Computers & Graphics 138 (2026) 104629

Contents lists available at ScienceDirect

Computers & Graphics

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

Special Section on CEIG 2026

Strokes2Deform: Physics-informed learning of deformation fields on 3D stroke clouds☆

Shervin Rasoulzadeh [a,*], Raman Suliman [a], Arvin Rasoulzadeh [b], Iva Kovacic [a], Michael Wimmer [a]

[a] TU Wien, Center for Geometry and Computational Design, Karlsplatz 13, A-1040, Vienna, Austria
[b] METROM Mechatronische Maschinen GmbH, Schönaicher Straße 6, 09232, Hartmannsdorf, Germany

ARTICLE INFO

Dataset link: https://gitlab.cg.tuwien.ac.at/srasoulzadeh/strokes2deform.git

Keywords:
Sketch-analysis
Deformable models
Physical AI

ABSTRACT

In the 3D architectural design process, deformation analysis of a designed object is a fundamental step. However, such a step is usually inaccessible to designers in the early sketching phase, limiting designers' ability to reason about the shape's structural stability. To bridge this gap, we propose Strokes2Deform, an end-to-end learning-based model that enables deformation-aware 3D sketching, requiring neither (surface) reconstruction nor (physical) simulation. Our physics-informed neural network takes as input a 3D sketch stroke cloud and user-specified boundary conditions, and directly predicts the induced deformation field on the 3D stroke cloud. Key to our method is: (i) a synthetic dataset of 40K 3D sketch–deformation pairs of architectural thin-shell structures and their corresponding deformation fields derived from Finite Element (FE) analysis, (ii) the use of fVDB to encode 3D sketch stroke clouds and their per-point features into sparse voxel grids with attributes, and leveraging its differentiable splatting and sampling operators for bidirectional point–voxel mappings, (iii) a dual-head neural network architecture that decouples deformation field prediction into unit-length displacement vectors and scalar displacement magnitudes, and (iv) our physics-informed loss functions derived from thin-shell deformation principles. We validate our method through extensive experiments, demonstrating accurate deformation prediction across sketches of varying complexity and strong agreement with reference deformations derived from FE simulations.

1. Introduction

Drawing, whether physical or digital, has been a quintessential tool for visual communication for millennia. Beyond the established practice of drawing with pen and paper, i.e., 2D sketching, 3D pen-on-tablet devices and Virtual and Augmented Reality (VR/AR) environments are giving designers unprecedented freedom to draw directly in 3D [1]. The rapid advancement and growing availability of these 3D sketching interfaces across architectural and industrial design applications [2–4] are driving a paradigm shift and prompting exploration in these domains [5].

In the natural workflows of such applications, sketching goes beyond serving as a medium for communicating design ideas and geometric shapes and is often followed by digitization and, eventually, the realization of the digital model. This requires designers to manage requirements of very different, often contradictory, nature: create appealing designs, use little material, and ensure the structure is stable [6]. As a result, designers are often forced into a cumbersome design workflow loop that begins with shaping an idea as a rough sketch, followed by creating the corresponding digital model and running physical simulations under various engineering constraints, whose feedback then informs revisions to the original sketch [7]. This workflow loop should be as short as possible and should not disrupt the creative feedback flow. In practice, however, the time spent on manual 3D modeling, simulation, and feedback communication impedes a seamless creative feedback flow.

Existing works on 3D sketch-based shape analysis introduce a range of new interaction techniques [4,8,9] and surface reconstruction methods for various input data types stemming from such interfaces, including point clouds [10], stroke clouds [1,11–14], and curve networks [15,16], whereas only a smaller body of related work considers physical properties of the design objects. These approaches typically focus on 2D

☆ This article is part of a Special issue entitled: 'CAG_CEIG 2026' published in Computers & Graphics.

* Corresponding author.
E-mail addresses: srasoulzadeh@cg.tuwien.ac.at (S. Rasoulzadeh), raman.suliman@tuwien.ac.at (R. Suliman), arvin.rasoulzadeh@metrom.com (A. Rasoulzadeh), iva.kovacic@tuwien.ac.at (I. Kovacic), wimmer@cg.tuwien.ac.at (M. Wimmer).

https://doi.org/10.1016/j.cag.2026.104629
Received 10 April 2026; Received in revised form 11 May 2026; Accepted 13 May 2026
Available online 26 May 2026

sketches [17,18] or rely on an intermediate surface reconstruction step in order to perform physical simulations such as deformation and structural analysis [19]. This has left physical analysis largely underexplored in the 3D sketching domain, limiting designers' ability to reason about a shape's physical properties and structural soundness within such interfaces. However, facilitating an informed exploration with respect to downstream design assessments such as structural stability analysis—where stability of a structure is traditionally expressed as the amount of deformation induced by particular boundary conditions [6]—during the early sketching phase can substantially reduce effort and time in the design pipelines by avoiding costly trial-and-error iterations [20].

Inspired by these challenges, we present *Strokes2Deform*, a physics-informed neural network that enables deformation-aware 3D sketching requiring neither (surface) reconstruction nor explicit (physical) simulation. Specifically, we introduce an end-to-end model that takes as input a 3D sketch stroke cloud and user-specified boundary conditions, and directly predicts the induced deformation field on the 3D stroke cloud, reflecting the physical response of the underlying user-intended manifold surface. Our method is enabled by four key factors: (i) We construct a synthetic dataset of 40K sketch–deformation pairs from architectural thin-shell 3D sketches, where each sketch is paired with Finite Element (FE) simulation results mapped directly onto the sketch stroke points, (ii) We use fVDB [21] to encode sketch topology and per-point features into a sparse voxel grid and leverage its differentiable splatting and sampling operations to map point-wise features to voxels and transfer the network's voxel-wise predictions back onto the 3D stroke points, (iii) we design a dual-head network architecture with a shared backbone that decouples deformation prediction into unit-length displacement vectors and scalar displacement magnitudes, stabilizing training and enabling geometry-aware supervision, and (iv) we formulate loss functions based on thin-shell deformation principles, where stretching and bending energies are discretized and defined on the family of 3D strokes instead of parametrized surfaces.

Our method accepts inputs from a variety of sources, including 3D sketching interfaces [3,4,22], traditional modeling software [23,24], and various curve-tracing algorithms. Through extensive experiments, we show that our method accurately predicts deformation for sketches of varying complexity, closely matching reference deformations obtained by projecting numerical FE simulation results onto the strokes. Our code and dataset are available at: https://gitlab.cg.tuwien.ac.at/srasoulzadeh/strokes2deform.git.

2. Related work

Although our method does not require surface reconstruction and explicit physical simulation, it is closely inspired by prior work on *3D sketch-based modeling* and data-driven physics-informed neural networks for *deformation and structural analysis*.

2.1. 3D sketch-based modeling

The advent of practical 3D sketching interfaces has motivated the development of algorithms for inputs originating from such interfaces, including curve networks, stroke clouds, and other geometric representations derived from 3D sketches, such as point clouds.

Surfacing curve networks. Numerous works focus on automatically surfacing sparse, designer-drawn sets of well-connected curves, also known as curve networks [15,25,26]. In general, surfacing curve networks is considered a two-step challenge. The first challenge is to discover which closed cycles in a curve network define valid surface patches, while the second challenge is to generate the patch geometries that interpolate the cycles' boundary curves. As a consequence, several surfacing algorithms strongly rely on the connectivity of the curve network to identify closed cycles delimiting surface patches [25,27,28]. Once surface-defining cycles are identified, curve connectivity provides strong geometric cues, enabling surface normal estimation at intersections to detect sharp features and flow-line curves, which are then propagated from boundary curves to reconstruct surface patches [26,29–31].

Surfacing stroke clouds. In the computer graphics and vision community, there has been a growing trend toward methods that reconstruct 3D geometries from stroke clouds, i.e., raw, imperfect, and unstructured sets of 3D strokes [1,11,13,14,16,32,33]. As the topology of the desired shape is unknown given the rough, unstructured inputs, some methods approach the problem by formulating optimization techniques that enforce approximation accuracy and piecewise smoothness [1,33], while other methods address the problem either by reconstructing a closed manifold surface from the exterior while interpolating a sparse set of disconnected strokes [13], or conversely, by progressively filling the interior via the generation and merging of Voronoi balls [14]. More recently, emerging learning-based methods have approached the problem by recovering curve networks as an intermediate representation [16] or by directly reconstructing 3D shapes using end-to-end pipelines trained on large datasets of 3D shapes [12,34].

