Character Articulation through Profile Curves

FERNANDO DE GOES, Pixar Animation Studios, USA
WILLIAM SHEFFLER, Pixar Animation Studios, USA
KURT FLEISCHER, Pixar Animation Studios, USA

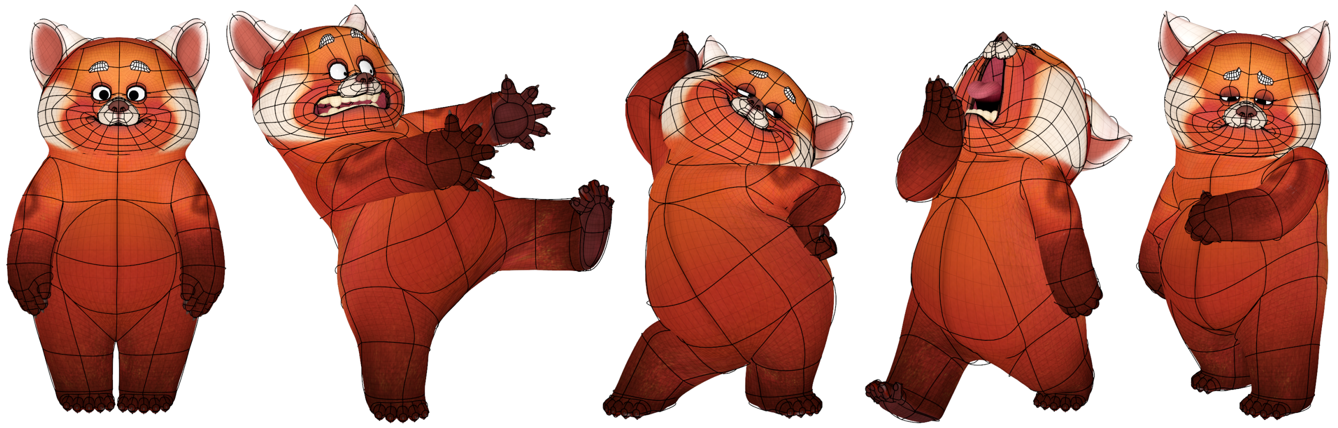


Fig. 1. **Profile Mover:** Examples of poses for a Panda character generated by our novel rigging technique that first articulates characteristic profiles of the deforming surface arranged into curvenets (displayed in black), and then computes the articulated character in a subsequent step by optimizing the mesh deformation that reconstructs surface details while interpolating the profile curves. ©Disney/Pixar

Computer animation relies heavily on rigging setups that articulate character surfaces through a broad range of poses. Although many deformation strategies have been proposed over the years, constructing character rigs is still a cumbersome process that involves repetitive authoring of point weights and corrective sculpts with limited and indirect shaping controls. This paper presents a new approach for character articulation that produces detail-preserving deformations fully controlled by 3D curves that profile the deforming surface. Our method starts with a spline-based rigging system in which artists can draw and articulate sparse curvenets that describe surface profiles. By analyzing the layout of the rigged curvenets, we quantify the deformation along each curve side independent of the mesh connectivity, thus separating the articulation controllers from the underlying surface representation. To propagate the curvenet articulation over the character surface, we formulate a deformation optimization that reconstructs surface details while conforming to the rigged curvenets. In this process, we introduce a cut-cell algorithm that binds the curvenet to the surface mesh by cutting mesh elements into smaller polygons possibly with cracks, and then derive a cut-aware numerical discretization that provides harmonic interpolations with curve discontinuities. We demonstrate the expressiveness and flexibility of our method using a series of animation clips.

CCS Concepts: • **Computing methodologies → Animation**.

Authors' addresses: Fernando de Goes, fernando@pixar.com, Pixar Animation Studios, USA; William Sheffler, sheffler@pixar.com, Pixar Animation Studios, USA; Kurt Fleischer, kurt@pixar.com, Pixar Animation Studios, USA.

0730-0301/2022/7-ART139
https://doi.org/10.1145/3528223.3530060

Additional Key Words and Phrases: character articulation, rigging, surface deformation, curvenet, mesh cutting.

ACM Reference Format:
Fernando de Goes, William Sheffler, and Kurt Fleischer. 2022. Character Articulation through Profile Curves. *ACM Trans. Graph.* 41, 4, Article 139 (July 2022), 14 pages. https://doi.org/10.1145/3528223.3530060

1 INTRODUCTION

Rigging plays a central role in character animation by defining the articulation setup that drives the deformation of digital characters. Over the last decades, character rigs have been built predominantly based on skinning schemes combined with corrective blend shapes. Despite the broad adoption in industry, these techniques are laborious to author with artists often hand-crafting weights and sculpts one pose at a time. Moreover, shaping character deformations involves repetitive trial and error in order to preserve surface details while neutralizing skinning artifacts. Another major challenge is the frequent need to repair the rig configuration after modeling updates, especially in areas of increased mesh resolution to avoid faceting artifacts. Consequently, designing high-quality character rigs remains a costly and specialized task in feature animations.

Our work is motivated by the observation that artists tend to iterate over the articulation setup by inspecting characteristic profiles of the character surface as the rig is exercised. Therefore, we propose to compute the character articulation by first rigging profile curves drawn by the user and propagating the curve deformation over the surface mesh afterwards (Figure 1). To this end, we introduce curvenets as a new rigging representation formed by the profile curves that separates the articulation controllers from the deforming surface. We construct curvenets using cubic Bézier splines traced

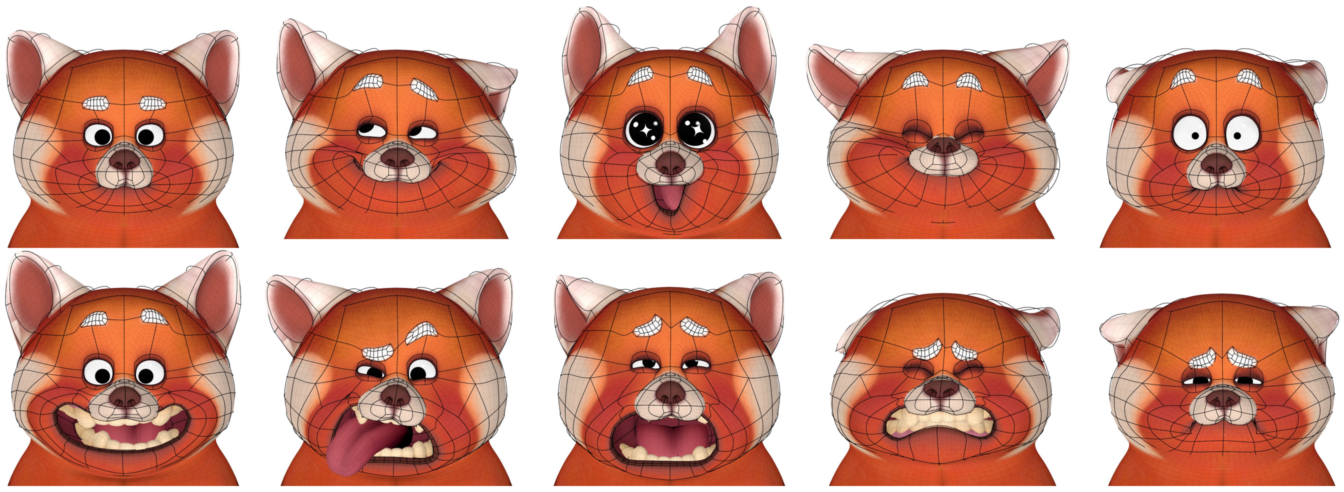


Fig. 2. **Face Rig:** By directly manipulating curvenets, our method generates a broad range of face deformations as shown in this sequence of expressions for the Panda character. Note how the curvenet layout is sparser than the underlying surface mesh, while providing prominent shaping controls. ©Disney/Pixar

near the character surface and arranged into connected components with no restrictions on their topological structure.

Equipped with curvenets, we can assemble the character rig by parts (e.g., hands, body, face) and independent of the mesh connectivity, thus facilitating concurrent work between modelers and riggers. We also exploit the curvenet layout to compute frames at the net intersections accompanied by a scaling amount per axis, which are then interpolated along the profile curves, bypassing the need for any curve optimization or manual authoring of normals and handles. Importantly, we estimate distinct scaled frames for each side of the profile curves so that the surface deformation is localized per curve side. As a result, curvenets significantly reduce the number of control points to be weighted or sculpted within the rig, while still producing a broad range of deformations.

To interpolate the curvenet deformation over the character surface, we draw inspiration from previous efforts for surface modeling and editing based on 3D curves [Gal et al. 2009; Nealen et al. 2007; Zhou et al. 2011]. Although these methods provide surface deformations through direct curve manipulation, they rely on workflows that are unsuited for character rigging. For instance, the work of Nealen et al. [2007] favors the generation of smooth shapes instead of preserving local details of a reference pose, while Gal et al. [2009] refits curve edits in order to restore spatial properties commonly found in man-made and engineered models but less relevant for articulated characters. Additionally, existing methods assume that the control curves are attached to mesh edges, thus limiting the deformation setup to a specific mesh resolution.

Instead, we present a novel deformation technique that produces detail-preserving character articulations driven by the rigged curvenet detached from the edges of the underlying surface mesh. In order to propagate the deformation from both sides of the profile curves, we develop a new mesh cutting algorithm that conforms the character surface to the curvenet by splitting the mesh polygons crossed by the profile curves into multiple sub-polygons possibly with cracks. We then extend the polygonal discretization introduced by de Goes et al. [2020] to construct cut-aware discrete differential operators over the resulting cut-mesh. Finally, we formulate a shape optimization adapted to the cut-mesh that computes the deformation of the character surface by interpolating the distortion and the pose of the rigged curvenet over the input mesh vertices.

By combining the curvenet representation (§3) with the surface deformation (§4), the technical contributions of our work are:

- A method for character articulation based on curvenets that reduces the number of control points and decouples the rig setup from the connectivity of the surface mesh.
- Construction of frames and non-uniform scaling at curvenet intersections followed by smooth interpolation along both sides of the profile curves.
- An algorithm for cutting polygonal meshes by curvenets that extends the Cartesian cut-cell method [Berger 2017; Tao et al. 2019] to curved surfaces.
- A mesh optimization that generates detail-preserving surface deformation while interpolating discontinuous constraints imposed by each side of the profile curves.

