Modeling Wireframe Meshes with Discrete Equivalence Classes

Pengyun Qiu*, Rulin Chen*, Peng Song, and Ying He

***Abstract*—We study a problem of modeling wireframe meshes where the vertices and edges fall into a set of discrete equivalence classes, respectively. This problem is motivated by the need of fabricating large wireframe structures at lower cost and faster speed since both nodes (thickened vertices) and rods (thickened edges) can be mass-produced. Given a 3D shape represented as a wireframe mesh, our goal is to compute a set of template vertices and a set of template edges, whose instances can be used to produce a fabricable wireframe mesh that approximates the input shape. To achieve this goal, we propose a computational approach that generates the template vertices and template edges by iteratively clustering and optimizing the mesh vertices and edges. At the clustering stage, we cluster mesh vertices and edges according to their shape and length, respectively. At the optimization stage, we first locally optimize the mesh to reduce the number of clusters of vertices and/or edges, and then globally optimize the mesh to reduce the intra-cluster variance for vertices and edges, while facilitating fabricability of the wireframe mesh. We demonstrate that our approach is able to model wireframe meshes with various shapes and topologies, compare it with three state-of-the-art approaches to show its superiority, and validate fabricability of our results by making three physical prototypes.**

***Index Terms*—Wireframe structure, mesh optimization, hierarchical clustering, discrete equivalence classes, rationalization**

I. Introduction

A wireframe structure is made up of nodes and rods, where multiple rods are joined together at each node. Wireframe structures are materially efficient and lightweight, making them widely used in many fields, such as art, sculpture, and architecture [1]. Wireframe structures generally have a mesh-like pattern and are modeled as wireframe meshes, where nodes and rods are thickened mesh vertices and edges, respectively. Traditionally, both nodes and rods in a wireframe structure are custom manufactured due to their distinct shapes.

In this paper, we study a new problem of *modeling wireframe meshes where the vertices and edges fall into a set of discrete equivalence classes, respectively*. Such wireframe meshes make it possible to fabricate large wireframe structures at lower cost and faster speed since both the nodes and rods can be mass-produced, e.g., by molding and cutting, respectively. Moreover, fewer number of distinct shapes of nodes and rods also simplify the assembly process and ease maintenance of the assembled structure. However, this modeling problem is non-trivial due to two challenges. First, the geometry of the vertices and edges has to be optimized at the same time to make them fall into discrete equivalence classes while maintaining their connectivity in the mesh. Second, both vertices and edges must satisfy geometric constraints for fabrication since nodes (thickened vertices) and rods (thickened edges) should not collide one another in an assembled structure. In recent years, researchers have studied a simplified version of the problem, i.e., modeling wireframe meshes with discrete equivalence classes of vertices [2], [3], but their approaches cannot be easily extended to address the above two challenges.

Given a 3D shape represented as a wireframe mesh, our goal is to compute *a set of template vertices* and *a set of template edges*, whose instances can be used to produce a *fabricable* wireframe mesh that approximates the input shape. To achieve this goal, we propose a computational approach that generates the template vertices and template edges by iteratively clustering and optimizing the mesh vertices and edges. At the clustering stage, we cluster mesh vertices according to the vertex shape defined by the vertex's incident edges and cluster mesh edges according to the edge length. In particular, we perform the vertex/edge clustering using *hierarchical clustering*, allowing to automatically find a suitable number of clusters. At the optimization stage, we first locally optimize the mesh to reduce the number of clusters of vertices and/or edges and then globally optimize the mesh to reduce the intra-cluster variance for the vertices and edges. Specifically, we make two technical contributions:

- We propose a computational framework to model wireframe meshes with discrete equivalence classes via iteratively clustering and optimizing mesh vertices and edges simultaneously.
- We propose a local-global optimization scheme to minimize the number of clusters of mesh vertices and edges, respectively, while facilitating fabricability of the mesh and preserving its shape and features.

We demonstrate that our approach is able to model wireframe meshes with discrete equivalence classes for approximating surfaces with various shapes and topologies, and compare it with three state-of-the-art approaches [4], [2], [3] to show its superiority in reducing the number of distinct vertices

* joint first authors.

Manuscript received 16 July 2024; revised 30 March 30 2025; accepted April 9, 2025. This work was supported by the Singapore MOE AcRF Tier 2 Grant (MOE-T2EP20123-0016) and the Singapore MOE AcRF Tier 1 Grant (RT19/22).

Pengyun Qiu, Rulin Chen, and Peng Song are with Singapore University of Technology and Design, Singapore (e-mail: qq475127379@gmail.com, rulin_chen@sutd.edu.sg, peng_song@sutd.edu.sg).

Ying He is with Nanyang Technological University, Singapore. (e-mail: yhe@ntu.edu.sg).

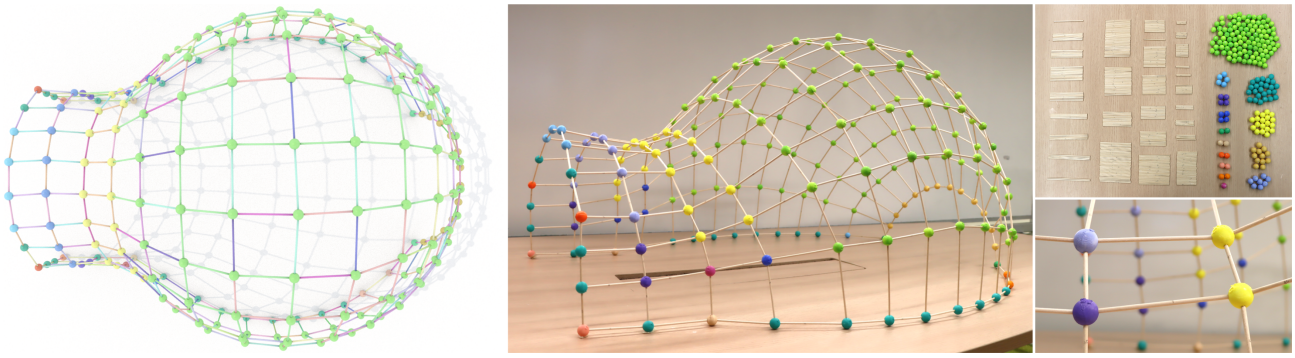


Fig. 1. An IGLOO surface with an original mesh containing 196 vertices and 364 edges optimized to 14 classes of vertices and 30 classes of edges using our approach. Left: a rendering of our optimized wireframe mesh, where vertices and edges in the same class are rendered with the same color, respectively. Middle: fabricating the wireframe mesh with 3D printed nodes and precision-cut wooden rods. Right: (top) nodes and rods of the same class are clustered together, and (bottom) a close-up view of the assembled nodes and rods.

and edges while preserving the input shape. We validate fabricability of our modeled wireframe meshes by making three physical prototypes with 3D printed nodes and precision-cut rods; see Figure 1 for an example.

II. Related Work

Fabricating wireframe meshes. Fabrication of wireframe meshes using common 3D printers, which print layer-by-layer, results in slow fabrication and low-quality prints. To address this limitation, researchers proposed methods to fabricate wireframe meshes using a custom-built 3D printer that extrudes filament directly in 3D space [5], a 5-DOF 3D printer (3 for translation and 2 for rotation) [6], and a 6-axis robotic arm with a customized extrusion head [7]. To fabricate wireframe meshes in a larger scale, an assembly-based method is typically used where vertices and edges are fabricated separately and then assembled to form the final structure. Richter and Alexa [8] proposed beam meshes, which are meshes with torsion-free edges, to approximate a given 3D shape, and fabricate both mesh vertices and edges using laser-cutting. Another common approach is to fabricate wireframe meshes as node-rod wireframe structures, where each node is 3D printed and each rod is fabricated by precision-cut [9], [10], [11].

Modeling wireframe meshes with discrete equivalence classes. A space frame structure or a space truss is an assembly of beams connected by nodes [1]. In recent years, there has been a research interest in rationalizing space frame structures by reducing the number of distinct nodes [12] or distinct beams [13], [14].

Wireframe structures are a specific kind of space frame structures whose shape can be represented as a *manifold mesh*. Two recent works [2], [3] succeed in rationalizing wireframe structures to reduce the number of distinct nodes by modeling wireframe meshes with discrete equivalence classes of vertices. They first define a metric to compare the shapes of different vertices, cluster all the vertices in a mesh based on the metric, and further perform local perturbation [2] or global optimization [3] on the mesh geometry until the shape variety of vertices in each cluster is under an acceptable tolerance. Compared to these two works, we address a new problem of modeling wireframe meshes with discrete equivalence classes of both vertices and edges, incorporate fabrication constraints in the mesh optimization, and enable automatic determination of the number of classes. In addition, quantitative comparisons show that our approach is able to achieve better performance than [2], [3] in terms of reducing the number of distinct vertices; see Section VIII.

Another way to rationalize wireframe structures is to use the Zometool construction set, which consists of nine template struts of different lengths and one universal node. The universal node is a slightly modified rhombicosidodecahedron with 62 slots, and only a few slots of each node are actually used for connecting struts in Zometool meshes. Zimmer and Kobbelt [15] proposed an advancing-front approach to approximate a freeform disk-topology surface using the Zometool construction set. To approximate a freeform surface with a more complex topology, Zimmer et al. [4] proposed an optimization approach that iteratively applies a set of local mesh modification operators on an initial Zometool mesh computed from a voxelization of the input shape. One limitation of this class of approaches is that the predefined template struts and universal node restrict 3D shapes that can be represented by these parts; see Section VIII for a quantitative comparison between our approach and the Zometool shape approximation [4].

Modeling polygonal surfaces with discrete equivalence classes. This family of works models freeform surfaces with discrete equivalence classes of polygons. Some research works compute a small set of template triangles [16], [17], [18], quads [19], or polygons [20] and use instances of these templates to approximate a given freeform surface, while others make use of a predefined set of template triangles [21] or quads [22] to model a variety of surfaces. Instead of reusing polygons (without thickness), computational approaches have been developed to model template shell blocks (with thickness), whose instances can be used to approximate an architectural surface [23] or an organic shape [24]. Beyond direct reuse of polygons or blocks, researchers have also computed reusable molds for fabricating an architectural surface with curved panel elements [25] or triangle-based point-folding elements [26], where elements of the same shape are fabricated with molding and then cut into different sizes and forms for making the surface.

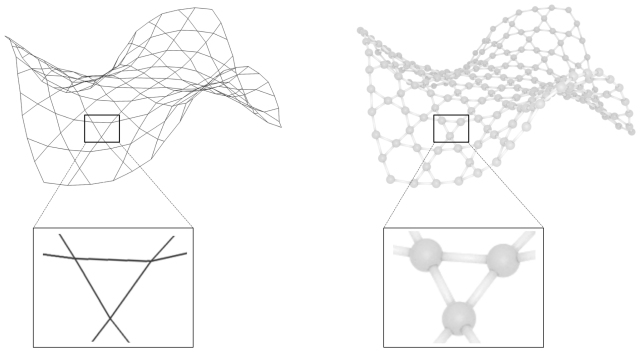


Fig. 2. Modeling (right) a wireframe structure from (left) a wireframe mesh by thickening both mesh vertices and edges.

