Vision, Modeling, and Visualization (2020)
J. Krüger, M. Niessner, and J. Stückler (Eds.)

A Compact Patch-Based Representation for Technical Mesh Models

Lars Kammann[1] Stefan Menzel[2] Mario Botsch [1,3]

[1]Bielefeld University [2]Honda Research Institute Europe [3]TU Dortmund University

Abstract

We present a compact and intuitive geometry representation for technical models initially given as triangle meshes. For CAD-like models the defining features often coincide with the intersection between smooth surface patches. Our algorithm therefore first segments the input model into patches of constant curvature. The intersections between these patches are encoded through Bézier curves of adaptive degree, the patches enclosed by them are encoded by their (constant) mean and Gaussian curvatures. This sparse geometry representation enables intuitive understanding and editing by manipulating either the patches' curvature values and/or the feature curves. During decoding/reconstruction we exploit remeshing and hence are independent of the underlying triangulation, such that besides the feature curve topology no additional connectivity information has to be stored. We also enforce discrete developability for patches with vanishing Gaussian curvature in order to obtain straight ruling lines.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ ***Mesh geometry models;***

1. Introduction

When creating virtual or physical objects cooperatively as a team (human-human or human-machine) different design targets of each actor are processed on different levels of geometric abstraction, e.g., surface modeling in CAD teams or triangulated mesh modeling in simulation teams. Unfortunately, in the low-level mesh domain the high-level design features of the CAD geometry are no longer accessible, making it difficult to communicate design manipulations and intentions. In the high-level CAD domain, the geometric shape is the primary design target, and the mesh model resulting from a tessellation engine is often of secondary concern, sometimes leading to rather coarse triangulations with poor element quality.

Automatic conversion between these two geometry representations is still a challenging topic, in particular the conversion from a triangle mesh to a higher-level CAD representation. We therefore aim for an intermediate geometry representation that reveals the high-level design intentions underlying the initial CAD geometry by analyzing a given input triangle mesh and extracting an as-compact-as-possible set of intuitive geometric parameters. This novel geometry representation on the one hand captures the major design features (like a CAD model) and on the other hand provides a high-quality tessellation (like a mesh model). Since the decoding of our representation into a triangle mesh incorporates automatic remeshing, the representation itself is agnostic to the low-level mesh connectivity.

A compact and sparse geometry representation is not just more intuitive for the user, who then has to deal with less but more intuitive parameters. It should also enable machine learning approaches to be trained from fewer examples, and thereby remedy the problem of obtaining a sufficient number of *high-quality* training data for geometric learning. Even intuitive surface editing by adjusting the model parameters is possible with our representation. However, we do not explore learning and manipulation in this paper, but instead focus on the compact model representation itself as well as the encoding and decoding methods.

Given an input triangle mesh that represents a technical CAD-like model, our encoding phase first segments the model into patches of (nearly) constant curvature. We achieve a robust segmentation even in the presence of coarsely tessellated models with poorly shaped triangles by observing that the patches of constant curvature consist of planes, cylinders, and spheres only [KO67]. Instead of computing and clustering (approximate) curvature values, we therefore detect the surface patches using a customized version of variational shape approximation [CSAD04, WK05]. The intersection curves between the extracted patches contain the defining feature curves of the geometric model and are encoded as Bézier curves of adaptive degree. The surface patches are encoded by storing their (constant) curvature values. Besides this geometric information, we only store the coarse patch connectivity, which we determine by aggressive mesh decimation [GH97].

During the decoding phase, we first read the coarse connectivity graph, sample the features curves from their Bézier representation, and generate a sufficiently dense high-quality triangulation through uniform remeshing [DVBB13]. We then numerically optimize the vertex positions to satisfy the stored per-patch curvature values through a combination of the FiberMesh approach [NISA07] and

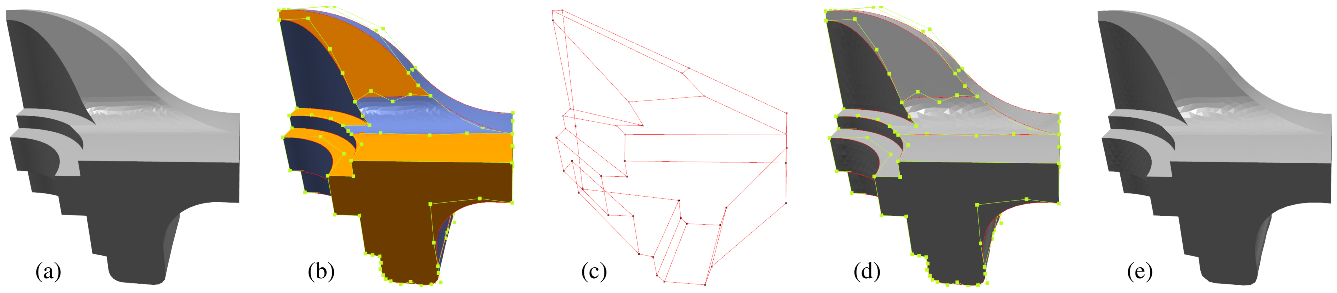

Figure 1: *Overview of our encoding and decoding framework. The initial mesh (a) is segmented into patches of constant curvature (b) leading to planes (orange), cylinders (blue), boundary curves (red), and Bézier control points (green). Aggressive decimation reveals the minimum connectivity graph of the patch corners (c). The decoding phase after loading, curve sampling, patch triangulation and remeshing (d) and after enforcing patch curvatures (e).*

the curvature-based optimization of Eigensatz et al. [ESP08, EP09]. The approximation error of our representation can be easily controlled by prescribing the geometric tolerance for the patch clustering and curve fitting steps.

