CNets: An Interactive Design Framework for Cone-Nets

Klara Mundilova[*,a], Michele Vidulis[a], Quentin Becker[a,b], Florin Isvoranu[a], Mark Pauly[a]

[a] Geometric Computing Laboratory, EPFL, Switzerland

[b] The University of Tokyo, Japan

Advances in Architectural Geometry 2025

Abstract

Planar quadrilateral (PQ) meshes play an important role in Architectural Geometry, with ongoing research focused on developing effective tools for their design. *Cone-nets* are a special class of regular PQ meshes in which one family of PQ strips forms discrete (projective) cones. This paper builds on recent advances in the study of cone-nets and presents a constructive implementation in an interactive design tool that enables intuitive real-time exploration of the design space of cone-nets within the Grasshopper/Rhino environment. We provide novel theoretical insights into cone-nets, introduce a user-friendly interface, and incorporate a correction mechanism to repair degenerate configurations. Our aim is to make cone-nets more accessible to the broader community of practitioners and to promote their application in advanced architectural design scenarios.

1 Introduction

PQ meshes (quadrilateral meshes with planar faces) are a fundamental tool in architectural geometry. They are particularly suitable for modeling freeform facades, where the planarity of faces ensures cost-effective cladding, e.g., by glass panels. As discrete analogs of conjugate parameterizations, PQ meshes also provide direct links to differential properties of surfaces, which has led to a rich volume of research in discrete differential geometry [1].

Special classes of PQ meshes are particularly relevant in architectural design as they directly relate to crucial benefits of construction. For example, conical meshes, where all face planes incident on a vertex are tangent to a right circular cone, allow for planar face offsets at constant distance and torsion-free support structures [7]. Circular meshes, i.e., PQ meshes where each face has a circumcircle, are a discrete counterpart of principal curvature parameterizations and allow for planar vertex offsets [13].

In this paper, we propose new constructive design techniques and interactive tools for modeling with discrete cone-nets. Not to be confused with conical meshes, *discrete cone-nets*, introduced by Kilian et al. [6], are regular PQ meshes in which one family of quadrilateral strips lies on discrete cones or cylinders

*Corresponding author: klara.mundilova@epfl.ch

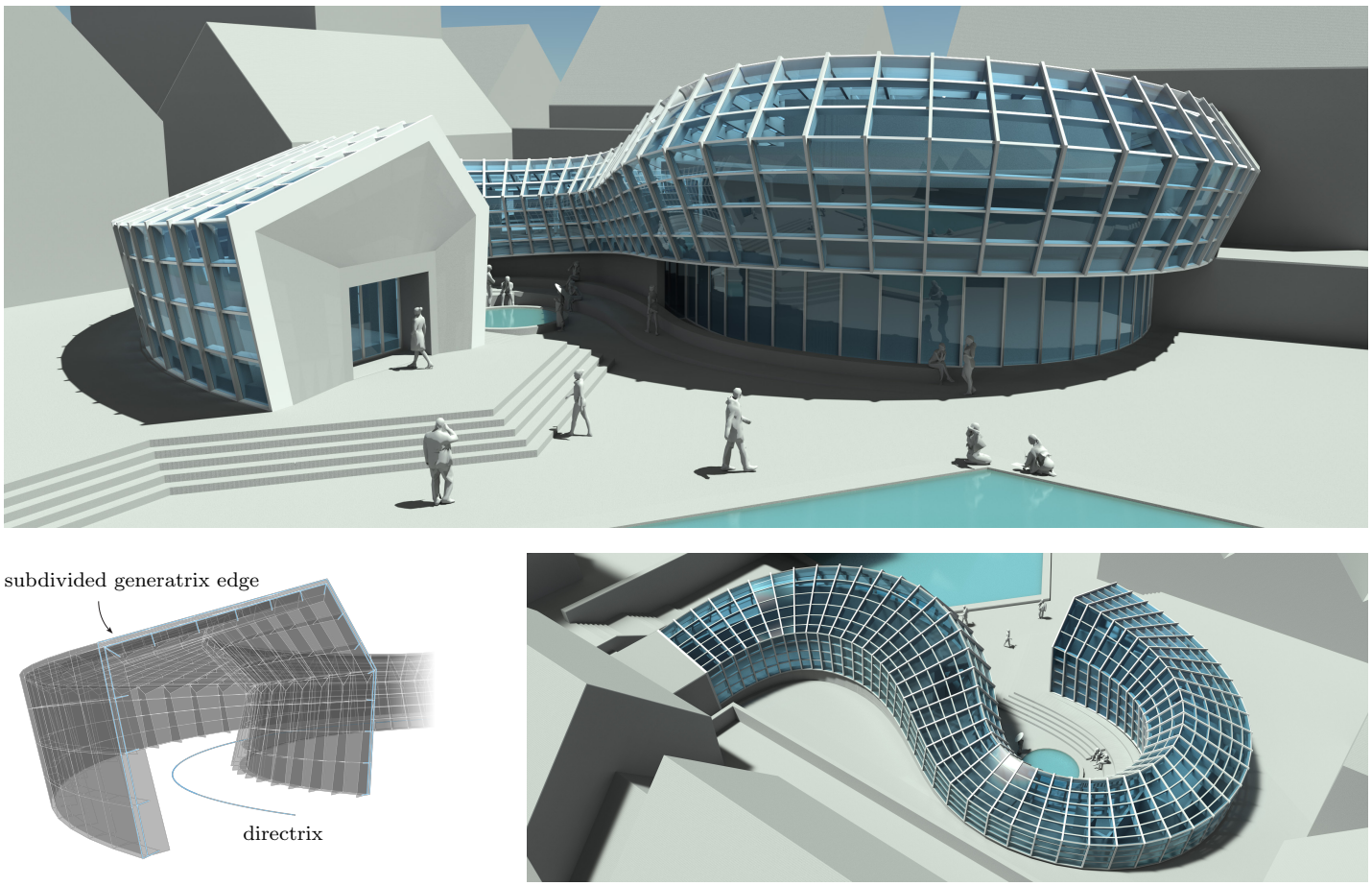
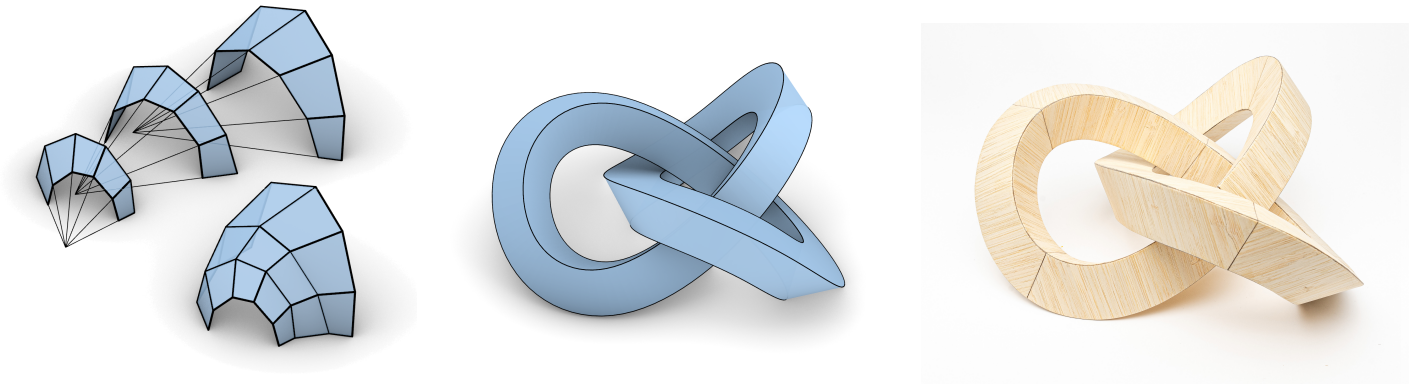


Figure 1: Architectural facade with PQ faces and torsion-free support structure designed using our *CNets* plugin. The generatrix is further subdivided in segments sweeping flat glass panels that pave the cone-net.

(see Figure 2a). Vidulis et al. [16] developed a construction method for semi-discrete and discrete cone-nets to generate tubular structures from developable strips. This method was incorporated into an optimization-based design framework that adjusts input parameters to control the local orientation of the PQ strips (see Figure 2b).

Our work builds on this research and extends the construction proposed by Vidulis and coauthors in several ways. With a focus on applications in architectural design, we propose different controls for more intuitive modeling, and introduce a guiding algorithm that prevents singularities and repairs invalid designs. We show how this facilitates interactive form finding and exploration, allowing the user to navigate the complete space of discrete cone-nets with intuitive control handles. These innovations are implemented in a Grasshopper/Rhino plugin that is made publicly available. We demonstrate the effectiveness of our tool in several design studies that highlight the versatility and benefits of discrete cone-nets. In particular, we create freeform shell designs with planar faces and torsion-free support structures.