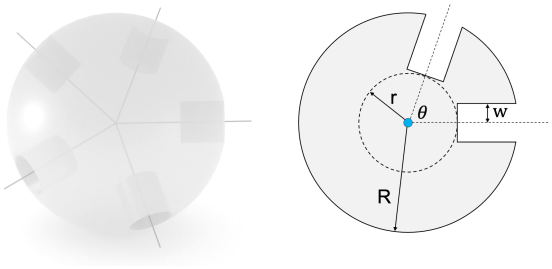


Fig. 3. Modeling the geometry of a node corresponding to a mesh vertex with valence 5, where the vertex is at the center of the node and each incident edge defines a cylindrical hole on the node. A cross-section of the node is shown on the right, where the cross-section plane passes through two cylindrical holes on the node.

III. Problem Formulation

Our input is a 3D shape represented as a triangle mesh, denoted as $\mathbf{P}$. Our goal is to model a fabricable polygonal mesh $\mathbf{M}$ to approximate the shape $\mathbf{P}$, where the vertices and edges fall into discrete equivalence classes, respectively. We begin by describing the modeling of a wireframe structure for fabricating a wireframe mesh $\mathbf{M}$. Subsequently, we introduce geometric constraints on the mesh $\mathbf{M}$ to facilitate its fabricability. Following this, we outline our problem formulation of modeling wireframe meshes with discrete equivalence classes of vertices and edges.

Modeling wireframe structures. To fabricate a wireframe mesh $\mathbf{M}$, we model a wireframe structure $\mathbf{S}$ from the mesh; see Figure 2. For each edge in the mesh, we model a rod as a cylinder centered at the edge with radius w. For each vertex in the mesh, we model a node as a sphere with radius R centered at the vertex. The sphere has m cylindrical holes, where m is the vertex valence; see Figure 3. Each cylindrical hole and the corresponding rod end form a friction joint for connecting the rod with the node. The radius of each hole is set as w. The depth of each hole is $R-r$, where r is typically set as $0.4R$. The length of each rod is $l-2r$, where l is the length of the corresponding mesh edge. This is because mesh edges meet exactly at a mesh vertex but rods meet at a spherical surface in the node with radius r.

In this paper, we make the following assumptions for the rods, nodes, and cylindrical holes in a wireframe structure: 1) all rods have a uniform cross section with radius w; 2) all nodes have a consistent radius R; and 3) all cylindrical holes in the nodes have the same depth $R-r$. Given these conditions, the geometry of wireframe structure $\mathbf{S}$ is precisely determined by the geometry of the wireframe mesh $\mathbf{M}$, including the shape of each mesh vertex (i.e., vertex valence and its incident edges) and length of each mesh edge.

Fabrication constraints. To ensure that each node is usable in practice, cylindrical holes in the node should not intersect with one another; see again Figure 3. This constraint is formulated as:

$$\theta > 2\arctan\frac{w}{r} \tag{1}$$

where θ is the angle between center lines (i.e., mesh edges) of the two cylindrical holes. In the mesh $\mathbf{M}$, any pair of edges joining at a vertex should satisfy Equation 1.

In a wireframe structure, it is essential that any pair of nodes connected by a rod does not collide with each other; see Figure 2 (right). This constraint is formulated as:

$$l > 2R \tag{2}$$

where l is the length of a mesh edge. All edges in the mesh $\mathbf{M}$ must satisfy Equation 2.

In the mesh $\mathbf{M}$, only vertices with the same valence can potentially fall into the same discrete equivalence class. Hence, to reduce the number of distinct vertices, we aim for the valence m of each vertex to be as close as possible to the optimal value m_{optim} of mesh $\mathbf{M}$, which is 6 in the interior and 4 on the boundary for triangle meshes.

Modeling wireframe meshes with discrete equivalence classes. To reduce the cost and time of fabricating mesh $\mathbf{M}$, the vertices and edges in mesh $\mathbf{M}$ should be grouped into discrete equivalence classes, respectively. We denote the number of discrete equivalence classes of vertices and edges as K_v and K_e, respectively. We consider a subset of vertices $\{v_{k,i}\}$ to fall into the same discrete equivalence class if

$$D_v(v_{k,i},\ \bar{v}_k) < \epsilon_v, \quad \forall i \tag{3}$$

where D_v is a shape dissimilarity metric of two vertices (defined in Section V), $\bar{v}_k$ is the centroid vertex of the class, and ϵ_v is tolerance on the vertex shape dissimilarity. Similarly, a subset of edges $\{e_{l,j}\}$ falls into the same discrete equivalence class if

$$D_e(e_{l,j},\ \bar{e}_l) < \epsilon_e, \quad \forall j \tag{4}$$

where D_e is a length dissimilarity metric of two edges (defined in Section V), $\bar{e}_l$ is the centroid edge of the class, and ϵ_e is tolerance on the edge length dissimilarity.

Modeling of a wireframe mesh $\mathbf{M}$ with discrete equivalence classes should fulfill the following requirements:

1) *Shape closeness.* The geometry of the mesh $\mathbf{M}$ should closely approximate the input shape $\mathbf{P}$, particularly for shape features.
2) *Mesh fabricability.* The mesh $\mathbf{M}$ should adhere to the fabrication constraints described in Equations 1 and 2.
3) *Discrete equivalence class of vertices.* Vertices in each discrete equivalence class should have very similar shapes by satisfying Equation 3.

4) *Discrete equivalence class of edges.* Edges in each discrete equivalence class should have very similar lengths by satisfying Equation 4.
5) *Distinct vertices.* The total number of discrete equivalence classes of vertices, K_v, should be small.
6) *Distinct edges.* The total number of discrete equivalence classes of edges, K_e, should be small.

Our wireframe mesh modeling process outputs a set of template vertices $\{\bar{v}_k\}$, $k \in [1, K_v]$ and a set of template edges $\{\bar{e}_l\}$, $l \in [1, K_e]$, where a template vertex $\bar{v}_k$ (a template edge $\bar{e}_l$) is the centroid vertex (edge) of each discrete equivalence class. These template vertices and edges will be thickened to form template nodes and rods, respectively. Instances of these template nodes and rods will be fabricated and assembled to build a wireframe structure $\mathbf{S}$ that approximates the input shape $\mathbf{P}$.

IV. Computational Framework

The problem of modeling wireframe meshes with discrete equivalence classes formulated in Section III is highly non-trivial due to large discrete-continuous search space (i.e., mesh topology and geometry) as well as many requirements on the mesh vertices and edges. To address this challenging problem, we propose a computational framework with two stages: *mesh initialization* that determines mesh topology and provides initial mesh geometry, and *mesh clustering-and-optimization* on the mesh vertices and edges to satisfy the requirements in Section III.

Mesh initialization. The input triangle mesh $\mathbf{P}$ typically contains an excess number of vertices and is not fabricable; see Figure 4 (left) for examples. Hence, we first initialize a fabricable mesh $\mathbf{M}$ by remeshing mesh $\mathbf{P}$; see Figure 4 (right). Users are allowed to specify a desired number of vertices for mesh $\mathbf{M}$. In case mesh $\mathbf{P}$ has salient shape features, users are allowed to identify these features interactively by brushing the feature vertices. More advanced methods such as [27] can also be employed to identify the features automatically. The objectives of our remeshing are to:

- Remove angles in the mesh that are too small (see Equation 1).
- Remove edges in the mesh that are too short (see Equation 2) or too long.
- Remove vertices whose valence is far from the optimal value m_{optim}.
- Reduce the total number of mesh vertices to approximately match the user-specified number.
- Preserve salient shape features during remeshing.

In general, we solve the mesh initialization problem by using the adaptive remeshing algorithm in [28], where the initialized mesh $\mathbf{M}$ is still a triangle mesh; see Figure 4 (top). The key idea in [28] is to replace the constant target edge length in [29] by an adaptive sizing field $L(\mathbf{x}_i)$ to guide the remeshing process, where $\mathbf{x}_i$ is a vertex on the input mesh $\mathbf{P}$. To avoid too short or too long edges in the mesh, the adaptive sizing field $L(\mathbf{x}_i)$ is clamped to user-specified bounds, such that $L(\mathbf{x}_i) \in [L_{\text{min}},\ L_{\text{max}}]$. To control the number of vertices in the remeshing result, the lower bound L_{min} is adjusted, where a larger lower bound corresponds to a smaller number of vertices.

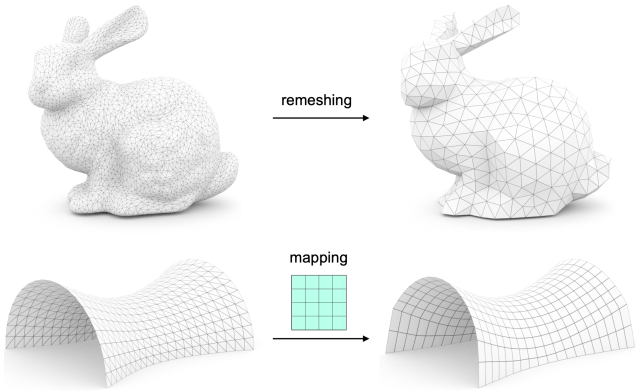


Fig. 4. Initialization of a fabricable mesh via (top) adaptive remeshing a surface or (bottom) mapping a 2D tessellation (in cyan color) onto an open surface.

For an open input mesh $\mathbf{P}$, we provide an alternative approach to solving the mesh initialization problem. In detail, we first parameterize the input mesh $\mathbf{P}$ (e.g., using a as-similar-as-possible mapping [30]) and then map a 2D regular or semi-regular tessellation onto the 3D surface $\mathbf{P}$; see Figure 4 (bottom). A mesh $\mathbf{M}$ initialized by this approach is a polygonal mesh, and its topology is defined by the 2D tessellation. In practice, we opt for 2D tessellations that incorporate triangles and/or quads since wireframe structures composed of those shapes tend to exhibit higher structural stability.

After the remeshing, the initialized mesh $\mathbf{M}$ satisfies requirements 1 and 2 in Section III. We further perform mesh clustering-and-optimization to satisfy the remaining requirements (requirements 3-6). Since the initialized mesh $\mathbf{M}$ already has a well-defined topology, the mesh optimization process will only modify its geometry but not topology.

Mesh clustering-and-optimization. Modeling a mesh with discrete equivalence classes is generally done via clustering and optimizing vertices and/or edges in the mesh. A typical computational framework is to cluster the vertices using K-means and then to minimize variance in each cluster via local [2] or global [3] mesh optimization. Since the optimization does not guarantee that the variance in each cluster is smaller than the allowed tolerance, users need to gradually increase the number of clusters, K, and repeat the clustering-and-optimization, until the tolerance requirement is satisfied. Although this framework is effective, one limitation is the difficulty to specify K. This limitation is severe in our problem since we have to specify two K's, i.e., K_v for clustering vertices and K_e for clustering edges.

To address the above limitation, we propose a new computational framework that enables automatic determination of K_v and K_e. Our framework has two unique features:

- *Hierarchical clustering.* Our framework uses hierarchical clustering instead of K-means to cluster vertices and edges in a mesh, respectively. This is because hierarchical clustering is able to automatically determine K for a prescribed tolerance; see Section V.

