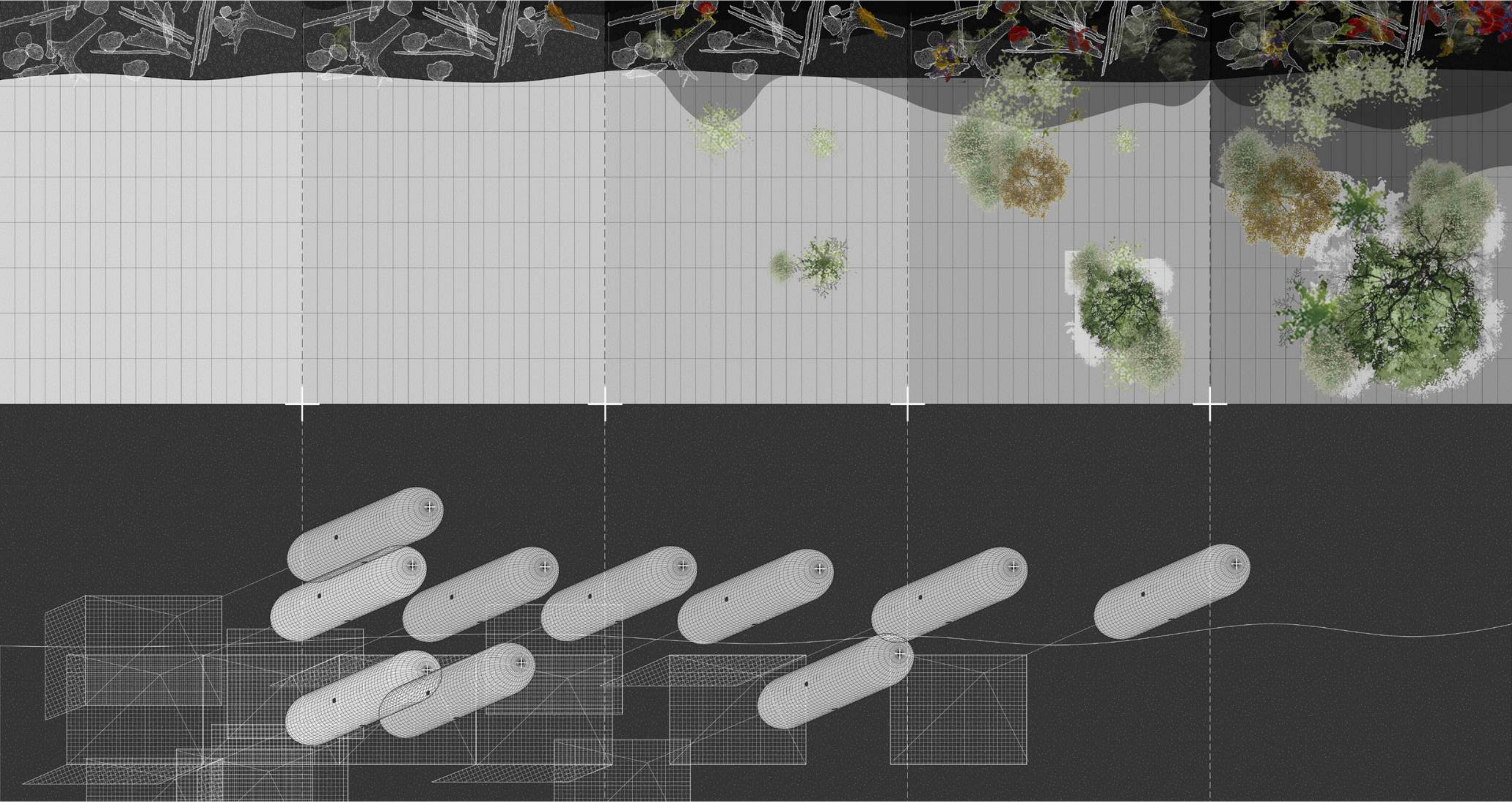


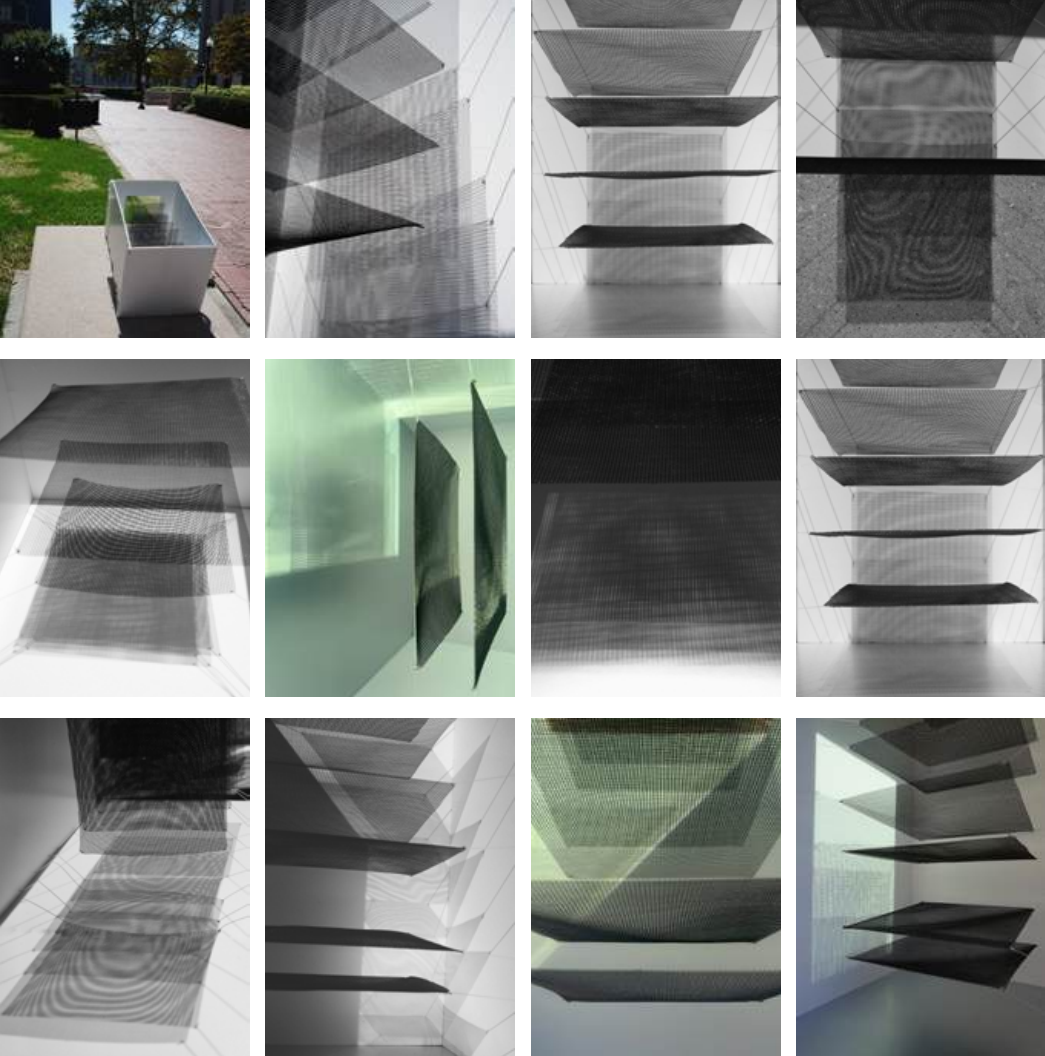
04 The Lichen Archive

Fall 2025
ADV V Studio
Instructors: Leslie Gill and Khoi Nguyen
Individual Work

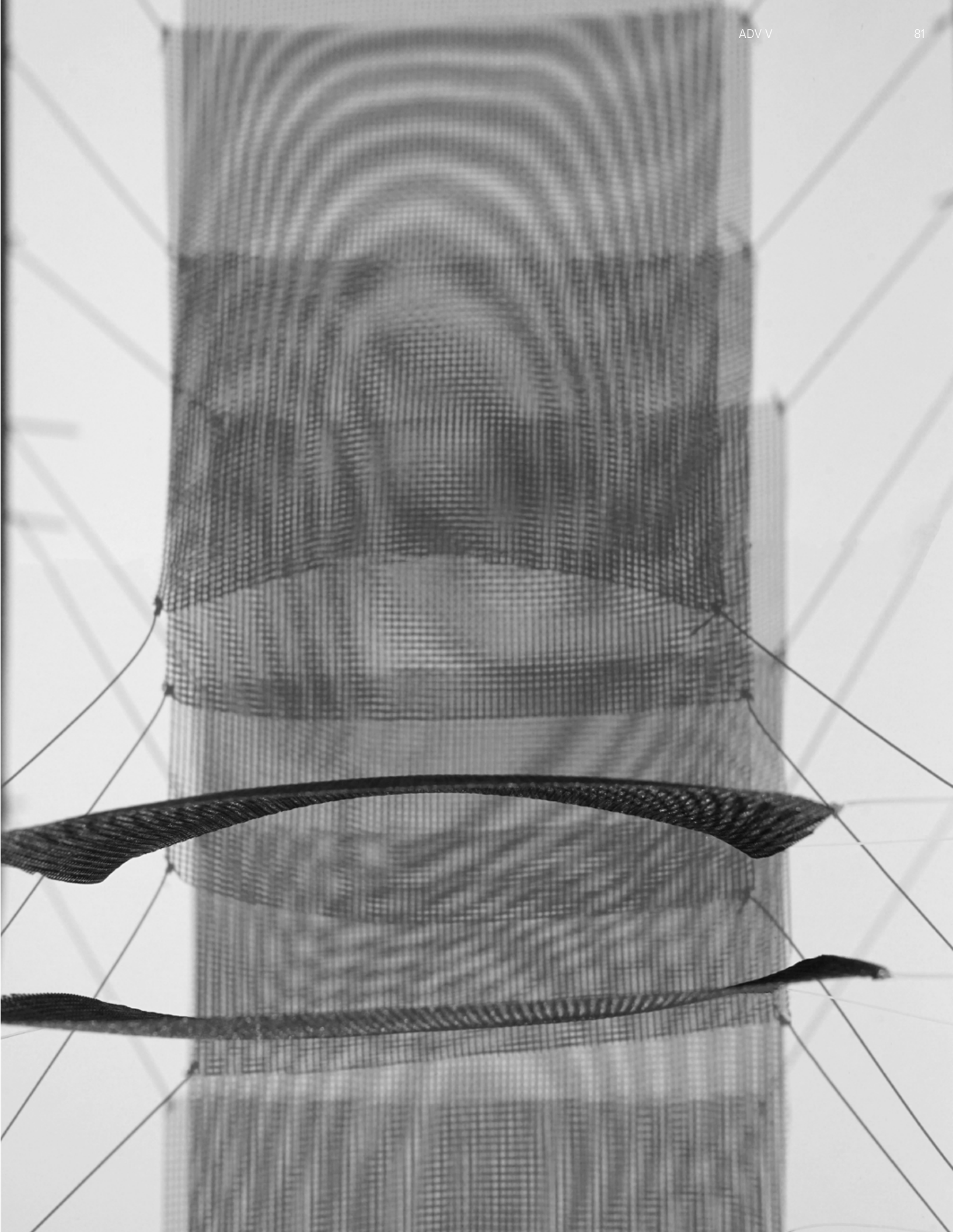
Site: Inari, Finland

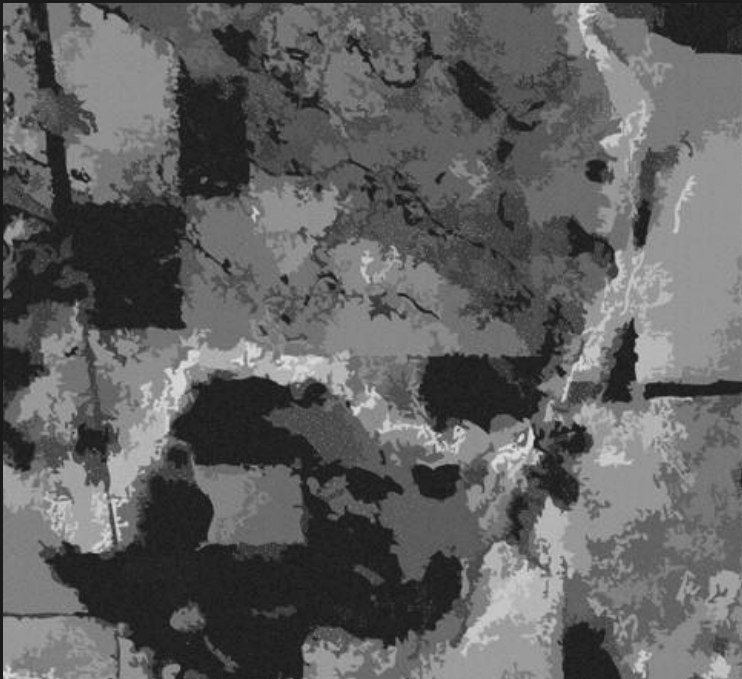
Situated within the old-growth **boreal forests of the arctic Finnish Lapland**, the site exists as an ecological archive whose continuity has been progressively **fragmented** through contemporary forest management practices. Logging roads, canopy clearings, and managed planting grids disrupt the forest's ability to regulate light, humidity, and temperature, dismantling the microclimatic conditions required for microspecies such as **lichens, mosses, and fungi** to survive. Growing over decades and absorbing moisture directly from the atmosphere, these organisms function as indicators of permanence, appearing only within undisturbed forest systems. **Their disappearance signals both ecological rupture and the erosion of forest memory.** The Lichen Archive operates as a speculative research framework that reappropriates **forest management infrastructures** and byproducts to redistribute substrates and attenuate light, **stabilizing microclimates within fragmented canopies.** Through temporary, dismantlable outposts and monitored experiments, architecture acts as a temporal scaffold, enabling regrowth while leaving behind living archives.





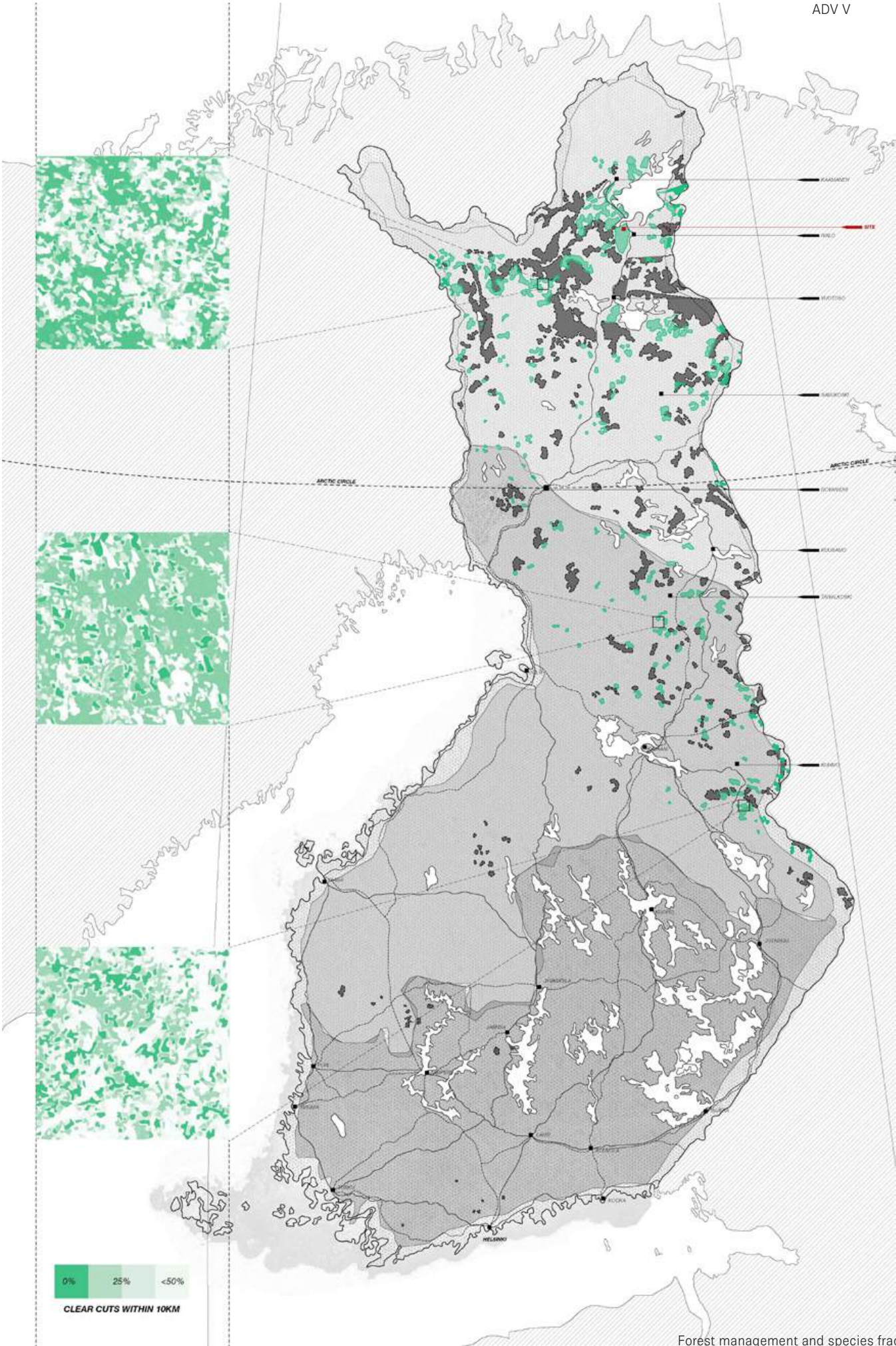
Light Attenuation in the Boreal Forest
Filtration of light through the forest's strata: a light study device mimicking the superimposition of leaves in coniferous trees.