2 RELATED WORK

Before delving into our contributions, we briefly recap prior work that relates to our approach. Since character articulation has been tackled by a broad variety of techniques, we focus our exposition on recent methods for surface deformation, curve-based representation, and numerical discretization with cut discontinuities.

Spatial methods: Skinning is the prevalent approach for character rigging in the industry, blending articulation controllers such as joints, skeletons, and lattices directly to the deforming mesh vertices [McLaughlin et al. 2011]. Although various weighting strategies have been considered [Jacobson et al. 2014], these techniques notoriously introduce deformation artifacts, which are often mitigated through pose-space sculpting [Lewis et al. 2000] and post-process relaxation [Mancewicz et al. 2014]. Some methods have also attempted

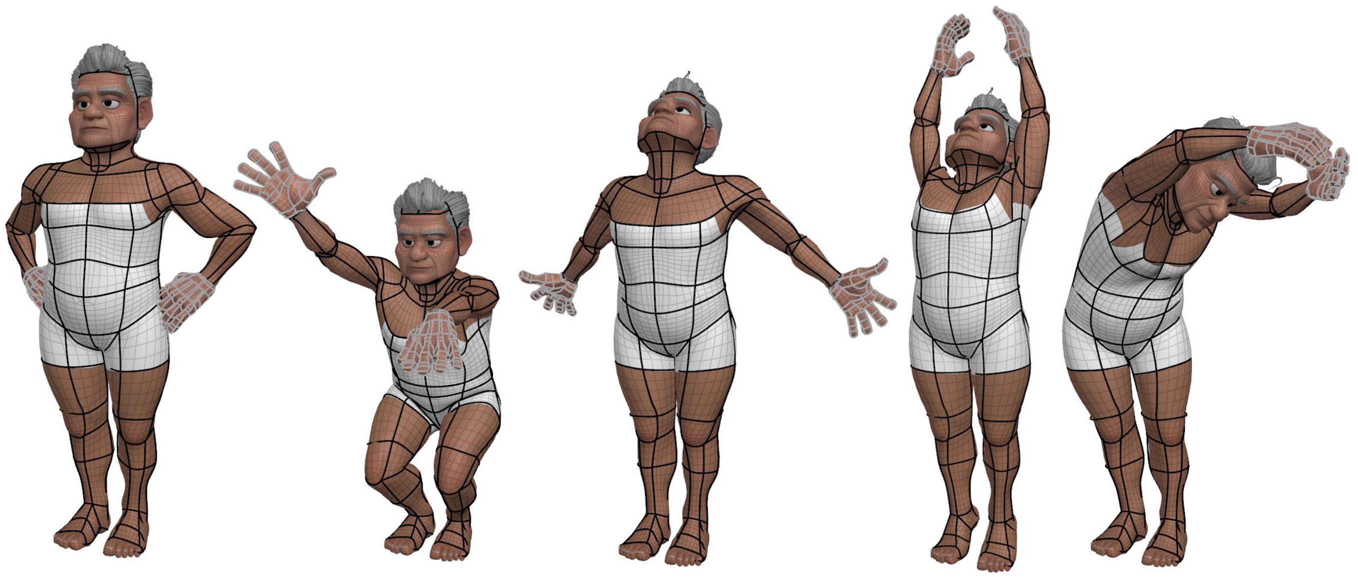


Fig. 3. **Biped Rig:** Poses for a character articulated by our method using a body curvenet (in black) and a pair of symmetric hand curvenets (in gray). Curvenets simplify the authoring of complex articulations such as the combo between shoulder, neck, and head, free of any corrective surface sculpts. ©Disney/Pixar

to reduce skinning artifacts by coupling space deformations with surface smoothing [Le and Lewis 2019; Le et al. 2021], elasticity optimizations [Jacobson et al. 2012; Kavan and Sorkine 2012], and contact-aware projections [Vaillant et al. 2013, 2014]. More recently, learning approaches were proposed for skeleton-based deformation by generating skinning weights and blend shapes together [Li et al. 2021; Xu et al. 2020]. In common, these spatial methods offer limited and indirect control over surface features. In sharp contrast, we construct the character rig by directly manipulating curves that profile the deforming surface.

Cage-based deformations: Character articulation can also be computed using generalized barycentric coordinates attached to cage meshes. Many approaches have focused on the generation of cage coordinates, either evaluated analytically [Lipman et al. 2008; Thiery et al. 2018] or optimized numerically [Joshi et al. 2007; Wang and Solomon 2021]. Other methods have investigated the assembly of cage meshes based on reusable template rigs [Ju et al. 2008] and bounding proxies [Calderon and Boubekeur 2017]. Despite their lower resolution, cages must form closed manifold meshes that loosely encapsulate the deforming character and define spatial deformations agnostic to surface details. We instead propose curvenets as a more flexible rigging representation with no structural constraints that produces detail-preserving deformations.

Detail preservation: Several surface-based methods have been developed in order to produce detail-preserving deformations [Botsch and Sorkine 2008]. For instance, Yu et al. [2004] proposed a Poisson editing scheme that propagates handle transformations using geodesic distances, while Zayer et al. [2005] and Lipman et al. [2007] computed local rotations through harmonic interpolations. In [Lipman et al. 2005], geometric features were restored by reconstructing local frames through discrete fundamental forms, whereas the method of Sorkine et al. [2004] employed rotated Laplacian coordinates. Non-linear techniques have also been devised at the cost of iterative optimizations [Botsch et al. 2006; Sorkine and Alexa 2007]. Our approach builds upon previous methods by presenting a modified Poisson solver that generates detail-preserving deformations conforming to a net of rigged curves.

Curve-based editing: Similar to our work, some research efforts have also advocated for sparse curves as a more expressive interface for surface deformation. In [Singh and Fiume 1998], space deformations were computed driven by prominent surface curves but agnostic to geometric features. The work of Nealen et al. [2005] combined detail-preserving deformation with silhouette curves for mesh editing. A sketching system interleaved with surface remeshing was also presented by Nealen et al. [2007] for free-form modeling. In [Gal et al. 2009], an analyze-and-edit scheme was customized for man-made shapes by annotating and refitting representative curves based on spatial properties. This method was later extended by Zhou et al. [2011] using curvature feature curves such as surface ridges and valleys. Unfortunately, many of these methods restrict control curves to mesh edges, thus entangling the deformation setup with the mesh connectivity. Instead, our curvenet construction is detached from the underlying mesh and produces reliable deformations for any tessellation of the character surface.

Curve networks: In addition to deformations, curve networks are also relevant in other geometry processing applications. For instance, Campen [2017] surveys approaches for generating curve networks used by quad meshing algorithms. The work of Gori et al. [2017] extracted curve networks for abstracting man-made shapes, while de Goes et al. [2011] proposed a user-assisted segmentation that computes a surface exoskeleton. In [Pan et al. 2015], a technique for surfacing curve networks was presented by minimizing curvature variation. The lofting approach described by Schaefer et al. [2004] also interpolates curve networks but using a modified scheme for Catmull-Clark subdivision surfaces. Our work is complementary to these prior methods since we target character rigging and assume that the surface mesh and curves are given as user inputs. In particular, our curvenet representation is intended to convey deformation profiles, even if the individual curves do not depict perceptual cues. As an example, Figure 1 includes a curve surrounding the belly of a Panda character, which may not be considered a shape descriptor but it provides meaningful shaping controls.

Frames from curves: Another important aspect in articulating surfaces based on curves is how to estimate the orientation and the stretch of the deforming curves. A naive approach is to incorporate an extra rigging component for posing transformation handles along curves at the expense of more manual authoring. Rotation minimizing frames [Wang et al. 2008] can also offer consistent curve orientations, but it relies on prior knowledge of the normals at the curve endpoints and lacks stretching estimates. Alternatively, Nealen et al. [2007] proposed a curve optimization that relaxes user edits by favoring as-rigid-as-possible deformations. The work of Huang et al. [2019] infers smooth normals and curvature values from unstructured curves through a costly implicit function fitting. We instead leverage the curvenet layout to compute scaled frames on both curve sides. Notably, we account for frame differences on each curve side in order to bound the surface deformation.

Diffusion curves: Our approach for smooth interpolation with curve discontinuities is closely related to the concept of diffusion curves, which was originally presented in [Orzan et al. 2008] to generate resolution-independent image gradients. Various extensions to diffusion curves have been proposed including flexible boundary conditions [Boyé et al. 2012], random-access image evaluation via the fast multipole method [Sun et al. 2014], and diffusion surfaces for volume modeling [Takayama et al. 2010], to cite a few. Surprisingly, computing diffusion curves on non-flat surface meshes remains far less explored. One exception is the work of Lucquin et al. [2017] that produces parametrization cuts by approximating user strokes as harmonic field discontinuities. In this method, input strokes are duplicated, moved apart by an adjustable parameter, and then used as soft constraints in the harmonic interpolation. However, these displaced curves can introduce overlaps between nearby curves and at curve intersections, damaging the interpolation smoothness. In contrast, we present a new mesh cutting algorithm that enables the exact enforcement of curve discontinuities. Our approach can be interpreted as an extension of the cut-cell method [Berger 2017; Tao et al. 2019], commonly used to clip Cartesian grids against boundary geometry, but now adapted to polygonal surface meshes.

Cut-aware discretizations: Finally, we point out that numerical methods for meshes with cuts have been broadly investigated in computational mechanics [Fries and Belytschko 2010] as well as in graphics [Wu et al. 2015]. While many approaches have addressed tearing and cracking in physically based simulations, we focus on the numerical discretization of Poisson problems with interface discontinuities. The work of Burman et al. [2015] proposed to duplicate mesh elements intersected by curves and impose discontinuous boundary conditions via ghost penalty terms. In extended finite element methods (see, e.g., [Moës et al. 1999]), cut discontinuities are captured by incorporating enrichment basis functions, which are approximated using adaptive numerical integration. To support multiple branched curves, Mousavi et al. [2011] precomputed harmonic enrichment functions encoded by auxiliary texture images within each mesh element. More recently, Benvenuti et al. [2019] introduced the extended virtual element method that adapts enrichment functions over 2D polygons with improved numerical stability. Departing from previous methods, our approach bypasses enrichment functions and cubature schemes while handling complex surface cutting. To this end, we modify the assembly of discrete differential operators presented by de Goes et al. [2020] in order to account for arbitrary 3D polygons with cuts.

