Shape: Geometry Understanding for CFD Pre-Processing via Graph Contrastive Embeddings

–

Abstract

Computational Fluid Dynamics (CFD) workflows spend substantial time in geometry inspection and setup before any solver iteration begins: identifying bends, junctions, symmetry, and characteristic lengths; deciding boundary regions; and estimating pressure-loss drivers. We present *Shape*, a geometry-understanding system that converts pipe-like mesh inputs into a centerline graph, segments the graph into primitives (straight, arc, junction), and computes compact embeddings with a Graph Neural Network (GATv2) trained using supervised contrastive learning [1, 2]. These embeddings, paired with curvature- and alignment-based heuristics, enable an agent interface (powered by Gemini 3 [3, 4]) to answer topology questions and highlight detected segments in a 3D viewer. We validate on a synthetic dataset of 3D centerline graphs [5] and discuss limitations and paths toward a foundation model for engineering geometry understanding.

Live demo: https://shape-frontend.vercel.app/ **Code:** https://github.com/tihiera/shape **Dataset:** https://huggingface.co/datasets/bayang/shape
Video demo: https://youtu.be/ECAE9blk_NQ

1 Motivation

CFD engineers frequently lose hours (or days) on geometry *pre-processing*: (i) sanity-checking topology (e.g., unexpected junctions), (ii) measuring bend angles and radii, (iii) deciding symmetry planes or simplifying assumptions, and (iv) mapping regions for meshing and boundary conditions. These choices directly impact simulation cost (mesh size, time step stability) and correctness. *Shape* targets this bottleneck by making geometry queryable: *"How many bends above* $30°$*? Where are the junctions? Is this pipe symmetric?"* The end goal is an assistant that understands geometry before the expensive CFD solve begins.

2 System Overview

Figure 1 summarises the pipeline.

Centerline extraction. For tubular meshes, a robust approach is to compute a centerline from the surface geometry (e.g., VMTK-style centerline pipelines) [6, 7]. For this prototype, mesh inputs (e.g., `.msh`) are accepted and converted to a centerline graph, then normalised.

Segmentation. We detect junction candidates by node degree (≥ 3) on the centerline graph, then extract branches between junctions. Each branch is resampled to uniform spacing and partitioned into straight vs arc using curvature and alignment statistics (Section 4).

Embedding and inference. Each segment is downsampled (e.g., to 16 nodes) and embedded by a GATv2 encoder [2]. Embeddings support retrieval/classification and, for arcs, angle approximation via nearest-neighbour retrieval within the arc manifold.

Agent and UI. A Gemini 3-powered agent consumes structured outputs (segment list, angles, lengths, junction graph) and answers user questions [3]. The UI maps segment IDs back to node IDs for 3D highlighting and interactive inspection.

Ingest (mesh) → **Centerline** (graph) → **Segment** (junction/straight/arc) → **Resample** (uniform arc-length) → **Embed** (GATv2 + SupCon) → **Agent** (Gemini 3) + **3D Highlighting**

Figure 1: End-to-end geometry understanding pipeline.

