A geometry-preserving shape optimization tool based on deep learning*

Andrea Favilli$^{1,2\,[0009-0004-9342-7106]}$, Francesco Laccone$^{1\,[0000-0002-3787-7215]}$, Paolo Cignoni$^{1\,[0000-0002-2686-8567]}$, Luigi Malomo$^{1\,[0000-0001-7892-894X]}$, and Daniela Giorgi$^{1\,[0000-0002-6752-6918]}$

[1] ISTI - CNR, via G. Moruzzi 1, Pisa, Italy
[2] University of Pisa, Lungarno Pacinotti 43, Pisa, Italy
andrea.favilli@isti.cnr.it

Abstract. In free-form architecture, computational design tools have made it easy to create geometric models. However, obtaining good structural performance is difficult and requires further steps, such as shape optimization, to enhance system efficiency and material savings. This paper provides a user interface for form-finding and shape optimization of triangular grid shells. Users can minimize structural compliance, while ensuring small changes in their original design. A graph neural network learns to update the nodal coordinates of the grid shell to reduce a loss function based on strain energy. The interface can manage complex shapes and irregular tessellations. A variety of examples prove the effectiveness of the tool.

Keywords: Design tool · Shape optimization · Graphical User Interface · Geometric learning.

1 Shape optimization of shells and grid shells

In the structural design of shells and grid shells, shape optimization refers to methods and tools for maximizing the performance of an input surface. As building components, shells and grid shells are subject to unique contextual, functional, constructional or aesthetic constraints. Therefore, the relationship between shape and structural variables can differ case by case. Additionally, these structures are often shaped as free-form surfaces, which complicates the structural response.

A properly-designed shell or grid shell should possess a prevailing membrane behavior, avoid bending forces, and have good robustness [1]. In the ideal case, a 'natural' shape is sculpted by providing loading and boundary conditions to form-finding methods [2]. However, in the general case the shape is not naturally efficient and requires several design checks and improvements, which can

* 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)

either rely on heuristics and experience, or adopt automated shape optimization methods. The objective functions usually involve structural quantities that are iteratively updated by moving the design variables within predefined bounds.

The literature offers a large variety of shape optimization methods, differing in the types of variables, objective functions, optimization methods etc. In [3], finite element mesh nodes are the variables updated through deformation gradient. The solution is enriched with surface regularization and distortion control. The authors of [4] emply a NURBS surface formulation, using control points and thickness as variables. The shape modifications result from finite element analysis and gradient evaluations, performed using Automatic Differentiation (AD). In [5], structural stiffness is maximized in a grid shell by iteratively updating nodal coordinates according to sensitivity information. Since the procedure is initialized with a displacement perturbation, which can lead to a jagged surface, a filtering scheme is adopted to normalize the non-smooth gradient fields. Other shape optimization methods are formulated as multi-criteria optimization. In [6], curves and surfaces can have discontinuities in tangent vectors and curvatures (creases). A multi-objective optimization problem is solved by the constraint approach to generate a trade-off design between smoothness and mechanical compliance.

In this paper, we implement a Graphical User Interface based on the method in [7], which modifies a grid shell shape using deep learning. The user can edit a set of input parameters, such as the beam characteristics and the solution speed. The graph neural network in the background is fed with geometric input features and the structural analysis of the grid shell. The variables are node coordinates. We showcase the potential of our tool with three representative examples. In two examples we perform shape optimization on free-form shells. Additionally, since shape optimization methods are often applied to solve form-finding problems [8], in a third example we solve a form-finding task on a flat surface input.

After briefly introducing the problem formulation and the learning model in Section 2, we describe how to reformulate the model for the tool implementation and a typical usage scenario in Section 3. Finally, Section 4 presents some example results that a user can achieve.

2 Geometric deep learning for shape optimization

Our tool is based on a graph neural network model [7] driven by the static analysis of grid shells. One of the main innovations is the introduction of a differentiable Euler-Bernoulli approach which fits usage in a learning model.