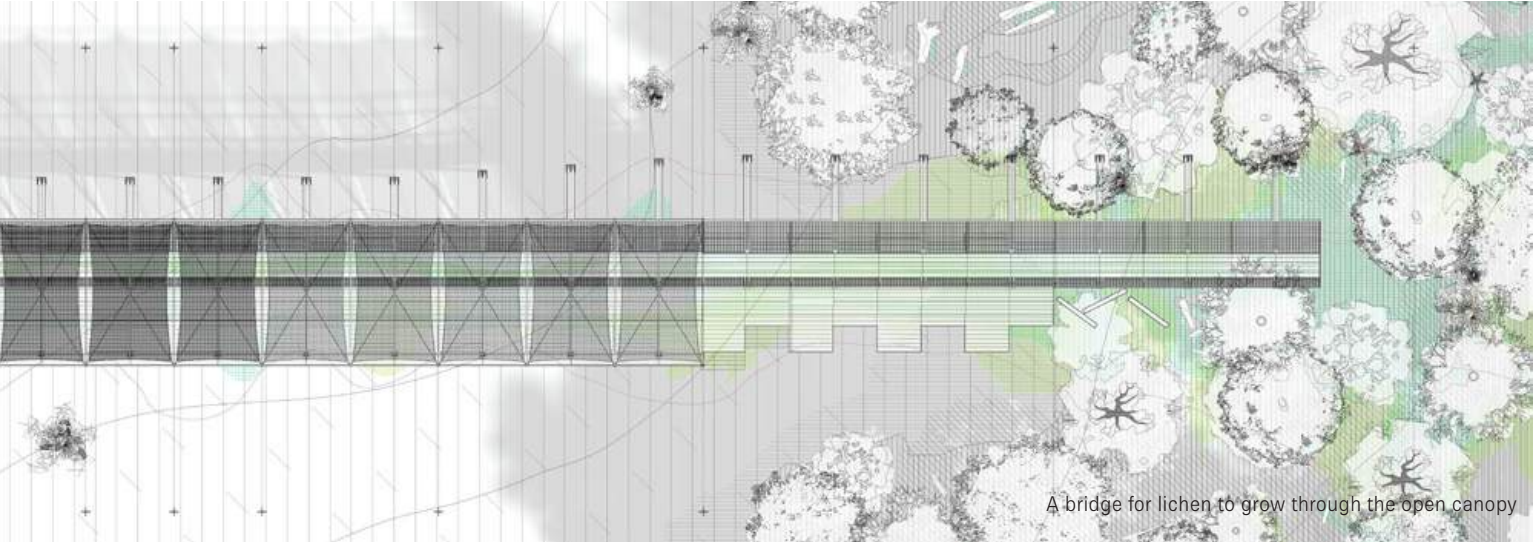
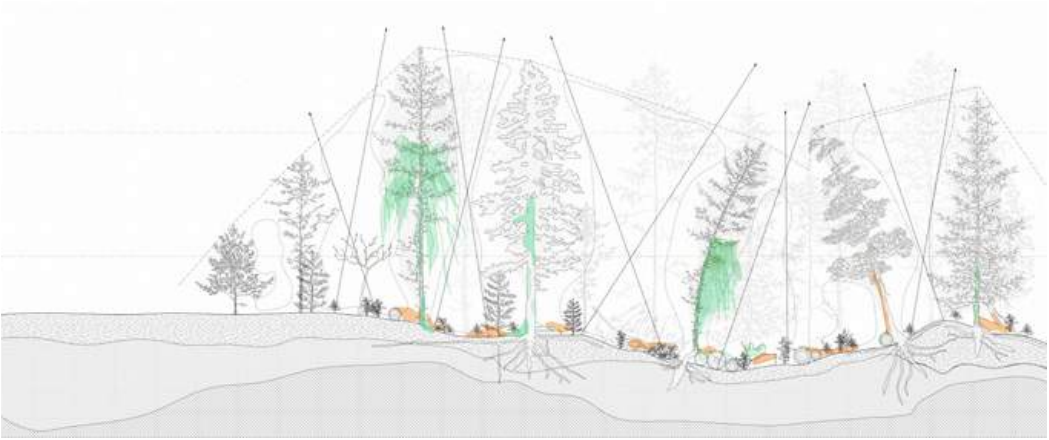
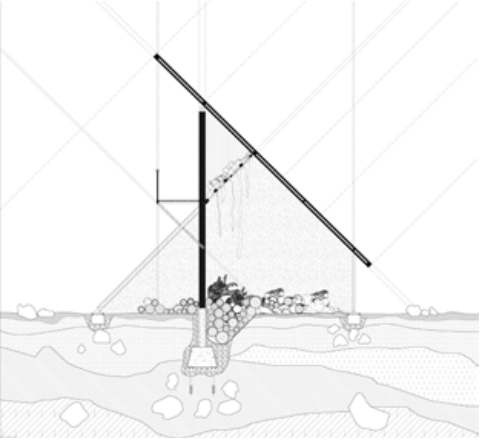


1 Managed Forest fragmentation

2 Old Growth Forest striation, erosion



Forest management and species fragmentation in Finland



Lichen lives on old growth substrates, most of which are removed in the process of deforestation. These early studies begin to take an approach to the redistribution of lichen, fungi, and moss through a linear bridge of deforestation waste. The initial outpost becomes an active experiment intended only for arctic microbiology scientists.

A bridge for lichen to grow through the open canopy

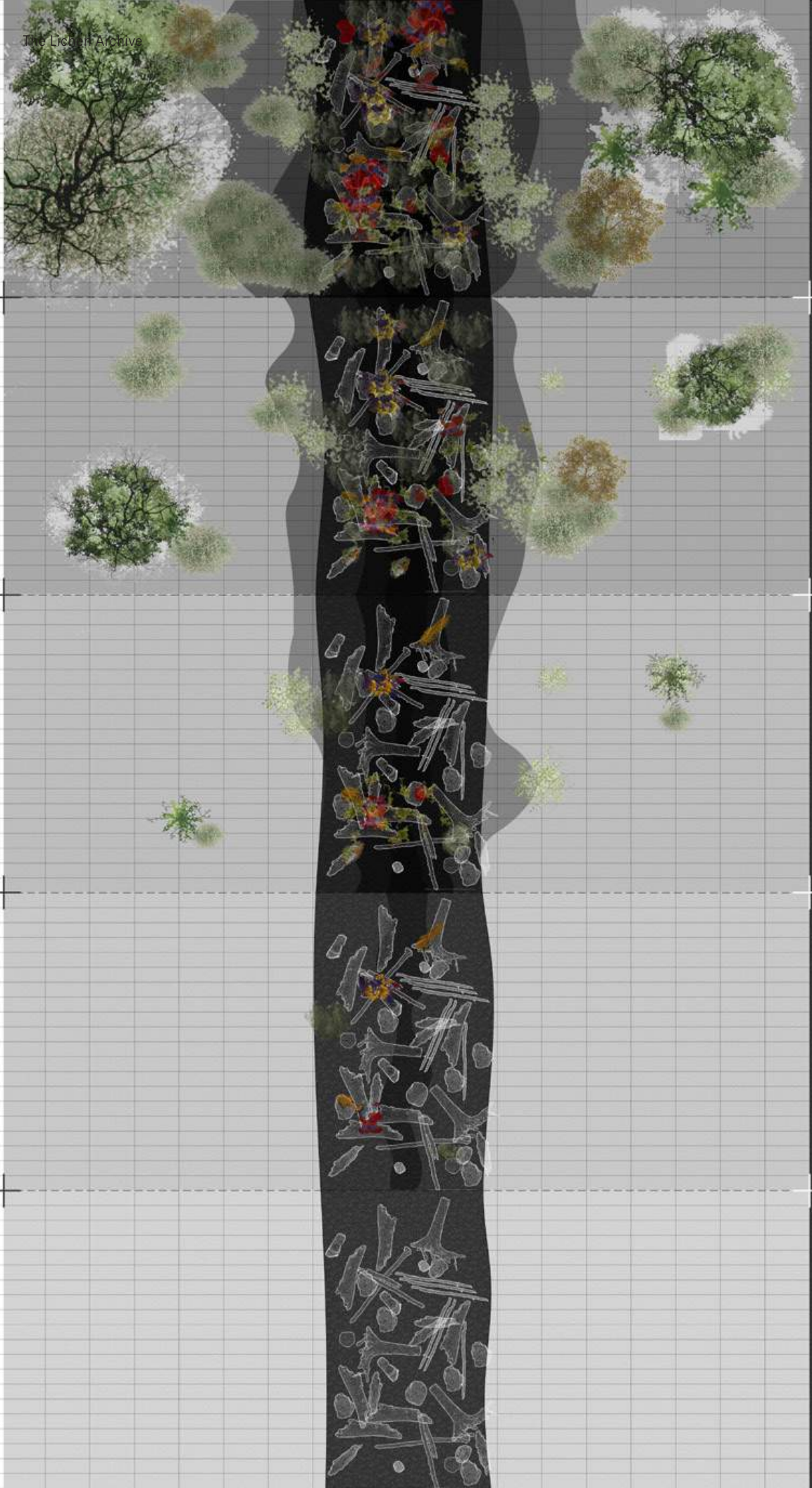
<30 YEARS
CLOSURE OF CANOPY

15 YEARS
DIVERSITY IS REINTRODUCED

5 YEARS
FOREST FLOOR REGROWTH

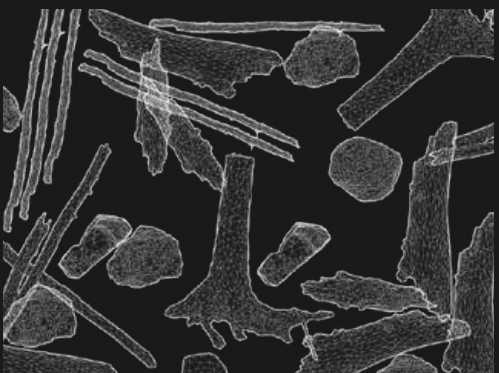
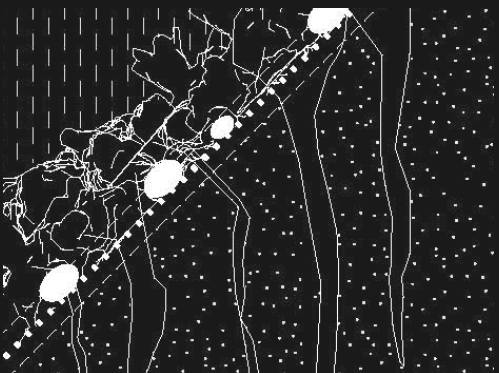
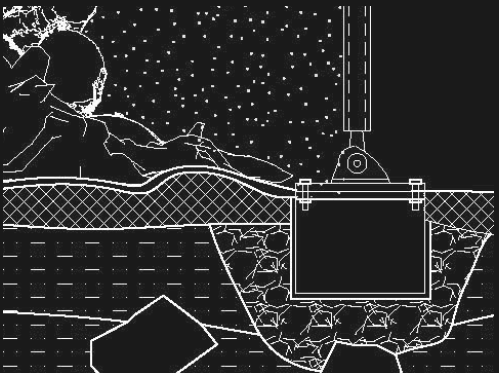
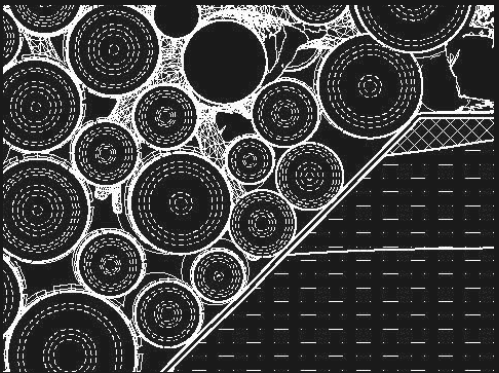
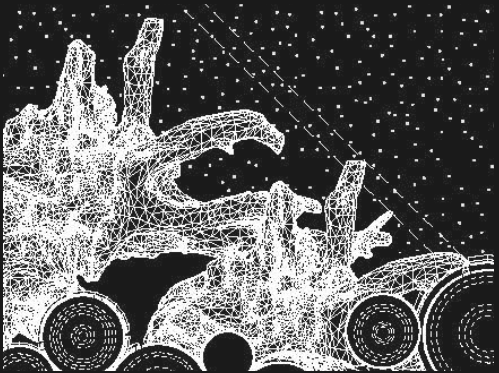
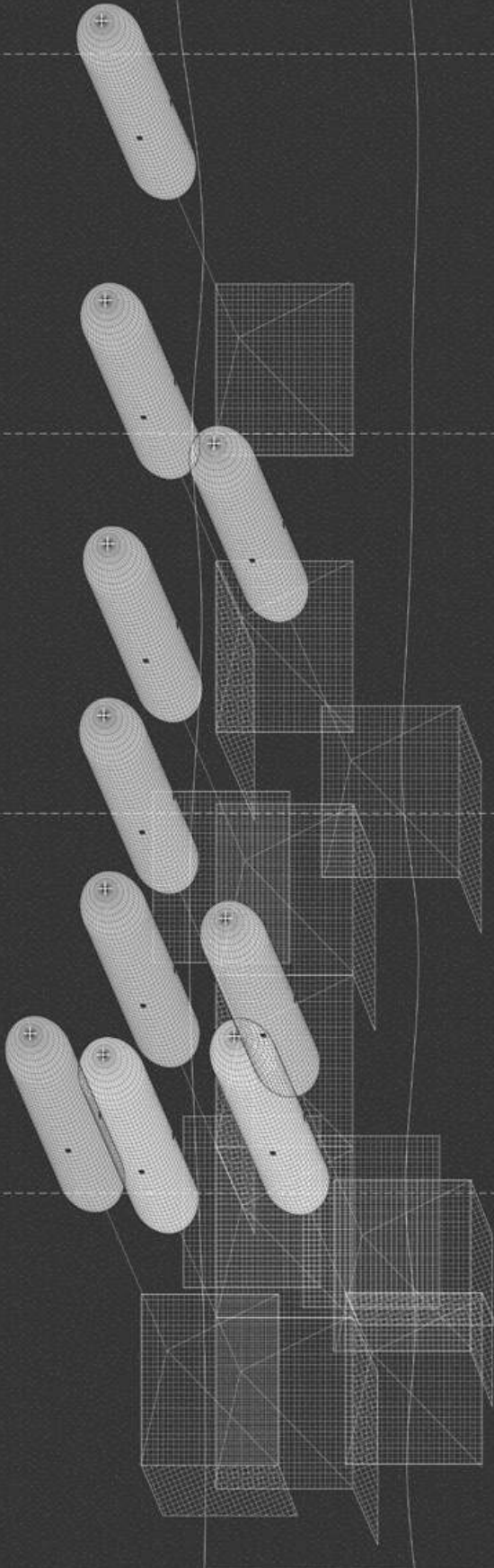
1 YEAR
LICHEN GROWTH BEGINNING

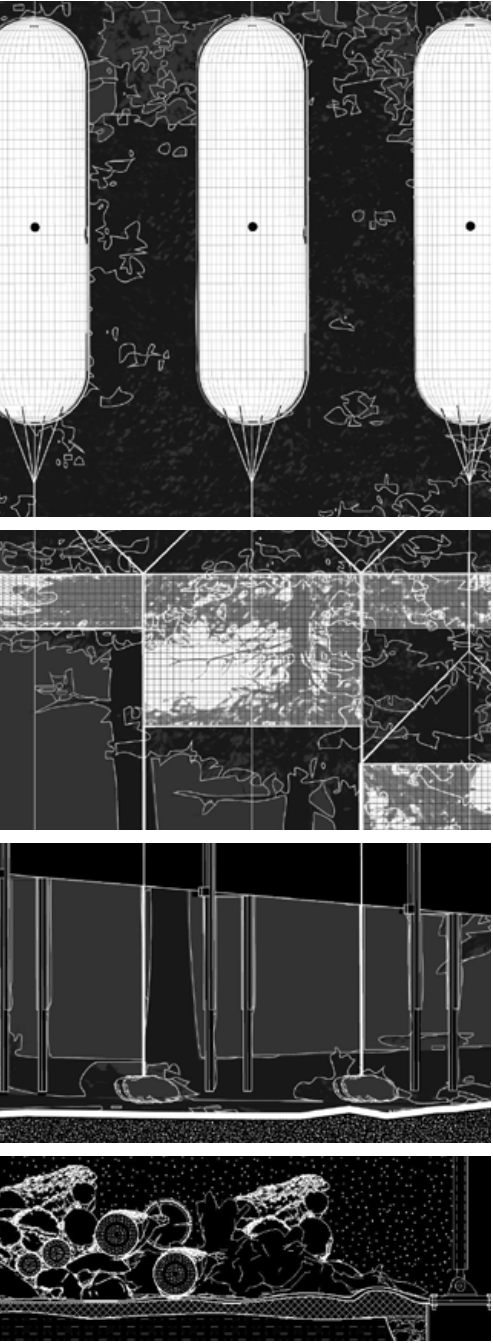
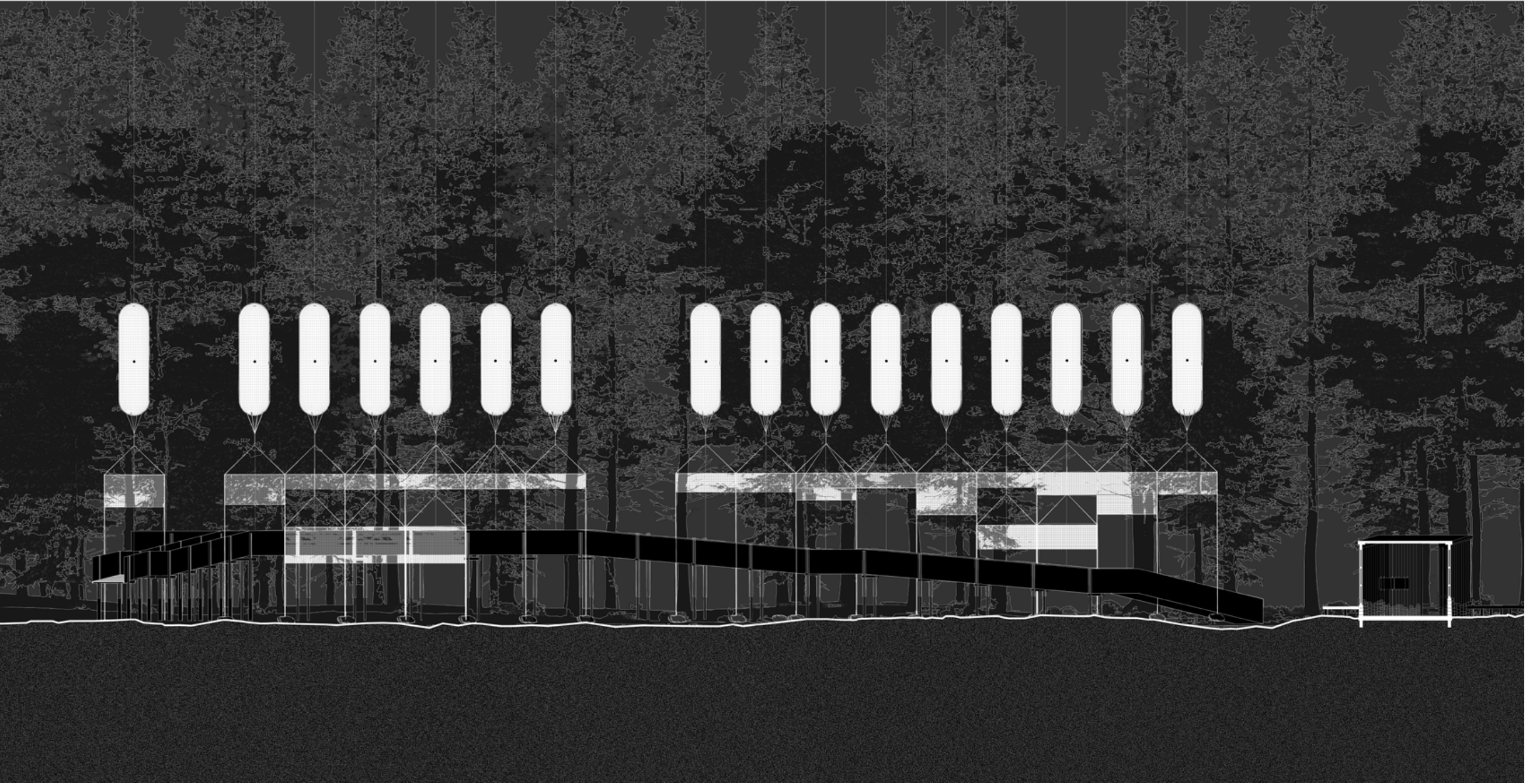
0 YEAR
INTRODUCTION OF
SUBSTRATES