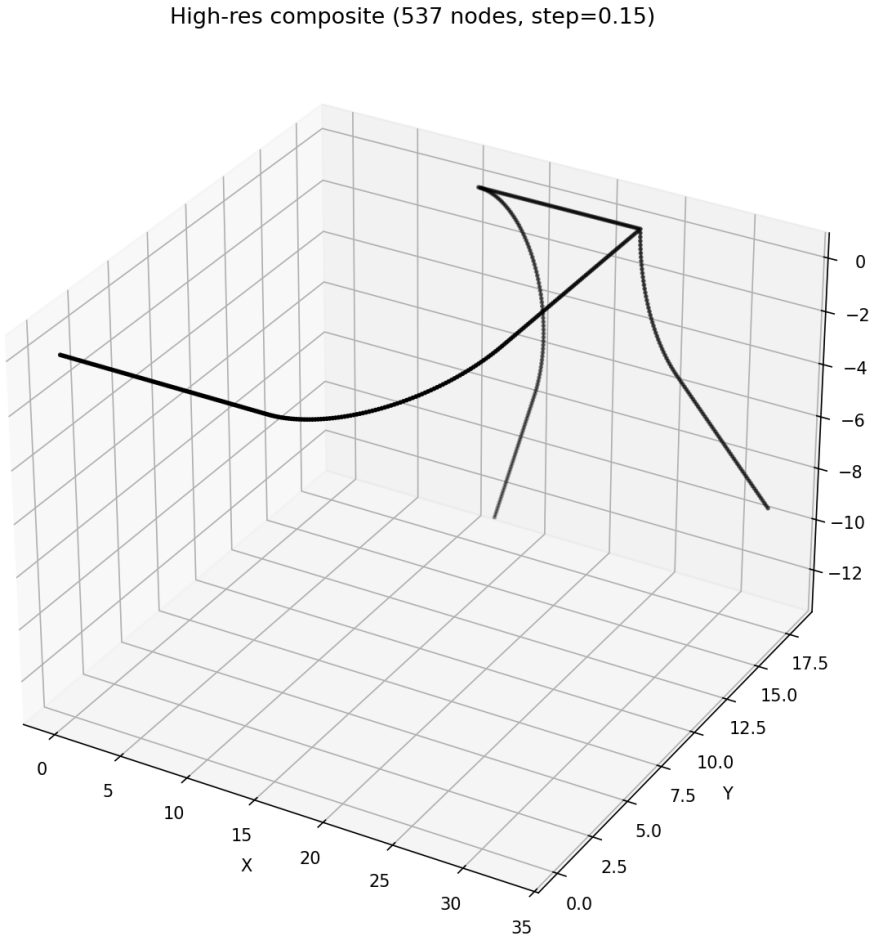

Figure 2: Example high-resolution centerline graph from the synthetic dataset with ground-truth primitive labels.

3 Synthetic dataset

We release a synthetic dataset of 3D pipe-centerline graphs with labeled primitives and arc angles [5]: *straight*, *corner*, *junction*, and *arc_θ* for $\theta \in \{10°, 20°, \ldots, 170°\}$. Each sample is a small graph with node coordinates and connectivity; node-wise geometric features (e.g., local curvature) are optionally stored for analysis. Check Figure 2.

4 Methods

Curvature and alignment segmentation

Given three consecutive points (p, q, r) along a polyline, we estimate curvature using the Menger curvature [8]:

$$\kappa_M(p, q, r) = \frac{4A(p, q, r)}{\|p - q\| \, \|q - r\| \, \|r - p\|}, \tag{1}$$

where $A(p, q, r)$ is the triangle area. This provides a scale-aware curvature estimate suitable for detecting straight sections ($\kappa \approx 0$) vs arcs ($\kappa > 0$) after smoothing/resampling.

Straightness test (cosine alignment). To reduce false positives (e.g., near-straight arcs), we also compute the cosine alignment:

$$\cos\theta = \frac{(q-p)^\top (r-q)}{\|q-p\|\,\|r-q\|}. \tag{2}$$

If κ_M is below a small threshold *and* $\cos\theta \approx 1$ (e.g., $\cos\theta \geq 1-\epsilon$), then the three points are approximately collinear in 3D, and the local region is classified as straight. Practically, we use robust statistics (median or trimmed mean) over a sliding window.

Graph encoder (GATv2)

We use a GATv2-based encoder that maps a variable-size graph to a fixed-dimensional embedding [2]. The network applies several attention layers and pools node features into a graph feature using attentional aggregation, implemented with PyTorch Geometric primitives [9, 10].

Supervised contrastive learning

We train embeddings using supervised contrastive learning (SupCon) [1]. For each anchor i in a batch, let $P(i)$ be the set of positives. With embedding vectors $\{z\}$ and temperature τ, the loss is:

$$\mathcal{L}_{\text{SupCon}} = \frac{1}{|I|}\sum_{i\in I}\left[-\frac{1}{|P(i)|}\sum_{p\in P(i)}\log\frac{\exp(z_i^\top z_p/\tau)}{\sum_{a\in I\setminus\{i\}}\exp(z_i^\top z_a/\tau)}\right]. \tag{3}$$

Positive definition. Non-arc segments are positives if they share the same motif label. Arc segments are positives if their angle difference is within a tolerance Δ (degrees). Mixed arc/non-arc pairs are negatives.

5 Implementation

Training Setup

We implement the encoder with PyTorch Geometric (GATv2Conv + attentional pooling) [9, 10] and train on the synthetic dataset [5]. A motif-balanced batch sampler improves positive coverage by ensuring multiple samples per motif in each batch. Retrieval@K on a stratified validation subset measures whether nearest neighbours match the query under the same correctness rule as training.

Real Geometry Ingest

In production, mesh ingest should: (1) extract a centerline, (2) build a graph, and (3) preserve a mapping from centerline nodes back to the original geometry for visual highlighting. For tubular meshes, VMTK-style pipelines can compute centerlines reliably when provided with inlet/outlet seeds [6, 7].

Gemini 3 Agent

Gemini 3 is used as the reasoning layer that converts structured geometry facts (segment list, angles, connectivity) into natural-language answers and UI actions (highlight segment IDs, zoom, explain) [3, 4]. The agent is tool-driven: it queries endpoints for segmentation/embedding results and formats responses for the front-end.

6 Results and Discussion

On the synthetic benchmark, retrieval@5 rapidly saturates, indicating that primitives are strongly separable under the current generation distribution. This is encouraging for a demo, but it also suggests a need for stronger dataset diversity if the target deployment includes noisier or atypical geometries.

Why "too perfect" accuracy can happen. If the generator produces highly regular centerlines (uniform spacing, clean arcs, canonical junctions), even modest models can reach near-perfect retrieval. Real-world meshes introduce sampling noise, discretisation artifacts, small manufacturing offsets, and ambiguous junction regions. The next dataset iteration should add: random perturbations, variable radii, partial segments, near-straight arcs, merged junctions, and centerline extraction noise.