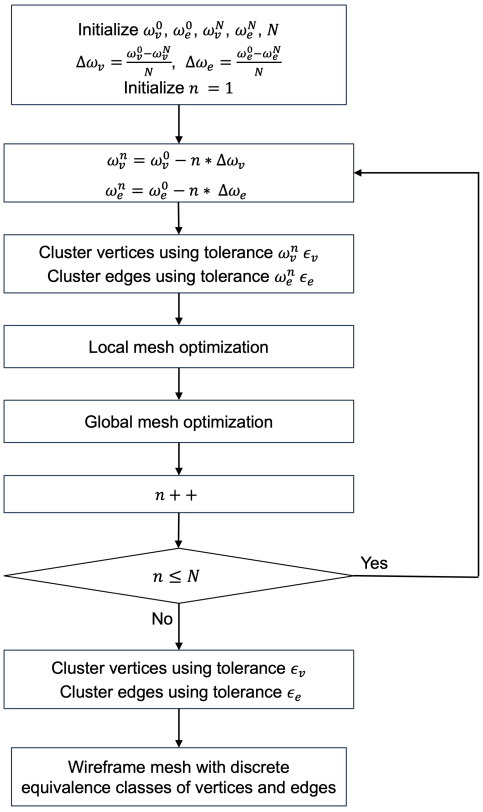


Fig. 5. A diagram of our computational framework to model wireframe meshes with discrete equivalence classes.

- *Local-global optimization.* Our framework combines the strength of local and global optimization to reduce the number of distinct vertices and edges. The strength of local mesh optimization is to reduce the number of clusters by eliminating clusters with few elements via cluster merging; see Section VI. The strength of global mesh optimization is to satisfy a set of requirements that may conflict with one another; see Section VII.

Figure 5 shows the diagram of our computational framework for iterative mesh clustering-and-optimization. At the beginning, we use a large vertex tolerance $\omega_v^0\epsilon_v$ and a large edge tolerance $\omega_e^0\epsilon_e$ for the clustering, where ω_v^0 and ω_e^0 are coefficients that are set to 3.0 by default. These large tolerances lead to a small number of vertex clusters and edge clusters, respectively. The intra-cluster variances for vertex clusters and edge clusters are then reduced by the (global) mesh optimization. At each iteration, we decrease the vertex tolerance and edge tolerance by $\Delta\omega_v\epsilon_v$ and $\Delta\omega_e\epsilon_e$, respectively, and re-cluster the mesh vertices and edges using hierarchical clustering to discover new clusters of vertices and edges subject to the decreased tolerances. We perform this iterative clustering-and-optimization process until vertex tolerance is reduced to $\omega_v^N\epsilon_v$ and the edge tolerance is reduced to $\omega_e^N\epsilon_e$, where ω_v^N and ω_e^N are typically set to 1.0 and N is set to 20 by default. At the end, we perform hierarchical clustering on the vertices and edges in the optimized mesh using tolerance ϵ_v and ϵ_e to determine K_v and K_e, respectively. Note that hierarchical clustering using tolerance ϵ_v (ϵ_e) does not guarantee that the radius of each vertex (edge) cluster is always less than ϵ_v (ϵ_e); see Equations 3 and 4. Hence, we perform a final check on the radius of each vertex/edge cluster and split the cluster if it does not satisfy Equation 3 and/or 4. Thanks to our global optimization that minimizes the intra-cluster variance of vertex/edge clusters, the cluster splitting is rarely needed in our experiments.

Besides the ability to automatically determine K_v and K_e, our framework also allows users to specify their preference in reducing the number of distinct vertices or edges. For example, users may have a preference to reduce the number of distinct vertices to decrease the total fabrication cost in case that nodes are made with expensive molds while rods are fabricated with cheap wood cutting. The preference is specified via adjusting the values of ω_v^N and ω_e^N; see Figure 15 for an example. A large value of $\omega_v^N > 1.0$ ($\omega_e^N > 1.0$) means a preference on reducing the number of distinct vertices (edges). A small value of $\omega_v^N < 1.0$ ($\omega_e^N < 1.0$) means less interest in reducing the number of distinct vertices (edges). In particular, our framework allows to model wireframe meshes with discrete equivalence classes of vertices (or edges) by setting $\omega_e^0 = \omega_e^N = 0$ ($\omega_v^0 = \omega_v^N = 0$).

V. Clustering

In this section, our goal is to partition all the edges and vertices in the mesh $\mathbf{M}$ into:

- clusters of edges $\mathbf{C}_e = \{C_e^l\}$, where the radius of each edge cluster $C_e^l = \{e_{l,j}\}$ is less than a prescribed tolerance $\omega_e^n\epsilon_e$; and
- clusters of vertices $\mathbf{C}_v = \{C_v^k\}$, where the radius of each vertex cluster $C_v^k = \{v_{k,i}\}$ is less than a prescribed tolerance $\omega_v^n\epsilon_v$.

We solve the problem via hierarchical clustering as mentioned in Section IV.

Clustering mesh edges. To cluster mesh edges, each edge is represented by a single number, which is the edge length. We cluster edges using hierarchical agglomerative clustering [31], which starts with each data point (e.g., an edge length number) in its own cluster and progressively join the closest clusters to reduce the number of clusters by one until there is a single cluster comprising all the data points. We measure the distance

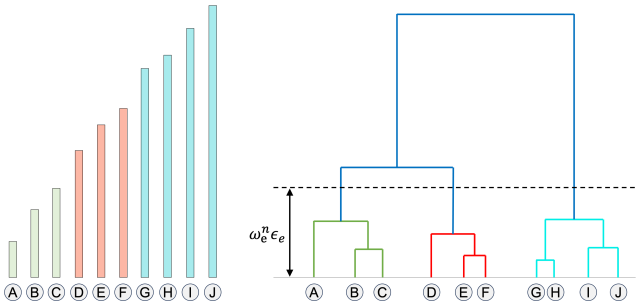


Fig. 6. Hierarchical clustering of (left) 10 edges and (right) the resulting dendrogram. The horizontal dashed line shows the prescribed tolerance $\omega_e^n\epsilon_e$, resulting in three clusters of edges.

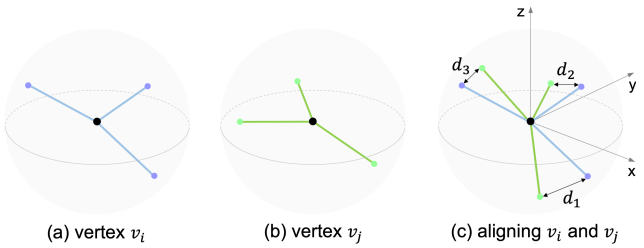


Fig. 7. We measure the shape dissimilarity of two vertices (a) v_i and (b) v_j with valence 3 by (c) finding the best alignment transformation that minimizes the sum of squared distances (i.e., d_1, d_2 and d_3 in this example) between corresponding neighboring vertices projected onto a unit sphere.

between two edges based on the absolute difference of the two edge lengths:

$$D_e(e_i,\ e_j) = |\text{len}(e_i) - \text{len}(e_j)| \tag{5}$$

We measure the distance between two edge clusters using:

$$D_e(C_e^i,\ C_e^j) = D_e(\bar{e}_i,\ \bar{e}_j) \tag{6}$$

where $\bar{e}_i$ and $\bar{e}_j$ are the centroid of the edge clusters C_e^i and C_e^j, respectively. The centroid of each cluster is calculated as the mean of the edge lengths.

Figure 6 shows the dendrogram that visualizes the result of hierarchical clustering of 10 edges. The number of edge clusters can be easily determined based on the dendrogram and the prescribed tolerance $\omega_e^n \epsilon_e$, since the height in the dendrogram represents the distance between clusters. In detail, we first draw a horizontal line at height $\omega_e^n \epsilon_e$; see the dashed line in Figure 6. Then, we count the number of vertical lines in the dendrogram that intersect with the horizontal line, which is the number of clusters. Mesh edges that are joined together below each vertical line form a cluster.

Clustering mesh vertices. A mesh vertex's shape is defined by its incident edges; see Figure 7(a&b). The normalized direction of each incident edge is represented by 2 variables in a spherical coordinate system. Hence, the shape of a mesh vertex with valence m is represented by $2m$ independent variables. In this paper, only vertices with the same valence can fall into the same cluster. We cluster mesh vertices with the same valence using hierarchical agglomerative clustering. The only difference from clustering mesh edges is the distance metric for the clustering, which we introduce below.

We compute the distance metric between a pair of vertices v_i and v_j with valence m by measuring their shape dissimilarity. This problem has been addressed by different approaches [12], [2], [3]. In this paper, we choose the approach in [3] due to its efficiency and give a brief introduction of it for completeness. To measure shape dissimilarity between a pair of vertices v_i and v_j with valence m, we first move both vertices to the origin with their neighboring vertices projected onto a unit sphere. By this, each vertex is represented as an ordered set of its neighboring vertices. Then, we compute their best alignment by fixing one vertex v_i and rotating v_j to minimize the sum of squared distances between corresponding neighboring vertex pairs; see Figure 7(c). Due to the choice of starting vertex and the order of neighboring vertices, there are $2m$ possible permutations for v_j. Thus, the shape dissimilarity is measured using:

$$D_v(v_i,\ v_j) = \min_{k=1}^{2m} \min \sqrt{\frac{\sum_{l=1}^{m}(d_l)^2}{m}} \tag{7}$$

where d_l is the distance between lth pair of corresponding neighboring vertices and k is the permutation index. We solve the minimization problem using singular value decomposition to find the optimal rotation matrix $\mathbf{T}_j$ of vertex v_j [3]. In particular, $D_v(v_i,\ v_j) = 0$ indicates that the two vertices have exactly the same shape.

We measure the distance between two vertex clusters using:

$$D_v(C_v^i,\ C_v^j) = D_v(\bar{v}_i,\ \bar{v}_j) \tag{8}$$

where $\bar{v}_i$ and $\bar{v}_j$ are the centroid of the vertex clusters C_v^i and C_v^j, respectively. To compute the cluster centroid, we align all the vertices in a cluster using Equation 7 and then compute the shape of the centroid by averaging the corresponding neighboring vertices projected onto the unit sphere.