(a) Discrete cone-net [6].

(b) A smooth C-tube optimized for closure and its realization from veneer [16].

Figure 2: Previous work on cone-nets.

2 Related Work

In architectural geometry, the design of planar quadrilateral (PQ) meshes typically follows two main paradigms: optimization-based methods, which adjust an initial mesh to achieve approximately planar faces, and constructive approaches, which guarantee planarity through geometric principles.

Optimization-based methods begin with an initial freeform surface and mesh, and iteratively adjust the geometry to improve quad planarity while maintaining an approximation of the target shape [2] . Additional design objectives, such as enabling offsets [7,13] or enforcing planarity of the mesh polylines [5], can be incorporated into the optimization. However, the outcomes are sensitive to the initial mesh configuration. For a broader survey of existing methods, we refer the reader to Pottmann et al. [11].

Construction-based approaches yield exact PQ meshes but are limited in the range of achievable designs. Since these methods often rely on guiding curves for the construction process, they typically produce only regular mesh topologies. Examples of such constructions include Marionette meshes [8], which are generated from a single projected planar view and two elevation curves. Tellier et al. [15] construct conical and circular meshes with planar parameter lines using a method based on a discretization of the Gauss map.

In addition, classical surface types that allow for straightforward construction have been studied in relation to architectural geometry. Monge surfaces, also known as isogonal moulding surfaces, are a classical sweep-based construction, where a generatrix curve is swept along a directrix curve in a rotation-minimizing fashion [4, 9]. In the discrete setting, this results in PQ meshes whose polylines lie in planes that bisect the edges of the directrix. The resulting meshes feature conical vertices and edge offsets. Translational PQ meshes are generated by translating one polyline along another, and feature two families of congruent mesh polylines that lie in parallel planes. Glymph et al. [3] extend this concept by combining translation with scaling, resulting in *scale-trans surfaces*, where the mesh polylines lie in parallel planes and are similar up to scaling.

Discrete cone-nets, as introduced by Kilian et al. [6], encompass Monge and scale-trans surfaces.

Building on a recent work by Vidulis et al. [16] who introduce a novel construction method for discrete cone-nets, we propose an interactive design tool enabling the generation of PQ meshes defined by two guiding polylines and additional user-specified input.

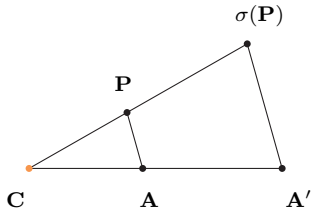
3 Preliminaries

In the following, we review the construction of C-tubes, and refer to Vidulis et al. [16] for more details.

The C-tube construction is based on using scalar parameters to represent the cone apices as points located on tangent lines to the discrete directrix. This approach is inspired by Mundilova and coauthors [10] and avoids potential numerical instabilities associated with explicitly handling cone apices that are located far from the mesh geometry.

To encode lines that pass through the cone apices, the construction relies on scale transformations with respect to those apices. For a scalar $a \in \mathbb{R}$ locating the apex of the cone on the line passing through two points $\mathbf{A}$ and $\mathbf{A}'$, the scale transformation is

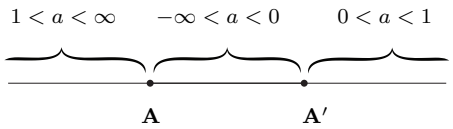
$$\sigma : \mathbf{P} \mapsto a(\mathbf{P} - \mathbf{A}) + \mathbf{A}'.$$



Scale mapping

If $a = 1$, the resulting transformation is a translation by vector $\mathbf{A}' - \mathbf{A}$; otherwise, it is a scaling with respect to the center

$$\mathbf{C} = \frac{1}{1-a}\left(\mathbf{A}' - a\mathbf{A}\right). \tag{1}$$



Location of the scale center $\mathbf{C}$

3.1 C-Tubes Construction Input

The construction for discrete C-tubes is inspired by the classical Monge surface construction and requires the following input parameters (see Figure 3):

(1) *Discrete directrix:* A polyline $\mathbf{X} = \{\mathbf{X}_0, \ldots, \mathbf{X}_{M-1}\}$, such that $|\mathbf{X}_j - \mathbf{X}_{j+1}| \neq 0$ for all j.

(2) *Discrete generatrix:* A polyline $\mathbf{G}_{:,0} = \{\mathbf{G}_{0,0}, \ldots, \mathbf{G}_{N-1,0}\}$, with $\mathbf{G}_{0,0} = \mathbf{G}_{N-1,0}$ if closed.

(3) *Apex-locating function:* A scalar-valued sequence $\mathbf{a} = \{a_1, \ldots, a_{M-1}\}$ that specifies the location of the cone apices along the tangents to the directrix.

(4) *Construction planes:* A collection of N families of $M-1$ construction planes $\Pi_{i,:} = \{\Pi_{i,1}, \ldots, \Pi_{i,M-1}\}$, where each plane $\Pi_{i,j}$ passes through the vertex $\mathbf{X}_j$ and is defined by a unit normal vector $\mathbf{P}_{i,j}$. To avoid degeneracies, each plane must not be parallel to the incident segments of the directrix, i.e.,

$$\mathbf{P}_{i,j} \cdot (\mathbf{X}_j - \mathbf{X}_{j-1}) \neq 0 \quad \text{and} \quad \mathbf{P}_{i,j} \cdot (\mathbf{X}_{j+1} - \mathbf{X}_j) \neq 0.$$

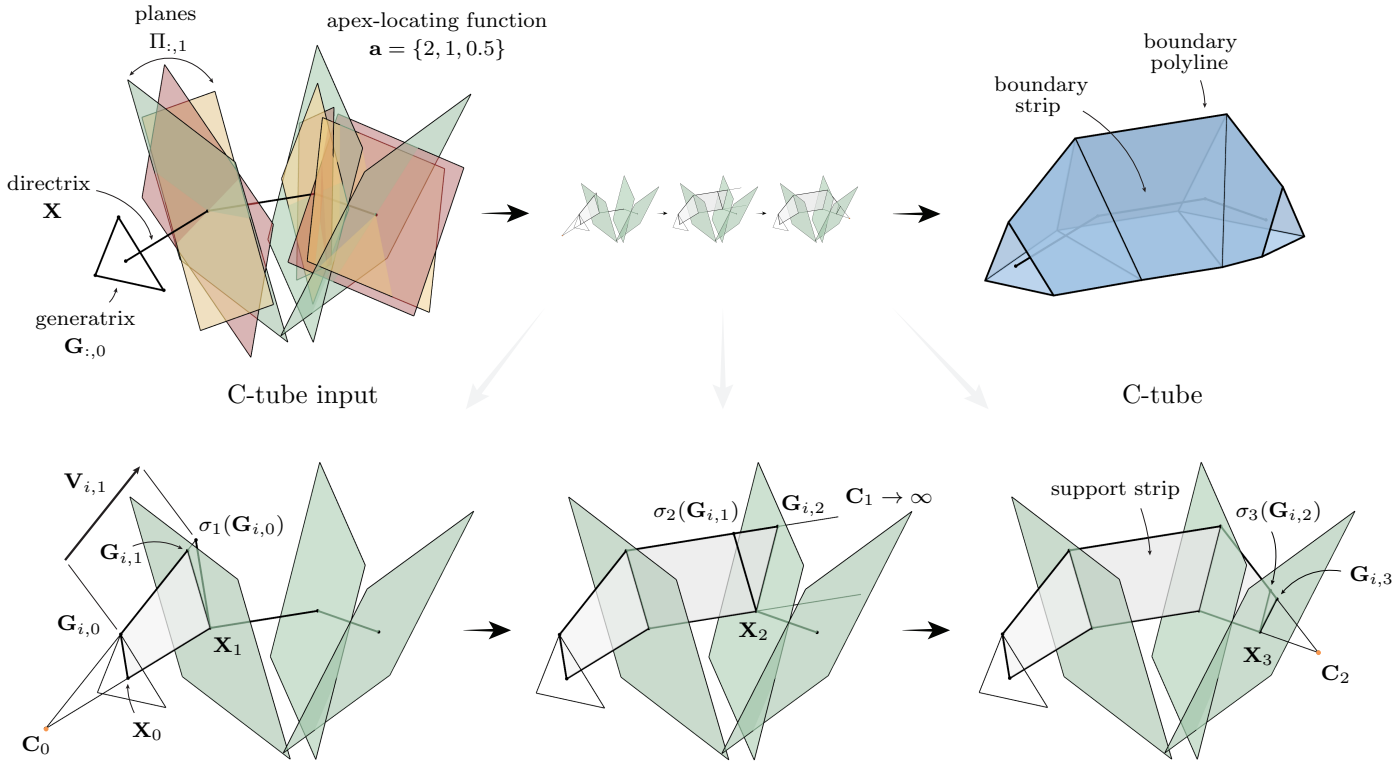


Figure 3: Illustration of the C-tube construction by Vidulis et al. [16].