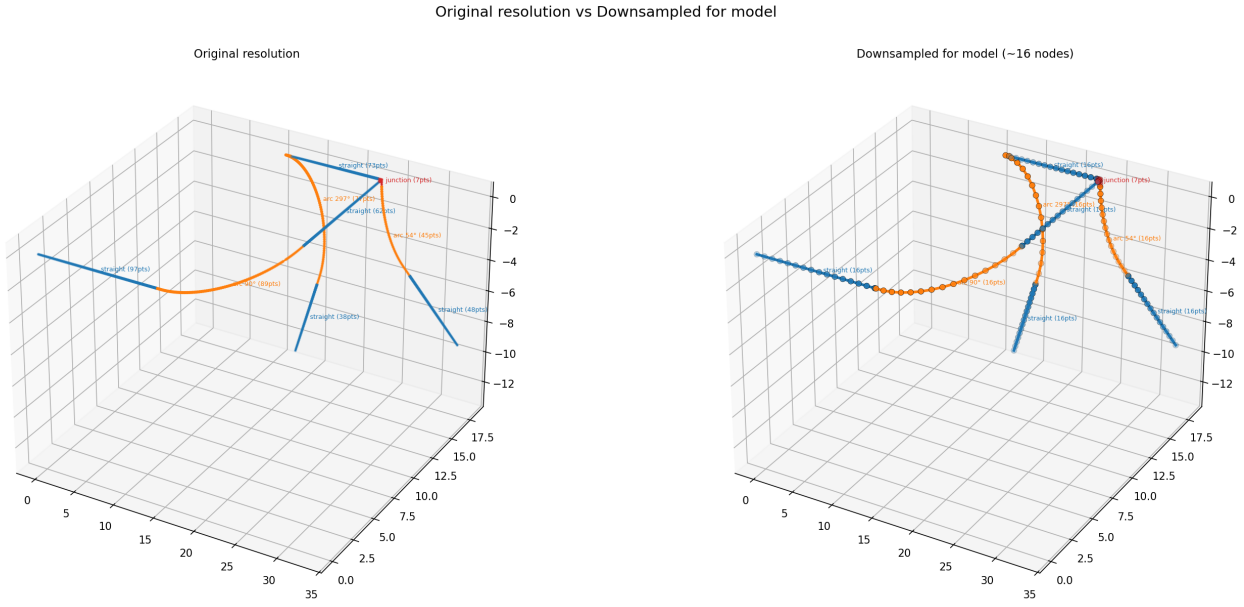

Figure 3: Qualitative result: primitive prediction before vs after downsampling. The downsampled graph is the representation consumed by the encoder, enabling efficient inference with minimal loss in topology cues.

Leakage considerations. If node features include direct encodings of labels (e.g., total bend proportional to arc angle), the model may learn shortcuts. Removing such features improves robustness and makes metrics more meaningful.

We downsample each segment to a fixed number of nodes (e.g., 16) before encoding; a qualitative example of the effect on predictions is shown in Figure 3.

7 Future Work

(i) Production centerline extraction: integrate a robust centerline method for mesh inputs and add seed inference [6].
(ii) Richer primitives: expand beyond straight/arc/junction to reducers, expansions, elbows with varying radii, and compound junction angles.
(iii) Foundation model for engineering geometry: scale training to noisy real meshes and multi-part assemblies.
(iv) CFD sandbox loop: connect geometry understanding to automated simulation setup and cost reduction.

Acknowledgements

This project was built as a rapid prototype for a 3-minute demo: (1) synthetic dataset generation, (2) GNN contrastive embedding, (3) centerline segmentation, and (4) a Gemini 3 agent for interactive geometry Q&A.

References

[1] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *arXiv preprint arXiv:2004.11362*, 2020.

[2] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[3] Gemini 3 developer guide (gemini api). https://ai.google.dev/gemini-api/docs/gemini-3. Accessed 2026-02-10.

[4] Gemini models: Model list and capabilities (gemini api). https://ai.google.dev/gemini-api/docs/models. Accessed 2026-02-10.

[5] bayang/shape: Synthetic 3d pipe centerline graph dataset. https://huggingface.co/datasets/bayang/shape. Accessed 2026-02-10.

[6] Richard Izzo, David A. Steinman, Simone Manini, and Luca Antiga. The vascular modeling toolkit: A python library for the analysis of tubular structures in medical images. *Journal of Open Source Software*, 3(25):745, 2018.

[7] Vmtk centerline computation notes (slicer extension documentation). https://github.com/vmtk/SlicerExtension-VMTK/blob/master/Docs/CenterlineComputation.md. Accessed 2026-02-10.

[8] Emil Saucan. Metric curvatures and their applications i. *Geometry, Imaging and Computing*, 2(4):257–334, 2015.

[9] Pytorch geometric: `GATv2Conv` documentation. https://pytorch-geometric.readthedocs.io/en/stable/generated/torch_geometric.nn.conv.GATv2Conv.html. Accessed 2026-02-10.

[10] Pytorch geometric: `AttentionalAggregation` documentation. https://pytorch-geometric.readthedocs.io/en/stable/generated/torch_geometric.nn.aggr.AttentionalAggregation.html. Accessed 2026-02-10.