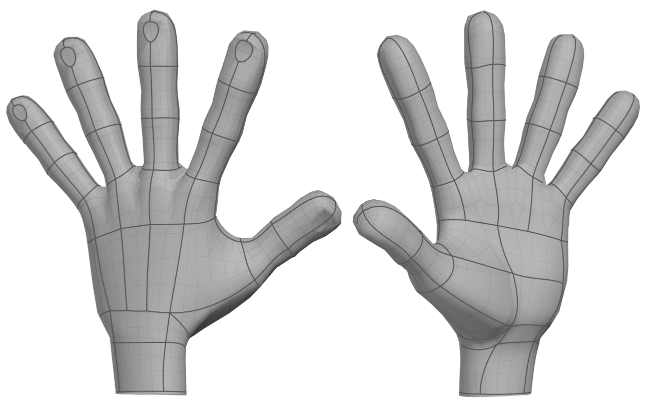


Fig. 4. **Curvenet:** Example of a curvenet modeled over a quad-dominant surface mesh of a hand model. Observe how the curves resemble deformation profiles such as the hand ligaments and knuckles, while forming intersections and T-junctions detached from the mesh connectivity. ©Disney/Pixar

3 CURVENETS

In this section, we introduce curvenets as a new rigging representation for character articulation. We begin by presenting the key definitions and detailing our approach for modeling, rigging, and sampling curvenets. In addition, we describe a routine for estimating local frames and scales aligned to curvenets, which we then use to quantify the deformation between different curvenet poses.

Definition: We define a curvenet as a collection of 3D curves that can intersect each other. These curves are intended to represent characteristic profiles of a surface mesh we wish to articulate. In order to support flexible designs, we devise curvenets free of any topological restriction, including curves with open endpoints, intersections of arbitrary valence, and multiple connected components. Moreover, the individual curves are not required to be aligned to mesh edges or embedded onto the underlying surface. As a result, curvenets are agnostic to the mesh connectivity and the surface topology. This is in sharp contrast to previous curve network representations that rely on a selection of mesh edges [Gori et al. 2017] and partition surfaces into patches [Campen 2017]. Figure 4 shows a curvenet built over a quad-dominant mesh of a hand model. Observe how the curves cross mesh elements in order to delineate deformation profiles, while some regions such as the webbing space between fingers are surrounded but not cut by any curve.

Modeling: To construct curvenets, we implemented a modeling interface based on cubic Bézier splines inside a rigging package. The system starts with the surface mesh we want to deform in a neutral pose. The user can then insert control points at arbitrary locations on the surface and click-and-drag curves resembling surface profiles. Note that the individual profile curves lie near the surface mesh but are not explicitly attached to the mesh connectivity. When creating a cubic Bézier spline, we allocate two interior control points forming tangent handles and initialize them perpendicular to the surface normals. Importantly, we allow endpoints to be shared by multiple splines. To this end, we encode each cubic Bézier spline as a tuple of four indices mapping to a pool of control points with their respective 3D positions. Our toolkit also includes operations such as split and merge splines, weld and break control points, project endpoints to the surface mesh, and flatten tangents, to cite a few.

Rigging: Interleaved with modeling, our implementation also enables the user to articulate curvenets by simply posing their control points based on preexisting rigging techniques such as skinning and sculpting. Since curvenets have fewer points than a typical surface mesh, the resulting rig is more compact and simpler to setup, while keeping the same articulation controllers and authoring workflows. Curvenets can be further exploited as direct surface manipulators, similar to curve handles used by modeling and editing methods [Gal et al. 2009; Nealen et al. 2007], thus enhancing rigs with surface-based shaping controllers. We point the reader to the supplemental video for an interactive session of our system showcasing the construction and articulation of curvenets.

Intersections, anchors & curves: Once the curvenet is modeled, we identify every endpoint shared by three or more splines and denote them as intersections. We also gather endpoints incident to a single spline and mark them as anchors. Using these labeled endpoints, we rearrange the curvenet by grouping splines into curves, each one formed by a sequence of connected splines bridging between intersections and/or anchors. We also group any remaining spline connected solely by unlabelled endpoints, thus forming isolated closed curves. Finally, we precompute a list mapping each intersection to its emanating curves sorted counter-clockwise based on their corresponding tangent vectors projected orthogonal to the normal of the closest surface point in a neutral pose.

Sampling: Given control points positioned by the rig, we approximate the curvenet shape by converting its cubic Bézier splines into polylines with evenly spaced sample points. Each spline is first refined uniformly in parametric space and then resampled evenly based on arc-length. The number of samples inside each spline is set to a user-specified value (default to 5) multiplied by the ratio between the length of the polyline connecting the spline's control points and the mean edge length of the underlying surface mesh, both calculated in a neutral pose. With this sampling strategy, we

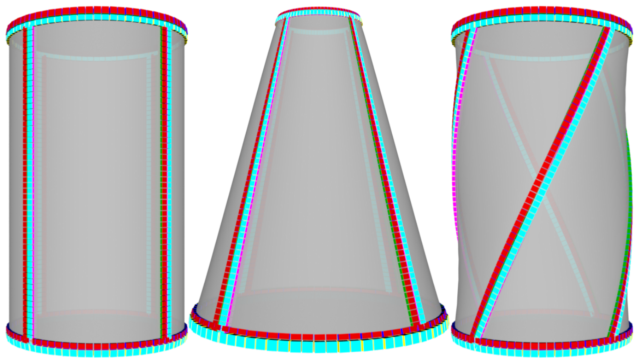


Fig. 5. **Scaled Frames:** In these images, we use deformed boxes to display the orthogonal frames with axis-aligned scales deduced from our curvenet representation. Red-green-blue boxes indicate the scaled frames on one curve side and the complementary cyan-yellow-magenta colors refer to the reverse curve side. Note how the scaled frames are oriented and sized based on the normals and widths estimated at every corner of intersecting curves.

obtain a piecewise-linear representation of the curvenet with resolution proportional to the size of surface mesh elements and adaptive to the length of the undeformed splines. Hereafter, we refer to a segment as a pair of consecutive sample points within a curve of the discretized curvenet.

Normal & width at intersections: Along a piecewise-linear curve, we can use the sample positions to calculate the unit tangent vector $\mathbf{t}_s$ and the length l_s for each segment s. However, the individual curves are not sufficient to estimate a normal orientation or a width for any curve segment. To address this issue, we leverage the intersection points and their precomputed list of incident curves available in the curvenet. We denote the ordered list of tangent vectors and lengths for the segments $\{s_i\}$ emanating from an intersection point by $\{(\mathbf{t}_i, l_i)\}$. For every corner of consecutive tangents at an intersection, we define the corner vector $\mathbf{c}_i = \mathbf{t}_i \times \mathbf{t}_{i+1}$ and set the corner normal to $\mathbf{m}_i = \mathbf{c}_i / \|\mathbf{c}_i\|$. In the special case of a corner with parallel tangents (e.g., at T-junctions), we resolve the degenerate normal by including adjacent corners via $\mathbf{m}_i = (\mathbf{c}_{i+1} + \mathbf{c}_{i-1}) / \|\mathbf{c}_{i+1} + \mathbf{c}_{i-1}\|$. These corner normals provide two normal candidates to every segment outgoing an intersection. Therefore, we assign the normal vector $\mathbf{n}_i^+ = \mathbf{m}_i$ to the left side and $\mathbf{n}_i^- = \mathbf{m}_{i-1}$ to the right side of each segment s_i at an intersection. We can also use the alignment between tangent vectors in each intersection corner to estimate a width per segment side, thus implicitly describing curvenets as a net of ribbons with a scale along and across its segments. We assign the width for the left side of the segment s_i to $w_i^+ = l_i + \|\mathbf{c}_i\|(l_{i+1} - l_i)$, while its right side is set to $w_i^- = l_i + \|\mathbf{c}_{i-1}\|(l_{i-1} - l_i)$. Notice that the width in each segment side is a positive scale with an anisotropy relative to the segment length proportional to the length difference and the orthogonality of its intersection corner.

Normal & width interpolation: We can further interpolate the normal and width estimates from the intersection segments to each side of their corresponding curves. For clarity, we omit below the left (+) and right (−) superscripts from our notation, but the interpolation scheme applies to both curve sides. First, we consider a curve made of k segments with one end at an intersection and the other end at an anchor. Starting at the intersection, we parallel transport the normal from a segment s_i to the next segment s_{i+1} using the smallest rotation matrix $\boldsymbol{R}_i$ mapping $\mathbf{t}_i$ to $\mathbf{t}_{i+1}$. As a result, we can express the normal $\mathbf{n}_i$ at the segment s_i in terms of the normal $\mathbf{n}_1$ from the intersection segment s_1, i.e., $\mathbf{n}_i = \boldsymbol{\Omega}_i \mathbf{n}_1$, where $\boldsymbol{\Omega}_i = \prod_{j=1}^{i-1} \boldsymbol{R}_j$ indicates the accumulated rotation along the curve up to the segment s_i. Since this curve ends at an anchor, we simply copy the width from the intersection segment to every other segment, thus producing a uniform width along each curve side. When a curve has intersections at both ends, the normal $\boldsymbol{\Omega}_k \mathbf{n}_1$ of the last segment s_k computed through parallel transport may not match the normal $\mathbf{n}_k$ estimated at the intersection corner. This discrepancy amounts to the torsion induced by the curvenet shape over the curve, and the total torsion angle is given by $\theta = \operatorname{atan}((\boldsymbol{\Omega}_k \mathbf{n}_1)^{\mathrm{t}}(\mathbf{n}_k \times \mathbf{t}_k)/(\boldsymbol{\Omega}_k \mathbf{n}_1)^{\mathrm{t}} \mathbf{n}_k)$. To blend the torsion along the curve, we calculate the normalized curve arc-length $\alpha_i = \left(\sum_{j=1}^{i-1} l_j\right)/\left(\sum_{j=1}^{k} l_j\right)$ for each segment s_i, and assemble the torsion matrix $\boldsymbol{\Theta}_i$ as the rotation by the angle $\alpha_i \theta$ around the tangent vector $\mathbf{t}_i$. The interpolated normal at segment s_i is then assigned to $\mathbf{n}_i = \boldsymbol{\Theta}_i \boldsymbol{\Omega}_i \mathbf{n}_1$. Similarly, we employ the arc-length values to interpolate the width for each segment s_i, yielding $w_i = (1-\alpha_i) w_1 + \alpha_i w_k$. To complete the frame with axis-aligned scales per segment s_i, we associate the width w_i with the binormal vector $\mathbf{b}_i = \mathbf{n}_i \times \mathbf{t}_i$ and set the geometric mean $h_i = \sqrt{l_i w_i}$ to be the scale along the normal vector $\mathbf{n}_i$. We can also arrange these axis vectors into the columns of an orthonormal matrix $\mathbf{B}_i = [\mathbf{t}_i\ \mathbf{b}_i\ \mathbf{n}_i]$ and the respective axis-aligned scales into a diagonal matrix $\mathbf{S}_i = \operatorname{diag}(l_i, w_i, h_i)$, thus defining the scaled frame per segment s_i as the matrix $\mathbf{B}_i \mathbf{S}_i$. Figure 5 illustrates scaled frames generated by different curvenet poses.