Growth, Substrates, and Light

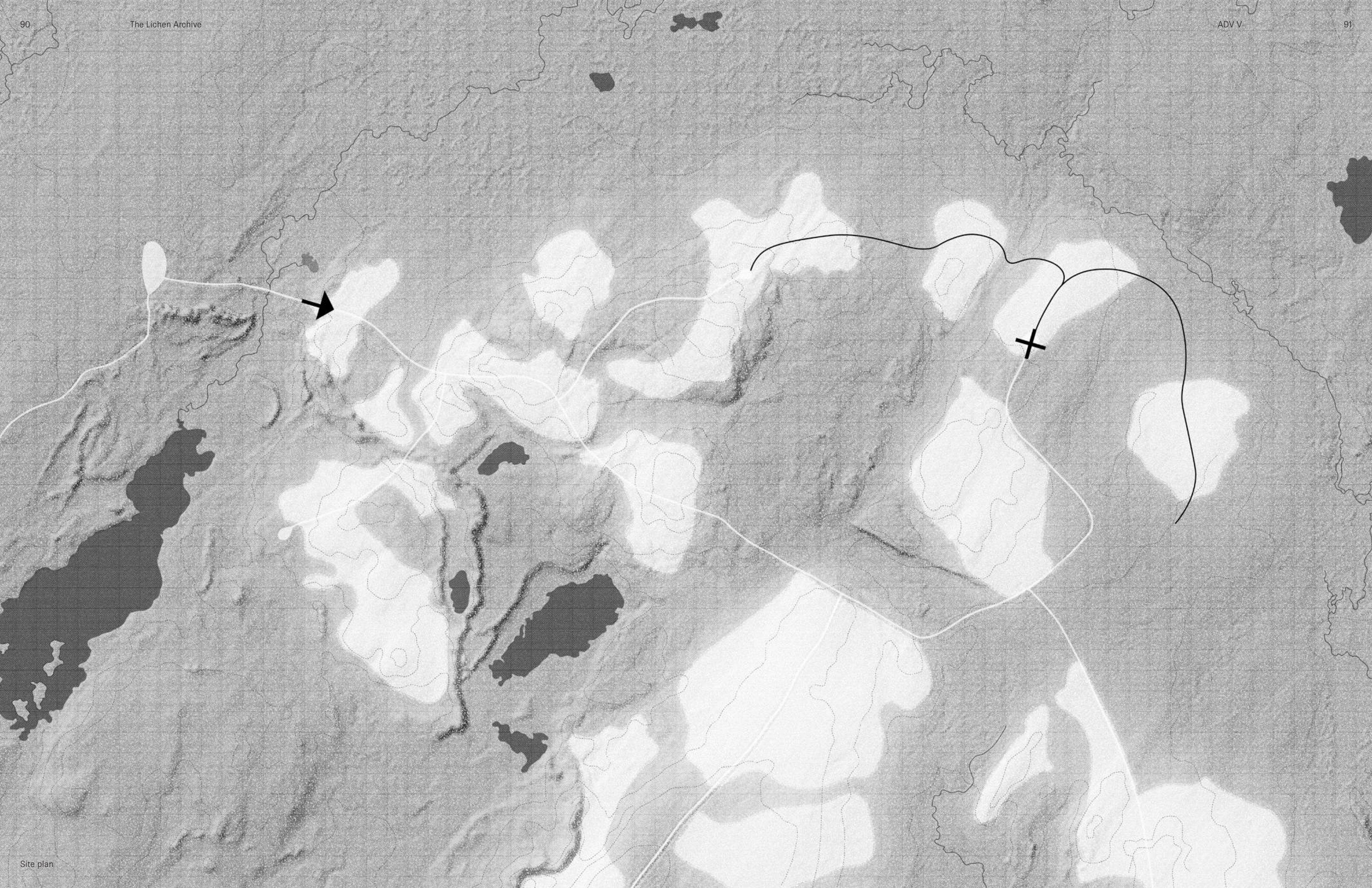
As lichen, moss, and fungi reclaim the reappropriated byproduct of forest management, the intervention disappears. Screens are dismantled and weather balloons are deflated.





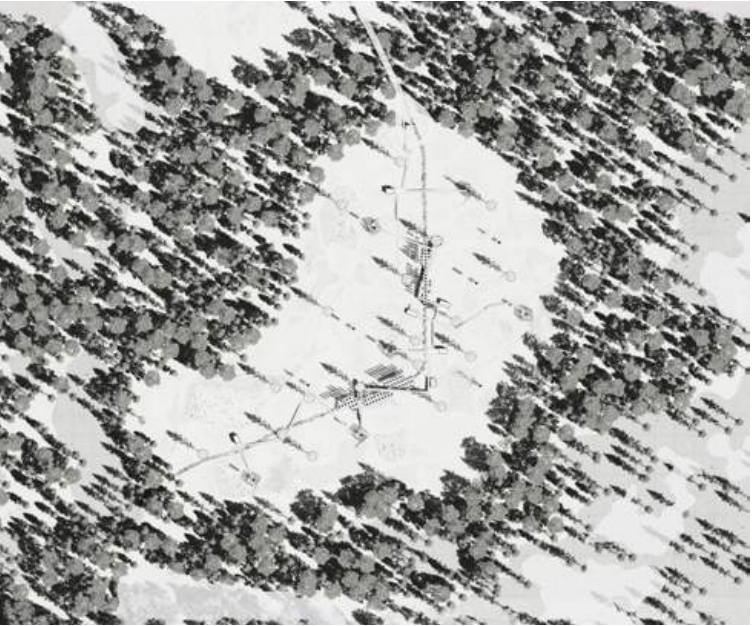
Active mesh superimposition: How the system mediates light
Weather balloons temporarily lift anchored meshes that attenuate the harsh light that reaches the forest's floor. Remediating the growth of the smallest of organisms in the boreal forest- which are indispensable to the health of the ecosystem.

1 custom weather balloon (1 yr) 2 pvc wire mesh 3 rock anchoring 4 substrate path





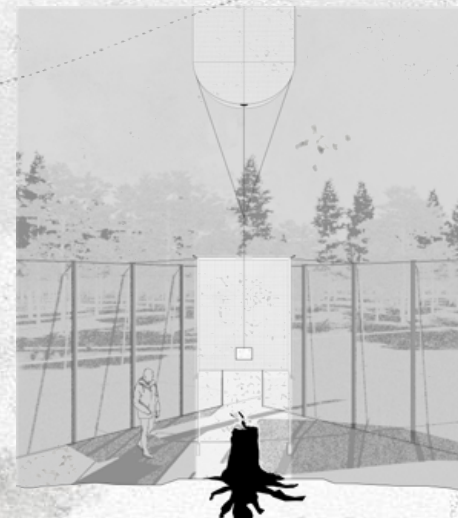
-  visitor entrance
-  substrates path
-  permanent archive
-  outpost



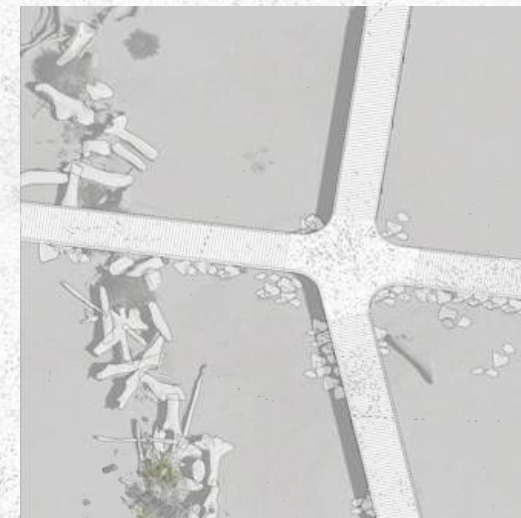
Occupying a 5 kilometer radius, the project begins at the farthest sites moving north as trees and lichen begin to reoccupy the canopy openings.



The park's path weaves itself through the substrates' path, reaching both the permanent archives and temporary outposts. The balloon-held meshes are in active use for the primary stages of the project, providing attenuated light to the redistributed substrates below.



Outpost



Boardwalk



Permanent archive: storing lichen that grows

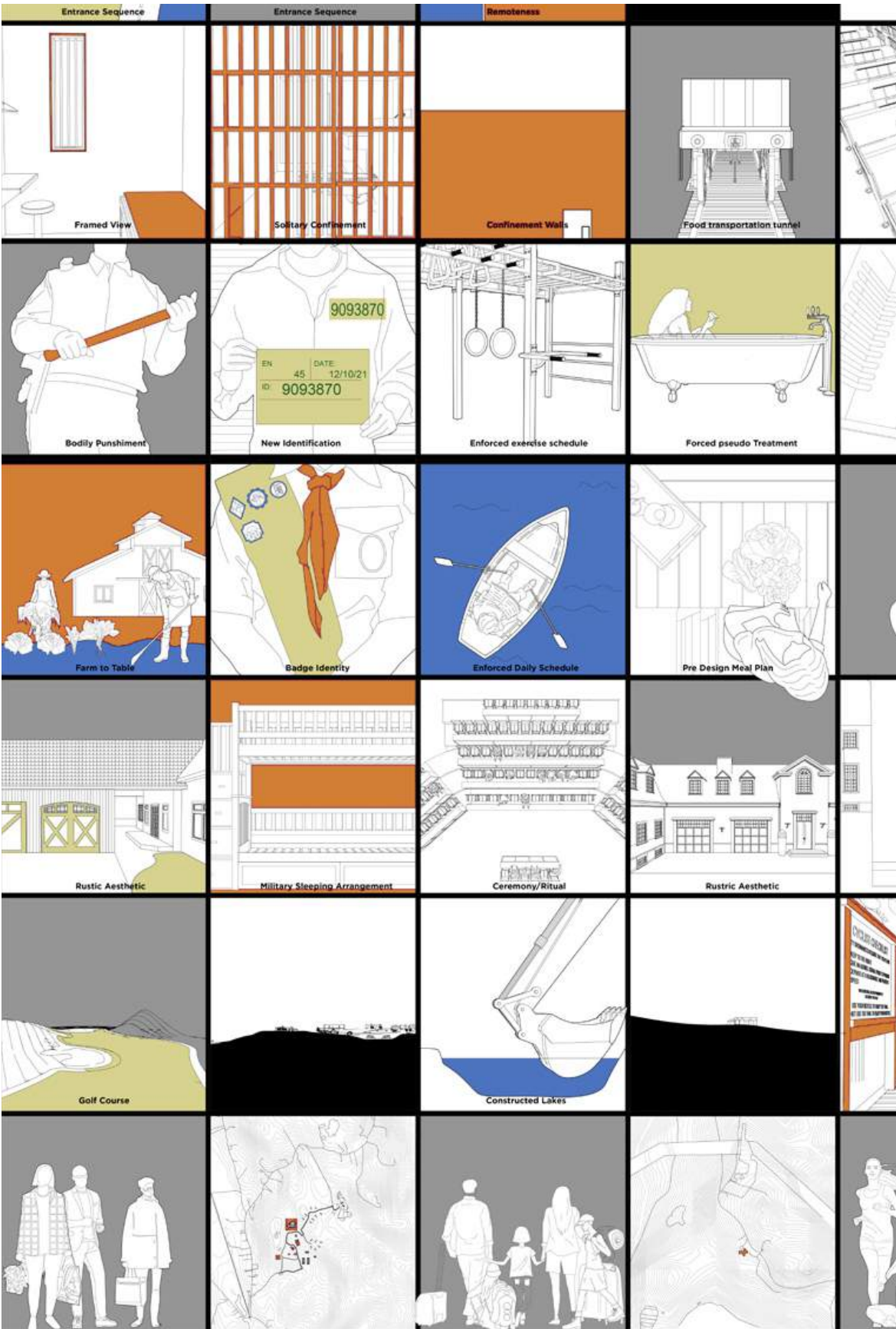
05 A Deterritorialized Sanctuary

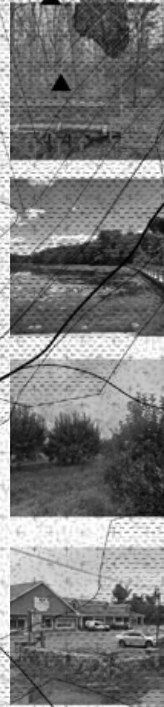
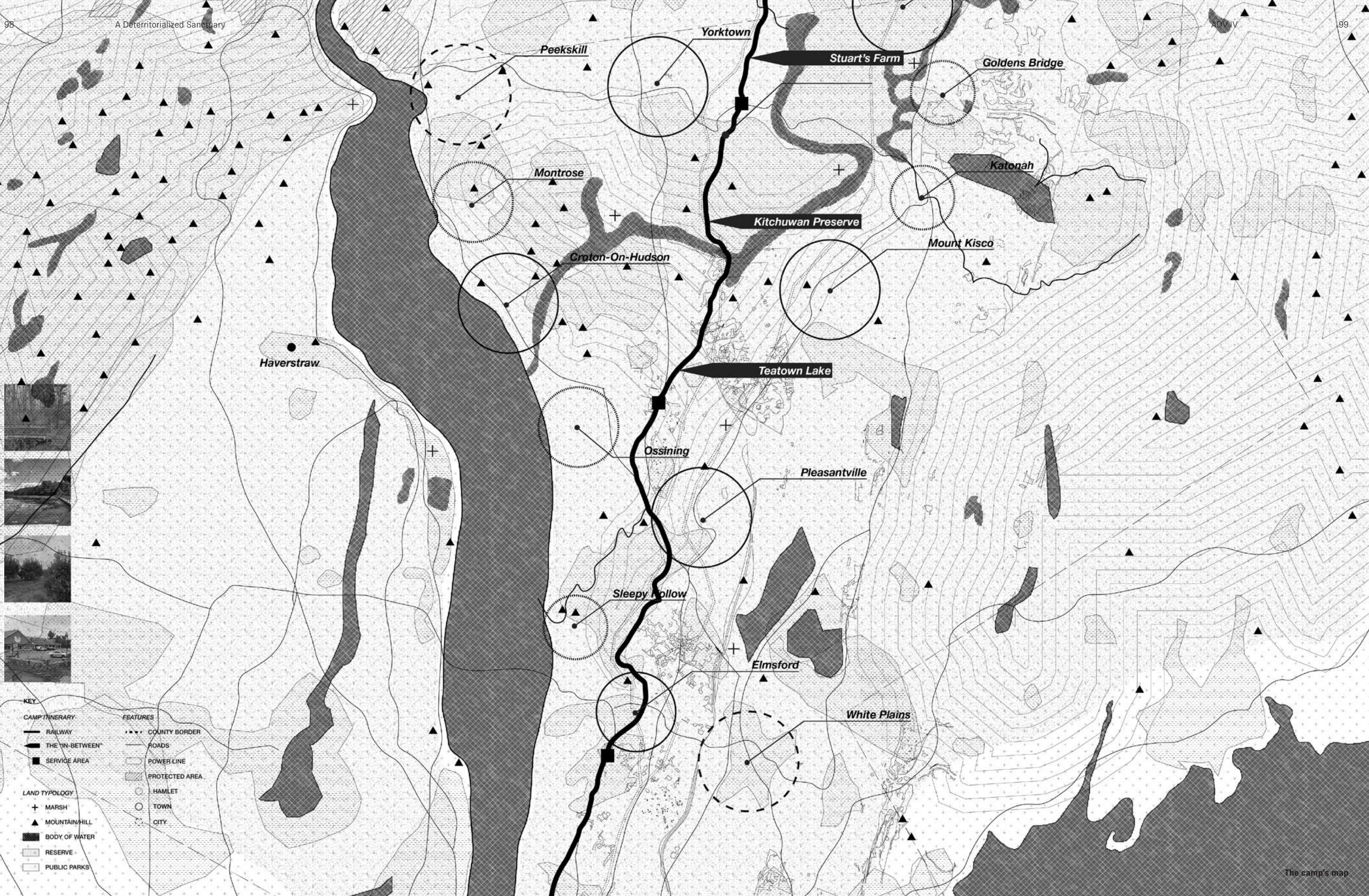
Spring 2025
ADV IV Studio
Instructor: Nahyun Hwang
Collaborative Work with Annie An

Site: Hudson Valley, NY

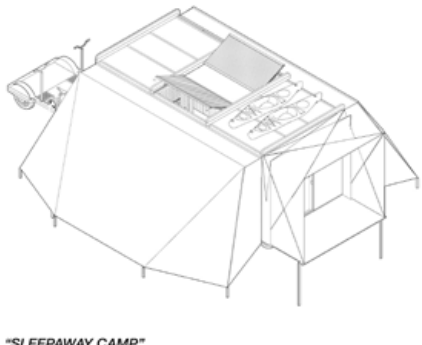
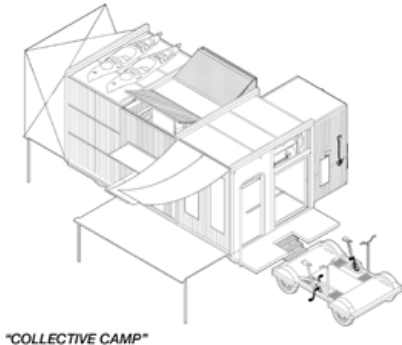
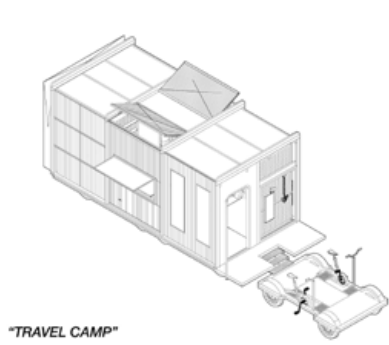


A mobile summer camp and site-responsive sanctuary that navigates the landscapes of the Hudson Valley by rail. The design critiques the **curated wilderness of franchised camps** and proposes a **mobile scout camp** that reclaims access to nature through deterritorialized occupation. **Repurposing abandoned railway infrastructure**, the system operates outside zoning constraints, creating temporary, educational, and community-building encounters across terrains ranging from quarries and marshes to hamlets and farms. The architecture is housed in a modular rail-borne carriage pulled by an electric bike, outfitted with tools for cooking, sleeping, foraging, and gathering. **Camp routines** - storytelling, tent pitching, communal meals - unfold not on a master-planned site but along a **customizable itinerary**. Plug-in nodes for energy and sanitation support temporary anchoring, while interactions with local food systems and trailheads foster reciprocal engagement. The project resists static ideas of retreat, offering instead an adaptive, **site-less architecture that critiques the commodification of nature and redefines sanctuary as movement across spaces of exception.**