3.2 C-Tubes Construction Overview

The construction generates a regular $N \times M$ planar quad mesh with vertices $\mathbf{G}_{i,j}$, such that:

- *Constraint 1:* The vertices of the mesh coincide with corresponding construction planes, i.e., $\mathbf{G}_{i,j} \in \Pi_{i,j}$.
- *Constraint 2:* The vertices $\mathbf{G}_{i,j-1}$ and $\mathbf{G}_{i,j}$ are collinear with the cone apex $\mathbf{C}_{j-1}$ lying on the tangent line to edge $\mathbf{X}_{j-1}\mathbf{X}_j$. We encode the cone apex as the center of a scaling σ_j using the scalar a_j and the corresponding pair of points $\mathbf{X}_{j-1}$ and $\mathbf{X}_j$.

In the following, we refer to the polylines $\mathbf{G}_{:,j}$ as *generatrix-aligned*, and the polylines $\mathbf{G}_{i,:}$ as *directrix-aligned*. The use of $\mathbf{G}$ to denote the mesh vertices is motivated by the directrix-aligned polylines $\mathbf{G}_{i,:}$, which for C-tubes are referred to as *gluing curves* of the directrix-aligned PQ strips.

Mathematically, the sequential generation of the mesh can be described as follows. For $j > 0$, we compute each vertex as $\mathbf{G}_{i,j} = \mathbf{G}_{i,j-1} + d_{i,j}\mathbf{V}_{i,j}$, where $d_{i,j}$ is a scalar and $\mathbf{V}_{i,j}$ is a direction vector, both determined by the constraints stated above.

To satisfy *Constraint 2*, we define the direction vector

$$\mathbf{V}_{i,j} = \sigma_j(\mathbf{G}_{i,j-1}) - \mathbf{G}_{i,j-1} = a_j\,(\mathbf{G}_{i,j-1} - \mathbf{X}_{j-1}) + \mathbf{X}_j - \mathbf{G}_{i,j-1},$$

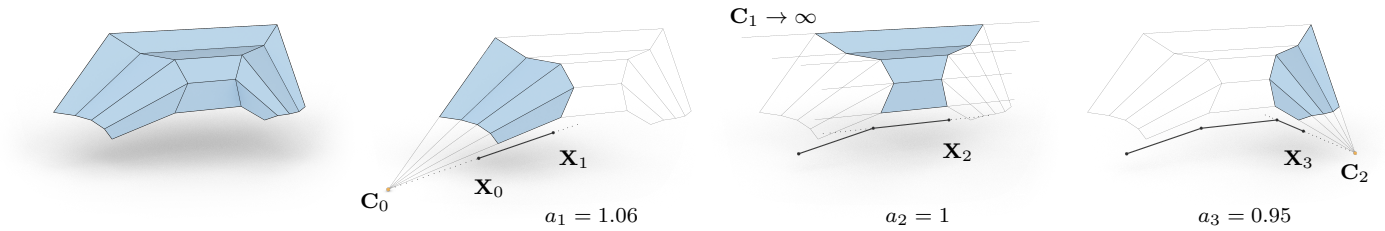


Figure 4: Any given cone-net (left) can be encoded as an open C-tube. A valid directrix and the corresponding apex-locating function can be sequentially determined using the location of the cone apices.

where $\sigma_j(\mathbf{G}_{i,j-1})$ denotes the image of $\mathbf{G}_{i,j-1}$ under the scale transformation associated with the edge $\mathbf{X}_{j-1}\mathbf{X}_j$. To satisfy *Constraint 1*, we determine $d_{i,j}$ from $(\mathbf{G}_{i,j} - \mathbf{X}_j) \cdot \mathbf{P}_{i,j} = 0$. If $\mathbf{V}_{i,j} \cdot \mathbf{P}_{i,j} \neq 0$, this yields

$$d_{i,j} = \frac{(\mathbf{X}_j - \mathbf{G}_{i,j-1}) \cdot \mathbf{P}_{i,j}}{\mathbf{V}_{i,j} \cdot \mathbf{P}_{i,j}}.$$

Pseudo-code for the sequential construction is provided in Vidulis et al. [16].

For each fixed i, the directrix $\mathbf{X}$ and directrix-aligned polylines $\mathbf{G}_{i,:}$ form the boundaries of a PQ strip by construction, since the extensions of their edges intersect at the apex of the incident cone. Due to the close relationship with the support structures of the mesh, we refer to these strips as *support strips* (see Figure 3, center). Further details are discussed in Section 4.4.2.

4 Interactive Design of Cone-Nets

As demonstrated by Vidulis and coauthors, the straightforward construction of C-tubes offers numerous computational advantages and is well-suited for the design of discrete tubular cone-nets. In this section, we show how *any* discrete cone-net can be constructed using this method, and we propose a modification to the construction input that favors interactive exploration.

4.1 Cone-Nets as Open C-Tubes

Recall that a given discrete Monge surface can be equivalently generated from multiple distinct directrices, provided they have pairwise parallel edges and share the same set of bisecting planes. Similarly, Vidulis and coauthors observe that a given C-tube can be constructed from an even broader family of inputs. Building on the discussion in [16, Appendix D], here we demonstrate that any discrete cone-net can be represented using the inputs from the *open* C-tube construction, i.e., C-tubes with an open generatrix, while constraining the apex-locating function to only assume positive values.

Given a cone-net $\mathbf{G}$, let $\{\mathbf{C}_0, \ldots, \mathbf{C}_{M-2}\}$ denote the set of real or infinite cone apices $\mathbf{C}_j$ corresponding to the PQ strips bounded by the generatrix-aligned polylines $\mathbf{G}_{:,j}$ and $\mathbf{G}_{:,j+1}$, see Figure 4.

First, we discuss how to find a suitable polyline representing the directrix such that all $a_j > 0$. Note that this directrix must satisfy the following two constraints:

(1) Each cone apex $\mathbf{C}_j$ must lie on the line defined by the segment $\mathbf{X}_j\mathbf{X}_{j+1}$.

(2) For the apex-locating function to be positive, the cone apex $\mathbf{C}_j$ must not lie on the segment $\mathbf{X}_j\mathbf{X}_{j+1}$.

For a given set of cone apices, appropriate directrices satisfying the two constraints can be determined sequentially. We begin by choosing a starting point $\mathbf{X}_0 \notin \{\mathbf{C}_0, \mathbf{C}_1\}$ that does not coincide with any plane spanned by the faces of the cone-net[1]. Then, for all $j > 0$, we select $\mathbf{X}_j \notin \{\mathbf{C}_j, \mathbf{C}_{j+1}\}$ on the line $\mathbf{C}_{j-1}\mathbf{X}_{j-1}$ such that $\mathbf{C}_{j-1}$ does not lie between $\mathbf{X}_{j-1}$ and $\mathbf{X}_j$. Overall, the choice of vertices of the directrix construction has a total of $M + 2$ degrees of freedom: three degrees of freedom for the choice of the initial point $\mathbf{X}_0$, and one degree of freedom for each subsequent vertex. Directrices that remain "close" to the cone-net provide more intuitive control in design applications and are thus generally preferable.

Once an appropriate directrix is determined, the scalar values encoding the apices can be assigned as follows. If $\mathbf{C}_{j-1}$ is a point at infinity, set $a_j = 1$. Otherwise, define

$$a_j = \frac{|\mathbf{X}_j - \mathbf{C}_{j-1}|}{|\mathbf{X}_{j-1} - \mathbf{C}_{j-1}|}.$$

Finally, we assign the normal vectors $\mathbf{P}_{i,j}$ of the construction planes to be perpendicular to $\mathbf{X}_j\mathbf{G}_{i,j}$, but not to $\mathbf{X}_{j-1}\mathbf{X}_j$ and $\mathbf{X}_j\mathbf{X}_{j+1}$. We set $\mathbf{G}_{:,0}$ as the initial values for the construction. We conclude:

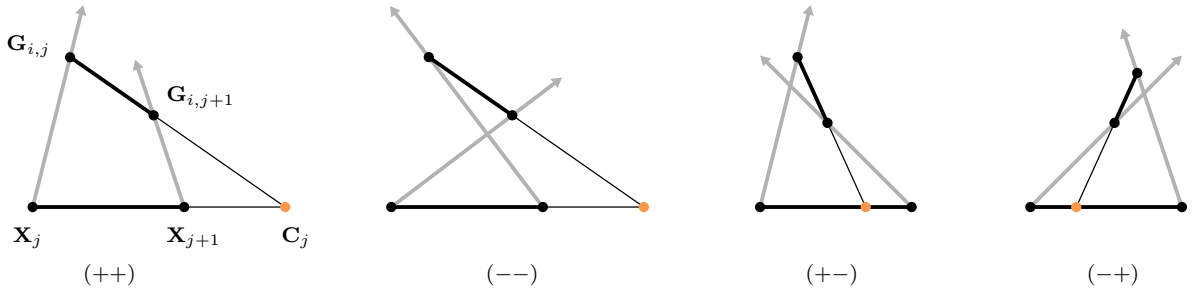
Proposition 1. *A regular discrete cone-net can be encoded using the (open) C-tube construction input with positive apex-locating function $a_j > 0$ and a generatrix that does not coincide with the planes incident to the faces of the cone-net.*

The open C-tube construction therefore provides suitable foundation for the interactive exploration of discrete cone-nets.