The resulting sparse geometry representation, besides being more compact than previous approaches, consists of geometrically intuitive parameters only, which enables geometric modeling by manipulating a few high-level parameters, as we demonstrate in Section 5. In the following we first discuss related approaches (Section 2), before describing the encoding and decoding phases in Section 3 and Section 4, respectively.

2. Related Work

This work is inspired by the idea that many man-made objects can be described by a sparse collection of characteristic feature curves [SF98]. This idea was successfully combined with sketch-based modeling in the FiberMesh approach [NISA07], thereby fully defining an entire object. iWires [GSMCO09] equip the feature curves with additional information about their shape and relation to other curves, such that high-level structure editing can be performed without compromising feature and shape properties of the input model. Another approach is to analyze a given input shape by approximating it by a set of geometric primitives, which could be simple planar proxies [CSAD04] or more complex spherical or cylindrical proxies as well as rolling ball blends [WK05]. Those features can further be enhanced by detecting global symmetries [MGP06] or other relations between feature curves or surface patches [LWC*11].

Given extracted feature points and curves, many surface patches can intuitively be described by certain energy minimizing behaviors subject to boundary constraints [WW92]. Botsch and Kobbelt [BK04a] provide an intuitive framework for minimizing (the change of) surface area, surface curvature, or curvature variation. While a surface (or a triangle mesh) is typically described by assigning a 3D position to each *uv*-parameter (or each vertex), it can also be determined (up to rigid motion) by specifying first- and second-order derivative information (fundamental forms, metric, curvatures) at each parameter [ESP08, EP09]. Zhou et al. [ZWS11] extend this modeling method by altering the extracted feature curves [HPW05] and restoring the curvature values in the affected regions by a subsequent numerical optimization.

Developable surfaces, having vanishing Gaussian curvature everywhere, are attractive for physical manufacturing as they can be created by bending flat pieces. Solomon et al. [SVWG12] present a framework that enforces developability at every step during shape manipulation. A more recent definition of developability for triangle meshes, which drives a given mesh towards developable pieces, is provided by Stein et al. [SGC18].

When it comes to compact mesh representations, the approach of Lavoué and colleagues [LDB05, LDB07] is conceptually most similar to ours. They segment the given model into surface patches of constant curvature and extract subdivision curves and subdivision patches as a compressed geometry representation. Their decoder, however, does not exploit the (initially known) per-patch curvature values, but instead defines the surface solely through a subdivision process, which might lead to less accurate reconstructions (see the convex outer rim of the model shown in Fig. 17e of [LDB07]). Similar to the approach of Lavoué et al. [LDB05, LDB07], our method is specifically tailored for technical models that are composed of smooth surface patches with clean boundary curves. However, we do not put any requirements on the quality of the tessellation and can process input meshes with poorly shaped triangles.

Even though our method converts a given triangle mesh into a more compact representation, mesh compression is not our primary goal. While compression approaches might accept lossy compression of geometry (vertex positions), they typically require *lossless* compression of mesh connectivity [MLDH15], with very few exceptions: Szymczak et al. [SRK02] and Attene et al. [AFSR03] change the surface triangulation to achieve better compression rates for the connectivity encoding [Ros99, TG98, AD01]. Our approach, in stark contrast, aims to be agnostic to the underlying triangulation by completely replacing the input tessellation with a high-quality isotropic remesh. While our method also leads to a certain compression effect, our main goal is to represent the input model with few and *geometrically intuitive* parameters.

Our method consists of two algorithmic parts: *encoding* a given triangle mesh into our novel representation and *decoding* the resulting sparse geometry representation into a triangle mesh. Figure 1 visualizes the main steps of our method.

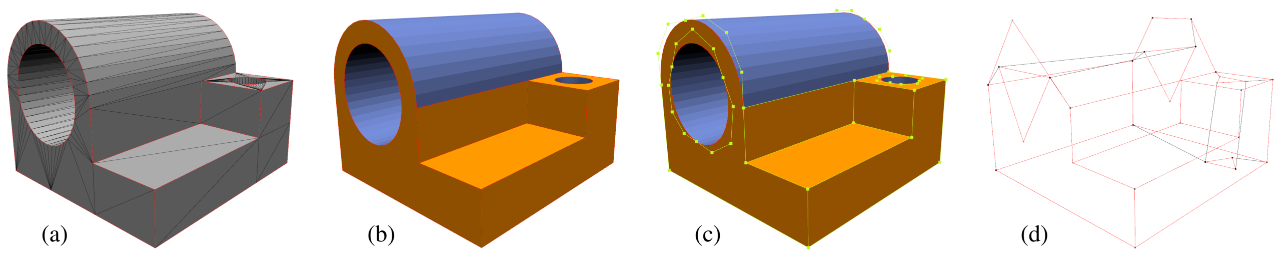

Figure 2: *Encoding pipeline: (a) Initial mesh with sharp feature edges. (b) Segmentation into constant curvature patches (planes: orange, cylinders: blue, boundary curves: red). (c) Bézier control points for feature curves (green). (d) Minimum connectivity graph of corner vertices.*

3. Encoding

The encoding phase starts by extracting sharp feature edges and vertices (Section 3.1), which then constrain the subsequent steps, e.g., the initial seeding for the segmentation algorithm. Then patches of (nearly) constant curvature are extracted based on customized variational shape approximation (Section 3.2). Afterwards feature corners, i.e., vertices where more than two patches meet, are extracted and Bézier curves are fitted to the feature curves connecting two corner vertices along patch boundaries (Section 3.3). Finally, the mesh is aggressively decimated, leaving only the connectivity graph of the corner vertices in form of a general polygon mesh (Section 3.4). The final sparse representation stores the positions of feature corners, the Bézier representation of the feature curves, the (constant) mean and Gaussian curvature per patch, and the connectivity graph of corner vertices.