A triangular mesh $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ represents the input grid shell. $\mathcal{V}$ are the vertices (identified with the structural nodes), $\mathcal{E}$ are the edges (identified with the beams), and $\mathcal{F}$ are the mesh faces. The learning-based shape optimization acts on the vertices by imposing translations, while retaining the original mesh connectivity. In other words, the neural network model predicts an optimal translation vector $\delta_{\mathbf{v}} \in \mathbb{R}^3$ for each vertex $\mathbf{v} \in \mathcal{V}$. Fixed nodes can be constrained so that translations are null for a given subset of vertices $\overline{\mathcal{V}} \subseteq \mathcal{V}$.

The neural network model takes as input an augmented encoding of the original structure. We combine different pieces of information in an *input feature vector* $\mathbf{x}_\mathbf{v} \in \mathbb{R}^{12}$, for each vertex $\mathbf{v} \in \mathcal{V}$. The input feature vector includes vertex coordinates, vertex normals, principal curvatures of the underlying free-form surface, and four distance and centrality measures of vertices with respect to the shell boundary $\partial\mathcal{V}$ and the fixed nodes $\overline{\mathcal{V}}$. Then, a deep sequence of layers transforms features until the final layer yields the prediction $\delta_\mathbf{v}$. At each layer, vertex features are updated by computing the weighted average of the features of the nearest-neighboring vertices. The network weights are determined by minimizing the *loss*, a target function accounting for structural compliance. The loss minimization follows a gradient descent method: starting from randomly initialized network weights, the weights are moved along the direction of maximum loss decrease (the opposite of the gradient) in an iterative procedure. At each step, the decrease direction vector is scaled by the *learning rate* parameter.

The learning model uses a two-node Euler-Bernoulli linear beam formulation. The loss function $\mathcal{L}(\mathcal{M})$ is defined as the mean strain energy over all edges:

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

with E_e the strain energy on a single edge e.

The loss in Eq. (1) is made differentiable with respect to vertex coordinates so that it can be used to determine the neural network weights in accordance with gradient descent minimization. The method relies on *automatic differentiation* (AD) to get punctual evaluations of gradients, thus avoiding the need to express analytically the partial derivatives of $\mathcal{L}$.

Our shape optimization task falls within the frame of *single-instance learning.* For each input mesh, we reset the network weights and we start a new loss minimization procedure. The task can be formalized as finding an optimal mesh $\mathcal{M}^* := T_{\boldsymbol{\theta}^*}(\mathcal{M})$ such that

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

where the neural network is a function $T_{\boldsymbol{\theta}}$ of the weights $\boldsymbol{\theta}$.

While [7] adopts a scale-dependent stopping criterion for the iterative procedure, the graphical user interface proposed in this paper allows the user to enter the execution flow. The user can change the optimization parameters on the fly and visualize the shape modification from different perspectives. Moreover, the user can replay or restart the procedure until he/she founds a balance between performance metrics and amount of shape modification.

Finally, since a user interface needs a fast response for functional evaluation of the loss and its partial derivatives, we express the loss from Eq. (1) as a sequence of vectorized operations that support AD, without using code branching to build intermediate results. Our array-based implementation can benefit from GPU parallelization to speed up the process.

