Poietic Veil
TU Delft Science Centre
Prototype Testbed 2022

Philip Beesley, Matt Gorbet, Aadjan van der Helm & Teun Verkerk

Living Architecture Systems Group

This folio documents an installation that was constructed by the Living Architecture Systems Group in collaboration with Science Centre Delft and the Interactive Environments program of TU Delft on the occasion of a workshop with TU Delft design students which ran November 7–12, 2022. The workshop culminated in the creation of Poietic Veil, a prototype testbed for communication between TU Delft's own Connected Interaction Kit of distributed electronics hardware and software, and the Living Architecture Systems Group's own behaviour programming systems.

The physical structures that have been developed for the Poietic Veil installation include filamentary triangulated skeletal frameworks for highly efficient waffle, shell, and spherical envelopes. Alongside these physical components, the sculpture environment also serves as a foundation for a new software-based system of intercommunication between different programming languages used to program responsive environments.

Building on these exchanges, LASG and TU Delft are now creating a future permanent testbed that will be used for ongoing collaborative research and curriculum development around responsive, sentient architecture.

ISBN 978-1-988366-45-6

9 781988 366456
Riverside Architectural Press

Poietic Veil

TU DELFT SCIENCE CENTRE
PROTOTYPE TESTBED 2022

PHILIP BEESLEY, MATT GORBET
AADJAN VAN DER HELM & TEUN VERKERK

LIVING ARCHITECTURE SYSTEMS GROUP

Poietic Veil
TU Delft Science Centre Prototype Testbed 2022

Philip Beesley, Matt Gorbet,
Aadjan van der Helm & Teun Verkerk
Living Architecture Systems Group

Publisher: Riverside Architectural Press, www.riversidearchitecturalpress.ca

Title: Poietic Veil | TU Delft Science Centre | Prototype Testbed 2022
Authors: Philip Beesley, Matt Gorbet, Aadjan van der Helm and Teun Verkerk.
Living Architecture Systems Group, issuing body.

Description: Series statement: Living Architecture Systems Group folio series | With Living Architecture Systems Group. Includes Index.

Identifiers: ISBN 978-1-988366-45-6 (Online)

Design and Production: Philip Beesley, Timothy Boll, Adrian Chîu, Kevan Cress, Lisa Jiang, Abida Rahman, Rekha Ramachandran

Publication: December 2022
Riverside Architectural Press
7 Melville Street
Cambridge, Ontario, N1S 2H4
Canada

This book is set in Garamond and Zurich BT.

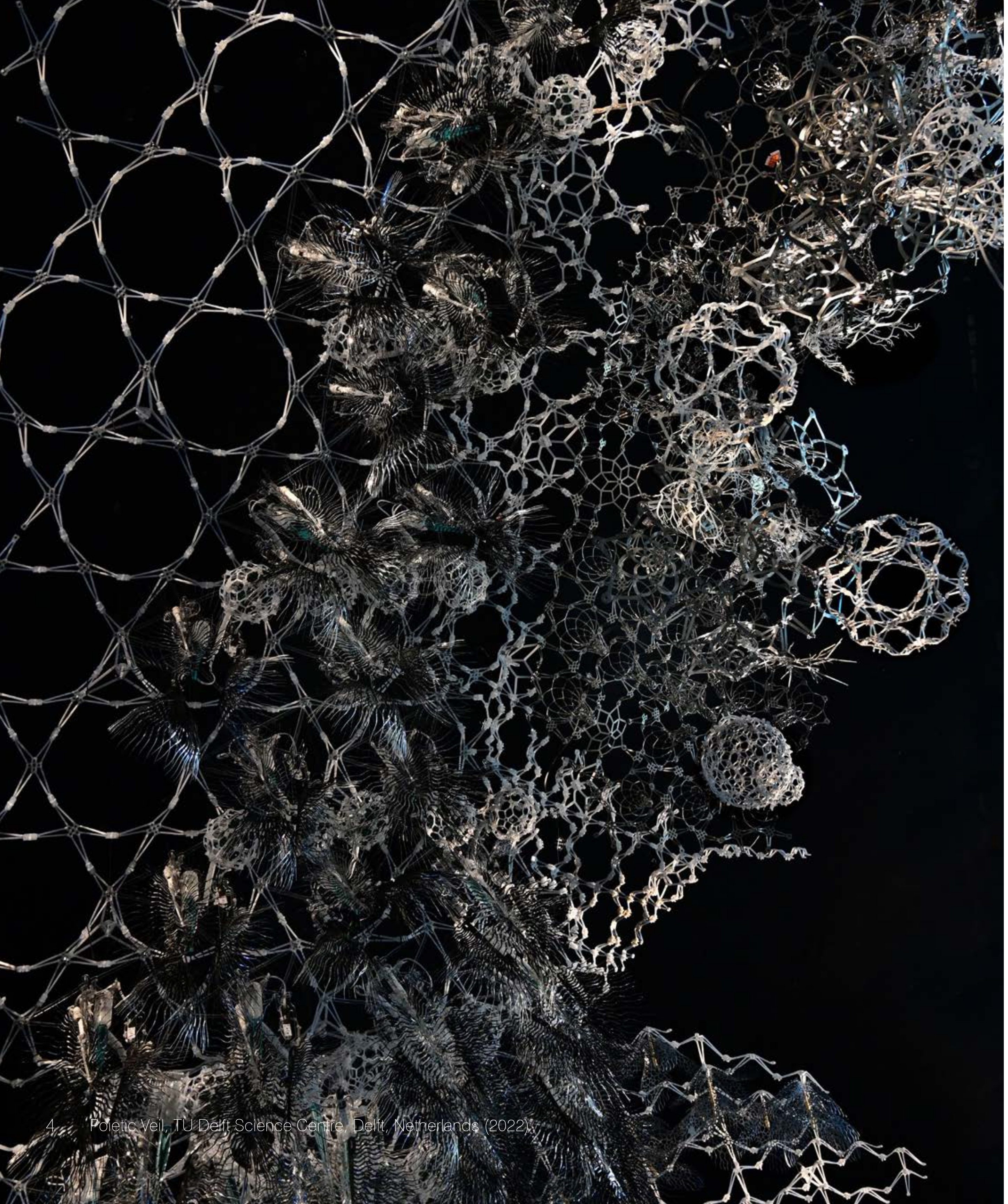
Supported by: Art on Campus, Canada Council for the Arts, Delft University of Technology, Gorbet Design Inc., Social Sciences and Humanities Research Council of Canada, Philip Beesley Studio Inc., University of Waterloo. This work has received funding from the European Union's Horizon 2020 Research and innovation programme under the EnTimeMent grant agreement No. 824160.

Contents

About the Living Architecture Systems Group

The publication forms part of a series of work-in-progress reports and publications by Living Architecture researchers and contributors. The Living Architecture Systems Group is an international partnership of researchers, artists, and industrial collaborators studying how we can build living architectural systems—sustainable, adaptive environments that can move, respond, and learn, and that are inclusive and empathic toward their inhabitants. "Smart" responsive architecture is rapidly transforming our built environments, but it is fraught with problems including sustainability, data privacy, and privatized infrastructure. These concerns need conceptual and technical analysis so that designers, urban developers and architects can work positively within this deeply influential new field. The Living Architecture Systems Group is developing tools and conceptual frameworks for examining materials, forms, and topologies, seeking sustainable, flexible, and durable working models of living architecture.

A series of far-reaching critical questions can be explored by using the tools and frameworks that are described within this specialized publication series: can the buildings that we live in come alive? Could living buildings create a sustainable future with adaptive structures while empathizing and inspiring us? These questions can help redefine architecture with new, lightweight physical structures, embedded sentient and responsive systems, and mutual relationships for occupant that provide tools and frameworks to support the emerging field of living architecture. The objective of this integrated work envisions embodied environments that can provide tangible examples in order to shift architecture away from static and inflexible forms towards spaces that can move, respond, learn, and exchange, becoming adaptive and empathic toward their inhabitants.



Poietic Veil, TU Delft Science Centre, Delft, Netherlands (2022).

Introduction

View the Poietic Veil digital twin models with the QR code above

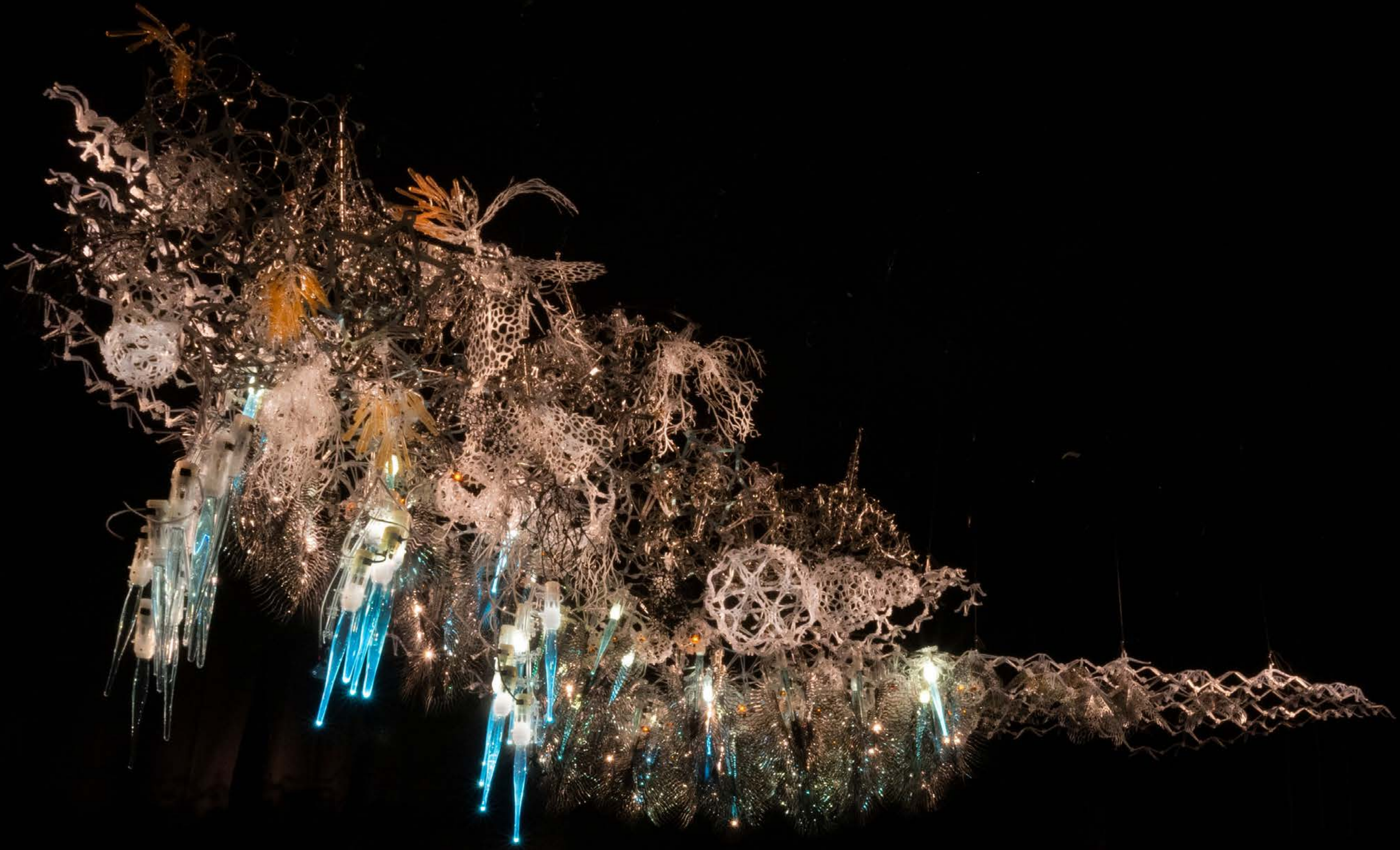
This folio documents an installation that was constructed by the Living Architecture Systems Group (LASG) in collaboration with Science Centre Delft and the Interactive Environments program of TU Delft on the occasion of a workshop with TU Delft design students which ran November 7–12, 2022. The installation is accompanied by digital twin models that can be accessed at http://media.lasg.ca/poieticveil/.

Poietic Veil has been developed as part of a research and creation initiative to develop architecture that approaches the qualities of living systems.[1] Poietic Veil is a prototype testbed for communication between TU Delft's own Connected Interaction Kit of distributed electronics hardware and software, and the Living Architecture Systems Group's own behaviour programming systems. Building on these exchanges, LASG and TU Delft are now creating a future permanent testbed that will be used for ongoing collaborative research and curriculum development around responsive, sentient architecture.

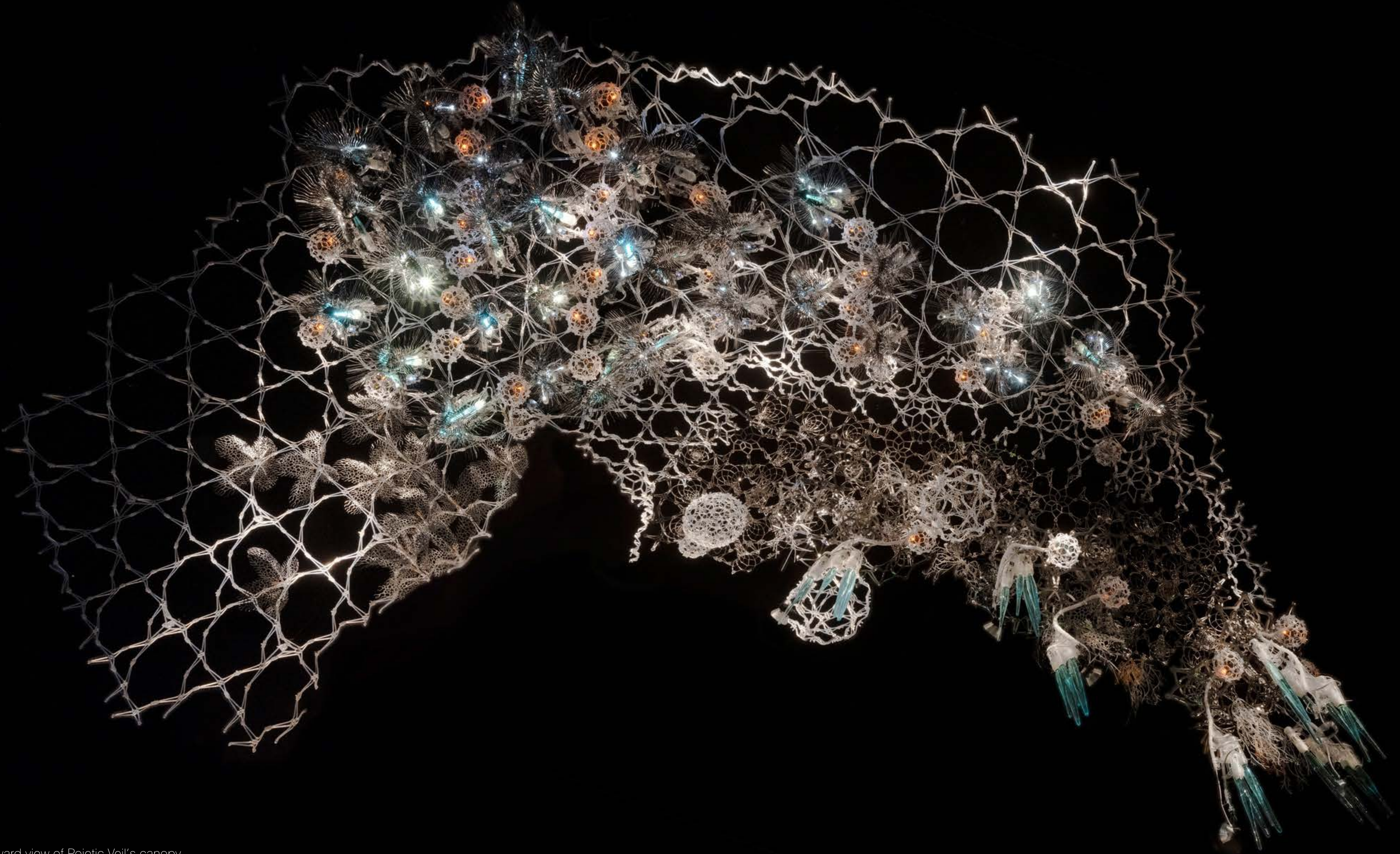
The physical structures that have been developed for the Poietic Veil installation include filamentary triangulated skeletal frameworks for highly efficient waffle, shell, and spherical envelopes. Alongside these physical components, the sculpture environment also serves as a foundation for a new software-based system of intercommunication between different programming languages used to program responsive environments.[2]

[1] Amy M. Youngs, "The Fine Art of Creating Life," *Leonardo* 33, no. 5 (2000): 377–80.

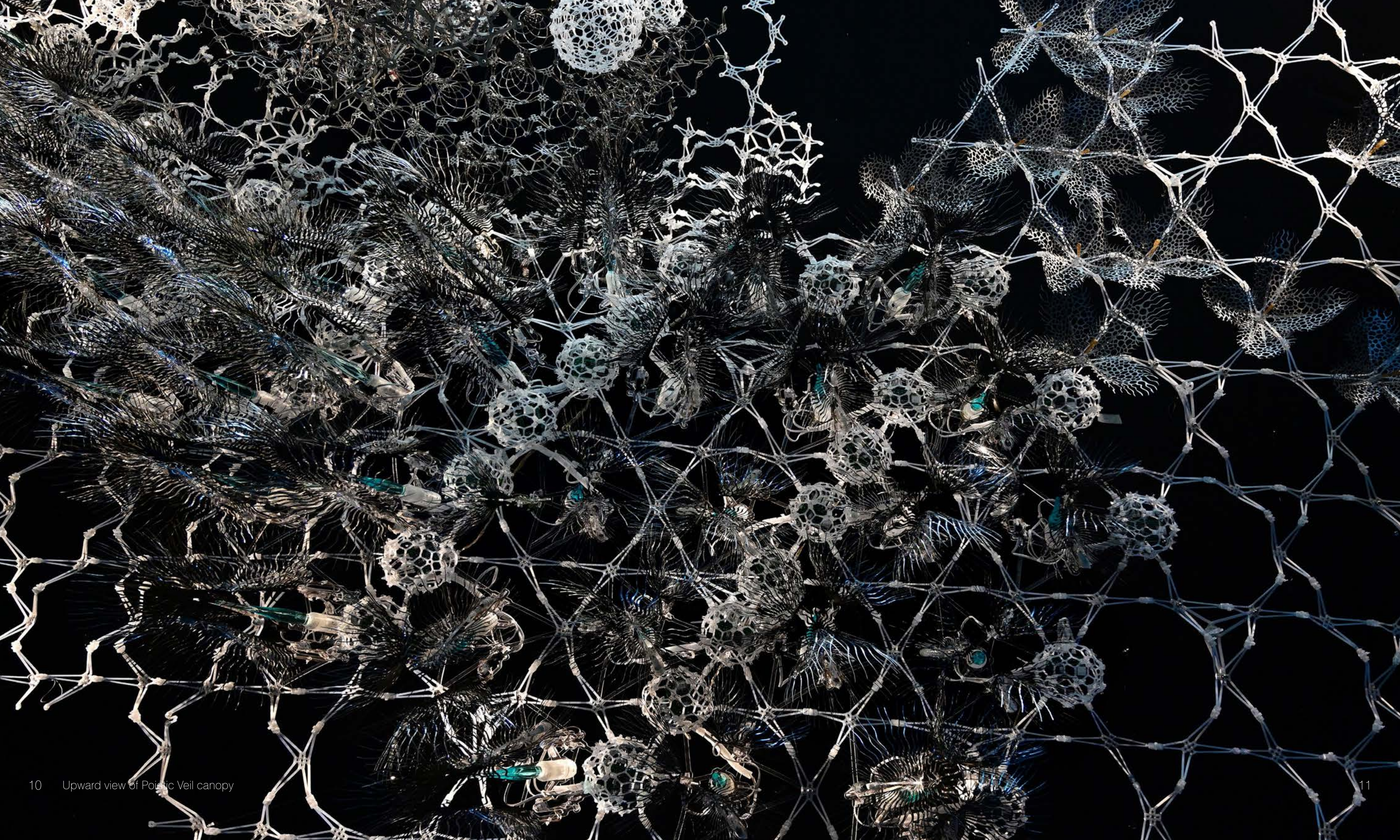
[2] Ben Salem, Jorge Alves Lino, and Jan Simons, "A Framework for Responsive Environments," in *Ambient Intelligence*, eds. Werner Weber, J. M. Rabaey, and Emile H. L. Aarts (Berlin: Springer International Publishing, 2017), 263–77.