3.1. Sharp Feature Extraction

In a pre-processing step we detect sharp feature edges based on the dihedral angle between their adjacent faces. We define an edge to be a feature edge if the normal vectors of its two adjacent triangles form an angle higher than a given threshold. This threshold can be chosen rather loosely to decrease the chance of false positives, which would later lead to unnecessary patch boundaries and a less compact encoding.

The extracted feature edges typically are (a super-set of) the feature curves between the piecewise-smooth surface patches (see Figure 2a). We detect these edges in order to exclude them from the later curvature computation and clustering, since curvature information is not well-defined on feature edges.

Unlike [LDB07] we do not need to enrich the input mesh based on detected feature edges in order to improve the robustness of curvature computation, since we do not compute curvatures (see below). Instead, we only use the extracted sharp feature edges to guide the initial seeding of the segmentation algorithm.

3.2. Segmentation

Given the previously extracted feature edges we now segment the mesh into patches of (nearly) constant curvature. In contrast to Lavoué [LDB07], however, we avoid the explicit computation of per-vertex or per-face curvature values, since those computations of second-order derivative information become unreliable for poorly tessellated models and therefore require sophisticated preprocessing/enrichment, which in turn might alter the true curvature values.

We instead observe that patches of constant mean and Gaussian curvature consist of planes, cylinders, and spheres only, as discussed by Klotz and Ossermann [KO67]. Those primitives can be very robustly detected based on positional information only. To this end, we adopt the popular hybrid variational shape approximation [WK05], which is a variant of Lloyd's clustering and consists of the two alternating phases *partitioning* and *fitting*. Given an input mesh $\mathcal{M}$, a partitioning $\mathcal{R}$ into regions $\mathcal{R}_i$ is computed. Each region $\mathcal{R}_i$ contains a set of triangles $\{t_j\}$ with barycenters $\{\mathbf{g}_j\}$. Each region is approximated by a shape proxy, which may be one of the following primitives:

- Planes $P_i = (\mathbf{x}_i, \mathbf{n}_i)$, defined by the region's barycenter $\mathbf{x}_i$ and an average normal vector $\mathbf{n}_i$,
- Spheres $S_i = (\mathbf{c}_i, r_i)$, defined by center $\mathbf{c}_i$ and radius r_i,
- Cylinders $C_i = (\mathbf{x}_i, \mathbf{d}_i, r_i)$, defined by a point $\mathbf{x}_i$ on the cylinder axis, the axis direction $\mathbf{d}_i$, and the radius r_i.

Thus the full proxy set is defined as $\mathcal{P} = \{P_i\} \cup \{S_i\} \cup \{C_i\}$. We omit the rolling-ball blends of [WK05] since these do not have constant curvature. The total approximation error is defined as

$$E(\mathcal{R}, \mathcal{P}) = \sum_i E(\mathcal{R}_i, \mathcal{P}_i),$$

where $\mathcal{P}_i$ is the best-fitting shape proxy, i.e., the optimal plane P_i, sphere S_i, or cylinder C_i that minimizes the fitting error within the region $\mathcal{R}_i$. If not otherwise specified, we always use the $\mathcal{L}^{2,1}$ metric [CSAD04]:

$$E(\mathcal{R}_i, \mathcal{P}_i) = \sum_{t_j \in \mathcal{R}_i} |t_j| \cdot \left\| \mathbf{n}(t_j) - \mathbf{N}_i(t_j) \right\|^2, \tag{1}$$

with $\mathbf{n}(t_j)$ being the normal vector of triangle t_j and $|t_j|$ its area. $\mathbf{N}_i(t_j)$ denotes the normal of proxy $\mathcal{P}_i$ evaluated at triangle t_j, which is the average normal $\mathbf{n}_i$ for planes, $\mathbf{g}_j - \mathbf{c}_i$ for spheres, and $\mathbf{g}_j - \pi_{C_i}(\mathbf{g}_j)$ for cylinders (where $\pi_{C_i}(\mathbf{g}_j)$ is the projection of $\mathbf{g}_j$ onto the axis of the cylinder C_i). The direction of $\mathbf{N}_i$ might have to be flipped to minimize the approximation error. Since (1) does not consider positions, it cannot distinguish parallel planes. For our region growing, however, it is fully sufficient though.

We use the same robust least-squares method [Pra87] for fitting spheres and cylinders as in [WK05], with the exception that we do not use the minimum curvature directions to compute the cylinder axis $\mathbf{d}_i$, as there may not be any non-feature vertices in a given region due to coarse/poor triangulation. Instead we compute the cylinder axis $\mathbf{d}_i$ as the eigenvector corresponding to the smallest eigenvalue of the normal covariance matrix

$$\sum_j |t_j| \cdot \mathbf{n}(t_j)\, \mathbf{n}(t_j)^\top .$$

Weighting by triangle areas ensures independence of the underlying triangulation.

For *spheres*, the fitting is done by minimizing

$$E(A,B,C,D,E) = \sum_j f\left(\mathbf{g}_j\right)^2 |t_j| \qquad (2)$$

under the constraint $B^2 + C^2 + D^2 - 4AE = 1$ and with

$$f(x,y,z) = A(x^2+y^2+z^2) + Bx + Cy + Dz + E = 0$$

being the implicit sphere representation.

For *cylinders*, the barycenters $\mathbf{g}_j$ are projected onto the plane P passing through the origin with normal vector $\mathbf{d}_i$, resulting in projected positions $\pi(\mathbf{g}_j)$. Then a circle is fitted to these projected barycenters by minimizing

$$E(A,B,C,D) = \sum_j f\left(\pi(\mathbf{g}_j)\right)^2 |t_j| \qquad (3)$$

under the constraint $B^2 + C^2 - 4AD = 1$ and with

$$f(x,y) = A(x^2+y^2) + Bx + Cy + D$$

being the implicit circle representation.