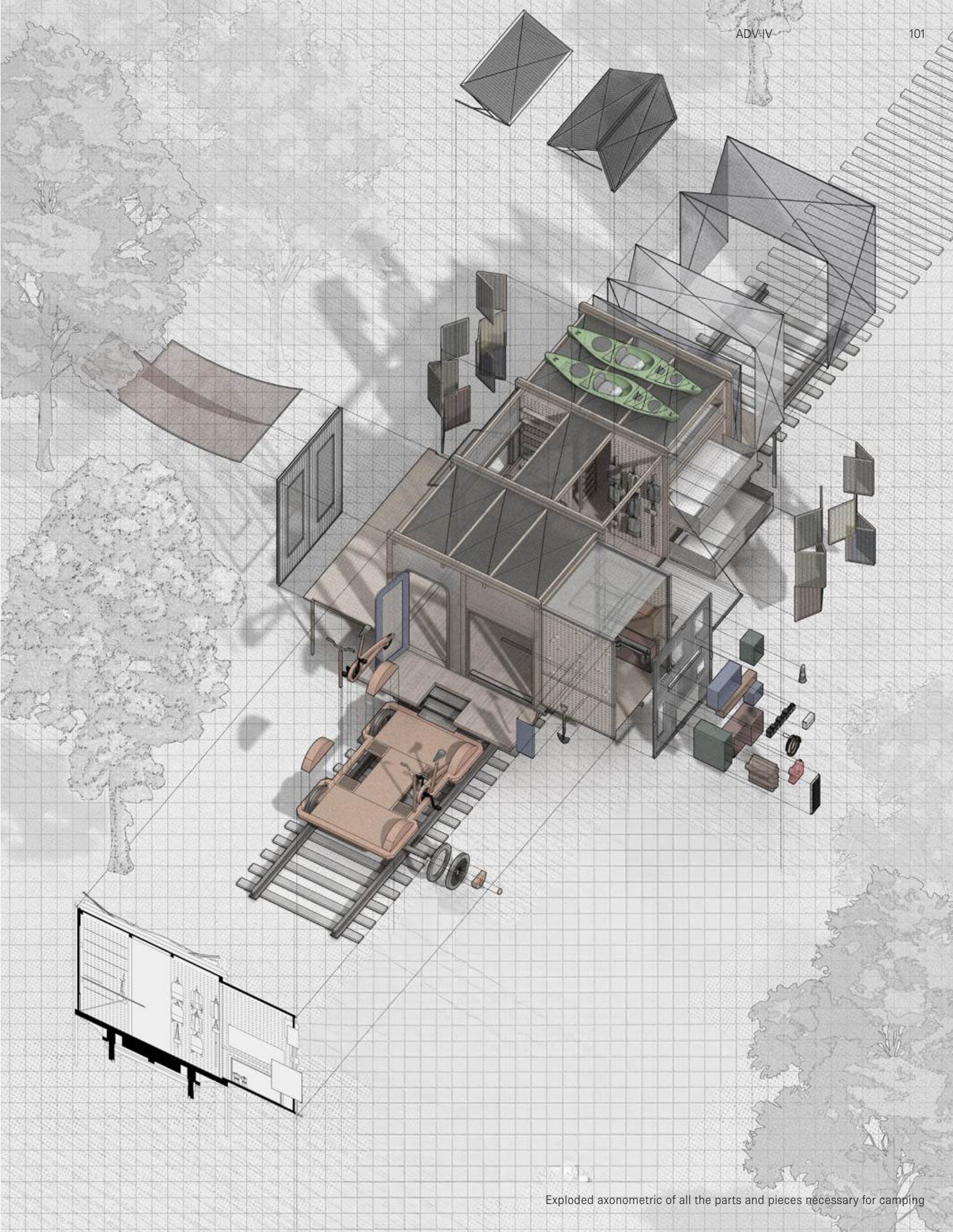




- KEY**
- CAMP ITINERARY**
- RAILWAY
 - THE "IN-BETWEEN"
 - SERVICE AREA
- LAND TYPOLOGY**
- MARSH
 - MOUNTAIN/HILL
 - BODY OF WATER
 - RESERVE
 - PUBLIC PARKS
- FEATURES**
- COUNTY BORDER
 - ROADS
 - POWER LINE
 - PROTECTED AREA
 - HAMLET
 - TOWN
 - CITY



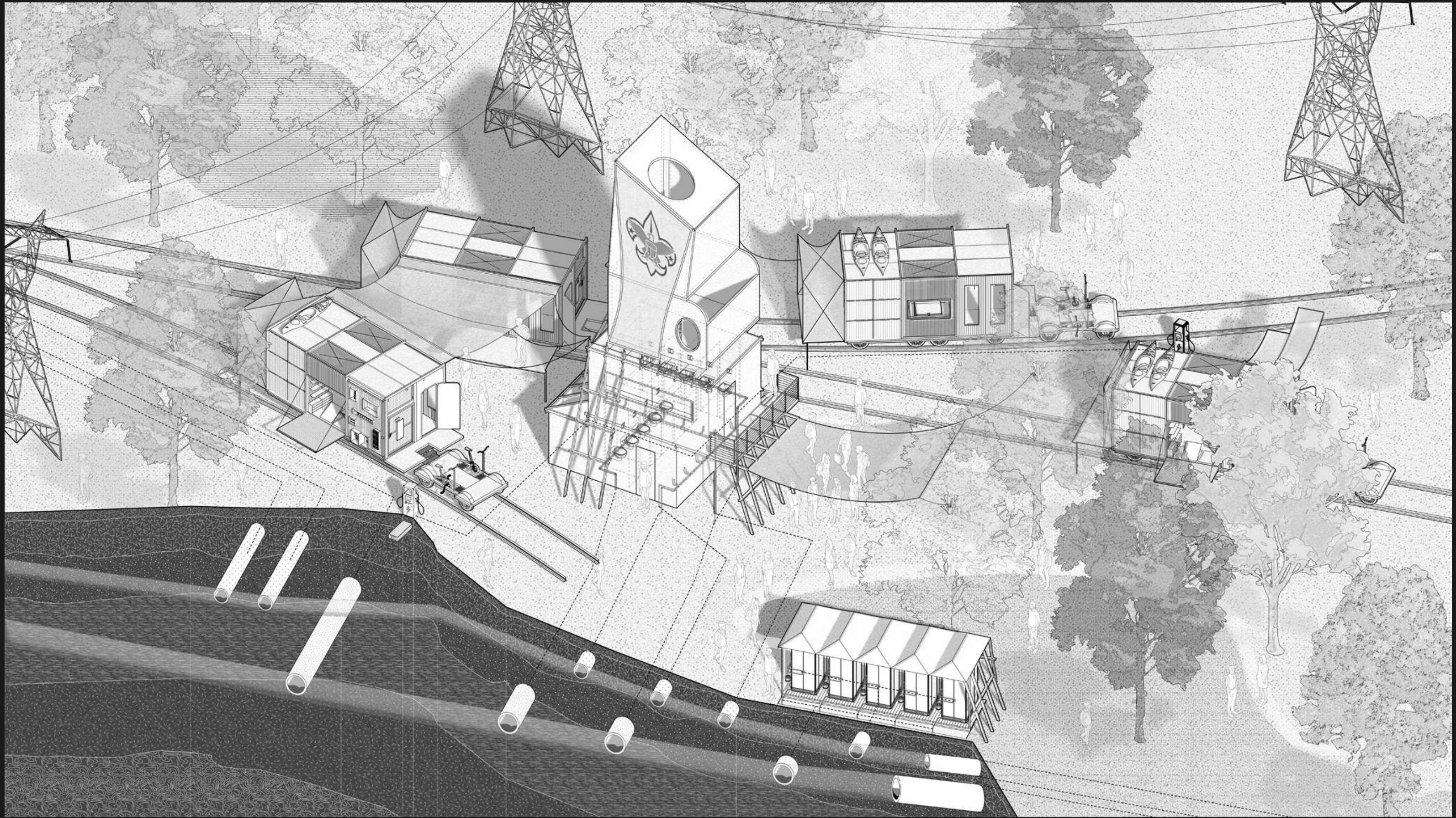
The Storyboard:
Mobility redefines the foundations of the scout camp, dissolving its dependence on fixed locations and reimagining it as a practice of **continuous movement**. These scenes trace a new spatial logic where routines like cooking, gathering, sleeping, and exploring unfold across a shifting terrain. Moments once confined to master-planned sites now emerge from the carriage itself, transforming the camp into an **adaptable system** that interacts with its surroundings on the move. Architecture becomes episodic: a bunk at night, a canoe launch by morning, a shared meal in transit. No longer an enclosed destination, **the camp becomes a mobile sanctuary**.



Exploded axonometric of all the parts and pieces necessary for camping



As the mobile camp navigates the Hudson Valley, it reveals the often obscured realities of land access. Moving through dense forests, quarries, and fenced-off properties, scouts encounter the boundaries - both physical and ideological - that define who is allowed to engage with "nature." This moment reframes the landscape not as untouched wilderness, but as a constructed, privatized territory shaped by ownership, extraction, and exclusion.





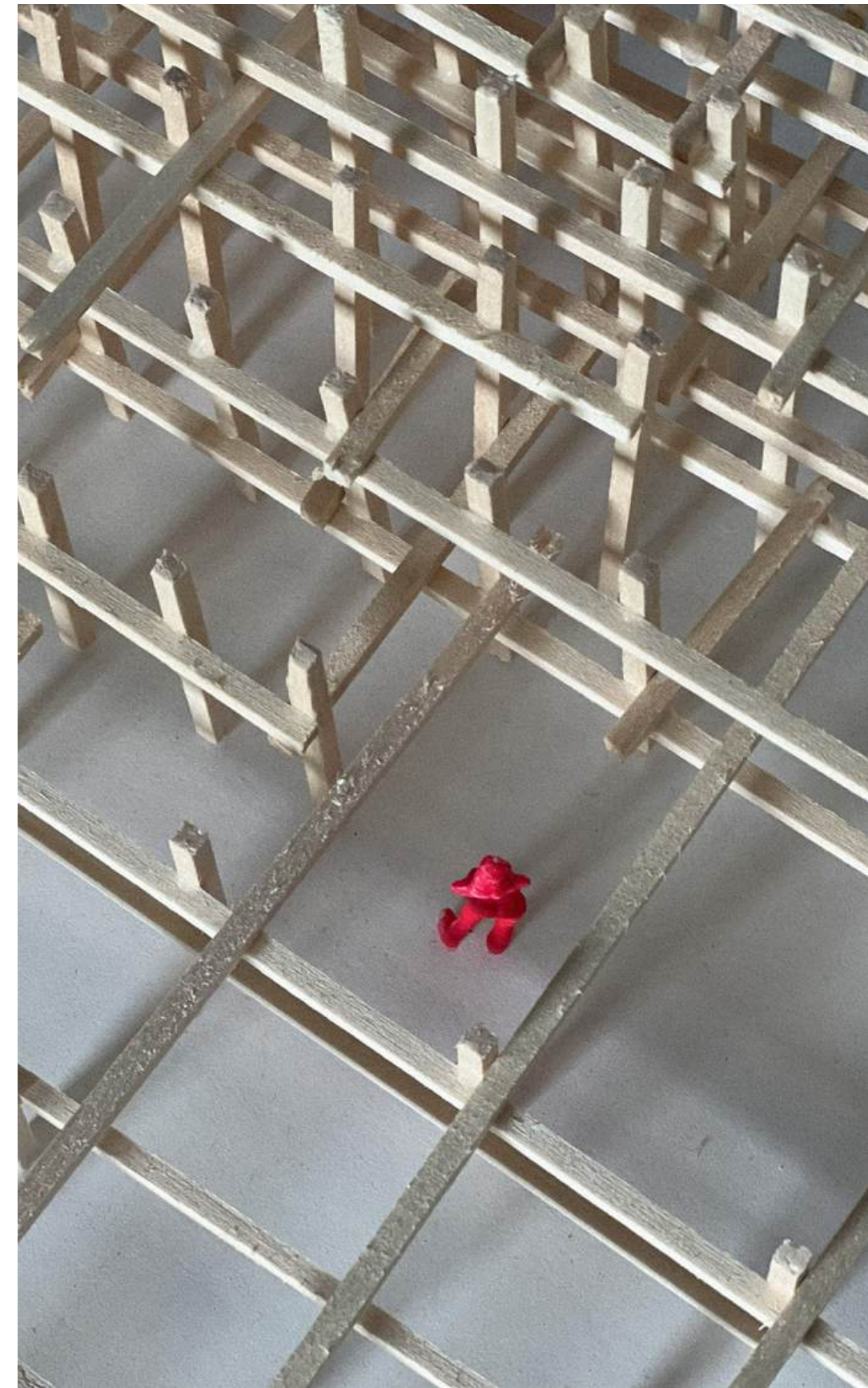
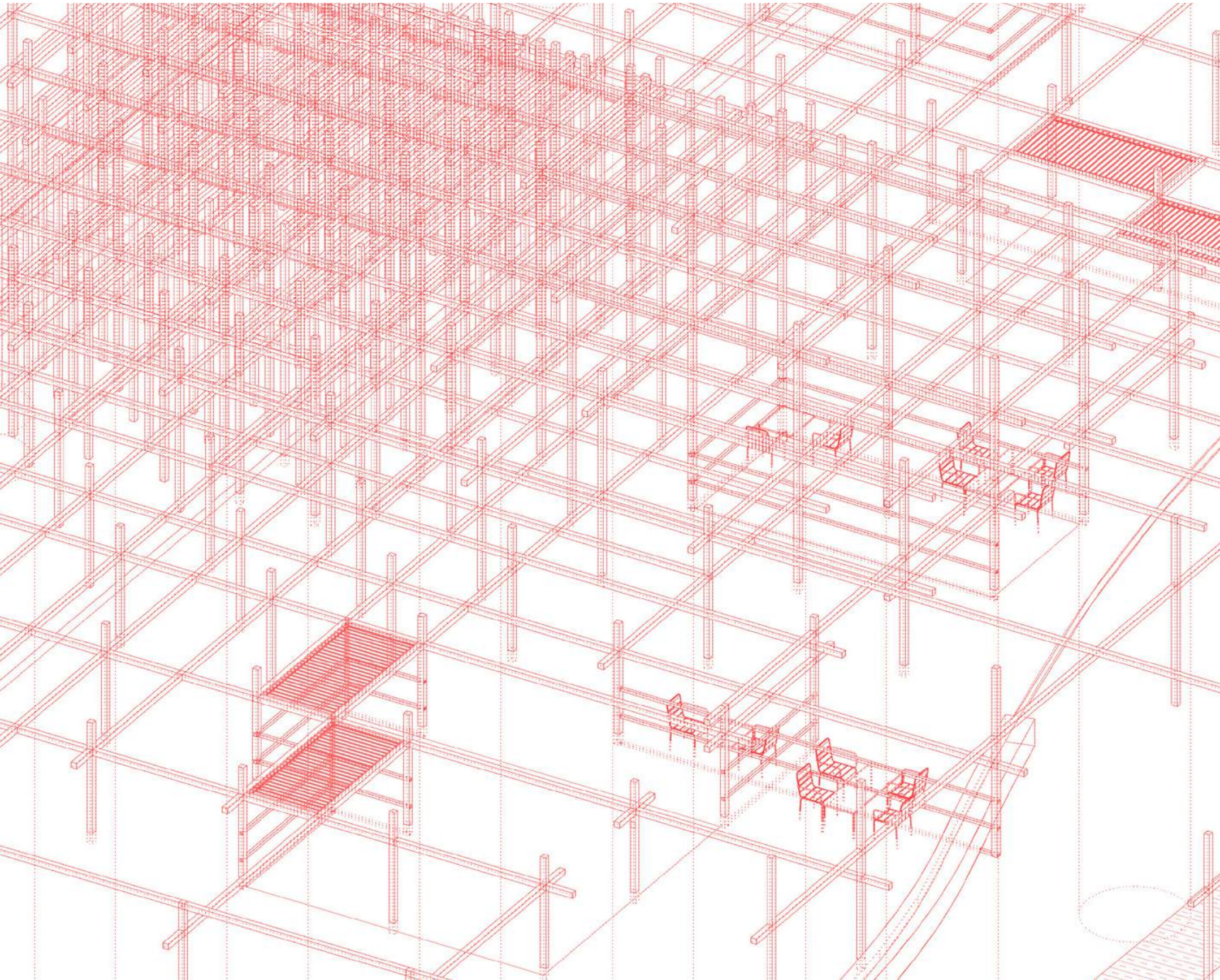
At designated stops, the camp opens up to accommodate communal cooking and eating, turning the carriage into an open-air kitchen and gathering space. Scouts engage in foraging, fishing, or visiting local farms, learning about food systems while preparing meals together. This moment becomes an educational exchange, reconnecting youth with the origins of food and embedding the camp within broader agricultural cycles.

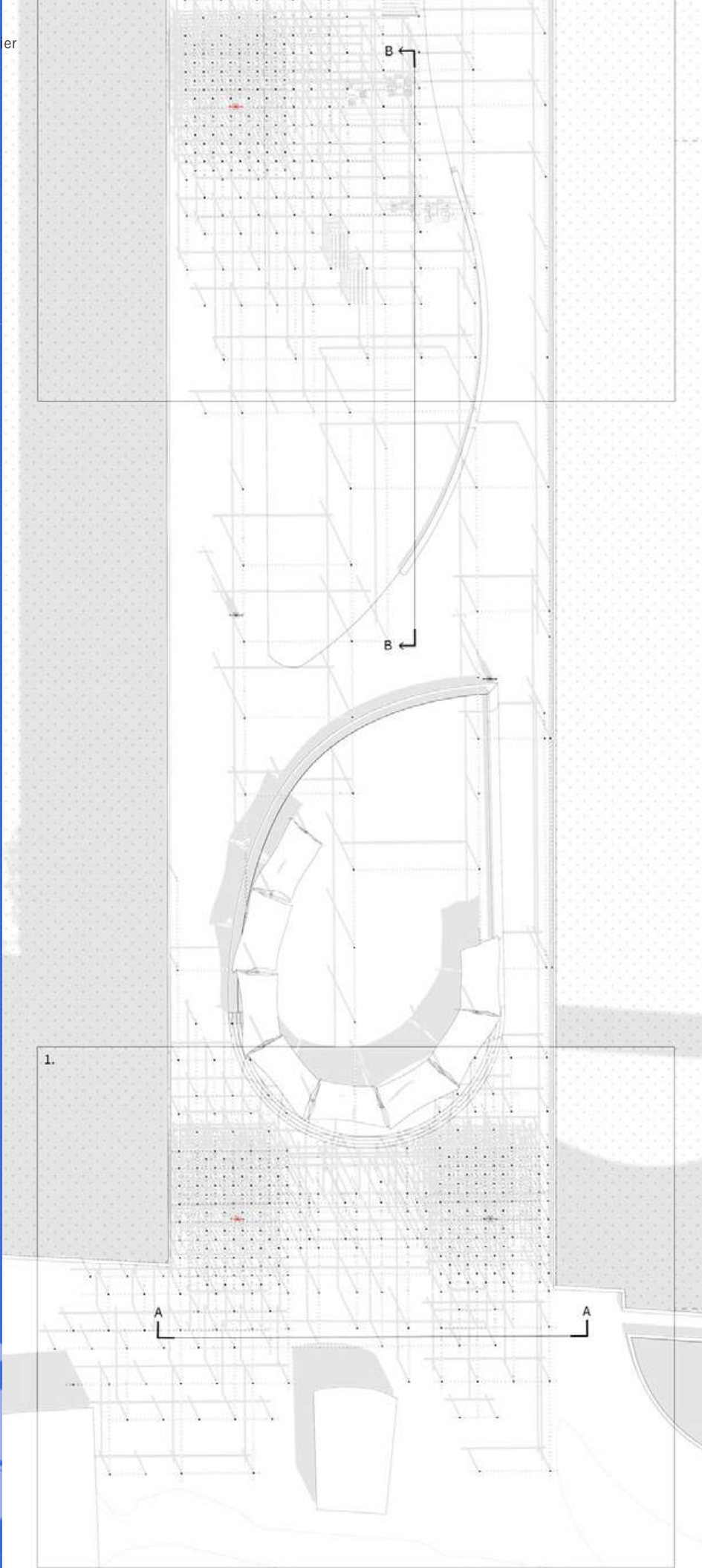
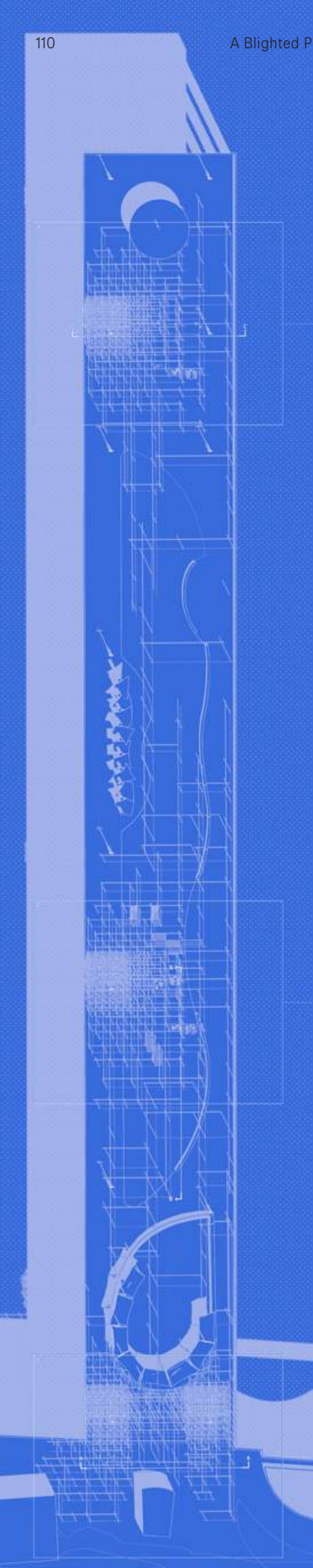
06 A Blighted Pier