Deformation Gradient: Since we approximate each curve with a fixed number of evenly spaced samples per spline, these samples define an one-to-one mapping between the segments of different curvenet poses. With this correspondence, we can measure the distortion of a deformed curvenet relative to an undeformed configuration by evaluating the deformation gradient at each segment. The deformation gradient is a matrix $\mathbf{F}_i$ that quantifies the amount of rotation and stretching deforming a rest segment to its respective posed segment. For curves incident to intersections, we compute the deformation gradient for each side of a segment s_i as the matrix $\mathbf{F}_i = (\mathbf{B}_i \mathbf{S}_i)(\check{\mathbf{B}}_i \check{\mathbf{S}}_i)^{-1}$ that transforms the rest scaled frame $\check{\mathbf{B}}_i \check{\mathbf{S}}_i$ to the posed scaled frame $\mathbf{B}_i \mathbf{S}_i$. We can further expand the deformation gradient matrix using axis vectors and scales, yielding

$$\mathbf{F}_i = (l_i/\check{l}_i)(\mathbf{t}_i \otimes \check{\mathbf{t}}_i) + (w_i/\check{w}_i)(\mathbf{b}_i \otimes \check{\mathbf{b}}_i) + (h_i/\check{h}_i)(\mathbf{n}_i \otimes \check{\mathbf{n}}_i), \quad (1)$$

where the symbol $\otimes$ denotes the outer product between two vectors. Once again, we omit the superscript $\pm$ encoding each segment side for conciseness. Note that the deformation gradient in Eq. (1) includes a tangential rotation, a normal twist, and non-uniform scaling all deduced directly from the curvenet representation. Moreover, the matrices for both segment sides share the same tangential transformation but they can have different twisting and non-tangential stretching amounts. In the case of isolated curves, either closed or with both ends at anchors, there is no curvenet structure that complements the tangent and the length of the curve segments. We thus compute the deformation gradient for a segment s_i in these simplistic curves using the smallest rotation from the rest tangent $\check{\mathbf{t}}_i$ to the posed tangent $\mathbf{t}_i$ multiplied by the uniform scaling given by the length ratio $l_i/\check{l}_i$.

Algorithm 1 Surface deformation via rigged curvenet.

```
   // precomputation
1: Compute cut-mesh (§4.1).
2: Assemble matrices $\{\mathbf{L}, \mathbf{C}, \mathbf{V}\}$ (§4.2).
3: Factorize Laplacian matrix $\mathbf{V}^{\mathrm{t}}\,\mathbf{L}\,\mathbf{V}$ (§4.3).
   // at runtime
4: Compute curvenet deformation (§3).
   a: Evaluate target position $\mathbf{q}_i$ for every sample $i$.
   b: Estimate deformation gradient matrices $(\mathbf{F}_s^+, \mathbf{F}_s^-)$ for both sides
      of every segment $s$.
5: Rearrange deformation gradients into constraint matrix $\mathbf{f}_c$.
6: Solve for deformation gradient $\mathbf{f}_v$ at mesh vertices via Eq. (4).
7: Compute matrix $\mathbf{y}_h$ with deformed cut-face polygons.
8: Compute matrix $\mathbf{x}_c$ with displaced curvenet samples.
9: Solve for deformed vertex positions $\mathbf{x}_v$ via Eq. (5).
```

4 SURFACE DEFORMATION

We now present our approach to compute the deformation of the character surface by propagating the curvenet articulation through the input mesh. We assume the character surface is given by an oriented manifold mesh, possibly with boundaries, formed predominantly by triangles and quadrilaterals, as commonly found in character animation. Our method involves a precomputation step performed in a neutral pose that combines the surface mesh with the sampled curvenet, followed by a runtime solve that produces the deformed surface based on the rigged curvenet. Algorithm 1 provides a pseudocode with an overview of our deformation routine.

4.1 Mesh Cutting

We begin by precomputing the binding of the piecewise-linear curvenet onto the surface mesh representing the character model in a neutral pose. In order to capture the different values estimated from each side of the curvenet segments, we propose an adaptation of the Cartesian cut-cell method [Berger 2017; Tao et al. 2019] to curved surfaces that cuts the input polygonal mesh by tracing curvenet segments. The resulting cut-mesh retains the input mesh vertices as well as the curvenet samples, while splitting the mesh faces into smaller polygons, which can be non-planar, non-convex, and even include cracks (see an example in the bottom row of Figure 13).

We implement the cut-mesh using a customized halfedge data structure that annotates halfedges with their corresponding oriented curvenet segments, thus defining a mapping from the cut-mesh to the sampled curvenet. We also include a bitmask to encode if each cut-vertex refers to a mesh vertex, a curvenet sample, or an intersection between a curvenet segment and a mesh edge. To indicate cracks cutting a polygon, we permit cut-edges to branch out of the cut-face boundary with their respective pairs of opposite halfedges both pointing to the same cut-face.

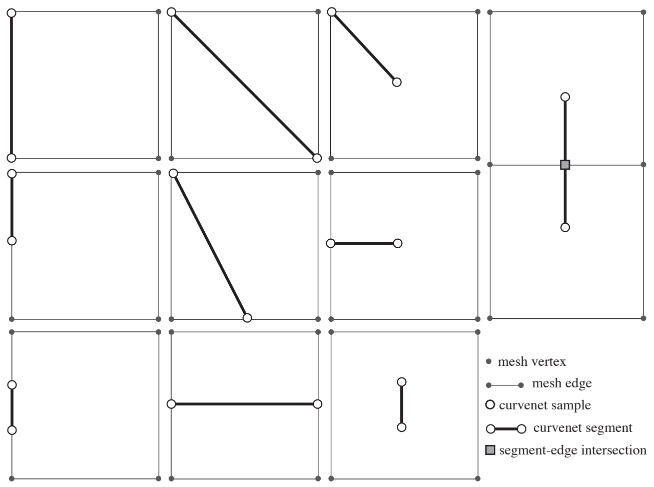
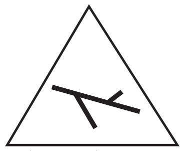


Fig. 6. **Mesh Cutting:** To cut a mesh polygon by a curvenet segment, we enumerate different arrangements between the curvenet samples and the mesh elements. The first column shows a segment with both samples incident to the same mesh edge, which subdivides the cut-edge for every sample located inside the edge. The second column corresponds to a curvenet segment that divides the mesh polygon into smaller cut-faces. The third column illustrates cracks in the mesh polygon formed by the curvenet segment, which can be completely inside the cut-face or slicing its boundary. The fourth column exemplifies a segment traced through multiple mesh elements that inserts new cut-vertices at the segment-edge intersections.

After initializing the cut-mesh with a copy of the input polygonal mesh, we project the neutral position $\check{\mathbf{q}}$ for each curvenet sample to the closest point $\check{\mathbf{p}}$ on the surface mesh. Based on the mesh element hit by the projection, we categorize each sample as a vertex, edge, or face-sample. In our implementation, we detect if a projected sample is at a vertex or within an edge using a numerical tolerance of 0.001% of the diagonal length of the surface bounding box. We employ the vertex-samples to tag the bitmask of their coincident cut-vertices. We also subdivide every cut-edge containing edge-samples by inserting new cut-vertices at their respective projected locations. The face-samples are allocated as isolated cut-vertices and later connected by curvenet segments.

To incorporate the curvenet segments into the cut-mesh, we reuse the sample-to-mesh categories and enumerate all possible sample combinations per segment (Figure 6). The simplest scenario is when a segment corresponds to an existing cut-edge that connects vertex and/or edge-samples (left column). The other scenarios require creating new cut-edges that pass across the input polygons. To this end, we first insert new cut-edges with halfedge pairs pointing to the same cut-face, and in a subsequent step we update the cut-mesh by splitting the affected cut-faces. If the segment has samples at the boundary or contained by the same mesh face, we place the new cut-edge inside the corresponding cut-face (middle columns). When the segment samples do not share a mesh face, we adopt the method of Polthier and Schmies [1998] and trace the straightest path between the projected sample points over the surface mesh. As a result, the segment is rasterized into a chain of new cut-edges with cut-vertices added at the intersections between the segment path and the input mesh edges (right column).

Next, we update the cut-mesh connectivity in order to identify the split cut-faces produced by the curvenet segments. We approach this task by computing a tangent space for each cut-vertex, projecting its incident halfedges onto the tangent space, and then sorting the projected halfedges counter-clockwise. To compute the tangent spaces, we take advantage of the fact that the cut-mesh is constructed on top of the input polygonal mesh. Therefore, we can assign the tangent space for any cut-vertex by referring to the corresponding point on the surface mesh. More concretely, when a cut-vertex is inside a mesh face, the tangent space is set to the plane orthogonal to the normal of the underlying polygon. In case the cut-vertex lies inside a mesh edge, we define the tangent space by unfolding the pair of faces sharing the mesh edge to a common plane. Otherwise, if the cut-vertex is at a mesh vertex, we flatten the one-ring of faces incident to the mesh vertex. With sorted halfedges at the cut-vertices, we can circulate from one halfedge to the next and reset the cut-mesh with new cut-faces for every loop of halfedges.