Minimizing (2) or (3) results in solving a 5×5 (respectively 4×4) general eigensystem by using Lagrange multipliers, with the solution being the eigenvector corresponding to the smallest eigenvalue. The primitive parameters can then be computed as

$$\mathbf{c}_i = -\frac{1}{2A}\left(B,C,D\right)^\top \quad \text{and} \quad \mathbf{r}_i^2 = \|\mathbf{c}_i\|^2 - \frac{E}{A}$$

for spherical proxies and

$$\mathbf{x}_i^\perp = -\frac{1}{2A}\left(B,C\right)^\top \quad \text{and} \quad \mathbf{r}_i^2 = \|\mathbf{c}_i\|^2 - \frac{D}{A}$$

for cylindrical proxies.

The complete segmentation process is conceptually similar to k-means clustering and consists of the following four steps:

1. **Initial seeding**: We use the extracted sharp feature edges (see Section 3.1) to guide the initial seeding of the segmentation. We take one random triangle out of every fully enclosed region as initial seed. If there are no enclosed regions we start with two random triangles, because at least two regions are needed to extract any boundaries. Each seed is initially approximated by a plane P_i.
2. **Region growing**: Starting from the seed triangles, new regions are grown in a breadth-first manner by considering all unconquered triangles incident to a region boundary and adding the triangle with least approximation error (1) to its best-fitting region. The region proxies $\mathcal{P}_i$ are kept fixed during the region growing, which is implemented through a priority queue.
3. **Fitting primitives**: Once all triangles have been assigned to a region $\mathcal{R}_i$, we fit planes, spheres, and cylinders to all triangles $t_j \in \mathcal{R}_i$. Each region is then assigned the primitive $\mathcal{P}_i$ that results in the least approximation error (1).
4. **New seeding**: From each region the best-fitting triangle according to its approximation error (see (1)) is detected. These triangles serve as new seeds for the next region growing phase. Furthermore, the worst-fitting triangle out of all mesh triangles is extracted. If the segmentation has settled in a local minimum, this triangle is added as a new seed, initially approximated by a new plane P_i.

Steps 2–4 are repeated until convergence. Convergence is reached if the error of the worst triangle (Step 4) is less than a user-defined threshold or if a given maximum number of iterations is reached.

For increased numerical robustness, the curvature values H_i and K_i (mean and Gaussian curvature) associated to the region R_i are not computed based on the (potentially poor) triangulation, but instead derived from the curvature of the fitted primitive, i.e., $H_i = K_i = 0$ for planes, $H_i = \frac{1}{r_i}$ and $K_i = \frac{1}{r_i^2}$ for spheres, and $H_i = \frac{1}{2r_i}$ and $K_i = 0$ for cylinders. Figure 2b visualizes an exemplary segmentation of a typical CAD model.

Unlike [WK05] we do not use the fitted primitives to describe/encode the given mesh. Instead, we fit Bézier curves to the boundaries between the extracted regions and use the curvature values of the primitive as new geometry representation. Reconstructing these curvature values in the least squares sense then distributes the approximation error evenly over the surface patches.

3.3. Corner vertices and boundaries

We denote by $\mathcal{C}$ the minimal set of *corner vertices* that are needed to describe the patch connectivity of the input mesh. Any vertex where more than two different patches meet is considered a corner vertex. We then trace the edges along each patch boundary to extract a boundary set $\mathcal{B}$. Each boundary consist of an edge strip connecting two corner vertices.

We then fit a Bézier curve $B^n(s)$ to each boundary curve in an adaptive least squares manner. To this end we uniformly sample the chain of edges defining the boundary curve, leading to positions $\mathbf{x}_1, \ldots, \mathbf{x}_N$, and use their (normalized) discrete arc lengths long the edge chain as parameter values t_i. We then determine the control points $\mathbf{c}_j$ by a least squares fit, i.e., by minimizing

$$\min_{\mathbf{c}_2,\ldots,\mathbf{c}_{n-1}} \frac{1}{N} \sum_{i=1}^{N} \left\| \sum_{j=1}^{n} \mathbf{c}_j B_j^n(t_i) \; - \; \mathbf{x}_i \right\|^2$$

The two corresponding corner vertices define the end-points of the Beziér curve and therefore determine the first and last control point $\mathbf{c}_1$ and $\mathbf{c}_n$, respectively. We determine the required polynomial degree n in an adaptive manner: We start from degree $n = 1$ and increase the polynomial degree n until the root-mean-square (RMS) approximation error is below a user defined threshold. To avoid oscillations due to high-degree polynomials, we split the boundary curve if the degree would increase beyond 4. When splitting the boundary the vertex with the largest error serves as new corner vertex, thereby producing two new boundaries.

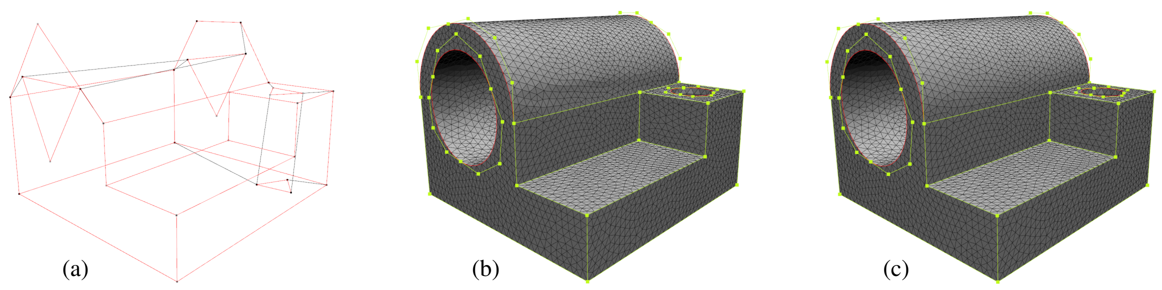

Figure 3: *Decoding pipeline: Starting from the minimal connectivity graph (a) we triangulate its faces and apply uniform remeshing (b), before optimizing the vertex positions to meet the stored per-patch curvature values (c) Note the vanishing of the dent in the top cylindrical part from (b) to (c).*

Note that more accurate boundary fits could be achieved by (i) using rational Bézier curves, since those can represent circular arcs and cone sections exactly, and (ii) interleaving fitting with parameter correction of the t_i. However, we found our simple least squares fits to be sufficiently accurate for our experiments.