Front view of Poietic Veil canopy



Upward view of Poietic Veil’s canopy



Upward view of Poietic Veil canopy

Side view of Poietic Veil

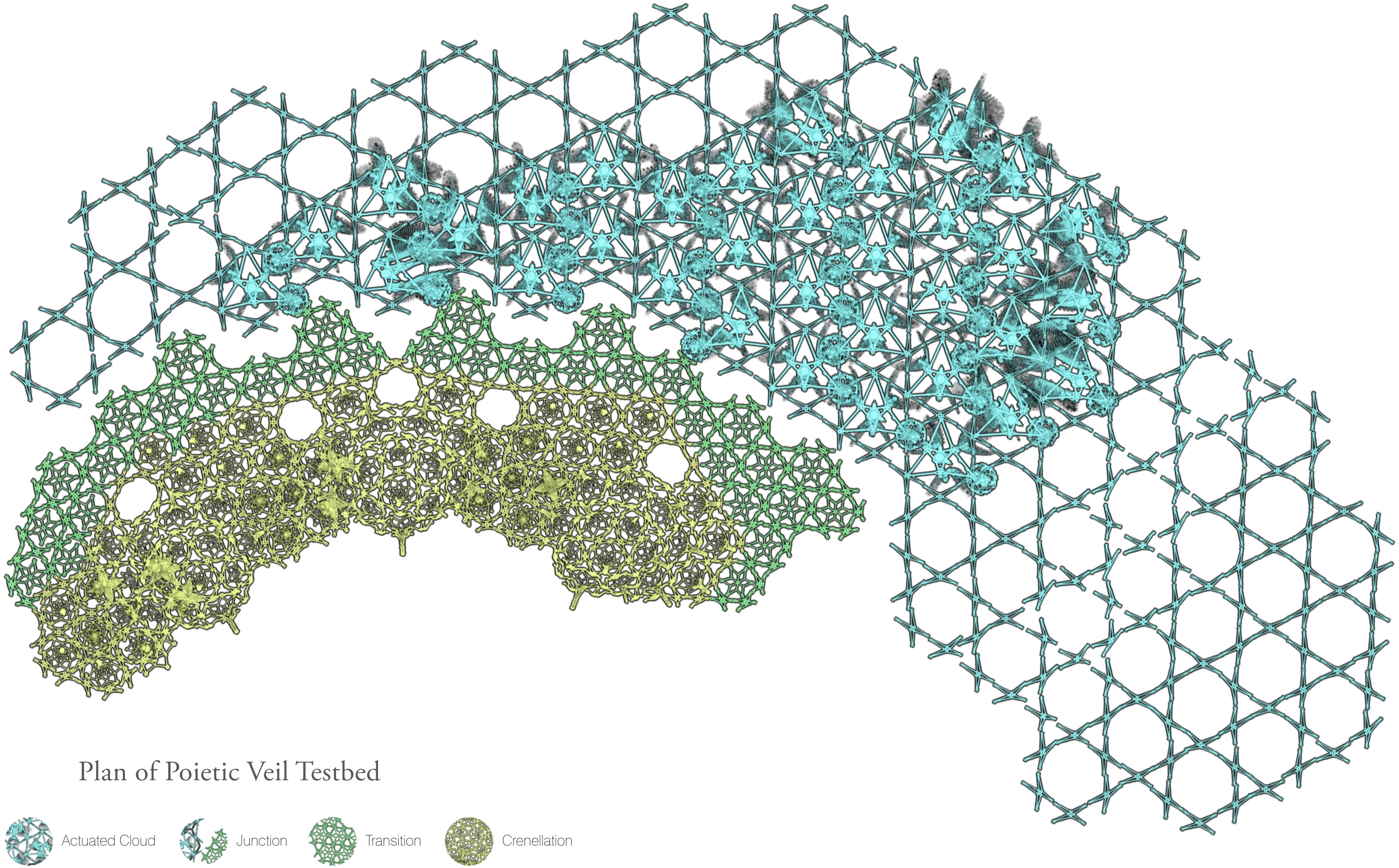
Detail of actuated fronds embedded within the sculpture

Detail of actuated fronds embedded within the sculpture

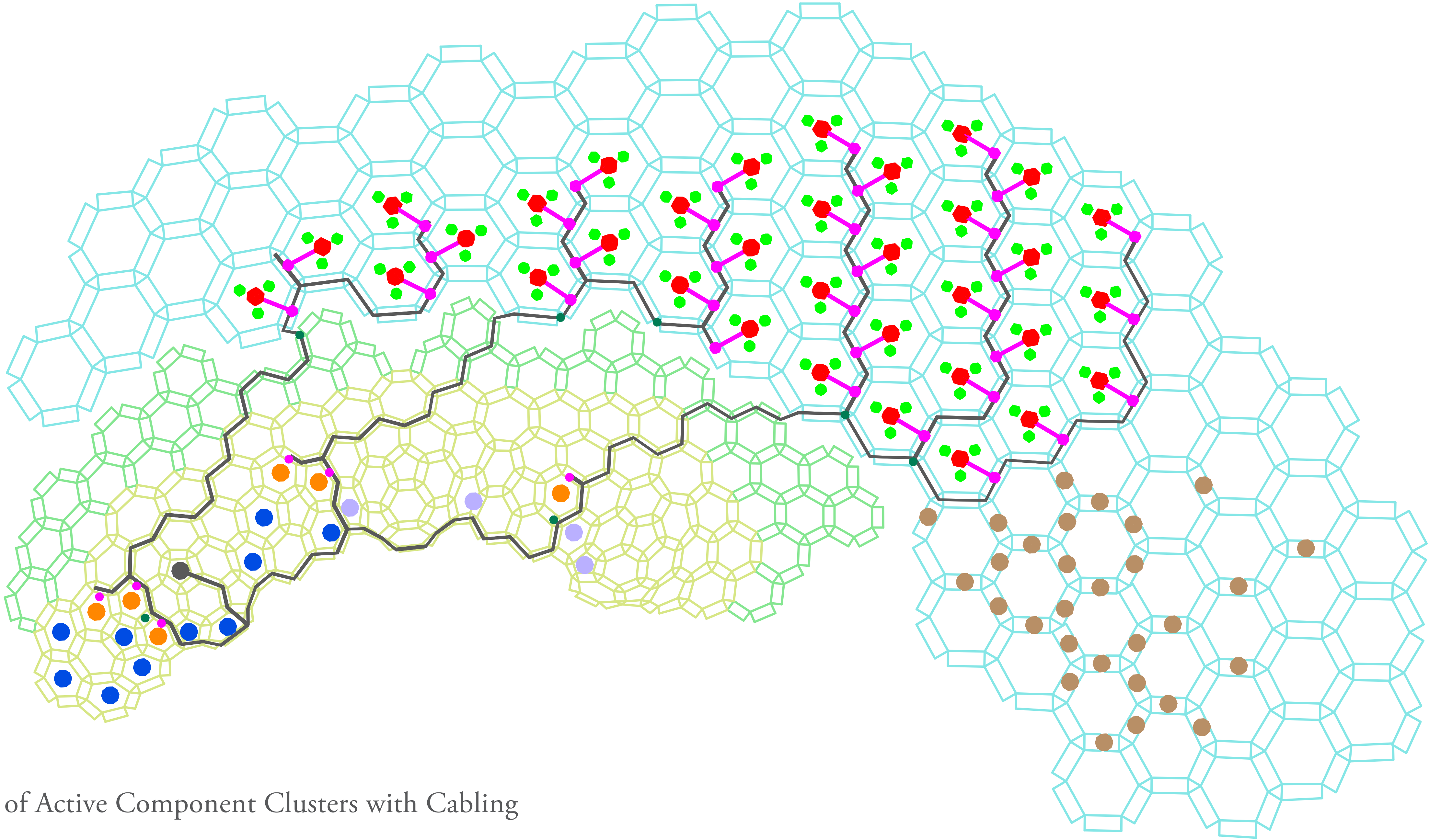
Detail of actuated glass chains within the sculpture

Detail of quad frond veil within the sculpture

Plans and Views



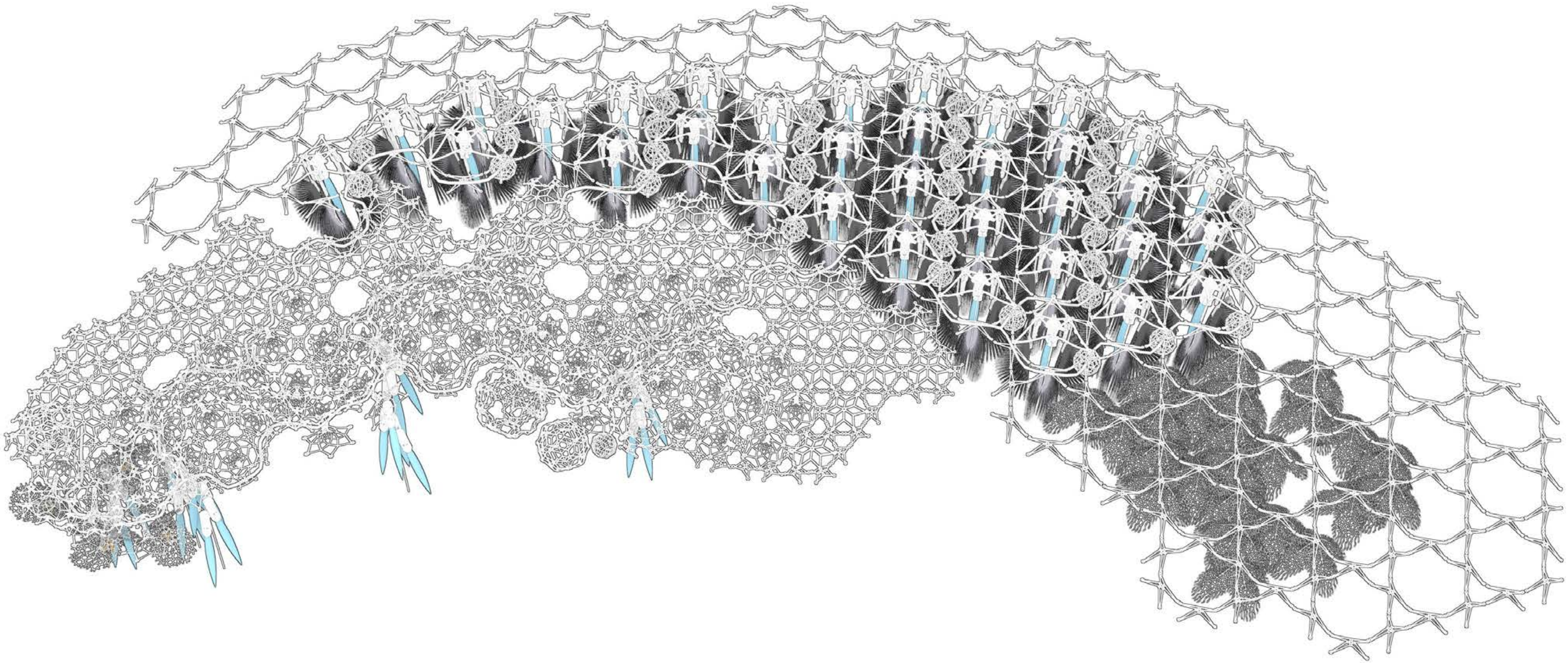
Plan of Poietic Veil Testbed



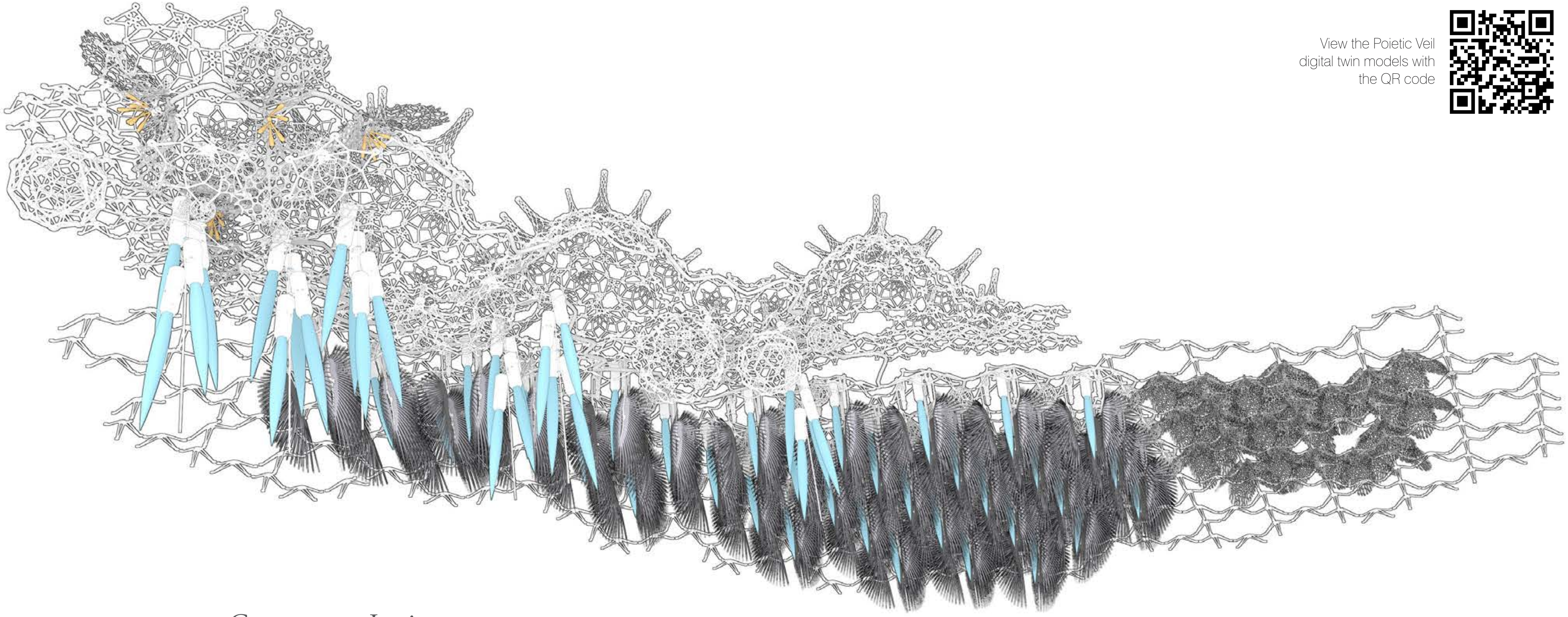
Plan of Active Component Clusters with Cabling