Surfacing point clouds. Methods for surface reconstruction from point clouds [35] can potentially be applied as-is to stroke polyline vertices or to a dense set of points sampled along strokes' triangle strips. However, these methods have primarily been designed to process dense point clouds acquired via 3D scanning technology [36,37]. As a result, they may provide only coarse approximations for densely sketched inputs and often fail on sets of 3D sketches that contain only a very sparse, non-uniform sampling of the envisioned surface.

2.2. Deformation and structural analysis

Detecting structural weaknesses in 3D designs and models is a problem that has received considerable attention from the graphics community. One line of research attempts to integrate structural analysis/recommendation into the early sketching phase [17,18,20,38]. However, these methods only target 2D sketches of particular objects, such as building plans or trusses. Close to our work in spirit is [18], which introduces a data-driven generative model that enables users to perform stress prediction on desired 2D sketched objects by transforming the problem into an image-to-image translation task. Other studies aim to support informed 3D design by providing structural feedback during the early stages of design; however, they rely on an intermediary reconstruction step from point sets or curve networks and on connections to Finite Element Analysis (FEA) software [19,39]. Another related work addresses the simplification of curve networks with respect to structural stability and material cost [6]. Their method assumes highly regular networks with *a priori* known connectivities and does not extend to rough sketches where such information is unavailable or ill-defined. In contrast, we target unstructured stroke clouds that are imprecise, exhibit over-sketching or gaps, and lack inter-stroke connectivity while collectively depicting a surface. More related to this work are Physics-Informed Neural Networks (PINNs), which go beyond fitting observed data by incorporating physical constraints through loss functions or regularization terms [40–42].

3. Overview

In this work, we focus on deformation analysis for 3D-sketched thin-shell structures under user-specified boundary conditions placed at arbitrary locations. Extending physical simulation from well-defined 3D geometries to 3D sketches is nontrivial, as imperfect and unstructured collections of 3D strokes provide an ill-posed representation of continuous surfaces. We address this challenge by marrying 3D stroke clouds with deformation fields defined on their stroke polyline vertices and devising an end-to-end learning-based model that trains on these

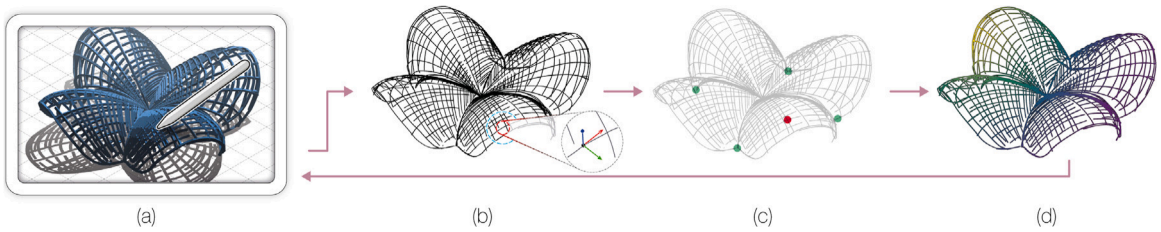

Fig. 1. Strokes2Deform Overview. The user sketches a sparse set of unoriented 3D strokes to depict the intended surface (a). The input to our model consists of the stroke cloud topology, together with per-point orthonormal frame components and user-specified boundary condition annotations (load and support points), encoded as per-point feature vectors (b, c). The model outputs deformation field on the 3D stroke cloud, providing immediate feedback that identifies structurally weak regions (d).

pairs to predict induced deformation fields without requiring reconstruction and simulation (see Fig. 1). To this end, we first describe the core common characteristics of 3D architectural design sketches (see Section 4). Next, as 3D sketches lack sizable ground-truth datasets with corresponding surfaces or simulations, we collect a set of unique 3D sketches and devise a synthetic data generation pipeline that couples 3D sketch stroke clouds with induced deformation fields (see Section 5). Finally, we propose a Physics-Informed Neural Network (PINN) that incorporates thin-shell deformation laws into the loss function and learns to predict deformation fields on 3D stroke clouds (see Section 6).

4. Input drawing characteristics

Strokes2Deform operates directly on 3D curves and requires no additional input information, allowing it to accept inputs originating from a variety of sources, such as traditional modeling software (e.g., Blender and Rhino) and various curve tracing algorithms. However, the method is practically tailored to early-stage conceptual design, where 3D sketch strokes serve as the primary input representation [3,4,22].

Interaction in 3D sketching interfaces can take a variety of forms, including 3D pen-on-tablet sketching, where drawing surfaces (aka *proxy canvases*) are used to project strokes into 3D, and freehand mid-air drawing in immersive VR/AR environments, where a sketch pen becomes a magic wand allowing users to step into their creations for directly drawing in 3D. In architectural design use cases, recent studies reveal that users often draw different types of strokes to communicate their design ideas. Specifically, they often sketch the boundaries of the intended object to make the idea more concrete, and fill in the enclosed area using techniques such as hatching or scribbling to further solidify the concept [8,16] (see Fig. 2). From a design-computation perspective, this leads to dense 3D sketches that are visually expressive and readily perceived by viewers, yet remain non-trivial to process for further design analysis and communication.

Motivated by these observations, we design our method to operate directly on unstructured 3D sketches that naturally arise during the conceptual design phase, while handling different levels of stroke density and sparsity. In the remainder of this manuscript, we refer to this raw set of 3D stroke polylines forming the 3D sketch as a *stroke cloud*. Overall, we evaluate our method on 3D sketches created with Feather [3], a commercial 3D pen-on-tablet sketching interface that uses translucent 3D guide surfaces to project strokes from the drawing viewpoint in order to generate 3D curves.

5. Synthetic data generation pipeline

No existing dataset couples 3D sketch stroke clouds with deformation fields obtained from physical simulation. To address this gap, we collect a set of unique 3D sketches depicting architectural thin-shell structures (see Fig. 2) and devise a synthetic data generation pipeline (see Fig. 3) that establishes this coupling in three stages: (i) *modeling*, where we create a surface model according to each sketch and edit it into the desired shape, (ii) *deformation analysis*, where we perform FE-based deformation analysis under 500 different boundary conditions on each surface model, and (iii) *mapping*, where we map both the input boundary conditions and the simulation-derived deformation fields from the surface models to the stroke clouds to form sketch–deformation pairs. In total, this process yields 40K sketch–deformation pairs. In the following, we describe each stage in more detail.

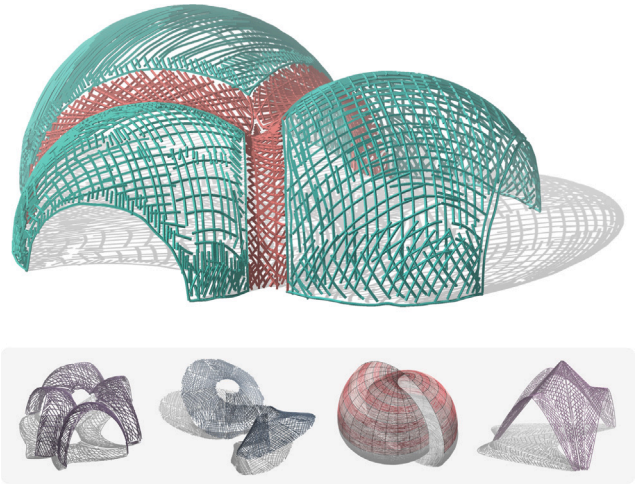

Fig. 2. Sample 3D sketches of architectural thin-shell structures created using Feather [3]. Input sketch: Raman Suliman.

5.1. Modeling

For each 3D sketch, we begin by applying Poisson reconstruction [37] to the stroke polyline vertices as-is to obtain an initial approximation of the underlying surface model. We then manually trim and refine the resulting mesh into the user-intended surface by correcting reconstruction artifacts and adjusting regions where the inferred shape deviates from the intended form. This is followed by adaptively resampling the surface mesh vertices to reduce the simulation's computational cost. Finally, we remesh the surface, constrained at the boundary curves, into a uniform triangulation that preserves salient geometric features, such as sharp edges, corners, ridges, and high-curvature regions, in preparation for the subsequent shell-based FE simulation.