There may remain some boundaries between patches without any corner vertices along them, e.g., a patch fully enclosed by another patch. In such a case we split this boundary into three parts: First, we fit a Bézier curve to the boundary while using a random vertex as start point *and* end control point. We then sample the Bézier curve at $s = \frac{1}{3}$ and $s = \frac{2}{3}$ and find the vertices on the boundary curve with the smallest distance. Finally, these two vertices together with the initial random vertex form new corner points with three boundaries in between them. These new boundaries are also subject to the splitting mentioned above. Figure 2c visualizes the resulting Bézier control points for approximating the feature/boundary curves.

3.4. Decimation and final representation

As the final step of the encoding a minimal connectivity graph of the corner vertices is computed. We utilize aggressive decimation following the approaches of Garland and Heckbert [GH97] and Kobbelt et al. [KCS98]. We preserve the boundary curve connectivity by only collapsing edges that have both vertices either on a common boundary or not on a boundary. Corner vertices remain fixed. The result is a minimum-connectivity triangle mesh. We further reduce the connectivity graph by deleting any non-boundary edges when the deletion does not destroy the mesh topology, since we can always re-triangulate the mesh during decoding. An example of a resulting minimal *polygon* mesh is shown in Figure 2d. The final sparse representation stores the positions of the corner vertices, the Bézier control points (excluding the corner positions), the (constant) mean and Gaussian curvature per patch, and the polygonal connectivity graph of the corner vertices.

4. Decoding

In the decoding phase we first load the corner vertices, their connectivity graph, and the per-boundary Bézier curves from a file. The boundary curves are then uniformly sampled according to their Bézier representation, leading to an initial coarse polygon mesh with very high face degree. Those polygons are subsequently triangulated and remeshed, providing enough vertices per patch to approximate the geometry (Section 4.1). The correct geometry is finally reconstructed by optimizing the positions of the inner patch vertices such that their curvature values meet the prescribed per-patch curvatures (Section 4.2). As a final polishing we enforce discrete developability to obtain straight ruling lines in patches with zero Gaussian curvature [SGC18] (Section 4.3).

4.1. Sampling, triangulation, and remeshing

Starting with the connectivity graph of the corner vertices (Figure 3a), the boundary Bézier curves are uniformly sampled (using the stored control points), thereby refining the edges of the connectivity graph to meet the geometry of the original boundary curves. The resulting high-degree polygons of the connectivity graph are then triangulated by minimizing dihedral angles and surface area, following the approach of Liepa [Lie03]. A uniform isotropic remeshing [BK04b] yields enough degrees of freedom (i.e., mesh vertices) to approximate the original per-patch geometry through high-quality close-to-equilateral triangles. Figure 3b visualizes a resulting refined triangle mesh.

While we let the user control the target edge length, one could also derive the required per-patch edge length from a prescribed error tolerance and the known per-patch curvature values, using the adaptive isotropic remeshing of [DVBB13].

4.2. Curvature Optimization

Given the initial geometry approximation we now solve for the positions $\mathbf{x}_k$ of the inner vertices (not lying on a feature/boundary curve) such that the prescribed curvature values H_i and K_i of each patch region $\mathcal{R}_i$ are met (see Section 3.2). We employ the FiberMesh approach [NISA07] to enforce the mean curvature values because of its robustness and fast convergence. This involves to minimize the difference between the current per-vertex Laplacian $\Delta\mathbf{x}_k$ and a target Laplacian, which itself is computed as the mean curvature normal based on the prescribed per-patch mean curvature H_i and the vertex normal $\mathbf{n}(\mathbf{x}_k)$:

$$\min_{\{\mathbf{x}_k\}} \sum_i \sum_{\mathbf{x}_k \in \mathcal{R}_i} \left\| \Delta\mathbf{x}_k - 2H_i\mathbf{n}(\mathbf{x}_k) \right\|^2 , \tag{4}$$

	initial				encoded				time	time	Hausdorff
	#V	#F	#int	#float	#V	#F	#int	#float	encode	decode	error
Joint	221	446	1338	663	30	18	254	237	13 ms	27 s	$1.1 \cdot 10^{-3}$
Oblong	422	840	2520	1266	73	46	658	518	58 ms	16 s	$2.3 \cdot 10^{-3}$
Anchor	519	1050	3150	1557	66	40	562	517	59 ms	8.4 s	$1.3 \cdot 10^{-3}$
Pinion	650	1300	3900	1950	126	66	1130	816	143 ms	7.9 s	$1.2 \cdot 10^{-3}$
Coupling	1841	3714	11142	5523	188	98	1675	2075	46 ms	13 s	$3.0 \cdot 10^{-3}$
Fandisk	6475	12946	38838	19425	40	22	362	389	746 ms	8.3 s	$7.7 \cdot 10^{-3}$

Table 1: *Mesh complexity (as numbers of vertices and faces) and memory consumption (in terms of numbers of 32-bit `int`-indices and `float`-coordinates) of the initial and the encoded models, the computation time for encoding and decoding, as well as the bounding-box-relative Hausdorff error between the initial and encoded/decoded models.*

We solve this minimization problem by decoupling the vertex positions $\mathbf{x}_k$ and their normal vectors $\mathbf{n}(\mathbf{x}_k)$ in an alternating optimization approach [NISA07].