Fall 2023
Core I Studio
Instructor: Kevin Hai Pham
Individual Work

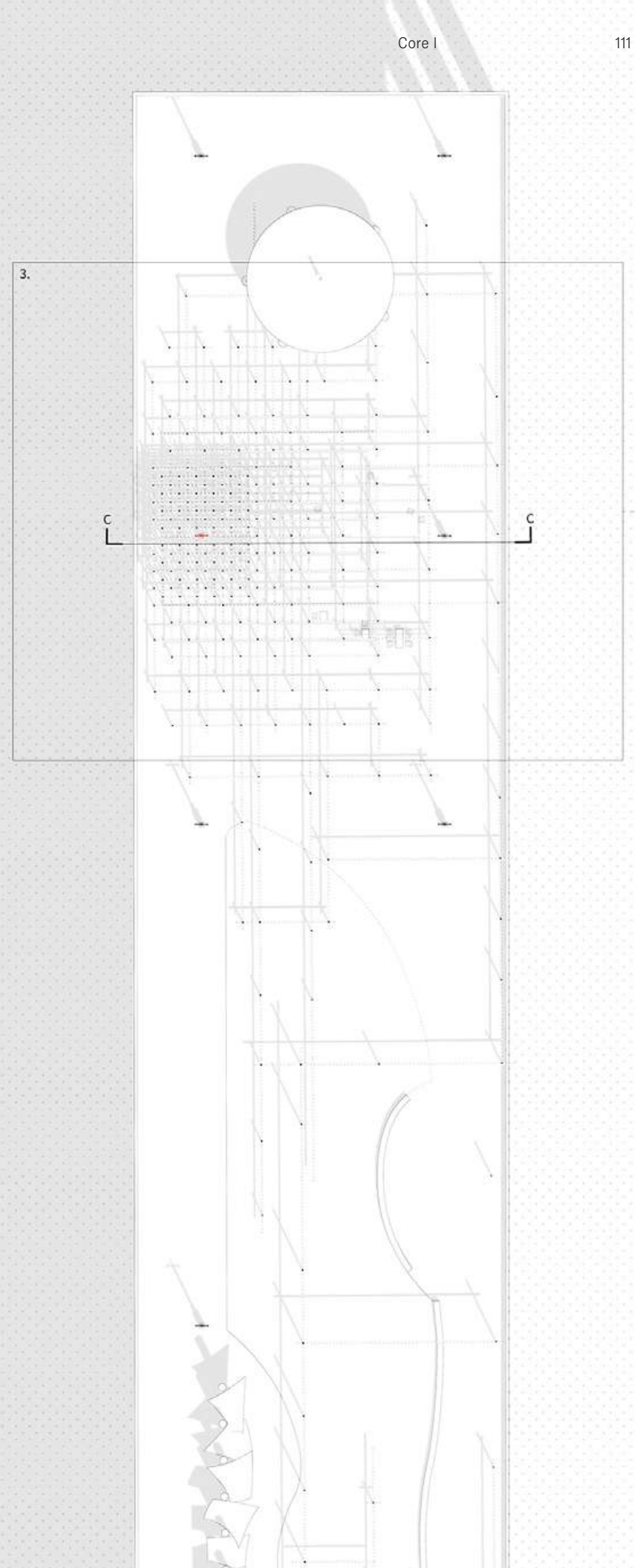
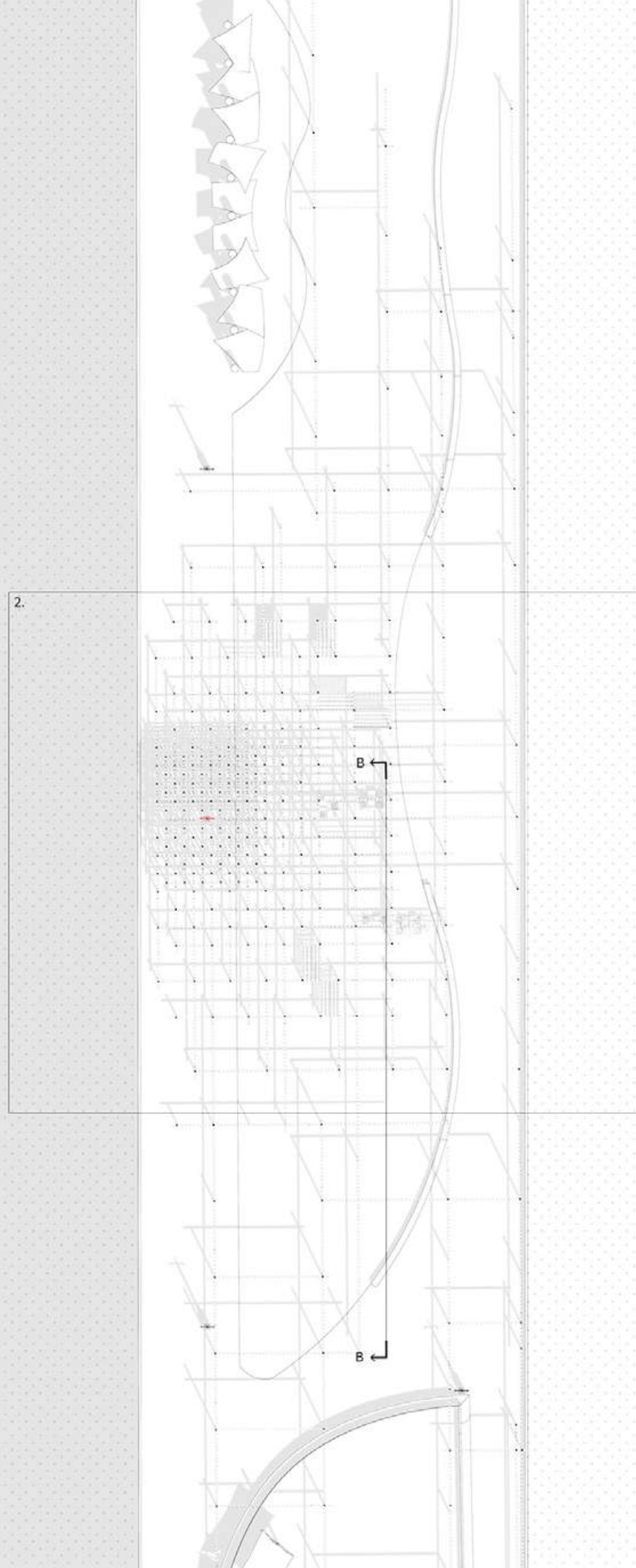
Site: New York, NY, USA

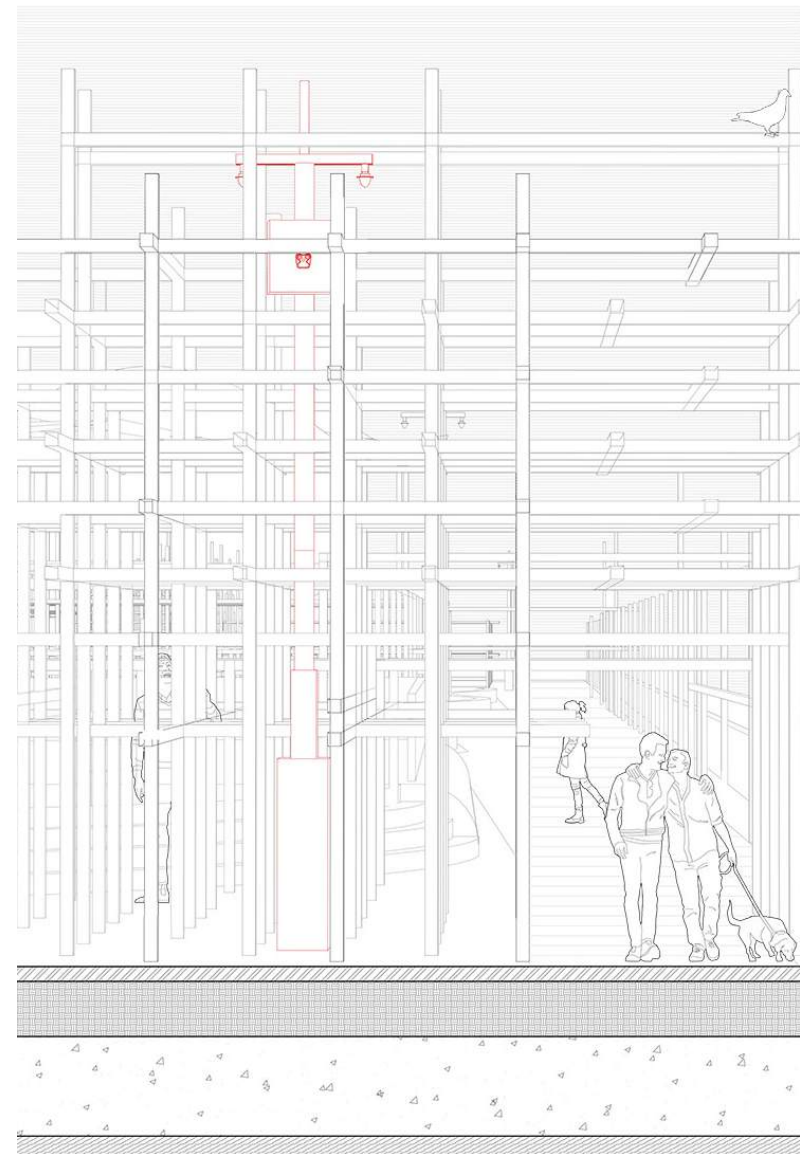
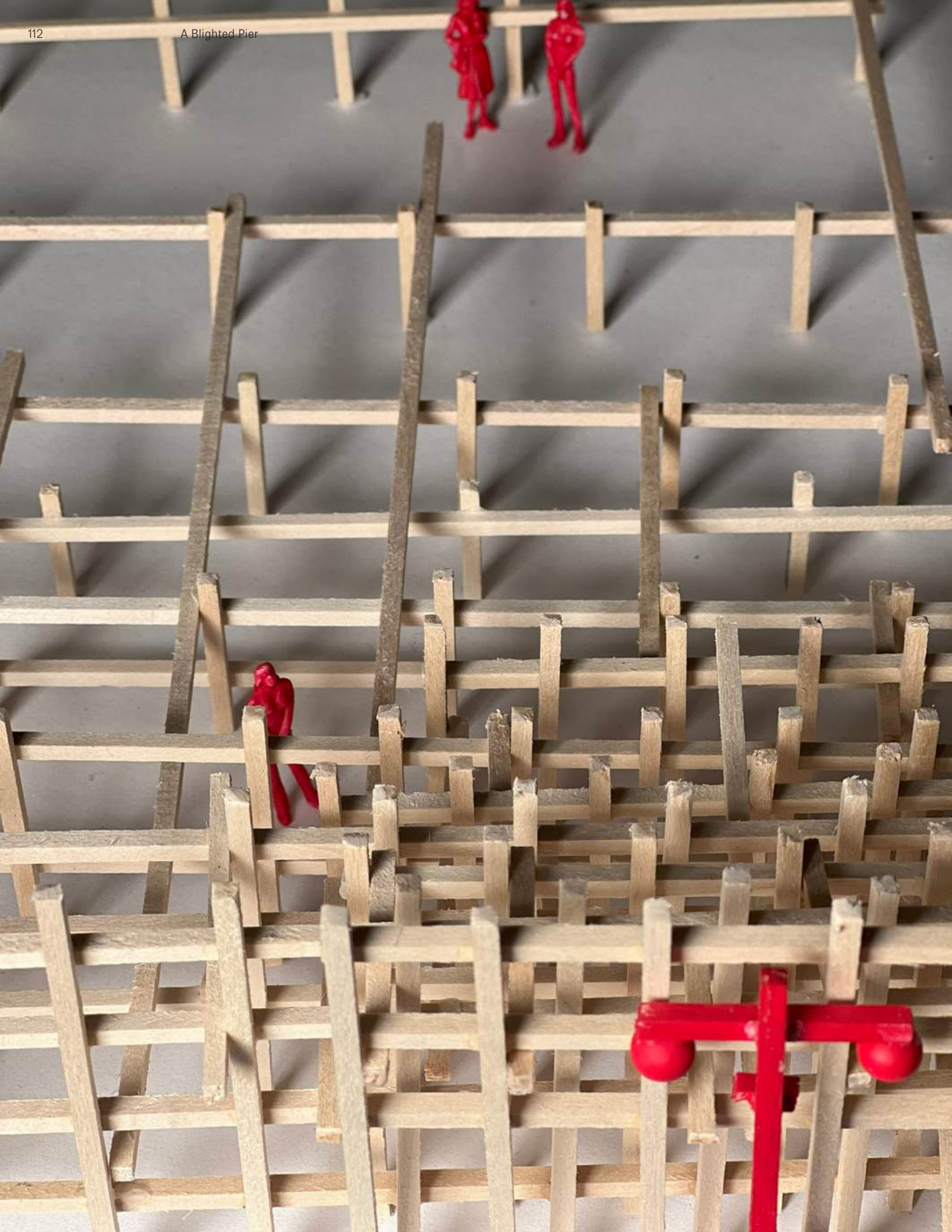
This project proposes a **cluster of membranes intended to obfuscate** the Christopher Street Pier (Pier 45) landscape from CCTV cameras. By extrapolating the 1980s history of the displacement of marginalized communities, the project contends with a **long history of gentrification and overexposure**. With the Hudson River Park Act, the pier, once known for its celebration of diversity and freedom of expression, became a representation of mundane and lightweight design. By **overflowing the pier** with aluminum columns, the project speaks to the **friction between the anonymity** of the current highly surveilled landscape and the **once dynamic and personalized environment**. Without using solid architectural elements, such as walls and roofs, the project **superimposes itself** on the current pier to reintroduce the areas of privacy that once encouraged escape and freedom of expression.

