Computers and Structures 292 (2024) 107238

Contents lists available at ScienceDirect

Computers and Structures

journal homepage: www.elsevier.com/locate/compstruc

Geometric deep learning for statics-aware grid shells

Andrea Favilli [a,b], Francesco Laccone [a,*], Paolo Cignoni [a], Luigi Malomo [a], Daniela Giorgi [a]

[a] Institute of Information Science and Technologies "A. Faedo" (ISTI), National Research Council of Italy (CNR), via G. Moruzzi 1, Pisa, 56124, Italy
[b] University of Pisa (Italy), Lungarno Pacinotti 43, Pisa, 56126, Italy

ARTICLE INFO

Dataset link: https://github.com/cnr-isti-vclab/GeomDL4GridShell#geometric-deep-learning-for-statics-aware-grid-shells

Keywords:
Grid shell
Freeform surface
Form finding
Shape optimization
Structural design
Automatic differentiation

ABSTRACT

This paper introduces a novel method for shape optimization and form-finding of free-form, triangular grid shells, based on geometric deep learning. We define an architecture which consumes a 3D mesh representing the initial design of a free-form grid shell, and outputs vertex displacements to get an optimized grid shell that minimizes structural compliance, while preserving design intent. The main ingredients of the architecture are layers that produce deep vertex embeddings from geometric input features, and a differentiable loss implementing structural analysis. We evaluate the method performance on a benchmark of eighteen free-form grid shell structures characterized by various size, geometry, and tessellation. Our results demonstrate that our approach can solve the shape optimization and form finding problem for a diverse range of structures, more effectively and efficiently than existing common tools.

1. Introduction

Free-form surfaces are a major trend in contemporary architecture. However, while current tools well support the modeling of complex free-form geometries, their actual fabrication at architectural scale still poses many problems. Indeed, a design solution has to comply with structural constraints, including for example statics, space requirements, lighting, and cost. In the last fifteen years, it became apparent that many practical fabrication problems are of geometrical nature: architectural applications started drawing increasing attention from the geometry processing community, marking the birth of a new field called *architectural geometry*. Relevant works range from paneling [1] to complex equilibrium problems [2]. However, until now, the main ingredients in architectural geometry have been discrete differential geometry and numerical optimization [3], while the potential of artificial intelligence and of 3D deep learning, in particular, have been explored to a lesser degree.

In this paper, we leverage innovative geometric deep learning techniques and introduce a model to design free-form grid shells with optimized statics performance. We define a deep learning architecture which consumes a 3D mesh representing the initial design of a free-form grid shell and outputs vertex displacements to get a design-preserving, optimized, statics-aware grid shell (Fig. 1).

Grid shell structures are discrete networks of straight bars connected by joints at the nodes; they play an important role in free-form surface design, as they can cover large spaces while keeping the amount of material relatively small. We focus on statics, as equilibrium and stiffness are paramount to all architectural structures. Given a free-form surface design with boundary conditions, our model learns how to modify the surface shape to better balance forces, observe boundary conditions, and preserve the visual appearance of the original design. The main ingredients are a deep architecture able to learn on complex, unstructured 3D geometric data, and the definition of a differentiable loss accounting for statics-related factors; the differentiability of the loss allows learning the parameters of the 3D deep learning model via gradient descent and backward propagation. The result is the first 3D deep learning pipeline for statics-aware grid shell design.

The pipeline is shown in Fig. 2. Our proposal is to compute local geometric features that are relevant to architectural free-forms (including curvature and geodesic distances), and feed them to attention-based, neural message passing layers that learn optimal vertex displacements. The learning process is iterative, as the input to each step is the displaced output at the previous step. The process is guided by a loss evaluating the mean strain energy of the structure.

Our work fits within the trend of *fabrication-aware design*, a paradigm which fosters the development of digital design tools to assist users in modeling geometric shapes that automatically take into account fac-

* Corresponding author.
E-mail address: francesco.laccone@isti.cnr.it (F. Laccone).

https://doi.org/10.1016/j.compstruc.2023.107238
Received 8 June 2023; Accepted 15 November 2023
Available online 1 December 2023

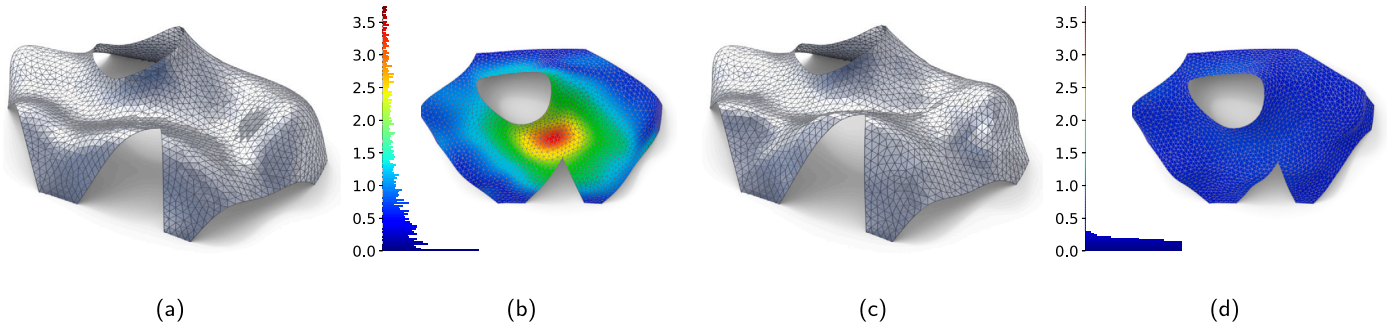

Fig. 1. (a) A 3D mesh representing a first design of a free-form grid-shell, and (b) the deformation it would undergo due to Service Load, in false colors; deformation is expressed in meters with frequency histograms per node. (c) The optimized, statics-aware grid-shell produced by our 3D deep learning technique, which preserves the design intent while reducing deformation (d).

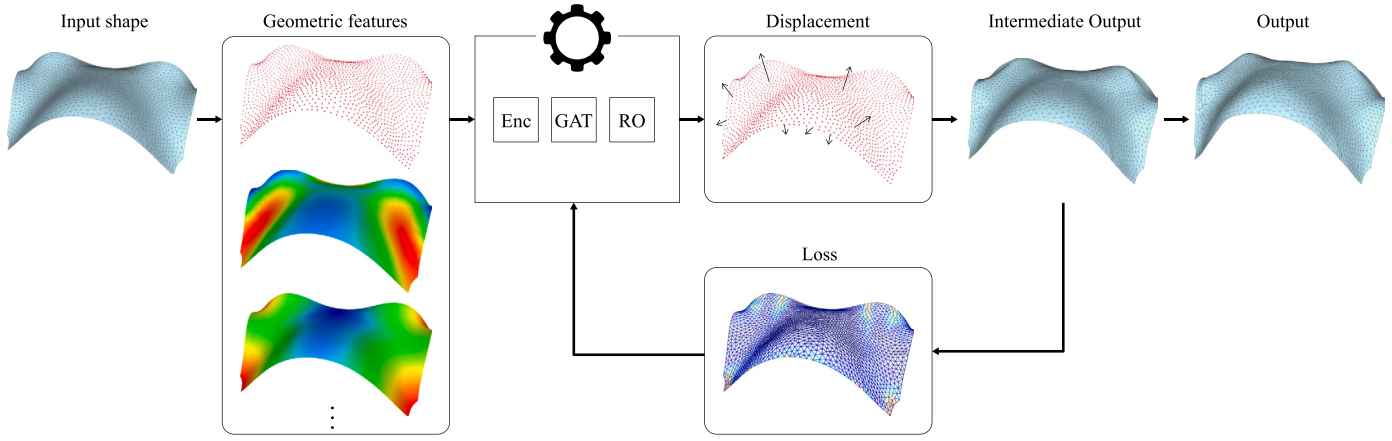

Fig. 2. Our method takes as input a 3D mesh, computes a set of geometric features (vertex coordinates, normals, curvature, geodesic distances) and feeds them to a network for 3D deep learning. The network is made up of an encoder (Enc), attention-based graph convolutional layers (GAT), and a readout function (RO) which outputs learned vertex displacements. The intermediate output is the displaced mesh. A differentiable loss evaluates the mean strain energy over the edges of the displaced mesh, while gradient descent and backward propagation enable the network to learn the optimal vertex displacements. The network learns on the single mesh itself, as the input to each iteration is the intermediate output at the previous iteration until convergence.

tors related to buildability and costs, such as statics. The goal is to shorten the classical design loop, in which a first design is analyzed, feedback is provided to the designer, and a second loop is started. A tool that can automatically correct shape and preserve the design intent would significantly reduce the number of iterations and, consequently, the time needed to finalize the design. In our context, statics-aware grid shell design implies finding structures that are able to withstand specific loads without stresses or deformations exceeding safe limits [4–6]. A big challenge is to find a good trade-off between statics and visual fidelity. Our method enables one to automatically optimize the shape of an input design, without the need for tweaking different design parameters until the requirements are met, thus reducing the burden on the user's side: our neural network only requires a single parameter to set, namely the learning rate. We believe this is an important methodological step towards advancing the state of the art on computational design of free-form surfaces.

The search for shell shapes with predominantly in-plane or membrane stress is a process known as *form finding*, in which the geometry is controlled to be in equilibrium for given loads and boundary conditions [7]. Thus, forces shape geometry and vice versa. Form finding traces back to Hooke's law, clamped beam problem formulated by Galileo Galilei, and it was traditionally performed via the physical technique of 'hanging models', in which deformable meshes and elastic membranes are hung from the supports under their own weight. The form obtained is ideal for supporting tension, and if reversed with respect to the horizontal plane, it leads to a purely compressed structure with no bending. This idea inspired designers like Gaudi, Isler, Otto, and also originated different computational form finding tools [8,9]. Most modern techniques, however, are based on *shape optimization* and the minimization of a fitness criterion, such as the strain energy of the structural system. For discrete structures such as grid shells, the variables of the optimization problem are the node coordinates. The topology, in terms of the number and connectivity of nodes, is fixed, unlike layout [10] and topology optimization [11] methods. Existing techniques include gradient-free techniques, such as genetic algorithms, and gradient-based optimizers [12–14].

Until now, machine learning techniques have only been proposed in architecture for other tasks, such as designing architectural materials by substituting costly experiments or physics-based simulations with neural networks [15]. Moreover, with the exception of [16] that consumes 3D point clouds to learn the mechanical response of deformable bodies, existing techniques only deal with simple domains and grid-like inputs, such as feature vectors and images [17]. In contrast, our method can learn on complex 3D free-form geometries, and generate variations with improved global statics performance. The main challenges are how to encode the local geometry of a 3D structure, predict its global statics behavior, and learn the local updates that improve global performance. We draw inspiration from the paradigm of *geometric deep learning* [18], which is gaining momentum in the computer graphics and vision communities to learn on complex data such as 3D point clouds and triangle meshes and solve problems such as 3D shape classification and segmentation [19]. However, we deal with a different problem, namely, a *single-instance learning* problem with no ground truth data, in which

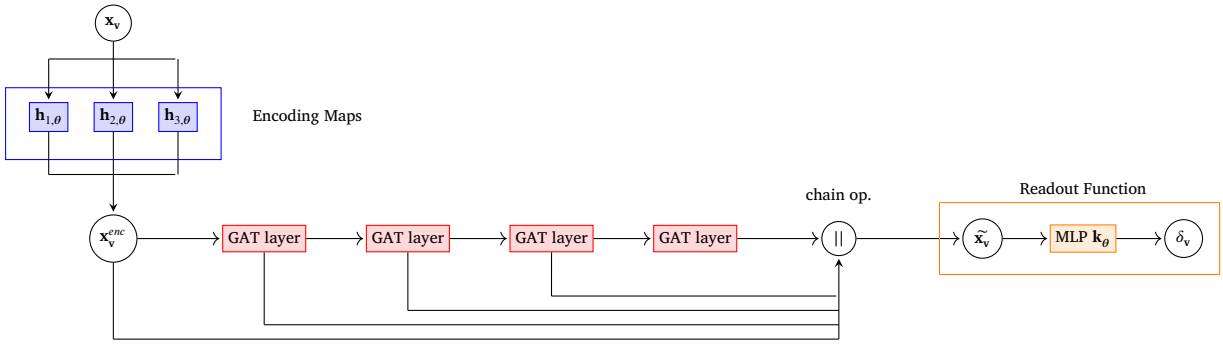

Fig. 3. The model architecture. The input to the network is per-vertex feature vectors $\mathbf{x}_v$, which are encoded in a higher-dimensional space. The encoded features $\mathbf{x}_v^{enc}$ are fed to four Graph Attention Layers (GAT), whose outputs are concatenated to get per-vertex deep vectors $\widetilde{\mathbf{x}}_v$. Finally, a readout function implemented as a multilayer perceptron maps deep features into per-vertex displacements δ_v.

learning takes place iteratively on a single item; this problem has received considerably less attention in the literature [20].

For gradient evaluation, we leverage the paradigm of Automatic Differentiation (AD), in which functions are expressed as sequences of basic operations and function derivatives are computed via the chain rule in calculus. The advantages of AD with respect to either numerical or symbolic derivation are accuracy and efficiency. Also, AD is suited to training neural architectures via backpropagation. Until now, AD in architectural design has been used for topology optimization [21], for optimizing shell structures expressed via NURBS (Non-Uniform Rational B-Splines) [22], and for form finding via CEM (Combinatorial Equilibrium Model) [23]. However, the problems considered have been simpler in terms of models, degrees of freedom, and the dimension of the searching space. Also, none of these works implements a fully end-to-end neural architecture [24]. In contrast, we design a network and demonstrate it can be trained even with a function whose computational graph is highly articulated, being the loss a strain energy whose computation requires solving linear systems. We also provide a fast implementation, which ensures reasonable inference times to get optimized designs.

To validate our method, we use eighteen complex free-form grid shell structures, characterized by various sizes, geometry, and tessellation. The results demonstrate that our approach can solve the shape optimization and form finding problem for diverse structures more effectively and efficiently than competitors.

2. Methods

Section 2.1 describes how we cast the problem of finding statics-aware grid shells in terms of deep learning on 3D meshes. The learning architecture is detailed in Section 2.2, while Section 2.3 describes how we define and compute the loss.

2.1. Problem formulation

This paper deals with triangular grid shells, that is, three-dimensional single-layer structures in which external loads are supported by a network of beams. A triangular grid shell can be encoded naturally as a triangle mesh $\mathcal{M} := (V, E, F)$, with V, E, F the sets of vertices, edges, and faces, respectively: mesh vertices are structural nodes, edges are beam elements, and faces are cladding panels.

We cast the problem of learning a statics-aware grid shell as a problem of learning mesh vertex displacements, which minimize a loss function

$$\mathcal{L}(\mathcal{M}) := \frac{1}{|E|} \sum_{e \in E} S_e \tag{1}$$

representing the *mean strain energy* over mesh edges. The strain energy S_e on edge e is the total stress acting on the corresponding beam. We consider an orthonormal local frame $\{\mathbf{x}_e, \mathbf{y}_e, \mathbf{z}_e\} := \{\mathbf{e}, \mathbf{n}_e, \mathbf{e} \times \mathbf{n}_e\}$, where $\mathbf{e}$ is the edge direction and $\mathbf{n}_e$ the edge normal. Then, S_e is given by the sum of six components (or *degrees of freedom*): axial elongation and torsion along $\mathbf{x}_e$, and transverse bending and shear along planes $\{\mathbf{x}_e, \mathbf{y}_e\}$ and $\{\mathbf{x}_e, \mathbf{z}_e\}$ [25]. We assume part of the boundary ∂V is free to move, while a subset $\overline{\partial V} \subset \partial V$ of boundary vertices are prescribed as fixed.

A classical approach would optimize directly on mesh vertex displacements. However, the locality of the approach can lead to suboptimal solutions and result in noisy and jagged meshes. Therefore, we switch from a vertex-centric to a mesh-centric approach, resorting to geometric deep learning. In particular, we define a neural model T_θ (Section 2.2), and formulate the problem as learning the optimal set of network parameters that minimize the loss. In other words, we formulate the problem as searching for

$$\mathcal{M}^* := T_{\theta^*}(\mathcal{M})$$

such that

$$\theta^* \in \operatorname{argmin}_\theta \mathcal{L}(T_\theta(M)). \tag{2}$$

We underline that our goal is to improve the grid shell efficiency by finding optimal vertex displacements while leaving the grid topology unchanged. In other words, the connectivity of beams is left unchanged during the learning and optimization procedure.

2.2. Deep learning model

Fig. 3 summarizes our deep learning architecture. The input to the network is a matrix $X \in \mathbb{R}^{|V| \times d}$, whose rows are per-vertex feature vectors $\mathbf{x}_v$, with d the number of encoding channels; the mesh connectivity is also fed as input. The features describe both positional properties (vertex coordinates and normals) and differential properties of the underlying manifold surface (curvature and geodesic distances), cf. Section 2.2.1. The feature vectors are fed to three feature encoders, then to a sequence of four Graph Attention (GAT) Layers [26,27] with skip connections.

GAT layers are applied to the k-nearest neighborhood graph in the vertex feature space; in other words, GAT layers update every vertex representation by aggregating the representations of its k neighboring vertices (cf. details in Section 2.2.2). GAT layers are preferred over classical Graph Neural Network (GNN) layers [28], as they learn a weighted aggregation of representations using attention scores, thus improving the expressing power of the neural message passing mechanism which underlies GNNs. Moreover, in our architecture, the graph is dynamically updated at each layer, that is, the graph is recomputed using nearest neighbors in the feature space produced by each layer [19]. The advantage is that our architecture *learns* the graph used at each layer, rather than assuming it fixed: information is passed considering not only geometric proximity but also semantic proximity in the space of learned features.