We enforce the Gaussian curvature by adopting the method of [ESP08, EP09]. The principal curvatures κ_1, κ_2 of a vertex $\mathbf{x}_k$ are computed using the curvature tensor method of [CSM03]. We then minimize the difference between the current Gaussian curvature $K(\mathbf{x}_k) = \kappa_1\kappa_2$ and the target per-patch Gaussian curvature K_i, for all regions $\mathcal{R}_i$ of all types (planes, cylinders, spheres):

$$\min_{\{\mathbf{x}_k\}} \sum_i \sum_{\mathbf{x}_k \in \mathcal{R}_i} \left\| K(\mathbf{x}_k) - K_i \right\|^2 . \tag{5}$$

To increase the robustness of the geometry reconstruction we split the optimization into two parts: We first minimize (4) to obtain a piecewise smooth surface. In the second step we optimize for mean curvature (4) *and* Gaussian curvature (5) simultaneously. This successfully enforces the curvature values of the patch (see Figure 3c).

4.3. Developability Optimization

In many industrial applications it is desirable to have a model made from developable pieces, since those that can be created by smoothly bending planar surface pieces (e.g., a piece of paper) without stretching or shearing and thus are easy to manufacture.

For developable patches, which are identified by vanishing Gaussian curvature $K_i = 0$, we enforce discrete developability in a post-process based on the method of Stein et al. [SGC18]. Since flattenability alone is not sufficient to ensure easy fabrication, e.g., a crumbled piece of paper, Stein and colleagues enforce straight ruling lines passing through each surface point. According to their definition, a surface is discrete developable if every vertex star is a hinge, i.e., the adjacent faces of a vertex can be partitioned into two edge-connected regions over which the face normals $\mathbf{n}(t_j)$ are constant. This alignment can be quantified by the smallest eigenvalue λ_k of the per-vertex normal covariance matrix:

$$\sum_{t \ni \mathbf{x}_k} \theta_{t,k} \mathbf{n}(t)\, \mathbf{n}(t)^{\mathsf{T}} ,$$

where $\theta_{t,k}$ denotes the opening angle of triangle t at vertex k. Developability is then measured by summing over all λ_k. Minimizing this measure by optimizing vertex positions yields piecewise

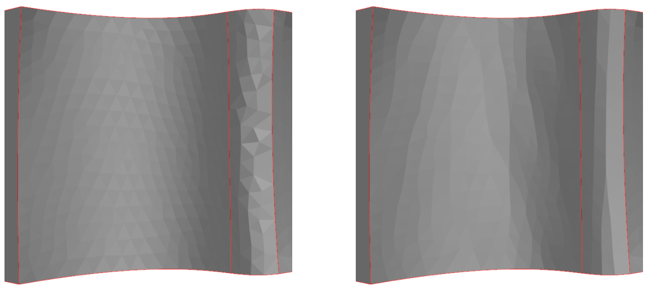

Figure 4: *Enforcing per-patch discrete developability. Left: Triangle mesh after curvature-based optimization. Right: Piecewise discrete developable surface with straight(ened) ruling lines.*

developable surfaces (see Figure 4). Starting from the curvature-optimized mesh (Section 4.2) considerably improves the robustness of this rather sensitive nonlinear minimization.

5. Results

We have tested our algorithm on several technical CAD-like models while measuring reconstruction error as well as compression rate. For better comparison we scale all input meshes to a bounding box with unit diagonal. If not stated otherwise we set the maximum boundary error (see Section 3.3) to $5 \cdot 10^{-4}$.

Figure 5 visualizes our encoding/decoding on different models (without the developability optimization of Section 4.3). There often is virtually no visual difference between the initial and reconstructed model. On some model cylindrical regions are even better approximated as in the initial input mesh—an effect that we can control through the user-prescribed approximation tolerance. We are able to achieve a high compression rate paired with low Hausdorff error, as shown in Table 1. Real-world applications might require a smaller error tolerance, which can easily be controlled by the user. As also shown in the table, our encoding results are well below one second, while the decoding, due to the involved remeshing and nonlinear optimization, might take from 8 to 30 seconds (measured on an 8-core Intel Xeon 3.6 GHz with 8 GB RAM).

Regarding the encoding part, we achieve comparable results to similar approaches like the method proposed by Lavoué et