● Demi Moth ● Rebel Star ● Nest ● Junction Nest ● Glass Chain ● Main Arterial

● Birds on a Wire ● Sphere Dressing ● Quad Fronds — Power Cable

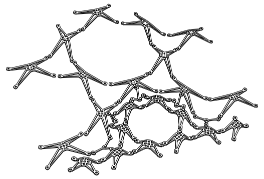


View of Canopy from Above

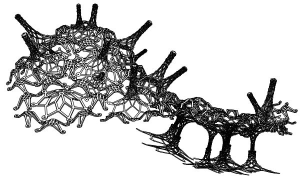


View the Poietic Veil digital twin models with the QR code

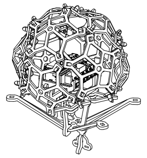
Component Lexicon



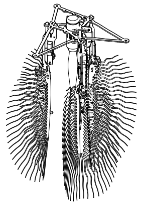
Cloud Scaffold



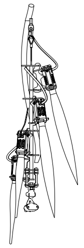
Concave and Convex Crenellation



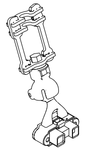
Microcontroller Nest



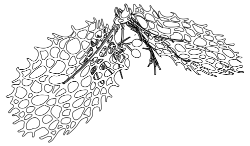
Cloud Cell



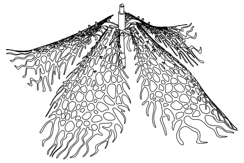
Actuated Glass Chain



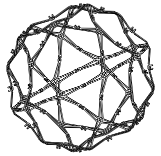
Sensor



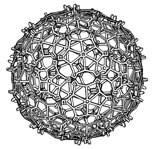
Frond Cluster



Quad Frond

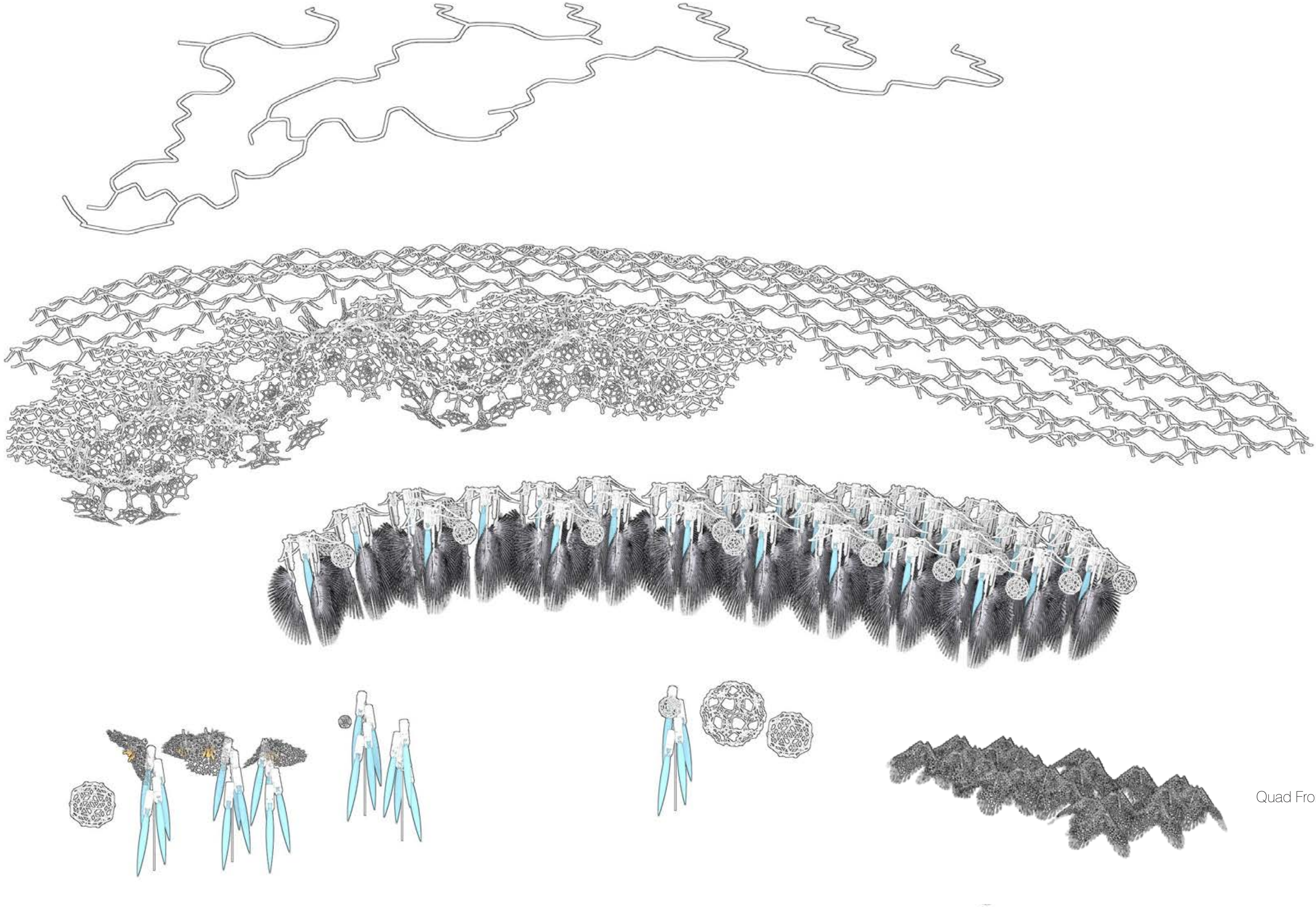


Dodecahedron Sphere

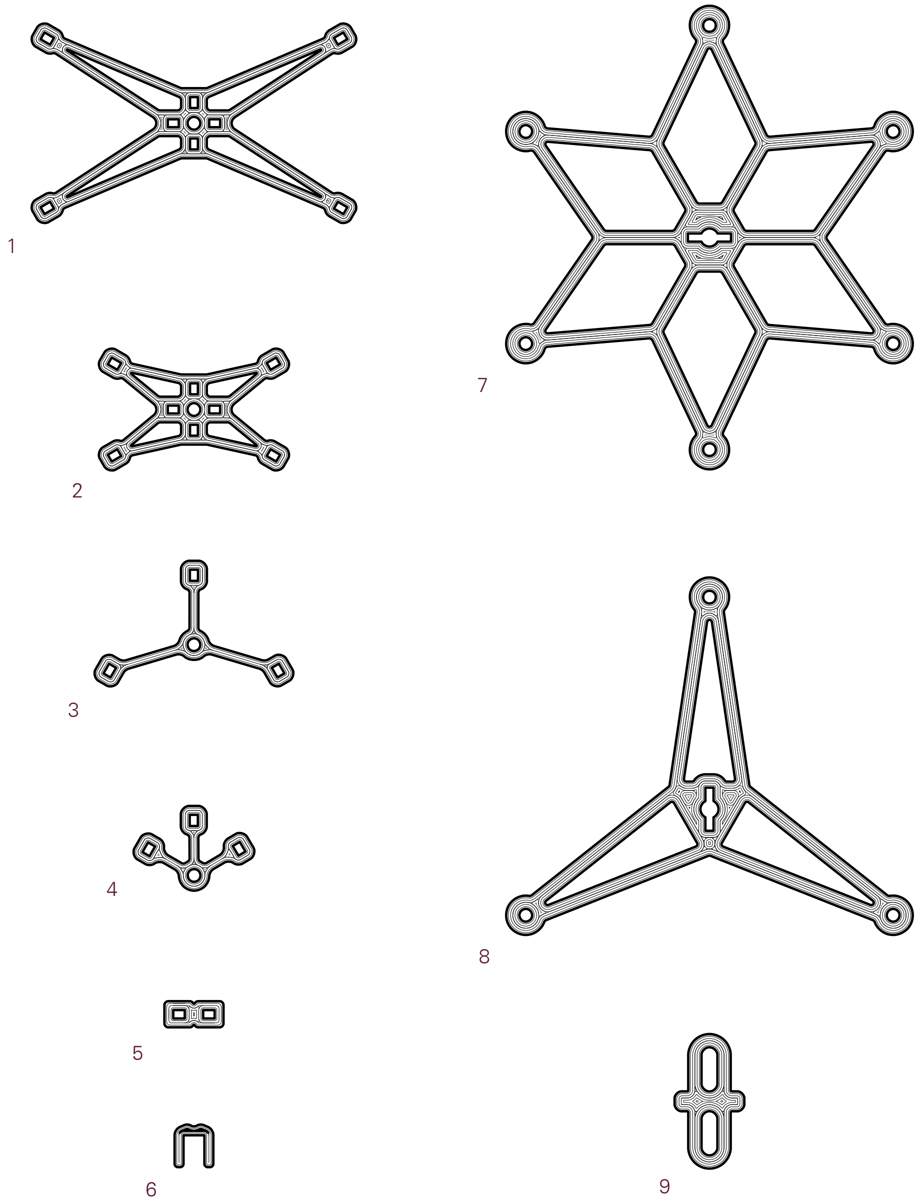


Truncated Icosahedron Sphere

Canopy Layers

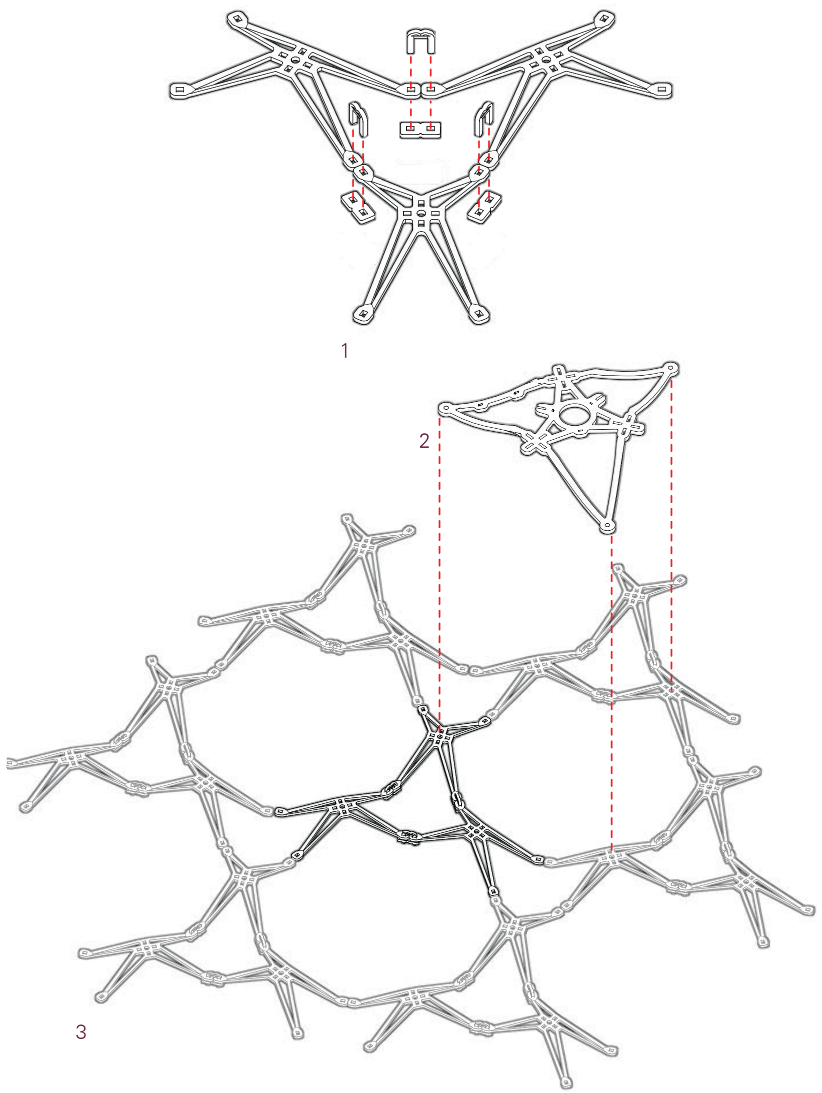

Cable Harnesses
Cloud Scaffold and Crenellated Veil
Cloud Cell Array with Lights and Vibrating Fronds
Actuated Glass Chains with Sensors and Spheres
Quad Frond Veil

Component Assemblies



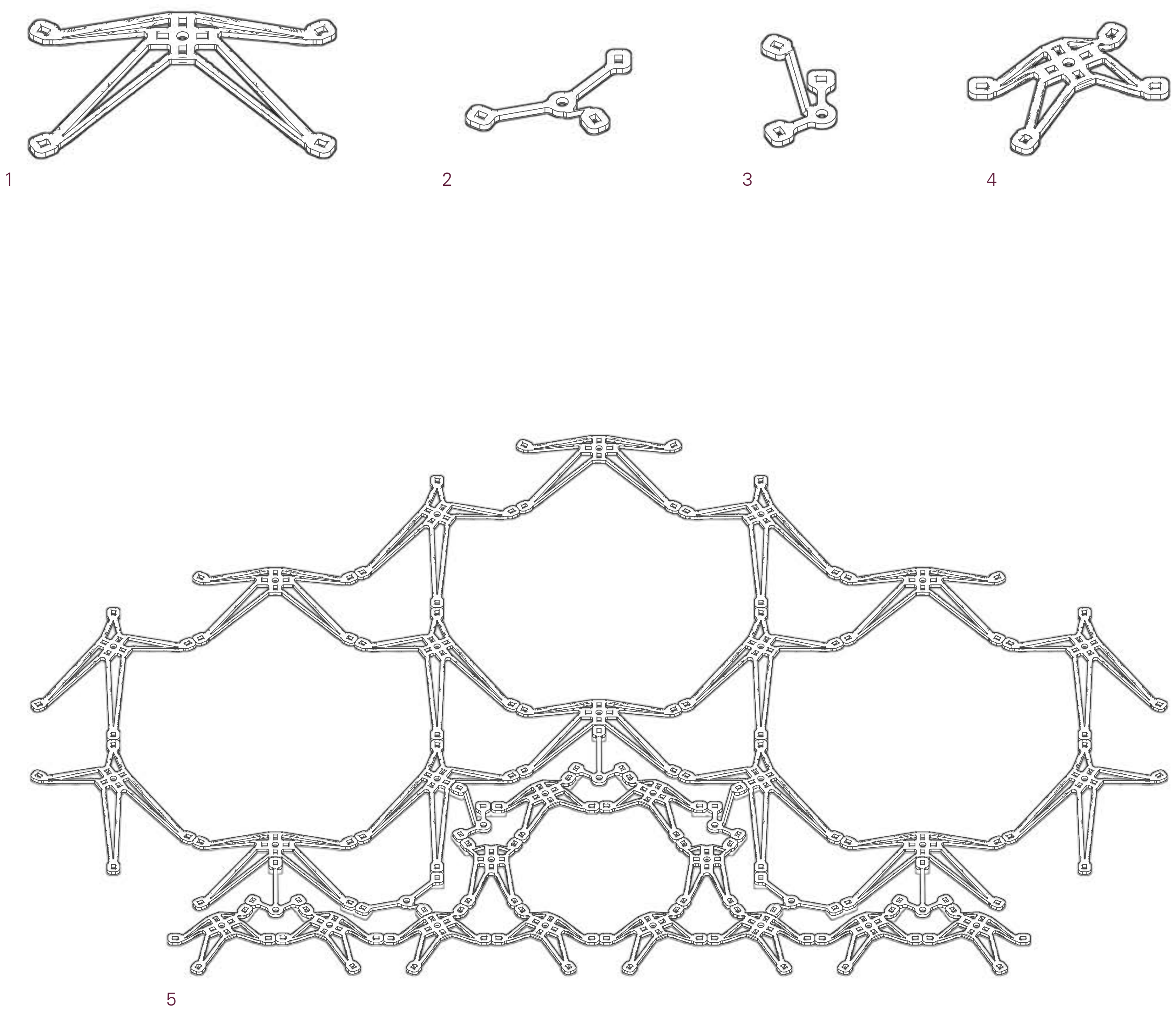
Cloud Scaffold Profiles

1 300mm X-plate 2 150mm X-plate 3 300mm-150mm Exterior Corner Adapter 4 300mm-150mm Interior Corner Adapter
5 Staple 6 Staple Backing 7 150mm Hex Hanger 8 300mm Tri Hanger 9 Hanging Eye



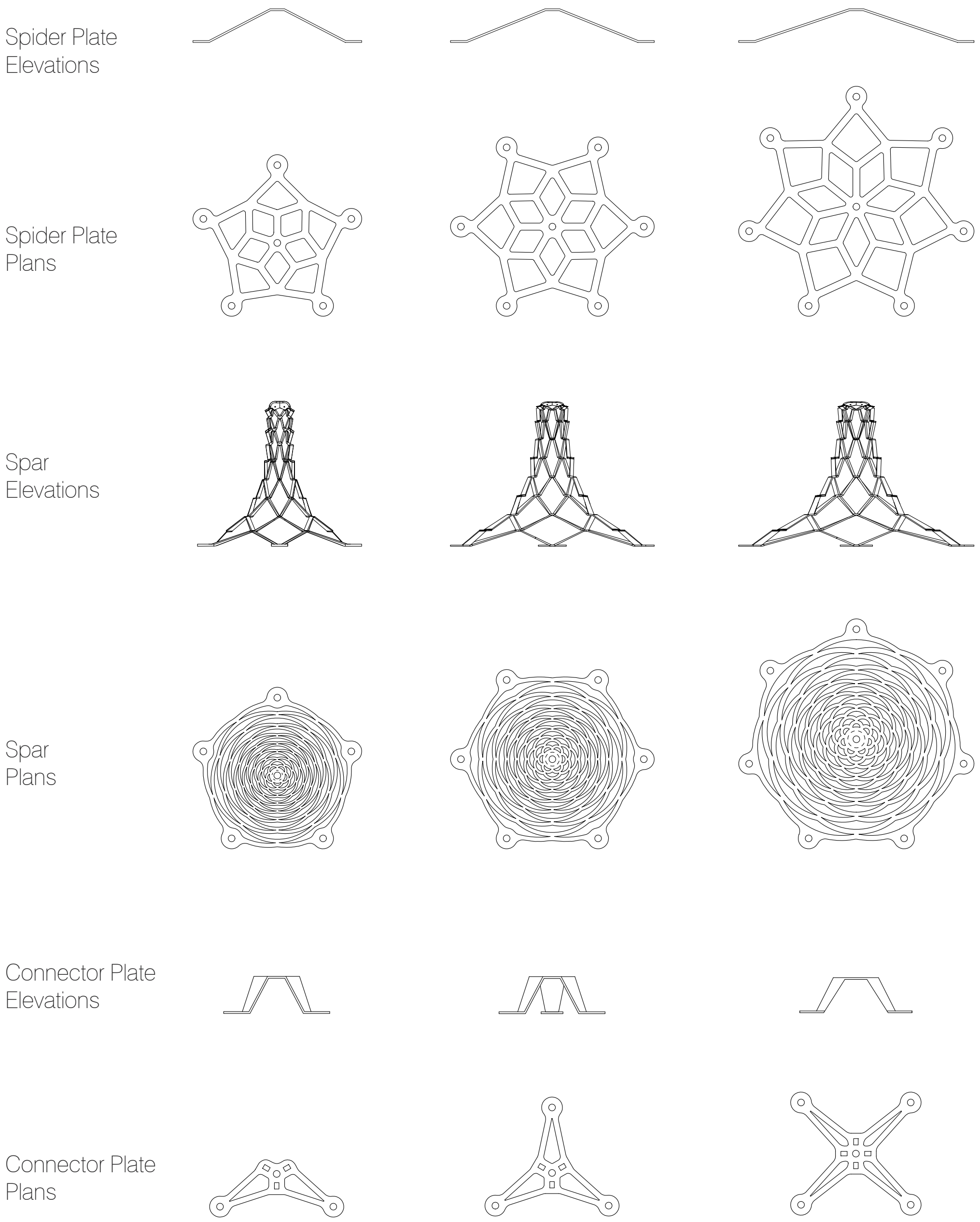
Cloud Scaffold Assembly

1 300mm X-plate Scaffold Unit 2 Actuator Mounting Plate 3 300mm Cloud Scaffold Assembled View

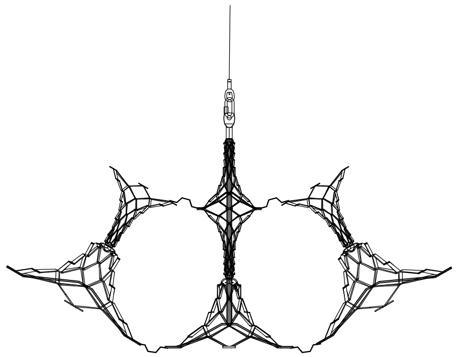


Cloud Scaffold Size Transitions

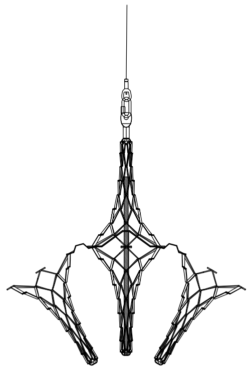
1 300mm X-plate 2 300mm-150mm Exterior Corner Adapter 3 300mm-150mm Interior Corner Adapter
4 150mm X-plate 5 Assembled Junction View