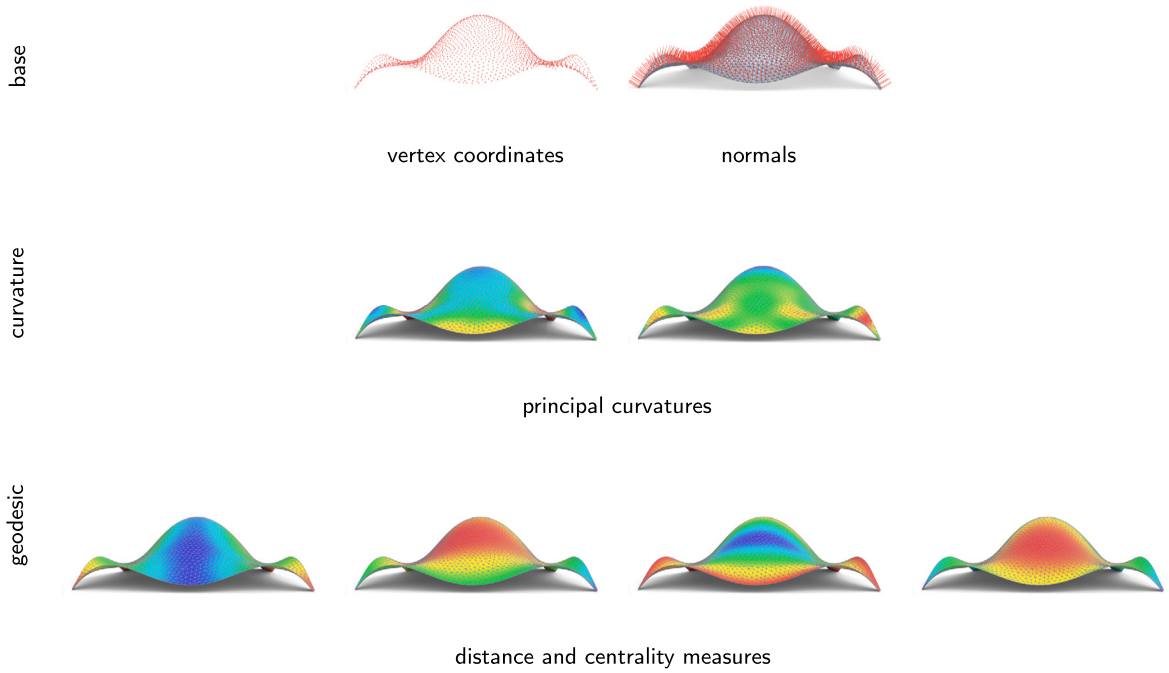

Fig. 4. Input features computed on model Station, in false colors: blue stands for lower values, red for higher values. Top: base features, including vertex coordinates and normals. Middle: the two surface principal curvatures. Bottom: from left to right, geodesic distance from boundary vertices, geodesic distance from fixed boundary vertices, vertex centrality with respect to the boundary and vertex centrality with respect to the fixed boundary.

Skip connections are implemented by concatenating each embedded nodal feature vector after every GAT layer into a final nodal embedding vector; this is done to retain local information from previous layers. Finally, a Readout Function, implemented as a shared Multi Layer Perceptron (MLP), outputs vertex displacements.

The loss evaluates the static performance of the mesh updated with vertex displacements, in terms of mean strain energy over edges, according to Eq. (1). The loss computation requires solving a linear system; we provide a differentiable and fast loss implementation, enabling us to plug it into the deep learning pipeline (Section 2.3). At each training iteration, the displaced mesh delivered at the previous iteration is fed as input to the network until convergence.

Our architecture is designed to take full advantage of the information encoded in the 3D mesh representation of grid shells. The whole mesh is used to evaluate discrete counterparts of differential properties of the underlying surface, used as input geometric features. Then, the set of mesh vertices defines the point cloud on which GAT Layers are applied; this time, connectivity is defined and updated according to proximity in the learned feature space, instead of using the connectivity of the original mesh. Finally, the wire skeleton made of vertices and edges is used to compute the loss, which is based on the structural analysis of the grid shell, whose load-bearing elements are uniquely the beams.

The following paragraphs detail the single steps in our model.

2.2.1. Input features

For each vertex $\mathbf{v} \in V$, we feed the network with an input feature vector $\mathbf{x}_\mathbf{v} \in \mathbb{R}^{12}$, whose entries are shape features relevant in the context of architectural freeforms. Features are partitioned into three clusters, namely *base*, *curvature* and *geodesic* features (cf. the ablation study in Section 3.4.2).

Base features include vertex coordinates $\mathbf{v} \in \mathbb{R}^3$ and vertex normals $\mathbf{n}(\mathbf{v}) \in \mathbb{R}^3$. Base features are local, extrinsic features, i.e., they are related to how the underlying surface is embedded in the Euclidean 3D space (Fig. 4, Top). Vertex coordinates and normals are the features traditionally considered for 3D deep learning [29]. To these, we add features describing differential properties of the underlying surface.

Curvature features include principal curvatures $\kappa_1(\mathbf{v})$, $\kappa_2(\mathbf{v})$, computed via APSS fitting [30,31] (Fig. 4, Middle). Curvatures and changes of curvatures are relevant as they influence the distribution of internal forces within the grid shell surface. The efficiency of a shell depends on the capacity of transferring the external load into in-plane (normal and shear) forces. As a result of curvature, the surface gains spatial stiffness and develops membrane behavior.

Geodesic features serve to feed the network with information about the relative position of vertices with respect to the structure boundary, since points farthest from the fixed boundary undergo higher load deformation (Fig. 4, Bottom). The first two geodesic features are the distance $d(\mathbf{v}, \partial V)$ of vertices from the set ∂V of boundary vertices and the distance $d(\mathbf{v}, \overline{\partial V})$ from the set $\overline{\partial V}$ of fixed boundary vertices, where $d(\mathbf{v}, X) := \min\{d(\mathbf{v}, \mathbf{w}) \mid \mathbf{w} \in X\}$ denotes the distance from a set X of vertices and $d(\mathbf{v}, \mathbf{w})$ is the geodesic distance between two mesh vertices, i.e. the length of the shortest edge path connecting them. The other two features are the vertex centrality measure $C_{\partial V}(\mathbf{v})$ with respect to the boundary, and the vertex centrality measure $C_{\overline{\partial V}}(\mathbf{v})$ with respect to the fixed boundary, where vertex centrality with respect to a set X is defined as

$$C_X(\mathbf{v}) := \sum_{\mathbf{w} \in X} d(\mathbf{v}, \mathbf{w}).$$

In contrast to base and curvature features, geodesic features are *intrinsic*, as they are related to the inner metric of the underlying manifold and are preserved by isometric deformations.

Smoothing iterations via neighborhood averaging are performed on both geometric and geodesic features.

Each of the three groups of features (which have different dimensions n_i, $i = 1, 2, 3$) are encoded to 256 channels via encoding maps

$$\mathbf{h}_{i,\theta}(\mathbf{x}) := \tanh(W_{i,\theta}\mathbf{x} + \mathbf{b}_{i,\theta})$$

with $W_{i,\theta} \in \mathbb{R}^{256 \times n_i}$ and $\mathbf{b}_{i,\theta} \in \mathbb{R}^{256}$; the hyperbolic tangent is employed to map channels onto the interval $(-1, 1)$.

The resulting encoded vector $\mathbf{x}_{\mathbf{v}}^{enc} \in \mathbb{R}^{768}$ is then fed to a sequence of four convolutional Graph Neural Network (GNN) layers working on the k-nearest neighbor graph induced by vertex feature spaces.

2.2.2. Graph convolution layers

To learn deep features on vertices, we employ graph convolutional layers defined on the set of mesh vertices. We adopt the GATv2 paradigm [27], and perform convolution using the k-nearest neighborhood graph in the vertex feature space. Let us denote $\mathbf{x}_{\mathbf{v}}^{\ell}$ the input features to layer ℓ for each vertex $\mathbf{v}$ ($\mathbf{x}_{\mathbf{v}}^{1} = \mathbf{x}_{\mathbf{v}}^{enc}$ for the first layer). Then, the updated vertex representation $\mathbf{x}_{\mathbf{v}}^{\ell+1}$ is a weighted average between $\mathbf{x}_{\mathbf{v}}^{\ell}$ itself and the features from the set $\mathcal{N}_{\mathbf{v}}$ of its k neighboring vertices ($k = 16$ in this paper). In formulae:

$$\mathbf{x}_{\mathbf{v}}^{\ell+1} := \alpha_{\mathbf{vv}} W_\theta \mathbf{x}_{\mathbf{v}}^{\ell} + \sum_{\mathbf{w} \in \mathcal{N}_{\mathbf{v}}} \alpha_{\mathbf{vw}} W_\theta \mathbf{x}_{\mathbf{w}}^{\ell} \tag{3}$$

where W_θ is a learned matrix of parameters, and the *attention weights* $\alpha_{\mathbf{vw}}$ are scalars in the interval $[0, 1]$ with $\sum_{\mathbf{w}} \alpha_{\mathbf{vw}} = 1$. Attention weights are computed from learned $\mathbf{a}_\theta, W_\theta$ as follows:

$$\alpha_{\mathbf{vw}} := \frac{\exp\left(\mathbf{a}_\theta^T \phi\left(W_\theta\left(\mathbf{x}_{\mathbf{v}} \,||\, \mathbf{x}_{\mathbf{w}}\right)\right)\right)}{\sum_{\mathbf{w} \in \mathcal{N}_{\mathbf{v}} \cup \{\mathbf{v}\}} \exp\left(\mathbf{a}_\theta^T \phi\left(W_\theta\left(\mathbf{x}_{\mathbf{v}} \,||\, \mathbf{x}_{\mathbf{w}}\right)\right)\right)} \tag{4}$$

with ϕ a *leaky ReLU* activation function. Attention weights improve the expressing power of the neural message passing mechanism (cf. the ablation study in Section 3.4.3). Moreover, we use *multi-head* attention to enhance model stability [26,27]: the output of a layer is averaged from the results of a given number of feature transforms (heads), each one with its own set of learned parameters $\mathbf{a}_\theta, W_\theta$. Five heads are used in this paper.

The neighboring vertices in $\mathcal{N}_{\mathbf{v}}$ are computed using the Euclidean distance in the space of features. The set of k neighboring vertices is dynamically updated at each layer, since distances are recomputed in the feature space learned at the previous layer. Therefore, the connectivity of the graph on which convolution takes place is *learned* at each layer, and information is shared among semantically-close vertices.

The four 256-dimensional GAT layer outputs are concatenated with $\mathbf{x}_{\mathbf{v}}^{enc}$ to get a deep feature vector $\widetilde{\mathbf{x}_{\mathbf{v}}} \in \mathbb{R}^{1792}$. Finally, a multilayer perceptron $\mathbf{k}_\theta$ trained on the whole mesh transforms the deep vector $\widetilde{\mathbf{x}_{\mathbf{v}}}$ into a vertex displacement $\delta_{\mathbf{v}} \in \mathbb{R}^3$.

Since we are performing single-instance learning, the optimal parameters $\boldsymbol{\theta}$ of the model T_θ in Eq. (2) are found via an iterative training on the mesh itself, by minimizing the loss $\mathcal{L}$. The final, displaced mesh is given by $\mathcal{M}^* = T_\theta(\mathcal{M})$ after the last training step.

2.3. Strain energy loss

In our model, the loss evaluates the static performance of the grid shell in terms of mean strain energy over mesh edges (Eq. (1)). A major challenge is how to implement the loss to be differentiable with respect to vertex coordinates, so that it can be plugged into the deep learning architecture, and gradient-based descent can be used to learn optimal vertex displacements. In this Section, we first introduce the linear system we solve to evaluate the loss, then give the implementation details.

Note that we use the term *deformation* to denote the effect of load on vertices, while the term *displacement* indicates the vertex coordinate changes learned by the network and applied to the structure.

2.3.1. Loss definition

The strain energy S_e on an edge e representing a beam depends on the forces and moments $\mathbf{F}^e = [\mathbf{F}_1^e, \mathbf{F}_2^e] \in \mathbb{R}^{12}$ acting on the two beam endpoints, in the local frame. The forces are related to the vector beam deformation at endpoints $\mathbf{u}^e := \left[\mathbf{u}_1^e, \mathbf{u}_2^e\right] \in \mathbb{R}^{12}$ through the *beam stiffness* in a *linear system*:

$$\mathbf{F}^e = K^e \mathbf{u}^e, \tag{5}$$

with the *beam stiffness matrix* $K^e \in \mathbb{R}^{12\times12}$ having a known sparse structure whose non-zero entries depend on beam material and geometry [25]. The global load-deformation relation on the whole grid shell structure is described by the *global stiffness linear system*

$$\mathbf{F} = K\mathbf{u} \tag{6}$$

where $K \in \mathbb{R}^{6|V|\times6|V|}$ is the sparse *global stiffness matrix*, $\mathbf{F} \in \mathbb{R}^{6|V|}$ is the vector of forces and moments in the observer's reference system, and $\mathbf{u} \in \mathbb{R}^{6|V|}$ is the vector of vertex deformations. The global stiffness matrix K can be built by summing up contributions from the matrices $\hat{K}^e$ obtained from the local stiffness matrices K^e after a change of coordinates from the local frame to the observer's reference frame (Fig. 5). Load is uniform and directed along the observer's $\mathbf{z}$ axis, in the direction of gravity. Therefore, we express the load inside $\mathbf{F}$ as an external force acting on the structure nodes simulating a common Service Load. To ensure no deformation for constrained degrees of freedom, we reduce Eq. (6) by removing their corresponding rows and columns. After solving the reduced linear system for $\mathbf{u}$, the inverse change of basis yields $\mathbf{u}^e$ and $\mathbf{F}^e$ via Eq. (5). Finally, since we adopt a 2-node beam formulation, in which the internal forces are computed only at the endpoints, the discretized strain energy S_e is constant over the half-length of the beam, and can be computed as the sum of the six components:

$$\begin{aligned}
S^e_{axial} &= \frac{L_e}{4} \cdot \frac{(F_{x,1}-F_{x,2})^2}{YA} & S^e_{shear_y} &= \frac{L_e}{4} \cdot \frac{(F_{y,1}-F_{y,2})^2}{GA} \\
S^e_{bend_y} &= \frac{L_e}{4} \cdot \frac{(M_{y,1}-M_{y,2})^2}{YI_y} & S^e_{shear_z} &= \frac{L_e}{4} \cdot \frac{(F_{z,1}-F_{z,2})^2}{GA} \\
S^e_{bend_z} &= \frac{L_e}{4} \cdot \frac{(M_{z,1}-M_{z,2})^2}{YI_z} & S^e_{tors_x} &= \frac{L_e}{4} \cdot \frac{(M_{x,1}-M_{x,2})^2}{GI_x},
\end{aligned} \tag{7}$$

where we called $F_{x,i}$, $F_{y,i}$, $F_{z,i}$, $M_{x,i}$, $M_{y,i}$, $M_{z,i}$ the components of $\mathbf{F}_i^e$ for $i = 1, 2$, L_e the beam length, A the beam cross-section area, Y the *Young's modulus*, G the *Poisson's ratio* and I_x, I_y, I_z the second moments of area.

The final loss is given by the average strain energy over mesh edges.

We do not add to the loss explicit constraints to control adherence to the input geometry, such as a term which measures the Chamfer distance between the original and the displaced mesh [20]. Instead, we let the shape change at each iteration, being controlled by the learning rate only. This strategy is an advantage, as it enables us to address both problems of shape optimization from an input design and form finding from the flat (cf. the results in Section 3).

2.3.2. Boundary constraints

The final problem to solve is how to ensure that vertex displacements $\delta_{\mathbf{v}}$ are null for the subset of prescribed fixed vertices $\mathbf{v} \in \overline{\partial V}$.

Imposing a hard constraint $\delta_{\mathbf{v}} = 0$ for prescribed nodes at each training iteration may affect boundary fairness. Therefore, we opt for a soft constraint and introduce a penalty term

$$\mathcal{B} := \frac{1}{|\overline{\partial V}|} \sum_{\mathbf{v} \in \overline{V}} ||\delta_{\mathbf{v}}|| \tag{8}$$

so that the loss to minimize becomes

$$\mathcal{L} + \zeta\mathcal{B} \tag{9}$$

with the weight $\zeta > 0$. The weight is set so that the penalty term at the first step is a fixed percentage of the initial loss (30% in our implementation). This ensures that vertex displacements are kept small during the training, and can be simply set to zero at the final iteration.