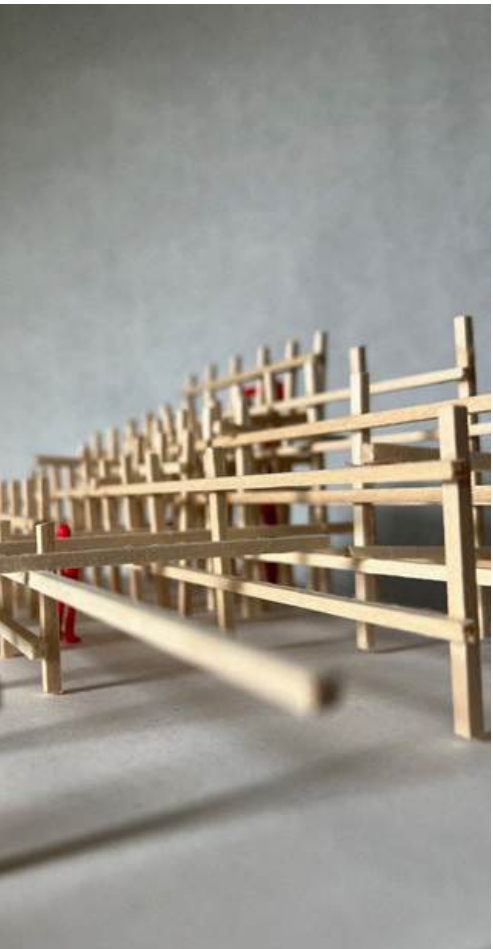
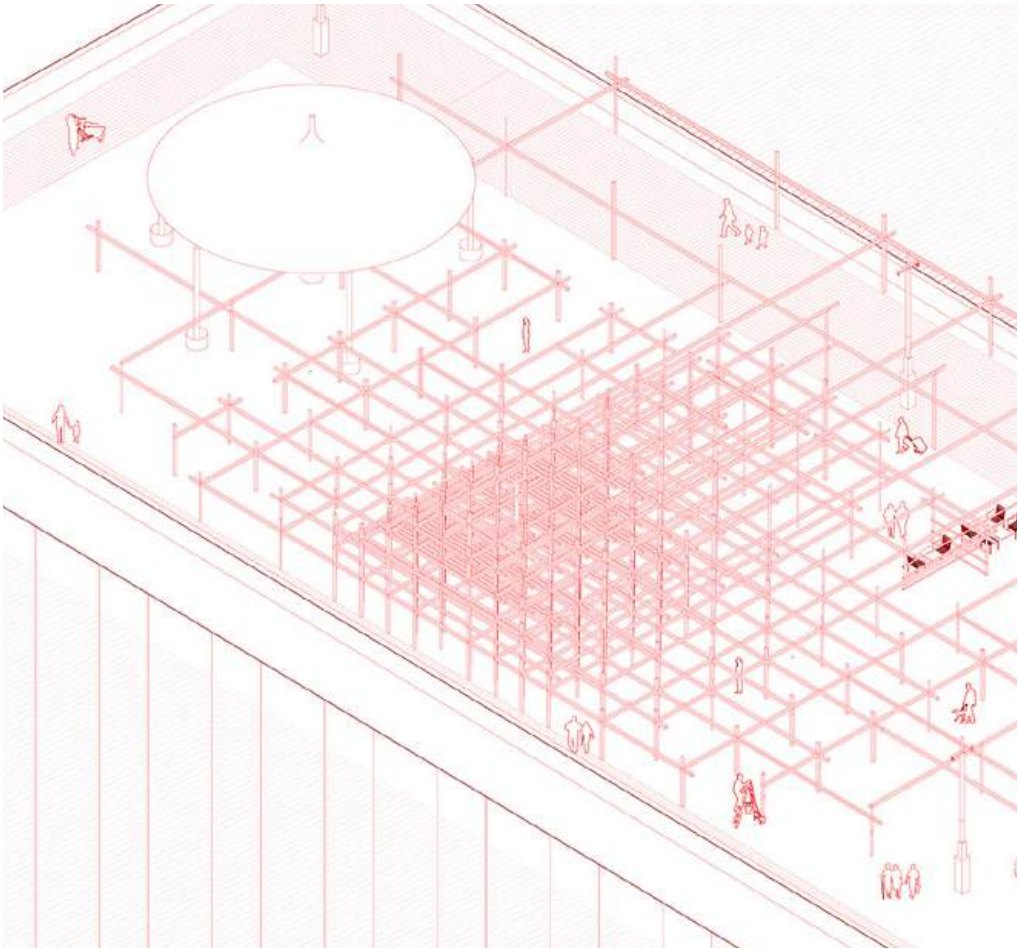
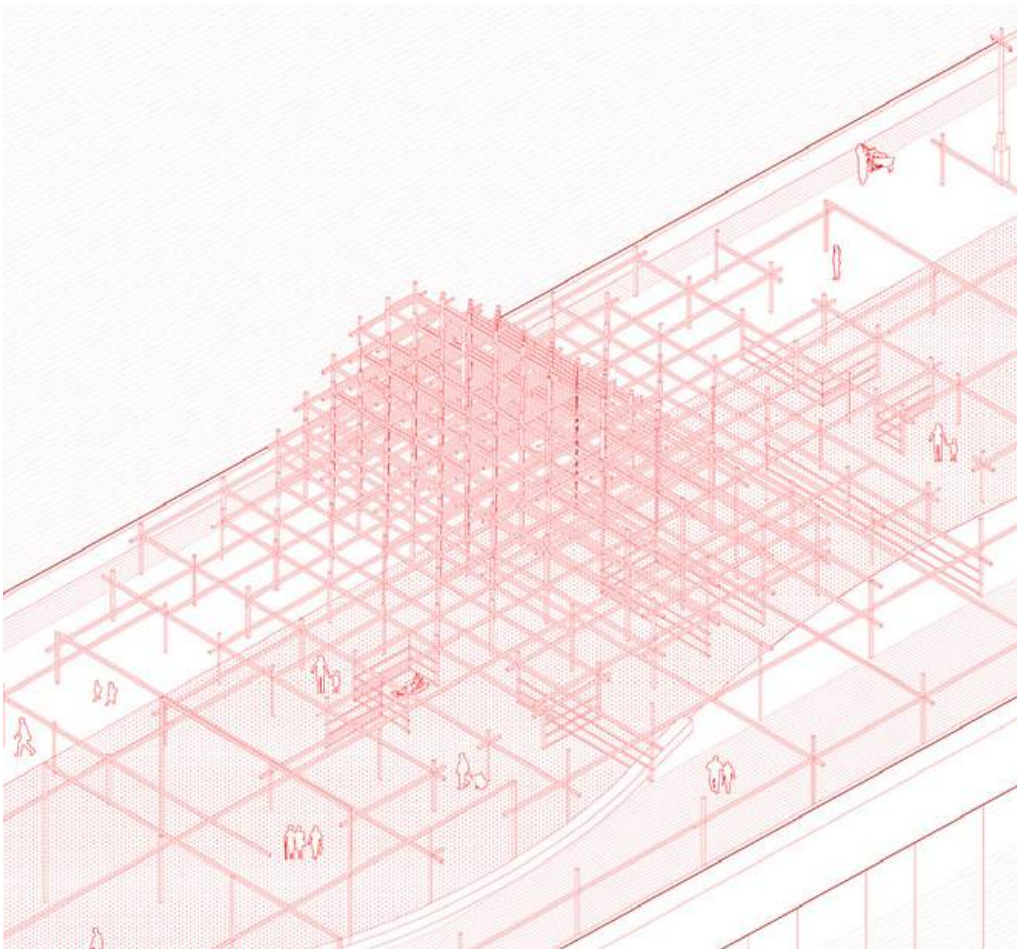
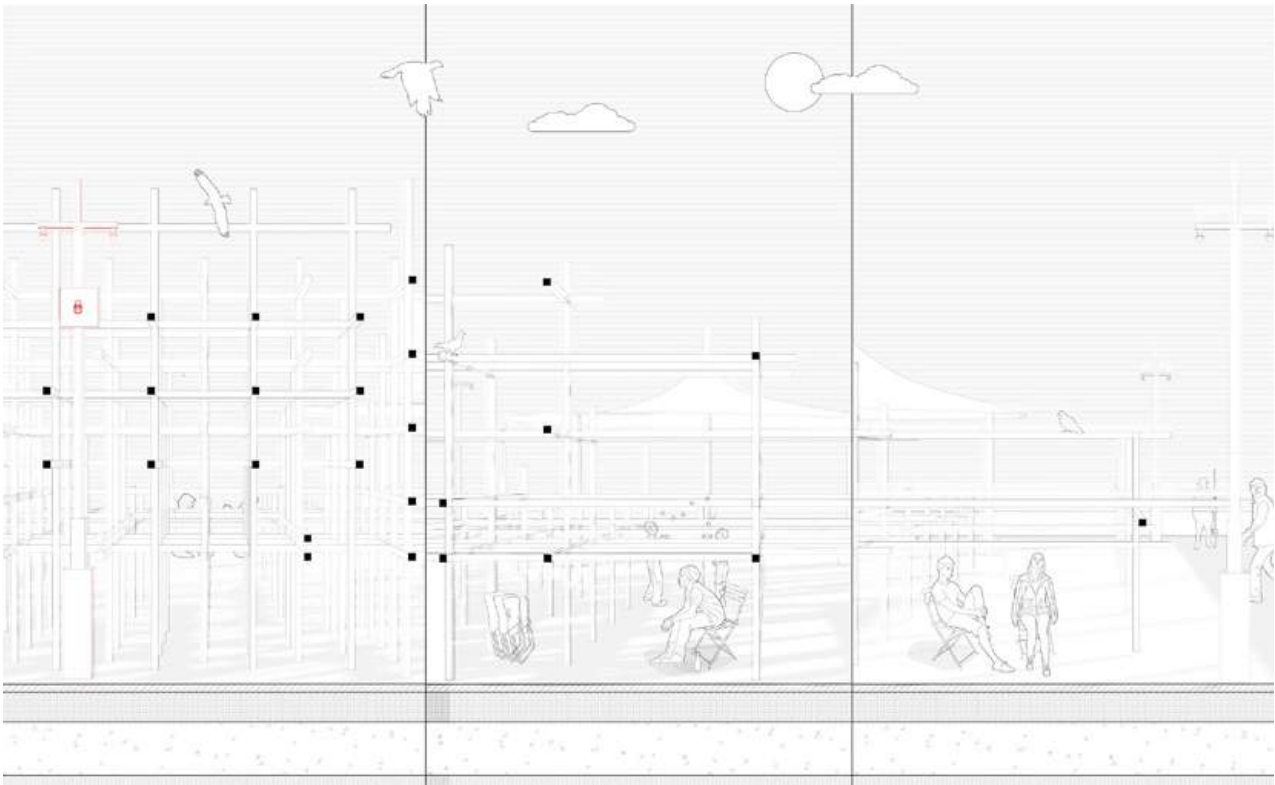
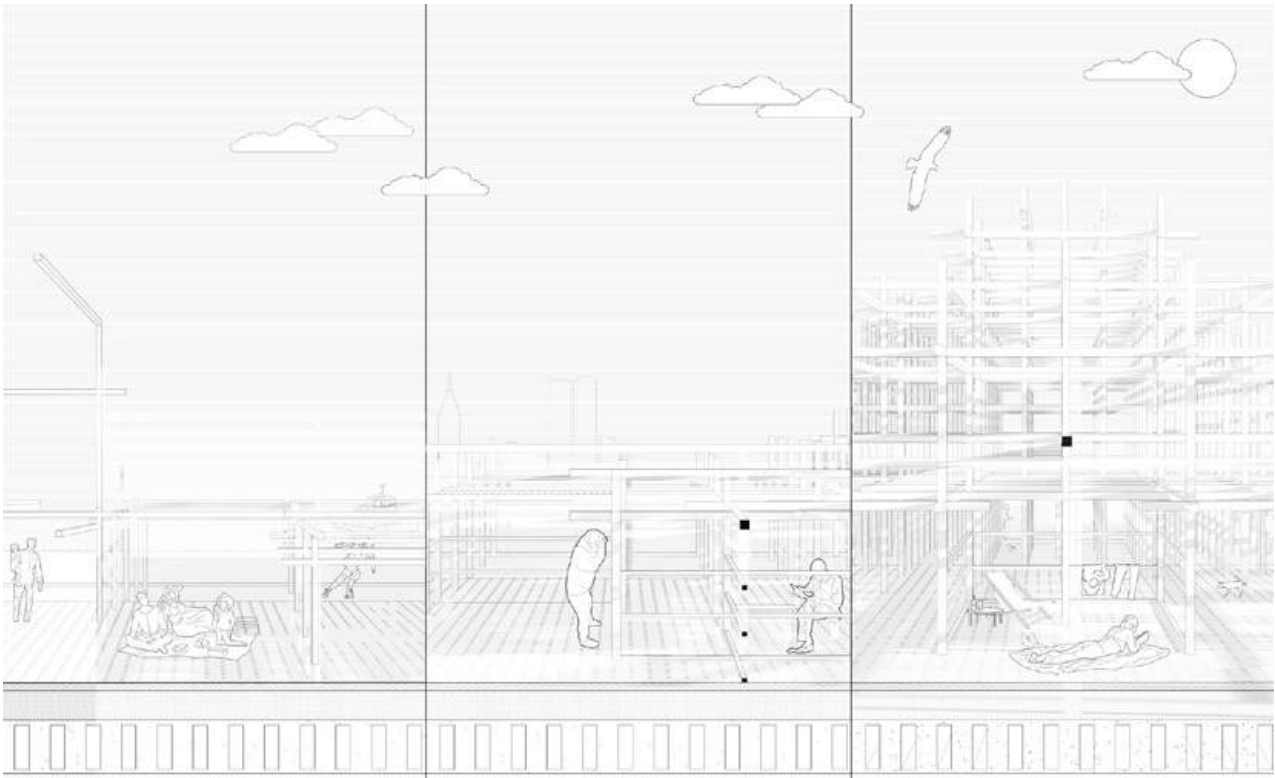


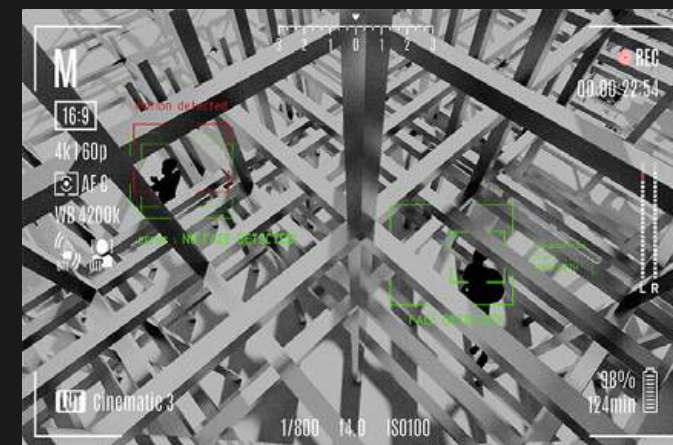
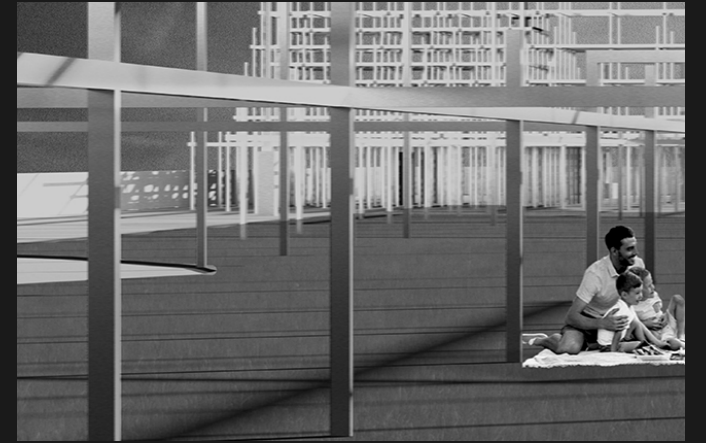
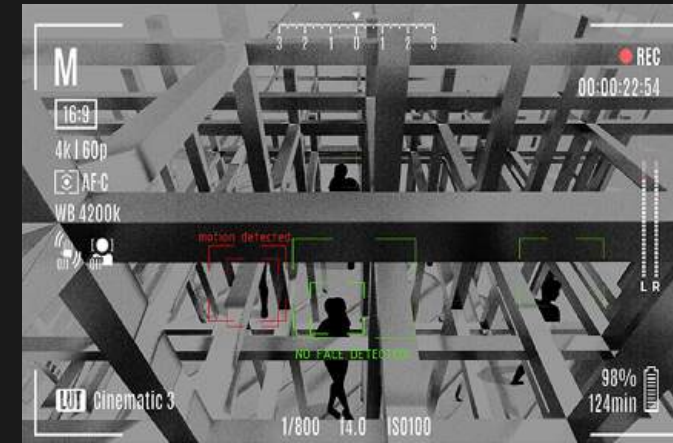
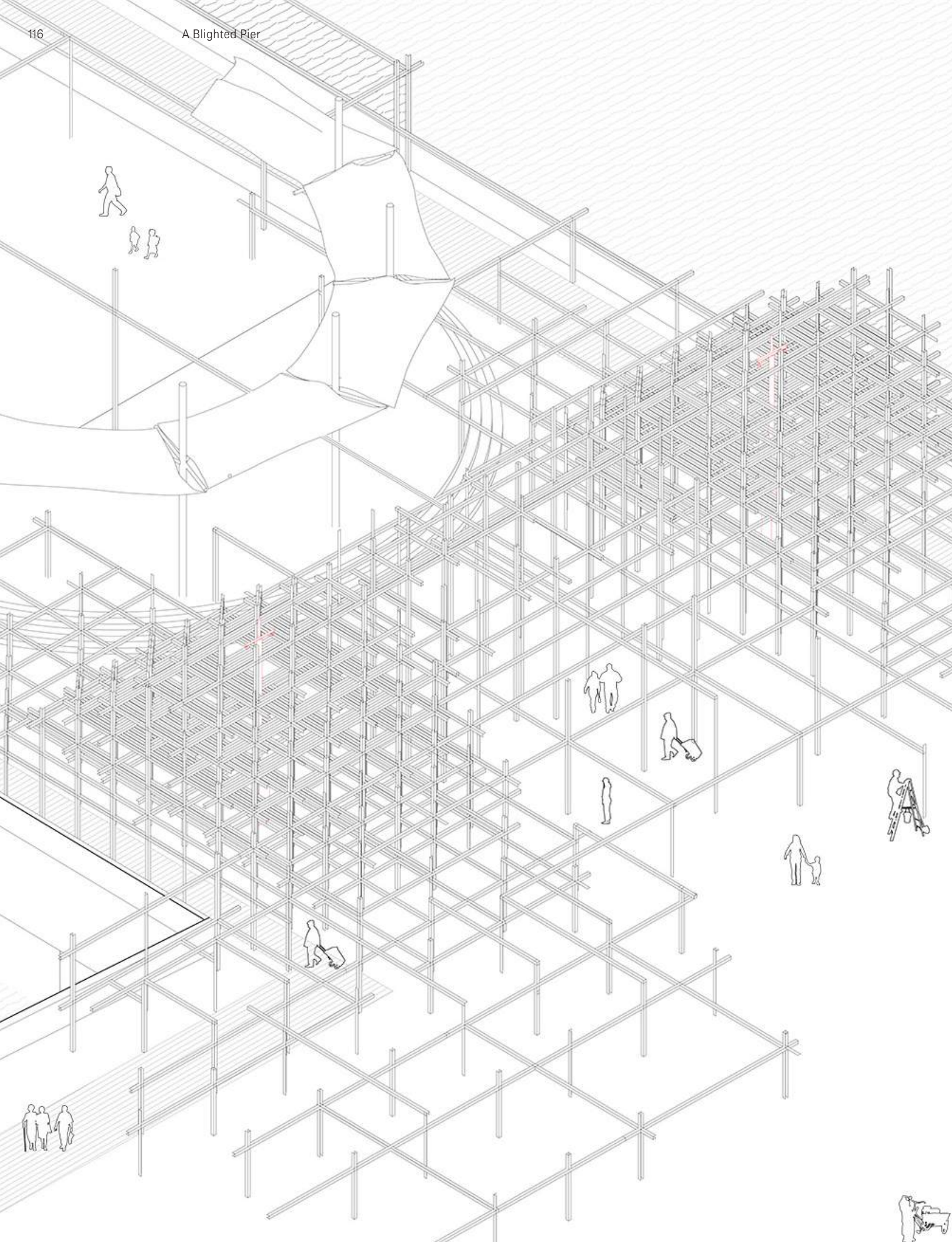


2.





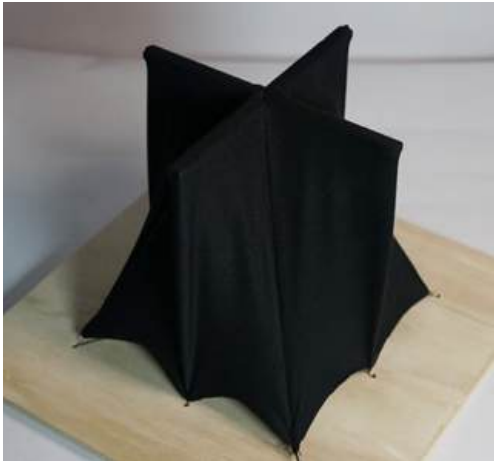




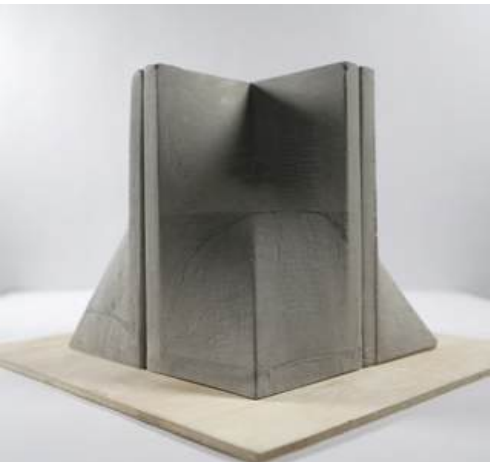
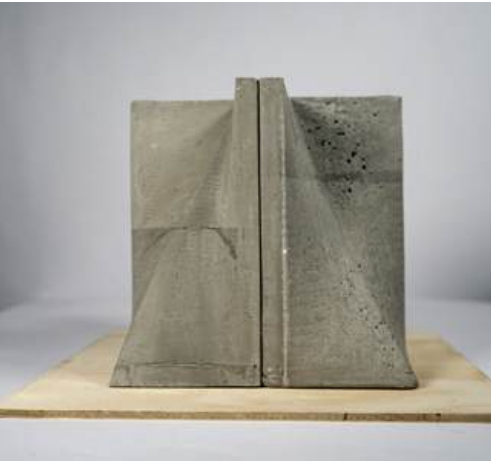
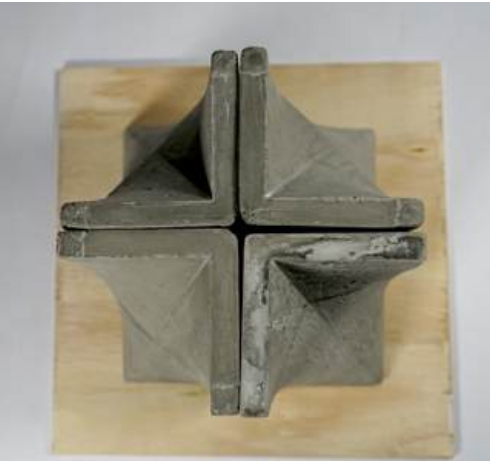
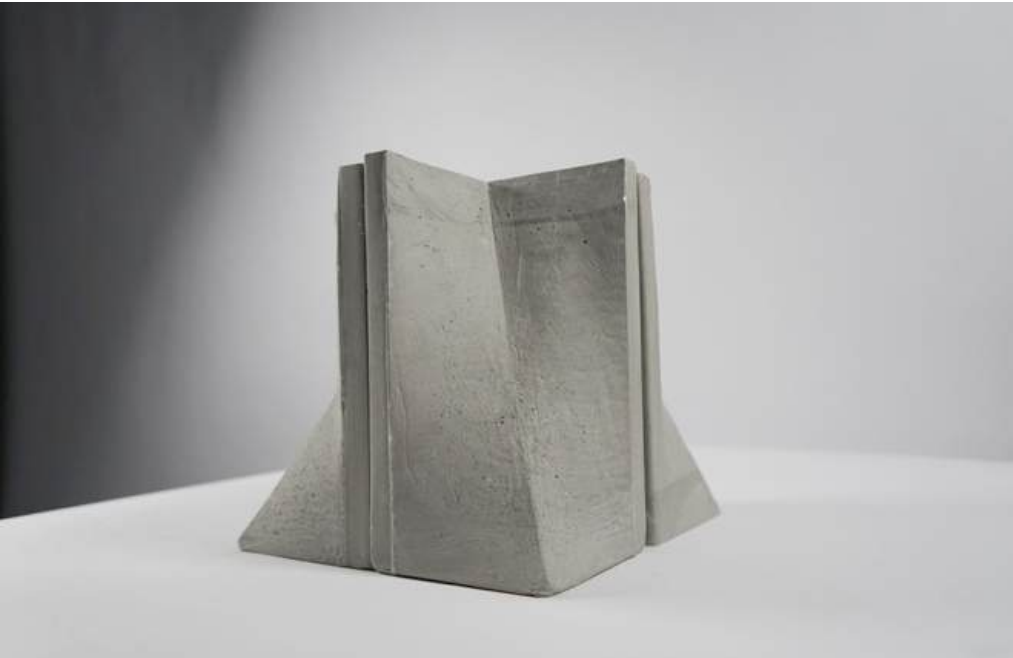
The camera is unable to fully identify the users of the pier as the the superimposition of columns obfuscates the camera's view and offers a new level of privacy.