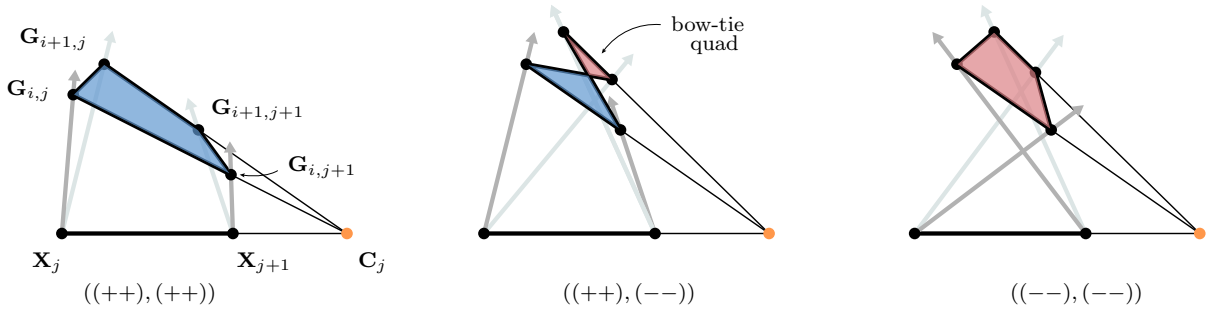
4.2 Handling Undesirable Configurations

Having established that every cone-net can be encoded using the C-tube construction, and that every such construction yields a valid cone-net, we note that not all cone-nets are desirable in practice. In particular, with poorly chosen input, the planar quads may loose convexity and become bow-tie shaped, see Figure 5b.

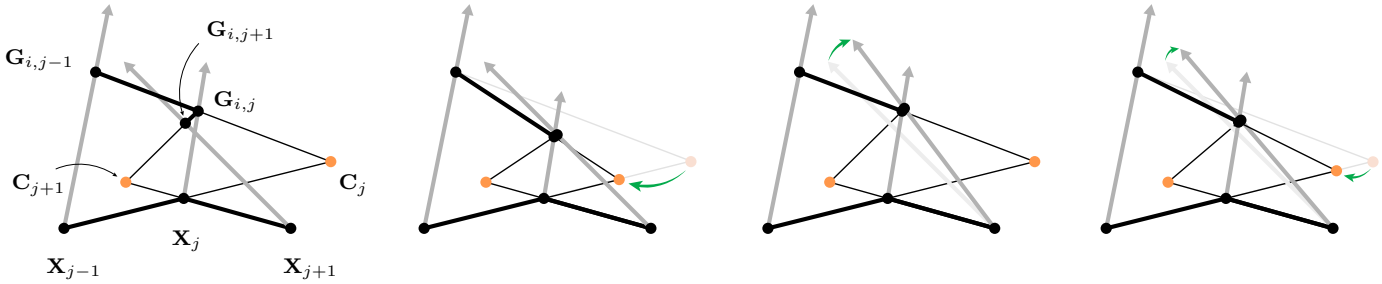
In the following, we analyze the conditions under which such fold-overs occur and propose a strategy for adjusting either the apex-locating function or the construction planes to recover from these configurations. In what follows, we assume that the construction plane normals are oriented such that they point away from the preceding directrix vertex, that is $\mathbf{P}_{i,j} \cdot (\mathbf{X}_{i,j} - \mathbf{X}_{i,j-1}) > 0$.



(a) Cases for a single quad of the support strip. The case (++) corresponds to a convex quad.



(b) Cases for a single quad of the boundary strip and its two neighboring support strips, assuming $a_{j+1} > 0$. The case $((++), (++))$ corresponds to a convex and properly oriented quad.



(c) An undesired configuration of a support strip and three proposed fixes, indicated by green arrows: Change of the apex-locating function a_{j+1}, change of the next construction plane $\Pi_{i,j+1}$, both.

Figure 5: Identification and correction of undesirable configurations. Thick black lines represent the directrix and the cone-net mesh, thin black lines represent cones whose apex is indicated by an orange dot. Gray arrows indicate the rulings of the support strips.

4.2.1 Identification of Undesirable Configurations

To analyze the undesirable configurations, we start by considering the relative positions of two vertices, $\mathbf{G}_{i,j}$ and $\mathbf{G}_{i,j+1}$, with respect to the two construction planes $\Pi_{i,j+1}$ and $\Pi_{i,j}$, respectively. Depending on

[1]Note that this constraint is sufficient but not necessary. For instance, setting $\mathbf{X} = \mathbf{G}_{i,:}$ is valid; however, for the sake of brevity, we omit such cases from the discussion.

the sidedness of each point with respect to the other plane, four configurations arise, see Figure 5a:

$$c_{i,j} = \begin{cases} (++), & \text{if } (\mathbf{G}_{i,j+1} - \mathbf{X}_j) \cdot \mathbf{P}_{i,j} > 0 \text{ and } (\mathbf{X}_{j+1} - \mathbf{G}_{i,j}) \cdot \mathbf{P}_{i,j+1} > 0, \\ (--), & \text{if } (\mathbf{G}_{i,j+1} - \mathbf{X}_j) \cdot \mathbf{P}_{i,j} < 0 \text{ and } (\mathbf{X}_{j+1} - \mathbf{G}_{i,j}) \cdot \mathbf{P}_{i,j+1} < 0, \\ (+-), & \text{if } (\mathbf{G}_{i,j+1} - \mathbf{X}_j) \cdot \mathbf{P}_{i,j} > 0 \text{ and } (\mathbf{X}_{j+1} - \mathbf{G}_{i,j}) \cdot \mathbf{P}_{i,j+1} < 0, \\ (-+), & \text{if } (\mathbf{G}_{i,j+1} - \mathbf{X}_j) \cdot \mathbf{P}_{i,j} < 0 \text{ and } (\mathbf{X}_{j+1} - \mathbf{G}_{i,j}) \cdot \mathbf{P}_{i,j+1} > 0, \end{cases}$$

Among the four possible cases, the configurations $(+-)$ and $(-+)$ can only occur when $a_{j+1} < 0$, i.e., when the cone apex $\mathbf{C}_j$ lies on the segment $\mathbf{X}_j\mathbf{X}_{j+1}$. If we restrict the construction to $a_{j+1} > 0$, only the cases $(++)$ and $(--)$ are possible.

Figure 5b illustrates the four possible cases arising from the combination of neighboring configurations $c_{i,j}$ and $c_{i+1,j}$, under the assumption that $a_{j+1} > 0$:

- $(c_{i,j}, c_{i+1,j}) = ((++), (++))$: The face is convex and properly aligned.
- $(c_{i,j}, c_{i+1,j}) = ((++), (--))$ or $(c_{i,j}, c_{i+1,j}) = ((--), (++))$: The face is not convex.
- $(c_{i,j}, c_{i+1,j}) = ((--), (--))$: The face is convex.

Overall, we find that only the combination of two cases $(++)$ is desirable. Pairs of neighboring $(--)$ configurations, while valid in principle, can produce sharp creases along the directrix direction. Consequently, in the following, we aim to generate cone-nets composed exclusively of $(++)$ configurations.

4.2.2 Correction of Undesirable Configurations

If the apex-locating function is positive, it follows that cone-nets consisting of only $(++)$ configurations can be enforced if

$$(\mathbf{X}_{j+1} - \mathbf{G}_{i,j}) \cdot \mathbf{P}_{i,j+1} > 0 \quad \text{for all } i, j. \tag{2}$$

During sequential construction, it can occur that the vertex $\mathbf{G}_{i,j}$, where $0 < j < M - 1$, lies on the wrong side of the next construction plane $\Pi_{i,j+1}$. This case[2] can be addressed in one of the following ways (see Figure 5c):

by adjusting the current value of the apex-locating function, by modifying the next construction plane, or by a combination of both. Detailed computations are provided in Appendix A.