2.3.3. Differentiable loss implementation

The loss is implemented in PyTorch [32]. For a fast computation, we order the six degrees of freedom and the edge endpoints in accordance with the ordering of vertices in the mesh data structure, so that the non-zero entries in the global stiffness matrix contributed by each beam are known in advance, and remain constant as all steps in our pipeline preserve mesh connectivity. Then, we perform coordinate changes and

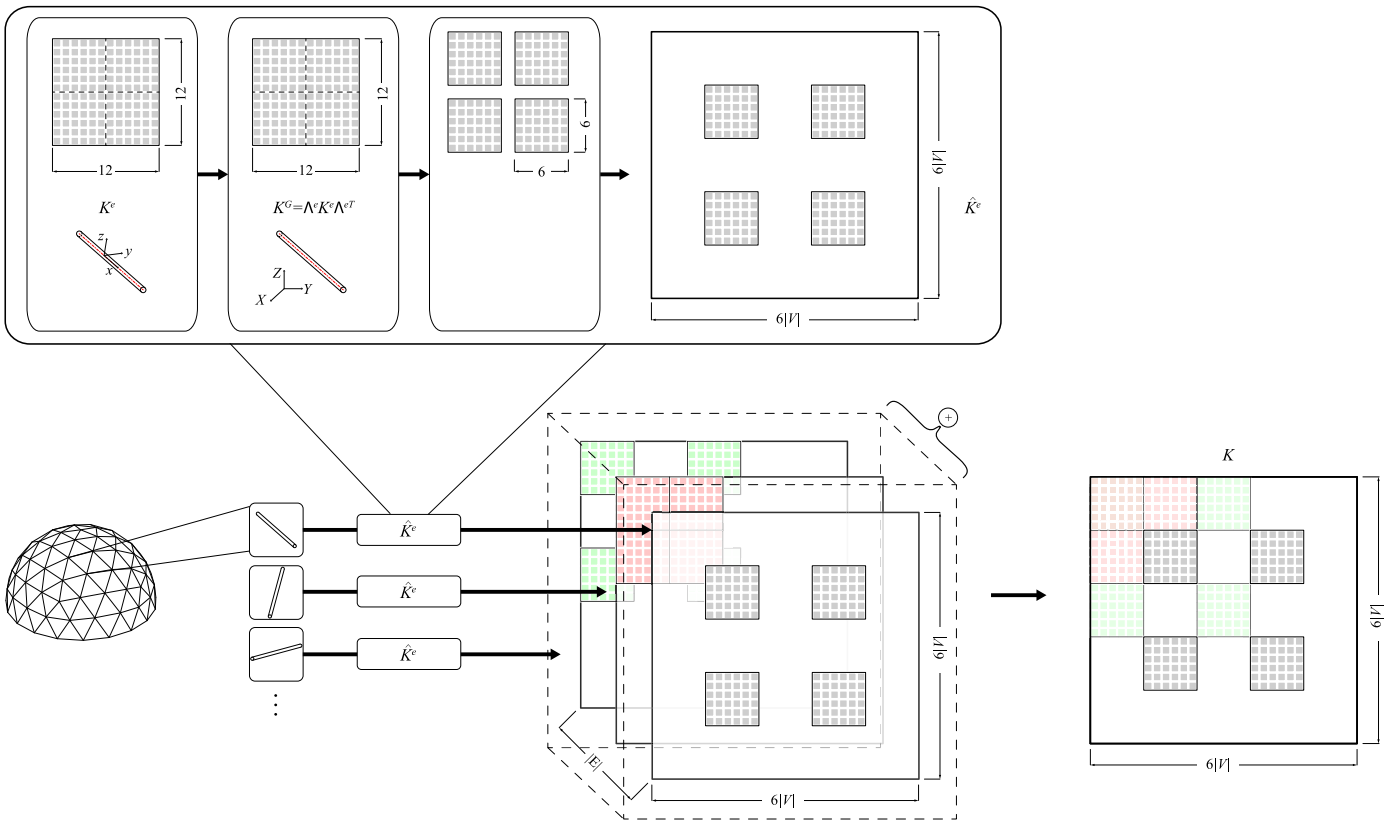

Fig. 5. For each beam, we compute the beam stiffness matrix $K^e \in \mathbb{R}^{12\times12}$ in the local reference frame, and apply a change of coordinates from the local frame to the observer's reference frame. The four 6×6 blocs, corresponding to the 6 degrees of freedom and the 2 beam endpoints, are arranged in a $6|V|\times6|V|$ matrix $\hat{K}^e$ according to the indices of the beam endpoints in the mesh data structure. Finally, the global stiffness matrix $K \in \mathbb{R}^{6|V|\times6|V|}$ is the sum of all per-beam matrices $\hat{K}^e$.

sums of beam contributions in parallel for each edge: we stack all the local stiffness matrices K^e and the coordinate change matrices Λ^e along the first dimension, to get three-dimensional tensors $\mathbf{K}, \mathbf{\Lambda} \in \mathbb{R}^{|E|\times12\times12}$. The coordinate changes from local to global are computed via matrix multiplication as $\mathbf{\Lambda K \Lambda}^T$ via PyTorch 3D tensor product. Subsequently, to compute the stiffness matrix K, we take advantage of PyTorch sparse tensors to assemble in parallel all edge contributions. PyTorch sparse tensors admit multiple non-zero entries at a same position, and switching from sparse to dense automatically solves collisions by summing colliding entries. Therefore, we initialize a PyTorch sparse tensor in which the positions of non-zero entries are known from the ordering of mesh vertices and filled with the flattened $\mathbf{\Lambda K \Lambda}^T$. The global stiffness matrix K is obtained after switching the tensor from sparse to dense. Finally, we compute the force vectors $\mathbf{F}^e$ in parallel as $(\mathbf{K\Lambda}^T)\mathbf{U}$, where $\mathbf{U} \in \mathbb{R}^{|E|\times12\times1}$ is the beam endpoint deformation stacked per edge along the first dimension, and $\mathbf{K\Lambda}^T$ was previously stored as a partial computation.

Our implementation ensures the loss is differentiable with respect to vertex coordinates, and that Automatic Differentiation can be used for backward propagation. Also, the loss is tensor-based and therefore fast to compute and highly parallelizable on GPUs. This strategy ensures reasonable timings for design optimization at inference time (cf. the timings reported in the next Section).

3. Results

To discuss the effectiveness and relevance of our method, we analyze fifteen free-form grid shells. Section 3.1 introduces the dataset and the experimental settings. Section 3.2 presents qualitative and quantitative results on some models. Section 3.3 compares our results with popular form finding tools (Kangaroo and Karamba). Finally, Section 3.4 presents an ablation study to discuss some of our design choices about the neural architecture. The complete results on the whole dataset are reported in the Appendix A.

3.1. Dataset and experimental settings

Tale 1 reports the main statistics about the fifteen models analyzed, in terms of mesh size, average edge length, area of the grid shell surface, area of the beam cross section. The structures analyzed are characterized by various sizes, geometric features, and triangular tessellations. For all cases, we fix all degrees of freedom on the ground, while we do not restrain any movement along the other boundaries, such as holes and openings. All examples are subject to a vertical loading of 3 kN/m^2, in the gravity direction, which is distributed on the nodes by tributary area computed through the Voronoi graph of the triangular mesh. Additionally, the weight of the beams is included as a lumped load, assuming the material density as $78.5kN/m^3$. This scenario simulates a Serviceability Limit State. We adopt two solid circular cross section types: $0.017m$ tube radius for the smaller examples, and 0.08 m tube radius for the larger examples.

The different cross section radius and the different size of the bounding box for the two types of meshes change the scale of the initial loss during the gradient descent. Therefore, we tune the learning rate to 0.01 for the cross section of radius 0.017 m and 0.0005 for the cross section of radius 0.08 m. The optimization is performed via Stochastic Gradient Descent (SGD); the stopping criterion has a relative component (difference between the maximum and the minimum loss of $0.1 \cdot \mathcal{L}_0$) and an absolute component (loss variation of 0.005 in the last 50 iterations). We employed a Microsoft Windows® 10 Pro machine with a i7-6700K CPU, 32 GB of RAM, a NVIDIA GeForce GTX 1080 GPU with 8 GB of dedicated memory.

Table 1
Dataset models: model name; average edge length $\overline{L_e}$; area of the grid shell surface A_{tot}; beam cross section radius C; number of vertices; number of edges; size of the axis-aligned bounding box.

name	scale metrics					
	$\overline{L_e}$ (m)	A_{tot} (m^2)	C (m)	verts (#)	edges (#)	size (m x m x m)
2Spheres	1.03	1197.47	0.017	1373	3989	34.23 x 30.56 x 11.38
Blob	1.06	2381.45	0.017	2551	7482	55.51 x 34.98 x 19.57
Botanic	0.97	918.11	0.017	1121	3272	30.80 x 28.57 x 7.69
CreaseShell	2.02	1487.93	0.017	465	1304	36.75 x 33.08 x 17.19
Gopher	0.72	704.13	0.017	1709	4888	20.26 x 29.09 x 12.12
Neumunster	0.81	514.63	0.017	1209	3440	31.11 x 15.36 x 2.96
QuadFlat	0.68	259.81	0.017	780	2229	15.00 x 17.32 x 0.00
Ruled	2.01	270.27	0.017	99	258	14.77 x 18.71 x 4.13
Wave	1.02	958.08	0.017	1130	3239	28.27 x 49.03 x 9.91
BoonieBraced	2.48	1150.63	0.08	284	803	36.48 x 33.13 x 8.90
BoonieChaotic	2.38	1190.88	0.08	308	878	36.48 x 36.45 x 8.93
BubbleShell	1.52	1190.88	0.08	1530	4394	61.50 x 48.56 x 20.40
Hall	1.74	1584.07	0.08	645	1846	39.74 x 38.59 x 13.53
HallCrease	1.72	1790.21	0.08	753	2159	41.72 x 40.36 x 14.21
Envelope	1.72	3041.46	0.08	1257	3626	57.06 x 55.71 x 20.16
Station	1.74	4368.66	0.08	1773	5151	84.77 x 52.49 x 23.93
Tent	1.74	4302.68	0.08	1785	5133	87.05 x 59.25 x 21.64
Yarn	1.74	3059.84	0.08	1233	3561	51.28 x 71.56 x 26.18

Table 2
Results: model name; learning rate; number of gradient descent iterations; average loss gradient computation time $\overline{T_\nabla}$ and average iteration time $\overline{T_{it}}$ on both GPU and CPU executions.

name	learning rate	iterations	average times			
			GPU		CPU	
		(#)	$\overline{T_\nabla}$ (s)	$\overline{T_{it}}$ (s)	$\overline{T_\nabla}$ (s)	$\overline{T_{it}}$ (s)
2Spheres	0.01	197	0.14	0.59	1.16	4.00
Blob	0.01	103	0.47	1.82	3.17	16.4
Botanic	0.01	73	0.1	0.45	0.87	2.48
CreaseShell	0.01	110	0.03	0.15	0.24	0.57
Gopher	0.01	170	0.21	0.85	1.62	5.09
Neumunster	0.01	156	0.1	0.46	0.92	2.53
QuadFlat	0.01	74	0.05	0.29	0.49	1.13
Ruled	0.01	92	0.02	0.05	0.04	0.09
Wave	0.01	159	0.11	0.46	0.91	2.52
BoonieBraced	0.0005	857	0.02	0.11	0.33	0.62
BoonieChaotic	0.0005	763	0.02	0.12	0.36	0.68
BubbleShell	0.0005	599	0.18	0.78	2.45	6.09
Hall	0.0005	463	0.04	0.22	0.39	0.92
HallCrease	0.0005	738	0.05	0.26	0.48	1.25
Envelope	0.0005	462	0.128	0.52	1.02	3.02
Station	0.0005	612	0.26	0.96	1.81	7.58
Tent	0.0005	952	0.25	0.94	1.83	6.46
Yarn	0.0005	445	0.13	0.51	1.03	3.26

Table 2 reports the execution times. Our method provides solutions in a reasonable time. Moreover, having an implementation that works with GPUs brings a considerable advantage over CPUs.

3.2. Evaluation

We assess our results visually to check if the main features, in particular surface smoothness, are preserved. We report the displacement color map to highlight the areas of shape alterations. Additionally, we report load deformation maps as a practical quantitative measure of the stiffness. Together with color maps, we also provide the maximum Euclidean norm of displacement vectors (δ_{max}) and the maximum Euclidean norm of load deformation vectors (u_{max}).

Fig. 6 reports a first illustrative example on model Blob, a large-scale non-membrane grid shell with a hole and large areas of low/null curvature. The optimized results reduce, on average, the strain energy by 91.4%. Compared with the starting condition, the shape slightly changes and acquires smooth curvature in the flat areas. Also, relevant features, such as the hole position and shape, are well preserved during the displacement. Smoothness is also maintained, as shown by the continuous reflections of the panels in the rendered view. Even though a maximum displacement of $\delta_{max} = 1.287\ m$ occurs, the shapes are perceived to be similar, and the stiffness is significantly increased.

While Blob is continuously supported along the base boundary curve, the case Envelope (Fig. 7) has long free boundaries. In particular, the freeform shape and large openings make it inherently inefficient if used as a single-layer structure because the bending behavior becomes predominant over the membrane behavior. Large deformation and strain energy accumulate close to the front profile, which is shaped as a multi-centered arch. For a change of 4.315 m, the structure earns relevant stiffness with the deformations reducing from 9.27 m to 0.128 m. For such a large-scale example, the shape is raised in the area of the front profile, however the plan outline is preserved. Even though, remarkably, the strain energy drop is about 98%, it must be pointed out that the adopted linear solver for the strain energy computation can be less accurate in case of large deformations, therefore the strain energy reduction and deformations can be overestimated.

The case Station in Fig. 8 is a shallow freeform shape and suffers from large deformations in the starting configuration. Moreover, it relies on very few supports, making it structurally inefficient. Like the previous examples, it is meshed with an isotropic triangulation that distributes the structural elements randomly. In symmetric shapes, this strategy might produce, as in the present case, an asymmetrical displacement plot. The impressive loss reduction of 96% comes at the cost of an increased height of the structure, for a maximum displacement $\delta_{max} = 12.691$ m.

Finally, the two case studies in Fig. 9 show a highly variable distribution of internal forces (positive/negative axial and bending forces). Moreover, the case BoonieChaotic exhibits an uneven triangulation, and the case BubbleShell has multiple boundaries with different geometry.

The learned, optimal shapes show high geometric fidelity for all cases considered. As expected, the more inefficient the shape, the higher the displacement. However, when the structure can rely on enough fixed boundary nodes, the method returns the finest result as the shape gets stiffer with features-preserving displacement. Most importantly, all optimized results achieve a high structural efficiency. The beams utilization is more uniform than the starting condition, and the material is more properly employed.

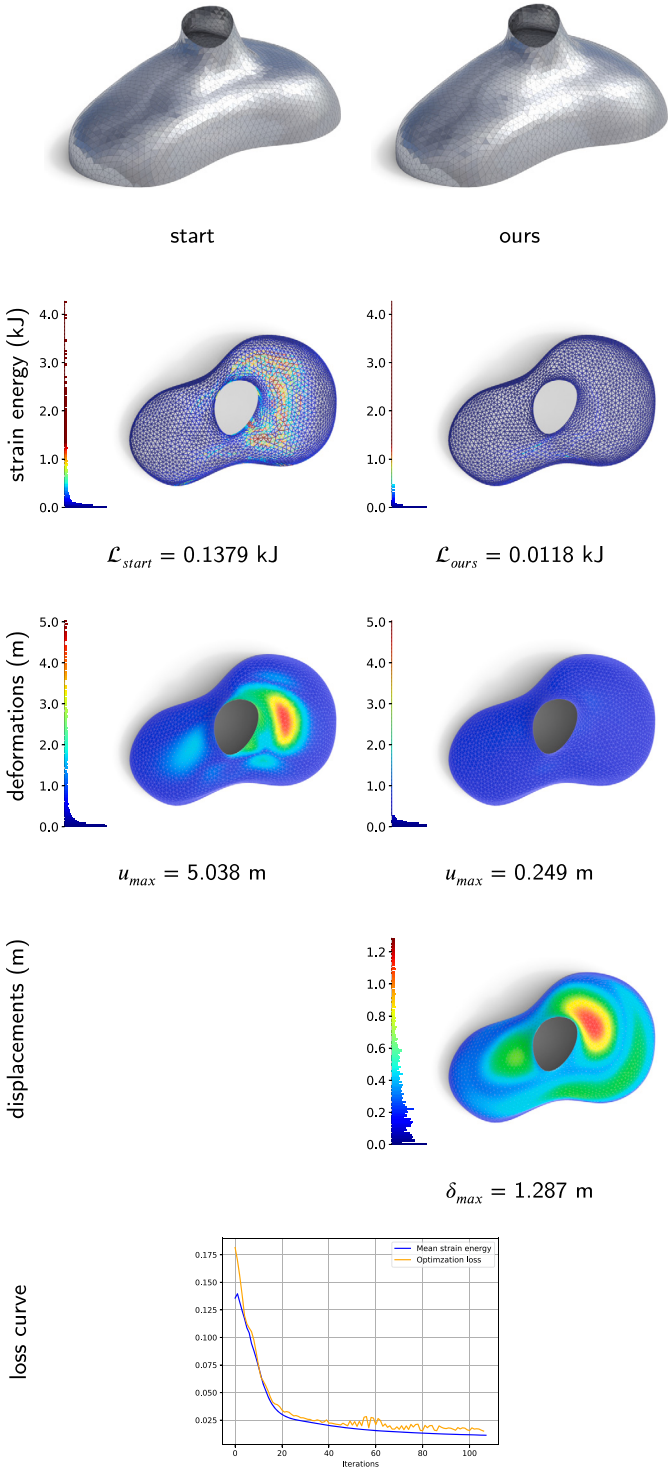

Fig. 6. Results on model Blob. Left: the input shape; Right: the displaced model. From top to bottom, the edge strain energy, the deformations, the displacements, and the loss curve.

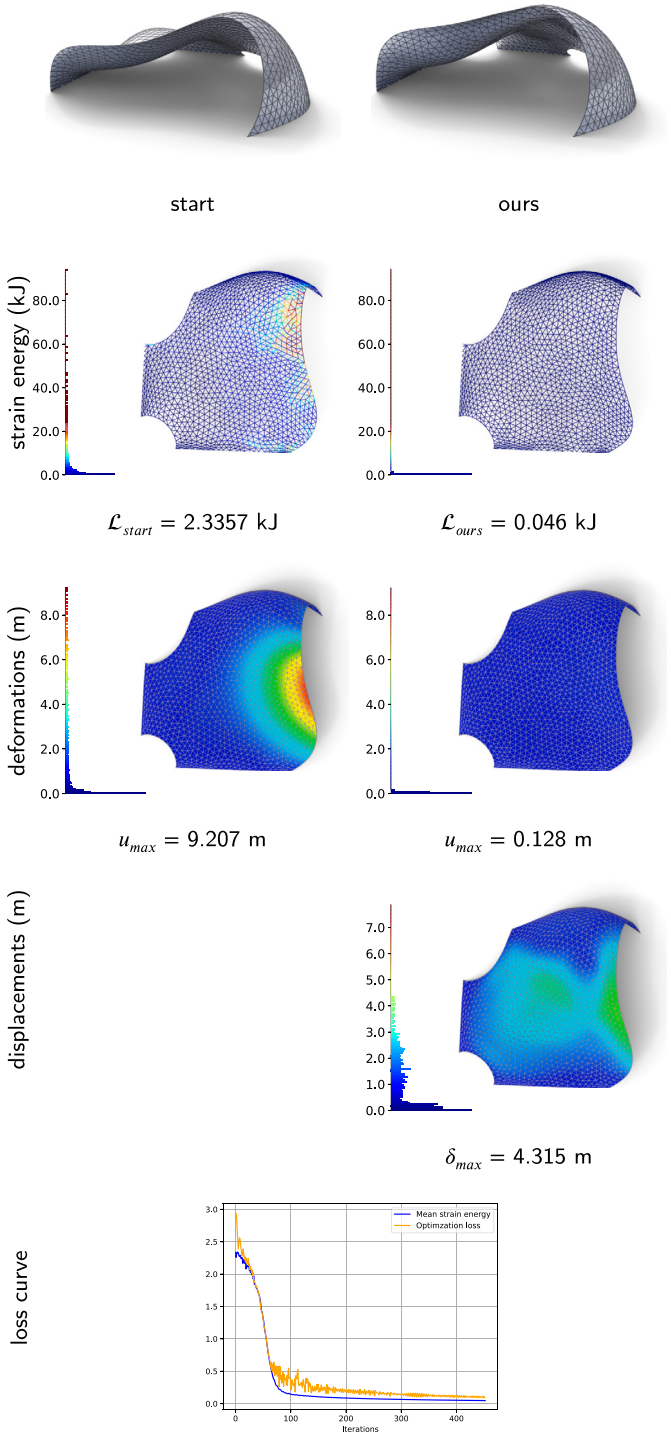

Fig. 7. Results on model Envelope. Left: the input shape; Right: the displaced model. From top to bottom, the edge strain energy, the deformations, the displacements, and the loss curve.