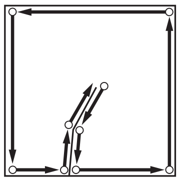
Lastly, we address the special case that arises when the curvenet has clusters of segments held entirely inside individual faces of the input mesh (see inset). These clusters indicate holes that cut the interior of the mesh polygons. Since we have inserted every segment into the cut-mesh, we can detect these curvenet islands by computing connected components of the cut-mesh formed exclusively by face-samples attached to the same mesh face. In our implementation, we remove every element of the cut-mesh incident to these isolated components, because they represent a level of detail finer than the resolution of the surface mesh.

4.2 Discretization

The cut-mesh provides a discrete representation of the input surface that conforms to the curvenet. We describe next how to leverage this conforming mesh as a computational domain in which the curvenet deformation is interpolated over the vertices of the surface mesh. In particular, we present a numerical discretization tailored to the cut-mesh that handles polygons with cracks, thus enabling smooth interpolations with localized discontinuities.

We start by discretizing the space of functions defined over the cut-mesh. In order to convey discontinuities, we represent discrete functions by assigning a scalar value for each corner of a cut-face, akin to the typical encoding of uv-coordinates on surface meshes. Since face-corners can be indexed by halfedges, we assemble the discrete version of a scalar function ϕ by a vector $\boldsymbol{\phi}_h$ of size equal to the number of halfedges n_h in the cut-mesh. These values are interpolated linearly along the halfedges forming each cut-face and the discontinuities are indicated by any cut-vertex that reads different values from its incident halfedges.

To evaluate the smoothness of a discrete function, we base our approach on the polygonal discretization introduced by de Goes et al. [2020], which assumes that the mesh polygons are simple. At first sight, the cut-mesh seems to fail this requirement because the cut-edges can depict cracks inside the mesh polygons. To accommodate polygons with cracks, we construct the polygon of a cut-face using the ordered list of face-corners defined by the loop of halfedges outlining the cut-face (see inset). By doing so, crack edges are split into pairs of halfedges, cut-vertices are duplicated and, consequently, the cut-face is represented by a simple polygon of arbitrary shape.



Given a cut-face f formed by a loop of n_f halfedges, we assemble the matrix $\check{\mathbf{X}}_f$ of size $n_f \times 3$ containing the 3D points that define the simple polygon associated with f ordered counter-clockwise, stacked row-wise, and in a neutral pose. As detailed in the Appendix A, we employ the matrix $\check{\mathbf{X}}_f$ to compute the polygonal Laplacian matrix $\mathbf{L}_f$ of size $n_f \times n_f$. We then define the cut-mesh Laplacian as the matrix $\mathbf{L}_h$ of size $n_h \times n_h$ that gathers the contribution of every cut-face Laplacian $\mathbf{L}_f$. Finally, we measure the smoothness of a discrete function $\boldsymbol{\phi}_h$ over the cut-mesh via the Dirichlet energy:

$$E_D(\boldsymbol{\phi}_h) = \boldsymbol{\phi}_h^{\mathrm{t}}\, \mathbf{L}_h\, \boldsymbol{\phi}_h. \tag{2}$$

Since our goal is to compute an interpolation from the curvenet to the input mesh, we restrict the solution space of discrete functions so that discontinuities are imposed as constraints by the curvenet, while mesh vertices are assigned to smooth values shared by their incident halfedges. To this end, we organize the cut-vertices into two groups based on their bitmasks. The first group defines the interpolation unknowns represented by every cut-vertex coincident to a mesh vertex. We arrange these unknowns as a vector $\boldsymbol{\phi}_v$ of size n_v and introduce the matrix $\mathbf{V}$ of size $n_h \times n_v$ that copies values from these cut-vertices to their adjacent halfedges. Conversely, the second group includes every cut-vertex corresponding to a curvenet sample or created by the intersection of a curvenet segment and a mesh edge. Note that some of the halfedges emanating from these cut-vertices are annotated by curvenet segments and, therefore, indicate constraints to be set onto the cut-mesh. We denote by $\boldsymbol{\phi}_c$ the vector with the values of the curvenet samples estimated from each side of every curvenet segment. In total, the size n_c of the vector $\boldsymbol{\phi}_c$ is twice the sum of the number of samples per curve. We also assemble the matrix $\mathbf{C}$ of size $n_h \times n_c$ that maps these sample values to their adjacent halfedges. In the case of a cut-vertex incident to both a mesh vertex and the curvenet, we give precedence to the curvenet constraint, thus ensuring that $\mathbf{V}^{\mathrm{t}}\mathbf{C} = 0$. By construction, the matrices $\mathbf{V}$ and $\mathbf{C}$ define a partition of unity, i.e., $\mathbf{V}\mathbf{1}_v + \mathbf{C}\mathbf{1}_c = \mathbf{1}_h$, where $\mathbf{1}_x$ indicates a constant vector of ones of size n_x $(x \in \{v, c, h\})$. By combining these terms, we write our restricted solution space as discrete functions of the form

$$\boldsymbol{\phi}_h = \mathbf{V}\,\boldsymbol{\phi}_v + \mathbf{C}\,\boldsymbol{\phi}_c. \tag{3}$$

With the cut-mesh discretization, we can now present our method that computes the final shape of the surface mesh based on the deformation of the sampled curvenet, as detailed in the next subsection.

4.3 Mesh Optimization

We approach the deformation of the surface mesh induced by the curvenet articulation as a two-step optimization. Given the undeformed and deformed configurations of the sampled curvenet indicated by their respective sample positions $\check{\mathbf{q}}$ and $\mathbf{q}$, we first interpolate the deformation gradients from the curvenet segments to the mesh vertices, and then we compute the vertex positions that best match the interpolated deformation gradients while preserving surface details and reproducing the target curvenet samples.

In the first optimization step, we make use of the deformation gradient matrices $(\mathbf{F}_s^+, \mathbf{F}_s^-)$ estimated from each side of every curvenet segment s, as previously described in §3. We remap these matrices from the segments to their incident samples. For a sample inside a curve, we set its left and right matrices by averaging the values from the previous and next segments along the curve. When the sample is a curve endpoint, we make a copy of the sample for every incident segment and assign each copy to the values from the left and the right side of the corresponding segment. With these sample-based deformation gradients, we assemble the curvenet constraints as a matrix $\mathbf{f}_c$ of size $n_c \times 9$ that flattens the deformation gradient matrices into row-vectors. We then compute the matrix $\mathbf{f}_v$ with the flattened deformation gradients at the mesh vertices by solving

$$\min_{\mathbf{f}_v} E_D(\mathbf{V}\,\mathbf{f}_v + \mathbf{C}\,\mathbf{f}_c). \tag{4}$$

The optimization in Eq. (4) produces a harmonic interpolation over the cut-mesh for each column in $\mathbf{f}_v$ with discontinuities at the curvenet samples prescribed by $\mathbf{f}_c$. After combining the interpolated and constrained matrices $\mathbf{f}_v$ and $\mathbf{f}_c$ via Eq. (3), we obtain the deformation gradient for the corner of every cut-face in the cut-mesh. We then average the corner values within each cut-face f and unfold the resulting row-vector back into a 3×3 matrix denoted by $\mathbf{F}_f$. The deformation gradient $\mathbf{F}_f$ per cut-face f indicates how the rest configuration $\check{\mathbf{X}}_f$ of the simple polygon associated with f should be rotated and stretched. Since every halfedge is incident to a single cut-face, we can gather the deformed polygons $\check{\mathbf{X}}_f \mathbf{F}_f^{\mathrm{t}}$ from every cut-face f into a matrix $\mathbf{y}_h$ of size $n_h \times 3$.

Our second optimization step seeks new vertex positions that approximate the transformed cut-face polygons given by $\mathbf{y}_h$, while enforcing the sample locations constrained by the posed curvenet. In particular, we must account for the residual vector defined between each rest sample point $\check{\mathbf{q}}$ and its projected point $\check{\mathbf{p}}$ so that the

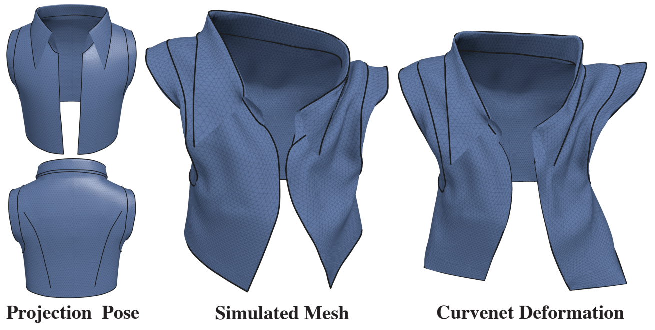


Fig. 7. **Surface Editing:** Our method can be used as an interactive shaping tool well-suited for post-simulation editing. We start by drawing the curvenet over the original model, defining the projection pose used in the precomputation of our solver (left). We then warp the curvenet to the simulated mesh, which indicates the undeformed surface configuration (middle). By directly manipulating the curvenet, we can edit the surface mesh while preserving the cloth details produced by simulation (right). ©Disney/Pixar