4.3 Normalization of the Apex-Locating Function

The encoding of the cone apices presented in Vidulis et al. [16] is suitable for approximately uniform edge lengths along the directrix. However, when exploring designs with varying numbers of subdivisions

[2]The case where the vertex $\mathbf{G}_{i,0}$ lies on the wrong side of the construction plane $\Pi_{i,1}$ can be resolved similarly, by either moving the vertex towards the first point of the directrix $\mathbf{X}_0$ or adjusting the construction plane.

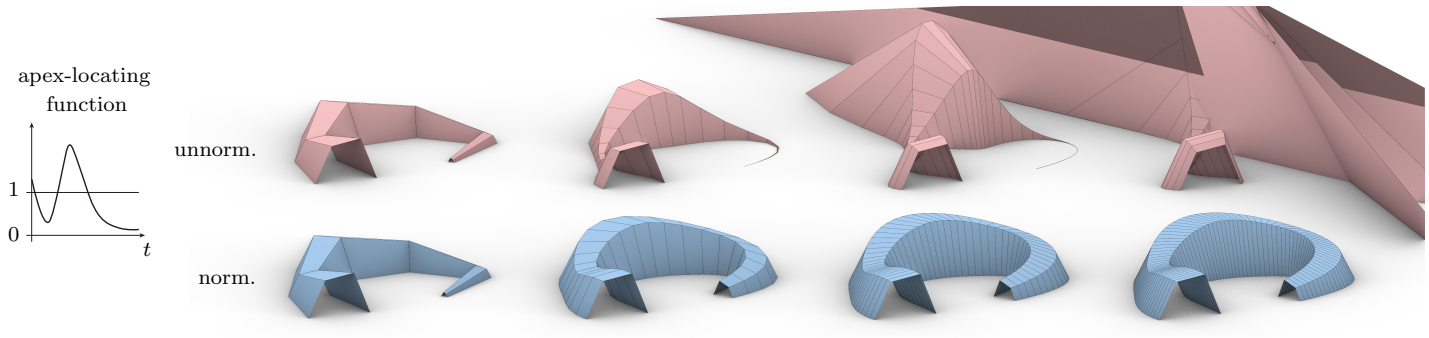


Figure 6: Comparison between an unnormalized (top, red) and a normalized (bottom, blue) discrete cone-net using the same apex-locating function (left), generatrix, directrix, and ratios profiles, at different discretization levels. The normalization of the apex-locating function guarantees that the result remains consistent under refinement, preventing extreme shrinkage and blow-up of the mesh.

and non-uniform edge lengths, it becomes advantageous to encode cone apex locations independently of the polyline's edge lengths. In the following, we map the *normalized apex-locating function* $\bar{\mathbf{a}}$ to an apex-locating function $\mathbf{a}$, such that resulting cone-nets maintain a similar appearance across different levels of subdivision, see Figure 6.

Recall that the locations of the cone apices are encoded as the center of the scale transformation

$$\sigma_j \colon \mathbf{Q} \mapsto a_j(\mathbf{Q} - \mathbf{X}_{j-1}) + \mathbf{X}_j.$$

To achieve an edge-length independent formulation, we define the normalized scaling

$$\bar{\sigma}_j \colon \mathbf{Q} \mapsto \bar{a}_j(\mathbf{Q} - \mathbf{A}_j) + \mathbf{A}'_j,$$

where $\mathbf{A}_j = \mathbf{X}_{j-1}$ and $\mathbf{A}'_j$ is a point on $\mathbf{X}_j\mathbf{X}_{j+1}$, independent on the corresponding edge length. Our goal is to find a_j corresponding to a normalized scale factor $\bar{a}_j$ such that the scale transformations σ_j and $\bar{\sigma}_j$ share the same center and thus result in the same construction.

For the following computations, we abbreviate $l_j = |\mathbf{X}_j - \mathbf{X}_{j-1}|$, and let $L > 0$ be a scalar, which will act as an edge-length independent reference for the distance between the points $\mathbf{A}_j\mathbf{A}'_j$. We consider two cases:

- *Case $\bar{a}_j \neq 1$:* Without loss of generality, we assume that $\mathbf{A}_j = \mathbf{X}_{j-1} = (0,0)$, $\mathbf{A}'_j = (L,0)$, and $\mathbf{X}_j = (l_j, 0)$. It follows from Equation (1) that the scale centers are

$$\mathbf{C} = \frac{1}{1-a_j}\left(\mathbf{X}_j - a_j\mathbf{X}_{j-1}\right) = \left(\frac{l_j}{1-a_j}, 0\right) \quad \text{and} \quad \bar{\mathbf{C}} = \frac{1}{1-\bar{a}_j}\left(\mathbf{A}'_j - \bar{a}_j\mathbf{A}\right) = \left(\frac{L}{1-\bar{a}_j}, 0\right),$$

  and therefore the scale transformations share the same center if

$$\frac{l_j}{1-a_j} = \frac{L}{1-\bar{a}_j}.$$

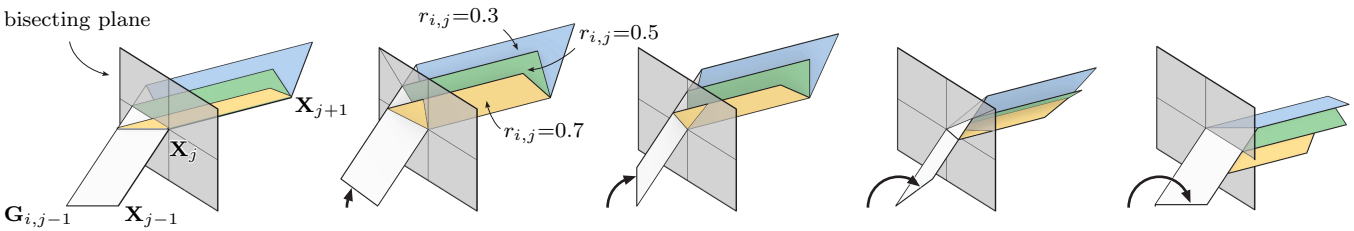


Figure 7: Effect of the ratios on the ruling directions of the support patches as it rotates about the incoming generatrix edge. A value of 0.5 (green) always yields a ruling direction incident with the bisecting plane, independently of previous generatrix position.

Consequently, if provided with a normalized $\bar{a}_j$, we can use the construction presented in Section 3.2 for the transformed scale factor

$$a_j = 1 - (1 - \bar{a}_j)\frac{l_j}{L}. \tag{3}$$

- *Case* $\bar{a}_j = 1$*:* In this case, the scale transformation $\bar{\sigma}_j$ is a translation, and the corresponding scale factor a_j is also equal to 1.

Therefore, the cone apices can alternatively be encoded using a scale factor L and the normalized apex-locating functions, independently of the directrix edge lengths l_j.

Choosing an appropriate scale factor L remains the main difficulty. Due to the correction mechanism introduced in Section 4.2.2, we aim to ensure that the normalized values $\bar{a}_j$ and the scale factor L encode cone apices that do not coincide with the edges of the directrix. Assuming $\bar{a}_j > 0$, we require that L be greater than the length of the longest directrix edge in a reasonable subdivision. On the other hand, L should not be so large that the apex locations become insensitive to variations in $\bar{a}_j$. The heuristic used in our implementation sets L to one-fifth of the total length of the directrix. Since this value remains approximately consistent across subdivisions, we observe that using normalized apex-locating functions in combination with this choice of L yields stable and comparable results[3], see Figure 6.

4.4 Cone-Net Design Using the *CNets* Plug-in

We can now present our forward cone-net design tool, *CNets*, developed as a compiled C# plug-in for Grasshopper/Rhino. The *CNets* plug-in comprises components for interactive exploration of discrete cone-nets, along with supplementary tools that facilitate design and fabrication workflows.

4.4.1 Cone-Net Construction With *CNets*

The core of the *CNets* plug-in are two components, *CNets Construct* and *CNets Construct Coplanar*, which allow the user to interactively explore the design space. In the following, we provide a more detailed discussion of their interface and use.

[3]The sign of $\bar{a}_j$ is no longer an indicator of whether the cone vertices lie within the edge $\mathbf{X}_{j-1}\mathbf{X}_j$, since $a_j > 0$ if and only if $\bar{a}_j > 1 - 1/l_j$. While this condition is less intuitive than the sign of a_j, it remains efficient to check in practice.

CNets Construct. While the directrix, generatrix, and apex-locating function inputs used in the C-tube construction have direct analogues in the proposed *CNet Construct* component, the way we encode the construction planes differs. Vidulis et al. [16] represent construction planes via their normals. This approach presents challenges in the context of an exploratory design tool, particularly when selecting an appropriate reference frame. For example, osculating plane normals are not readily defined for collinear vertices. Furthermore, in constructing a vertex $\mathbf{G}_{i,j}$, there exists a one-parameter family of normal directions, specifically, those perpendicular to $\mathbf{G}_{i,j} - \mathbf{X}_j$, that yield the same result. Experiments also suggest that manipulating construction plane normals using two degrees of freedom is not especially intuitive from a design perspective.