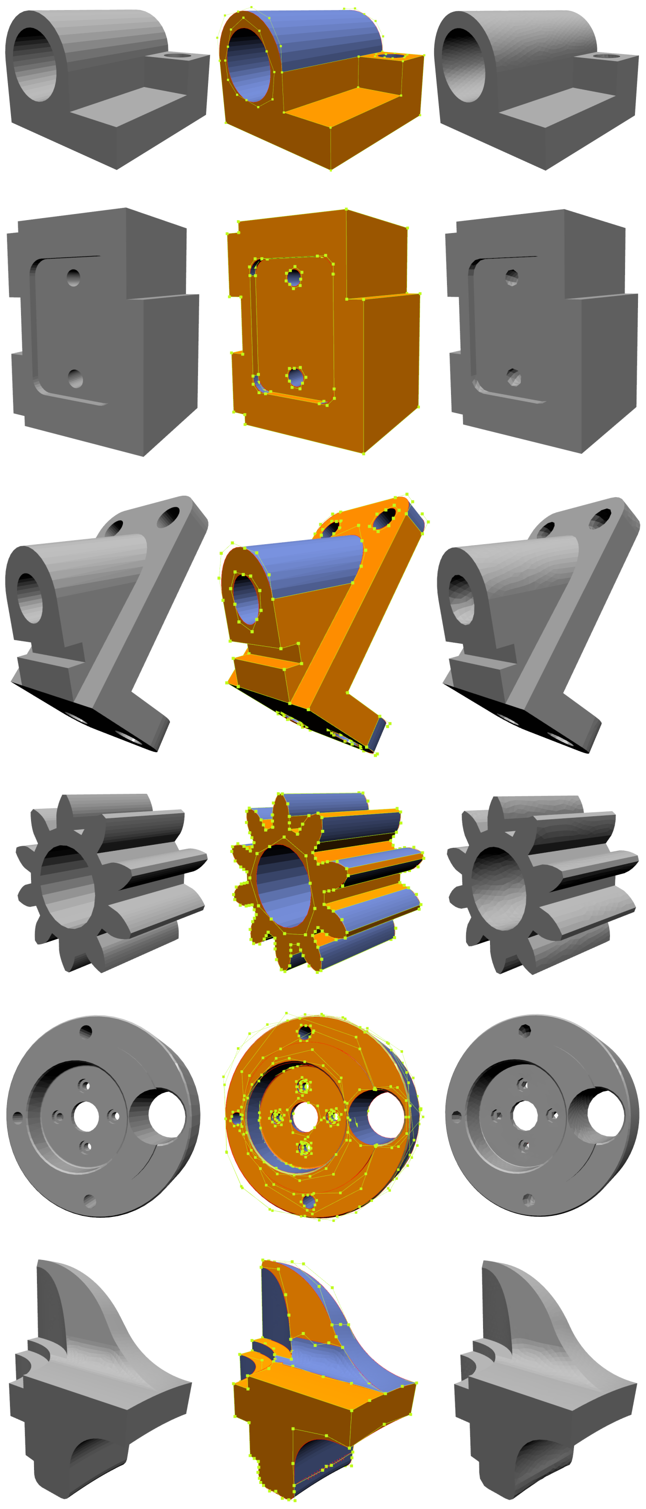

Figure 5: *Encoding and decoding for different models, showing for each model the input mesh (left), segmented patches (center), and reconstructed model (right). Model names from top to bottom: Joint, Oblong, Anchor, Pinion, Coupling, Fandisk.*

al. [LDB05]. While their coarse representation of the Fandisk model contains 75 vertices and 89 faces (see Fig. 10 of their paper), we need only 40 vertices and 22 faces. Since their method is targeted toward compact storage, they further compress connectivity and geometry of this coarse mesh, thereby achieving impressive compression rates. Our goal is not a compact representation in terms of required bits per vertex, but in terms of the number of mesh elements and geometric parameters. Hence, we did not implement further compression of our representation, but leave that as potential future work. Another difference to Lavoué et al. [LDB05] is that our method does not require non-feature vertices in patch interiors, since we do not directly cluster vertices based on their curvature. Therefore we are able to robustly handle even poorly triangulated meshes, as is often the case for tessellated CAD models, without requiring a dedicated mesh enrichment as in [LDB05]. This problematic case is depicted in Figure 6.

6. Conclusion

We presented a method for analyzing a given input triangle mesh and encoding it into a compact and intuitive sparse geometry representation. We robustly detect patches of constant curvature based on the hybrid variational shape approximation technique [WK05]. Boundaries between the resulting patches are extracted and encoded as Bézier curves of adaptive degree. Reconstruction is as simple as sampling the Bézier curves, triangulating and remeshing the sparse connectivity graph, and solving for the inner vertex positions. The approximation quality is easily controlled by prescribing an approximation tolerance.

For future work, the extension to patches of non-constant curvature is enticing. The segmentation process can easily be extended to other simple primitives like cones or tori. It will be interesting (and challenging) to investigate patches of harmonic surface distribution (e.g., with $\Delta H = 0$, being conceptually similar to clothoid curves).

In terms of future applications, we want to explore the potential of our sparse representation for user-controlled interactive shape manipulation as well as for geometric learning approaches, since we expect both applications to benefit from our sparse representation with geometrically intuitive parameters.

Acknowledgements

Lars Kammann gratefully acknowledges the financial support from Honda Research Institute Europe (HRI-EU). The authors are grateful to Moritz Kassmann, Bert Jüttler, and Ulrich Reif for inspiring discussions about constant-curvature surfaces.

References

[AD01] ALLIEZ P., DESBRUN M.: Valence-driven connectivity encoding for 3d meshes. In *Computer Graphics Forum* (2001), vol. 20, pp. 480–489. 2

[AFSR03] ATTENE M., FALCIDIENO B., SPAGNUOLO M., ROSSIGNAC J.: Swingwrapper: Retiling triangle meshes for better edgebreaker compression. *ACM Trans. Graph. 22*, 4 (2003), 982–996. 2

[BK04a] BOTSCH M., KOBBELT L.: An intuitive framework for real-time freeform modeling. *ACM Trans. Graph. 23*, 3 (2004), 630–634. 2

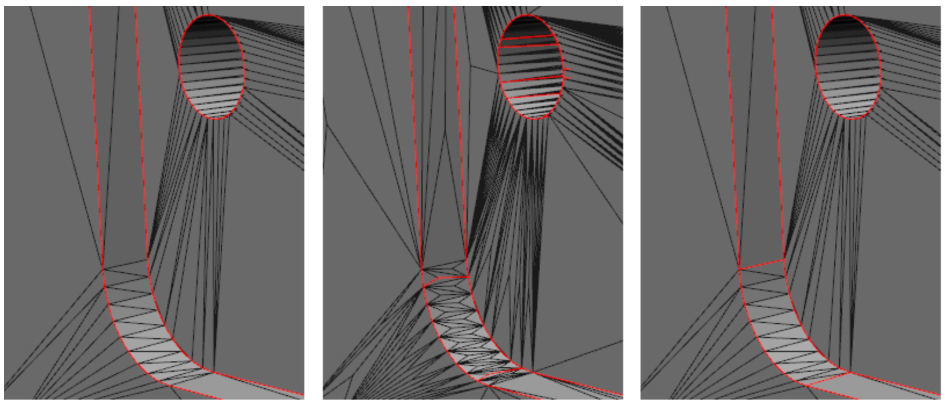

Figure 6: *Enriching the mesh by splitting faces (proposed in [LDB05]) leads in cylindrical regions to edges where the opposite vertices have zero curvature. Finding a real boundary edge based on curvature values is difficult. Our method does not rely on vertex curvature and is therefore suitable even for poorly triangulated meshes. From left to right: initial mesh, segmentation using [LDB05], our segmentation.*