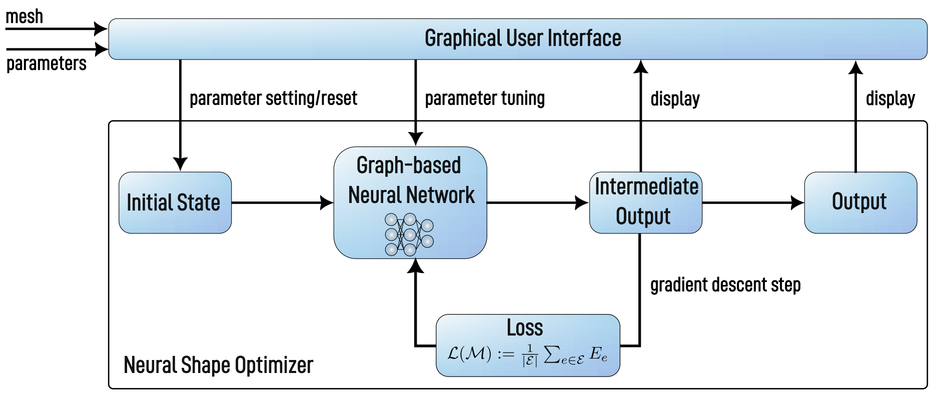


Fig. 1. The pipeline of our tool. The input is a grid shell encoded as a triangle mesh. The user can also input parameters concerning the structure (e.g., beam cross sections, Young's modulus, etc). As the learning and optimization procedure starts, partial shape modifications are visualized in real time, so that the user can tune parameters on the fly, control the optimization flow or restart the procedure with different settings. The user can explore the outcomes until he/she reaches a good compromise between shape and performance.

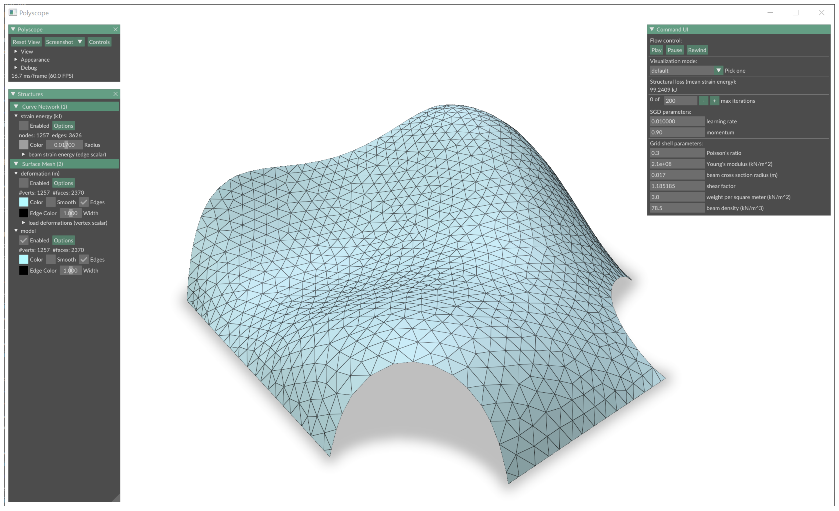
3 Tool design and user interface

The tool is implemented in the Python programming language and is made of two main components: the Graphical User Interface and the Neural Shape Optimizer (Fig. 1). Each component runs on a different process, so computations can run asynchronously from the interface display task. This scheme implies that the user can explore the interface functionalities while the neural network modifies the shape. The Graphical User Interface detects parameter changes and updates the Neural Shape Optimizer process. Inter-process communication is implemented through two directed queues; one sends parameter changes from the Graphical User Interface to the Shape Optimizer, and the other sends back results to update the visualization.

We employed the PyTorch deep learning library [9] for the Neural Shape Optimizer. PyTorch offers an easy way to define the architecture of neural networks, with full support to Automatic Differentiation and a large variety of array-based operations. We also used PyTorch Geometric [10] to implement the graph-based network layers. To implement the Graphical User Interface we used Polyscope [11], a flexible 3D object viewer with a customizable user interface. Fig. 2 shows the GUI.

3.1 Usage scenario

To describe the usage and features of the user interface, we introduce Frank, a fictitious designer who can benefit from the tool in an early conceptual design



16.7 ms/frame (60.0 FPS)

strain energy (kJ)
Enabled
nodes: 1257 edges: 3626
Color 0.01700 Radius
beam strain energy (edge scalar)

deformation (m)
Enabled
#verts: 1257 #faces: 2370
Color Smooth ✓ Edges
Edge Color 1.000 Width
load deformations (vertex scalar)

model
✓ Enabled
#verts: 1257 #faces: 2370
Color Smooth ✓ Edges
Edge Color 1.000 Width