Algorithm 1 Algorithm to decrease the number of vertex clusters via local mesh optimization.

```
function LocalMeshOptimization(M)
    list L ← ∅
    for each cluster C_v^k in C_v do
        if number of vertices in C_v^k is ≤ h_thres then
            list L.push(C_v^k)
        end if
    end for
    sort list L in ascending order in terms of cluster size
    for each cluster C_v^i in list L do
        eliminate_success ← True
        for each vertex v in cluster C_v^i do
            select a cluster C_v^t from C_v − L
            if ReassignVertexToCluster(v, C_v^t) == True then
                update centroids of clusters C_v^i and C_v^t
            else
                eliminate_success ← False
                break
            end if
        end for
        if eliminate_success == True then
            C_v.pop(C_v^i)
        else
            reset vertices and centroid of cluster C_v^i
            reset vertices and centroid of clusters {C_v^t}
        end if
    end for
end function
```

VI. Local Mesh Optimization

The goal of local mesh optimization is to reduce the number of distinct vertices and/or edges by perturbing the mesh $\mathbf{M}$. Denote the number of distinct vertices and the number of distinct edges at nth iteration in our framework as K_v^n and

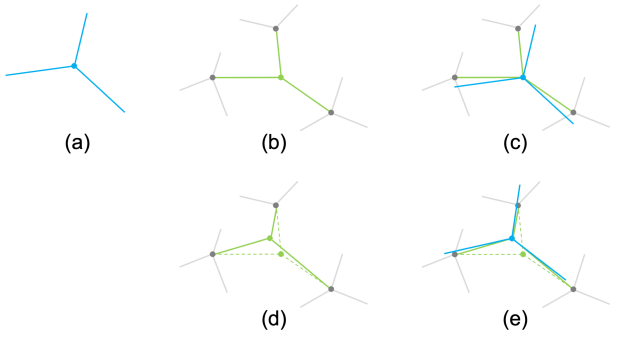


Fig. 8. Vertex perturbation to reassign (b) a vertex (in green) to a cluster whose (a) centroid (in blue) has (c) a similar shape. (d&e) Vertex perturbation makes the vertex's shape closer to that of the centroid, yet it may also modify the shapes of neighboring vertices (in black) and the lengths of incident edges (in green).

K_e^n, respectively. According to the user's preference, the goal can be classified into three cases:

1) decrease K_v^n only;
2) decrease K_e^n only;
3) decrease K_v^n and K_e^n.

We introduce the formulation and approach of our local mesh optimization to achieve the goal of the first case (Section VI-A). After that, we explain how to make slight changes to the formulation and approach to achieve the goal of the other two cases (Sections VI-B and VI-C).

A. Decrease K_v^n

After the clustering in Section V, there may exist some clusters with few vertices. Our idea is that we are possible to eliminate a cluster with few vertices by modifying the shape of each vertex and reassigning the modified vertex to another cluster without violating the requirement of tolerance $\omega_v^n \epsilon_v$. Algorithm 1 shows the process of eliminating clusters with $h \leq h_{\text{thres}}$ vertices, where h_{thres} is set to 2 in our experiments. For each cluster with h vertices, the algorithm tries to reassign each of the h vertices to another cluster by slightly modifying the shape of the vertex via vertex perturbation. The cluster to reassign the vertex is chosen as the one whose centroid has the same valence as the vertex and has the closest shape to the vertex.

The key operation in our local mesh optimization is to reassign a vertex v to a selected cluster C_v^t via perturbing the vertex to modify its shape; see `ReassignVertexToCluster(`v`,` C_v^t`)` in Algorithm 1 and Figure 8. The vertex reassignment via perturbation has to satisfy several requirements, which we will introduce below. If a vertex reassignment is not successful, it means that the cluster where the vertex falls in cannot be eliminated. In this case, we undo all the changes on the mesh $\mathbf{M}$ as well as the vertex clusters $\mathbf{C}_v$.

Reassign a vertex via perturbation. Since our goal is to decrease the number (K_v^n) of distinct vertices, modifying the shape of a vertex v should reassign the vertex v to a selected cluster C_v^t:

$$D_v(v,\ \bar{v}_t) < \omega_v^n \epsilon_v, \tag{9}$$

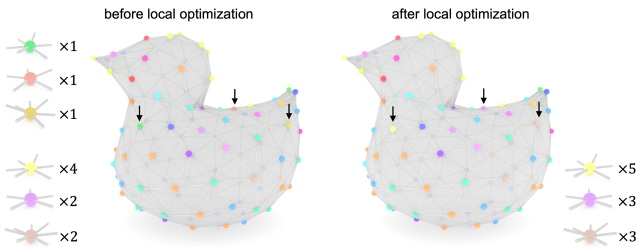


Fig. 9. Clustering of vertices (left) before and (right) after running our local mesh optimization to eliminate clusters with few vertices. Three clusters, each of which has a single vertex, eliminated by our vertex perturbation are shown on the top left corner, and also highlighted with black arrows in the mesh on the left. The centroids of three clusters to assign the three vertices, respectively, are shown at the bottom.

where $\bar{v}_t$ is the centroid of the selected cluster C_v^t.

To satisfy the shape closeness requirement in Section III, the perturbed vertex v should not be too far away from the input surface $\mathbf{P}$. This requirement is formulated as minimizing the following distance:

$$\text{Dist}(v,\ \mathbf{P}) = \|\mathbf{v} - \mathbf{c}(\mathbf{v})\|, \tag{10}$$

where $\mathbf{v}$ denotes the 3D position of vertex v and $\mathbf{c}(\mathbf{v})$ is the closest point on the input mesh $\mathbf{P}$ to the vertex v.

Perturbing a vertex v will likely change its own shape as well as the shape of each of the vertex's neighboring vertices $N(v)$ due to the edge shared between the vertex v and its neighbor; see Figure 8(d). Hence, the fabrication constraint on the vertex angle (Equation 1) has to be satisfied for the vertex v and each of its neighbors. Moreover, we require each neighboring vertex $v_k \in N(v)$ with modified shape to stay in its own cluster as a hard constraint:

$$D_v(v_k,\ \bar{v}_k) < \omega_v^n \epsilon_v, \quad \forall v_k \in N(v) \tag{11}$$

where $\bar{v}_k$ is the centroid of the vertex cluster that the neighboring vertex v_k falls in. The purpose of this hard constraint is to avoid any chance of increasing the number (K_v^n) of vertex clusters since large modification on the neighboring vertices' shapes may introduce new clusters of vertices.

Perturbing a vertex v will likely change the length of each of the vertex's incident edges $I(v)$ since the other end (i.e., the neighboring vertex) of the edge is fixed; see again Figure 8(d). Hence, the fabrication constraint on the edge length (Equation 2) has to be satisfied for each incident edge $e_l \in I(v)$. Moreover, we require each incident edge $e_l \in I(v)$ with modified length to stay in its own cluster as a soft constraint:

$$D_e(e_l,\ \bar{e}_l) < \omega_e^n \epsilon_e, \quad \forall e_j \in I(v) \tag{12}$$

where $\bar{e}_l$ is the centroid of the edge cluster that the incident edge e_l falls in. The purpose of this soft constraint is to avoid increasing the number (K_e^n) of edge clusters.

Vertex perturbation solver. In the above vertex perturbation problem, the search space is rather small, which is the 3D position $\mathbf{v}$ of the vertex v. One straightforward approach is to randomly sample many 3D points around the initial position of the vertex v, and then check if a sampled position satisfies the

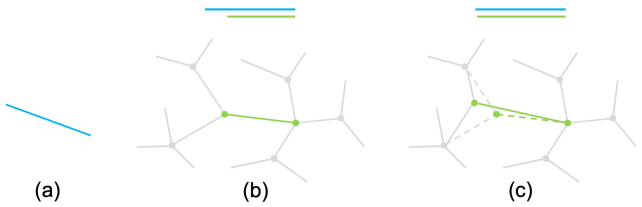


Fig. 10. Vertex perturbation to reassign (b) an edge (in green) to a cluster whose (a) centroid (in blue) has a similar length to the edge. (c) Perturbing one end (i.e., the left end) of the edge makes the edge's length closer to that of the centroid.

above constraints. However, one limitation of this sampling-based solver is its efficiency. To this end, we formulate an optimization problem to search for a feasible 3D position of the vertex v:

$$E_{\text{local}} = \omega_1 (D_v(v,\ \bar{v}_t))^2 \ +\ \omega_2 (\text{Dist}(v,\ \mathbf{P}))^2 \ + \\ \omega_3 \sum_k (D_v(v_k,\ \bar{v}_k))^2 \ +\ \omega_4 \sum_l (D_e(e_l,\ \bar{e}_l))^2 \qquad (13)$$

$$\text{s.t.} \quad \theta_i > 2 \arctan \frac{w}{r}, \qquad \forall \text{ mesh vertex} \in \{v, N(v)\} \\ l_j > 2R, \qquad \forall \text{ mesh edge} \in I(v)$$

where ω_1, ω_2, ω_3, and ω_4 are weights and we set their default values as 5, 1, 5, and 1, respectively. This optimization problem is a small-scale version (i.e., one-ring neighborhood around vertex v) of the global mesh optimization to be presented in Section VII. Although large weights (i.e., ω_1 and ω_3) have been assigned for the first and third terms in Equation 13, they cannot guarantee the hard constraints in Equations 9 and 11 are always satisfied. To resolve this issue, we perturb the optimization solution using the sampling-based approach if the solution does not satisfy Equations 9 and/or 11. The vertex reassignment via perturbation fails if the solver cannot find a feasible solution. Please refer to Section VII and the supplementary material for the optimization solver. Figure 9 shows a result before and after performing the local mesh optimization. In this example, 3 clusters of vertices, each of which has a single vertex, are eliminated by performing vertex perturbation.

B. Decrease K_e^n

Since our goal is to decrease the number (K_e^n) of distinct edges, we perturb either end of an edge e to change its length such that the modified edge can be reassigned to a selected cluster C_e^t; see Figure 10. The key operation is still vertex perturbation. Yet, the difference is that we have two vertices (i.e., two endpoints of the edge e) that can be perturbed. We perturb one endpoint vertex at a time using an approach similar to Section VI-A. Other than that, the process to perturb vertices for eliminating clusters with few edges follows Algorithm 1.

To eliminate clusters with few edges, there are two differences in the constraints of vertex perturbation. First, perturbing a vertex should reassign an edge e to a selected cluster C_e^t:

$$D_e(e,\ \bar{e}_t) < \omega_e^n \epsilon_e, \qquad (14)$$

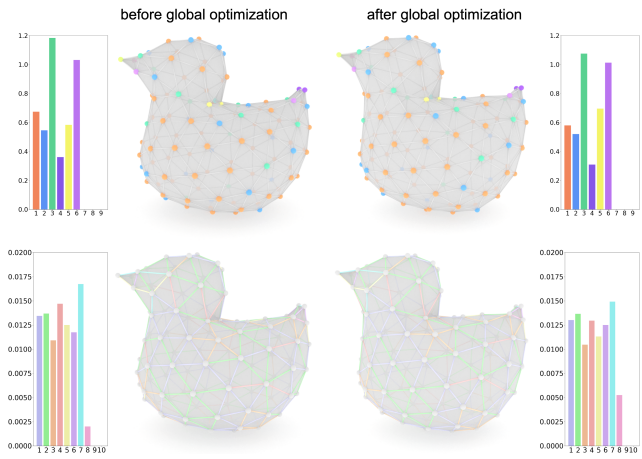


Fig. 11. (Top) 9 vertex clusters and (bottom) 10 edge clusters in a DUCK model (left) before and (right) after our global mesh optimization to reduce intra-cluster variance. The intra-cluster variance of each vertex/edge cluster is shown as a histogram beside the corresponding model. Note that vertex clusters and edge clusters are sorted according to the cluster size in descending order, respectively. The 7th, 8th, and 9th vertex clusters only have a single vertex and thus they do not have intra-cluster variance (i.e., no bar in the top histograms). The same applies to the 9th and 10th edge clusters.

where $\bar{e}_t$ is the centroid of the selected cluster C_e^t. Second, perturbing a vertex may change the shapes of its neighboring vertices and the lengths of its incident edges. Different from Section VI-A, we require each incident edge to stay in its own cluster as a *hard constraint* and require each neighboring vertex to stay in its own cluster as a *soft constraint*. To solve the vertex perturbation problem, the first energy term in Equation 13 is replaced with $(D_e(e,\ \bar{e}_t))^2$.