07 Doubly Ruled

Fall 2025
Tensile/Compression Surfaces
Instructor: Robert Marino
Collaborative Work with Zachary Poncher



Study of doubly ruled surfaces through tensile formwork and rockite casting. Investigating **curvature, structural continuity, and thin-shell** behavior through iterative physical modeling inspired by Pier Luigi Nervi's Cathedral of Saint Mary of the Assumption in San Francisco. By translating fabric tension into rigid concrete form, the model studies how curvature can simultaneously generate structure, enclosure, and **spatial expression**.



08 A Stair Generator

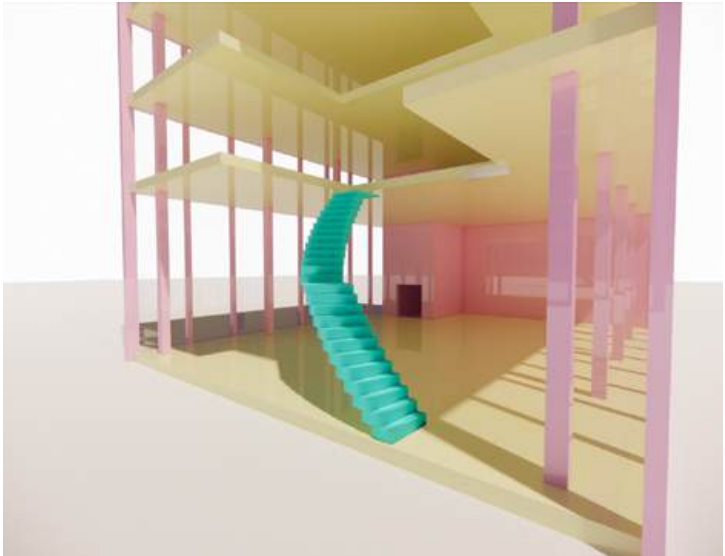
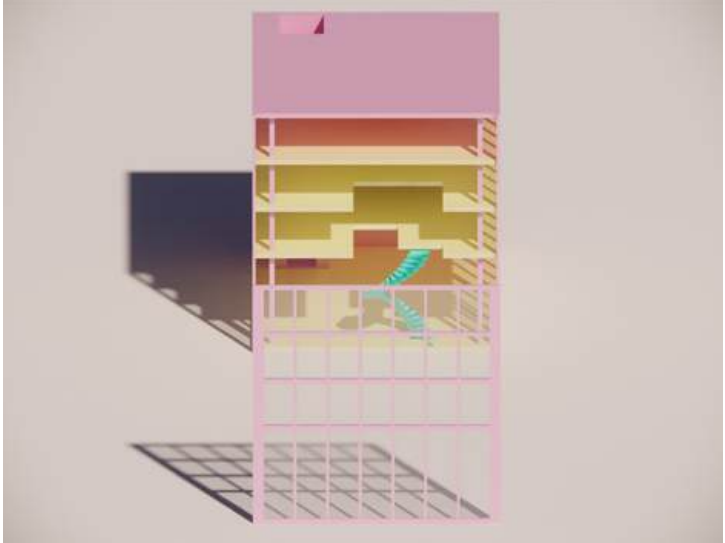
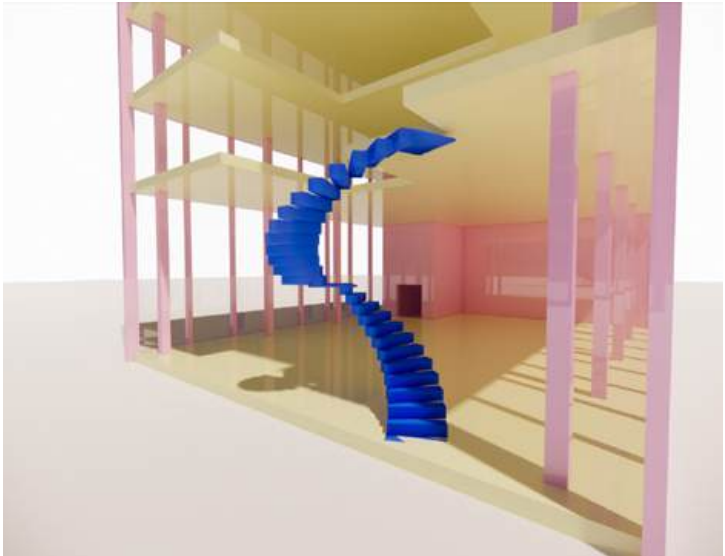
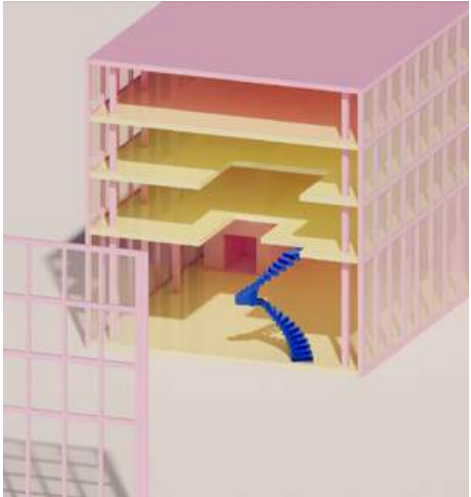
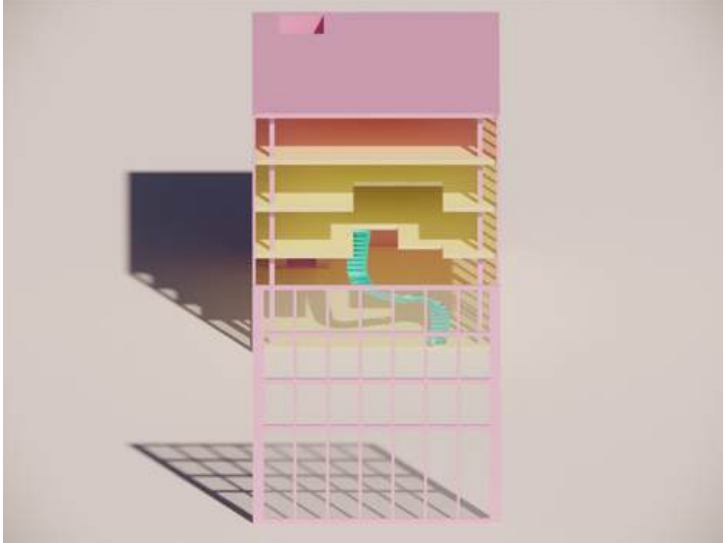
Spring 2025
Generative Design
Instructor: Danil Nasy
Collaborative Work with Zachary Poncher and Joshua Allen



```
"""Staircase Generator with Precise Path Control - Ensuring Exact  
Start/End Points and Path Shape"""  
import Rhino.Geometry as rg  
import math  
import random  
  
def generate_stairs(StartPoint, EndPoint, TargetRiserHeight, TreadDepth, StairWidth):  
    """Main function to generate stairs with paths that precisely follow a given path curve"""  
    try:  
        # Result list  
        result = []  
  
        # Basic calculations  
        total_rise = EndPoint.Z - StartPoint.Z  
        num_steps = max(3, int(round(total_rise / TargetRiserHeight)))  
        actual_riser_height = total_rise / num_steps  
  
        # Generate or validate path curve  
        if PathCurve is None or not PathCurve.IsValid:  
            # Generate path based on OptionIndex with precise start and end points  
            PathCurve = generate_precise_path(StartPoint, EndPoint, option_index)  
        else:  
            # Verify and adjust the provided curve to exactly match start and end points  
            PathCurve = adjust_curve_endpoints(PathCurve, StartPoint, EndPoint)  
  
        # Add path to result for visualization  
        result.append(PathCurve)  
  
        # Create 2D version of the path for stairs layout  
        path_2d = project_curve_to_xy(PathCurve, StartPoint.Z)  
  
        # Determine if we need landings  
        if MaxStepsPerFlight <= 0:  
            MaxStepsPerFlight = num_steps # No landings  
  
        needs_landings = num_steps > MaxStepsPerFlight  
  
        if not needs_landings:  
            # Simple staircase without landings  
            create_staircase_without_landings(  
                result, path_2d, StartPoint, EndPoint, num_steps,  
                actual_riser_height, TreadDepth, StairWidth  
            )  
        else:  
            # Staircase with landings  
            create_staircase_with_landings(  
                result, path_2d, StartPoint, EndPoint,  
                MaxStepsPerFlight, num_steps, actual_riser_height,  
                TreadDepth, StairWidth  
            )  
  
        return result  
  
    except Exception as e:  
        print(f"Error in generate_stairs: {e}")  
        return [rg.Line(StartPoint, EndPoint).ToNurbsCurve()]  
  
def generate_precise_path(start_point, end_point, option_index):  
    """Generate a path that precisely starts and ends at the given points with a specific shape"""  
    # Set consistent seed  
    random.seed(option_index + 42)  
  
    # Get XY coordinates (Z will be handled by stair generation)  
    start_2d = rg.Point2d(start_point.X, start_point.Y, 0)  
    end_2d = rg.Point2d(end_point.X, end_point.Y, 0)  
  
    # Calculate vector from start to end  
    dir_vector = rg.Vector2d(end_2d - start_2d)  
    dir_length = dir_vector.Length  
    dir_vector.Unitize()  
  
    # Get perpendicular vector for offsets  
    perp_vector = rg.Vector2d(-dir_vector.Y, dir_vector.X, 0)  
  
    # Choose one of 4 distinct path types  
    path_type = option_index % 4  
  
    try:  
        if path_type == 0:  
            # OPTION 0: STRAIGHT LINE  
            # Simple direct path from start to end  
            return rg.Line(start_2d, end_2d).ToNurbsCurve()  
  
        elif path_type == 1:  
            # OPTION 1: U-SHAPED PATH  
            # Create a dramatic U-shaped path with exactly 3 segments  
            # Get perpendicular offset (make it substantial)  
            offset = max(dir_length * 0.75, 5.0) # At least 5 units  
  
            # First leg - go perpendicular  
            corner1 = rg.Point2d(  
                start_2d.X + perp_vector.X * offset,  
                start_2d.Y + perp_vector.Y * offset,  
                0  
            )  
  
            # Second leg - move parallel to the original direct path  
            corner2 = rg.Point2d(  
                corner1.X + dir_vector.X * dir_length,  
                corner1.Y + dir_vector.Y * dir_length,  
                0  
            )  
  
            # Create U path with sharp corners - ensure it starts and ends exactly at the given points  
            polyline = rg.Polyline(  
                start_2d, # Exact start point  
                corner1, # Exact start point  
                corner2, # Exact end point  
                end_2d, # Exact end point  
            )  
            return polyline.ToNurbsCurve()  
  
    except Exception as e:  
        print(f"Error in generate_precise_path: {e}")  
        return [rg.Line(start_point, end_point).ToNurbsCurve()]
```