Visualization mode:
default Pick one

Structural loss (mean strain energy):
99.2409 kJ

0 of 200 - + max iterations

SGD parameters:

0.010000	learning rate
0.90	momentum

Grid shell parameters:

0.3	Poisson's ratio
2.1e+08	Young's modulus (kN/m^2)
0.017	beam cross section radius (m)
1.185185	shear factor
3.0	weight per square meter (kN/m^2)
78.5	beam density (kN/m^3)

Fig. 2. Our interface at launching with an input grid shell. On the left, the user can find the default Polyscope tools (Polyscope and Structures boxes), where visualization settings (camera and shape appearance) can be modified. On the right (Command UI box), the user can control the shape optimization algorithm, check the structural performance in real time, and switch between different visualizations of metrics.

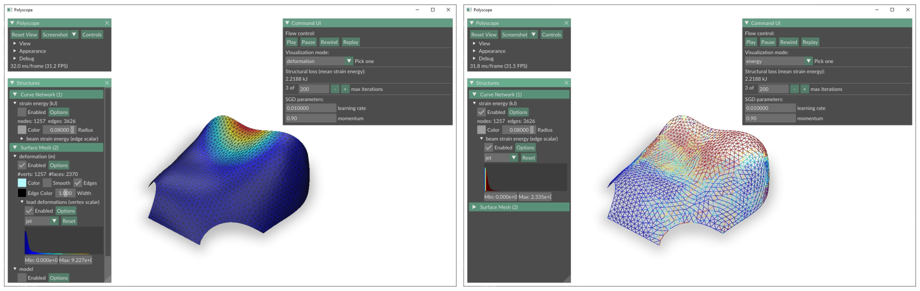
phase. Even though Frank has a clear idea about the grid shell shape he wants to design, he needs to produce a practically feasible solution. Thanks to our user interface, he can feed the optimization tool with a tentative shape concept, then drive the optimization towards a feasible solution from a structural point of view, while preserving his original design intent.

Frank loads the initial shell as a 3D model file and enters the optimization flow. He controls the shape optimization with a set of buttons recalling the videotape jargon: *Play*, *Pause*, *Rewind* and *Replay*. He knows almost nothing about Artificial Intelligence but was instructed about the only relevant parameter concerning the neural network he might want to tune: the learning rate. The interface proposes a default value for the learning rate, based on the dimension of the input structure. However, Frank might want to change the default parameter according to his needs, knowing that a lower learning rate would produce slow convergence of the gradient-based optimization, while a higher learning rate would lead to convergence in a smaller number of steps. On the other hand, Frank knows that the advantage of a lower learning rate is that it produces less distortion on shapes. The learning rate also depends on the initial loss, that is, on the energy of the input surface design: high energy inputs are likely to require lower rates to converge, while small energy inputs need higher rates to boost gradient descent towards a local optimum. In what follows, we describe Frank's interaction with the interface to refine his design.

Frank is in front of the initial interface screen, Fig. 2. Given the scale of the shell bounding box, he deems appropriate to set the *beam cross-section radius* field to 0.08 m, *learning rate* to 0.0001 and 200 *max iterations*. The other default *grid shell parameters* are considered valid, i.e. a *Poisson's ratio* of 0.3 and a *Young's modulus* of 2.1×10^8 kN/m^2. The Load is a sum of two terms: the structure's own weight, which is computed from the *beam density*, and an external face load, which is in the direction of gravity and is applied per node on a lumped scheme. Frank considers appropriate the default Load as a *weight per square meter* of 3 kN/m^2 and a *beam density* of 78.5 kN/m^3. At this point, he clicks on the *Play* button, and the shape optimization procedure starts.

After 50 iterations, Frank realizes that the loss decrease is too slow. Hence he clicks on the *Pause* button, increases the *learning rate* from 0.0001 to 0.0005, and resumes the procedure after clicking again *Play*. When all 200 iterations are carried out, Frank decides to check the evolution of the shape optimization process from a different perspective, switching to a different visualization.