C. Decrease K_v^n and K_e^n

Since our goal is to decrease the number (K_v^n) of distinct vertices and the number (K_e^n) of distinct edges, we need to first run Algorithm 1 to eliminate clusters with few vertices without increasing K_e^n. Hence, when perturbing a vertex v, we require each neighboring vertex to stay in its own cluster as a *hard constraint* (Equation 11), and require each incident edge to stay in its own cluster as a *hard constraint* (Equation 12). After that, we run a similar version of Algorithm 1 to eliminate clusters with few edges without increasing K_v^n, where Equations 11 and 12 are also formulated as hard constraints.

VII. Global Mesh Optimization

The goal of our global optimization on the mesh $\mathbf{M}$ is to minimize intra-cluster variance for K_v^n clusters of vertices and K_e^n clusters of edges. Since vertex shapes and edge lengths are highly coupled with each other, they have to be optimized at the same time. Moreover, the optimization has to satisfy the fabrication constraints described by Equations 1 and 2, and does not cause the mesh $\mathbf{M}$ to deviate too much from the input surface $\mathbf{P}$, especially for shape features (if any).

Vertex clusters. We minimize the intra-cluster variance of vertex clusters using the term:

$$E_{\text{vertex}} = \sum_{k=1}^{K_v} \sum_i (D_v(v_{k,i},\ \bar{v}_k))^2 \qquad (15)$$

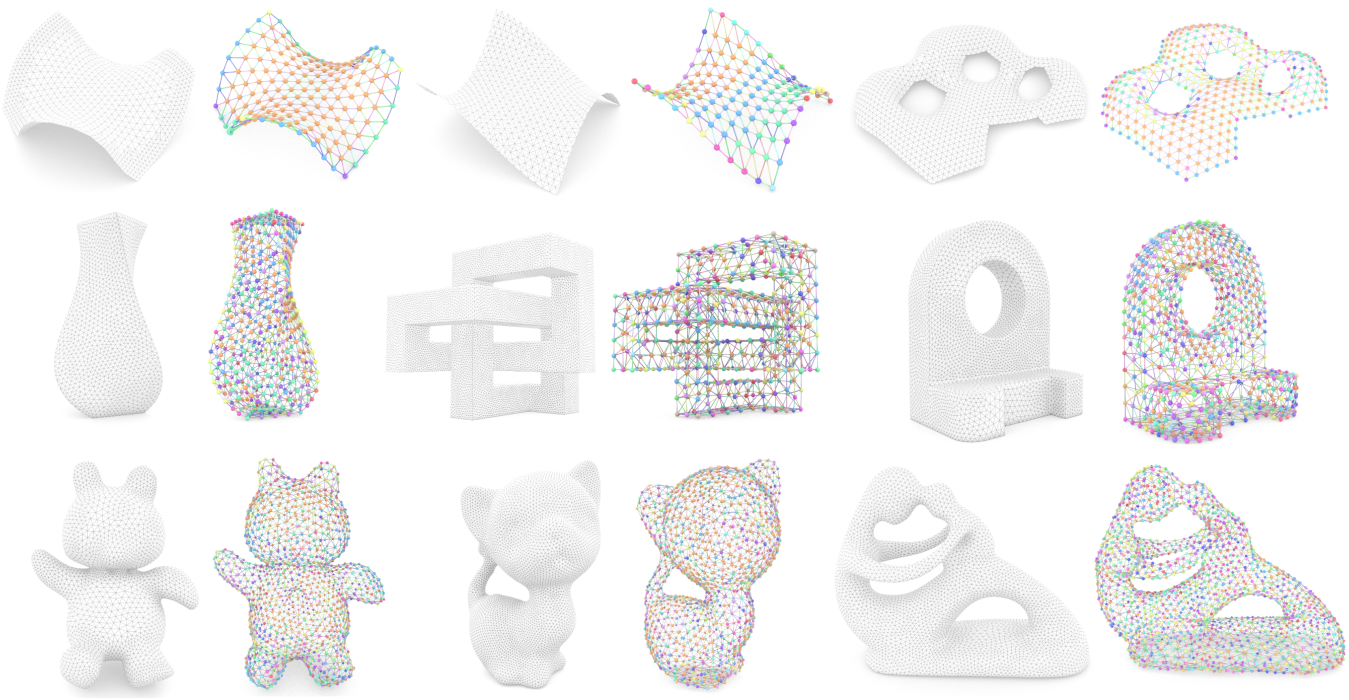


Fig. 12. Our approach allows modeling wireframe meshes with discrete equivalence classes for surfaces with various shapes and topologies. From left to right and then top to bottom: FLOWER, MONKEY SADDLE, FREE HOLES, VASE, KNOT, CAD PART, TEDDY, KITTEN, and FERTILITY. The input surface $\mathbf{P}$ is also shown beside each result.

where $v_{k,i}$ is the ith vertex in the kth vertex cluster, and $\bar{v}_k$ is the centroid of the kth vertex cluster.

Edge clusters. We minimize the intra-cluster variance of edge clusters using the term:

$$E_{\text{edge}} = \sum_{l=1}^{K_e} \sum_{j} \left(D_e(e_{l,j},\ \bar{e}_l)\right)^2 \tag{16}$$

where $e_{l,j}$ is the jth edge in the lth edge cluster, and $\bar{e}_l$ is the centroid of the lth edge cluster.

Shape closeness. We maintain shape closeness of the optimized mesh using the term:

$$E_{\text{shape}} = \sum_{i}^{N_v} \|\mathbf{v}_i - \mathbf{c}(\mathbf{v}_i)\|^2, \tag{17}$$

where $\{\mathbf{v}_i\}$ denotes the 3D positions of all the N_v vertices of the mesh $\mathbf{M}$, and $\mathbf{c}(\mathbf{v}_i)$ is the closest point on the input mesh $\mathbf{P}$ to the vertex $\mathbf{v}_i$.

Shape features. We preserve shape features using the term:

$$E_{\text{feature}} = \sum_{j}^{N_f} \|\mathbf{v}_j - \mathbf{v}'_j\|^2, \tag{18}$$

where $\{\mathbf{v}_j\}$ denotes the 3D positions of all the N_f feature vertices of the mesh $\mathbf{M}$, and $\mathbf{v}'_j$ is the position of a feature vertex in the input mesh $\mathbf{P}$.

Global optimization. The objective function of our global optimization is a weighted sum of the above energy terms:

$$E_{\text{global}} = \lambda_1 E_{\text{vertex}} + \lambda_2 E_{\text{edge}} + \lambda_3 E_{\text{shape}} + \lambda_4 E_{\text{feature}} \tag{19}$$

$$\text{s.t.} \qquad \theta_i > 2\arctan\tfrac{w}{r}, \quad \forall \text{ vertex angle}$$
$$l_j > 2R, \quad \forall \text{ mesh edge}$$

where θ_i is the angle between any two edges joined at the same vertex (see Equation 1), and l_j is the length of a mesh edge (see Equation 2). We empirically set the weights as $\lambda_1 = 3$, $\lambda_2 = 6$, $\lambda_3 = 4$, and $\lambda_4 = 20$ in our experiments.

Solver. The search space of our optimization problem is the positions $\{\mathbf{v}_i\}$ of all the vertices in the mesh $\mathbf{M}$. We represent each of the above energy terms and constraints using $\{\mathbf{v}_i\}$, reformulate our global optimization as a least squares problem, and solve it using the successive over-relaxation (SOR) in Eigen library [32]. Please refer to the supplementary material for implementation details of our solver. Figure 11 shows a mesh model before and after the global optimization for a single iteration in our computational framework, from which we can see that our global optimization reduces the intra-cluster variance of vertex clusters and edge clusters while preserving the input shape as well as its features (e.g., duckbill and tail).

VIII. RESULTS

We implemented our approach in C++ and libigl [33] on a desktop computer with a 3.7GHz CPU and 16GB memory. In our experiments, we set the tolerance ϵ_v in Equation 3 to 0.0872, which is equivalent to limiting the average deviation angle (between corresponding incident edges) smaller than 5° when aligning each vertex to the cluster centroid; see again Figure 7. We set the tolerance ϵ_e in Equation 4 to 1% of the average edge length in the mesh $\mathbf{M}$.

We present our virtual results using a consistent scheme. First, we show each modeled wireframe mesh with thickened vertices and edges (i.e., wireframe structure) to better visualize the result. Second, for each result, both vertex clusters and edge clusters are sorted according to the cluster size in

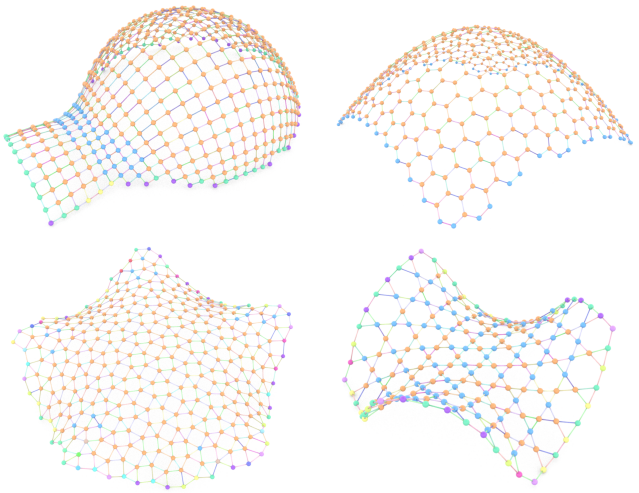


Fig. 13. Our approach allows modeling wireframe polygonal meshes with different patterns, including quad, hexagon, triangle-quad, and triangle-hexagon patterns.

descending order, respectively. Third, both vertex clusters and edge clusters are rendered using a consistent color scheme. For example, the first vertex cluster is rendered in orange and the second vertex cluster is rendered in blue; the first edge cluster is rendered in blue and the second edge cluster is rendered in green. Please refer to the histogram bar colors in Figure 11 for the color scheme.

Results. We show that our approach allows modeling wireframe meshes with discrete equivalence classes for surfaces with various shapes and topologies in Figure 12, including freeform architectural surfaces (top), man-made objects with sharp features (middle), and organic shapes with branches and/or holes (bottom). Shape features in all these surfaces are well preserved by our approach such as boundary curve of FLOWER, sharp edges in KNOT and CAD PART, ears of KITTEN, and holes in FERTILITY. All the results shown in Figure 12 are wireframe triangle meshes with a large variety in the number of vertices, e.g., MONKEY SADDLE with 121 vertices, and FERTILITY with 1857 vertices. Our approach is able to model wireframe polygonal meshes with discrete equivalence classes. Figure 13 shows four polygonal mesh results with different patterns, where the mesh $\mathbf{M}$ is initialized via mapping a 2D tessellation onto the input surface $\mathbf{P}$. Please refer to the accompanying video for visualization of these virtual results from different views.