3.3. Comparison with form finding results

We validate the effectiveness of our method against two computational form finding tools, namely Kangaroo [33,34] and Karamba [35]. Both tools are implemented as plug-ins in Grasshopper [36], and are commonly employed in the shaping and optimization of grid shells.

Kangaroo (kan in the following) is a multi-physics solver that uses the Dynamic Relaxation Method (DRM) to simulate the equilibrium of hanging models for given boundary constraints and loads. The DRM integrates the dynamics equilibrium equations tracking the deflections, and damps the motion of the structure artificially until it reaches the static equilibrium. During the simulation, the solver minimizes the bending energy of the structure while keeping the edge length changes small.

In Kangaroo, the user is allowed to modify several parameters to reach the desired solution. The most important are the edge strength and its length factor, the magnitude of external load and the convergence tolerance. For a fair comparison, we generate a solution in

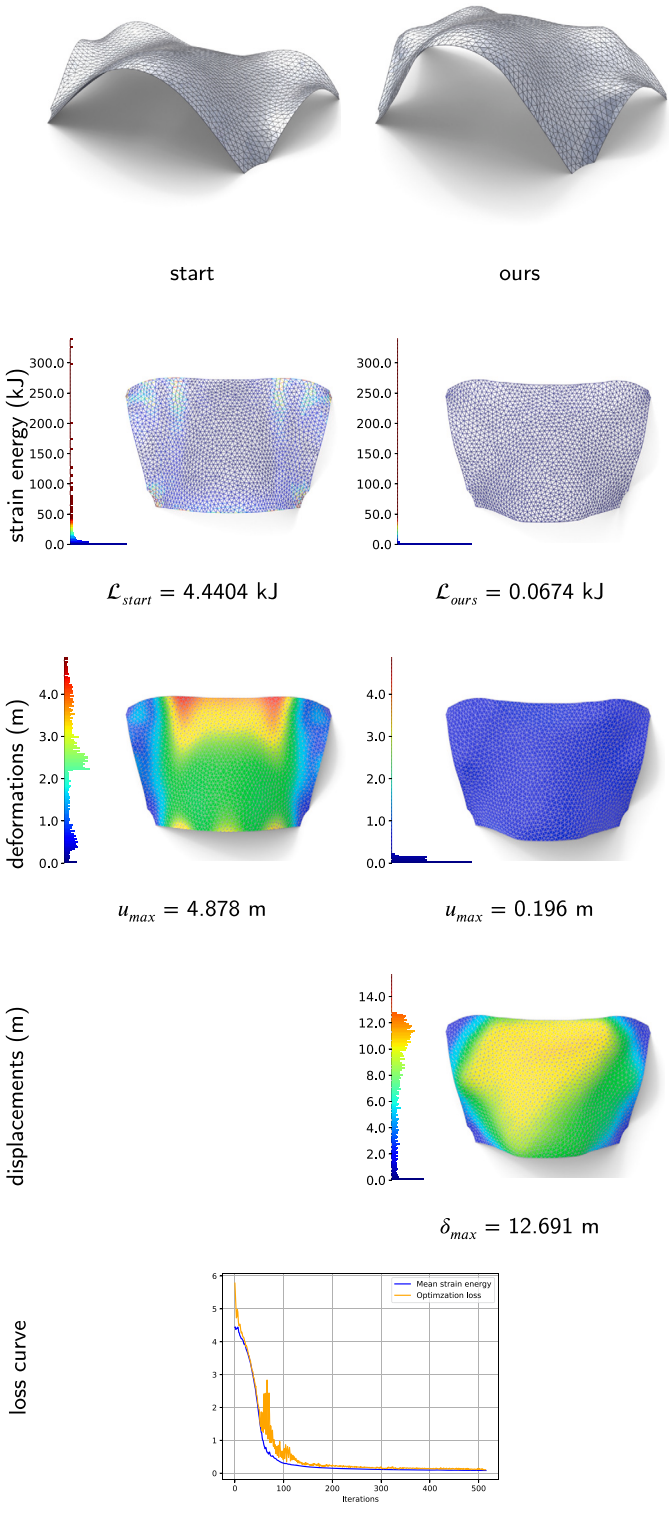

Fig. 8. Results on model Station. Left: the input shape; Right: the displaced model. From top to bottom, the edge strain energy, the deformations, the displacements, and the loss curve.

Kangaroo to have similar strain energy with respect to our model. The energy is computed with our 2-dof solver. Indeed, since the methods use different element formulations and solvers, the shape would be too distant if obtained for the same load. The optimal form-found shape has indeterminacy in the vertical direction. Therefore, we tentatively scale the load to match the criterion and act on tolerance (indirectly on the number of iterations) to reach convergence.

Table 3
Results: model name; initial mean strain energy on edges $\mathcal{L}_{start}$; mean strain energy on edges of the output of our method $\mathcal{L}_{ours}$; an energy configuration $\mathcal{L}_{kar(E)}$ similar to $\mathcal{L}_{ours}$ belonging to an output of the form finding tool Karamba; mean strain energy on edges $\mathcal{L}_{kar(\delta)}$ of a Karamba output with the same displacement maximum norm; mean strain energy on edges $\mathcal{L}_{kan}$ of the output of the form finding tool Kangaroo; list of figures featuring the example.

name	mean strain energies					Figs.
	$\mathcal{L}_{start}$ (kJ)	$\mathcal{L}_{ours}$ (kJ)	$\mathcal{L}_{kar(E)}$ (kJ)	$\mathcal{L}_{kar(\delta)}$ (kJ)	$\mathcal{L}_{kan}$ (kJ)	
2Spheres	0.0177	0.0079	0.0078	0.0071	0.0061	14, A.1
Blob	0.1379	0.0118	0.0121	0.0084	0.0117	6, A.2
Botanic	0.0021	0.0018	0.0018	0.0017	0.0018	A.3
CreaseShell	1.5337	0.0248	0.035	0.0295	0.0304	A.4
Gopher	0.0037	0.0012	0.0012	0.0017	0.0009	13, 16, A.5
Neumunster	0.0008	0.0004	0.0003	0.0003	0.0003	A.6
QuadFlat	0.404	0.0018	0.0019	0.0014	0.0018	10
Ruled	0.1112	0.0132	0.0245	0.0262	0.0355	A.7
Wave	0.1704	0.0253	0.0240	0.1786	0.0156	12, 14, 16
BoonieBraced	0.4651	0.0427	0.0393	0.0495	0.0587	A.8
BoonieChaotic	0.4783	0.0531	0.0500	0.1144	0.0478	9, A.9
BubbleShell	0.8108	0.1208	0.0935	0.2337	0.1129	9, A.10
Hall	0.2199	0.0751	0.0746	0.142	0.0035	A.11
HallCrease	0.2395	0.0375	0.035	0.103	0.0019	11
Envelope	2.3357	0.046	0.045	0.0631	0.0088	7, A.12
Station	4.4404	0.0674	0.0589	0.1603	0.031	2, 4, 8, A.13
Tent	1.9381	0.1261	0.1265	0.0416	0.0226	1, A.14
Yarn	3.3016	0.0558	0.0551	0.437	0.0608	A.15

Karamba is a finite element software that includes several algorithms for linear and nonlinear simulations, among which the 'Analyze Large Deformation'-component handles the behavior of hanging models through an incremental solver that considers geometric nonlinearity. The loads are applied in small steps for a better approximation. Similarly to kan, this tool employs an accurate beam model, in which we plug the specific cross sections and material. The solution is load-controlled and depends on the increment parameter, namely the number of load steps, which is also the rate of model updating. Being non-adaptive, this strategy usually leads to convergence issues. However, the most interesting feature is the possibility of constraining the maximum displacement at a user-defined value. To compare the performances, we generate two solutions for Karamba to have similar strain energy (kar(E)) and similar maximum displacement (kar(δ)) with respect to our model. Table 3 reports quantitative data: the value of the mean strain energy for the original model, and the values of the final energy of the displaced structure for our method and the competitors. Below is a qualitative, visual evaluation.

A first benchmark example is the form finding of a flat mesh built on a rectangular perimeter (Fig. 10), which is expected to return the classic pillow shape. The mesh is obtained by diagonalizing a quad mesh with uniform equal faces, so it turns into an anisotropic grid, as shown by the displacement and deformation maps. In kan, the model is akin to a perfect hanging net with hinge joints, and it outperforms the form finding task because it can produce membrane solutions even for small target height, i.e. obtaining very shallow shells. Our model instead uses a beam formulation with fixed boundary nodes. Thus, the shape is affected by a more stiff boundary condition. The starting tangent at the boundaries is horizontal, and the height is larger than other solutions. For this reason, our model is less stiff for a similar strain energy. A similar conclusion holds for the comparison with kar(δ). However, although the starting shape presents no relevant geometric features, our method is capable of delivering a good, smooth solution where the displacements are consistent.

HallCrease is a grid shell with a crease line, i.e. a chain of edges that break the continuity of the surface realizing a corrugation (Fig. 11). The large front opening results in two consecutive arches with unsupported spring point in between. Our model provides the smaller shape alteration by displacing the central part of the shape towards

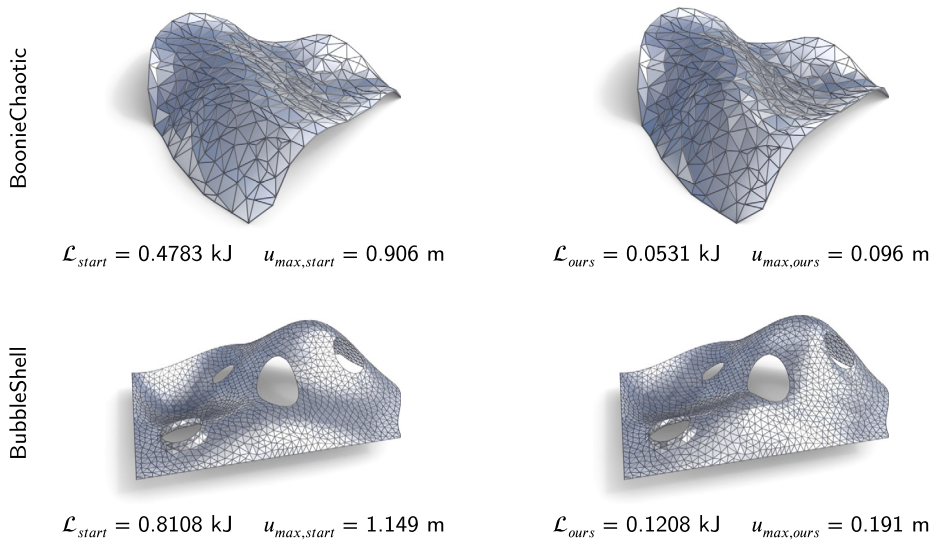

Fig. 9. Results on models BoonieChaotic (top) and BubbleShell (bottom). Left: the input shape; Right: the displaced model. Metrics assess the validity of our method. In both cases, the Service Load deformation (from $u_{max,start}$ to $u_{max,ours}$) is reduced, and the maximum vertex displacement norm ($\delta_{max} = 2.289$ m for BoonieChaotic, $\delta_{max} = 3.976$ m for BubbleShell) is broadly contained into the bounding box dimensions.

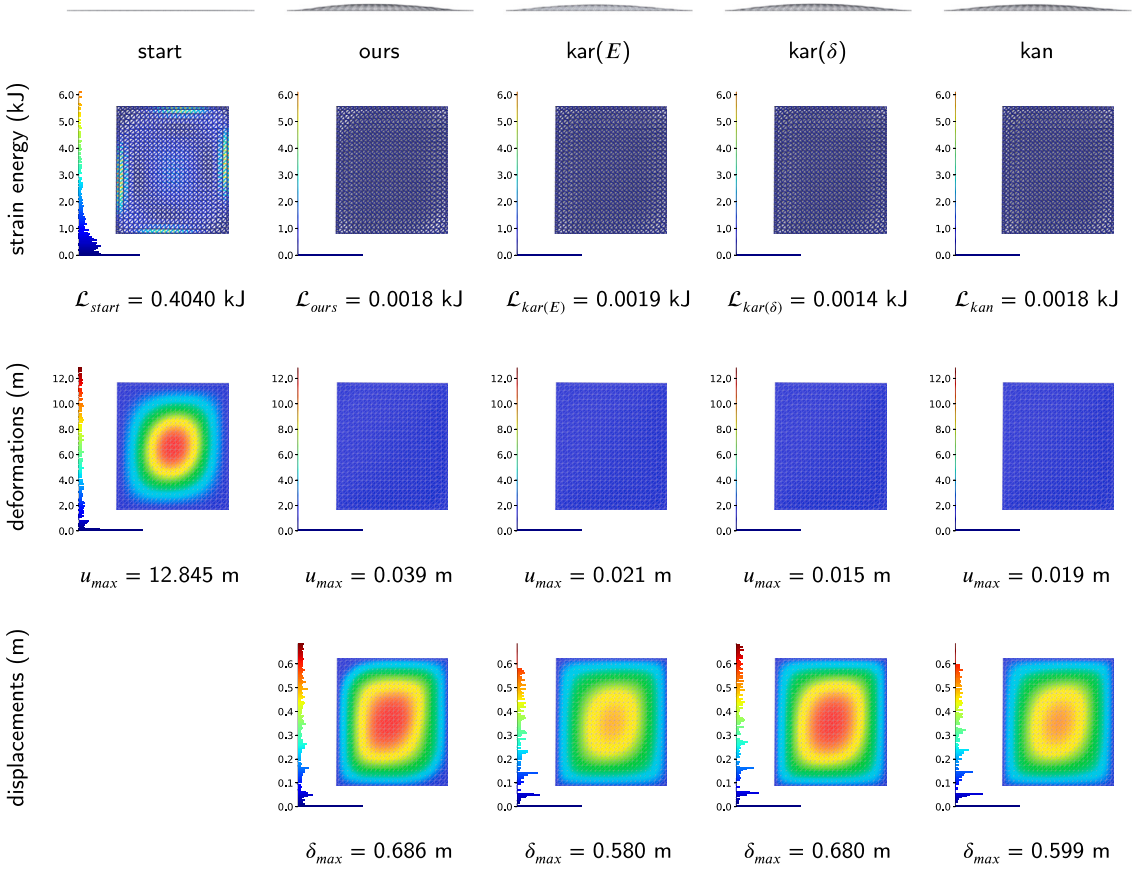

Fig. 10. Comparison of results from our model and form finding tools for similar strain energy (kar(E) and kan) and similar displacement (kar(δ)). Model: Quad.

an increased doubly curvature. The position of the crease line and the opening shape are kept in the original position. On the other hand, other solvers strongly alter the shape: in kar(E) the corrugation becomes more sloped and the front opening changed in both plan shape and height, up to $\delta_{max} = 4.905\ m$; in kan, the shape, although more efficient, is entirely changed into a relaxed hanging chain with a maximum displacement of $\delta_{max} = 8.580\ m$, and no affinity with the starting shape. In this latter case, the mesh is smoothed until the crease line is lost.