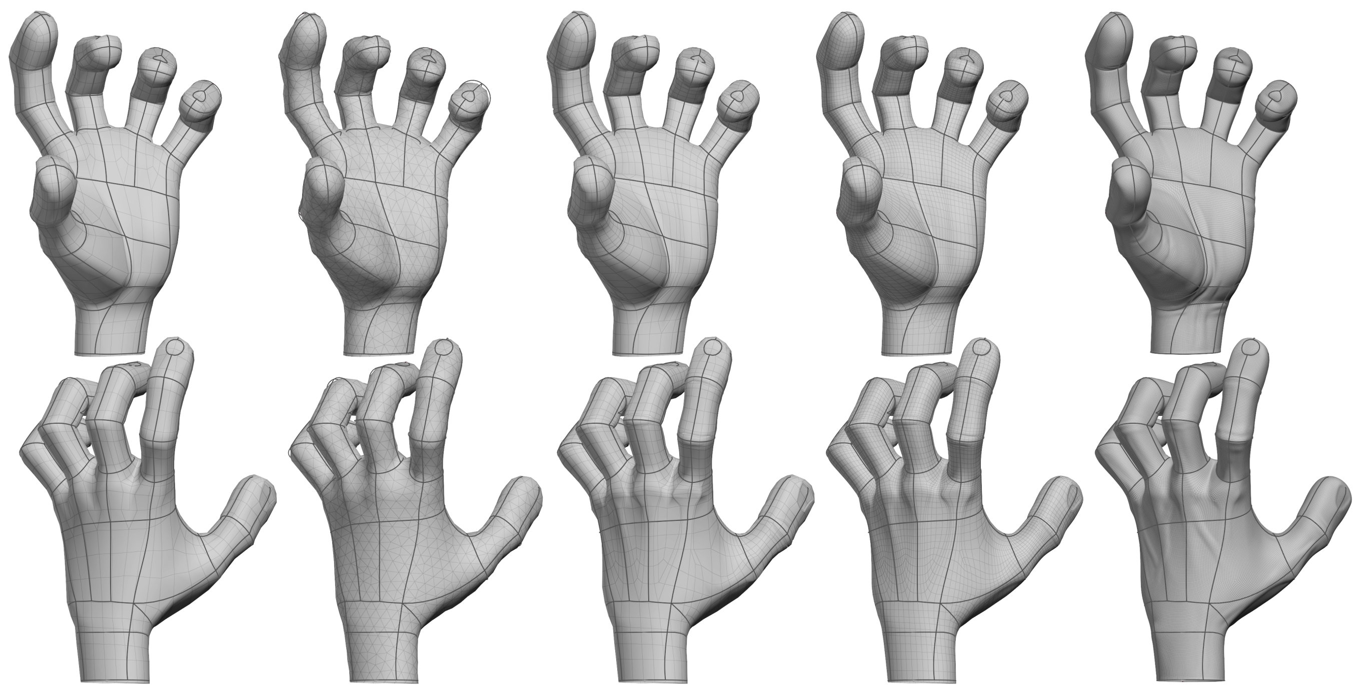


Fig. 8. **Robustness to mesh connectivity:** Using the same curvenet rig, our technique allows the articulation of multiple tessellations of a hand model. From left to right, we show the front and back deformations for a coarse quad-dominant mesh, a triangulated mesh, a subdivision cage, an one-level subdivided surface, and a high-resolution sculpted mesh. Observe that our curvenet deformation preserves different levels of surface details as well as the residual sample offsets caused by the loose spacing between the surface discretizations and the sampled curvenet. ©Disney/Pixar

deformed surface mesh preserves the offset relative to the deformed curvenet. To this end, we consider the copy of each curvenet sample with target position $\mathbf{q}_i$ and its corresponding deformation gradient matrix $\mathbf{F}_i$ indicated by the unfolded i-th row from the constraint matrix $\mathbf{f}_c$. We then estimate the target projected position of the curvenet sample as $\mathbf{p}_i = \mathbf{q}_i - \mathbf{F}_i(\breve{\mathbf{q}}_i - \breve{\mathbf{p}}_i)$. Note that the left and right copies of each curvenet sample may receive different projection offsets based on the deformation gradient matrix from each side of the curvenet segments. We rearrange these adjusted sample points $\{\mathbf{p}_i\}$ into the rows of the matrix $\mathbf{x}_c$ of size $n_c \times 3$ representing the curvenet positional constraints, and then compute the matrix of positions $\mathbf{x}_v$ for the mesh vertices via

$$\min_{\mathbf{x}_v} E_D(\mathbf{V}\,\mathbf{x}_v + \mathbf{C}\,\mathbf{x}_c - \mathbf{y}_h). \tag{5}$$

Finally, we point out that both Eqs. (4) and (5) correspond to unconstrained convex optimizations and, therefore, they can be minimized by solving a sequence of linear systems of the form

$$\begin{cases} (\mathbf{V}^{\mathrm{t}}\,\mathbf{L}_h\mathbf{V})\,\mathbf{f}_v = -\mathbf{V}^{\mathrm{t}}\mathbf{L}_h(\mathbf{C}\,\mathbf{f}_c), \\ (\mathbf{V}^{\mathrm{t}}\,\mathbf{L}_h\mathbf{V})\,\mathbf{x}_v = -\mathbf{V}^{\mathrm{t}}\mathbf{L}_h(\mathbf{C}\,\mathbf{x}_c - \mathbf{y}_h). \end{cases} \tag{6}$$

Notice that the shared left-hand side matrix $\mathbf{V}^{\mathrm{t}}\,\mathbf{L}_h\mathbf{V}$ defines a modified Laplacian operator restricted to the vertices of the input surface mesh that incorporates the contributions of the sub-polygons generated by the mesh cutting with the sampled curvenet, while retaining an one-ring sparsity pattern similar to the original polygonal mesh. Since this Laplacian matrix is assembled only once in the neutral pose, we can precompute a sparse factorization of this matrix and find the solutions of Eq. (6) through a direct solver.

5 RESULTS

In this section, we present a series of examples showcasing the visual quality, performance, and versatility of our method for character articulation based on curvenets. We also point the reader to the accompanying video which provides several animation sequences and interactive sessions using our technique.

Implementation: We developed the tools for modeling and rigging curvenets as well as our surface deformation solver as C++ plug-ins within the Pixar animation system (Presto). Once the character model is loaded, the user can construct curvenets by drawing cubic Bézier splines with shared endpoints over the surface and weight their knots to animation controllers using archetypal spatial deformers, as previously described in §3. The supplemental video includes a live session illustrating the creation and the articulation of a curvenet for the elbow of a human model. For optimized performance, we precompute the cut-mesh and the factorization of its Laplacian matrix only once using the neutral pose in which the curvenet is built. The surface deformation is then generated at runtime as the animator exercises the curvenet rig. Both the sparse matrix decomposition and the linear solves in Eq. (6) are computed using [Chen et al. 2008]. In Table 1, we report timings for all our examples split into mesh cutting, matrix factorization, and surface deformation,

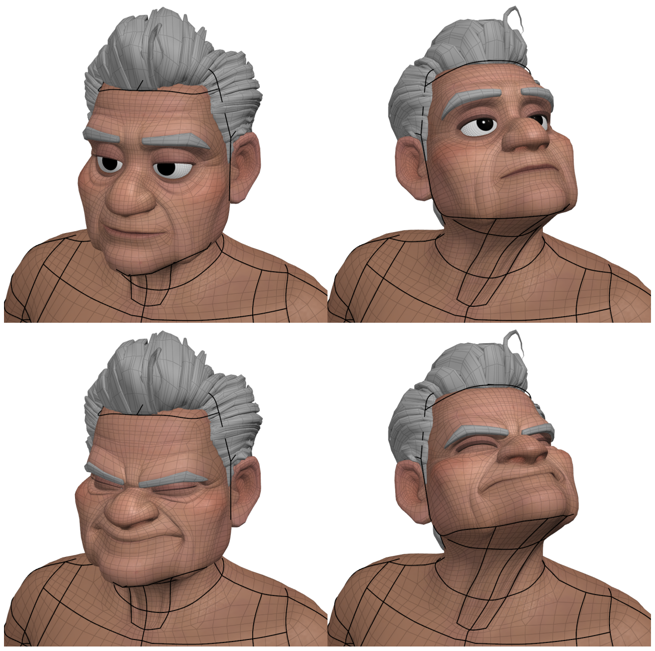


Fig. 9. **Hybrid Rig:** This example shows a hybrid rig setup that combines a curvenet body articulation with a face rig made of patch and joint deformers. The top row indicates the neutral configuration used as the projection pose in our precomputation step (left), followed by the head deformation produced by our method (right). The bottom row illustrates the expression generated using the face rig (left), which defines the rest pose used by our solver to compute the layered face-head deformation (right). ©Disney/Pixar

clocked on a 2.3 GHz Intel Xeon E5-2699 with 18 cores. Note that our solver achieves interactive rates even on high-resolution meshes.

Examples: Figure 1 shows a sequence of poses for a Panda character entirely articulated by our method, followed by facial animations in Figure 2. In this Panda rig, we used one curvenet for the torso and limbs and a second curvenet for the face, which we built with extra curves around the eyebrows and lips in order to provide more localized shaping controls. We also included additional curvenets for posing the detached ear meshes. Figure 8 shows a hand deformation computed on various polygonal tessellations, demonstrating that our method produces qualitatively similar results robust to the mesh connectivity. These hand meshes were all articulated using the same rigged curvenet displayed in Figure 4. We present in Figure 3 several calisthenic poses for a human model articulated by a pair of hand curvenets and a body curvenet. In particular, the body component was constructed enclosing the head but with no curve on the face. Since our approach bounds the deformation on each curve side, we can use the curves surrounding the face to isolate the facial features while tracking the head deformation. Our method is also well-suited for handling overlapping articulations such as the neck and shoulder combo. This is in sharp contrast to traditional rigging schemes that can involve multiple corrective sculpts to resolve blending artifacts. Please see our supplemental video for complete animation clips and additional examples.

Projection versus Rest: So far we have used the same neutral pose for projecting the curvenet onto the surface mesh and also to define the undeformed surface configuration. However, some rigging setups may benefit from hybrid approaches that layer the curvenet articulation with other surface deformations. In order to support more flexible rigs, we devised a small modification of our algorithm that accounts for different projection and rest poses of the input surface mesh. We indicate the projection pose as the surface shape in which the curvenet is designed, e.g., the model T-pose. After constructing the curvenet, we employ the projection pose to resume precomputing the cut-mesh and the Laplacian-based linear solver. Conversely, the rest pose represents the result of any surface deformation performed before the curvenet articulation. Since the curvenet and the cut-mesh are created in the projection pose, we warp both of them to the shape of the rest surface by reusing the binding of the curvenet samples onto the closest points of the surface mesh previously cached by our cutting routine. Given these warped poses, we evaluate the rest pose values for the scaled frames along the sampled curvenet as well as the undeformed cut-face polygons, which are then used by our surface optimization. Figure 9 shows a hybrid setup that combines a face rig made of patch and joint deformers with the body deformation controlled by curvenets. In this example, we also include separate meshes for the hair, brows, and eyeballs attached to the body surface. Figure 7 presents another layered deformation that edits a garment mesh via curvenets after running a cloth simulation. This rig configuration is useful for incorporating shot-specific and view-dependent corrections by directly manipulating the simulated shapes instead of rerunning the offline simulation. Since the projection pose is persistent throughout the animation frames, we can deform these simulated shapes efficiently by leveraging our precomputed curvenet solver.