Evaluating our computational framework. Figure 14 shows a running example of our computational framework. At the beginning, the wireframe mesh has a small cluster (K_v^n) of vertices and a small cluster (K_e^n) of edges due to the large prescribed tolerances (ω_v^n and ω_e^n). At each iteration, our framework gradually decreases the prescribed tolerances and performs clustering-and-optimization on the mesh vertices and edges to discover new clusters; see plots of K_v^n and K_e^n in Figure 14. Figure 15 shows that our approach allows users to specify their preference on reducing the number of distinct vertices or edges. This is achieved by setting values of ω_v^N and ω_e^N, where a large value of ω_v^N (ω_e^N) means a preference on reducing the number of distinct vertices (edges). When $\omega_e^N = 0$, it means our approach only reduce the number of distinct vertices but not edges. Despite this, the number of edge clusters is still smaller than the number of mesh edges, due to the tolerance ϵ_e; see the leftmost mesh in Figure 15. The same applies to the other extreme case of $\omega_v^N = 0$; see the rightmost mesh in Figure 15. In this experiment, we find that a smaller number (K_v^n) of vertex clusters make local shape features of the BUNNY more rounded, resulting a larger Hausdorff distance from the input surface; see Table I. In addition, we evaluate the effectiveness of the local optimization in our computational framework through an ablation study. In this study, we ran our approach with and without the local optimization, respectively, on three input surfaces. The results show that the local optimization can effectively reduce K_v and K_e without significantly increasing the computation time; see Table II for the statistics.

Statistics. Table I summarizes statistics of the results

TABLE I
STATISTICS AND TIMINGS. WE REPORT THE INPUT SURFACE, NUMBER OF VERTICES (N_v), NUMBER OF EDGES (N_e), NUMBER OF VERTEX CLASSES (K_v) AND EDGE CLASSES (K_e), AVERAGE SIZE OF VERTEX CLASSES (N_v/K_v), AVERAGE SIZE OF EDGE CLASSES (N_e/K_e), MAXIMUM AND MINIMUM OF THE SIZES ($\{s_v^i\}$) OF VERTEX CLASSES AND THE SIZES ($\{s_e^j\}$) OF EDGE CLASSES, RESPECTIVELY, NORMALIZED HAUSDORFF DISTANCE BETWEEN THE WIREFRAME MESH AND THE INPUT SURFACE, AND TIME TO GENERATE EACH WIREFRAME MESH RESULT.

Fig	Surface	N_v	N_e	K_v	K_e	N_v/K_v	N_e/K_e	$\{s_v^i\}$ max	$\{s_v^i\}$ min	$\{s_e^j\}$ max	$\{s_e^j\}$ min	Hausdorff distance	Time (min)
12	Flower	169	462	5	19	33.8	24.3	127	3	60	3	0.32%	0.07
	Monkey Saddle	121	320	27	37	4.5	8.6	28	1	51	1	0.23%	0.06
	Free Holes	382	1036	39	32	9.8	32.4	154	1	222	1	0.64%	1.54
	Vase	495	1479	62	65	8.0	22.8	157	1	82	1	0.43%	3.57
	Knot	506	1518	79	74	6.4	20.5	107	1	109	1	0.54%	2.63
	CAD Part	731	2193	153	120	4.8	18.3	143	4	148	1	0.29%	8.18
	Teddy	1353	4053	180	68	7.5	59.6	416	1	231	1	1.45%	33.67
	Kitten	1256	3768	149	64	8.4	58.9	597	1	201	1	1.32%	29.15
	Fertility	1857	5589	470	116	4.0	48.2	106	1	350	1	0.87%	39.22
13	Igloo	508	965	5	52	101.6	18.6	389	3	60	1	0.68%	1.93
	Arch	450	646	2	22	225.0	29.4	395	58	63	2	0.10%	1.01
	Pentagon	338	793	19	18	17.8	44.1	241	1	119	1	0.53%	1.15
	Flower	210	390	8	24	26.3	16.3	81	3	33	3	0.38%	0.22
15	Bunny	1001	2997	149	93	6.7	32.2	401	1	157	1	2.45%	3.27
				215	84	4.7	35.7	315	1	163	1	0.99%	15.85
				268	68	3.7	44.1	267	1	213	1	0.84%	14.45
				389	52	2.6	57.6	91	1	303	1	0.52%	14.32
				594	36	1.7	83.3	15	1	409	1	0.26%	11.74
19	Hyerbolic	116	206	4	24	29.0	8.6	68	4	68	1	0.61%	0.06
	Igloo	196	364	14	30	14.0	12.1	108	1	39	1	0.87%	0.25
	Duck	163	483	36	42	4.5	11.5	66	1	33	1	3.54%	0.37

TABLE II
STATISTICS OF THE RESULTS IN THE ABLATION STUDY ON THE LOCAL OPTIMIZATION.

Surface	Approach	N_v	N_e	K_v	K_e	N_v/K_v	N_e/K_e	Time (min)
Free Holes	global	382	1036	42	35	9.1	29.6	1.50
	local + global			**39**	**32**	9.8	32.4	1.54
CAD Part	global	731	2193	174	133	4.2	16.5	7.07
	local + global			**153**	**120**	4.8	18.3	8.18
Fertility	global	1857	5589	494	131	3.8	42.7	27.72
	local + global			**470**	**116**	4.0	48.2	39.22

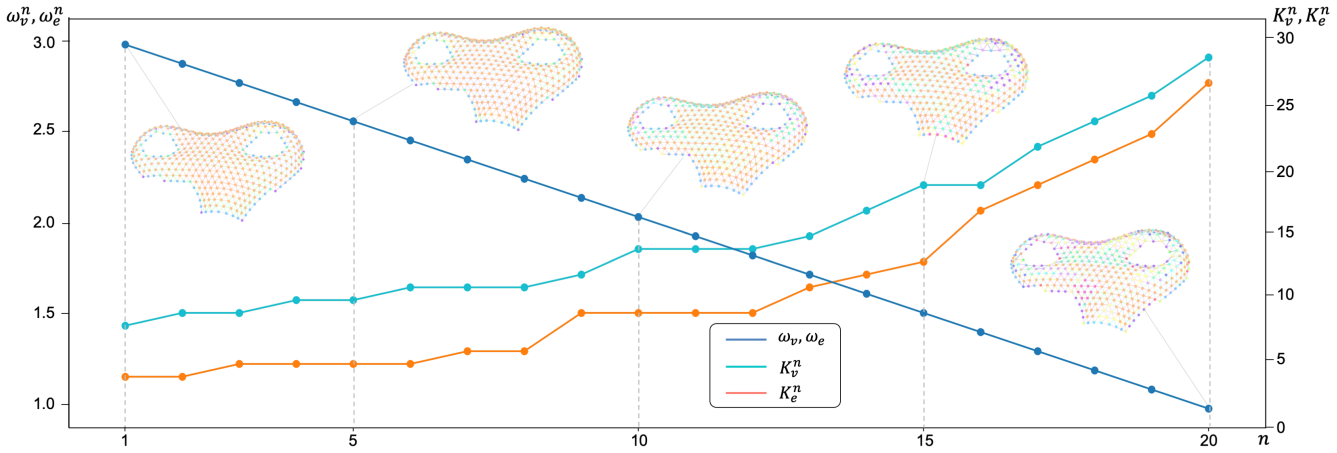


Fig. 14. Our computational framework iteratively clusters and optimizes the mesh to discover new clusters of vertices and edges (K_v^n, K_e^n) guided by the two prescribed tolerances (ω_v^n, ω_e^n) that are gradually decreased in each iteration.

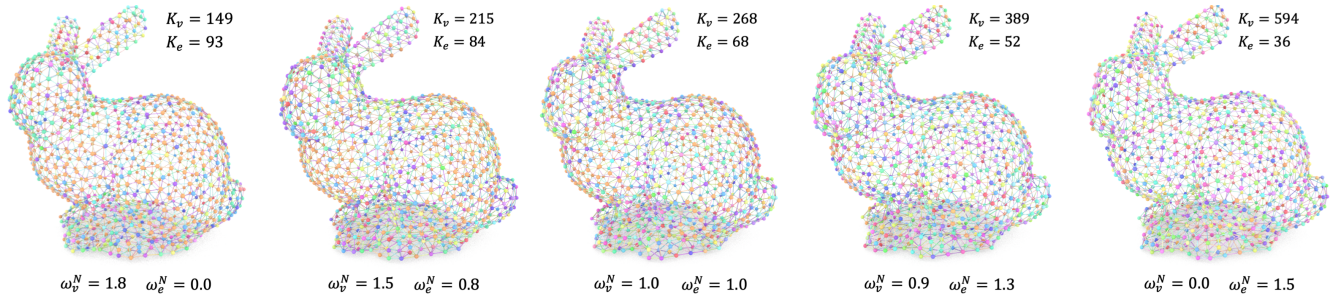


Fig. 15. Our approach allows users to specify their preference on reducing the number of distinct vertices or edges, by setting the values of ω_v^N and ω_e^N. We show the number (K_v) of vertex classes and the number (K_e) of edge classes beside each resulting mesh.

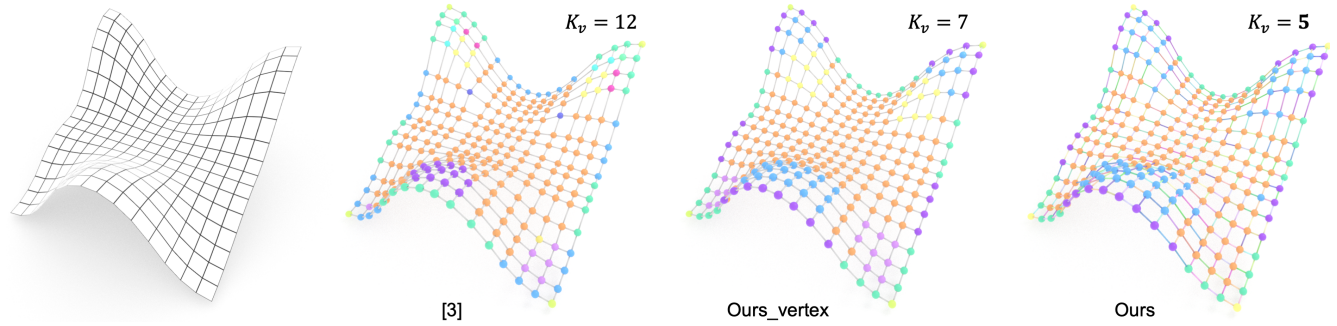


Fig. 16. Comparing [3] with our approach (Ours) as well as a variation of our approach (Ours_vertex) to model a quad mesh with discrete equivalence classes of vertices. The input surface is shown on the left and statistics of the three results are provided in Table III.

presented in the paper. For each result, both the sizes $\{s_v^i\}$ of vertex classes and the sizes $\{s_e^j\}$ of edge classes have a large variation. Large classes can have more than 100 elements while most smallest classes have a single element. Moreover, we find that a surface with more complex shape require a larger number of vertex/edge classes to approximate. For example, FLOWER in Figure 12 and DUCK in Figure 19 are both represented as triangles meshes with similar numbers of vertices and edges. Since DUCK has more complex shape than FLOWER, the average size of vertex classes are 4.5 vs 33.8 and the average size of edge classes are 11.5 vs 24.3. Our computational approach is efficient. Modeling a wireframe mesh with around 200 vertices takes less than 1 minute. For each result shown in the paper, we provide the Hausdorff distance between the modeled wireframe mesh $\mathbf{M}$ and the input surface $\mathbf{P}$, which is normalized by the diagonal length of the input surface $\mathbf{P}$'s bounding box. Please refer to the supplementary data for the 3D models of these meshes.