The Wave model (Fig. 12) is a freeform shape developing membrane behavior in the central part, with cantilevering boundaries and large openings, where the highest deformations occur. Our model achieves an impressive loss reduction of 85% and high stiffness with the low-

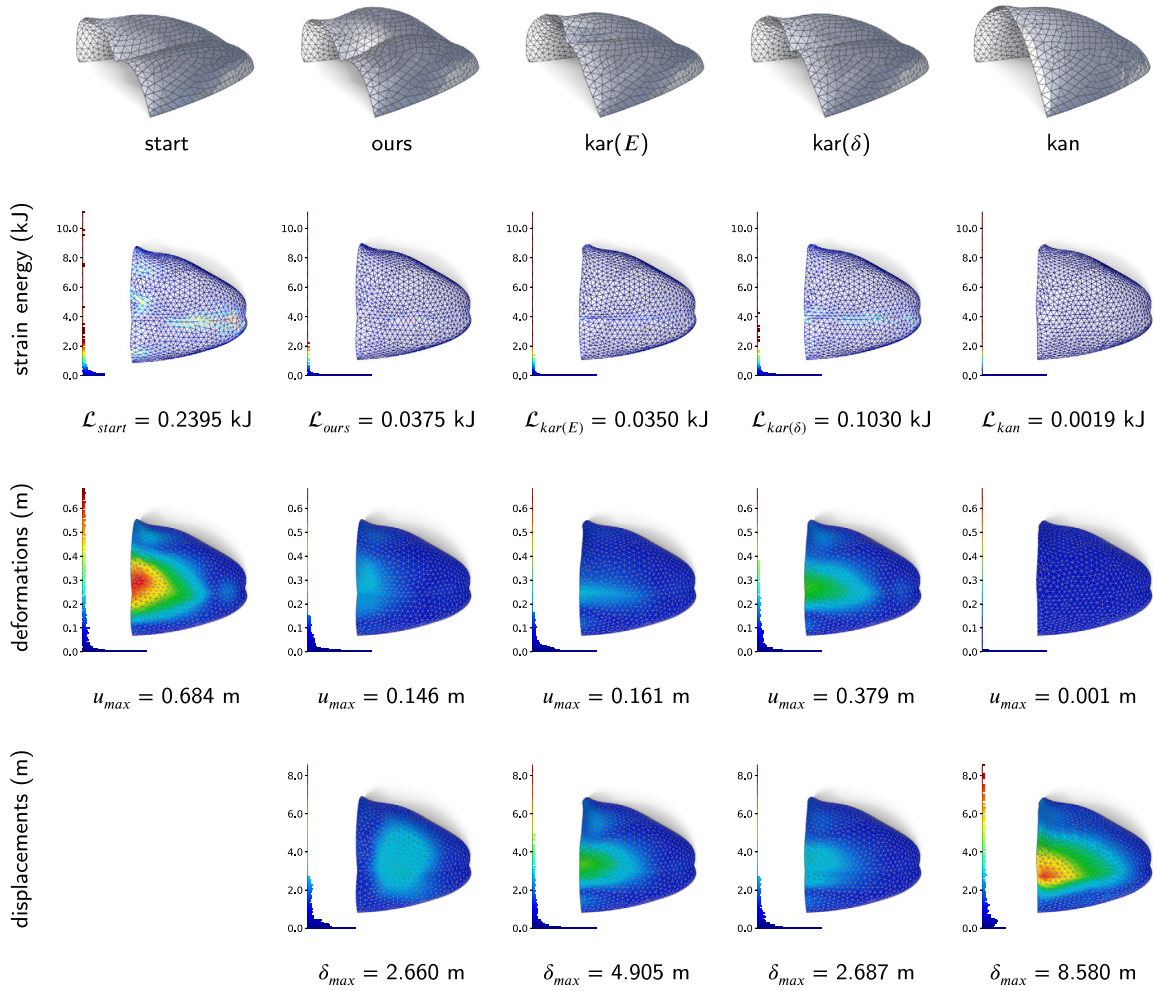

Fig. 11. Comparison of results from our model and form finding tools for similar strain energy (kar(E) and kan) and similar displacement (kar(δ)). Model: HallCrease.

est deformation $\delta_{max} = 0.918\ m$. The shape is slightly altered and the smoothness is preserved. The kan model presents several wrinkles as the solution of the hanging model stuck into a local minimum because the anchoring nodes are in small number. The kar(E) strongly changes the cantilevering profiles and the open boundaries up to $\delta_{max} = 2.199\ m$. In contrast, kar(δ) has a closer appearance to our solution, but it is more deformable ($u_{max} = 8.009\ m$, namely even more than the starting solution $u_{max} = 5.181\ m$). All form finding solutions tend to form negative Gaussian borders on the free boundaries to limit the inextensional deformation and increase the stiffness. Instead, our results well preserve the borders and the sharp features, which usually characterize the design of a grid shell from an architectural standpoint.

3.4. Ablation study

This Section proposes an ablation study to evaluate the single paper contributions. Section 3.4.1 evaluates the advantages of introducing feature encoding and convolutional layers over a plain optimization strategy not using deep learning. Section 3.4.2 discusses the relative importance of the geometric features we defined as input to the network. Finally, Section 3.4.3 evaluates the advantages of introducing the attention mechanism in the convolutional layers over a baseline without attention.

3.4.1. Advantages of 3D deep learning over optimization

One of the main advantages of reformulating shape optimization as a deep learning problem is that one switches from a vertex-centric approach, in which each single vertex is displaced without knowledge of what happens in its proximity, to a mesh-centric approach. In this paper, this is a consequence of the mechanism of neural message passing implemented by GAT layers, which enable the sharing of information among vertices in the same (geometric and semantic) neighborhood. The immediate effect is reducing geometric noise and the jagging effect, which can be observed with classical optimization strategies without resorting to geometric regularizers such as smoothing terms [14]. This is handy, as balancing energy terms and regularizers is often challenging, as they often have different and contrasting objectives.

In Fig. 13, we compare the results of our deep learning technique with the results of an optimizer implemented in PyTorch using the same loss on two different models (Gopher and a triangle flat structure supported at the vertices). The optimizer only adopts gradient descent to produce vertex displacements starting from the initial vertex positions and normals, without the feature encoding and the convolutional layers we introduced in this paper; the loss is the same for both procedures. It can be observed that the introduction of feature encoding and convolutional layers produce smoother results, the final energy and deformation being equal. Therefore, our method implicitly regularizes the displaced mesh, without the need for additional smoothing terms, which might divert from the optimal solution in terms of shape design.

3.4.2. Role of geometric features

An important aspect of the network we introduced is the use of differential quantities (curvature and geodesic distances) as features,

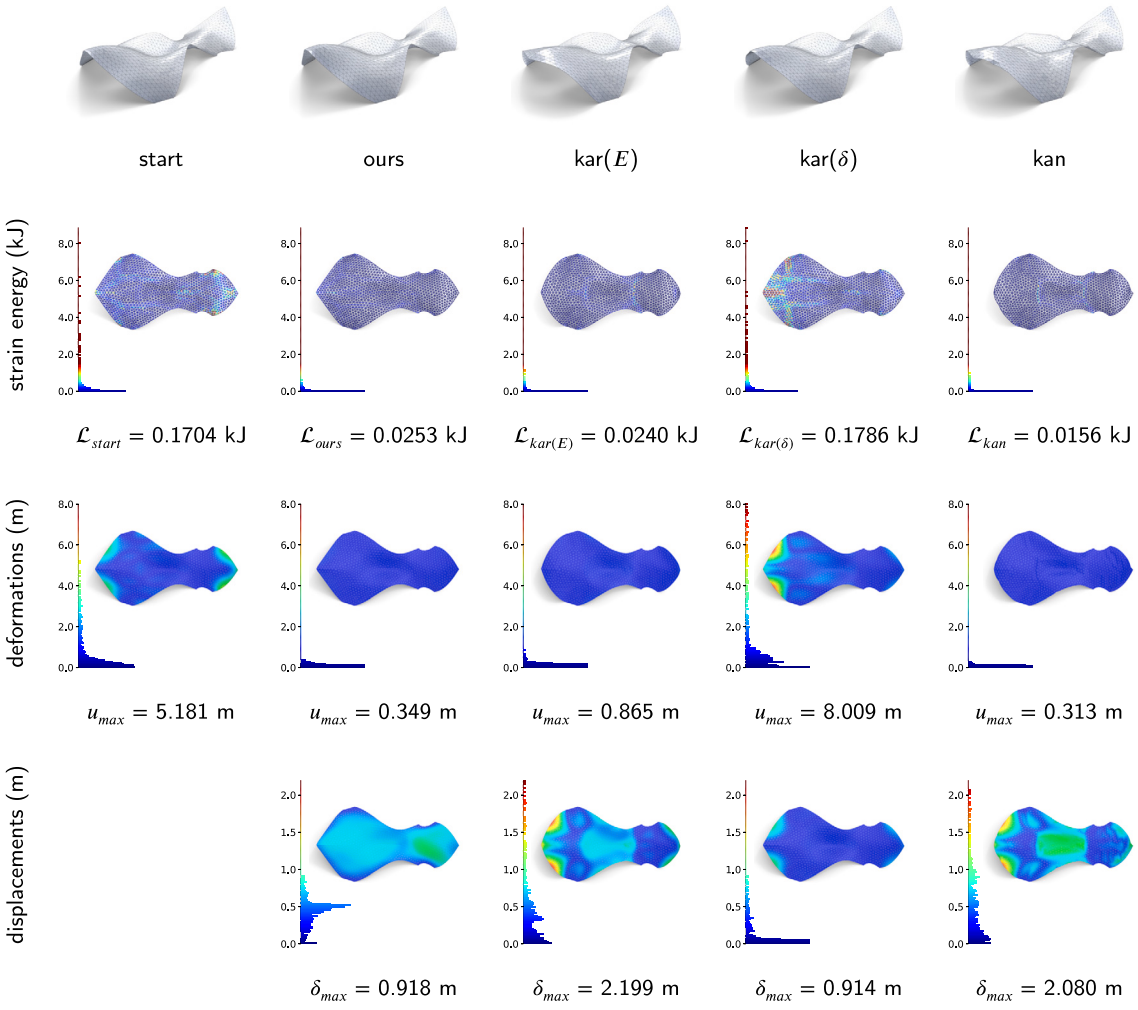

Fig. 12. Comparison of results from our model and form finding tools for similar strain energy (kar(E) and kan) and similar displacement (kar(δ)). Model: Wave.

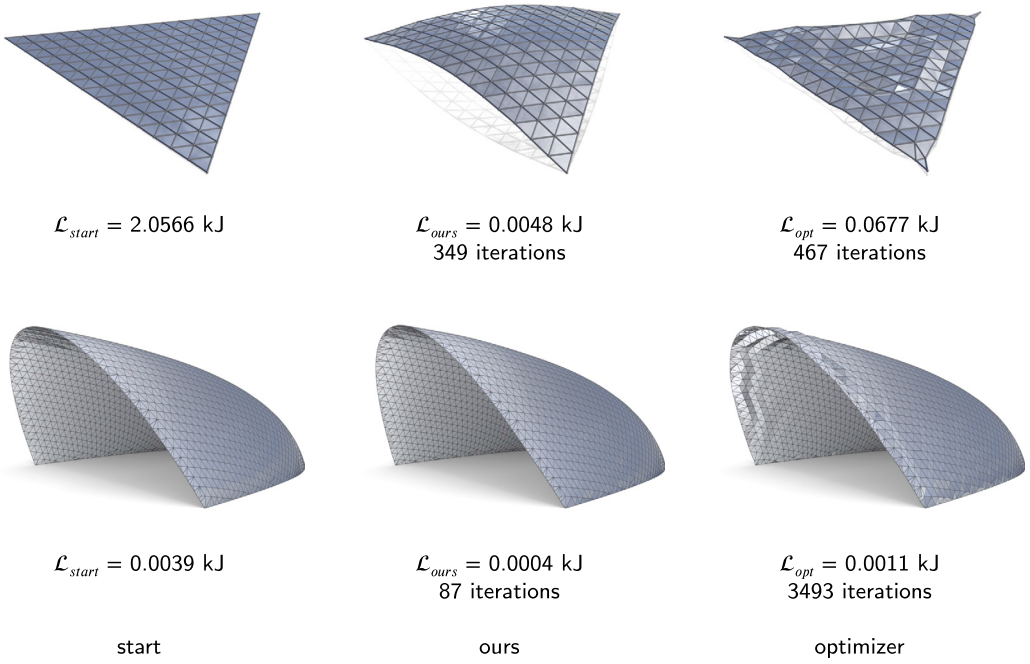

Fig. 13. Comparison between the results of our deep learning technique (middle) and a plain optimizer (right) on a flat triangle and on model Gopher (left). It can be observed that the message-passing mechanism among vertices during displacement learning has the effect of reducing geometric noise at the same time.

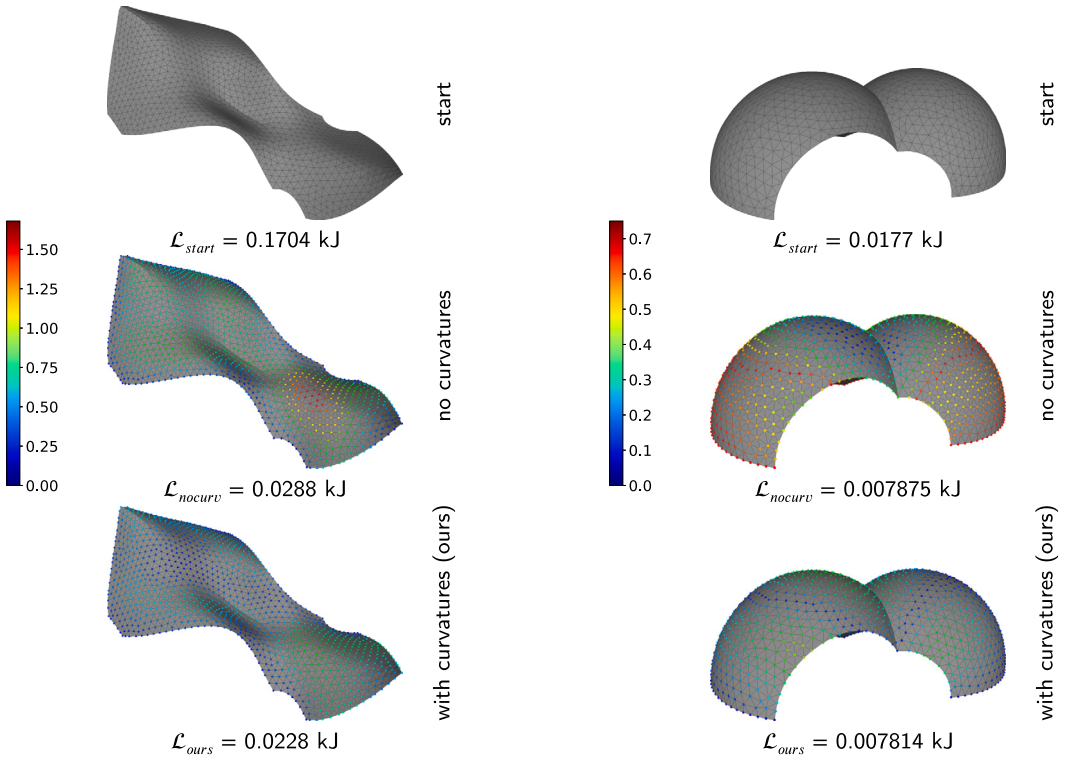

Fig. 14. Introducing principal curvatures as input features helps preserve the design intent, as it reduces the Hausdorff distance between the input mesh and the displaced mesh after learning. The effect on model Wave (left) and 2Spheres (right), with the distance values in false colors. Learning rate: 0.01, 200 SGD iterations.

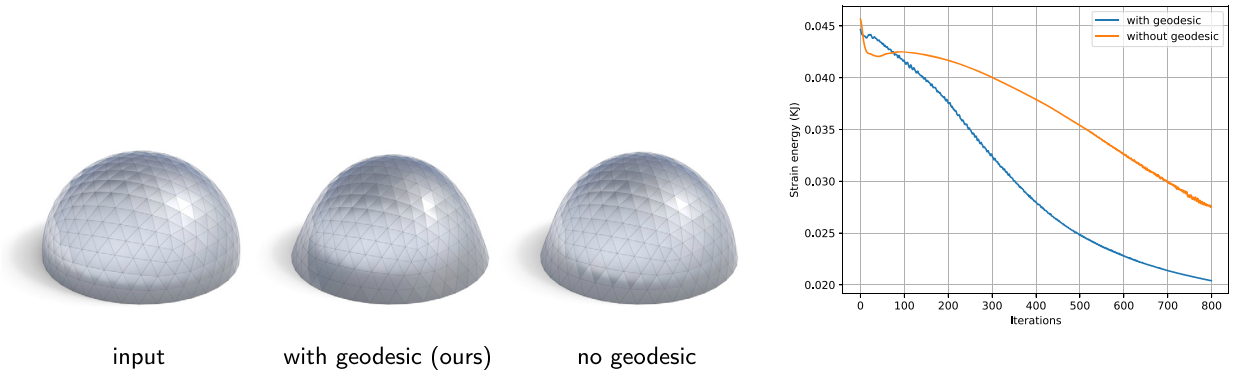

Fig. 15. Introducing features based on geodesic distances favors convergence towards sensible shapes and improves the loss reduction process. An input half-sphere model converges to a catenary dome if geodesic features are fed to the learning model, while full symmetry is not reached if geodesic features are omitted. The loss descent curve is also improved. Learning rate: 0.01, 800 SGD iterations.

alongside positional features traditionally employed in 3D deep learning. We propose an ablation study to discuss the effects of curvature and geodesic features, showing results with and without them.

We observe that using principal curvatures favors solutions with smaller vertex displacements for the same strain energy reduction, as shown in Fig. 14.

Concerning geodesic features, we observe they tend to favor convergence towards funicular shapes, which is sensible in the context of form finding, as shown in Fig. 15. The same Figure also shows that curvature features act as regularizers in terms of loss reduction.

3.4.3. *Effects of attention mechanisms*

Attention mechanisms in graph convolutional layers are expected to improve the expressing power of message passing among vertices, as the representations of neighboring vertices are aggregated using a weighted scheme, with learned weights.

Fig. 16 compares the results of GATv2 layers with respect to plain dynamic edge convolution (DGCNN, [19]). It can be observed that attention improves the fidelity to the original shape design for comparable static performance. The output from the DGCNN model results often in a more trivial pulled-up version of the original mesh, which, besides the increased surface area and weight, is characterized by more shape stiffness. Instead, adding attention leads to solutions with slighter shape modifications and less displacement in absolute terms.

4. Discussions and conclusions

This paper introduces a 3D deep learning pipeline for statics-aware grid shell design. We define a network that consumes a 3D mesh rep-

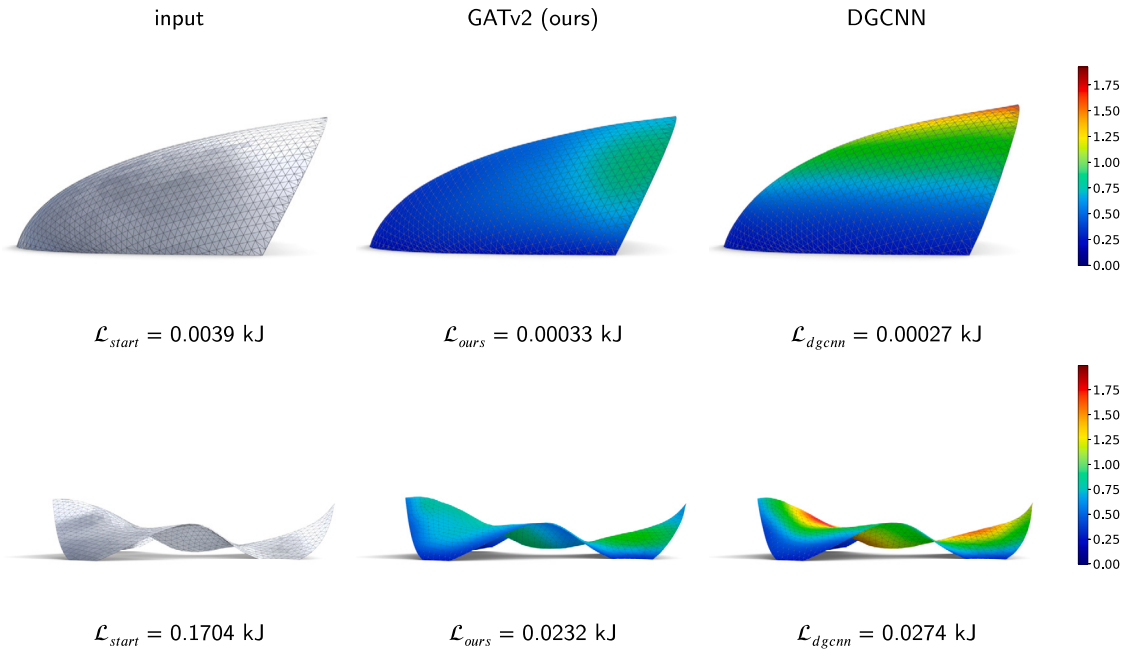

Fig. 16. The results with attention layers (middle) and with simple dynamic edge convolution (left). Color mappings refer to the vertex displacements of the model outputs from the input. Attention layers offer better shape preservation with similar static performance. Models: Gopher, learning rate 0.1, 200 SGD iterations; Wave, learning rate 0.01, 200 SGD iterations.

resenting an input design and learns optimal vertex displacements to performance while preserving the design intent. Our network can perform both shape optimization from first designs and form finding from flat input shapes. The results demonstrate that our method performs better than competitors in shape optimization, as it produces structures with optimized performance and preserved geometric features, such as curvature and boundaries. The results in the form finding task are comparable with existing techniques. A major advantage, however, is that our method only requires a parameter to set, namely, the learning rate; therefore, it reduces the burden on the user's side.