In the proposed interface, we address this issue by allowing the user to directly manipulate the rulings of the support patches, that is, the PQ strips spanned between the directrix and the directrix-aligned polylines. In the sequential construction, the inclination of rulings is influenced by *ratios* $r_{i,j} \in (0, 1)$, for $j > 0$, see Figure 7. Considering Monge surfaces as the default, we set $r_{i,j} = 1/2$ to encode the ruling lying in the bisecting plane of the edges incident to $\mathbf{X}_j$. The assignments $r_{i,j} = 0$ and $r_{i,j} = 1$ correspond to corner cases where the ruling direction $\mathbf{R}_{i,j}$ is aligned with the direction $\mathbf{X}_j - \mathbf{X}_{j-1}$ and $\mathbf{X}_{j-1} - \mathbf{X}_j$, respectively. The intermediate values of $r_{i,j}$ correspond to ruling directions between these extremes.

To summarize, in addition to items (1)-(3) of the input presented in Section 3.1, the *CNet Construct* input consists of:

(4) For every directrix-aligned polyline $\mathbf{G}_{i,:}$, a collection of ratios $\{r_{i,0}, \ldots, r_{i,M-1}\}$ encoding the ruling directions.

(5) A numeric parameter, *activateAssistance*, which indicates whether the assisted construction described in Section 4.2.2 is enabled. Negative values disable the assistance. Values between 0 and 1 determine the relative influence of the construction plane modification versus the apex-locating function. Specifically, *activateAssistance* $= 0$ means only planes are modified, while *activateAssistance* $= 1$ means only the apex-locating function is modified.

(6) A boolean *apexFunctionNormalized* indicating whether the provided apex-locating function is normalized, as discussed in Section 4.3.

Figure 8 illustrates the range of design possibilities enabled by combining the apex-locating function with ratio parameters. Starting from a discrete torus segment generated as a Monge surface from two circular arcs, we broaden the design space by first varying the apex-locating function, and then by applying two successive modifications to the ratio functions corresponding to all generatrix vertices individually.

CNets Construct Coplanar. While the above interface of the *CNets Construct* component is general and provides individual handles for the rulings of the support strips via the associated ratios $r_{i,j}$, it has the drawback that achieving planar generatrix-aligned polylines, a potentially desirable target that can

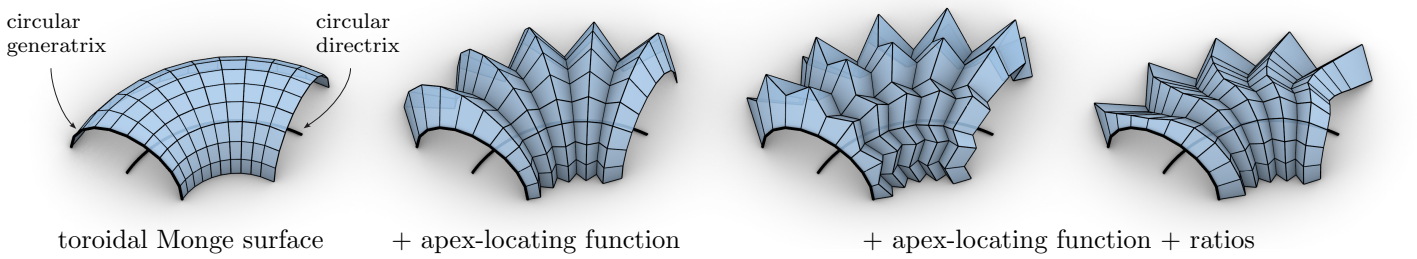


Figure 8: Examples demonstrating incremental use of the design flexibility of the *CNets Construct* plugin, all based on the same directrix and generatrix.

ease fabrication, is not straightforward. We thus provide a second component, *CNets Construct Coplanar*, which constructs only cone-nets with planar generatrix-aligned curves.

The *CNets Construct Coplanar* input differs from *CNets Construct* only in that the N sets of ratios in item (4) are replaced by just two arrays: the first corresponds to the ratios of the rulings on the support patch incident to the first generatrix point, while the second controls the rotation of the construction plane around this ruling. Again, we default to the Monge surface construction when all ratios are $1/2$, while the extremal cases of 0 and 1 correspond to configurations where the planes are aligned with the edges of the directrix.

While the apex-locating function and ratios can be specified individually for low-resolution meshes, this approach becomes impractical for higher face counts. We recommend using graph mappers, such as the Grasshopper's native or Rich Graph Mapper, to efficiently manipulate large arrays of values with just a few control handles.

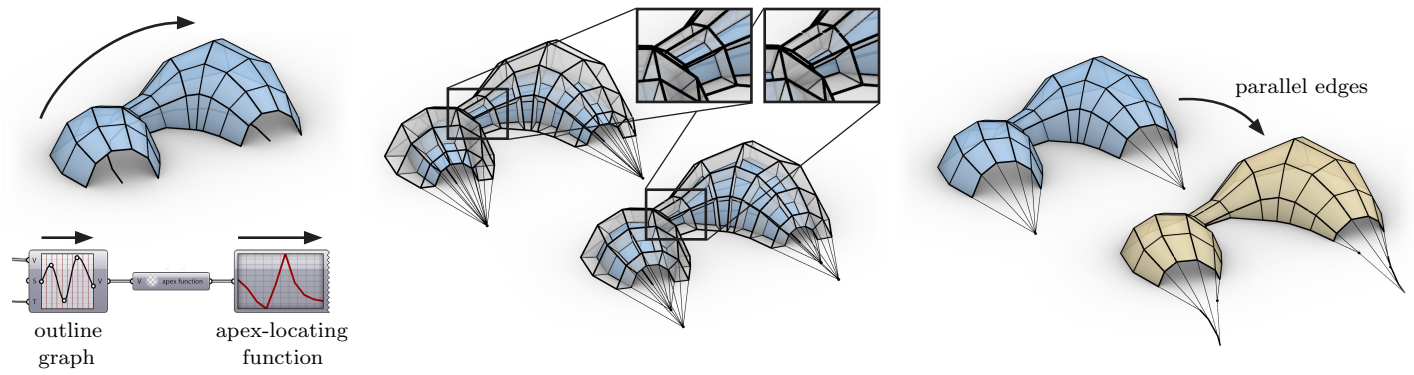
Unlike the work of Vidulis et al. [16], our components do not directly support optimization. Nevertheless, the linear-time construction of our *CNets* components enables integration with high-level optimization frameworks, as discussed in Appendix B.

4.4.2 Supplementary *CNets* Components

In addition to the cone-net construction components, our *CNets* plug-in includes additional tools to support fabrication of the resulting structures.

Outline graph to apex-locating function. When visualized as a graph, the apex-locating function does not offer a clear visual correspondence to "outline" of the resulting cone-net. Its manipulation is thus not particularly intuitive from a design standpoint.

The component *CNets Graph to Apex Function* transforms a user-defined graph that roughly encodes the widths of the support strips along the directrix into an approximate apex-locating function, see Figure 9a. This transformation is based on an idealized scenario where all ratios are set to $1/2$ (Monge surface) and the directrix is straight and has uniform edge-lengths. If the input graph is defined by the



(a) Outline graph to apex-locating function.

(b) Comparison of a conical (top) with a parallel second layer (bottom).

(c) Mesh parallelism used to generate a PQ mesh that is not a cone-net.

Figure 9: Illustrations of supplementary *CNets* components.

values y_j, the apex-locating function is obtained by setting $a_j = {}^{y_j}/_{y_{j-1}}$.

Support structures. In freeform facade design, PQ meshes are often used to generate multi-layered constructions, typically consisting of a support structure made of beams for structural stability, and a secondary mesh layer that provides additional structural or physical properties, such as insulation.