Blend Shapes: In addition to hybrid rigs, our method can also be coupled with blend shapes that enhance the surface deformations with mesh-specific details finer than the curvenet resolution. We superpose blend shapes as vertex offsets relative to the rest surface mesh and then use vertex frames to reconstruct these displacements on top of the shape produced by the curvenet deformation. Figure 10 exhibits snarling sculpts triggered by the curvenet rig when posing the curves associated with the muzzle of the Panda model. Note

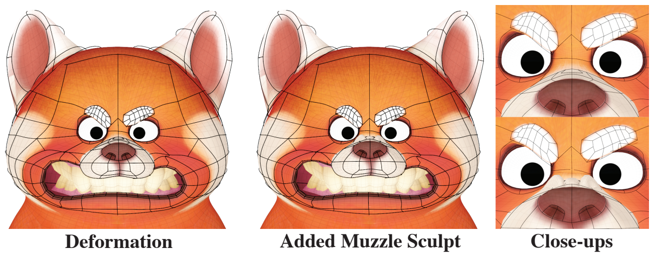


Fig. 10. **Blend Shapes:** Curvenet rigs can be superposed by blend shapes sculpted on the high-resolution surface mesh. The left image shows a face articulation of the Panda character fully controlled by the curvenet, while the middle image exhibits muzzle sculpts applied on top of the curvenet deformation that emphasizes the snarling expression. Close-ups of the muzzle before and after the sculpt are displayed in the right column. ©Disney/Pixar

that we represent blend shapes separated from the surface rest pose so that vertex sculpts can be animated without recomputing the curvenet deformation, but they can also be included as part of the rest configuration if preferred by the rigger. It is also worth pointing out that we add sculpts only when high frequency mesh details are needed, in contrast to typical pose-space deformation methods that employ blend shapes to resolve deformation artifacts.

Discussion: Our formulation for character articulation bears many similarities with existing detail-preserving deformation techniques [Botsch and Sorkine 2008]. In particular, we adopt a two-step optimization akin to [Zayer et al. 2005] that first interpolates local transformations using harmonic functions and then reconstructs the surface mesh via a Poisson solve. However, we account for arbitrary deformation gradient matrices including non-uniform stretching, while previous methods employed specialized parametrizations limited to similarity transformations. Moreover, we advocate for separate boundary conditions imposed on each curve side and estimated directly from a rigged curvenet, instead of prescribing pointwise constraints via user handles. Figure 11 compares the deformed surface produced by discontinuous versus averaged curve constraints. Observe how the latter leads to an undesirable bowing shape (see side-view), mixing the deformation from one curve side to the other, while our result enforces the hinge discontinuity keeping the mesh planar on both curve sides. Another key difference is our surface discretization that cuts the input mesh through the profile curves. By implementing a cut-aware Laplacian (see Appendix A), we obtain smooth interpolations robust to the presence of irregular and cracked mesh polygons, bypassing any need for mesh repairs. As a byproduct, our approach can also be used to compute diffusion curve attributes over non-flat polygonal surfaces. Figure 13 shows an example of discontinuous colors diffused from a collection of closed curves over a quadrangulated shirt mesh.

Limitations: Our solver tends to cause volume loss when the curvenet is under large twisting deformations. Figure 12 illustrates surface meshes deformed by a curvenet twist of 90 and 180 degrees with volumes reduced to 93% and 81% of the original value, respectively. We note that prior Poisson-like methods have reported similar issues and attempted to remediate them via corrective steps in exchange of increased computational cost [Lipman et al. 2007; Sorkine and Alexa 2007]. We also argue that the twist shapes generated by our method are superior than the typical linear blend skinning deformation and are competitive to skinning variants like those surveyed by Jacobson et al. [2014]. Although our implementation supports multiple curvenets cutting the same surface mesh, we assume these curvenets do not overlap each other. Moreover, we focus our discretization on control curves that cut across multiple mesh elements, as expected by characteristic surface profiles, but our Laplacian assembly (Appendix A) can be generalized to support cut-faces with interior boundaries defined by curvenet islands inside individual mesh faces. Pointwise handles are also straightforward to incorporate into our formulation via soft constraints weighted by barycentric coordinates. Finally, we point out that, even though curvenets can be densely constructed in order to shape local features, our approach is most advantageous when rigging curves representing broad surface deformations and then add mesh-specific details using spatial deformers or blend shapes.

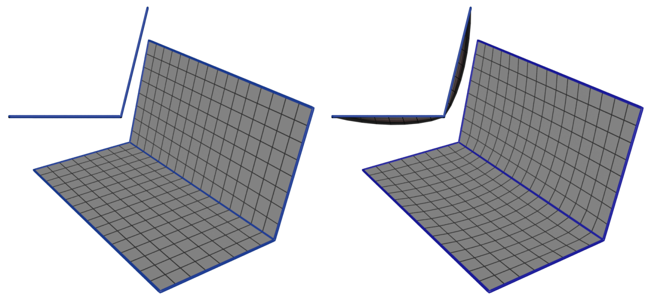


Fig. 11. **Discontinuous curve constraints:** The left image shows the surface mesh computed by our method that interpolates distinct rotation constraints imposed on each curve side. In contrast, the right image displays the result of interpolating the averaged rotation assigned to the central curve. Observe in the side-views that our approach reproduces the piecewise-flat mesh configuration instead of introducing an undesirable bowing shape that diffuses the averaged hinge orientation towards the outline curves.

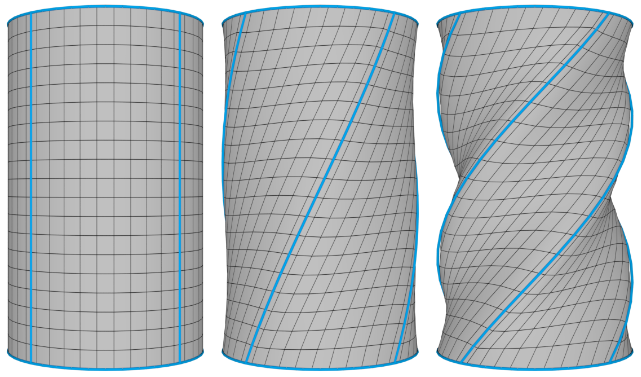


Fig. 12. **Limitation:** Our approach to propagate the curvenet articulation to the surface mesh can cause volume loss under large twisting deformations. In this example, a cylindrical surface (left) has its volume reduced to 93% of the original value after deforming the curvenet by a twist of 90 degrees (middle), and to 81% after a twist of 180 degrees (right).

6 CONCLUSION

We have presented a new technique for character articulation that generates detail-preserving deformations controlled by curvenets. Our curvenet representation simplifies the construction of character rigs by detangling the articulation controllers and skinning weights from the surface mesh connectivity. As a result, we can author character deformations consistent across different mesh tessellations by directly manipulating prominent surface profiles and using rigs with orders of magnitude fewer weighted points. At its core, our approach introduces a novel mesh cutting scheme accompanied by a robust polygonal discretization that allows the numerical optimization of surface deformations constrained by curve discontinuities.

In addition to conforming surface meshes to curvenets, we have also described how to combine curvenet-based articulations with other deformation workflows such as skeletal rigs, sculpting, and offline simulations. We have demonstrated our method on a number of character animations used in production settings.

As future work, we are interested in speeding up our solver by computing localized solutions a la [Herholz et al. 2017] that update the mesh deformation per patches triggered by changes in nearby curvenet segments. We are also investigating extensions of our surface interpolation using higher-order smoothness terms [Stein et al. 2020] and including more flexible boundary conditions with discontinuities [Boyé et al. 2012]. It would also be interesting to revisit tearing and cracking simulations based on our polygonal discretization, since we can handle complex cut layouts while avoiding quadrature approximations [Nguyen-Thanh et al. 2018]. Finally, combining surface cutting with shape optimization is an exciting direction in order to devise robust curve-aware geometry processing applications.

ACKNOWLEDGMENTS

We are grateful to Danielle Feinberg and Christian Hoffman for their support in adopting our technique in Pixar's feature film *Turning Red.* Andrew Butts provided invaluable assistance with the Presto animation system and Emma Staniforth helped with the UI development. We thank Jared Fong, Jeremie Talbot, and Kris Campbell for feedback during early deployment on *Onward* and *Soul.* We also thank Francesco Olcese, Grace Gilbert, Jacob Speirs, Jason Davies, Mark Piretti, Michael Comet, and Michael Nieves for their collaboration on *Lightyear.* Finally, we thank Mark Meyer, Mathieu Desbrun, and Doug L. James for proofreading this document.

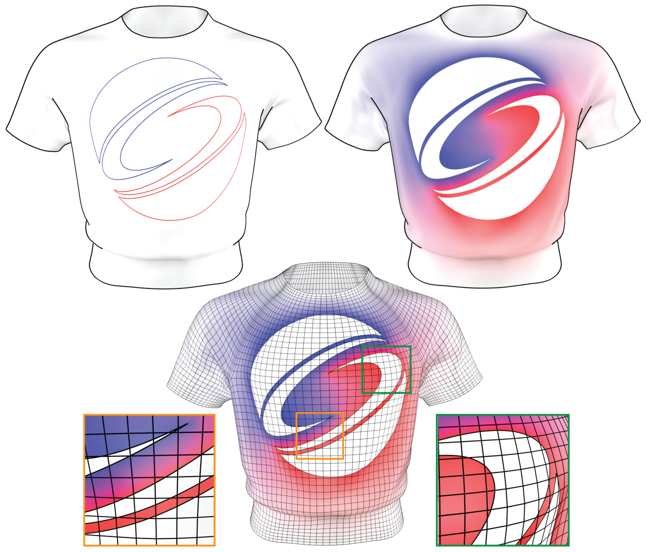


Fig. 13. **Diffusion Curves:** Our cut-aware discretization offers an extension of diffusion curves [Orzan et al. 2008] to non-planar surface meshes. The top-left image shows blue and red closed curves tracing the ACM SIGGRAPH logo on top of a quadrangulated shirt mesh. The top-right image displays the diffusion curve shading generated by our method that smoothly interpolates the curve colors outside the logo while keeping the inside and the surface boundaries white. The bottom row overlays the diffused colors with a wireframe of the cut-mesh, including close-ups of the split polygons with the curve discontinuities.