Indeed, our interface enables the user to visualize not only the surface geometry and tessellation, but also the deformation under Service Load and the strain energy, mapped on the surface mesh in false colours (Fig. 3).


Left panel:
Polyscope: 32.0 ms/frame (31.2 FPS)
Curve Network (1): strain energy (kJ); nodes: 1257 edges: 3626; Radius 0.08000
Surface Mesh (2): deformation (m) Enabled; #verts: 1257 #faces: 2370; Edge Color Width 1.000; load deformations (vertex scalar) Enabled, jet; Min: 0.000e+0 Max: 9.227e+0
Command UI: Visualization mode: deformation; Structural loss (mean strain energy): 2.2188 kJ; 3 of 200 max iterations; SGD parameters: 0.010000 learning rate, 0.90 momentum

Right panel:
Polyscope: 31.8 ms/frame (31.5 FPS)
Curve Network (1): strain energy (kJ) Enabled; nodes: 1257 edges: 3626; Radius 0.08000; beam strain energy (edge scalar) Enabled, jet; Min: 0.000e+0 Max: 2.335e+0
Command UI: Visualization mode: energy; Structural loss (mean strain energy): 2.2188 kJ; 3 of 200 max iterations; SGD parameters: 0.010000 learning rate, 0.90 momentum

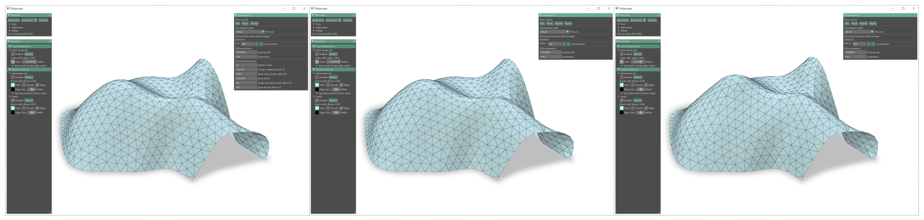
Fig. 3. Switching to different visualizations than geometry only: (Left) deformation under Service Load, (Right) strain energy visualization.

Frank selects the *energy* option from the dropdown menu, and clicks *Replay*. The *Replay* option is available only when the execution flow is paused in a state different from the initial. Finally, Frank can accept the outcome, perform some more iterations, or restart the procedure with different settings after clicking *Rewind*.

4 Results and discussion

We analyze three different input shapes. The objective is to highlight the relation between user interaction inputs, parameter settings, and outcomes. Two examples shapes, Ice and Waving, are free-form grid shells with different structural compliance. The third example is a flat grid shell (Flat) to test the form-finding capabilities of our tool.

According to the size of the examples, we first use a learning rate of 0.01 (λ in Figs. 4, 5 and 6) and 200 gradient descent iterations (it) for shape optimization. However, the user can edit these parameters to alter the output shape. Ice and Waving show a different behavior for shape optimization in terms of design preservation. While both are complex surfaces designed sculpturally, Waving has a better initial static configuration, independently of the chosen beam cross-section. While static performance is the objective function, our tool tries to preserve the input shape. This happens with the settings mentioned above, as shown in Fig. 4 (b), as well as with different input parameters set via the interface, as in Fig. 4 (c), where the cross-section radius reduces to 0.04 m, the learning rate increases to 0.05 and the number of iterations to 500. For both cases, it can be seen that the loss goes down, while the overall design and surface features are preserved.