The learning rate has a fundamental role, as it controls how much the shape changes at each iteration. Setting properly the learning rate drives the convergence to an optimized shape, while controlling that the resulting geometry is smooth and close to the original shape. We show that good results can be obtained by simply setting the same learning rate for classes of objects that share the same beam cross section and scale. However, in the future, we plan to research adaptive methods to set the learning rate according to structural information.

Moreover, even though our results show that the original shape can be preserved by a carefully designed architecture, it would be interesting to introduce explicit constraints to control adherence to the input design, such as sophisticated regularizers based on topological descriptors that analyze the preservation of geometric features [37].

Finally, future work will include the statics-driven optimization of beam connectivity besides node positioning. Also, we plan to include material reduction as an optimization criterion to pursue sustainability besides statics performance. This extension could possibly be done by including the optimal shape of beam cross sections as an additional variable to learn to reduce steel weight.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data and source code are available on GitHub at https://github.com/cnr-isti-vclab/GeomDL4GridShell#geometric-deep-learning-for-statics-aware-grid-shells.

Acknowledgements

This work was supported by the NextGenerationEU programme under the funding schemes PNRR-PE-AI scheme (M4C2, investment 1.3, line on AI) FAIR "Future Artificial Intelligence Research", grant id PE00000013. This paper has received financial support by the Horizon Europe Research & Innovation Programme under Grant agreement N. 101092612 (Social and hUman ceNtered XR - SUN project). Views and opinions expressed in this paper are those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the European Commission can be held responsible for them.

Appendix A. Additional results

This Section reports the results for all 18 case studies of Table 1 (see Figs. A.1–A.15). The performance of the present method is expressed in terms of mean strain energy on the beams $\mathcal{L}_{ours}$, modification of the node coordinates (displacement), and achieved stiffness (load deformation) with respect to the input shape. Moreover, comparisons with other computational tools are included. Using Karamba, we target a similar mean strain energy on the beams (kar(E)) and a similar displacement maximum norm (kar(δ)). Using Kangaroo, we target a similar mean strain energy on edges (kan).

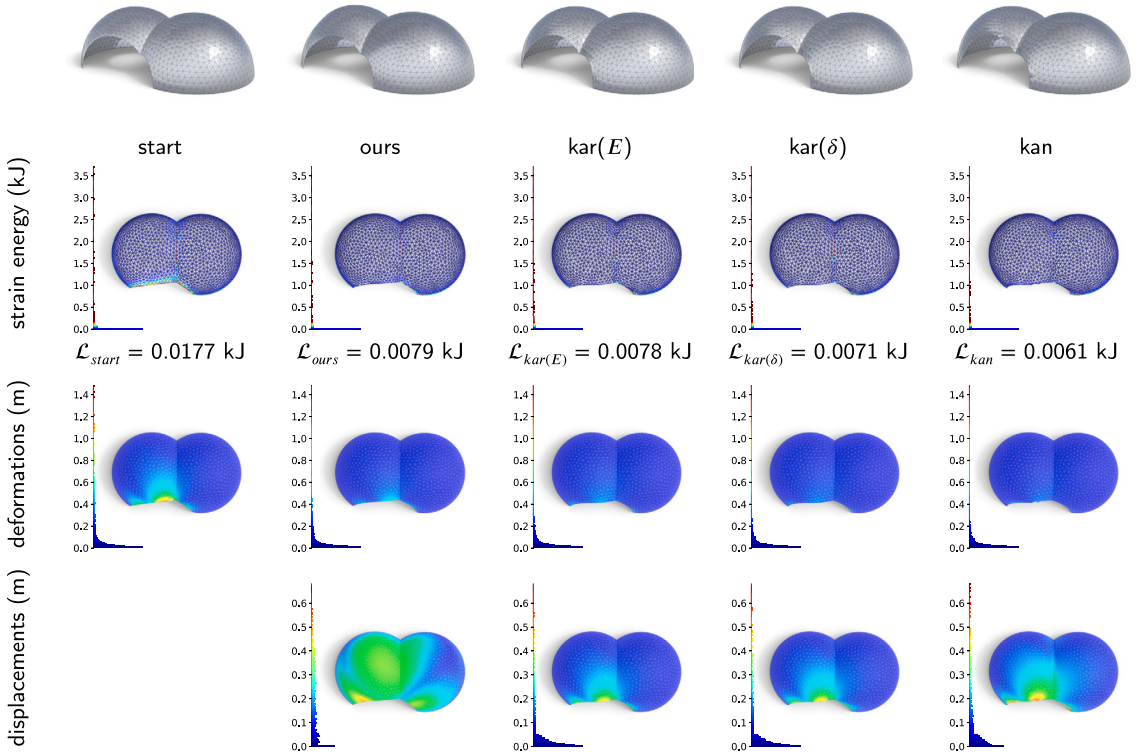

Fig. A.1. 2Spheres. Results and comparisons with other tools.

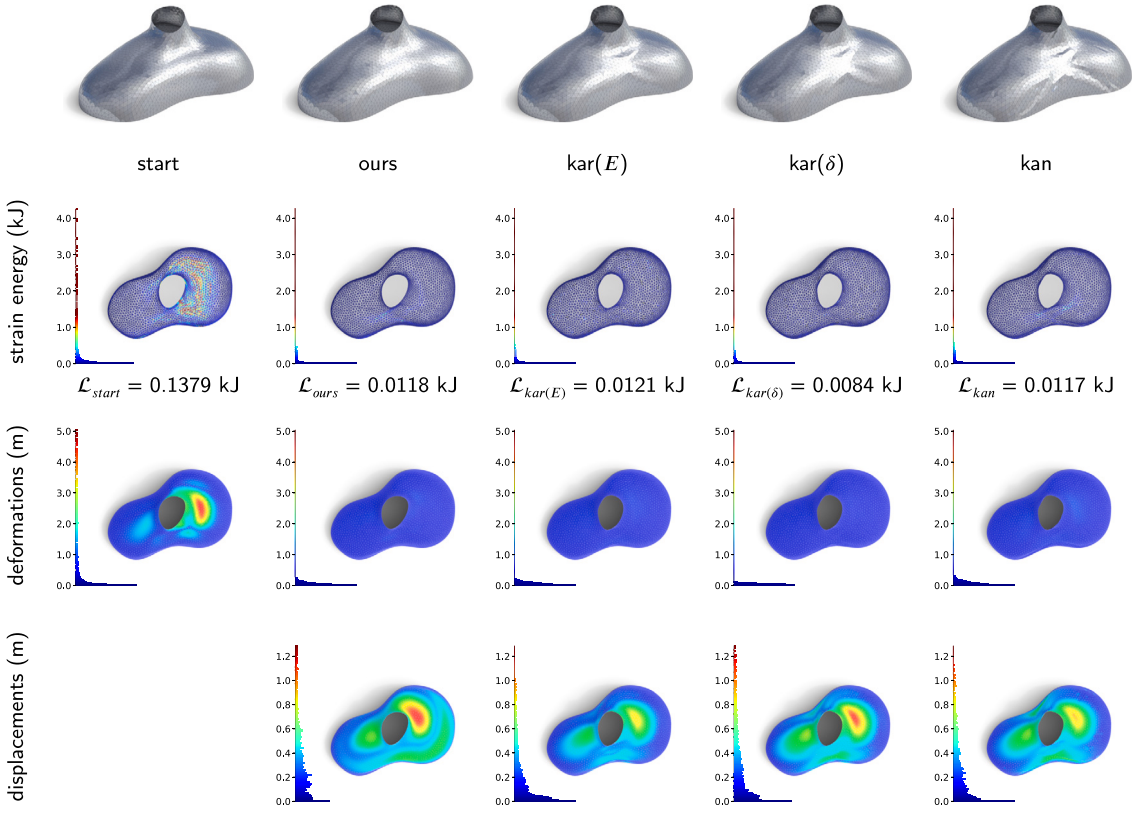

Fig. A.2. Blob. Results and comparisons with other tools.

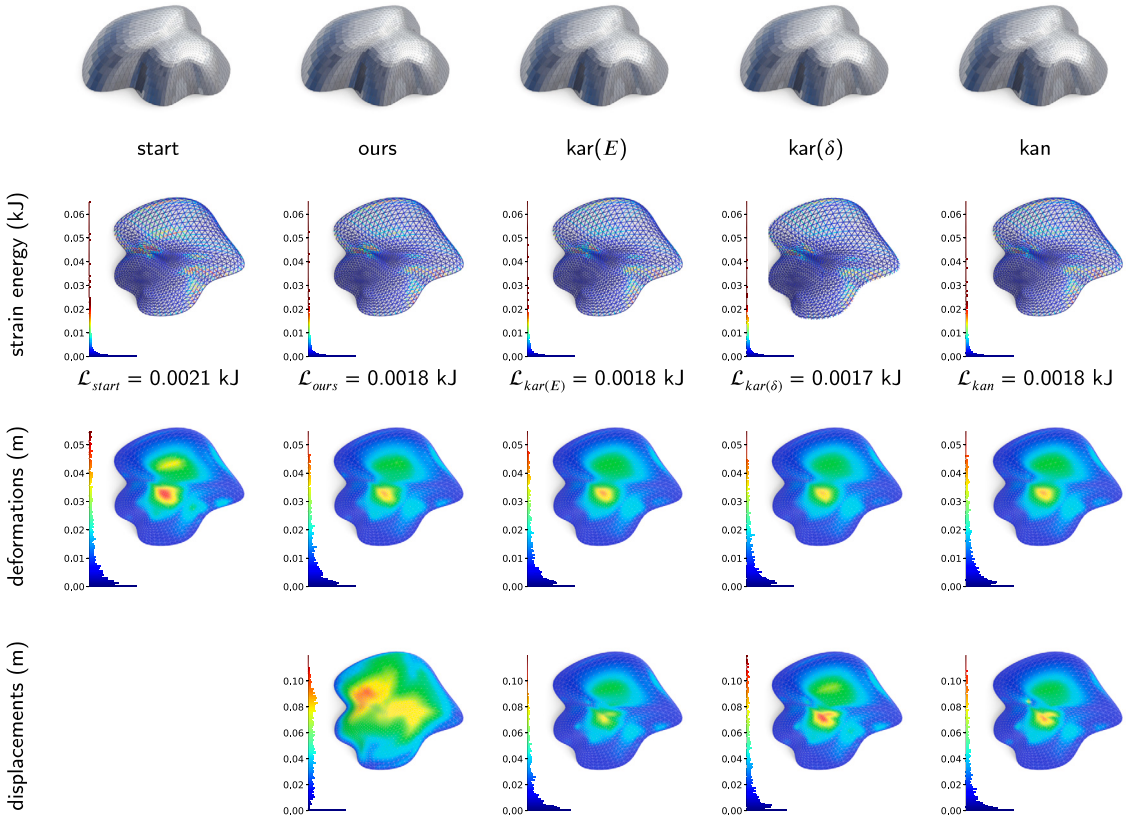

Fig. A.3. Botanic. Results and comparisons with other tools.

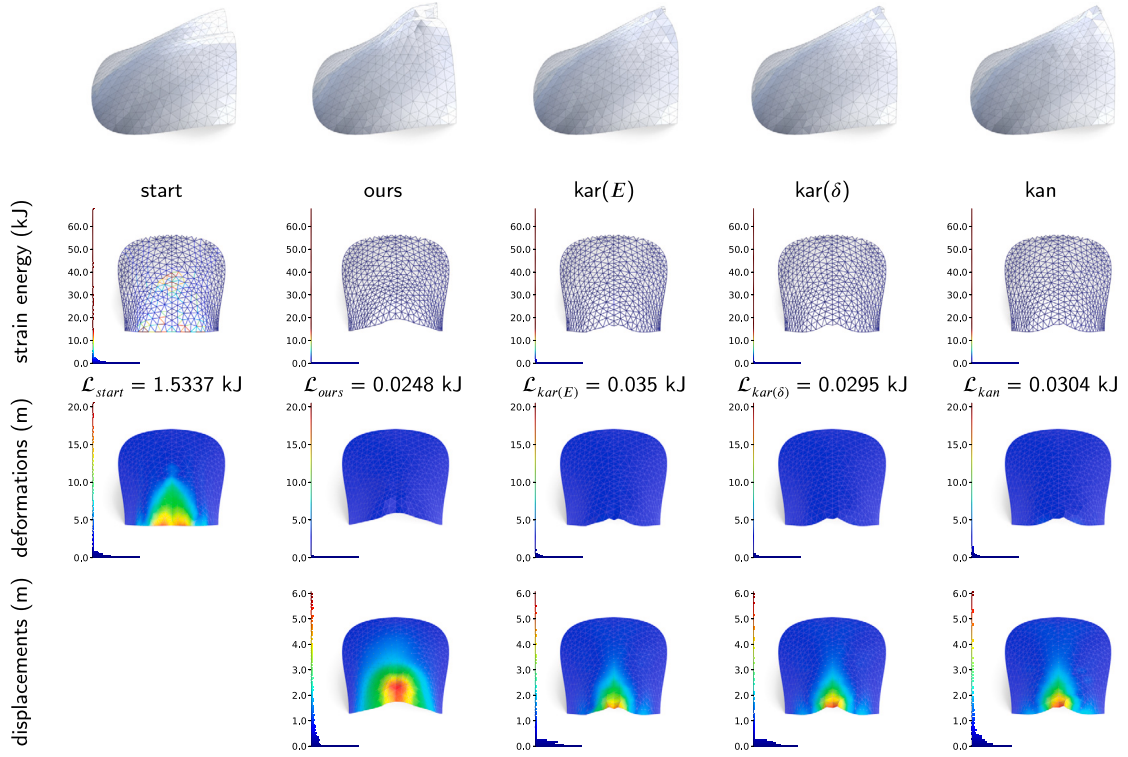

Fig. A.4. CreaseShell. Results and comparisons with other tools.

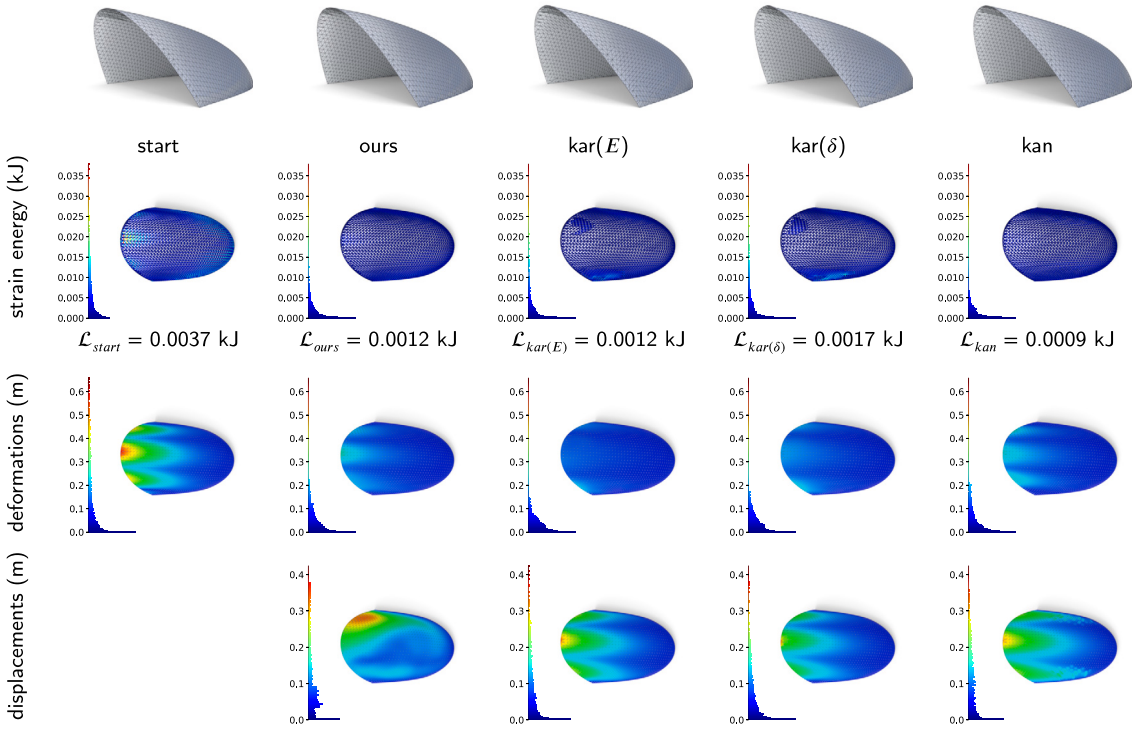

Fig. A.5. Gopher. Results and comparisons with other tools.