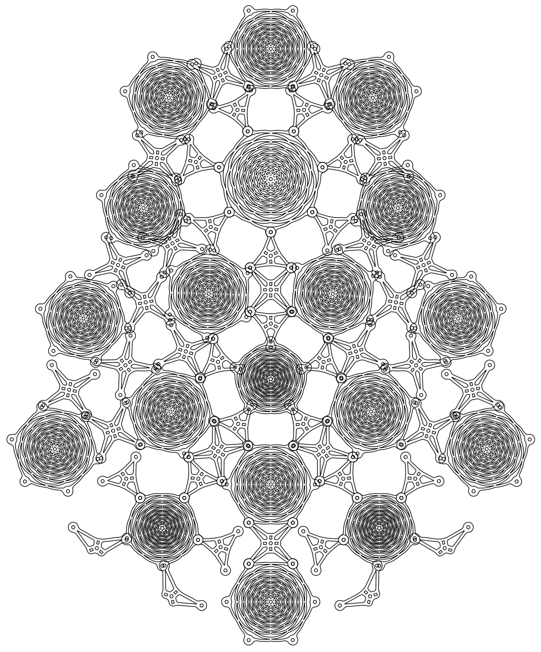
Crenellated Veil Edge Components



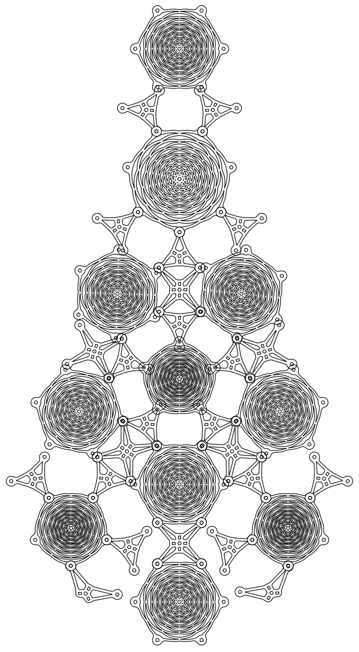
Crenellation Convex Unit Elevation



Crenellation Concave Unit Elevation

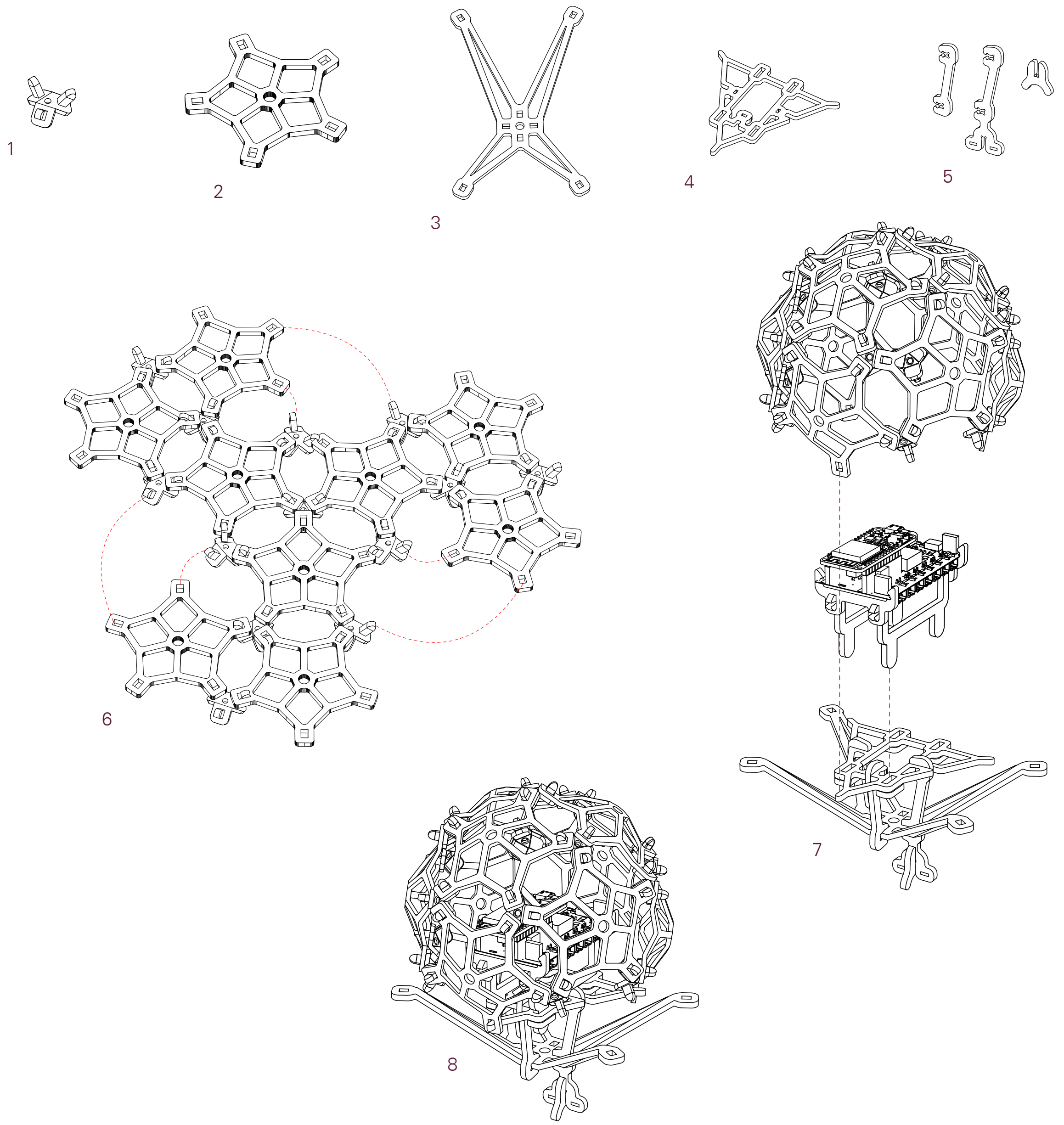


Crenellation Convex Unit Plan



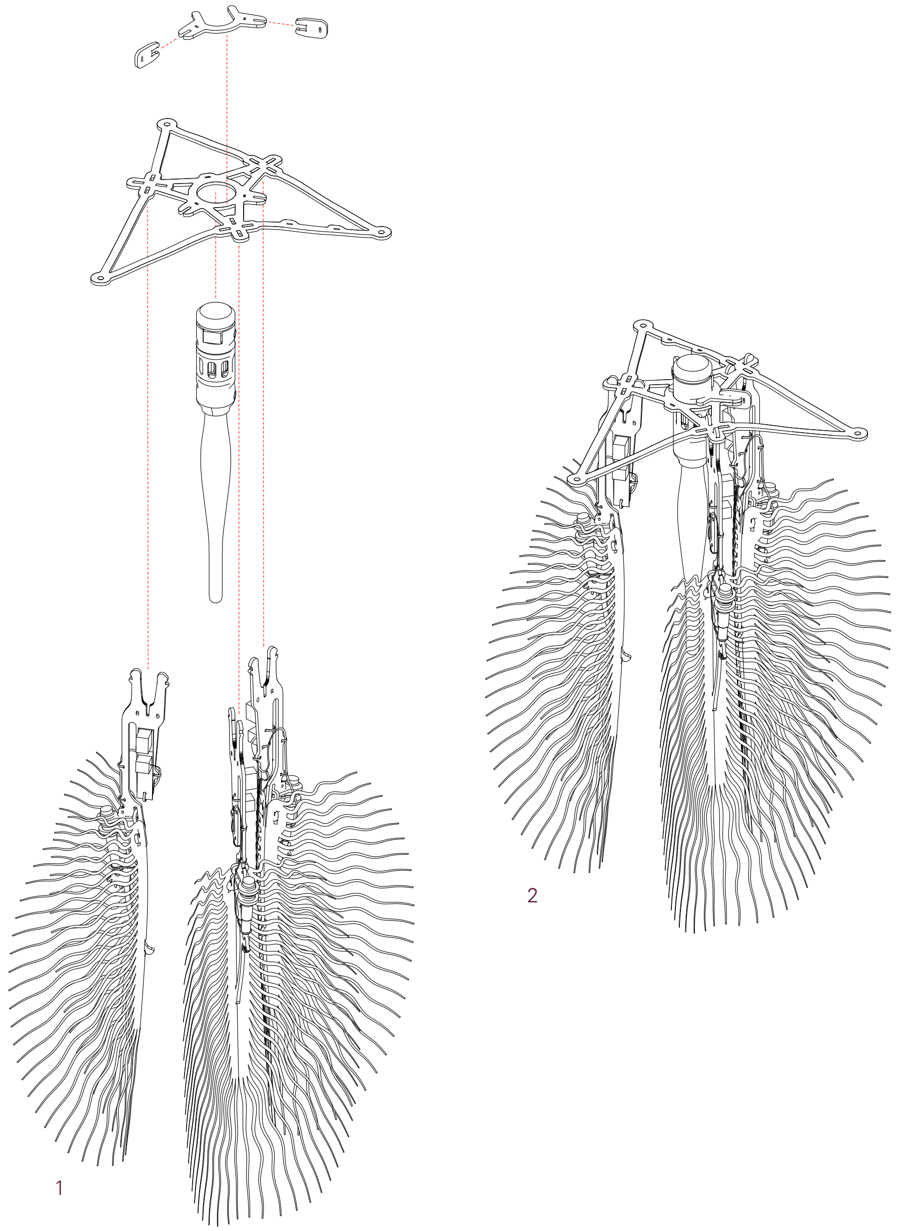
Crenellation Concave Unit Plan

Crenellated Veil Edge Assemblies



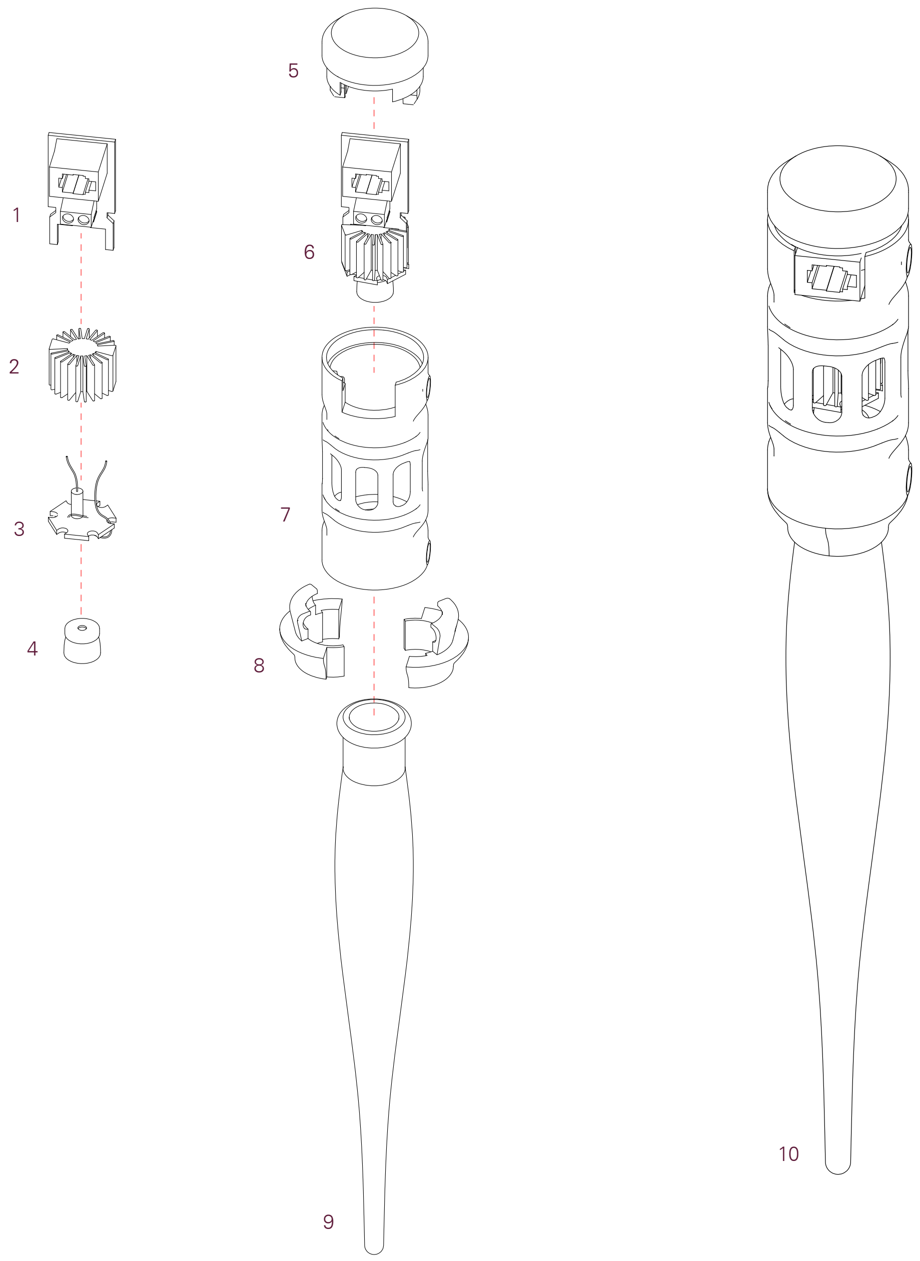
Microcontroller Nest

1 Nest Tri Clip 2 Nest Pent Plate 3 300mm X-plate 4 Nest Sled 5 Nest Standoffs and Cable Mount
6 Nest Dodecahedron Unfold 7 Nest Exploded View 8 Nest Assembled View



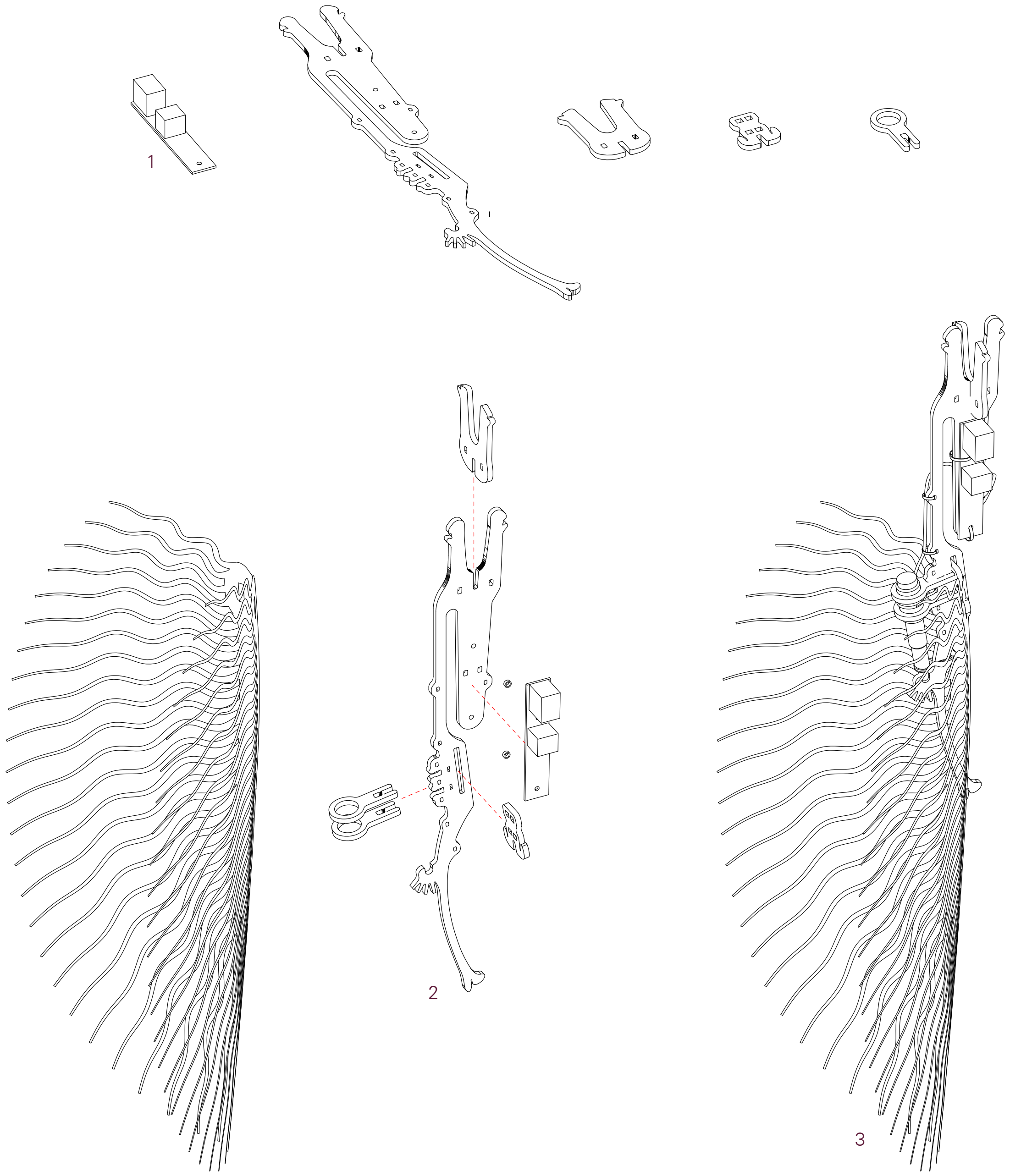
Cloud Cell with Light and Vibrating Fronds

1 Actuated Cloud Cell Exploded View 2 Actuated Cloud Cell Assembled View



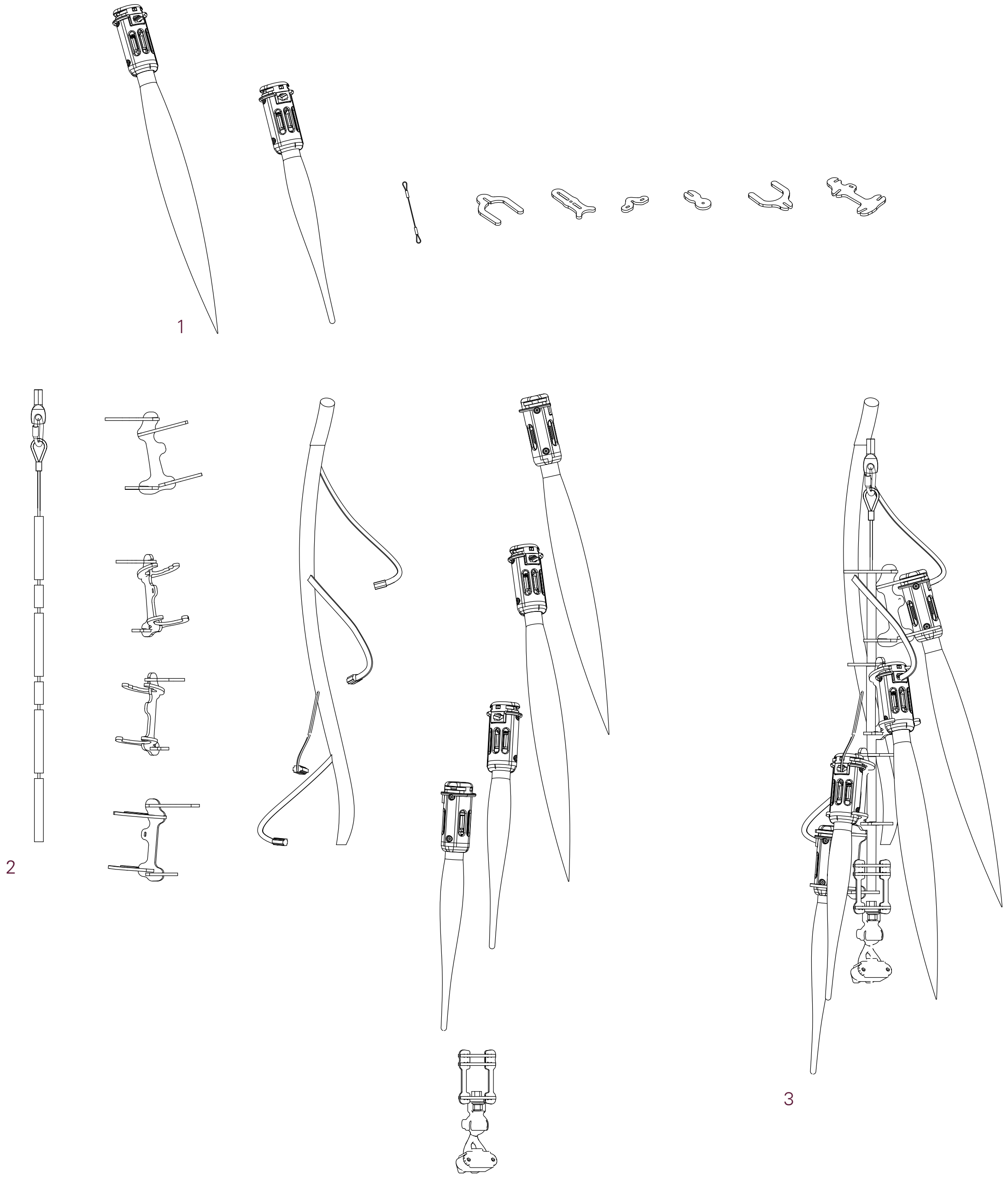
Glass Vessel

1 Jack Plate 2 Aluminum Heatsink 3 Rebel Star with Resistor 4 Self Adhesive Reflector 5 End Cap
6 Core 7 Body 8 Retention Ring Collar 9 Tapered Glass (Small) 10 Glass Vessel Assembled View