5.2. Deformation analysis

The resulting models are then processed through an FEA pipeline based on a first-order linear formulation under the assumption of small deformations (see Fig. 4). In this setting, each surface mesh is modeled as an FE shell element [43], representing the medial surface of a thin shell, and is assigned the same linear-elastic material with identical

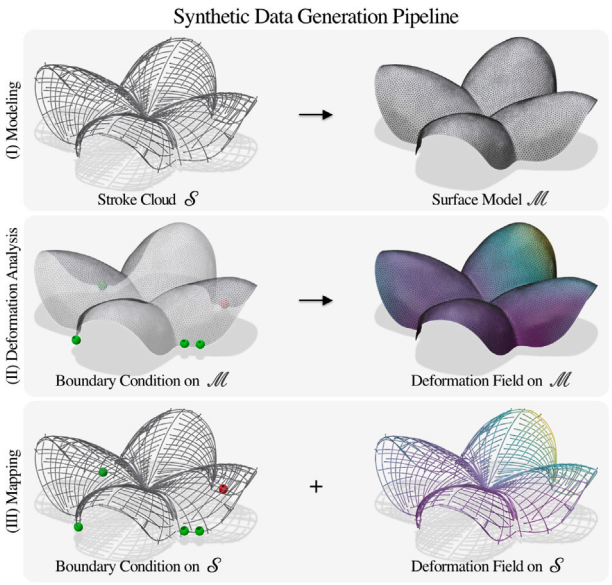

Fig. 3. Overview of the synthetic data generation pipeline for a single exemplar sketch under one boundary condition configuration. Overall, the process consists of three stages: (i) manually modeling each sketch $\mathcal{S}$ in our curated dataset into a surface model $\mathcal{M}$, (ii) performing deformation analysis under 500 different boundary conditions over the surface model, and (iii) mapping the prior stage inputs and outputs to the sketch.

material parameters across all dataset samples, namely concrete. Under these assumptions, each sketch's corresponding surface model is prescribed 500 boundary condition configurations for analyzing its deformation behavior, with configurations differing in the selection of load and support points. In each FE simulation, a vertex is randomly selected as the load point, where a constant external force of magnitude 10 kN is applied along the inward-pointing surface normal. Support points are selected from mesh vertices that lie below a prescribed height threshold relative to the sketch bounding box in order to mimic structurally realistic placements. To maintain reasonable spacing between the support points, we cluster vertices by distance using a Gaussian Mixture Model (GMM) by randomly setting the number of components to 3–8, and sampling one vertex from each cluster as a support point. Each support point is assigned its six degrees of freedom (6 DoF), with translational degrees fixed and rotational degrees left free.

5.3. Mapping and feature extraction

To enable end-to-end learning of deformation fields on 3D stroke clouds, without relying on an intermediate surface reconstruction step at inference time, we transfer both the prescribed boundary conditions and the resulting deformation fields from the surface mesh vertices to the stroke polyline vertices. To this end, we spatially align each 3D stroke cloud with its corresponding surface model and construct an input–output pair of vectors for each stroke vertex containing information transferred from nearby surface mesh vertices: an input feature vector encoding boundary condition annotations, namely whether a vertex is a load point or a support point, and a corresponding target deformation vector obtained from simulation. We additionally extract local orthonormal frame coefficients that characterize the stroke geometry and append them to the previously defined per-vertex input feature vector that encodes annotations. This process yields a large-scale dataset of 3D sketch stroke cloud topologies and per-vertex input feature vectors, coupled with corresponding ground-truth output deformation vectors. In the following, we describe this process in detail.

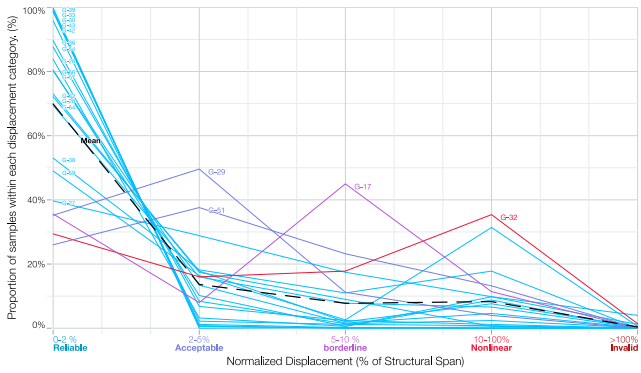

Fig. 4. Distribution of maximum displacement values in the dataset, normalized by the *structural span* and grouped into five ranges, for a subset of 20 sketches. Each colored line denotes the distribution over 500 deformation instances for one sketch, and the dark line shows the mean across sketches. Displacements above 10% of the structural span indicate cases where the small-displacement assumption breaks down, while values above 100% indicate physically implausible responses. *Structural span is defined as the longest edge of the 3D sketch's axis-aligned bounding box.

Input boundary conditions. Load and support annotations are originally defined on the vertices of the reconstructed surface meshes used for simulation. We transfer these annotations to the sketch stroke vertices via a one-to-one nearest-neighbor mapping ($k = 1$), assigning each sketch vertex the label of its closest surface-mesh vertex (see inset). We define a 2-dimensional binary feature vector for each sketch stroke vertex, whose first entry encodes load information and second entry encodes support information. Concretely, for a stroke vertex p, we set $x_p^1 = 1$ if it is the nearest neighbor of a load point and $x_p^2 = 1$ if it is the nearest neighbor of a support point; otherwise, the corresponding entries are set to 0. This yields a per-vertex 2-dimensional feature vector $\mathbf{x}_p^{\mathrm{BC}} = (x_p^1, x_p^2) \in \{0,1\}^2$.

Input orthonormal frame coefficients. The deformation field can be decomposed into a linear combination of a local orthonormal basis at each point (see inset). Therefore, the local orthonormal basis can serve as a powerful geometry-aware inductive bias that can be fed as input to regress the deformation vector at each point. To this end, given each stroke as an ordered sequence of vertices, we define at each point p an orthonormal frame $(\mathbf{t}_p, \mathbf{n}_p, \mathbf{b}_p) \in \mathbb{R}^{3\times3}$ consisting of the unit tangent vector $\mathbf{t}$, estimated from consecutive stroke vertices positions; the surface normal $\mathbf{n}$ of the underlying canvas geometry, automatically derived using KNN-based PCA normal estimation; and the tangent normal $\mathbf{b} = \mathbf{t} \times \mathbf{n}$. We then concatenate the

components of the three frame vectors, namely $\mathbf{t}_p$, $\mathbf{n}_p$, and $\mathbf{b}_p$, yielding a 9-dimensional per-vertex feature vector $\mathbf{x}_p^{\text{Basis}} = (\mathbf{t}_p, \mathbf{n}_p, \mathbf{b}_p) \in \mathbb{R}^9$.

Output deformation field. Given the physical simulation results, we obtain deformation vectors at stroke polyline vertices by interpolating the simulated displacements from neighboring mesh vertices (see inset). Specifically, for each stroke polyline vertex p, we compute its deformation vector $\mathbf{d}_p$ using Inverse-Distance Weighting (IDW) over its $k = 4$ nearest mesh vertices:

$$\mathbf{d}_p = \frac{\sum_{\mathbf{p}^{\mathcal{M}} \in \mathcal{N}_k(\mathbf{p})} w_{p^{\mathcal{M}}} \mathbf{d}_{p^{\mathcal{M}}}}{\sum_{\mathbf{p}^{\mathcal{M}} \in \mathcal{N}_k(\mathbf{p})} w_{p^{\mathcal{M}}}},$$

where $\mathcal{N}_k(\mathbf{p})$ denotes the set of the k nearest mesh $\mathcal{M}$ vertices to p, $w_{p^{\mathcal{M}}} = 1/(\|p - p^{\mathcal{M}}\| + \epsilon)$, and ϵ is a small constant for numerical stability. This process yields per-vertex ground-truth output 3D deformation vectors $\mathbf{d}_p$.

6. Method

To enable learning from the inherently irregular and unstructured nature of 3D sketches, we encode both the sketch topology and its per-vertex vectors using the *f*VDB sparse voxel data structure [21]. This representation provides a structured, efficient encoding that compactly captures the spatial occupancy of the stroke cloud and its associated attributes, making it well-suited for processing within neural network architectures and their constituent layers. In particular, it enables the use of *f*VDB's sparse convolutional operators, which operate on locally densified voxel neighborhoods without incurring the cubic scaling and memory overhead of dense 3D tensors. Moreover, *f*VDB supports differentiable feature splatting and sampling between points and voxels, allowing direct supervision on the attributes defined on stroke vertices.