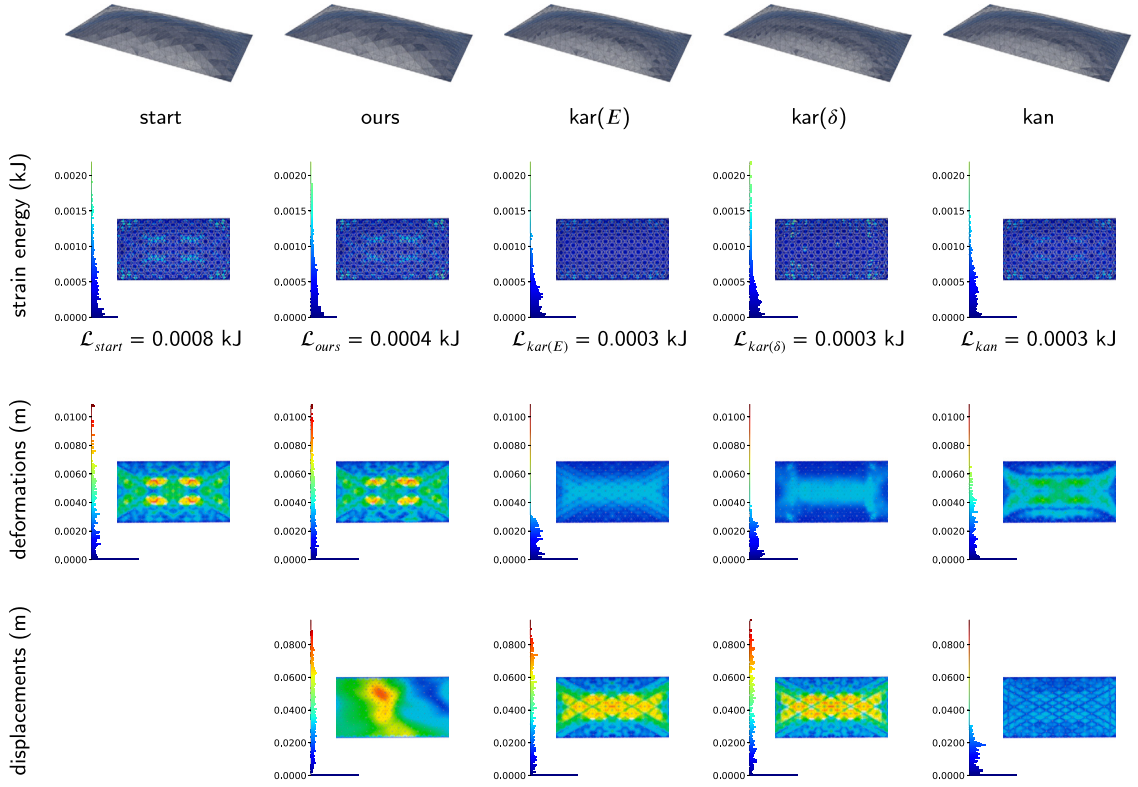

Fig. A.6. Neumunster. Results and comparisons with other tools.

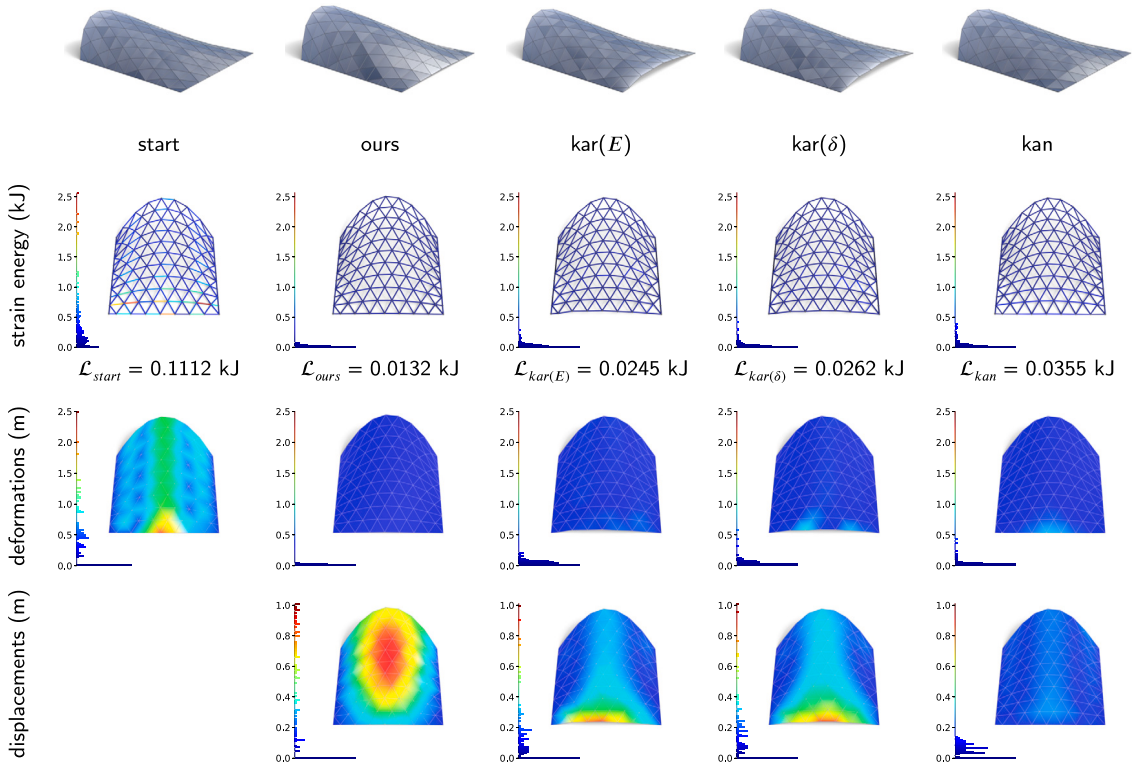

Fig. A.7. Ruled. Results and comparisons with other tools.

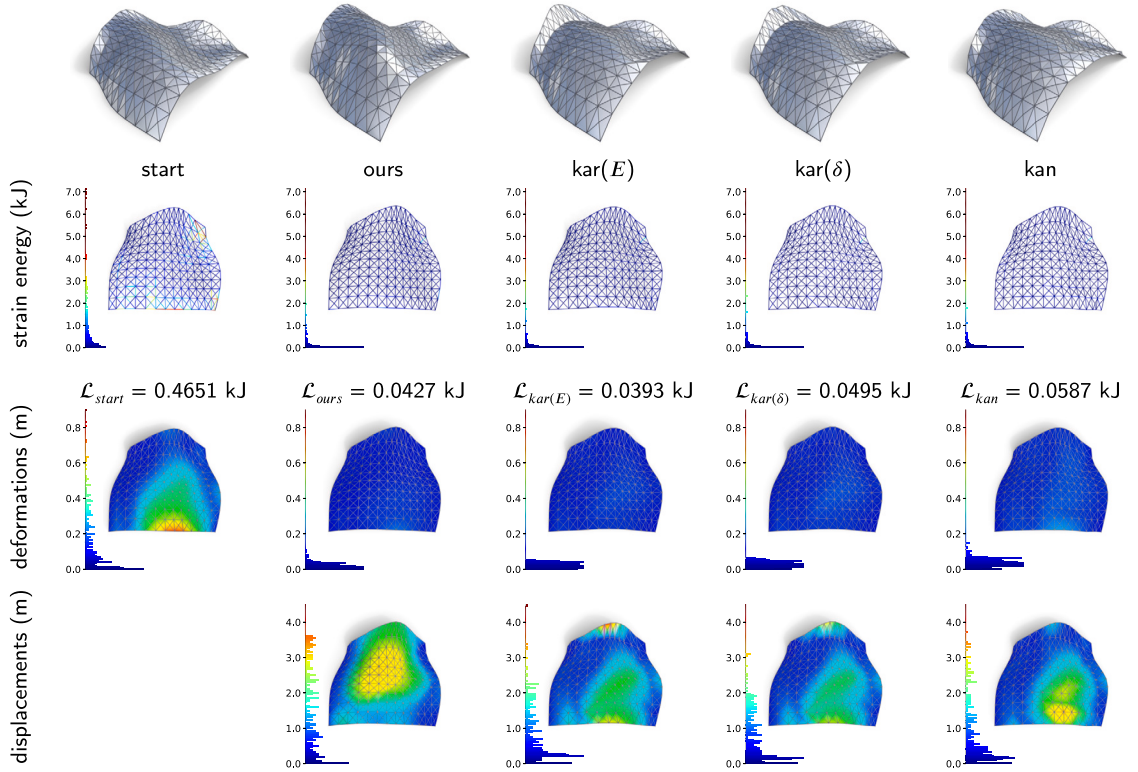

Fig. A.8. BoonieBraced. Results and comparisons with other tools.

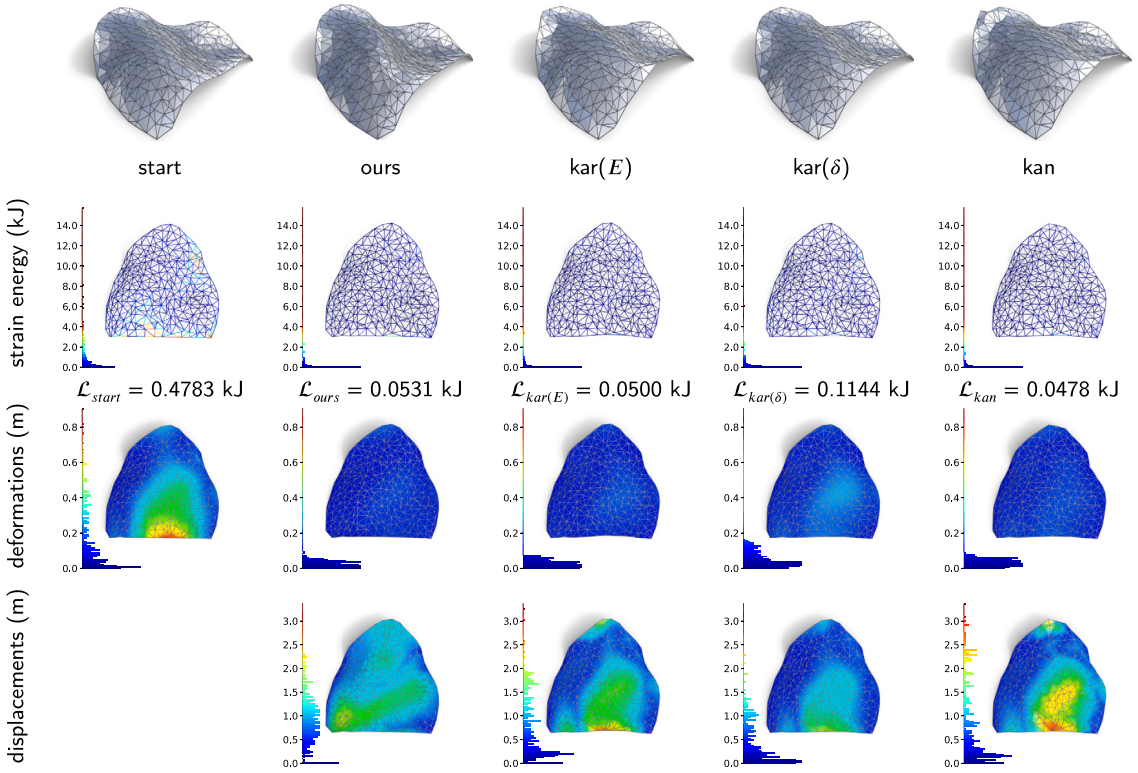

Fig. A.9. BoonieChaotic. Results and comparisons with other tools.

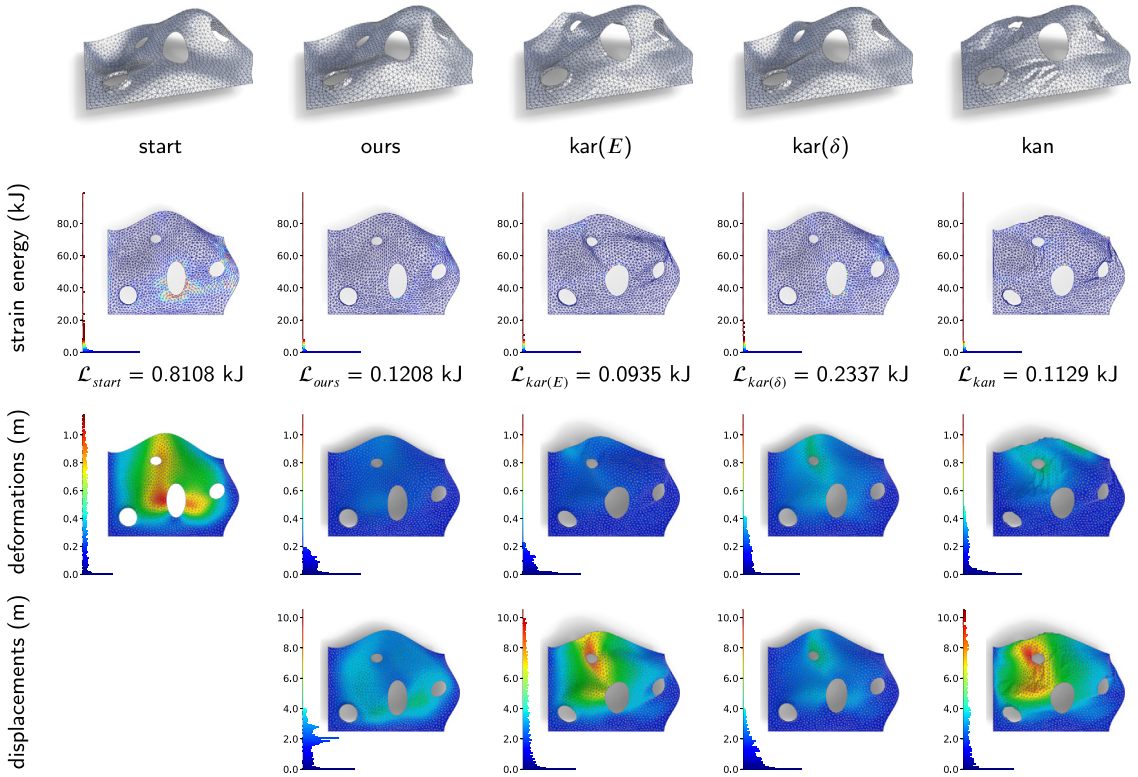

Fig. A.10. BubbleShell. Results and comparisons with other tools.

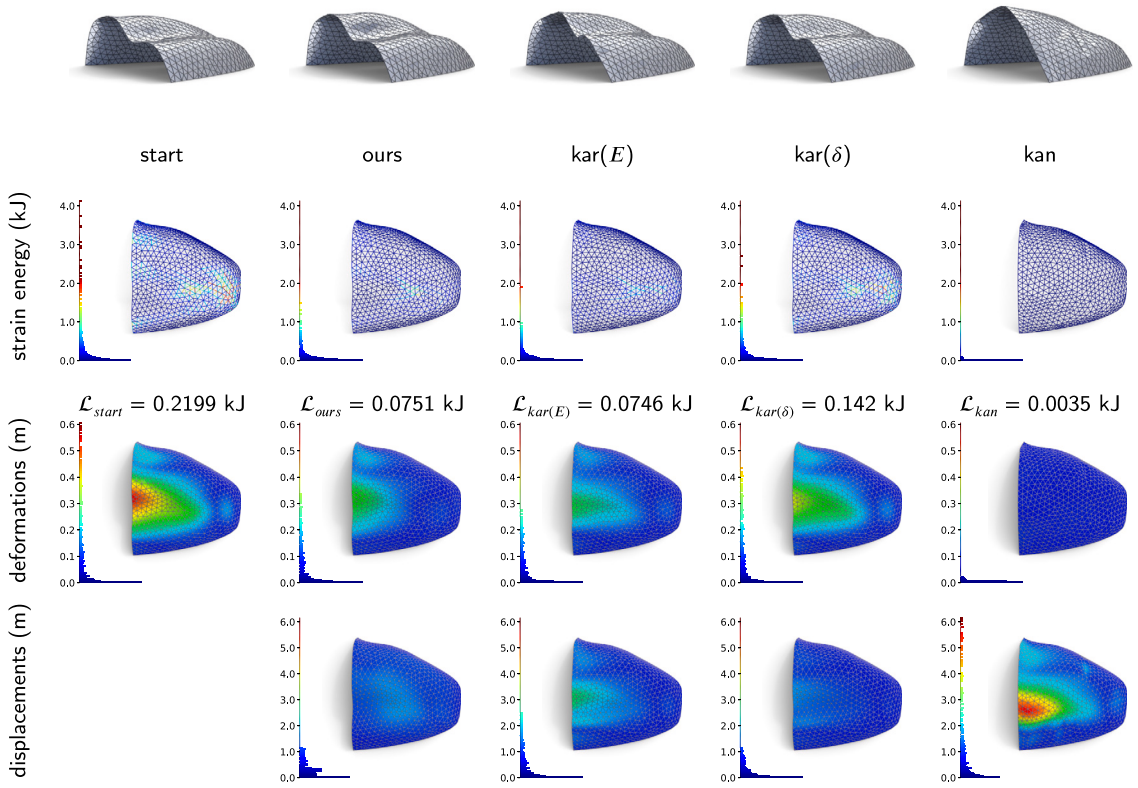

Fig. A.11. Hall. Results and comparisons with other tools.