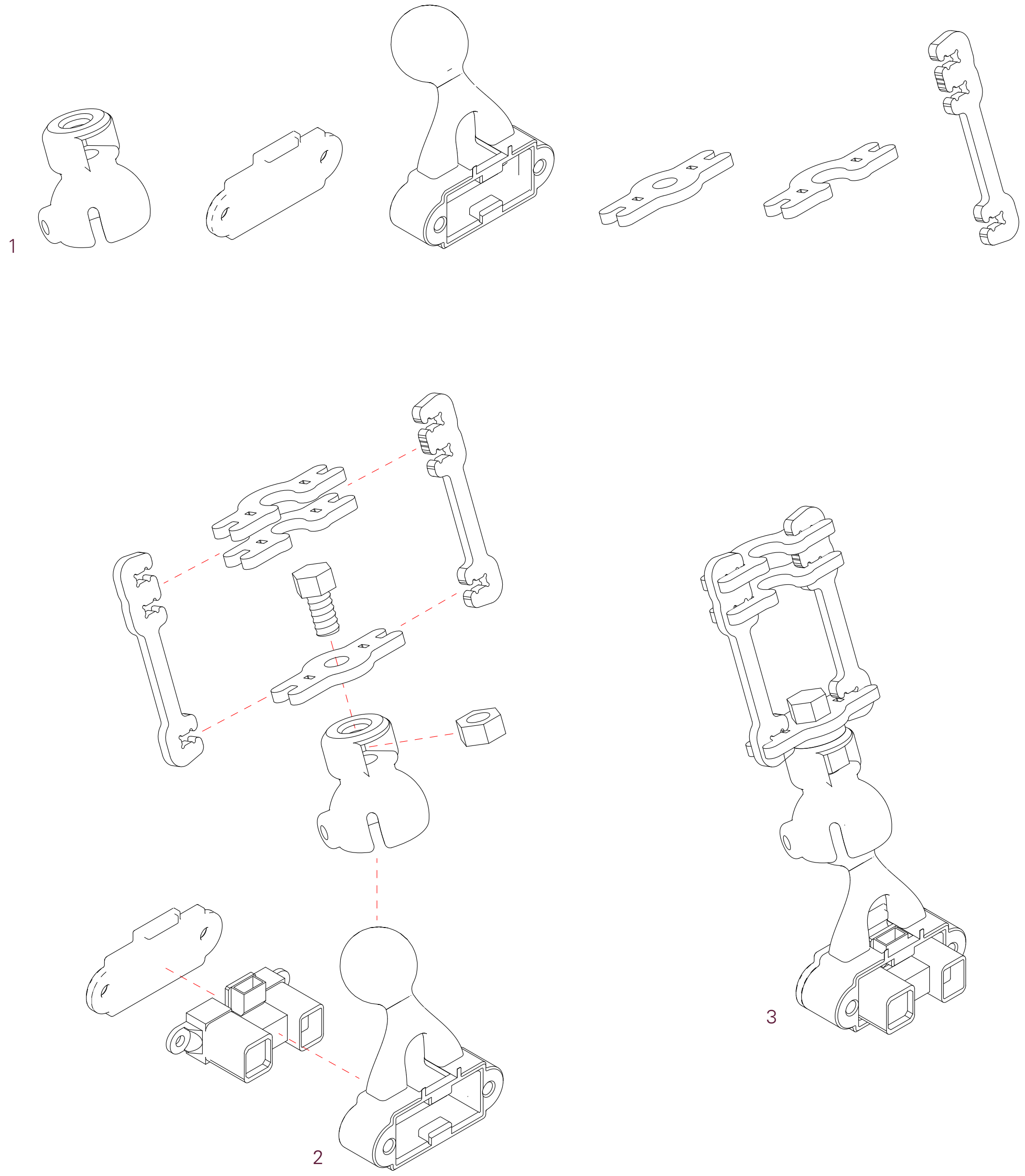
Vibrating Frond

1 Vibrating Frond Components 2 Vibrating Frond Exploded View 3 Vibrating Frond Assembled View



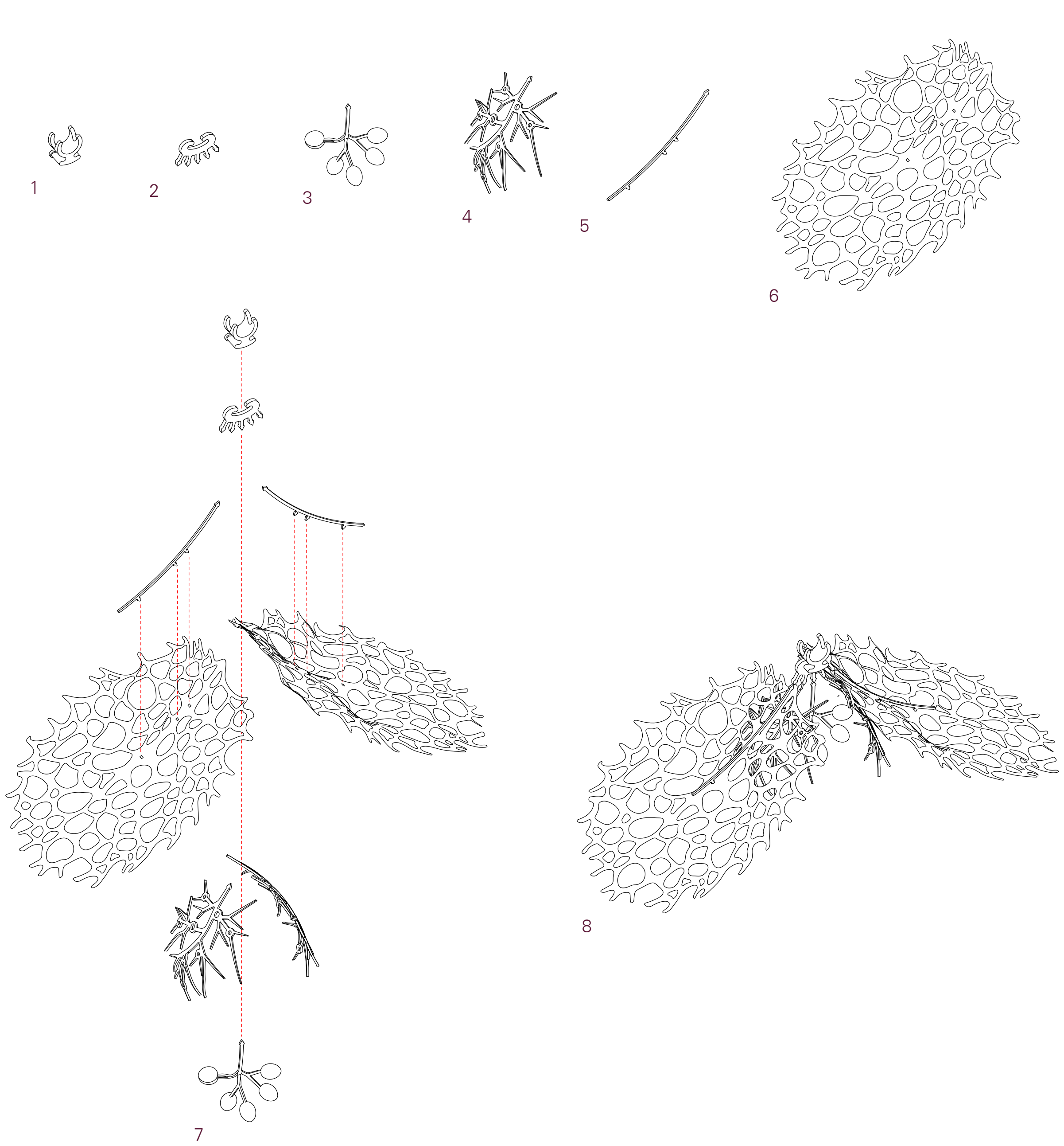
Actuated Chain with Filled Glass Vessels

1 Actuated Glass Chain Components 2 Actuated Glass Chain Exploded View 3 Actuated Glass Chain Assembled View



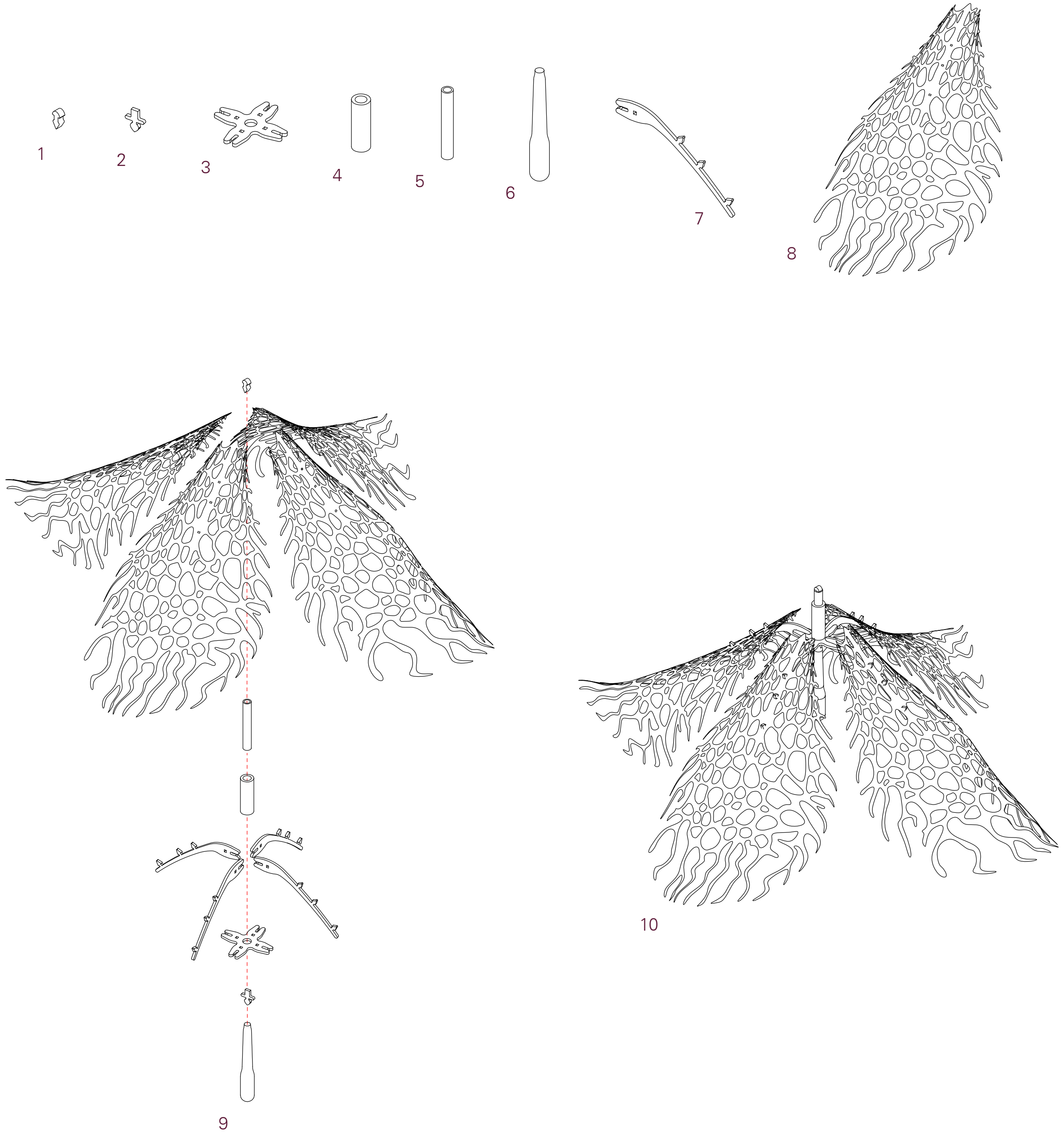
Sensor

1 Sensor Components 2 Sensor Exploded View 3 Sensor Assembled View



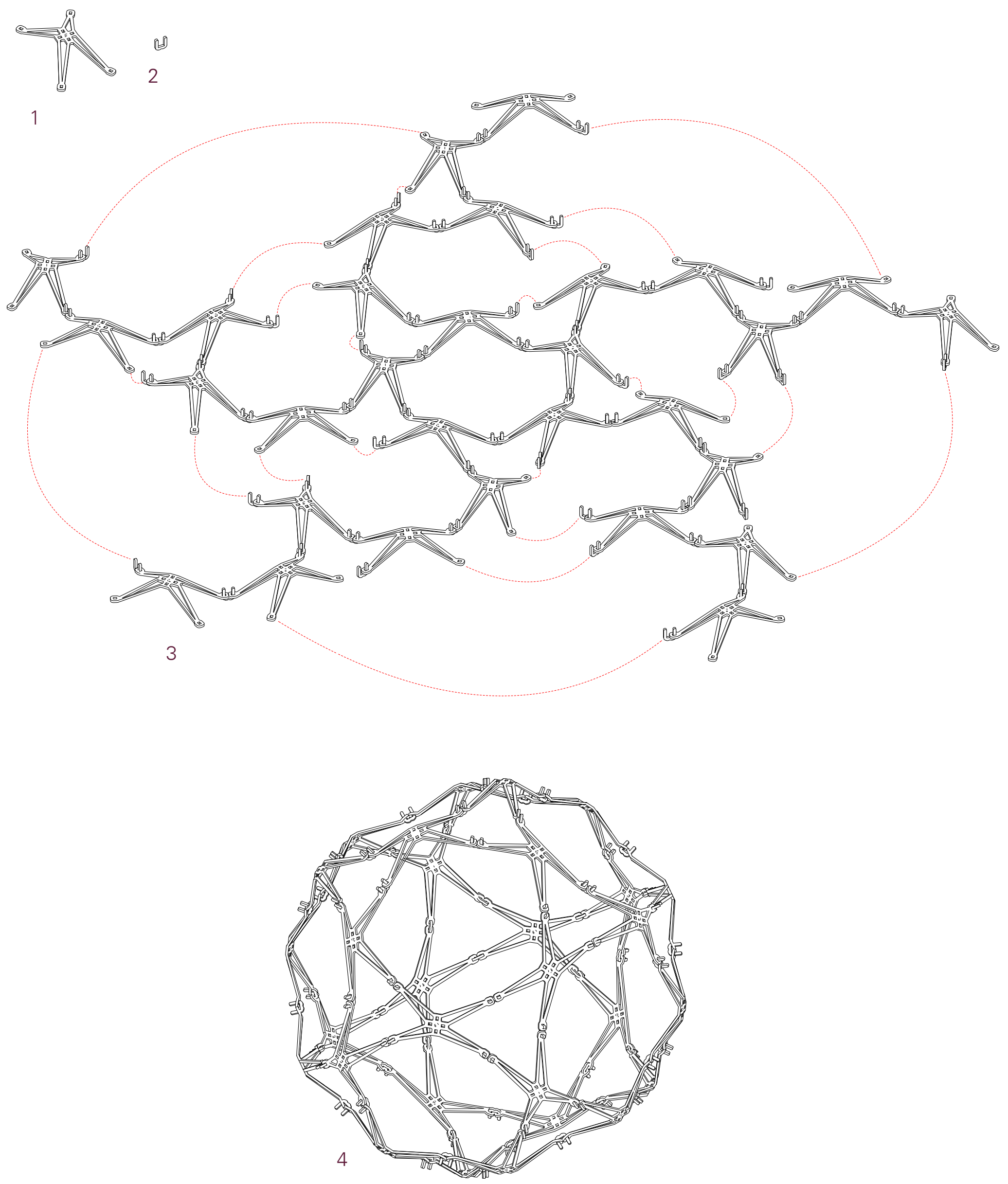
Frond Cluster with Cable Mount

1 Cable Mount 2 Frond Cluster Clip 3 Glass Branch 4 Spike Branch 5 Frond Arm
6 Frond 7 Frond Cluster Exploded View 8 Frond Cluster Assembled View