6.1. 3D sparse voxel grid representation

We denote a stroke cloud as a set of stroke polylines $\mathcal{S} = \{S_i\}_{i=1}^{M}$, where each stroke polyline $S_i = \{p_{i,j}\}_{j=1}^{N_i} \subset \mathbb{R}^3$ is an ordered sequence of vertices in 3D, and M and N_i denote the total number of strokes and the number of points in the i-th stroke, respectively. The union of all polyline points forms the stroke point cloud $\mathcal{P} = \bigcup_{i=1}^{M} S_i = \{p_{i,j}\}_{j=1}^{N_i} \subset \mathbb{R}^3$.

By design, *f*VDB compactly encodes spatial topology and numerical features in an intertwined tree structure. We leverage this representation to construct a sparse voxel grid from the 3D stroke cloud and associate each stroke vertex with an 11-dimensional input feature vector $\mathbf{x}_p = (\mathbf{x}_p^{\text{BC}}, \mathbf{x}_p^{\text{Basis}})$, formed by concatenating its boundary condition features and flattened orthonormal-frame components. We also associate each vertex with a target 3D deformation vector $\mathbf{d}_p$ as the output. Overall, this yields pairs $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X} = (\mathcal{S}, \{\mathbf{x}_p\}_{p\in\mathcal{P}})$ contains the stroke cloud topology with its input per-vertex features, and the output $\mathcal{Y} = \{\mathbf{d}_p\}_{p\in\mathcal{P}}$ contains per-vertex deformation vectors.

Training is enabled by *f*VDB's differentiable splatting and sampling operators, which allow information to flow between points in the 3D stroke cloud and voxels in the sparse grid. Concretely, during training, per-point input features and target deformation vectors are splatted onto the sparse voxel grid via trilinear interpolation, aggregating point-wise attributes into voxel-wise feature tensors. Conversely, the network's per-voxel output predictions are recovered at the original sketch points via trilinear sampling, enabling the loss function to be defined directly on the stroke vertices rather than on voxels.

6.2. Network architecture and physics-informed losses

Our network builds upon a Res-UNet architecture and extends it to a 3D Sparse Res-UNet with a decoupled dual-head output layer that stabilizes training and enables geometry-aware supervision. Specifically, we employ a 3D Sparse Res-UNet with four downsampling and four upsampling stages, each composed of stacked residual blocks and connected via skip connections. Standard dense 3D convolutional layers are replaced with sparse counterparts to operate efficiently on the sparse voxel representation. In addition, we decompose each deformation vector $\mathbf{d}_p$ into a unit-length displacement vector $\mathbf{e}_p = \mathbf{d}_p/\|\mathbf{d}_p\|$ and a scalar displacement magnitude $\|\mathbf{d}_p\|$. Accordingly, we adopt a dual-head architecture with a shared backbone where one head regresses the angular unit-length displacement vectors $\mathbf{e}_p$, while the other regresses the log-scaled displacement magnitudes $\log_2(\|\mathbf{d}_p\| + \epsilon)$ (see Fig. 5). During training, we further exploit the physical decomposition of the unit-length displacement vectors into in-plane (stretching) and out-of-plane (bending) components under the small-deformation assumption [44,45], and introduce physics-guided regularization terms. During inference, the predicted unit-length vectors are combined with the predicted scalar magnitudes and subsequently mapped back to the original scale via the inverse transformation $\exp_2(\cdot)$ to construct the full deformation vectors.

Angular loss. Stretching deformation of a shell element is caused by local x and y (i.e., in-plane) translations of its nodes (see Fig. 6(a)), whereas its bending deformation is caused by local z translations (i.e., out-of-plane) and local x and y rotations of its nodes (see Fig. 6(b)). In order to measure the angular difference between the predicted and ground-truth displacement vectors, we leverage the per-point orthonormal frames and devise an angular loss consisting of two complementary terms, namely a sign-invariant tangential alignment loss $\mathcal{L}_{\text{ang},t}$ and a sign-aware normal alignment loss $\mathcal{L}_{\text{ang},n}$.

Let $\hat{\mathbf{e}}_p$ and $\mathbf{e}_p$ denote the predicted and ground-truth unit-length displacement vectors at point p, respectively, and let $(\mathbf{t}_p, \mathbf{n}_p, \mathbf{b}_p)$ denote its associated local orthonormal frame components. We define the angular loss as the combination of the following two terms:

$$\mathcal{L}_{\text{ang},t} = \sum_p m_p \left(1 - |\hat{\mathbf{e}}_p^{\parallel} \cdot \mathbf{e}_p^{\parallel}|\right), \tag{1}$$

$$\mathcal{L}_{\text{ang},n} = \sum_p m_p \, |\hat{\mathbf{e}}_p \cdot \mathbf{n}_p - \mathbf{e}_p \cdot \mathbf{n}_p|, \tag{2}$$

where $\hat{\mathbf{e}}_p^{\parallel} = (\hat{\mathbf{e}}_p - (\hat{\mathbf{e}}_p \cdot \mathbf{n}_p)\,\mathbf{n}_p) / \left\|\hat{\mathbf{e}}_p - (\hat{\mathbf{e}}_p \cdot \mathbf{n}_p)\,\mathbf{n}_p\right\|$ denotes the normalized projection of the predicted displacement vector onto the local tangent plane spanned by $(\mathbf{t}_p, \mathbf{b}_p)$. The projected ground-truth vector $\mathbf{e}_p^{\parallel}$ is defined analogously. Moreover, $m_p = \mathbb{I}(\|\mathbf{d}_p\| > \tau)$ is a binary mask that suppresses points with negligible displacement magnitudes, where τ is set adaptively to the 10% quantile of displacement magnitudes.

Magnitude loss. For the displacement magnitudes, we define our regression loss as the Mean Squared Error (MSE) between predicted and ground-truth displacement magnitudes at each stroke vertex. Formally, the magnitude loss is given by

$$\mathcal{L}_{\text{mag}} = \frac{1}{|\mathcal{P}|} \sum_{p\in\mathcal{P}} \left(\log_2\left(\|\hat{\mathbf{d}}_p\| + \epsilon\right) - \log_2\left(\|\mathbf{d}_p\| + \epsilon\right)\right)^2. \tag{3}$$

Physics-guided regularization. A 3D stroke cloud can be considered as a family of curves lying on an ambient user-intended surface. Denoting the aforementioned surface as $\sigma(u, v)$, deformation can be viewed as bending and stretching $\sigma(u, v)$ into a new surface $\sigma'(u, v)$ whose geometric properties may differ from those of the original surface. These differences are encoded in their fundamental forms. When studying or applying the deformation concept, it is convenient to consider the associated displacement vectors $\mathbf{d}(u, v) := \sigma'(u, v) - \sigma(u, v)$ along $\sigma(u, v)$. This is due to the fact that the differential geometric analysis of deformations is naturally phrased in terms of such vector fields and a broad set of

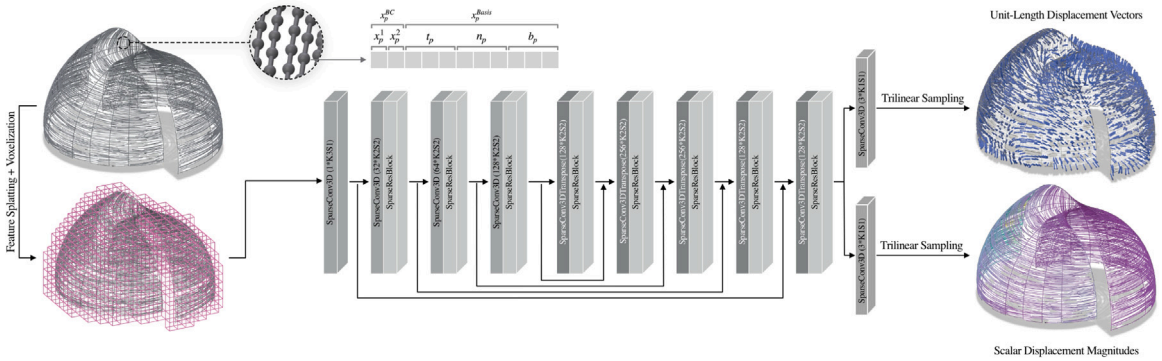

Fig. 5. Strokes2Deform's Network Architecture Overview. Input sketch stroke clouds, together with their 11-dimensional per-point features, are encoded into a sparse voxel grid and processed by a four-level encoder–decoder with residual blocks and skip connections. All operations are performed using sparse 3D convolutions. A shared backbone branches into two output heads that separately predict unit-length displacement vectors and scalar displacement magnitudes, which are recombined during inference to recover the full deformation field on the input sketch.