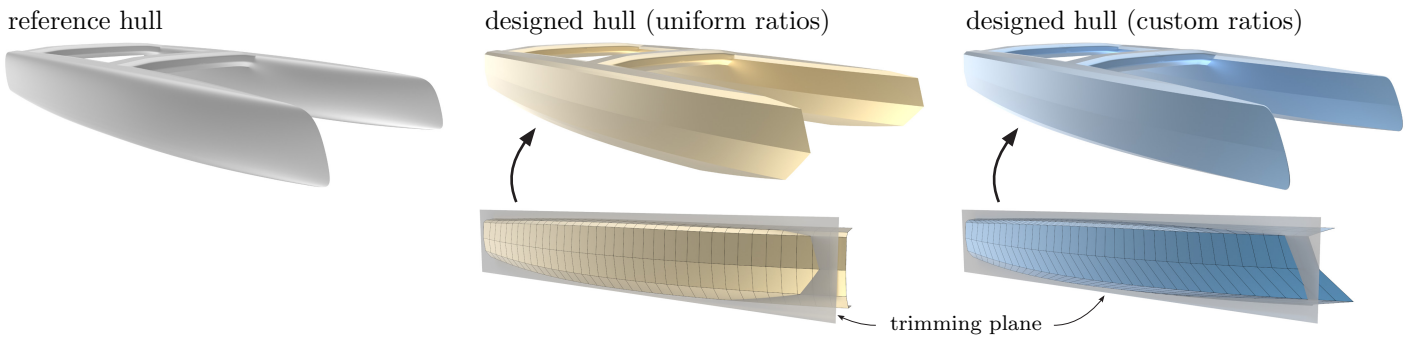
Torsion-free node axes are collections of lines, one per vertex of a mesh, such that lines corresponding to adjacent vertices lie in a common plane. They are essential for constructing support structures and secondary mesh layers. Given a regular PQ mesh with torsion-free support axes, one can trace a second mesh layer with parallel edges by propagating from an initial vertex.

As pointed out in Vidulis et al. [16], the C-tube construction allows for straightforward extraction of torsion-free node-axes by connecting the vertices $\mathbf{G}_{i,j}$ to the corresponding points on the directrix $\mathbf{X}_j$. The same principle applies to the construction of cone-nets.

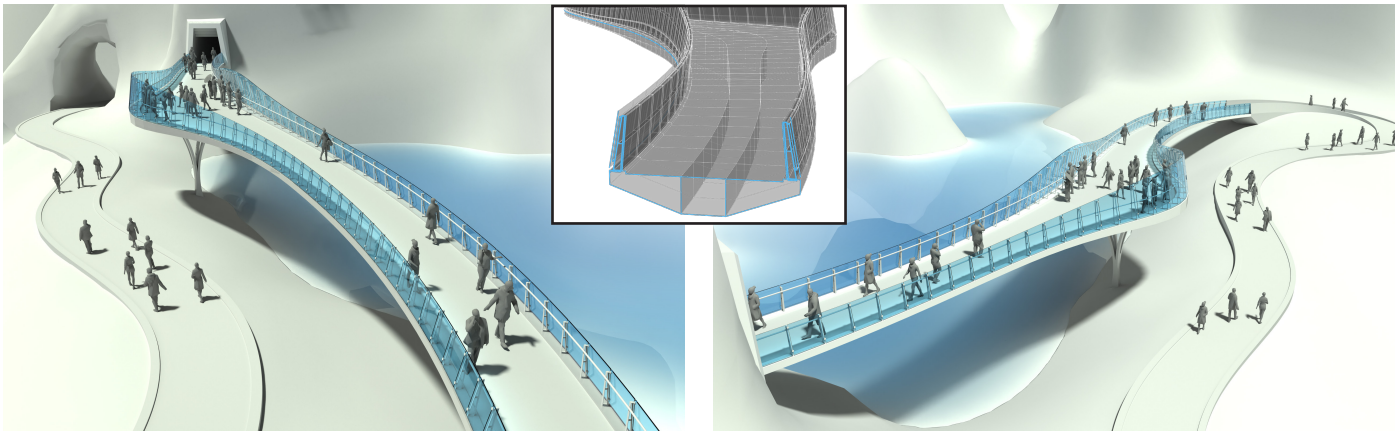
While second-layer meshes parallel to the original mesh are valuable in many applications, secondary layers derived from cone-nets via parallelism generally cease to be cone-nets. Nevertheless, torsion-free node-axes extracted from the construction enable the creation of a second layer that is also a cone-net, sharing the same cone apices. Although only one family of polylines remains parallel in this case, we observe that, in some instances, this construction aligns more nicely with the curvature of the original mesh, see Figure 9b.

Our implementation supports the extraction of construction-based torsion-free node-axes (*CNet Get Torsion-Free Axes*), as well as the creation of secondary layers that are either cone-nets (*CNet Construct Conical Support*) or parallel meshes (*PQ Mesh Construct Support*).

Mesh parallelism. Upon successful construction of a discrete cone-net, we can take advantage of its exact PQ structure and use mesh parallelism (Combescure transformations) to expand the design space [12]. Nevertheless, it is worth noting that not all regular PQ meshes are parallel to discrete cone-nets. This parallelism also applies to the generation of secondary layers, where the directions of



(a) Manual design of a catamaran hull as a cone-net. Tuning the ratio parameters allows better approximating the reference surface (left) after cutting and symmetrizing the cone-net with a plane.



(b) Pedestrian bridge. Our construction ensures the deck, the girders, and the glass panels are developable.

Figure 10: Case studies demonstrating the applications of the *CNets* plugin.

torsion-free node-axes remain torsion-free for the transformed parallel mesh.

The component *PQ Mesh Parallel Transform* supports the exploration of families of parallel meshes using two arrays of values that define the scale factors applied to the edges of the two boundary polylines $\mathbf{G}_{:,0}$ and $\mathbf{G}_{0,:}$, see Figure 9c. Unless the scale factors of the edges of $\mathbf{G}_{:,0}$ are all the same, the resulting transformed mesh will in general not be a cone-net.

Fabrication support. Once a design is successfully completed, it may be desirable to fabricate the resulting meshes from flat materials. The *CNets plug-in* offers a component for extracting PQ strips in either direction from the resulting PQ mesh (*PQ Mesh Extract Strips*), as well as a component for unrolling PQ strips (*PQ Strip Unroll*). The latter component can also be used to unroll a computed support structure.

4.5 Case Studies

We present potential use cases of our plugin, highlighting the versatility of the attainable shapes.

Figure 10a illustrates the design of a catamaran wave-piercing hull. Starting from a target geometry,

the *CNets Construct* component was initially used with uniform ratios of 0.5 to generate a rough approximation of the shape. In a second step, the ratios along each directrix-aligned polyline were manually adjusted to improve alignment between the cone-net and the target geometry. The resulting structure demonstrates very similar buoyancy and volume distribution and closely matches the intended form, enabling simplified manufacturing and reducing both costs and material waste compared to conventional fabrication methods such as in-mould composite laminations.

Figure 1 presents an architectural facade designed using the *CNets Construct* plugin, which enables cost-efficient fabrication through the use of planar faces. Specifically, our tool allows sweeping non-manifold generatrices and additionally preserving planarity for support structures in the constructed geometry.

Figure 10b shows a pedestrian bridge design. Here, the *CNets Construct Coplanar* component was used to generate both the support and glass elements of the structure. The outward bulge serves as a lookout point to enjoy the scenic views.

5 Conclusion and Future Work

This paper presents an intuitive framework for the forward design of cone-nets, along with an implementation of the construction and supplementary tools as a plugin for Grasshopper/Rhino.

Through a series of examples, we demonstrate how the *CNets* plug-in supports the design of meshes with exact planar quads, which, at high resolution, approximate structures composed of developable strips. By integrating our implementation into a widely used computational design platform, we make theoretical concepts accessible for practical applications.

While it is possible to optimize designs to achieve simple targets, a more sophisticated integration of optimization tools within Grasshopper/Rhino could further benefit the application. Additionally, exploring and streamlining the combination of multiple cone-nets could further expand the design space.

6 Acknowledgements

This research was funded by the Swiss National Science Foundation (SNSF) grants 200021-231293 and 200021-188582. Klara Mundilova is supported by the Swiss Government Excellence Scholarship.

References

[1] Alexander I Bobenko and Yuri B Suris. *Discrete differential geometry: Integrable structure*, volume 98. American Mathematical Soc., 2008.

[2] Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. Shape-up: Shaping discrete geometry with projections. *Comput. Graph. Forum*, 31(5):1657–1667, August 2012.

[3] James Glymph, Dennis Shelden, Cristiano Ceccato, Judith Mussel, and Hans Schober. A parametric strategy for free-form glass structures using quadrilateral planar facets. *Automation in Construction*, 13(2):187–202, 2004. Conference of the Association for Computer Aided Design in Architecture.

[4] Francisco Gonzalez-Quintial and Andrés Martín-Pastor. Monge Surfaces. Generation, Discretisation and Application in Architecture. *Nexus Network Journal*, 26(4):811–828, December 2024.

[5] Caigui Jiang, Cheng Wang, Xavier Tellier, Johannes Wallner, and Helmut Pottmann. Planar panels and planar supporting beams in architectural structures. *ACM Trans. Graphics*, 40(4):42:1–42:12, 2022.

[6] Martin Kilian, Christian Müller, and Jonas Tervooren. Smooth and Discrete Cone-Nets. *Results in Mathematics*, 78(3):110, June 2023.

[7] Yang Liu, Helmut Pottmann, Johannes Wallner, Yong-Liang Yang, and Wenping Wang. Geometric modeling with conical meshes and developable surfaces. *ACM Transactions on Graphics*, 25(3):681–689, July 2006.