Comparison with [3]. Liu et al. [3] proposed an approach to model wireframe meshes with discrete equivalence classes of vertices, which is based on iteratively performing K-means clustering of vertices and global mesh optimization to reduce intra-cluster variance. Our approach is different from [3] in two aspects. First, we use a new computational framework with hierarchical clustering and local-global optimization. Second, we perform clustering-and-optimization on both mesh vertices and edges. For an ablation study, we prepare a variation of our approach that performs clustering-and-optimization on the mesh vertices only. For a fair comparison, we cluster the final results generated by the three approaches using our

TABLE III
STATISTICS OF THE THREE COMPARISON EXPERIMENTS.

Fig	Surface	Approach	N_v	N_e	K_v	K_e	N_v / K_v	N_e / K_e	Hausdorff distance
16	Arch Surf	[3]	256	480	12	/	21.3	/	0.08%
		Ours_vertex			7	/	36.6	/	
		Ours			**5**	66	51.2	7.3	
17	Saddle	[2]	89	230	17	/	5.2	/	0.03%
		Ours			**15**	33	5.9	7.0	
	LV	[2]	1036	2973	17	/	60.9	/	0.05%
		Ours			**12**	50	86.3	59.5	
	Dome	[2]	948	2682	41	/	23.1	/	0.07%
		Ours			**26**	61	36.5	44.0	
18	Bunny	[4]	392	958	374	9	1.0	106.4	0.10%
		Ours		1170	**278**	**6**	1.4	195.0	0.07%

hierarchical clustering with the same tolerance $\epsilon_v = 0.0872$. We set $\omega_v^N = 1.0$ and $\omega_e^N = 1.0$ in our approach and the variation. Figure 16 shows the results of running the three approaches on an architectural surface represented as a quad mesh. To approximate the input surface with the same Hausdorff distance, the variation of our approach requires 7 classes of vertices while the approach in [3] requires 12 classes of vertices, showing the effectiveness of our computational framework to reduce the number of distinct vertices; see Table III. In addition, our approach requires only 5 classes of vertices to approximate the input surface, demonstrating that clustering-and-optimizing mesh vertices and edges simultaneously can help to reduce the number of distinct vertices, for an input surface with a semi-regular pattern.

Comparison with [2]. Xiong et al. [2] proposed another approach to model wireframe meshes with discrete equivalence classes of vertices, which is based on iteratively performing K-medoids clustering of vertices and local mesh perturbation to reduce the number of clusters. Xiong et al. did not consider fabrication constraints (i.e., Equations 1 and 2) in their modeling approach. For a fair comparison, we also ignore the fabrication constraints in our approach. Moreover, we cluster the final results generated by both approaches using our hierarchical clustering with the same tolerance $\epsilon_v = 0.0872$. Figure 17 shows the results of running both approaches on three architectural surfaces, each of which is represented as a triangle mesh. To approximate each surface with the same Hausdorff distance, our approach requires fewer vertex classes K_v than [2]; see Table III. This comparison result shows the effectiveness of our local-global optimization scheme to reduce the number of distinct vertices.

Comparison with Zometool [4]. Zimmer et al. [4] proposed an approach to approximate an input surface using the Zometool construction set, which consists of nine struts of different lengths and a universal node with 62 slots. Figure 18 shows the results of running both approaches on the BUNNY model. The result generated by [4] is a polygonal mesh with both triangles and quads, which consists of 392 vertices and 968 edges. If we consider nodes with exactly the same slot insertion configuration as one node class, the result generated by [4] has 374 node (i.e., vertex) classes. In contrast, the result generated by our approach is a triangle mesh with 392 vertices

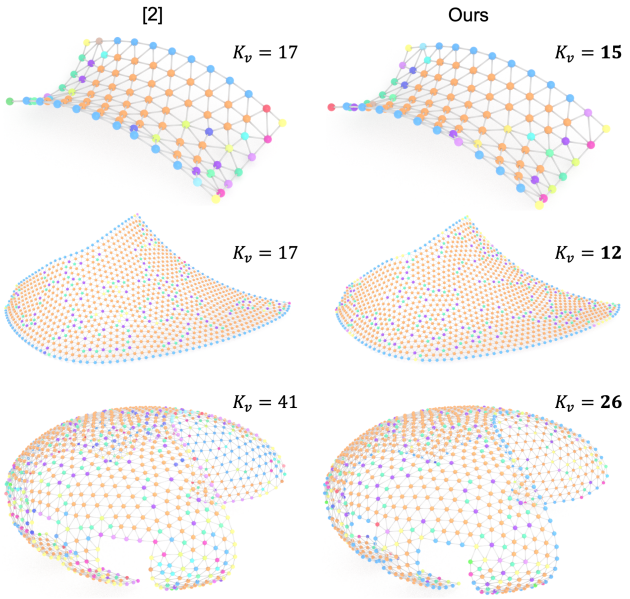


Fig. 17. Comparing [2] with our approach to model three architectural surfaces with discrete equivalence classes of vertices, where our approach generates a smaller number (K_v) of vertex classes for each surface. Statistics of the results are provided in Table III.

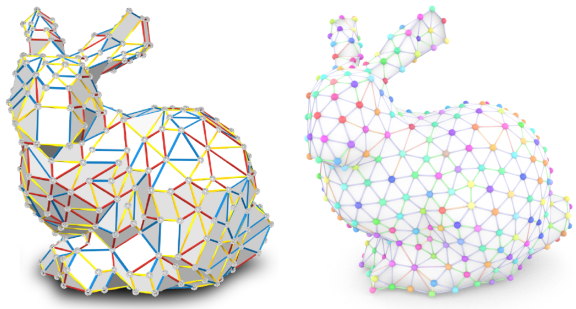


Fig. 18. Comparing (left) Zometool shape approximation [4] with (right) our approach. Our approach is able to better approximate the BUNNY model using a smaller number of vertex classes and a smaller number of edge classes; see Table III for the statistics.

and 1170 edges, where the vertices fall into 278 classes and the edges fall into 6 classes. This comparison experiment shows that our approach is able to approximate a 3D surface using a smaller number of vertex classes (278 vs 374) and a smaller number of edge classes (6 vs 9). Despite using a smaller number of templates, our modeled wireframe mesh approximates the input surface better. Table III shows that our modeled wireframe mesh has a smaller Hausdorff distance to the input surface. Figure 18 shows that our modeled wireframe mesh better preserves shape features such as the head and tail of BUNNY. One reason of our better performance is that our templates are computed for approximating a specific input shape while the templates in the Zometool construction set are designed to approximate a variety of shapes.

Fabricated prototypes. Our modeled wireframe meshes are ready for fabrication. We validate this by making three wireframe meshes modeled by our approach, i.e., HYPERBOLIC, IGLOO, and DUCK; see Figure 19. All the three physical wireframe structures share the same fabrication-related parameters described in Section III; i.e., the radius w of each rod is

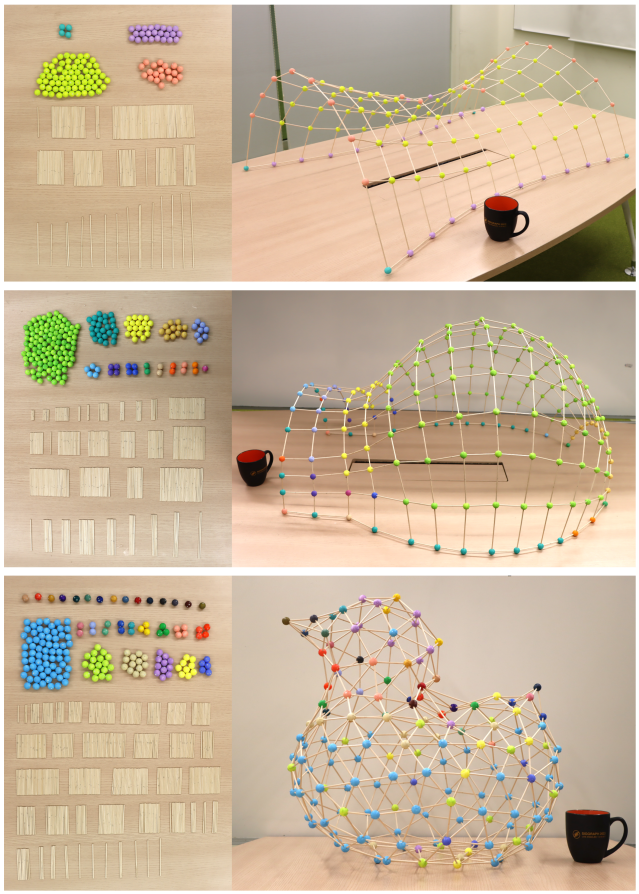


Fig. 19. We fabricate three of our modeled wireframe meshes with 3D printed nodes and precision-cut rods, from top to bottom, HYPERBOLIC, IGLOO, and DUCK. Fabricated nodes and rods of the same class are clustered together on the left. A mug is put beside each structure to show its real scale.

0.16cm, the radius R of each node is 0.9cm, and the depth $R-r$ of each cylindrical hole is 0.36cm. Based on this setting, each vertex angle θ should be greater than 33° according to Equation 1 and each edge length l should be larger than 1.8 cm according to Equation 2. We fabricate instances of template nodes by 3D printing them using Ultimaker S5 printer with tough PLA material. After 3D printing, we paint nodes in the same class using the same color. We fabricate instances of template rods by manually cutting wooden sticks with the computed lengths. Thanks to the discrete equivalence classes of edges, we are able to cut a bunch of wooden sticks of the same length together rather than cutting them one by one. We assemble nodes and rods by simply inserting each rod into the corresponding cylindrical hole in the node. The assembly order of nodes and rods are quite flexible since many of them have exactly the same shape (i.e., fall in discrete equivalence classes). Once assembled, each wireframe structure is stable purely based on node-rod joinery without using any glue [34], [35], and the shape of the structure is identical to that of its virtual counterpart. Please watch the accompanying video for the demo of assembling each structure. We provide the 3D models of the three wireframe structures as well as corresponding template nodes and rods in the supplementary data.

IX. Conclusion and Future Work

This paper studies a new problem of modeling wireframe meshes with discrete equivalence classes of vertices and edges. We propose a computational approach to solve the problem via iteratively clustering and optimizing a wireframe mesh. The key algorithmic components in our approach include hierarchical clustering of mesh vertices and edges according to prescribed tolerances, local mesh optimization to reduce the number of clusters of vertices and/or edges, and global mesh optimization to reduce intra-cluster variance while satisfying the fabrication constraints. Quantitative comparisons with three state-of-the-art approaches show that our approach has strengths in several aspects, including reducing the number of distinct vertices and edges, preserving the shape and features of an input surface, and determining the number of vertex classes and edge classes automatically.