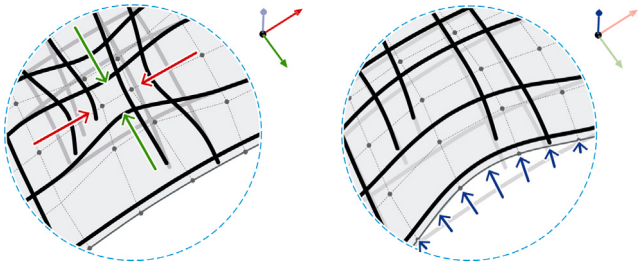

Fig. 6. The two modes of deformation: (a) stretching (in-plane) and (b) bending (out-of-plane), visualized on 3D strokes (shown in black) representing an underlying surface (shown in gray). Stretching and bending deformations of a shell element are caused by local x and y (i.e., in-plane) and by local z (i.e., out-of-plane) translations, respectively.

standard tools is available in this setting. Therefore, by resorting to one of these tools proposed by [46], we obtain the following *thin-shell energy* formulation:

$$E(\mathbf{d}) = \iint_{\Omega} k_s \left(\|\mathbf{d}_u\|^2 + \|\mathbf{d}_v\|^2 \right) + k_b \left(\|\mathbf{d}_{uu}\|^2 + 2\|\mathbf{d}_{uv}\|^2 + \|\mathbf{d}_{vv}\|^2 \right) du\, dv, \tag{4}$$

where k_s and k_b are material-dependent stiffness parameters that control resistance to stretching and bending, respectively.

In our setting, however, the sketch samples are organized as vertices *along each curve*, not as a 2D grid on the surface. Accordingly, we interpret v as the parameter that runs along a curve, and u as the parameter that indexes (switches between) different curves in the family. Since there are only finitely many curves in the family, we replace u with the discrete index i and simplify Eq. (4) to

$$E(\mathbf{d}) = \sum_i \int_{I_i} k_s \|\mathbf{d}_v\|^2 + k_b \|\mathbf{d}_{vv}\|^2 \, dv, \tag{5}$$

In order to employ Eq. (5) in practice, one needs a discrete analogue of it. Following this interpretation, let $\mathbf{d}_{i,j}$ be the displacement vector at the vertex j of the i-th curve. Writing $\mathbf{d}_v(u,v) \approx \mathbf{d}_{i,j+1} - \mathbf{d}_{i,j}$ and $\mathbf{d}_{vv}(u,v) \approx \mathbf{d}_{i,j+1} - 2\mathbf{d}_{i,j} + \mathbf{d}_{i,j-1}$ in Eq. (5), gives

$$E(\mathbf{d}) = \underbrace{k_s \sum_{i=1}^{M} \sum_{j=1}^{N_i - 1} \left\| \mathbf{d}_{i,j+1} - \mathbf{d}_{i,j} \right\|^2}_{\mathcal{L}_s} +$$

$$\underbrace{k_b \sum_{i=1}^{M} \sum_{j=2}^{N_i - 1} \left\| \mathbf{d}_{i,j+1} - 2\mathbf{d}_{i,j} + \mathbf{d}_{i,j-1} \right\|^2}_{\mathcal{L}_b}, \tag{6}$$

where the index j traverses the vertices of each sketch curve, while the index i enumerates the curves in the family. The integers M and N_i denote the total number of strokes in a sketch and the number of points in the i-th stroke, respectively. The norm of the first-order finite difference $\mathcal{L}_s$ approximates the change of the displacement field along a stroke and penalizes in-plane stretching, while the norm of the second-order finite difference $\mathcal{L}_b$ approximates the curvature variation and penalizes out-of-plane bending. We leverage this formulation to define physics-informed regularization terms that discretely approximate thin-shell stretching and bending energies on sketch stroke clouds.

Therefore, in light of Eqs. (1), (2), (3), and (6), we propose the following objective function:

$$\mathcal{L} = \lambda_{\text{ang}}(\mathcal{L}_{\text{ang},n} + \mathcal{L}_{\text{ang},t}) + \lambda_{\text{mag}}\mathcal{L}_{\text{mag}} + \lambda_s\mathcal{L}_s + \lambda_b\mathcal{L}_b. \tag{7}$$

6.3. Implementation details

We implement *Strokes2Deform* using the *f*VDB framework built on top of PyTorch, and train and evaluate our model on a machine equipped with an NVIDIA A100 GPU, using a batch size of 16. We use Adam [47] with a learning rate of 2×10^{-4}. We normalize each sketch to the unit cube and construct a sparse voxel grid from the stroke polyline vertices as-is, using a voxel size of 0.05. To improve robustness to varying stroke densities and reduce overfitting, we randomly subsample between 70% and 85% of the strokes per sketch during training. While some sketching interfaces provide surface normals derived from proxy geometries, we assume normals are unavailable and estimate them using PCA-based normal estimation. Alternatively, normals can be derived using methods such as [48], although this may be computationally expensive in interactive settings. Since our synthetic data is generated for a single material, namely concrete, we fix the stiffness parameters k_s and k_b during training runs.

6.4. Training details and ablations

We train and evaluate our model on a curated dataset of 40K *sketch–deformation* pairs. The dataset contains a set of unique sketches, each coupled with 500 deformation instances obtained from FE simulations and mapped to its constituent stroke vertices. Based on this dataset, we train and test our model under two complementary split strategies: one evaluates generalization to out-of-distribution sketch instances, referred to as *Topology Split*, and the other tests generalization to out-of-distribution boundary conditions, referred to as *Boundary-Condition Split*. In both cases, we use an 8:1:1 train/validation/test split ratio:

Table 1
Ablation of the stretching and bending regularization weights λ_s and λ_b. MAE measures the average displacement-magnitude error, EMD captures the distributional discrepancy between predicted and reference displacement magnitudes, and COS measures the average angular deviation of predicted displacement vectors from the FE-based reference results mapped to strokes.

Variant	Weights $(\lambda_{ang}, \lambda_{mag}, \lambda_s, \lambda_b)$	Topology Split			Boundary-Condition Split		
		MAE	EMD	COS	MAE	EMD	COS
w/o physics	$(100, 10, 0, 0)$	0.0402	0.0381	0.1076	0.0258	0.0227	0.0766
w/ stretching only	$(100, 10, 10, 0)$	0.0381	**0.0358**	0.1158	0.0241	0.0209	0.0634
w/ bending only	$(100, 10, 0, 10^{-5})$	0.0392	0.0366	0.1097	**0.0229**	**0.0196**	0.0672
w/ stretching and bending	$(100, 10, 10, 10^{-5})$	**0.0377**	0.0361	**0.0987**	0.0272	0.0242	**0.0580**

- **Topology Split.** We split the dataset at the *sketch* level: 80% of the sketches, together with all their corresponding deformation instances, are used for training, 10% for validation, and 10% for testing. This setting evaluates generalization to out-of-distribution sketch topologies and, naturally, to unseen boundary conditions defined on those unseen sketches.
- **Boundary-Condition Split.** We split the dataset at the *deformation* level: for each sketch instance, 80% of its deformation instances are used for training, 10% for validation, and 10% for testing. In this setting, the model observes every sketch during training, but only under a subset of boundary condition configurations. This setting evaluates generalization to previously seen sketch geometries under unseen boundary condition configurations.

Given these split settings, we ablate the role of the physics-guided regularization by comparing four loss weighting variants defined by the stretching and bending weights λ_s and λ_b: *without physics* ($\lambda_s = 0$, $\lambda_b = 0$), *with stretching only* ($\lambda_s = 10$, $\lambda_b = 0$), *with bending only* ($\lambda_s = 0$, $\lambda_b = 10^{-5}$), and *with stretching and bending* ($\lambda_s = 10$, $\lambda_b = 10^{-5}$). This allows us to analyze the individual contributions of the two regularization terms, as well as their combined effect.