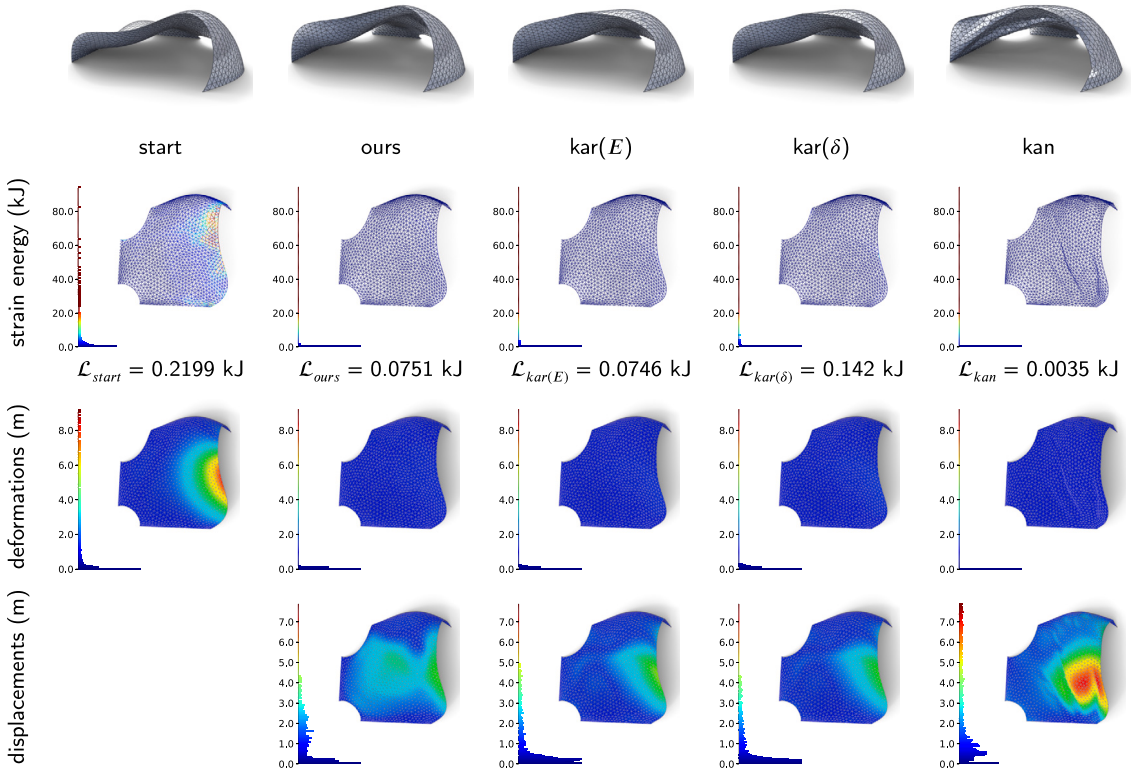

Fig. A.12. Envelope. Results and comparisons with other tools.

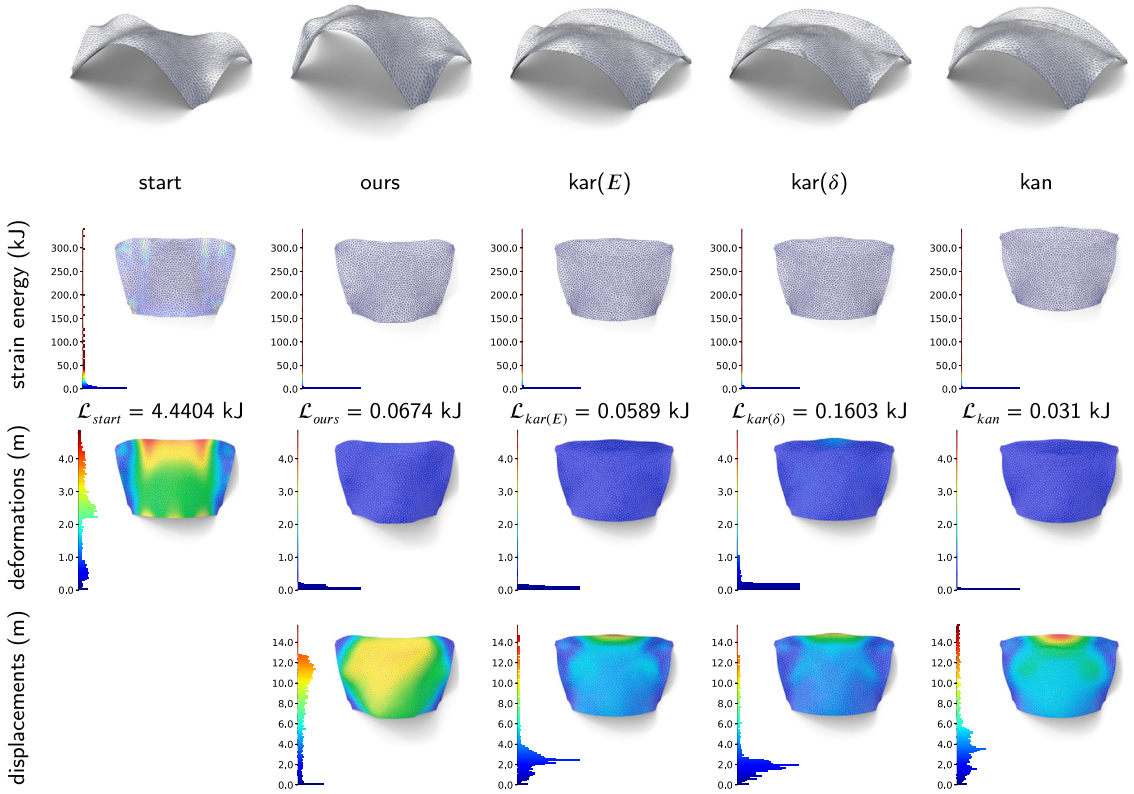

Fig. A.13. Station. Results and comparisons with other tools.

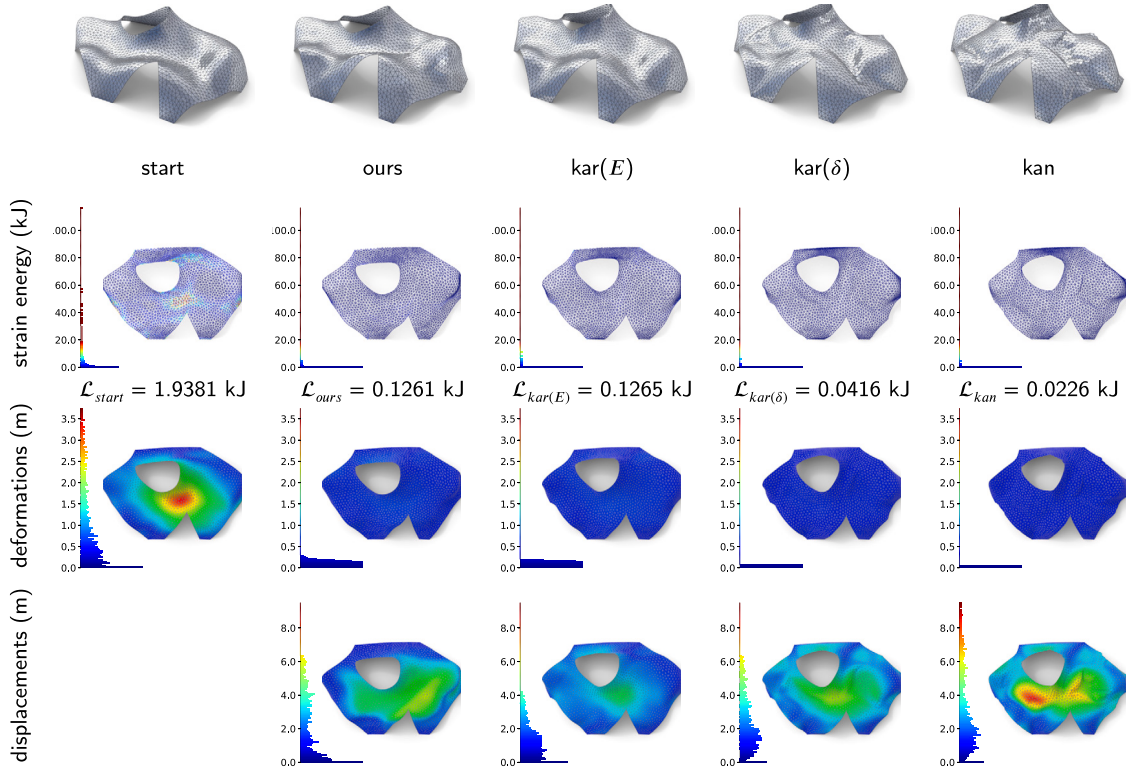

Fig. A.14. Tent. Results and comparisons with other tools.

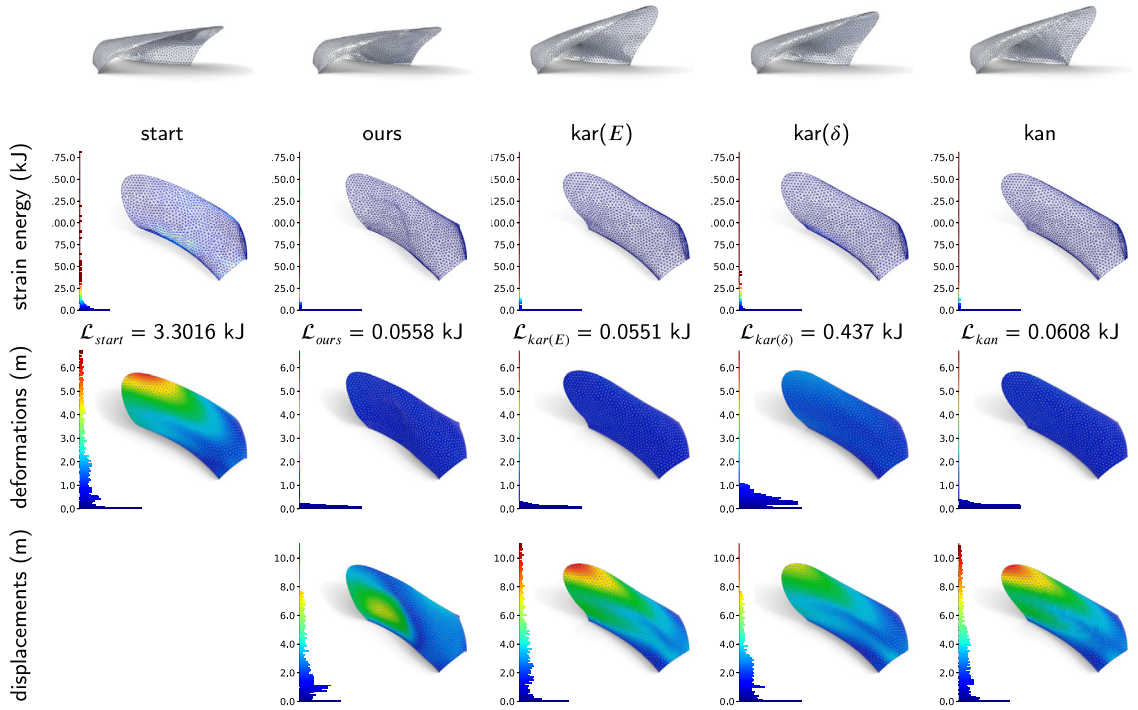

Fig. A.15. Yarn. Results and comparisons with other tools.

References

[1] Pottmann H, Schiftner A, Bo P, Schmiedhofer H, Wang W, Baldassini N, et al. Freeform surfaces from single curved panels. ACM trans. graph., vol. 27. New York, NY, USA: Association for Computing Machinery; 2008. p. 1–10.

[2] Deuss M, Panozzo D, Whiting E, Liu Y, Block P, Sorkine-Hornung O, et al. Assembling self-supporting structures. ACM trans. graph., vol. 33. New York, NY, USA: ACM; 2014. p. 214:1–21410:.

[3] Pottmann H, Eigensatz M, Vaxman A, Wallner J. Architectural geometry. Comput Graph 2015;47:145–64. https://doi.org/10.1016/j.cag.2014.11.002.

[4] Pietroni N, Tonelli D, Puppo E, Froli M, Scopigno R, Cignoni P. Statics aware grid shells. Comput Graph Forum 2015;34:627–41. https://doi.org/10.1111/cgf.12590.

[5] Kilian M, Pellis D, Wallner J, Pottmann H. Material-minimizing forms and structures. ACM trans. graph., vol. 36. New York, NY, USA: Association for Computing Machinery; 2017.

[6] Laccone F, Malomo L, Froli M, Cignoni P, Pietroni N. Automatic design of cable-tensioned glass shells. Comput Graph Forum 2020;39:260–73. https://doi.org/10.1111/cgf.13801.

[7] Adriaenssens S, Block P, Veenendaal D, Williams C. Shell structures for architecture: form finding and optimization. Routledge; 2014.

[8] Bletzinger K-U, Ramm E. Structural optimization and form finding of light weight structures. Comput Struct 2001;79:2053–62. https://doi.org/10.1016/S0045-7949(01)00052-9. Elsevier.

[9] Veenendaal D, Block P. An overview and comparison of structural form finding methods for general networks. Int J Solids Struct 2012;49:3741–53. https://doi.org/10.1016/j.ijsolstr.2012.08.008.

[10] Ahrari A, Deb K. An improved fully stressed design evolution strategy for layout optimization of truss structures. Comput Struct 2016;164:127–44. https://doi.org/10.1016/j.compstruc.2015.11.009.

[11] Sigmund O, Maute K. Topology optimization approaches: a comparative review, vol. 48. Springer; 2013. p. 1031–55.

[12] Fujita S, Ohsaki M. Shape optimization of free-form shells using invariants of parametric surface. Int J Space Struct 2010;25:143–57. https://doi.org/10.1260/0266-3511.25.3.143. SAGE Publications Sage UK: London, England.

[13] Firl M, Wüchner R, Bletzinger K-U. Regularization of shape optimization problems using FE-based parametrization. Struct. multidiscip. optim., vol. 47. Berlin, Heidelberg: Springer-Verlag; 2013. p. 507–21.

[14] Wang H, Chen Z, Wen G, Ji G, Xie YM. A robust node-shifting method for shape optimization of irregular gridshell structures. Structures 2021;34:666–77. https://doi.org/10.1016/j.istruc.2021.08.003. Elsevier.

[15] Tran-Ngoc H, Khatir S, Le-Xuan T, De Roeck G, Bui-Tien T, Abdel Wahab M. A novel machine-learning based on the global search techniques using vectorized data for damage detection in structures. Int J Eng Sci 2020;157:103376. https://doi.org/10.1016/j.ijengsci.2020.103376.

[16] Prachaseree P, Lejeune E. Learning mechanically driven emergent behavior with message passing neural networks. Comput Struct 2022;270:106825. https://doi.org/10.1016/j.compstruc.2022.106825.

[17] Leng Y, Tac V, Calve S, Tepole AB. Predicting the mechanical properties of biopolymer gels using neural networks trained on discrete fiber network data. Comput Methods Appl Mech Eng 2021;387:114160. https://doi.org/10.1016/j.cma.2021.114160.

[18] Bronstein MM, Bruna J, Cohen T, Velickovic P. Geometric deep learning: grids, groups, graphs, geodesics, and gauges. https://arxiv.org/abs/2104.13478. https://doi.org/10.48550/arXiv.2104.13478. arXiv:2104.13478, 2021.

[19] Wang Y, Sun Y, Liu Z, Sarma SE, Bronstein MM, Solomon JM. Dynamic graph CNN for learning on point clouds. ACM trans. graph., vol. 38. New York, NY, USA: Association for Computing Machinery; 2019.

[20] Hanocka R, Metzer G, Giryes R, Cohen-Or D. Point2Mesh: a self-prior for deformable meshes. ACM trans. graph., vol. 39. New York, NY, USA: Association for Computing Machinery; 2020.

[21] Norgaard S, Sagebaum M, Gauger N, Lazarov BS. Applications of automatic differentiation in topology optimization, structural and multidisciplinary optimization. https://doi.org/10.1007/s00158-017-1708-2, 2017.

[22] Yang B, Ma J, Zhang Q. Shape optimization of shell structures based on nurbs description using genetic algorithm. In: Proceedings of IASS annual symposia, volume 2013, international association for shell and spatial structures (IASS); 2013. p. 1–5.

[23] Pastrana R, Ohlbrock PO, Oberbichler T, D'Acunto P, Parascho S. Constrained form-finding of tension–compression structures using automatic differentiation. Comput Aided Des 2023;155:103435. https://doi.org/10.1016/j.cad.2022.103435.

[24] Wu G. A framework for structural shape optimization based on automatic differentiation, the adjoint method and accelerated linear algebra. https://doi.org/10.48550/arXiv.2211.15409. arXiv:2211.15409, 2022.

[25] Doyle JF. Static and dynamic analysis of structures: with an emphasis on mechanics and computer matrix methods. 1st ed. Netherlands: Springer; 1991.

[26] Veličković P, Casanova A, Liò P, Cucurull G, Romero A, Bengio Y. Graph attention networks. In: International conference on learning representations; 2018.

[27] Brody S, Alon U, Yahav E. How attentive are graph attention networks? In: International conference on learning representations; 2022. https://openreview.net/forum?id=F72ximsx7C1.

[28] Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE. Neural message passing for quantum chemistry. In: Proceedings of the 34th international conference on machine learning - volume 70; 2017. p. 1263–72.

[29] Qi CR, Yi L, Su H, Guibas LJ. Pointnet++: deep hierarchical feature learning on point sets in a metric space. In: Proceedings of the 31st international conference on neural information processing systems. Red Hook, NY, USA: Curran Associates Inc.; 2017. p. 5105–14.

[30] Guennebaud G, Gross M. Algebraic point set surfaces. In: ACM SIGGRAPH 2007 papers. New York, NY, USA: Association for Computing Machinery; 2007. p. 23.

[31] Guennebaud G, Germann M, Gross M. Dynamic sampling and rendering of algebraic point set surfaces. Comput Graph Forum 2008;27:653–62. https://doi.org/10.1111/j.1467-8659.2008.01163.x.
[32] Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, et al. Automatic differentiation in pytorch. In: NIPS 2017 workshop on autodiff; 2017. https://openreview.net/forum?id=BJJsrmfCZ.
[33] Piker D. Kangaroo: form finding with computational physics. Architectural design, vol. 83. Wiley Online Library; 2013. p. 136–7.
[34] Senatore G, Piker D. Interactive real-time physics: an intuitive approach to form-finding and structural analysis for design and education. Comput Aided Des 2015;61:32–41. https://doi.org/10.1016/j.cad.2014.02.007.
[35] Preisinger C, Heimrath M. Karamba - a toolkit for parametric structural design. Structural engineering international, vol. 24. Taylor & Francis; 2014. p. 217–21.
[36] McNeel R. Associates, Grasshopper generative modeling for Rhino, computer software. http://www.grasshopper3d.com, 2020.
[37] Hensel F, Moor M, Rieck B. A survey of topological machine learning methods. Front Artif Intell 2021;4. https://doi.org/10.3389/frai.2021.681108.