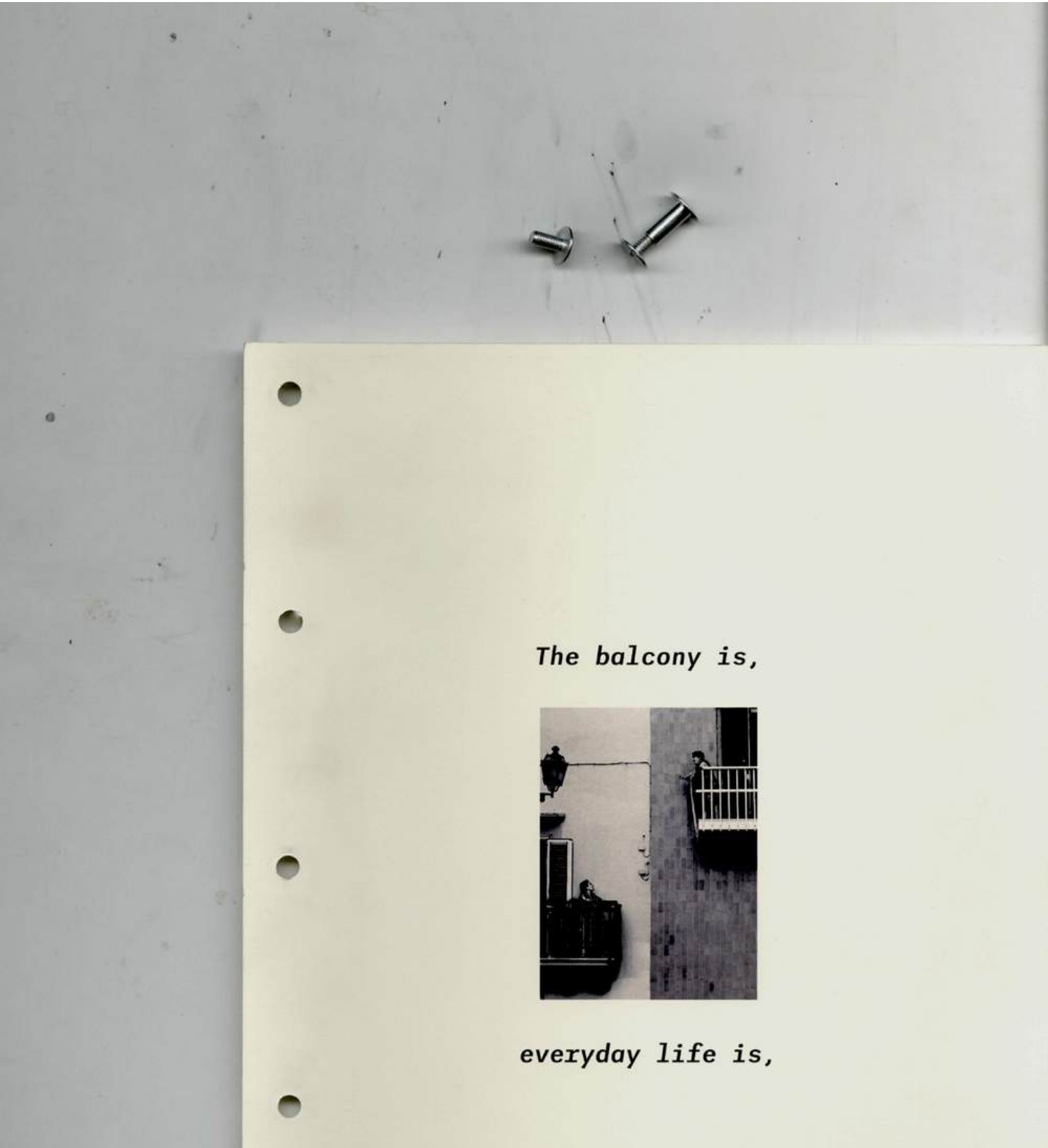
```
def create_handrails(result, step_points, landing_points, width):  
    """Create handrails on both sides of the staircase"""  
    # Handrail parameters  
    handrail_height = 0.9 # Height above steps  
    handrail_diameter = 0.04 # Thickness of handrail  
    post_diameter = 0.03 # Thickness of vertical posts  
  
    # Check if we have any steps  
    if not step_points:  
        return  
  
    # Evacuate left and right sides  
    for side_idx, side in enumerate(['left', 'right']):  
        # Group steps by flight  
        flights = []  
        for i, (left, right, flight_num) in enumerate(step_points):  
            if flight_num not in flights:  
                flights[flight_num] = []  
  
        # Use left or right point based on side  
        point = left if side == 'left' else right  
        flights[flight_num].append((i, point))  
  
        # Process each flight  
        all_handrail_points = []  
  
        for flight_num in sorted(flights.keys()):  
            flight_points = flights[flight_num]  
  
            # Create handrail points (slightly above steps)  
            handrail_points = []  
            for i, point in flight_points:  
                # Raise point for handrail height  
                handrail_point = rg.Point3d(point)  
                handrail_point.Z += handrail_height  
                handrail_points.append(handrail_point)  
  
            # Create handrail along these points  
            if len(handrail_points) > 1:  
                # Create a curve through these points  
                handrail_curve = rg.Curve.CreateInterpolatedCurve(  
                    handrail_points, 3, rg.CurveKnotStyle.Uniform  
                )  
  
                # If handrail curve:  
                # Create a pipe along this curve for the handrail  
                handrail = rg.Mesh.CreateFromPipe(  
                    handrail_curve, handrail_diameter, 8, 1, True, True  
                )  
                result.append(handrail)  
  
            # Add vertical posts at each step  
            for i, (step_idx, point) in enumerate(flight_points):  
                # Create a post at every other step or at beginning/end  
                if i == 0 or i % 2 == 0 or (i == len(flight_points) - 1 or i % 2 == 1):  
                    # Create line from step to handrail  
                    post_bottom = rg.Point3d(point)  
                    post_top = rg.Point3d(handrail_points[i])  
                    post_line = rg.Line(post_bottom, post_top).ToNurbsCurve()  
  
                    # Create pipe for post  
                    post = rg.Mesh.CreateFromPipe(  
                        post_line, post_diameter, 8, 1, True, True  
                    )  
  
                    if post:  
                        result.append(post)  
  
    # Add railings for landings if any  
    for landing_idx, (front_left, front_right, back_right, back_left, U) in enumerate(landing_points):  
        # Points for this side of landing  
        if side == 'left':  
            start_point = front_left  
            end_point = back_left  
        else:  
            start_point = front_right  
            end_point = back_right  
  
        # Create railing above landing  
        start_rail = rg.Point3d(start_point)  
        start_rail.Z = landing_idx * handrail_height  
        end_rail = rg.Point3d(end_point)  
        end_rail.Z = landing_idx * handrail_height  
  
        # Create landing handrail  
        landing_rail_line = rg.Line(start_rail, end_rail).ToNurbsCurve()  
        landing_rail = rg.Mesh.CreateFromPipe(  
            landing_rail_line, handrail_diameter, 8, 1, True, True  
        )  
        result.append(landing_rail)  
  
    # Add vertical posts at corners  
    for corner_point in [start_point, end_point]:  
        post_bottom = rg.Point3d(corner_point)  
        post_top = rg.Point3d(handrail_points[0])  
        post_line = rg.Line(post_bottom, post_top).ToNurbsCurve()  
        post = rg.Mesh.CreateFromPipe(  
            post_line, post_diameter, 8, 1, True, True  
        )  
        if post:  
            result.append(post)  
  
    """Improved Staircase Generator following a 2D PathCurve"""  
import Rhino.Geometry as rg  
import math  
  
def generate_stairs(StartPoint, EndPoint, TargetRiserHeight, TreadDepth, StairWidth):  
    """Main function to generate stairs with paths that precisely follow a given path curve"""  
    try:  
        # Result list  
        result = []  
  
        # Basic calculations  
        total_rise = EndPoint.Z - StartPoint.Z  
        num_steps = max(3, int(round(total_rise / TargetRiserHeight)))  
        actual_riser_height = total_rise / num_steps  
  
        # Handle default path curve  
        if PathCurve is None:  
            # Create a straight line from start to end (in XY plane only)  
            end_xy = rg.Point2d(EndPoint.X, EndPoint.Y, StartPoint.Z)  
            PathCurve = rg.Line(StartPoint, end_xy).ToNurbsCurve()  
  
        # Create a 2D version of the path curve (projecting to XY plane)  
        path_2d = PathCurve.DuplicateCurve()  
        for i in range(path_2d.Points.Count):  
            pt = path_2d.Points[i]  
            pt.Z = StartPoint.Z # Set all control points to start Z level  
  
        # Add curve to result for visualization  
        result.append(PathCurve)  
  
        # Determine if we need landings based on MaxStepsPerFlight  
        if MaxStepsPerFlight <= 0:  
            MaxStepsPerFlight = num_steps # No landings  
  
        # Force landings if we have more steps than MaxStepsPerFlight  
        needs_landings = num_steps > MaxStepsPerFlight  
  
        # Store step information for handrails  
        step_points = [] # Will hold (left_point, right_point, flight_number) for each step  
        landing_points = [] # Will hold (front_left, front_right, back_right, back_left, U) for each landing  
  
        if not needs_landings:  
            # Simple staircase without landings  
            step_points = build_stairs_along_2d_path(result, path_2d, StartPoint, EndPoint, actual_riser_height, TreadDepth, StairWidth)  
            landing_points = [] # No landings in this case  
        else:  
            # Staircase with landings  
            num_flights = (num_steps + MaxStepsPerFlight - 1) // MaxStepsPerFlight  
            steps_per_flight = []  
            remaining_steps = num_steps  
  
            for i in range(num_flights):  
                if i == num_flights - 1:  
                    # Last flight gets all remaining steps  
                    steps_per_flight.append(remaining_steps)  
                else:  
                    # Each flight gets MaxStepsPerFlight steps  
                    steps_per_flight.append(min(MaxStepsPerFlight, remaining_steps))  
                    remaining_steps -= steps_per_flight[i]  
  
            # Build staircase with landings  
            stair_result = build_stairs_with_landings_along_2d_path(result, path_2d, StartPoint, EndPoint, actual_riser_height, TreadDepth, StairWidth, steps_per_flight, landing_points)  
  
            # Handle return values  
            if stair_result is not None and len(stair_result) > 2:  
                step_points, landing_points = stair_result  
            else:  
                # In case of error, initialize empty lists  
                step_points = []  
                landing_points = []  
  
        # Create handrails  
        create_handrails(result, step_points, landing_points, StairWidth)  
  
        return result  
  
def build_stairs_along_2d_path(result, curve_2d, start_point, end_point, num_steps, riser_height, tread_depth, width):  
    """Build stairs along a 2D path curve with controlled tread depth and riser height"""  
    # Calculate total curve length  
    curve_length = curve_2d.GetLength()  
  
    # Calculate total horizontal distance needed based on tread depth  
    total_tread_length = (num_steps - 1) * tread_depth  
  
    # Determine step spacing along curve  
    step_spacing = min(tread_depth, curve_length / max(1, num_steps - 1))  
  
    # Store step points for handrails  
    step_points = [] # (left_point, right_point, flight_number)  
  
    # Create steps evenly spaced along the curve based on tread depth  
    for step in range(num_steps):  
        # Calculate parameter along curve based on step spacing  
        t = min(1.0, step * step_spacing / curve_length)  
  
        # Special case for last step - ensure it reaches the endpoint XY point  
        if step == num_steps - 1:  
            point = rg.Point3d(end_point.X, end_point.Y, start_point.Z)  
        else:  
            # Get point along curve at parameter t  
            point = curve_2d.PointAt(curve_2d.Domain.ParameterAt(t))  
  
        # Set Z height based on step index  
        point.Z = start_point.Z + step * riser_height  
  
        # Get tangent at this point  
        if step == num_steps - 1:  
            # For last step, use the tangent at the end of the curve  
            tangent = curve_2d.TangentAtEnd  
        else:  
            tangent = curve_2d.TangentAt(curve_2d.Domain.ParameterAt(t))  
  
        tangent.Z = 0 # Make tangent horizontal  
  
        if tangent.Length < 0.001:  
            # Default direction if tangent is zero  
            tangent = rg.Vector3d(1, 0, 0)  
            tangent.Unitize()  
  
        # Calculate width vector for step  
        width_vector = rg.Vector3d(-tangent.Y, tangent.X, 0)  
        width_vector.Unitize()  
  
        # Store points for handrails  
        left_point = point + width_vector * (width/2)  
        right_point = point - width_vector * (width/2)  
        step_points.append((left_point, right_point, step)) # step is the flight number  
  
    # Create step mesh  
    step_mesh = create_step_box(point, tangent, tread_depth, width, riser_height)  
    if step_mesh:  
        result.append(step_mesh)  
  
    return step_points  
  
def build_stairs_with_landings_along_2d_path(result, curve_2d, start_point, end_point, actual_riser_height, tread_depth, stair_width, steps_per_flight, landing_points):  
    """Build stairs with landings along a 2D path curve with controlled tread depth and riser height"""  
    # Calculate curve length  
    curve_length = curve_2d.GetLength()  
  
    # Calculate total horizontal distance needed based on tread depth  
    total_tread_length = (num_steps - 1) * tread_depth  
  
    # Determine step spacing along curve  
    step_spacing = min(tread_depth, curve_length / max(1, num_steps - 1))  
  
    # Store step points for handrails  
    step_points = [] # (left_point, right_point, flight_number)  
  
    # Create steps evenly spaced along the curve based on tread depth  
    for step in range(num_steps):  
        # Calculate parameter along curve based on step spacing  
        t = min(1.0, step * step_spacing / curve_length)  
  
        # Special case for last step - ensure it reaches the endpoint XY point  
        if step == num_steps - 1:  
            point = rg.Point3d(end_point.X, end_point.Y, start_point.Z)  
        else:  
            # Get point along curve at parameter t  
            point = curve_2d.PointAt(curve_2d.Domain.ParameterAt(t))  
  
        # Set Z height based on step index  
        point.Z = start_point.Z + step * riser_height  
  
        # Get tangent at this point  
        if step == num_steps - 1:  
            # For last step, use the tangent at the end of the curve  
            tangent = curve_2d.TangentAtEnd  
        else:  
            tangent = curve_2d.TangentAt(curve_2d.Domain.ParameterAt(t))  
  
        tangent.Z = 0 # Make tangent horizontal  
  
        if tangent.Length < 0.001:  
            # Default direction if tangent is zero  
            tangent = rg.Vector3d(1, 0, 0)  
            tangent.Unitize()  
  
        # Calculate width vector for step  
        width_vector = rg.Vector3d(-tangent.Y, tangent.X, 0)  
        width_vector.Unitize()  
  
        # Store points for handrails  
        left_point = point + width_vector * (width/2)  
        right_point = point - width_vector * (width/2)  
        step_points.append((left_point, right_point, step)) # step is the flight number  
  
    # Create step mesh  
    step_mesh = create_step_box(point, tangent, tread_depth, width, riser_height)  
    if step_mesh:  
        result.append(step_mesh)  
  
    return step_points
```

```
def create_handrails(result, step_points, landing_points, width):  
    """Create handrails on both sides of the staircase"""  
    # Handrail parameters  
    handrail_height = 0.9 # Height above steps  
    handrail_diameter = 0.04 # Thickness of handrail  
    post_diameter = 0.03 # Thickness of vertical posts  
  
    # Check if we have any steps  
    if not step_points:  
        return  
  
    # Evacuate left and right sides  
    for side_idx, side in enumerate(['left', 'right']):  
        # Group steps by flight  
        flights = []  
        for i, (left, right, flight_num) in enumerate(step_points):  
            if flight_num not in flights:  
                flights[flight_num] = []  
  
        # Use left or right point based on side  
        point = left if side == 'left' else right  
        flights[flight_num].append((i, point))  
  
        # Process each flight  
        all_handrail_points = []  
  
        for flight_num in sorted(flights.keys()):  
            flight_points = flights[flight_num]  
  
            # Create handrail points (slightly above steps)  
            handrail_points = []  
            for i, point in flight_points:  
                # Raise point for handrail height  
                handrail_point = rg.Point3d(point)  
                handrail_point.Z += handrail_height  
                handrail_points.append(handrail_point)  
  
            # Create handrail along these points  
            if len(handrail_points) > 1:  
                # Create a curve through these points  
                handrail_curve = rg.Curve.CreateInterpolatedCurve(  
                    handrail_points, 3, rg.CurveKnotStyle.Uniform  
                )  
  
                # If handrail curve:  
                # Create a pipe along this curve for the handrail  
                handrail = rg.Mesh.CreateFromPipe(  
                    handrail_curve, handrail_diameter, 8, 1, True, True  
                )  
                result.append(handrail)  
  
            # Add vertical posts at each step  
            for i, (step_idx, point) in enumerate(flight_points):  
                # Create a post at every other step or at beginning/end  
                if i == 0 or i % 2 == 0 or (i == len(flight_points) - 1 or i % 2 == 1):  
                    # Create line from step to handrail  
                    post_bottom = rg.Point3d(point)  
                    post_top = rg.Point3d(handrail_points[i])  
                    post_line = rg.Line(post_bottom, post_top).ToNurbsCurve()  
  
                    # Create pipe for post  
                    post = rg.Mesh.CreateFromPipe(  
                        post_line, post_diameter, 8, 1, True, True  
                    )  
  
                    if post:  
                        result.append(post)  
  
    # Add railings for landings if any  
    for landing_idx, (front_left, front_right, back_right, back_left, U) in enumerate(landing_points):  
        # Points for this side of landing  
        if side == 'left':  
            start_point = front_left  
            end_point = back_left  
        else:  
            start_point = front_right  
            end_point = back_right  
  
        # Create railing above landing  
        start_rail = rg.Point3d(start_point)  
        start_rail.Z = landing_idx * handrail_height  
        end_rail = rg.Point3d(end_point)  
        end_rail.Z = landing_idx * handrail_height  
  
        # Create landing handrail  
        landing_rail_line = rg.Line(start_rail, end_rail).ToNurbsCurve()  
        landing_rail = rg.Mesh.CreateFromPipe(  
            landing_rail_line, handrail_diameter, 8, 1, True, True  
        )  
        result.append(landing_rail)  
  
    # Add vertical posts at corners  
    for corner_point in [start_point, end_point]:  
        post_bottom = rg.Point3d(corner_point)  
        post_top = rg.Point3d(handrail_points[0])  
        post_line = rg.Line(post_bottom, post_top).ToNurbsCurve()  
        post = rg.Mesh.CreateFromPipe(  
            post_line, post_diameter, 8, 1, True, True  
        )  
        if post:  
            result.append(post)  
  
    """Improved Staircase Generator following a 2D PathCurve"""  
import Rhino.Geometry as rg  
import math  
  
def generate_stairs(StartPoint, EndPoint, TargetRiserHeight, TreadDepth, StairWidth):  
    """Main function to generate stairs with paths that precisely follow a given path curve"""  
    try:  
        # Result list  
        result = []  
  
        # Basic calculations  
        total_rise = EndPoint.Z - StartPoint.Z  
        num_steps = max(3, int(round(total_rise / TargetRiserHeight)))  
        actual_riser_height = total_rise / num_steps  
  
        # Handle default path curve  
        if PathCurve is None:  
            # Create a straight line from start to end (in XY plane only)  
            end_xy = rg.Point3d(EndPoint.X, EndPoint.Y, StartPoint.Z)  
            PathCurve = rg.Line(StartPoint, end_xy).ToNurbsCurve()  
  
        # Create a 2D version of the path curve (projecting to XY plane)  
        path_2d = PathCurve.DuplicateCurve()  
        for i in range(path_2d.Points.Count):  
            pt = path_2d.Points[i]
```



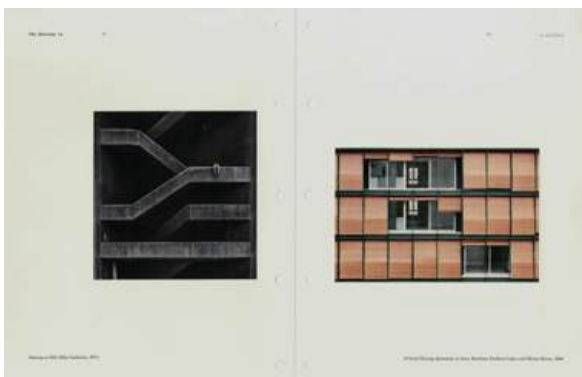
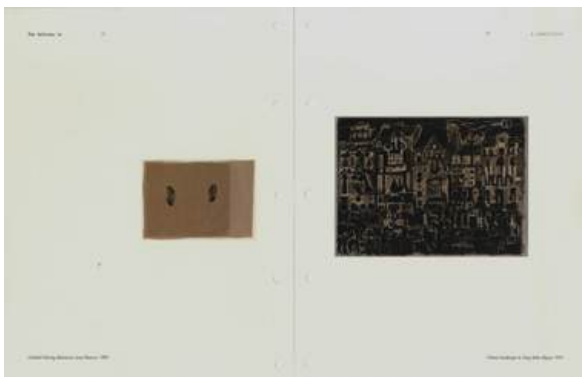
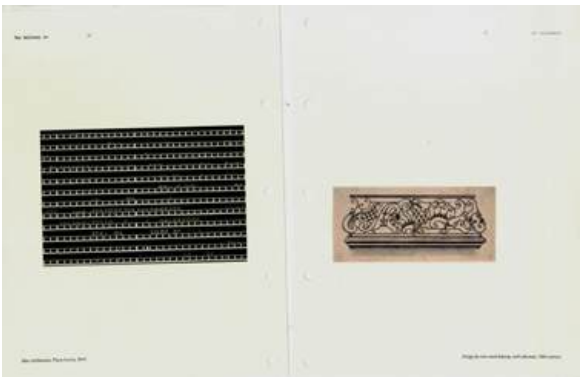
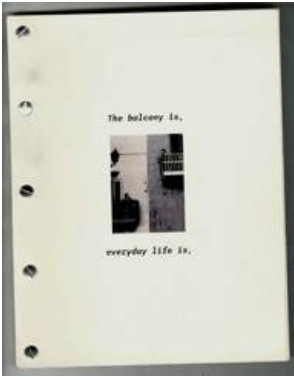
09 The Balcony Is

Spring 2026
GAP I: Design & Typography
Instructor: Yoonjai Choi
Individual Work



An investigation into the **balcony** as an overlooked architectural archetype embedded within everyday life. Through **collected imagery** from various artists, **layout composition**, and **excerpts from Georges Perec**, the project constructs a visual essay examining how **seemingly mundane** spatial elements accumulate **political, social, and historical meaning** across different contexts and histories. Conceived as an **open-ended archive**, the book remains an ongoing project and evolving collection.





This book unfolds as a sequence of oppositions.
 "Balconies" are placed beside one another.
 With fragments from Georges Perec, to guide attention toward
 what is already there, or what might be missing.
 In search of the multiple lives of an everyday archetype.



The balcony is:

the architecture of watching an in-between that turns distance
 into a form of participation;

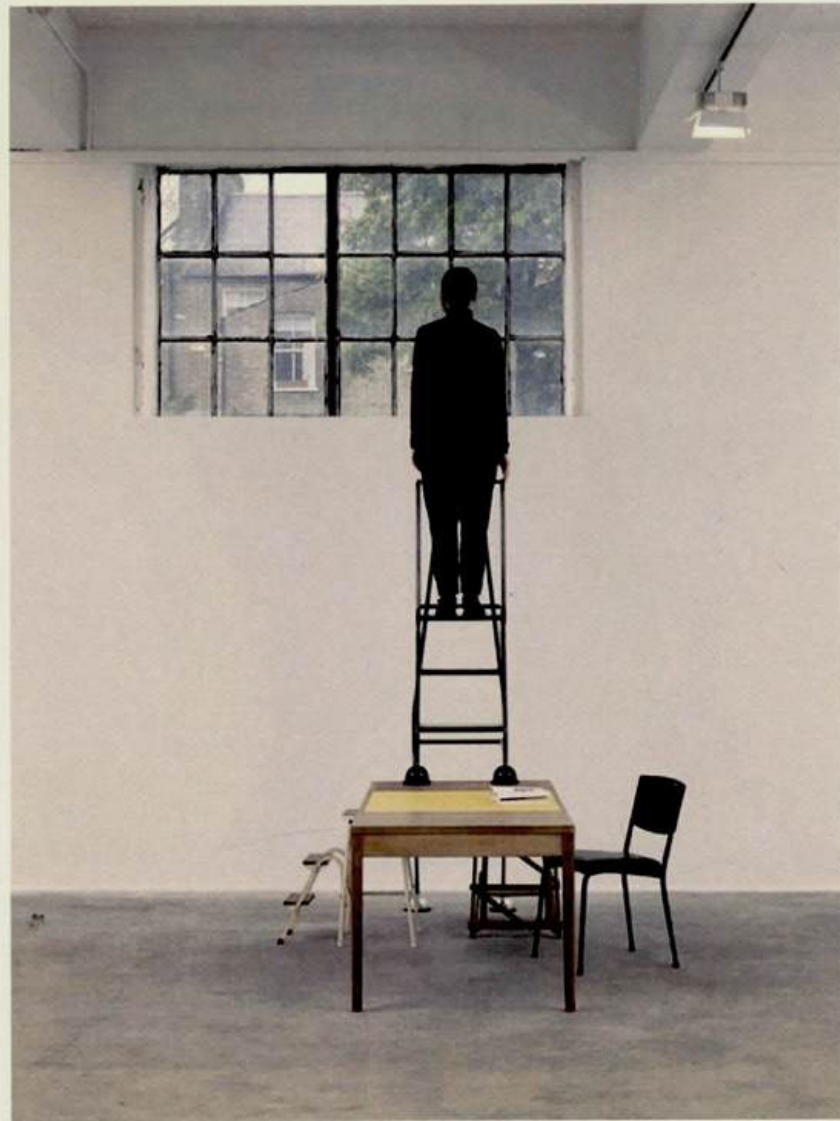
a domestic device for producing spectacle out of the everyday;

a site of public address and representational space of power, a
 liminal space between the private and public realms, between the
 inside and outside, between the "above" and "below".

an absence, precarious, an infrastructure, a playground, a
 gathering, an ornament, an extension, a threshold, an exit, a
 transition, a repetition, a stage, a perspective, a story, a
 surface, an end.

The balcony is

12



The Double And The Half (To Avery Gordon), Celine Condorelli, Chisenhale Gallery, 2014

13

an infrastructure



Centro di Permanenza Temporanea, Adrian Paci, 2007