We adopt three metrics to evaluate performance of our model against the ground-truth deformation fields obtained from the finite element method and mapped onto the stroke vertices. We use cosine distance (COS) for measuring the angular deviation between predicted and ground-truth unit-length displacement vectors, and we report Mean Absolute Error (MAE) and Earth Mover's Distance (EMD) to measure the deviation between predicted and ground-truth scalar displacement magnitudes. The ground-truth deformation fields obtained from FE simulations are originally defined on the surface mesh vertices. However, since the stroke cloud and its corresponding surface model differ in both the number and positions of their vertices, we use the corresponding deformation vectors mapped onto the stroke cloud vertices for evaluations. Therefore, the reported values essentially reflect how closely our predictions align with the FE-derived reference results. Our results show nearly similar prediction performance under both split settings, indicating that our method generalizes to out-of-distribution sketches as well as unseen boundary configurations. Moreover, the similar trends observed in the presence of the physics-guided stretching and bending regularization highlight their role in predicting physically more accurate and coherent predictions in alignment with real FE simulations, as reflected by the improvements in the reported error metrics across both splits (see Table 1). Based on the ablations, we set $\lambda_{ang} = 100$, $\lambda_{mag} = 10$, $\lambda_s = 10$, and $\lambda_b = 10^{-5}$ for the best results.

7. Results and validation

In this section, we evaluate the performance of *Strokes2Deform* both qualitatively and quantitatively. Specifically, we present results on a diverse set of 3D sketches of architectural thin-shell structures spanning a range of scales and stroke complexities (see Fig. 7). We also report the model's runtime and assess its robustness to sketches with different stroke density/sparsity ratios (see Table 2). The qualitative results further support the quantitative evaluation reported in Table 1, demonstrating the fidelity of our predictions against numerically solved FE simulations. Moreover, the model demonstrates strong robustness to stroke sparsity: retaining only 70–85% of the input strokes results in negligible accuracy degradation while reducing inference time.

7.1. Comparison to reconstruction-and-simulation

Our primary objective is to bypass the conventional reconstruction-and-simulation pipeline by predicting deformation directly onto sketched 3D strokes. For comparison, we process the same 3D stroke clouds using a state-of-the-art stroke cloud surfacing method [1] to reconstruct a piecewise-smooth surface mesh, followed by finite element analysis on the reconstructed geometry (see Fig. 8). As the comparisons show, surface reconstruction approaches such as [1] typically rely on the presence of boundary strokes; missing or ambiguously drawn boundaries can lead to poor reconstructions that propagate errors into subsequent simulations, whereas our method avoids expensive reconstruction and simulation operations, and results in faster runtime performance while enabling intuitive refinement through deformation visualization directly on the sketched strokes.

7.2. Single- versus multiple-load case

In elasticity, the *Principle of Superposition* states that *"the total displacement, stress, or strain at any point in a structure due to multiple loads is the sum of the effects caused by each load acting individually"*. These linear-elastic and small-deformation assumptions hold in our setting. We therefore handle multi-load cases by running the network independently for each load while keeping the support points fixed, yielding one deformation vector per load. We then obtain the final response by vector addition of the predicted vectors at each stroke polyline vertex (see Figs. 9 and 10). Note that in our experiments, the number of support points ranges from 3 to 8, however, the model is shown to be able to handle an arbitrary number of supports at inference time, thanks to the learned effect of supports during training.

7.3. Deformation interpolation

Beyond simple colorings of displacement magnitudes on the stroke cloud, a key advantage of predicting the full deformation field is that it naturally enables intuitive animation and visualization of the structural response directly in the sketch domain (see Supplementary Material for the demo video). Given the predicted unit-length displacement vectors and magnitudes at individual vertices, we reconstruct the full deformation vector at each stroke vertex and then linearly interpolate vertex positions between the undeformed and deformed states. This produces a smooth deformation sequence that visually communicates the sketch's structural behavior (see Fig. 11).

8. Conclusion

We presented **Strokes2Deform**, a physics-informed end-to-end model that predicts deformation fields directly on sparse 3D stroke clouds under user-specified load and support conditions, eliminating the need for surface reconstruction and explicit simulation during early-stage sketching. Our approach is enabled by (i) a large-scale synthetic dataset of sketch–deformation pairs, (ii) an *f*VDB-based sparse voxel representation with differentiable point–voxel splatting and sampling operations, (iii) a dual-head architecture that decouples angular and magnitude regression of deformation vectors, and (iv) physics-guided

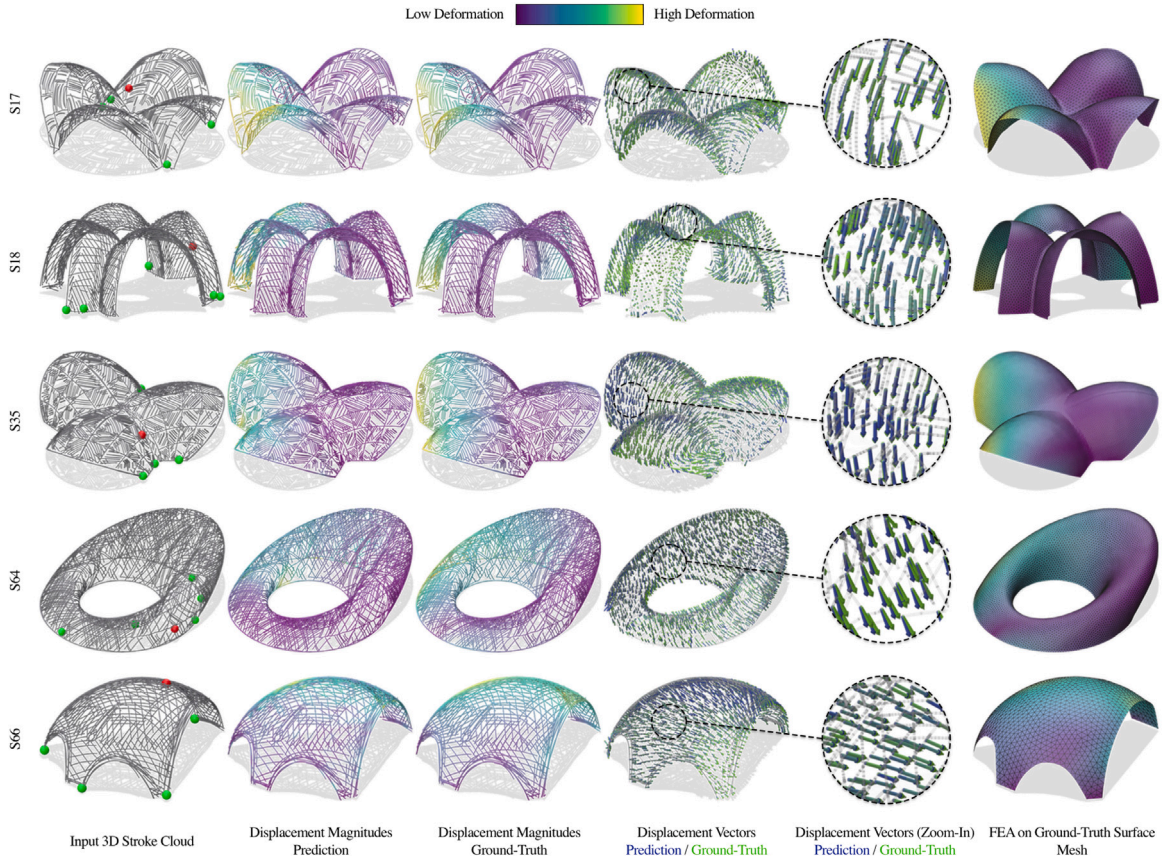

Fig. 7. Qualitative results showcasing predicted and ground-truth displacement magnitudes and vectors, and reference FE simulations on the underlying surface mesh. Close-ups highlight the alignment of predicted and ground-truth unit-length displacement vectors.

Table 2
Accuracy-runtime trade-off across different stroke sampling ratios for the sketches shown in Fig. 7. Runtime (s) is computed as the full inference time. This includes all steps required to produce the final prediction: computing per-vertex features, splatting these features onto the voxel grid, the network forward pass, and sampling the predicted deformation field back onto the stroke polyline vertices.