Limitations and future work. Our work has several limitations that open up interesting directions for future research. First, our approach may have difficulty in modeling and fabricating wireframe meshes with too many vertices, although this may not be common in practice. This is because the time and storage complexity of hierarchical clustering is high. One possible way to address this limitation is to model a wireframe mesh part by part using a divide-and-conquer strategy. Second, the searching space of our local-global optimization is the geometry of the wireframe mesh only. Extending the approach to optimize both the mesh geometry and topology is an interesting research challenge. Lastly, our approach models wireframe meshes to fabricate them as a single layer space frame structure. Generalizing our approach to model and fabricate non-manifold space structures like general space truss structures [36] and reciprocal frame structures [37], [38] would be an interesting future work.

Acknowledgments

The authors would like to thank Yi Min Xie for sharing the quad mesh in Figure 16, Weidan Xiong for sharing the three triangle meshes in Figure 17, and Leif Kobbelt for providing information about the Zometool model in Figure 18.

References

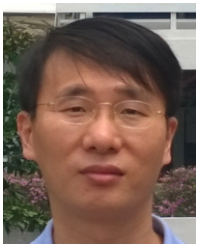
[1] T. T. Lan, "Space frame structures," in *Structural Engineering Handbook*, pp. 1–59, CRC Press LLC, 1999.
[2] W. Xiong, C. M. Cheung, P. V. Sander, and A. Joneja, "Rationalizing architectural surfaces based on clustering of joints," *IEEE Trans. Vis. & Comp. Graphics*, vol. 28, no. 12, pp. 4274–4288, 2022.
[3] Y. Liu, T.-U. Lee, A. Koronaki, N. Pietroni, and Y. M. Xie, "Reducing the number of different nodes in space frame structures through clustering and optimization," *Engineering Structures*, vol. 284, pp. 116016:1–116016:10, 2023.
[4] H. Zimmer, F. Lafarge, P. Alliez, and L. Kobbelt, "Zometool shape approximation," *Graphical Models*, vol. 76, no. 5, pp. 390–401, 2014.
[5] S. Mueller, S. Im, S. Gurevich, A. Teibrich, L. Pfisterer, F. Guimbretière, and P. Baudisch, "WirePrint: 3D printed previews for fast prototyping," in *Proc. ACM UIST*, pp. 273–280, 2014.
[6] R. Wu, H. Peng, F. Guimbretière, and S. Marschner, "Printing arbitrary meshes with a 5DOF wireframe printer," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 35, no. 4, pp. 101:1–101:9, 2016.
[7] Y. Huang, J. Zhang, X. Hu, G. Song, Z. Liu, L. Yu, and L. Liu, "FrameFab: Robotic fabrication of frame shapes," *ACM Trans. on Graph. (SIGGRAPH Asia)*, vol. 35, no. 6, pp. 224:1–224:11, 2016.

[8] R. Richter and M. Alexa, "Beam meshes," *Comp. & Graph.*, vol. 53, pp. 28–36, 2015.
[9] A. Jacobson, "RodSteward: A design-to-assembly system for fabrication using 3d-printed joints and precision-cut rods," *Comp. Graph. Forum (Pacific Graphics)*, vol. 38, no. 7, pp. 765–774, 2019.
[10] S. Chidambaram, Y. Zhang, V. Sundararajan, N. Elmqvist, and K. Ramani, "Shape structuralizer: Design, fabrication, and user-driven iterative refinement of 3D mesh models," in *Proc. ACM CHI*, pp. 663:1–663:12, 2019.
[11] Z. Wang, F. Kennel-Maushart, Y. Huang, B. Thomaszewski, and S. Coros, "A temporal coherent topology optimization approach for assembly planning of bespoke frame structures," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 42, no. 4, pp. 144:1–144:13, 2023.
[12] A. Koronaki, P. Shepherd, and M. Evernden, "Rationalization of freeform space-frame structures: Reducing variability in the joints," *International Journal of Architectural Computing*, vol. 18, no. 1, pp. 84–99, 2020.
[13] S. Dritsas, L. Chen, and L. Sass, "Small 3D printers / large scale artifacts - computation for automated spatial lattice design-to-fabrication with low cost linear elements and 3d printed nodes," in *Proc. of International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, pp. 821–831, 2017.
[14] J. Brütting, G. Senatore, and C. Fivet, "Design and fabrication of a reusable kit of parts for diverse structures," *Automation in Construction*, vol. 125, pp. 103614:1–103614:15, 2021.
[15] H. Zimmer and L. Kobbelt, "Zometool rationalization of freeform surfaces," *IEEE Trans. Vis. & Comp. Graphics*, vol. 20, no. 10, pp. 1461–1473, 2014.
[16] M. Singh and S. Schaefer, "Triangle surfaces with discrete equivalence classes," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 29, no. 4, pp. 46:1–46:7, 2010.
[17] M. Huard, M. Eigensatz, and P. Bompas, "Planar panelization with extreme repetition," in *Proc. Advances in Architectural Geometry 2014*, pp. 259–279, 2015.
[18] M. Bi, Y. Liu, T. Xu, Y. He, J. Ma, Z. Zhuang, and Y. M. Xie, "Clustering and optimization of nodes, beams and panels for cost-effective fabrication of free-form surfaces," *Engineering Structures*, vol. 307, pp. 117912:1–117912:12, 2024.
[19] C.-W. Fu, C.-F. Lai, Y. He, and D. Cohen-Or, "K-set tilable surfaces," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 29, no. 4, pp. 44:1–44:6, 2010.
[20] Y. Liu, T.-U. Lee, A. R. Javan, N. Pietroni, and Y. M. Xie, "Reducing the number of different faces in free-form surface approximations through clustering and optimization," *Computer-Aided Design*, vol. 166, pp. 103633:1–103633:12, 2023.
[21] Z.-Y. Liu, Z. Zhang, D. Zhang, C. Ye, L. Liu, and X.-M. Fu, "Modeling and fabrication with specified discrete equivalence classes," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 40, no. 4, pp. 41:1–41:12, 2021.
[22] T. Zhu, Z.-H. Xu, L. Liu, and X.-M. Fu, "Modeling with discrete equivalence classes of planar quads," *Comp. & Graph. (CAD/Graphics)*, vol. 115, pp. 404–411, 2023.
[23] R. Chen, P. Qiu, P. Song, B. Deng, Z. Wang, and Y. He, "Masonry shell structures with discrete equivalence classes," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 42, no. 4, pp. 115:1–115:12, 2023.
[24] D. Khan, C. Bohak, and I. Viola, "Dr. KID: Direct remeshing and k-set isometric decomposition for scalable physicalization of organic shapes," *IEEE Trans. Vis. & Comp. Graphics (IEEE VIS)*, vol. 30, no. 1, pp. 705–715, 2024.
[25] M. Eigensatz, M. Kilian, A. Schiftner, N. J. Mitra, H. Pottmann, and M. Pauly, "Paneling architectural freeform surfaces," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 29, no. 4, pp. 45:1–45:10, 2010.
[26] H. Zimmer, M. Campen, D. Bommes, and L. Kobbelt, "Rationalization of triangle-based point-folding structures," *Comp. Graph. Forum (Eurographics)*, vol. 31, no. 2, pp. 611–620, 2012.
[27] C. H. Lee, A. Varshney, and D. W. Jacobs, "Mesh saliency," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 24, no. 3, pp. 659–666, 2005.
[28] M. Dunyach, D. Vanderhaeghe, L. Barthe, and M. Botsch, "Adaptive remeshing for real-time mesh deformation," in *Eurographics - Short Papers*, pp. 29–32, 2013.
[29] M. Botsch and L. Kobbelt, "A remeshing approach to multiresolution modeling," in *Proc. Eurographics Symposium on Geometry Processing*, pp. 189–196, 2004.
[30] L. Liu, L. Zhang, Y. Xu, C. Gotsman, and S. J. Gortler, "A local/global approach to mesh parameterization," *Comp. Graph. Forum (SGP)*, vol. 27, no. 5, pp. 1495–1504, 2008.
[31] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: An overview," *WIREs Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.
[32] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3," 2010. url-http://eigen.tuxfamily.org.
[33] A. Jacobson, D. Panozzo, *et al.*, "libigl: A simple C++ geometry processing library," 2018. https://libigl.github.io/.
[34] R. Chen, Z. Wang, P. Song, and B. Bickel, "Computational design of high-level interlocking puzzles," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 41, no. 4, pp. 150:1 – 150:15, 2022.
[35] P. Song, "Interlocking assemblies: Applications and methods," in *Materials Today: Proceedings (International Conference on Additive Manufacturing for a Better World)*, vol. 70, pp. 78–82, 2022.
[36] K. J. Lee, R. Danhaive, and C. T. Mueller, "Spherical harmonic shape descriptors of nodal force demands for quantifying spatial truss connection complexity," *Architecture, Structures and Construction*, vol. 2, pp. 145–164, 2022.
[37] P. Song, C.-W. Fu, P. Goswami, J. Zheng, N. J. Mitra, and D. Cohen-Or, "Reciprocal frame structures made easy," *ACM Trans. on Graph. (SIGGRAPH)*, vol. 32, no. 4, pp. 94:1–94:13, 2013.
[38] P. Song, C.-W. Fu, P. Goswami, J. Zheng, N. J. Mitra, and D. Cohen-Or, "An interactive computational design tool for large reciprocal frame structures," *Nexus Network Journal*, vol. 16, pp. 109–118, 2014.

Pengyun Qiu was a research assistant in computer graphics laboratory, Singapore University of Technology and Design (SUTD). He received his Bachelor of Science degree from Nanjing University of Science and Technology in 2022 majoring in Information and Computing Science. His research interest is computer graphics.

Rulin Chen is a Postdoctoral Research Fellow in the Computer Graphics Lab, Singapore University of Technology and Design (SUTD), where he received his doctoral degree in 2024 under the supervision of Prof. Peng Song. Prior to joining SUTD, he received his bachelor's degree from Shantou University in 2020. His research interest is in computer graphics, with a focus on computational assemblies. He received SIGGRAPH Technical Papers Award Honorable Mention in 2022.

Peng Song is an Assistant Professor at the Pillar of Information Systems Technology and Design, SUTD since 2019. Prior to joining SUTD, Peng was a research scientist at EPFL, Switzerland. He received his PhD from Nanyang Technological University, Singapore in 2013, his Master's and Bachelor's degrees both from Harbin Institute of Technology, China in 2010 and 2007, respectively. His research interest is in computer graphics, with a focus on geometric modeling, computational design, and computational fabrication. He is an Associate Editor of Computers and Graphics (Elsevier) and Graphical Models (Elsevier). He received SIGGRAPH Technical Papers Award Honorable Mention in 2022 and Shape Modeling International Best Paper Award in 2024.



Ying He is an Associate Professor in the School of Computer Science and Engineering, Nanyang Technological University, Singapore. He received his Bachelor and Master degrees in Electrical Engineering from Tsinghua University, Master and Ph.D. degrees in Computer Science from Stony Brook University. His research interests fall into the general areas of visual computing and he is particularly interested in the problems which require geometric analysis and computation. He is an Associate Editor of Journal of Computational Visual Media (Springer) and Computer Graphics Forum (Wiley). He regularly serves on the Technical Program Committee for major research conferences in geometric modeling and computer graphics.