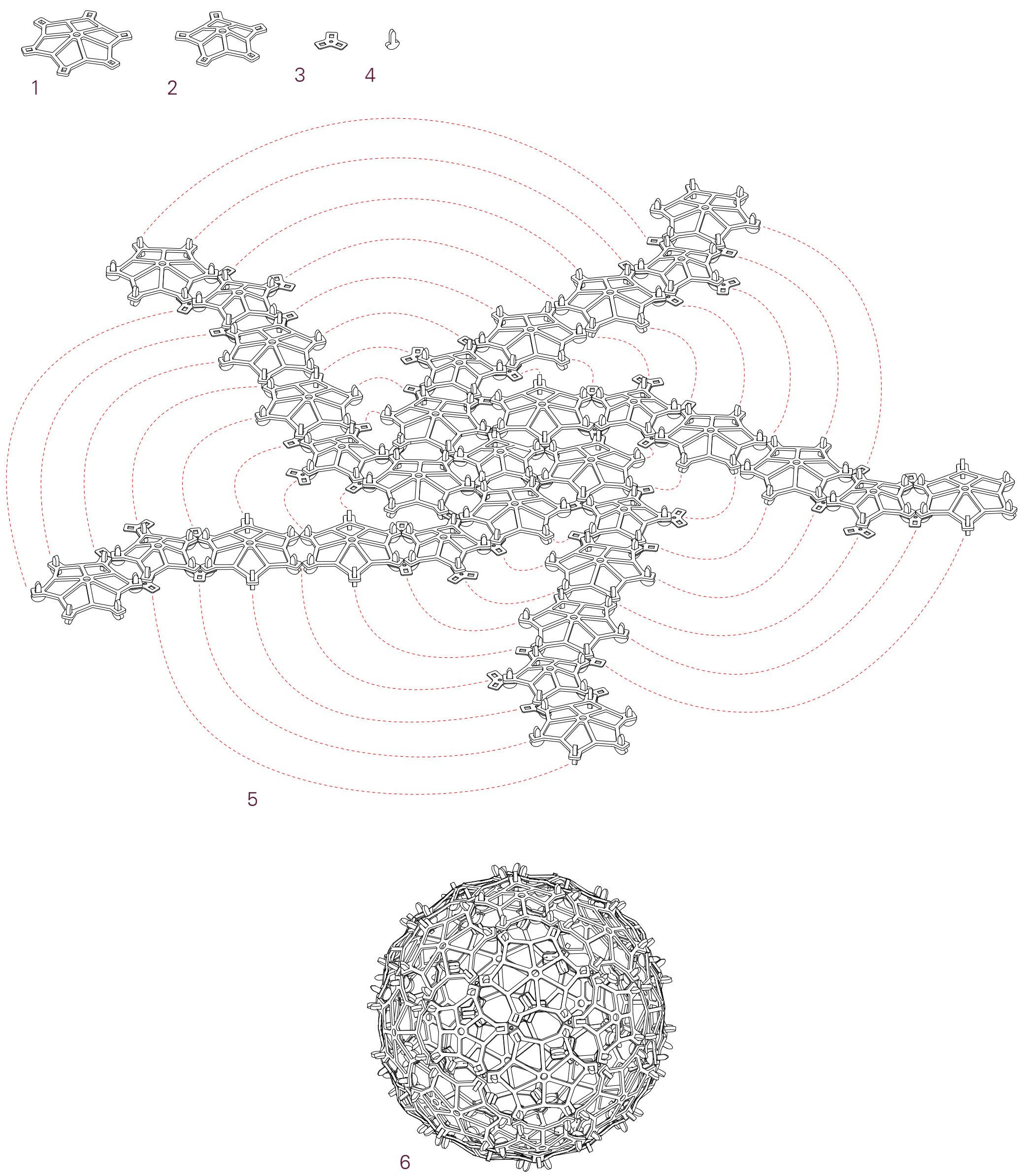
Quad Fronds

1 Mounting Clip 2 Bladder Clip 3 Quad Clip 4 PVC Tubing 5 Silicone Tubing 6 Tear Bladder
7 Frond Arm 8 Frond 9 Quad Frond Exploded View 10 Quad Frond Assembled View



Dodecahedron Sphere Dressing

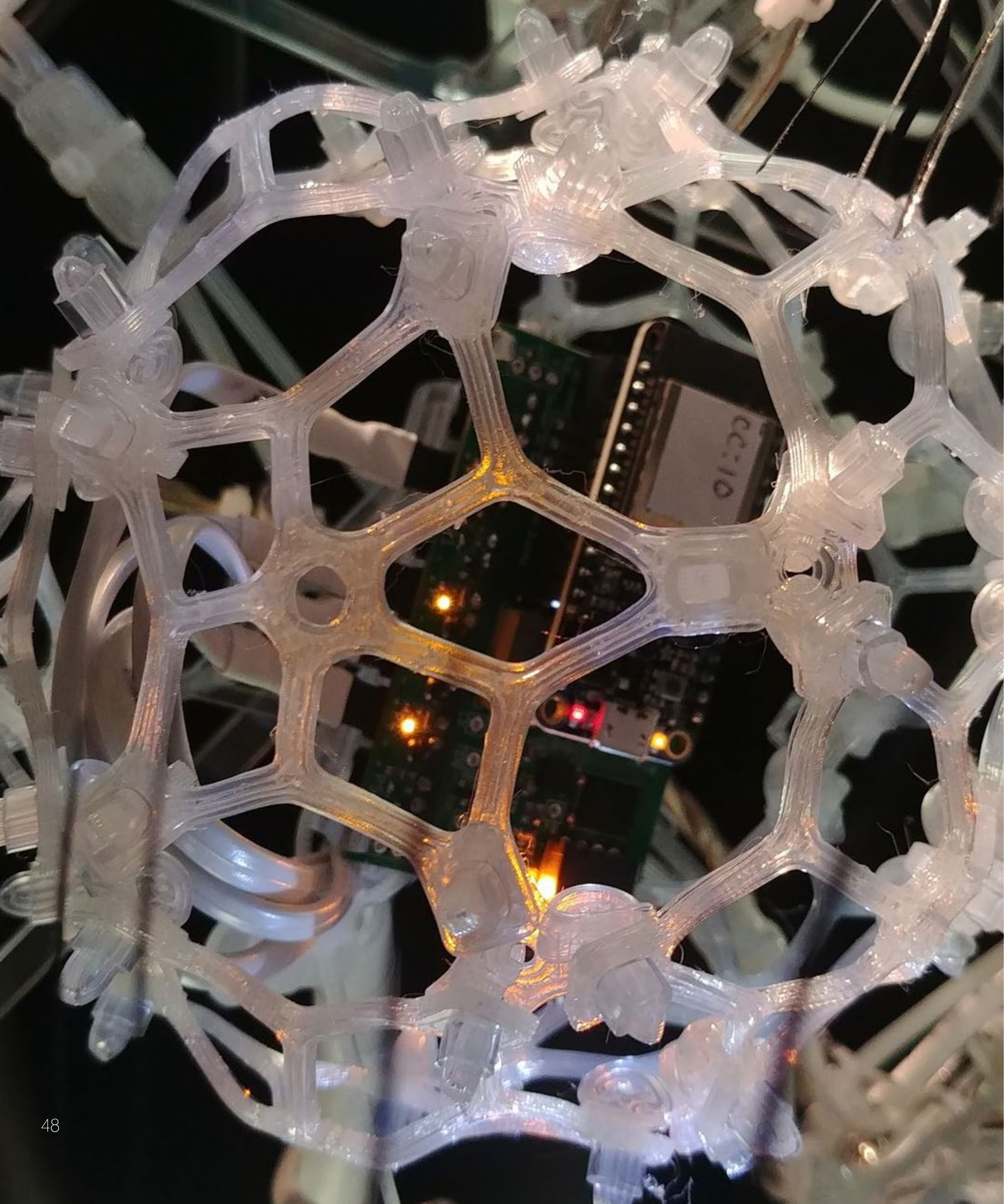
1 300mm X-plate 2 Staple 3 Dodecahedron Unfolded View 5 Dodecahedron Assembled View



Truncated Icosahedron Sphere Dressing

1 Hex Plate 3 Pent Plate 4 Tri Connector 5 T Pin

6 Truncated Icosahedron Unfolded View 7 Truncated Icosahedron Assembled View



Electronics Hardware

Facing Page

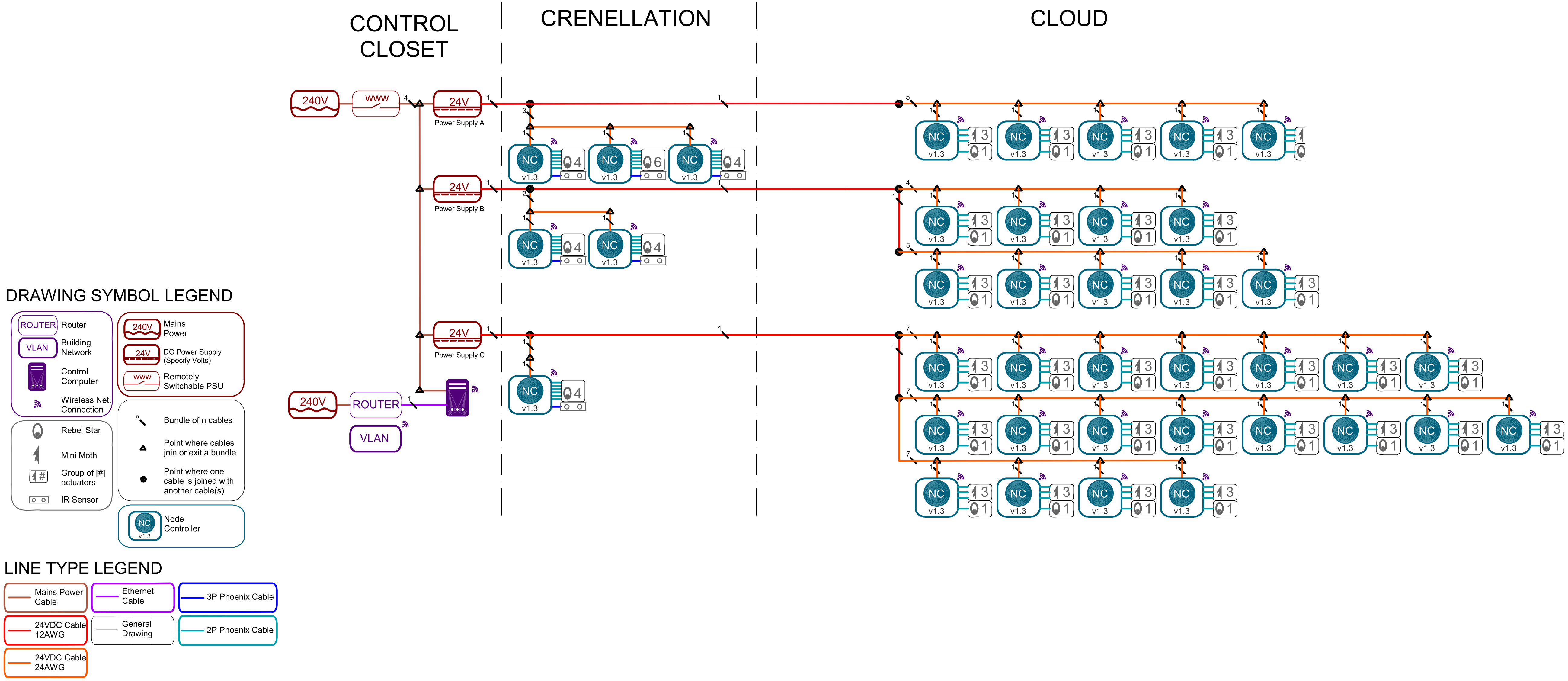
A Node Controller PCB embedded within a microcontroller nest in Poietic Veil

The electronics hardware appearing within the Poietic Veil prototype testbed features modular low-voltage circuitry that is designed to be easily extensible and modified. The system is based on a custom Smart Cell microcontroller. The Smart Cell system has been developed by Philip Beesley Studio Inc. (PBSI)/LASG in order to provide individual devices containing actuators and sensors with local “intelligence.”

Smart Cells take both physical and virtual forms. In parallel with individual physical devices that have their own embedded control profiles encoded within individual Smart Cell printed circuit boards, virtual devices within the firmware of microcontrollers are positioned to communicate across groups of Smart Cells. These virtual devices can be configured through an editor in a web browser on a personal laptop.

The electronics diagram appearing within this document represents electrical devices in the example cell system as 2D symbols. This notation can be used as shorthand for describing a larger concept design for a system.

Electronics Block Diagram



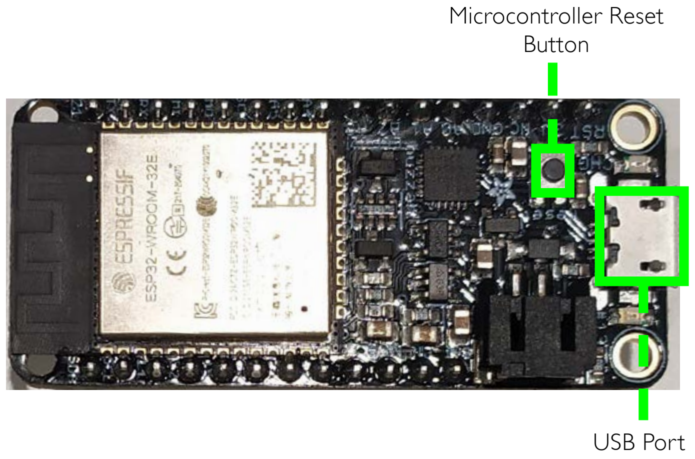
Node Controller

The Node Controller is a custom printed circuit board (PCB) that acts as a bridge between the microcontroller and the actuators and sensors. Connecting its USB port to a USB power supply (any phone power supply or battery pack) or a laptop/computer port gives the board power. Connecting it to a laptop also allows the user to upload Smart Cell profiles through the same cable. The Node Controller has on-board yellow LEDs that indicate when it is supplying power to an actuator through one of its 6 output ports. Two ports are also configured to support servo-type digital motors.

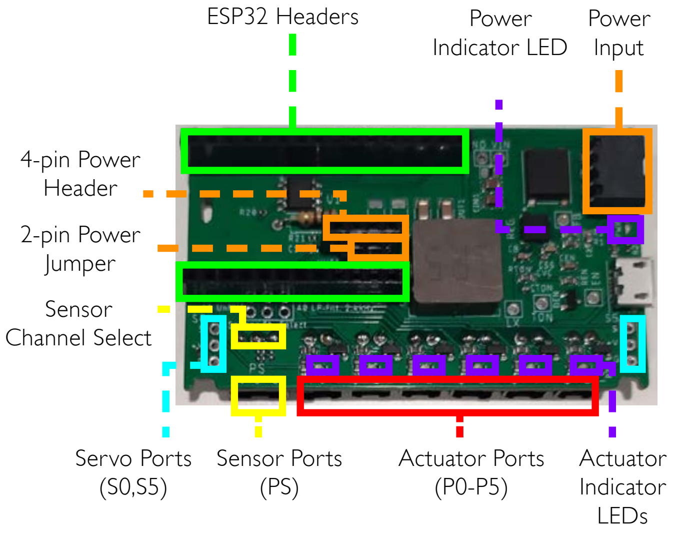
The PCB has one LED that comes on to confirm that the board is receiving power. It has one sensor port and a header for selecting the sensor channel. Both sensor channels pass through a voltage buffer, and one of the two additionally passes through a low-pass filter to eliminate high-frequency signals or noise.

While specialized cable connectors are used for input and output ports, they accommodate commonly available “hobbyist” Dupont-style socket-and-pin connectors. This creates compatibility with solderless and generic breadboard systems, supporting wide exploration by non-specialist users.

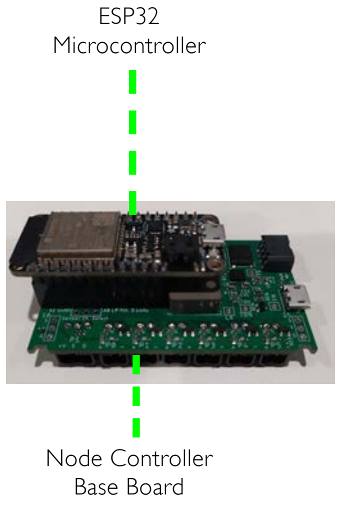
ESP32 Microcontroller



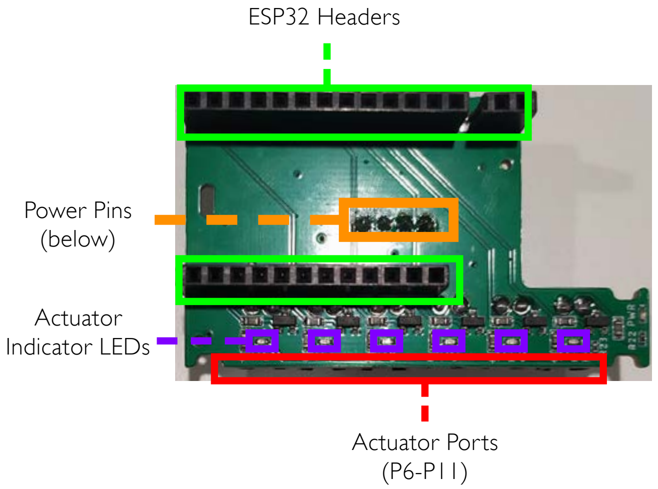
Node Controller Base Board



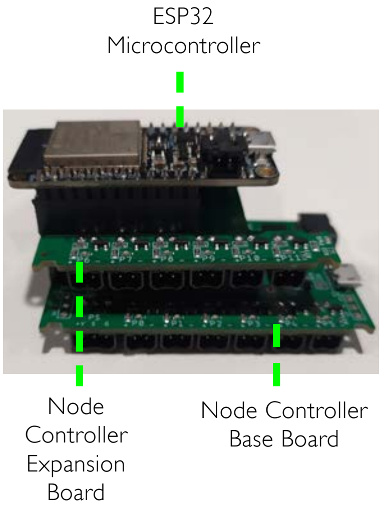
Base Configuration



Node Controller Expansion Board



Expansion Configuration



Behaviour Systems

Facing Page

An audience member interacts with Poietic Veil, as it attempts to learn from the individual's behaviour

The behaviour systems of LASG sculptures are divided into global (centralized) and local (distributed) parts. These two parts work together. The distinction between 'centralized' and 'distributed' behaviour is based on where decision-making takes place. These terms refer to the physical location and organization of decision-making software modules. For centralized behaviour the control computer is the decision maker, and for distributed behaviour a microcontroller (or in some cases, a Raspberry Pi embedded computer) is the decision maker. These two systems work together to choreograph a sculpture's expression.

The Science Centre Prototype at TU Delft represents a major step in the evolution of LASG behaviour systems, in that the logic of expression – that is, under what circumstances the sculptural elements generate behaviour – is far more decentralized than ever before. The microcontrollers distributed throughout the fabric of the sculptural canopy are responsible for their own outputs, based on shared information, parameters, and algorithms. Working in this way, they can create coordinated and responsive movements without needing constant flow of high-bandwidth information. Unlike a display screen, the individual units within the sculpture are only given information about high level environmental changes, and generate their own moment-to-moment output internally.

Global Behaviour: Influence Engines

At the heart of LASG's global behaviour system is the concept of Influence Engines. An influence engine is a piece of software that algorithmically generates 'influences' on the various spatially distributed parts of the sculpture. For example, a simple influence engine might create waveforms that move through the sculpture by determining the amplitude of a wave at various physical locations in the space. A different Influence Engine might use particle dynamics to calculate the trajectory of virtual objects which influence elements within the sculpture.

Key to the concept of Influence Engines is that they are not directly determining what each part of the sculpture should do, but rather 'influencing' the behaviour of the local computational elements. Elements can respond in many ways to the influences they are exposed to. For example, an actuator might glow in the presence of a virtual particle, or, that influence might dampen an already existing glow. This local logic of whether to respond to certain influences, how strong their impact becomes, and how exactly it becomes visible as part of the behaviour of the sculpture is inspired by independent natural lifeforms in a changing environment. Coordinated behaviour might occur if all the elements within an environment are similar (think about fireflies or crickets or ants) but the potential also exists for parts of the sculpture to respond independently as influences shift.

In the Science Centre Prototype at TU Delft, there are three main Influence Engines at work: SkyGen, a distributed parametric "weather system";Tendency, a probabilistic trigger system for local behaviour; and TouchPoint, a system for generating one-to-one impulses within the sculpture.These Influence Engines run simultaneously, and their parameters can be adjusted using simple web-browser-based controls. The responses of elements of the sculpture to the influences can be adjusted as well via a browser interface.