ID	Scale[a]	Strokes	Stroke sampling ratio								
			1.00			0.85			0.70		
			MAE	COS	Runtime	MAE	COS	Runtime	MAE	COS	Runtime
*S*17	27.95	765	0.0229	0.0482	2.18	0.0214	0.0402	1.94	0.0243	0.0595	1.73
*S*18	10.27	1060	0.0042	0.1412	1.70	0.0056	0.1387	1.12	0.0059	0.1483	1.06
*S*35	40.87	2327	0.0388	0.0875	2.50	0.0387	0.0896	2.12	0.0559	0.0957	1.89
*S*64	41.28	872	0.1333	0.0036	1.44	0.1394	0.0034	1.28	0.2355	0.0096	1.01
*S*66	18.67	443	0.0123	0.0787	1.70	0.0176	0.1544	1.66	0.0190	0.1405	1.37

[a] Scale (m) is computed as the diameter of the sketch's axis-aligned bounding box.

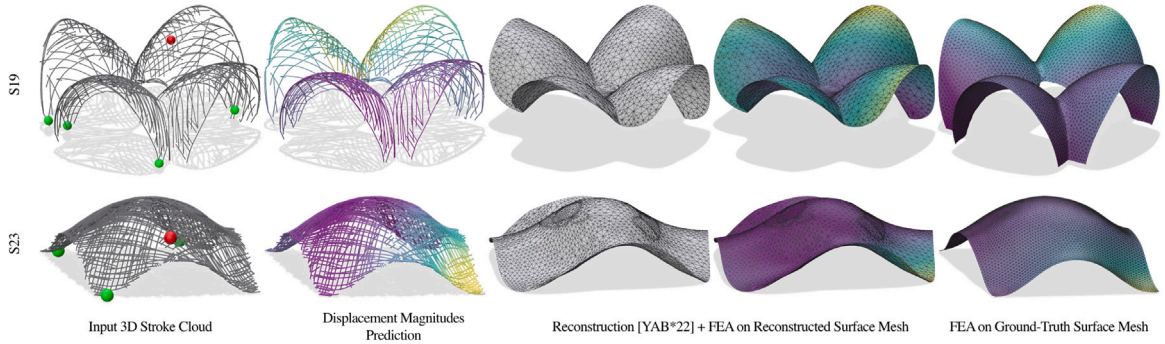

Fig. 8. Comparison to Reconstruction-and-Simulation. For *S*19, our predicted maximum displacement (0.24) closely matches the ground truth (0.26), while the reconstruction-based pipeline [1] predicts a maximum displacement of (> 10) due to poor reconstruction. Similarly, for *S*23, our prediction (0.20) remains close to the ground truth (0.19), whereas the reconstruction-based approach overestimates the maximum deformation magnitude (0.28).

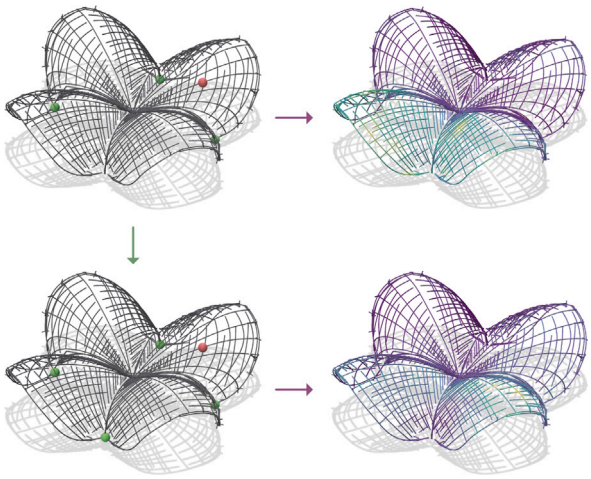

Fig. 9. Single-Load Case example interaction experiment. Given a single user-specified load point, this experiment can be understood as querying how Strokes2Deform responds to the number and locations of different support points. The designer first annotates three support points (top row) and then adds a fourth support point to obtain updated deformation results (bottom row).

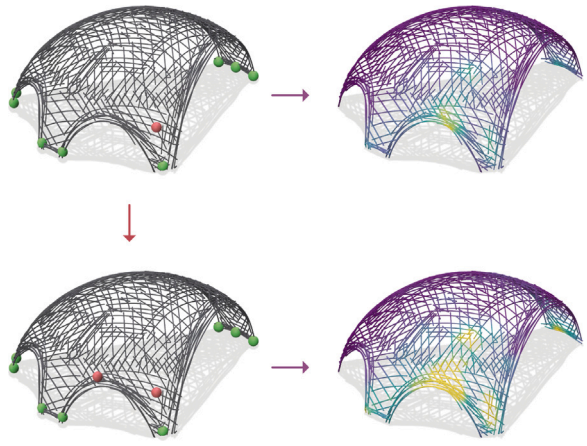

Fig. 10. Multi-Load Case example interaction illustrating the application of Superposition Principle. Although Strokes2Deform is trained on single-load cases, it generalizes to multiple load scenarios. The predicted deformation for a single load point (top row) is compared to the response under two (or more) simultaneously applied loads (bottom row).

regularization loss terms inspired by stretching and bending energies defined on 3D stroke polylines. Strokes2Deform is compatible with sketches originating from a variety of input sources as it operates directly on raw 3D curves and strokes. Our experiments demonstrate accurate and robust deformation prediction across a diverse set of sketches, fast runtime, and generalization across unseen 3D sketch topologies and boundary condition configurations.

Limitations and future work. Our method currently assumes fixed load magnitudes, material properties, and thickness across the user-intended surface. Future work could extend the framework to a broader range of materials, for example by introducing an additional network branch that learns material-specific behavior and propagates this information through the network layers, as well as by incorporating richer boundary conditions. Finally, a natural extension of our model is physics-informed stress prediction and, building on that, learning-based topology optimization driven by predicted deformations and stresses to guide designers by suggesting additional strokes that promote aesthetically coherent yet structurally plausible solutions.

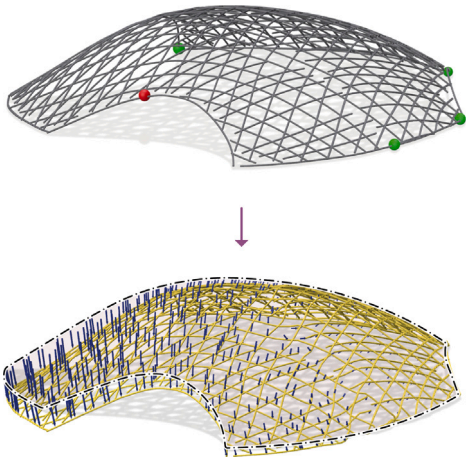

Fig. 11. Deformation Interpolation experiment. Given a 3D sketch and user-specified boundary conditions (gray strokes, top), we can compute and visualize the deformed state of the stroke cloud by leveraging the predicted per-vertex deformation vectors (yellow strokes, bottom).

CRediT authorship contribution statement

Shervin Rasoulzadeh: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Raman Suliman:** Writing – review & editing, Writing – original draft, Visualization, Methodology, Data curation, Conceptualization. **Arvin Rasoulzadeh:** Writing – original draft, Methodology, Conceptualization. **Iva Kovacic:** Writing – review & editing, Conceptualization, Supervision, Project administration, Funding acquisition. **Michael Wimmer:** Writing – review & editing, Conceptualization, Supervision, Project administration, Funding acquisition.

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.

Acknowledgment

This research was funded in whole or in part by the Austrian Science Fund (FWF) [10.55776/F77] through the SFB “Advanced Computational Design” (subproject 2). The third author, A. Rasoulzadeh (METROM GmbH), was financed by the European Union, Sächsische Aufbaubank – Förderbank (application number 100750317). The authors thank Cedric Kornacki for his support in developing the demo application and Xu Jie Wang for her support in preparing the demo video.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2026.104629.

Data availability

Our code and dataset are available at: https://gitlab.cg.tuwien.ac.at/srasoulzadeh/strokes2deform.git.

References

[1] Yu E, Arora R, Baerentzen JA, Singh K, Bousseau A. Piecewise-smooth surface fitting onto unstructured 3d sketches. ACM Transactions on Graphics (TOG) 2022;41(4).

[2] Dorsey J, Xu S, Smedresman G, Rushmeier H, McMillan L. The mental canvas: a tool for conceptual architectural design and analysis. In: 15th Pacific Conference on Computer Graphics and Applications (PG'07). IEEE; 2007.

[3] Kim Y, Hong K, Yang J. Feather: 3d sketchbook light as a feather. In: ACM SIGGRAPH 2023 Appy Hour. 2023.