[8] Romain Mesnil, Cyril Douthe, Olivier Baverel, and Bruno Leger. Marionette Meshes: Modelling free-form architecture with planar facets. *International Journal of Space Structures*, 32(3-4):184–198, June 2017.

[9] Romain Mesnil, Cyril Douthe, Olivier Baverel, Bruno Léger, and Jean-François Caron. Isogonal moulding surfaces: a family of shapes for high node congruence in free-form structures. *Automation in Construction*, 59:38–47, 2015.

[10] Klara Mundilova, Erik D. Demaine, and Tomohiro Tachi. Interactive curved crease origami design through constructive developable subdivision. In *Advances in Architectural Geometry 2025*, 2025. In press.

[11] Helmut Pottmann, Michael Eigensatz, Amir Vaxman, and Johannes Wallner. Architectural geometry. *Computers & graphics*, 47:145–164, 2015.

[12] Helmut Pottmann, Yang Liu, Johannes Wallner, Alexander Bobenko, and Wenping Wang. Geometry of multi-layer freeform structures for architecture. In *ACM SIGGRAPH 2007 papers*, pages 65–es. 2007.

[13] Helmut Pottmann and Johannes Wallner. The focal geometry of circular and conical meshes. *Advances in Computational Mathematics*, 29(3):249–268, October 2008.

[14] Rechenraum Simon Flöry. Goat. `https://www.rechenraum.com/goat/download.html`, 2016. Version 3.0.

[15] Xavier Tellier, Cyril Douthe, Laurent Hauswirth, and Olivier Baverel. Surfaces with planar curvature lines: Discretization, generation and application to the rationalization of curved architectural envelopes. *Automation in Construction*, 106:102880, October 2019.

[16] Michele Vidulis, Klara Mundilova, Quentin Becker, Florin Isvoranu, and Mark Pauly. C-Tubes: Design and Optimization of Tubular Structures Composed of Developable Strips. *ACM Transactions on Graphics*, 44(4):154:1–154:19, August 2025.

A Correction of Undesirable Configurations

In the following, we consider ways to resolve configurations where for all $j < j_0$ we have $c_{i,j} = (++)$, but there is at least one i_0 with $c_{i_0,j_0} = (--)$. The presented modifications do not affect the overall complexity of the construction which is still linear in the number of vertices of the resulting mesh.

Modification of the apex-locating function. One way to resolve the undesired case of $c_{i_0,j_0} = (--)$ is to move the vertex $\mathbf{G}_{i_0,j_0}$ toward $\mathbf{X}_{j_0}$ by modifying the value of the apex-locating function. Since $c_{i_0,j_0-1} = (++)$, this adjustment corresponds to reducing a_{j_0}. Specifically, we aim to choose a reduced, positive value for a_{j_0} such that all points $\mathbf{G}_{i,j_0}$ satisfy Equation (2) for the corresponding i. Note that modifying a_{j_0} affects all the vertices $\mathbf{G}_{i,j_0}$ at once.

An appropriate value of a_{j_0} can be determined as follows. Let I_{bad} denote the set of indices i, for which the constraint in Equation (2) for $\mathbf{G}_{i,j_0}$ is not satisfied. For all $i \in I_{\text{bad}}$, we parametrize the vertex $\mathbf{G}_{i,j_0}(a)$ in terms of the value of the apex-locating scalar a, that is,

$$\mathbf{G}_{i,j_0}(a) = \mathbf{G}_{i,j_0-1} + \frac{(\mathbf{X}_{j_0} - \mathbf{G}_{i,j_0-1}) \cdot \mathbf{P}_{i,j_0}}{(a(\mathbf{G}_{i,j_0-1} - \mathbf{X}_{j_0-1}) + \mathbf{X}_{j_0} - \mathbf{G}_{i,j_0-1}) \cdot \mathbf{P}_{i,j_0}} \left(a(\mathbf{G}_{i,j_0-1} - \mathbf{X}_{j_0-1}) + \mathbf{X}_{j_0} - \mathbf{G}_{i,j_0-1}\right).$$

and compute the value a_{i,j_0} such that

$$(\mathbf{X}_{j_0+1} - \mathbf{G}_{i,j_0}(a_{i,j_0})) \cdot \mathbf{P}_{i,j_0+1} = 0,$$

that is, the values a_{i,j_0} for which $\mathbf{G}_{i,j_0}(a_{i,j_0})$ lies in Π_{i,j_0+1}. Simplification of this condition results in

$$a_{i,j_0} = \frac{(\mathbf{G}_{i,j_0-1} - \mathbf{X}_{j_0}) \cdot (K_1\mathbf{P}_{i,j_0} + K_2\mathbf{P}_{i,j_0+1})}{(\mathbf{G}_{i,j_0-1} - \mathbf{X}_{j_0-1}) \cdot (K_1\mathbf{P}_{i,j_0} + K_2\mathbf{P}_{i,j_0+1})},$$

where

$$K_1 = (\mathbf{G}_{i,j_0-1} - \mathbf{X}_{j_0+1}) \cdot \mathbf{P}_{i,j_0+1} \quad \text{and} \quad K_2 = -(\mathbf{G}_{i,j_0-1} - \mathbf{X}_{j_0}) \cdot \mathbf{P}_{i,j_0}.$$

Finally, we set $a'_{j_0} = \min_{i \in I_{\text{bad}}} a_{i,j_0}$. If $a'_{j_0} > 0$, we update the apex-locating function to $(1 - \varepsilon)a'_{j_0}$, where $\varepsilon > 0$ is a small constant introduced to ensure that the mesh vertices do not coincide; otherwise a valid configuration cannot be recovered, and the user is invited to provide a different set of construction inputs.

Modification of the next construction plane. Another way to resolve the undesired configuration in which a vertex $\mathbf{G}_{i_0,j_0}$ lies on the wrong side of Π_{i_0,j_0+1} is to adjust the construction plane itself. Specifically, Π_{i_0,j_0+1} can be rotated about the axis defined by the cross product of the vectors $\mathbf{X}_{j_0} - \mathbf{X}_{j_0-1}$ and $\mathbf{G}_{i_0,j_0-1} - \mathbf{X}_{j_0-1}$ until $\mathbf{G}_{i_0,j_0}$ lies on the desired side. Unlike the case of the apex-locating function, this correction only affects the vertices of directrix-aligned polylines.

Modification of both apex-locating function and next construction plane. Adjusting the apex-locating function influences only the value of a single apex-locating function, whereas modifying a single ruling plane, especially in high-resolution meshes, can trigger a cascade of changes, resulting in global alterations to the construction planes. In some cases, a combined approach may be preferable: slightly adjusting the ruling plane, followed by fine-tuning the apex-locating function to resolve any remaining error. In Figure 5c we compare the two corrective options introduced above with this hybrid approach.

B Toward High-Level Optimization of *CNets*

The Grasshopper environment provides several optimization tools, such as Goat [14]. In our initial experiments, we used interpolating functions defined by a small number of control points as variables for both the apex-locating function and the ratio functions. By applying a simple custom goal that penalizes distances between points, this approach can optimize the input variables to align the end polygon with selected vertices, see Figure 11. Note that when the apex-locating function is fixed, the geometry of the support patches behaves independently, allowing optimizations to be performed separately for each directrix-aligned curve, thereby improving efficiency.

However, more advanced objectives, such as enforcing tangent continuity or face normal alignment, remain challenging to implement within this framework.

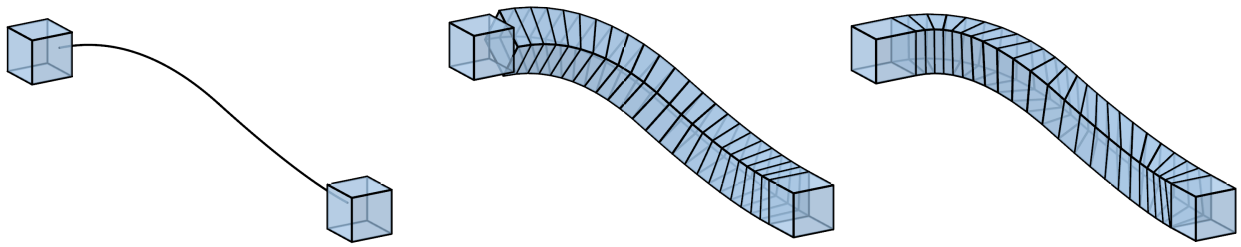


Figure 11: Illustration of how the Grasshopper plugin Goat was used to align the ends of a tubular cone-net with target points: the input directrix (left), a Monge tube with misaligned endpoints (center), and the optimized tubular cone-net (right) with $\mathbf{a} \equiv 1$, obtained by adjusting five ratio parameters per polyline.