SkyGen Influence Engine

SkyGen is an Influence Engine that produces variable naturalistic 'cloud' behaviour within the testbed. Accessible via a web browser interface, a few simple control parameters enable a wide variety of patterns, evoking a spectrum of attitudes within the space.

Technically, the SkyGen system uses a well-known 'noise' algorithm called Simplex Noise, which generates a coherent 3D volume of 3D noise according to known parameters. Any point within this volume can be queried at any time, resulting in a spatially coordinated influence that feels like clouds. Each of the 40 microcontrollers that

Bottom Left

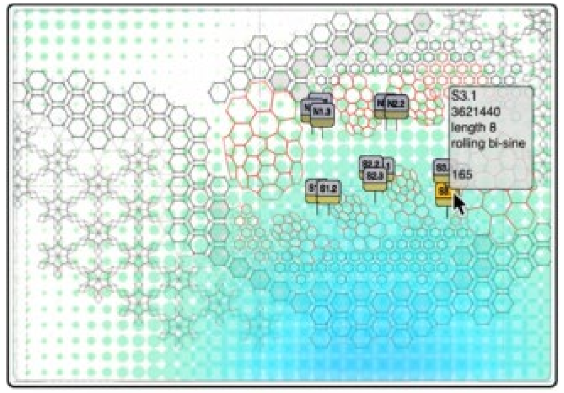
Physical Sculpture

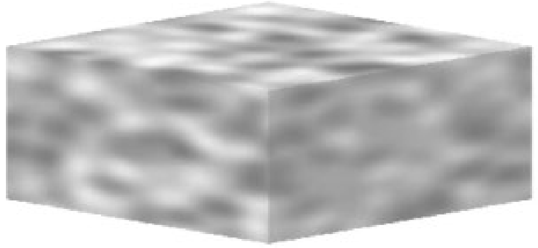
Bottom Right

Representation of the Sculpture in digital space with SkyGen Influence

Bottom

Three-dimensional slice through Simplex Noise Function





Simplex Noise Function (Perlin, Gustavson, etc.)

$P = (x, y)\ ,\ i = floor(x)\ ,\ j = floor(y)$

$g_{00} = gradient\ at\ (i, j),\ g_{10} = gradient\ at\ (i + 1, j)$

$g_{01} = gradient\ at(i,j{+}1),\ g_{11} = gradient\ at(i{+}1, j{+}1)$

$u = x - i,\ v = y - j$

$n_{00} = g_{00} \bullet \begin{bmatrix} u \\ v \end{bmatrix},\ n_{10} = g_{10} \bullet \begin{bmatrix} u-1 \\ v \end{bmatrix},\ n_{01} = g_{01} \begin{bmatrix} u \\ v-1 \end{bmatrix} \bullet\ ,\ n_{11} = g_{11} \bullet \begin{bmatrix} u-1 \\ v-1 \end{bmatrix}$

$f(t) = 6t^5 - 15t^4 + 10t^3$

$n_{x0} = n_{00}\,(1 - f(u)) + n_{10} f(u),\quad n_{x1} = n_{01}\,(1 - f(u)) + n_{11} f(u)$

$n_{xy} = n_{x0}\,(1 - f(v)) + n_{x1} f(v)$

make up the testbed knows the location of the actuators within the space that it controls, and they are synchronized via a clock signal sent from the server. Using a shared 'seed' value, shared algorithm and clock, each microcontroller efficiently manages the influence of SkyGen on its own actuators to keep them coordinated.

SkyGen provides an example of the scalability and efficiency of distributed processing vs. centrally choreographed behaviour.

Tendency Influence Engine

Tendency is a simple Influence Engine based on probability, with three parameters:

- Likelihood: How often a Smart Cell chain within the sculpture that is subscribed to this influence engine will trigger its local behaviour
- Strength Variation: How varied will be the strength of the trigger (with zero Strength Variation it will always be 100%, but it may be less as this parameter is increased.
- Time Variation: How fast will the Smart Cell profiles play out. Zero Time Variation will result in precise execution of defined profiles, and as this parameter is increased the runtime of triggered patterns will develop more variety, stretching or compressing to a greater degree.

These parameters can be controlled by a series of sliders on a browser-based interface, or linked to data feeds generated by other systems.

TouchPoint Influence Engine

The TouchPoint Influence Engine simply reports impulses generated by an external interface to all elements within the sculpture, with the following parameters:

- X, Y and Z location of 'touch' impulse
- Core Size in mm
- Full Size in mm
- Speed in mm/s

These parameters are received and acted on simultaneously by every element of the sculpture. The elements then calculate how to react. If their distance from the impulse location is greater than 'full size' range, the impulse will be ignored. If their distance is within the 'core size' the influence will be 100%. In between 'core size' and 'full size' there will be a linear decrease in the influence. Also, the response to this influence will be delayed in inverse proportion to the 'speed' parameter. That is, if the speed is slow, elements that are further from the impulse location will respond later than elements that are closer.

Though simple, this influence engine is capable of generating complex and varied behaviour such as ripples, waves and real-time 'paintbrush' effects.

A variety of interfaces can be imagined to control the four input parameters listed above, from a very direct manipulation of the sculptural map with a touch-screen GUI to a responsive system that generates impulse parameters based on the movement of individuals within a space, to a system for mapping impulses to external datasets for visualization.

Local Behaviour: Smart Cells

Complementing global Influence Engines, the Smart Cell system has been developed by PBSI/LASG in order to provide individual devices containing actuators and sensors with local "intelligence". Smart Cells are conceptual units that can store and play back time-based sequences of values, stored in a digital profile.

Smart Cells take both physical and virtual forms. Individual Smart Cells are physical devices that have their own embedded control profiles encoded within special-purpose printed circuit boards. Similarly, several virtual Smart Cells can be positioned within the firmware of microcontrollers. This is the approach used in the TU Delft Science Centre Prototype, with up to 4 Smart Cells in each of the 40 Node Controllers within the sculpture.

Smart Cells are configured using a custom made interface available online at:
https://smartcell.lasg.ca/

NOTE: This is a beta version of the software, and currently is not supported on all browsers or devices.

A web manual for the Smart Cell Software Interface can be found at:
https://smartcell.lasg.ca/manual.html

Whichever approach is used for Smart Cells, the profiles and arrangement of these virtual devices can be configured through an editor in a web browser, then uploaded to the hardware device.

When a Smart Cell receives a trigger, its two stored sequences are played simultaneously through its two outputs. These can be connected to different types of actuators, and are configured using the SAI Profile Editor.

For example, if the sequence is a sine wave, when the Smart Cell is triggered, a connected LED flashlight will get dimmer and brighter. If it is a series of pulses, the light will blink on and off. At a specified point in the sequence, the trigger will propagate, getting passed to any other Smart Cell connected as its children.

Smart Cells can receive these triggers in two ways:

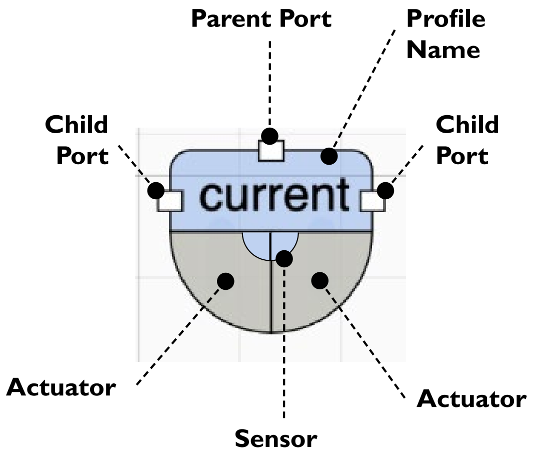
- From environmental stimuli (Influence Engines)
- From an attached physical sensor. In the TU Delft Science Centre Prototype, infrared proximity sensors are used to detect interaction, triggering Smart Cell profiles on the 'chain' components of the testbed.

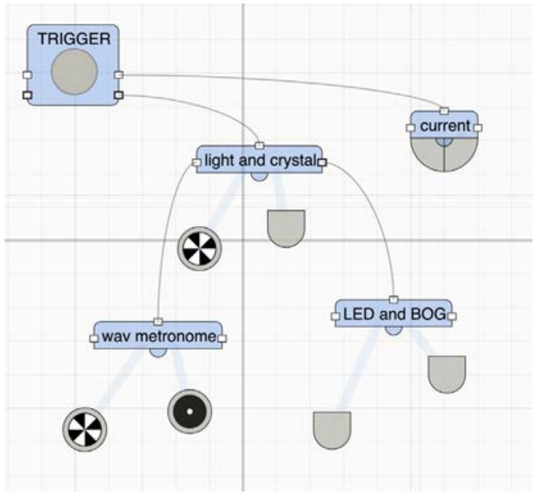
Bottom Left

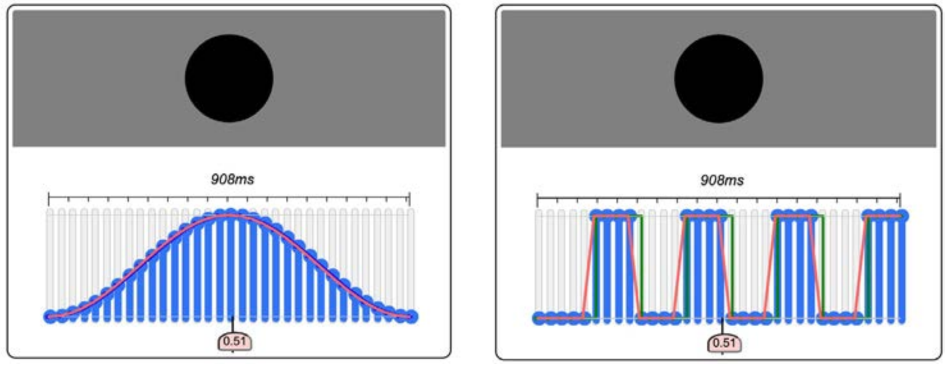
Diagram of Smart Cell Object in configuration window

Bottom Right

Example of smart cells connected in configuration window of smart cell editor webpage







Smart Cell Implementation: Physical or Virtual?

While the sensors and outputs are real in both cases, the Smart Cell and all its capabilities can be either physical or virtual. A physical Smart Cell is a small circuit board with one micro-USB parent port, two micro-USB child ports, two actuator output jacks and a sensor input jack. A tiny microcontroller runs firmware which stores the profiles and reacts to triggers by playing the sequences to its outputs, then triggering its children.

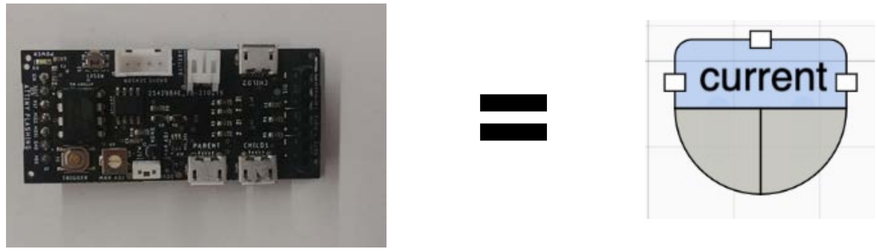
A virtual Smart Cell uses the same firmware, running on a more powerful microcontroller that can effectively "host" multiple Smart Cells at the same time. They are still connected together using parent and child ports, just like several physical Smart Cells could be, but their 'wiring' is virtual. Each virtual Smart Cell can store a unique profile and control different kinds of actuators. This is the method used in this testbed, with three virtual SAIs running on a node controller.

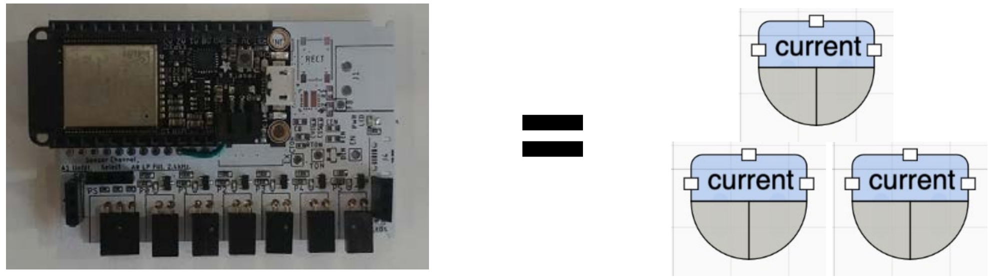
Top

Smart Cell profile editor - output 1(left), output 2 (right)

Bottom

1 smart cell board is equivalent to 1 smart cell virtual object





Top

1 node controller board is equivalent to 3 smart cell virtual objects

Bottom

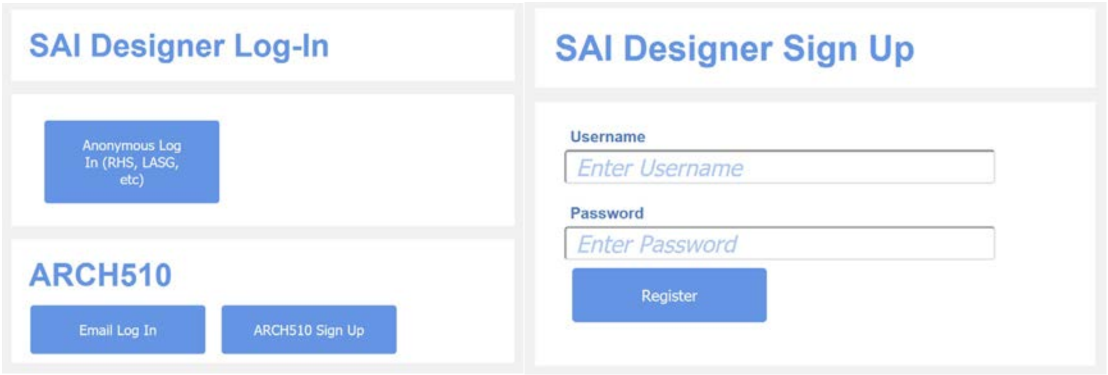
Smart Cell webpage login screens

Smart Cell Editor Software

The Smart Cell Editor software is a tool for creating profiles for an Smart Cell creating virtual arrays of devices to plan large systems. The editor can be used to upload configuration data to a microprocessor-based Node Controller, which hosts the virtual Smart Cells.

Browser Based Interfaces and Tools

The online interface, the SAI Designer, is a web page that provides a number of tools and views to help the user create small local behaviours and couple them together to create larger behaviour systems.


SAI Designer Log-In

ARCH510

SAI Designer Sign Up

Username

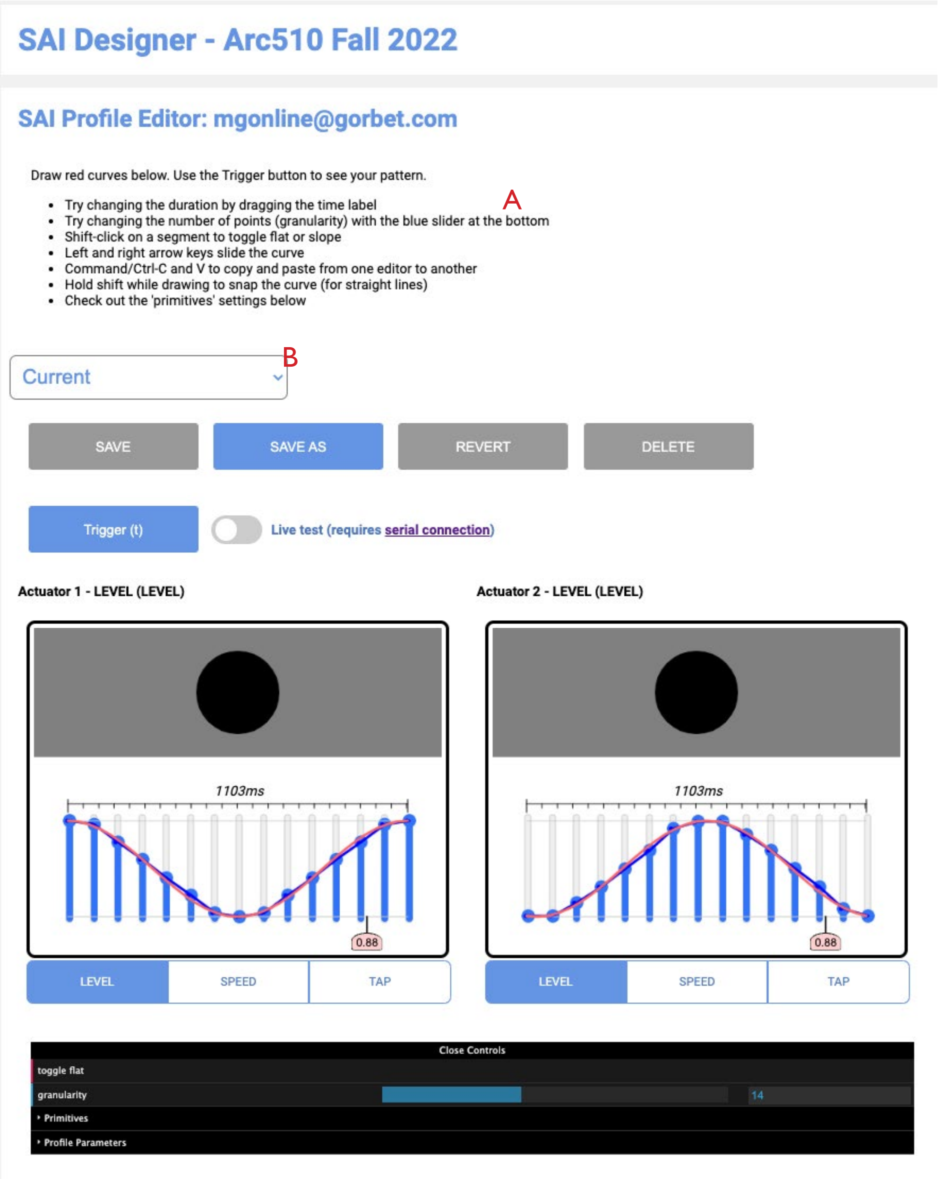
Enter Username

Password

Enter Password

Actuation Profile Previews

While designing the two output profiles for a Smart Cell, a preview is shown on an amplitude-time cartesian plot. These give the designer an approximate sense of the appearance of the output.



SAI Designer - Arc510 Fall 2022

SAI Profile Editor: mgonline@gorbet.com

Draw red curves below. Use the Trigger button to see your pattern.