[4] Kovacs BI, Erb I, Kaufmann H, Ferschin P. Mr. sketch. immediate 3d sketching via mixed reality drawing canvases. In: 2023 IEEE International Symposium on Mixed and Augmented Reality (ISMAR). IEEE; 2023.

[5] Wacker P, Arora R, Machuca MDB, Keefe D, Israel JH. 3D sketching application scenarios. In: Interactive Sketch-based Interfaces and Modelling for Design. River Publishers; 2023.

[6] Neveu W, Puhachov I, Thomaszewski B, Bessmeltsev M. Stability-aware simplification of curve networks. In: ACM SIGGRAPH 2022 Conference Proceedings. 2022.

[7] Hähnlein F. Binary optimization for the analysis and synthesis of concept sketches. 2022.

[8] Xin M, Sharlin E, Sousa MC. Napkin sketch: handheld mixed reality 3D sketching. In: Proceedings of the 2008 ACM symposium on Virtual reality software and technology. 2008.

[9] Yu X, DiVerdi S, Sharma A, Gingold Y. Scaffoldsketch: accurate industrial design drawing in vr. In: The 34th Annual ACM Symposium on User Interface Software and Technology. 2021.

[10] Huang Z, Carr N, Ju T. Variational implicit point set surfaces. ACM Transactions on Graphics (TOG) 2019;38(4).

[11] Rosales E, Rodriguez J, Sheffer A. Surfacebrush: from virtual reality drawings to manifold surfaces. ACM Transactions on Graphics (TOG) 2019;38(4).

[12] Luo L, Chowdhury PN, Xiang T, Song Y-Z, Gryaditskaya Y. 3D vr sketch guided 3d shape prototyping and exploration. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023.

[13] Sureshkumar A, Parakkat AD, Bonneau G-P, Hahmann S, Cani M-P. Vrsurf: surface creation from sparse, unoriented 3d strokes. In: Computer Graphics Forum. Wiley Online Library; 2025.

[14] Sureshkumar A, Parakkat AD, Bonneau G-P, Hahmann S, Cani M-P. Ribbonsculpt: voronoi ball based 3d sculpting from sparse vr ribbons. In: Proceedings of the SIGGRAPH Asia 2025 Conference Papers. 2025.

[15] Yu E, Arora R, Stanko T, Bæ rentzen JA, Singh K, Bousseau A. Cassie: curve and surface sketching in immersive environments. In: Proceedings of the 2021 CHI conference on human factors in computing systems. 2021.

[16] Rasoulzadeh S, Wimmer M, Stauss P, Kovacic I. Strokes2surface: recovering curve networks from 4d architectural design sketches. In: Computer Graphics Forum. 43, (2). Wiley Online Library; 2024.

[17] Mohammadi N, Wang J, Cao Y, Setareh M. Smats: sketch-based modeling and analysis of truss systems. In: ARCC Conference Repository. 2013.

[18] Yu D, Xiao C, Lau M, Fu H. Sketch2stress: sketching with structural stress awareness. IEEE Transactions on Visualization and Computer Graphics 2023;30(10).

[19] Rasoulzadeh S, Senk V, Königsberger M, Reisinger J, Kovacic I, Füssl J, Wimmer M. A novel integrative design framework combining 4d sketching, geometry reconstruction, micromechanics material modelling, and structural analysis. Advanced Engineering Informatics 2023;57.

[20] Ampanavos S, Nourbakhsh M, Cheng C-Y. Structural design recommendations in the early design phase using machine learning. In: International Conference on Computer-Aided Architectural Design Futures. Springer; 2021.

[21] Williams F, Huang J, Swartz J, Klar G, Thakkar V, Cong M, Ren X, Li R, Fuji-Tsang C, Fidler S, et al. Fvdb: a deep-learning framework for sparse, large scale, and high performance spatial intelligence. ACM Transactions on Graphics (TOG) 2024;43(4).

[22] Arora R, Singh K. Mid-air drawing of curves on 3d surfaces in virtual reality. ACM Transactions on Graphics (TOG) 2021;40(3).

[23] Blender Foundation. Blender. 2025, https://www.blender.org.

[24] McNeel & Associates. Rhinoceros 3d. 2025, https://www.rhino3d.com.

[25] Abbasinejad F, Joshi P, Amenta N. Surface patches from unorganized space curves. In: Proceedings of the twenty-eighth annual symposium on Computational geometry. 2012.

[26] Pan H, Liu Y, Sheffer A, Vining N, Li C-J, Wang W. Flow aligned surfacing of curve networks. ACM Transactions on Graphics (TOG) 2015;34(4).

[27] Orbay G, Kara LB. Sketch-based surface design using malleable curve networks. Computers & Graphics 2012;36(8).

[28] Zhuang Y, Zou M, Carr N, Ju T. A general and efficient method for finding cycles in 3d curve networks. ACM Transactions on Graphics (TOG) 2013;32(6).

[29] Zou M, Ju T, Carr N. An algorithm for triangulating multiple 3d polygons. In: Computer graphics forum. 32, (5). Wiley Online Library; 2013.

[30] Stanko T, Hahmann S, Bonneau G-P, Saguin-Sprynski N. Smooth interpolation of curve networks with surface normals. In: Eurographics 2016 Short Papers. Eurographics Association; 2016.

[31] Stanko T, Hahmann S, Bonneau G-P, Saguin-Sprynski N. Surfacing curve networks with normal control. Computers & Graphics 2016;60.

[32] Batuhan Arisoy E, Orbay G, Burak Kara L. Free form surface skinning of 3d curve clouds for conceptual shape design. Journal of computing and information science in engineering 2012;12(3).

[33] Zhang Y, Wang S, Bessmeltsev M. Variational neural surfacing of 3d sketches. In: Proceedings of the SIGGRAPH Asia 2025 Conference Papers. 2025.

[34] Chen T, Ding C, Zhang S, Yu C, Zang Y, Li Z, Peng S, Sun L. Rapid 3d model generation with intuitive 3d input. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2024.

[35] Berger M, Tagliasacchi A, Seversky LM, Alliez P, Guennebaud G, Levine JA, Sharf A, Silva CT. A survey of surface reconstruction from point clouds. In: Computer graphics forum. 36, (1). Wiley Online Library; 2017.

[36] Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Surface reconstruction from unorganized points. In: Proceedings of the 19th annual conference on computer graphics and interactive techniques. 1992.

[37] Kazhdan M, Bolitho M, Hoppe H. Poisson surface reconstruction. In: Proceedings of the fourth Eurographics symposium on Geometry processing. 7, (4). 2006.

[38] Peschel JM, Hammond TA. STRAT: a sketched-truss recognition and analysis tool. In: DMS. 2008.

[39] Senk V, Ghazanfari M, Rasoulzadeh S, Vasylevska K, Kovacs BI, Mortezapoor S, Vonach E, Kovacic I, Füssl J, Kaufmann H, et al. Hexa: haptic-enhanced extended reality framework for material-informed architectural design. Journal of Building Engineering 2025.

[40] Bolandi H, Sreekumar G, Li X, Lajnef N, Boddeti VN. Physics informed neural network for dynamic stress prediction. Applied Intelligence 2023;53(22).

[41] Jiang H, Hsu H-Y, Zhang K, Yu H-N, Wang S, Li Y. Phystwin: physics-informed reconstruction and simulation of deformable objects from videos. arXiv preprint arXiv:2503.17973 2025.

[42] Martinez T, Di Carlo D, Nugraha AA, Fontaine M, Yoshii K. Physics-informed learning of neural scattering fields towards measurement-free mesh-to-hrtf estimation. In: ICASSP 2026. 2026.

[43] Preisinger C. Linking structure and parametric geometry. Architectural Design 2013;83(2).

[44] Wang C, Reddy JN, Lee K. Shear deformable beams and plates: relationships with classical solutions. Elsevier; 2000.

[45] Botsch M, Kobbelt L, Pauly M, Alliez P, Lévy B. Polygon mesh processing. CRC press; 2010.

[46] Celniker G, Gossard D. Deformable curve and surface finite-elements for free-form shape design. In: Proceedings of the 18th annual conference on Computer graphics and interactive techniques. 1991.

[47] Kingma DP. Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 2014.

[48] Xu R, Dou Z, Wang N, Xin S, Chen S, Jiang M, Guo X, Wang W, Tu C. Globally consistent normal orientation for point clouds by regularizing the winding-number field. ACM Transactions on Graphics (TOG) 2023;42(4).