[BK04b] BOTSCH M., KOBBELT L.: A Remeshing Approach to Multiresolution Modeling. In *Proceedings of Eurographics Symposium on Geometry Processing* (2004), pp. 185–192. 5

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM Trans. Graph. 23*, 3 (2004), 905–914. 1, 2, 3

[CSM03] COHEN-STEINER D., MORVAN J.-M.: Restricted Delaunay Triangulations and Normal Cycle. In *Proceedings of Symposium on Computational Geometry* (2003), ACM, pp. 312–321. 6

[DVBB13] DUNYACH M., VANDERHAEGHE D., BARTHE L., BOTSCH M.: Adaptive Remeshing for Real-Time Mesh Deformation. In *Eurographics Short Papers* (2013). 1, 5

[EP09] EIGENSATZ M., PAULY M.: Positional, Metric, and Curvature Control for Constraint-Based Surface Deformation. *Computer Graphics Forum 28*, 2 (Apr. 2009), 551–558. 2, 6

[ESP08] EIGENSATZ M., SUMNER R. W., PAULY M.: Curvature-Domain Shape Processing. *Computer Graphics Forum 27*, 2 (2008), 241–250. 2, 6

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of ACM SIGGRAPH* (1997), pp. 209–216. 1, 5

[GSMCO09] GAL R., SORKINE O., MITRA N. J., COHEN-OR D.: Iwires: An analyze-and-edit approach to shape manipulation. *ACM Trans. Graph. 28*, 3 (2009). 2

[HPW05] HILDEBANDT K., POLTHIER K., WARDETZKY M.: Smooth Feature Lines on Surface Meshes. In *Eurographics Symposium on Geometry Processing* (2005). 2

[KCS98] KOBBELT L., CAMPAGNA S., SEIDEL H.-P.: A general framework for mesh decimation. In *Proceedings of the Graphics Interface Conference* (1998), pp. 43–50. 5

[KO67] KLOTZ T., OSSERMANN R.: Complete Surfaces in E^3 with Constant Mean Curvature. *Commentarii mathematici Helvetici 41* (1967), 3130–318. 1, 3

[LDB05] LAVOUÉ G., DUPONT F., BASKURT A.: Subdivision surface fitting for efficient compression and coding of 3d models. In *Visual Communications and Image Processing 2005* (2005), vol. 5960, International Society for Optics and Photonics. 2, 7, 8

[LDB07] LAVOUÉ G., DUPONT F., BASKURT A.: A framework for quad/triangle subdivision surface fitting: Application to mechanical objects. *Computer Graphics Forum 26*, 1 (2007), 1–14. 2, 3

[Lie03] LIEPA P.: Filling holes in meshes. In *Proceedings of Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), pp. 200–205. 5

[LWC*11] LI Y., WU X., CHRYSATHOU Y., SHARF A., COHEN-OR D., MITRA N. J.: Globfit: Consistently fitting primitives by discovering global relations. *ACM Trans. Graph. 30*, 4 (2011). 2

[MGP06] MITRA N. J., GUIBAS L. J., PAULY M.: Partial and approximate symmetry detection for 3d geometry. *ACM Trans. Graph. 25*, 3 (2006), 560–568. 2

[MLDH15] MAGLO A., LAVOUÉ G., DUPONT F., HUDELOT C.: 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Computing Surveys 47*, 3 (2015). 2

[NISA07] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Fibermesh: Designing freeform surfaces with 3d curves. *ACM Trans. Graph. 26*, 3 (2007). 1, 2, 5, 6

[Pra87] PRATT V.: Direct least-squares fitting of algebraic surfaces. In *Proceedings of ACM SIGGRAPH* (1987), pp. 145–152. 3

[Ros99] ROSSIGNAC J.: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61. 2

[SF98] SINGH K., FIUME E.: Wires: A geometric deformation technique. In *Proceedings of ACM SIGGRAPH* (1998), pp. 405–414. 2

[SGC18] STEIN O., GRINSPUN E., CRANE K.: Developability of Triangle Meshes. *ACM Trans. Graph. 37*, 4 (July 2018), 77:1–77:14. 2, 5, 6

[SRK02] SZYMCZAK A., ROSSIGNAC J., KING D.: Piecewise regular meshes: Construction and compression. *Graphical Models 64*, 3 (2002), 183–198. 2

[SVWG12] SOLOMON J., VOUGA E., WARDETZKY M., GRINSPUN E.: Flexible Developable Surfaces. *Comput. Graph. Forum 31*, 5 (Aug. 2012), 1567–1576. 2

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Proceedings of Graphics Interface* (1998), pp. 26–34. 2

[WK05] WU J., KOBBELT L.: Structure Recovery via Hybrid Variational Surface Approximation. *Computer Graphics Forum 24*, 3 (2005), 277–284. 1, 2, 3, 4, 7

[WW92] WELCH W., WITKIN A.: Variational surface modeling. *SIGGRAPH Comput. Graph. 26*, 2 (1992), 157–166. 2

[ZWS11] ZHOU Q., WEINKAUF T., SORKINE O.: Feature-Based Mesh Editing. In *Eurographics Short Papers* (2011). 2