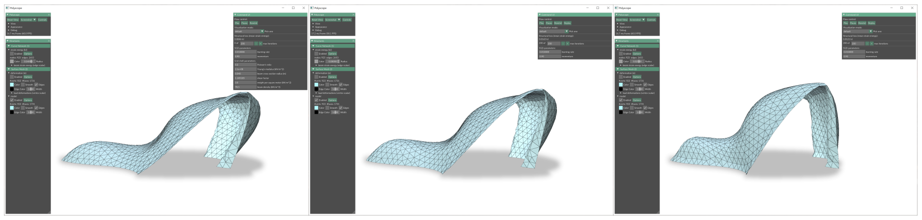


$r_{CS} = 0.08\ m$ $\lambda = 0.01$ 200 it | $r_{CS} = 0.04\ m$ $\lambda = 0.05$ 500 it

(a) (b) (c)

Fig. 4. Shape optimization on the Waving model: (a) the input structure ($\mathcal{L} = 0.0042\ kJ$ if $r_{CS} = 0.08\ m$, $\mathcal{L} = 0.0097\ kJ$ if $r_{CS} = 0.04\ m$); (b) the output for suitable parameters ($\mathcal{L} = 0.0038\ kJ$); (c) shape modification is still slight if we reduce the cross-section radius, and we increase the learning rate and the number of iterations ($\mathcal{L} = 0.0033\ kJ$).

Ice is more sensitive to the setting of the structural parameters, for the same gradient descent parameters $\lambda = 0.01$ and 200 iterations. In particular, as the size of the beam cross-section is closely involved with the edge strain energy, in Figs. 5 (b) and (c), we compare the results obtained with two different solid circular cross-section radii (r_{CS}). On the one hand, a higher radius means a higher beam weight and, consequently, a higher load. On the other hand, larger beams have more bending stiffness and collect less stress due to external load. The case $r_{CS} = 0.08\ m$ (Fig. 5 (b)) is less energized than the case $r_{CS} = 0.04\ m$ (Fig. 5 (c)), leading to a lighter shape modification with respect to the input. The case $r_{CS} = 0.04\ m$ tends towards a membrane solution.



$r_{CS} = 0.08\ m$
$\lambda = 0.01$
200 it

$r_{CS} = 0.04\ m$
$\lambda = 0.01$
200 it

(a) (b) (c)

Fig. 5. Shape optimization on the Ice model: (a) the input structure ($\mathcal{L} = 0.0466\ kJ$ if $r_{CS} = 0.08\ m$, $\mathcal{L} = 0.3103\ kJ$ if $r_{CS} = 0.04\ m$); (b) the output for suitable parameters ($\mathcal{L} = 0.0170\ kJ$); (c) the output if the cross-section radius is reduced ($\mathcal{L} = 0.0117\ kJ$).

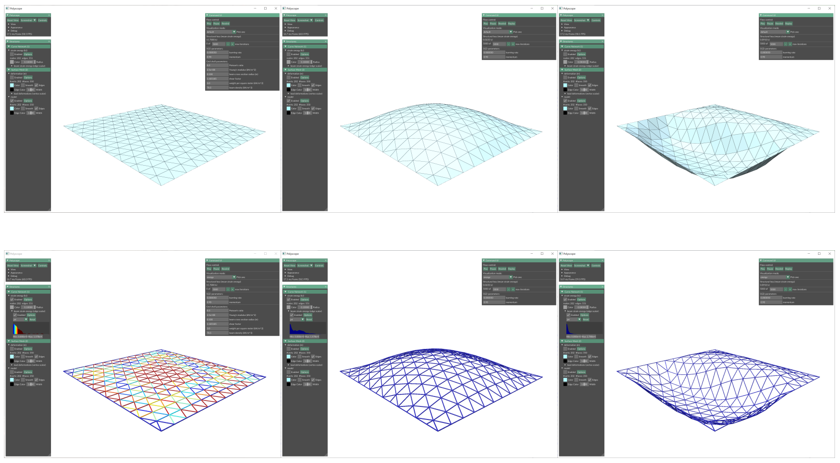
Our tool can also be employed for form-finding. Dealing with a flat surface requires re-calibrating the input parameters. Indeed, while shape optimization aims at validating and hopefully preserving an input shape, form-finding requires heavy shape modification. The third model, Flat, is a high-energy grid shell. Hence, it requires lower learning rates to ensure convergence compared to the previous examples. Fig. 6 reports the outcomes of form-finding with $\lambda = 5 \cdot 10^{-5}$. The figure also shows how beam cross-sections affect the initial strain energy, and consequently the behaviour of the optimizer, which converges towards different minima: the classic funicular pillow shape in Fig. 6 (b), or the symmetric membrane surface used in tension and compression in Fig. 6 (c).