- Try changing the duration by dragging the time label
- Try changing the number of points (granularity) with the blue slider at the bottom
- Shift-click on a segment to toggle flat or slope
- Left and right arrow keys slide the curve
- Command/Ctrl-C and V to copy and paste from one editor to another
- Hold shift while drawing to snap the curve (for straight lines)
- Check out the 'primitives' settings below

Current

Live test (requires serial connection)

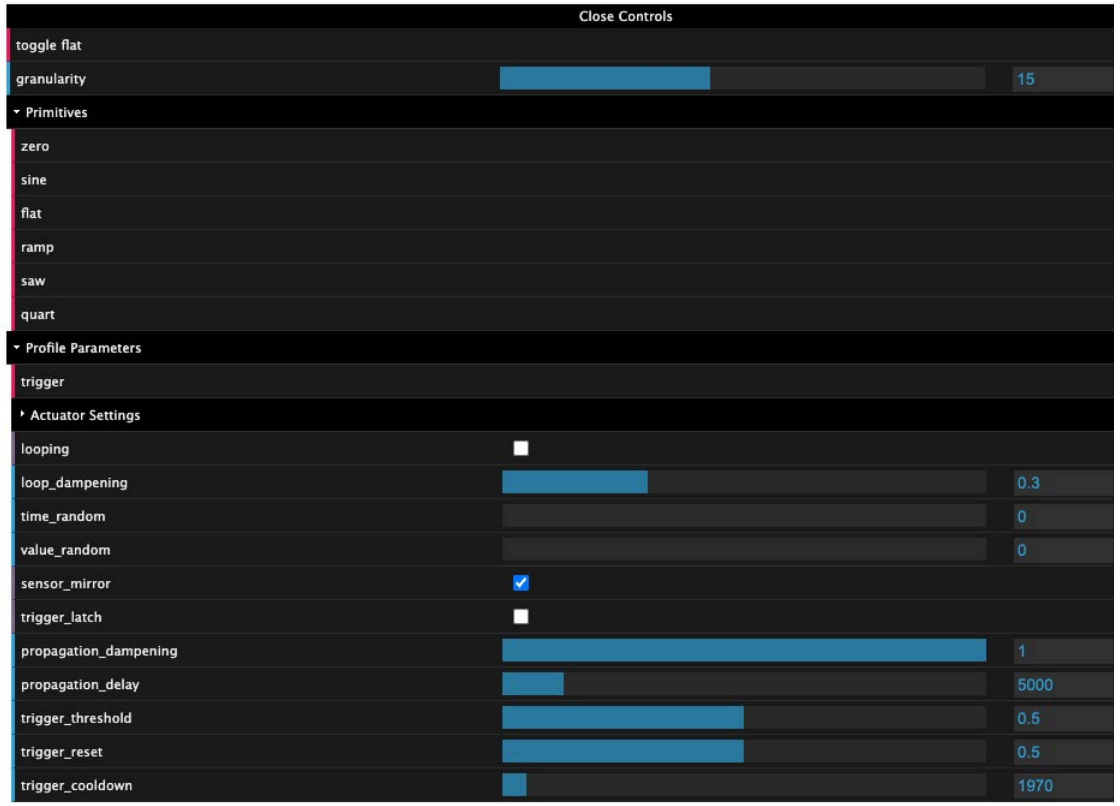
Actuator 1 - LEVEL (LEVEL)

Actuator 2 - LEVEL (LEVEL)

toggle flat

granularity 14

Profile Settings



Close Controls

toggle flat

granularity 15

▾ Primitives

zero

sine

flat

ramp

saw

quart

▾ Profile Parameters

trigger

▸ Actuator Settings

looping □

loop_dampening 0.3

time_random 0

value_random 0

sensor_mirror ☑

trigger_latch □

propagation_dampening 1

propagation_delay 5000

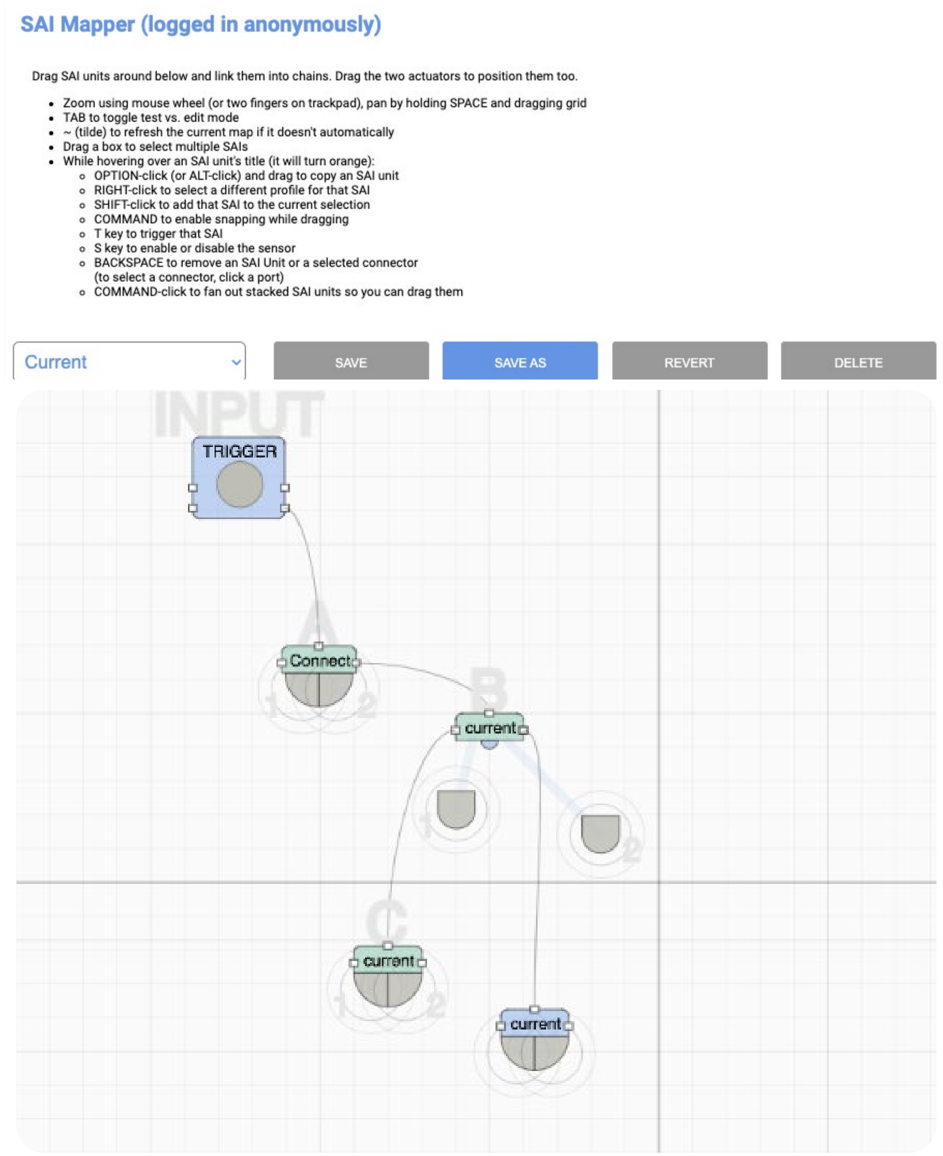
trigger_threshold 0.5

trigger_reset 0.5

trigger_cooldown 1970

The parameters that define the output profiles and sensor response are configured using a set of sliders and check boxes. These provide the ability to finely adjust an individual Smart Cell's behaviour, and create a saved configuration that can be uploaded to real devices.

Group Behaviour Designer



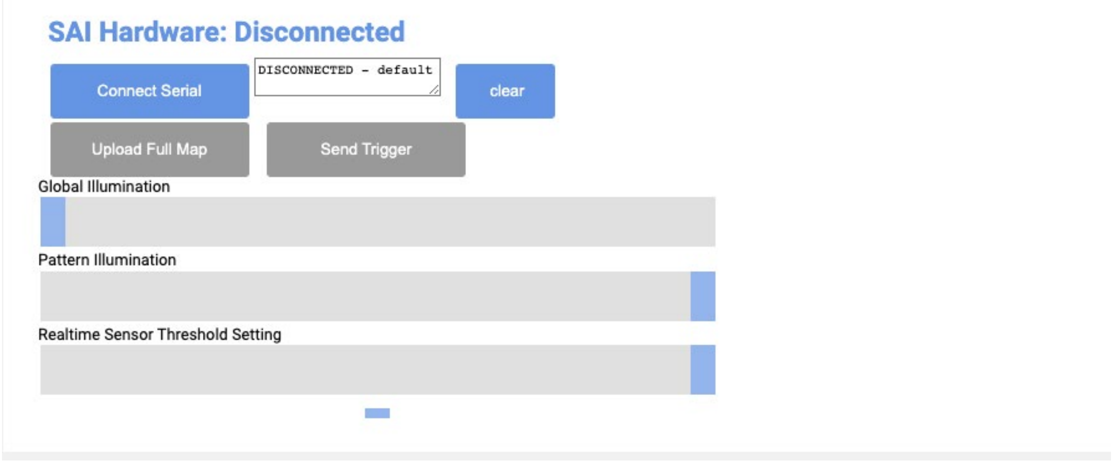
SAI Mapper (logged in anonymously)

Drag SAI units around below and link them into chains. Drag the two actuators to position them too.

- Zoom using mouse wheel (or two fingers on trackpad), pan by holding SPACE and dragging grid
- TAB to toggle test vs. edit mode
- ~ (tilde) to refresh the current map if it doesn't automatically
- Drag a box to select multiple SAIs
- While hovering over an SAI unit's title (it will turn orange):
    - OPTION-click (or ALT-click) and drag to copy an SAI unit
    - RIGHT-click to select a different profile for that SAI
    - SHIFT-click to add that SAI to the current selection
    - COMMAND to enable snapping while dragging
    - T key to trigger that SAI
    - S key to enable or disable the sensor
    - BACKSPACE to remove an SAI Unit or a selected connector
      (to select a connector, click a port)
    - COMMAND-click to fan out stacked SAI units so you can drag them

Virtual Smart Cell objects can be arranged in a 2D workspace and connected together to create complex cascading behaviour systems. This interface gives a designer a sense of how a larger interconnected system will behave.

Physical Device Interface



SAI Hardware: Disconnected

DISCONNECTED - default

Connect Serial: Establishes a connection between your computer and the Node Controller / Smart Cells.

Upload Full Map: Uploads the A, B, and C Smart Cells and their profiles to the Node Controller.

Send Trigger: Sends a trigger. Any Smart Cells that are connected to the /trigger box/ in the mapper will be triggered.

Global Illumination: Turns on all /level/ actuators on all Smart Cells.

Pattern Illumination: Controls the overall intensity of all the Smart Cell outputs while they are playing sequences.

Threshold: Adjusts the sensor threshold in realtime. The small indicator below the slider shows you what it's set to in the current profile, so you can adjust it back.

Appendix: Notes on Software and Communication Protocols

The following subsections describe the distributed system of Node Controllers in the TU Delft Science Centre Prototype. These node controllers use ESP32 Arduino microcontrollers.

Wi-Fi Connectivity

The testbed's Node Controllers (NCs) use credentials stored in a credentials header file in their firmware. This file lists multiple SSID/password pairs for access points they might connect to. On startup, the device scans available networks, and if one of the named SSIDs is available it connects to that one. If the NC does not yet have an assigned name, it will connect to the network via DHCP and, once subscribed to the MQTT broker so that it knows the address of the server, it will send an OSC command to the server requesting a name. An interactive naming process can be triggered to make the NC flash its LED, at which point the name can be entered. The server will send the new name and ID if this name is recognized as part of the sculpture. Once sent a new name, the NC will write this to flash memory and reboot so that it can reconnect with a static IP derived from the name (see below).

IP Setup

Devices in the testbed will go through the above process to have a name assigned consisting of a 2-letter code (e.g., CC or CH) and a number, in the format "CC:3" or "CH:2". The CC determines if the device will be a Cloud Canopy (CC) or a Chain (CH) node. Devices determine their static IP based on this designation. For example, IPs in the range of .100 - .149 are "CC", IPs in the range of .150 - 160 are "CH". The first three octets are determined by the network credentials stored in the header file.

The primary server device (control computer) will always have an IP ending in .99. Any peer controllers such as a sound or media server, video player, visualization engine, etc. will have IPs counting down from there: .98, .97, etc.

DHCP will assign temporary dynamic IPs to new devices like laptops or phones in the range of .2 to .79.

Communication

Devices in the testbed communicate via three protocols: MQTT, OSC, and Telnet. Telnet is used to output console messages remotely and is accessed using the device's IP address (e.g. "telnet 192.168.13.105").

MQTT is used to subscribe to short messages that are not time sensitive. These include configuration and positioning messages that help devices know where they are, what the state of the testbed is when they first join, and how to communicate with the testbed's server. The MQTT broker's address and credentials, if applicable, are stored in the NC's firmware via the credentials file.

These topics and messages are as follows:

MQTT Messaging		
Branches & Topics	**Format**	**Description**
/server/#		This subscribes to any messages related to the setup of the server, sent as subtopics with their own formats.
/server/OTA_update_path	String	A full path to a .bin file that the Node Controllers (ESP32) can use to update their firmware, on request.
/server/server_IP	String	The IP address of the server. This is useful when new nodes first join the network so they know where they can send information about themselves
/lasd/<nodeName>/#	<nodeName> is for example, "CC:3" or "CH:0"	This subscribes to any messages related to the configuration of specific nodes in the sculpture, sent as subtopics with their own formats.
/lasd/<nodeID>/position	JSON string with keys: X, Y, and Z	X, Y, and Z are integers measured in mm from the origin of the testbed.
/lasd/<nodeID>/devices/#	JSON string with keys: device_type, X, Y, and Z	The topic on the '/devices' branch will be the name of the actuator (i.e.: MO2 or RS0). Its position and type are indicated in the JSON payload.

MQTT Messaging		
Branches & Topics	**Format**	**Description**
/influences/#		This subscribes to messages about the state of any influences, sent as subtopics with their own formats
/influences/sky	JSON string with keys: nw, ww, tb, it, and cn	The state of the "SkyGen" influence engine. Numbers are floats of various ranges. Along with the X/Y/Z position and clock value, this can be used to determine accurate influence levels for any device in the testbed. • nw: North Wind • ww: West Wind • tb: Turbulence • it: Intensity • cn: Contrast

OSC (delivered via UDP on port 1212) is used for subscribing to real-time messages that do not have guaranteed delivery. This enables clock synchronization, real-time updates to changes in influence engines, and one-time commands such as reboot, firmware update, etc.

OSC Messaging		
Addresses (commands)	**Format**	**Description**
/skyParams	JSON string with keys: nw, ww, tb, it, and cn	The state of the "SkyGen" influence engine. Numbers are floats of various ranges. Along with the X/Y/Z position and clock value, this can be used to determine accurate influence levels for any device in the testbed. • nw: North Wind • ww: West Wind • tb: Turbulence • it: Intensity • cn: Contrast

/skyClock	2 integers: base, decimals Eg: 2624338 | 7736 concats to: 2624338.7736	2 integers, to be combined to make a number of format <base>.<decimals> for use in the SkyGen noise algorithm. Represents time since Jan 1 2022, and should grow by .0001 each millisecond. (full number is not actual seconds, but seconds /10)
/skyPrintXYZ	X, Y, Z as ints	Causes the node controller to output current results of the SkyGen noise algorithm given its position.
/setID	2 strings: name, id	Used by the server to tell the node controller its name (eg. “CC:4”) and unique ID (eg. “F23BA2”) during setup.
/OTAupdate	[none]	Triggers an update of the node controller’s firmware (using the path specified by MQTT)
/reconnectMQTT	[none]	Compels the node controller to attempt to reconnect to the MQTT server
/blinkFlag	Integer: 0, 1 or -1	Causes the onboard LED on the node controller to blink. 1 activates, 0 disactivates, -1 toggles.
/restart	[none]	Instantly restarts the node controller
/writeRemoteBytes	String: recipient ID String:JSON string with keys: type, data Int: length of packet	Used by the SAI system for all communication. First string is the ID of recipient - important if there are several devices sharing in IP address, such as a raspberry pi with several serial chains of SAIs. Ignored by ESP32 node controllers. Second string is JSON-formatted result of “JSON.Stringify()” on a Byte Array. ‘type’ should always be ‘Buffer’, and ‘data’ should always be an array of 8-bit values representing bytes (eg. [250, 255]. Int is simply the length of the byte array, for easier parsing.

SkyGen Algorithm and Libraries

To determine a real-time value for the Sky influence engine, devices use the following function, which outputs a result using the Simplex3D Noise algorithm:

C++:

```
double value = skyNoise.generateNoise(
           ((double)x / elementSize) -  westwind  * skytime   /
           mapRange(intensity+90., 0., 100., 1000., 10.),
           ((double)y / elementSize) +  northwind * skytime   /
           mapRange(intensity+90., 0., 100., 1000., 10.),
           (0 + turbulence * skytime ));
```

Javascript:

```
let v = this.skyNoise.generateNoise(
      (x -  (skyg.settings.westwind * this.time  )   ) /
      map(skyg.settings.intensity+90, 0, 100, 1000, 10),
      (y +  (skyg.settings.northwind * this.time )  ) /
      map(skyg.settings.intensity+90, 0, 100, 1000, 10),
      (0 +  (skyg.settings.turbulence * this.time) )) ;
```

The noise library used for the simplex noise function has been ported directly from Javascript to C++, and uses a common seed. No seed needs to be passed.

Library files in java and C++ are available here:
https://3.basecamp.com/3601494/buckets/28841918/vaults/5218964619

Credits

Project Leads

Philip Beesley
Timothy Boll
Matt Gorbet
Lisa Jiang
Michael Lancaster

LASG Executive

Rob Gorbet
Anne Paxton
Rekha Ramachandran
Alison Thompson

LASG Design & Production

Alexandros Angelidis
Adrian Chîu
Kevan Cress
Nicolas Désilles
Sebastián González Álvarez
Isabella Ieraci
Chris Kang
Lucia Kempe
Mike Nopper
Abida Rahman

TU Delft Interactive Environments Minor Students

Ada Fabrykiewicz
Benjamin van Schaik
Diana Lore De la Vega Valdés
Eli Hommes
François Andrés
Ha Yeun Kim
Ian Lin
Izzy Mico
Jafet Koeckhoven
Jeroen Heijmans
Joris Mooij
Julia Kleinwächter
Laura Arango Mejia
Laura Martinez Quesada
Leonor Falcão
Linjing Wu
Nika Umnov
Doruk Kırbeyi
Olaf Kamperman
Pablo Yániz González
Rita J. Gorriz Salanova
Sasha Kiselev
Sophie Timmerman
Stijn Gruben
Tasmin Whittle
Tieme van Wijnen
Tijmen Tonino

TU Delft Interactive Environments Minor Staff

Aadjan van der Helm
Wim Schermer
Martin Havranek
Govert Schilling
Dieter Vandoren
Justin Oosterbaan
Caspar Krijgsman
Vivian Nguyen
Nathan Douenburg
Doreen Mulder
Adriaan Bernstein

TU Delft Science Center Staff

Angela Hanna
Teun Verkerk
Jules Dudok
Michael van der Meer
Max Mahieu

4DSOUND

Poul Holleman
Joris Takken
Roman Willems

References

Salem, Ben, Jorge Alves Lino, and Jan Simons. "A Framework for Responsive Environments." In Ambient Intelligence, edited by Werner Weber, J. M. Rabaey, and Emile H. L. Aarts, 263–77. Berlin: Springer International Publishing, 2017.

Youngs, Amy M. "The Fine Art of Creating Life." Leonardo 33, no. 5 (2000): 377–80.