5 Conclusions

We developed a graphical interface tool to perform shape optimization and form finding based on deep learning. Our tool lets the user break up the linearity of a classical iterative procedure, and move the outcome back and forth as if the optimization task was a videotape playing.

This contribution is a first step towards the development of a complete AI-based software for assisted design. Indeed, architecture is for humans, and our tool puts humans at the center of the decision process. Automating the design process is not straightforward. However, we provide the designer with a tool to support the design space exploration. In real-time, different shape outcomes can be visualized after each parameter change.

This tool lays essential methodological foundations and can be expanded to include other feasibility/aesthetics metrics besides statics. A human-machine joint approach can enhance design quality, by accounting for both aesthetics and feasibility, and also benefit creativity.



$r_{CS} = 0.1\ m$
$\lambda = 5 \cdot 10^{-5}$
5000 it

$r_{CS} = 0.08\ m$
$\lambda = 5 \cdot 10^{-5}$
5000 it

(a) (b) (c)

Fig. 6. Form-finding starting from Flat grid shell. (a): the input structure ($\mathcal{L} = 15.7806\ kJ$ if $r_{CS} = 0.1\ m$, $\mathcal{L} = 27.8012\ kJ$ if $r_{CS} = 0.08\ m$); (b) and (c): two different local minima output ($\mathcal{L} = 0.0240\ kJ$ and $\mathcal{L} = 0.0901\ kJ$ respectively). The second row shows color-mapped strain energy on beams.

References

1. Ramm, E.: Shape finding of concrete shell roofs. Journal of the International Association for Shell and Spatial Structures **45**(1), 29–39 (2004)
2. Veenendaal, D., Block, P.: An overview and comparison of structural form finding methods for general networks. International Journal of Solids and Structures **49**(26), 3741–3753 (2012). https://doi.org/10.1016/j.ijsolstr.2012.08.008
3. Bletzinger, K.U., Firl, M., Linhard, J., Wüchner, R.: Optimal shapes of mechanically motivated surfaces. Computer methods in applied mechanics and engineering **199**(5-8), 324–333 (2010). https://doi.org/10.1016/j.cma.2008.09.009
4. Espath, L., Linn, R.V., Awruch, A.: Shape optimization of shell structures based on nurbs description using automatic differentiation. International Journal for Numerical Methods in Engineering **88**(7), 613–636 (2011). https://doi.org/10.1002/nme.3183
5. Wang, H., Chen, Z., Wen, G., Ji, G., Min Xie, Y.: A robust node-shifting method for shape optimization of irregular gridshell structures. Structures **34**, 666–677 (2021). https://doi.org/10.1016/j.istruc.2021.08.003
6. Ohsaki, M., Ogawa, T., Tateishi, R.: Shape optimization of curves and surfaces considering fairness metrics and elastic stiffness. Structural and Multidisciplinary Optimization **27**, 250–258 (2004). https://doi.org/10.1007/s00158-004-0382-3
7. Favilli, A., Giorgi, D., Laccone, F., Malomo, L., Cignoni., P.: Geometric deep learning for statics-aware 3d gridshells. Tech. rep., ISTI - CNR (07 2022). https://doi.org/http://dx.doi.org/10.32079/isti-tr-2022/016

8. Bletzinger, K.U., Ramm, E.: Form finding of shells by structural optimization. Engineering with computers **9**, 27–35 (1993)
9. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
10. Fey, M., Lenssen, J.E.: Fast graph representation learning with pytorch geometric (2019), http://arxiv.org/abs/1903.02428, cite arxiv:1903.02428
11. Sharp, N., et al.: Polyscope (2019), www.polyscope.run