Table 1. **Timing:** Wall clock time in milliseconds spent by our method to generate the cut-mesh (*Cutting*), assemble and factorize the cut-aware Laplacian matrix (*Factor*), and compute the surface deformation (*Solve*), all measured on a 2.3 GHz Intel Xeon E5-2699 with 18 cores. We also report the statistics for the surface mesh resolution, the number of Bézier knots used by the curvenets, and the sample count discretizing the curvenets in our examples. Note that curvenets simplify the character rig using orders of magnitude fewer knots than vertices in the surface mesh.

Model	#Faces	#Verts	#Knots	#Samples	Cutting	Factor	Solve
Panda	22388	22390	2535	34157	149.1	97.8	9.7
Human	17650	17649	894	24079	101.8	69.9	7.9
Jacket	22354	11778	220	4795	32.8	45.3	6.2
Hand Quad	1376	1387	448	3577	17.9	8.9	1.2
Hand Tri	2896	1459	448	3439	33.4	11.8	1.4
Hand Shell	2606	2618	448	5041	27.3	14.6	1.5
Hand Subd	10426	10447	448	10610	51.1	46.2	6.3
Hand Dense	93895	93991	448	32460	153.7	408.2	35.4

REFERENCES

E. Benvenuti, A. Chiozzi, G. Manzini, and N. Sukumar. 2019. Extended virtual element method for the Laplace problem with singularities and discontinuities. *Computer Methods in Applied Mechanics and Engineering* 356 (2019), 571–597.
M. Berger. 2017. Chapter 1 - Cut Cells: Meshes and Solvers. In *Handbook of Numerical Methods for Hyperbolic Problems*, R. Abgrall and C.-W. Shu (Eds.). Handbook of Numerical Analysis, Vol. 18. Elsevier, 1–22.
M. Botsch, M. Pauly, M. Gross, and L. Kobbelt. 2006. PriMo: Coupled Prisms for Intuitive Surface Modeling. In *Symposium on Geometry Processing*. 11–20.
M. Botsch and O. Sorkine. 2008. On Linear Variational Surface Deformation Methods. *IEEE Transactions on Visualization and Computer Graphics* 14, 1 (2008), 213–230.
S. Boyé, P. Barla, and G. Guennebaud. 2012. A Vectorial Solver for Free-Form Vector Gradients. *ACM Transactions on Graphics* 31, 6, Article 173 (2012), 9 pages.
E. Burman, S. Claus, P. Hansbo, M. G. Larson, and A. Massing. 2015. CutFEM: Discretizing geometry and partial differential equations. *Internat. J. Numer. Methods Engrg.* 104, 7 (2015), 472–501.
S. Calderon and T. Boubekeur. 2017. Bounding Proxies for Shape Approximation. *ACM Transactions on Graphics* 36, 4, Article 57 (2017), 13 pages.
M. Campen. 2017. Partitioning Surfaces into Quadrilateral Patches: A Survey. In *Proc. of the European Association for Computer Graphics: Tutorials*. Article 5, 25 pages.
Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. 2008. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Software* 35, 3 (2008), 14 pages.
F. de Goes, A. Butts, and M. Desbrun. 2020. Discrete Differential Operators on Polygonal Meshes. *ACM Transactions on Graphics* 39, 4, Article 110 (2020), 14 pages.
F. de Goes, S. Goldenstein, M. Desbrun, and L. Velho. 2011. Exoskeleton: Curve Network Abstraction for 3D Shapes. *Computer and Graphics* 35, 1 (2011), 112–121.
T.-P. Fries and T. Belytschko. 2010. The extended/generalized finite element method: An overview of the method and its applications. *Internat. J. Numer. Methods Engrg.* 84, 3 (2010), 253–304.
R. Gal, O. Sorkine, N. J. Mitra, and D. Cohen-Or. 2009. iWires: An Analyze-and-Edit Approach to Shape Manipulation. *ACM Transactions on Graphics* 28, 3, Article 33 (2009), 10 pages.
G. Gori, A. Sheffer, N. Vining, E. Rosales, N. Carr, and T. Ju. 2017. FlowRep: Descriptive Curve Networks for Free-Form Design Shapes. *ACM Transactions on Graphics* 36, 4, Article 59 (2017), 14 pages.
P. Herholz, T. A. Davis, and M. Alexa. 2017. Localized Solutions of Sparse Linear Systems for Geometry Processing. *ACM Transactions on Graphics* 36, 6, Article 183 (2017), 8 pages.
Z. Huang, N. Carr, and T. Ju. 2019. Variational Implicit Point Set Surfaces. *ACM Transactions on Graphics* 38, 4, Article 124 (2019), 13 pages.
A. Jacobson, I. Baran, L. Kavan, J. Popović, and O. Sorkine. 2012. Fast Automatic Skinning Transformations. *ACM Transactions on Graphics* 31, 4, Article 77 (2012),

10 pages.

A. Jacobson, Z. Deng, L. Kavan, and J.P. Lewis. 2014. Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH Courses*.

P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. 2007. Harmonic Coordinates for Character Articulation. *ACM Transactions on Graphics* 26, 3 (2007), 10 pages.

T. Ju, Q.-Y. Zhou, M. van de Panne, D. Cohen-Or, and U. Neumann. 2008. Reusable Skinning Templates Using Cage-Based Deformations. *ACM Transactions on Graphics* 27, 5, Article 122 (2008), 10 pages.

L. Kavan and O. Sorkine. 2012. Elasticity-Inspired Deformers for Character Articulation. *ACM Transactions on Graphics* 31, 6, Article 196 (2012), 8 pages.

B. H. Le and J. P. Lewis. 2019. Direct Delta Mush Skinning and Variants. *ACM Transactions on Graphics* 38, 4, Article 113 (2019), 13 pages.

B. H. Le, K. Villeneuve, and C. Gonzalez-Ochoa. 2021. Direct Delta Mush Skinning Compression with Continuous Examples. *ACM Transactions on Graphics* 40, 4, Article 72 (2021), 13 pages.

J. P. Lewis, M. Cordner, and N. Fong. 2000. Pose Space Deformation: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation. In *Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 165–172.

P. Li, K. Aberman, R. Hanocka, L. Liu, O. Sorkine-Hornung, and B. Chen. 2021. Learning Skeletal Articulations with Neural Blend Shapes. *ACM Transactions on Graphics* 40, 4 (2021), 1.

Y. Lipman, D. Cohen-Or, R. Gal, and D. Levin. 2007. Volume and Shape Preservation via Moving Frame Manipulation. *ACM Transactions on Graphics* 26, 1 (2007), 14 pages.

Y. Lipman, D. Levin, and D. Cohen-Or. 2008. Green Coordinates. *ACM Transactions on Graphics* 27, 3 (2008), 1–10.

Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. 2005. Linear Rotation-invariant Coordinates for Meshes. *ACM Transactions on Graphics* 24, 3 (2005), 479–487.

V. Lucquin, S. Deguy, and T. Boubekeur. 2017. SeamCut: Interactive Mesh Segmentation for Parameterization. In *ACM SIGGRAPH 2017 Technical Briefs*.

J. Mancewicz, M. L. Derksen, H. Rijpkema, and C. A. Wilson. 2014. Delta Mush: Smoothing Deformations While Preserving Detail. In *Symposium on Digital Production*. 7–11.

T. McLaughlin, L. Cutler, and D. Coleman. 2011. Character Rigging, Deformations, and Simulations in Film and Game Production. In *ACM SIGGRAPH Courses*.

N. Moës, J. Dolbow, and T. Belytschko. 1999. A finite element method for crack growth without remeshing. *Internat. J. Numer. Methods Engrg.* 46, 1 (1999), 131–150.

S. E. Mousavi, E. Grinspun, and N. Sukumar. 2011. Harmonic enrichment functions: A unified treatment of multiple, intersecting and branched cracks in the extended finite element method. *Internat. J. Numer. Methods Engrg.* 85, 10 (2011), 1306–1322.

A. Nealen, T. Igarashi, Olga Sorkine, and M. Alexa. 2007. FiberMesh: Designing Freeform Surfaces with 3D Curves. *ACM Transactions on Graphics* 26, 3, Article 41 (2007).

A. Nealen, O. Sorkine, M. Alexa, and D. Cohen-Or. 2005. A Sketch-Based Interface for Detail-Preserving Mesh Editing. *ACM Transactions on Graphics* 24, 3 (2005), 1142–1147.

V. M. Nguyen-Thanh, X. Zhuang, H. Nguyen-Xuan, T. Rabczuk, and P. Wriggers. 2018. A Virtual Element Method for 2D linear elastic fracture analysis. *Computer Methods in Applied Mechanics and Engineering* 340 (2018), 366–395.

A. Orzan, A. Bousseau, P. Barla, H. Winnemöller, J. Thollot, and D. Salesin. 2008. Diffusion Curves: A Vector Representation for Smooth-Shaded Images. *ACM Transactions on Graphics* 27, 3 (2008), 8 pages.

H. Pan, Y. Liu, A. Sheffer, N. Vining, C.-J. Li, and W. Wang. 2015. Flow Aligned Surfacing of Curve Networks. *ACM Transactions on Graphics* 34, 4, Article 127 (2015), 10 pages.

K. Polthier and M. Schmies. 1998. Straightest Geodesics on Polyhedral Surfaces. In *Mathematical Visualization: Algorithms, Applications and Numerics*. 135–150.

S. Schaefer, J. Warren, and D. Zorin. 2004. Lofting Curve Networks Using Subdivision Surfaces. In *Symposium on Geometry Processing*. 103–114.

K. Singh and E. Fiume. 1998. Wires: A Geometric Deformation Technique. In *Proc. of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 405–414.

O. Sorkine and M. Alexa. 2007. As-Rigid-As-Possible Surface Modeling. In *Symposium on Geometry Processing*. 109–116.

O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. 2004. Laplacian Surface Editing. In *Symposium on Geometry Processing*. 175–184.

O. Stein, A. Jacobson, M. Wardetzky, and E. Grinspun. 2020. A Smoothness Energy without Boundary Distortion for Curved Surfaces. *ACM Transactions on Graphics* 39, 3, Article 18 (2020), 17 pages.

T. Sun, P. Thamjaroenporn, and C. Zheng. 2014. Fast Multipole Representation of Diffusion Curves and Points. *ACM Transactions on Graphics* 33, 4, Article 53 (2014), 12 pages.

K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi. 2010. Volumetric Modeling with Diffusion Surfaces. *ACM Transactions on Graphics* 29, 6, Article 180 (2010), 8 pages.

M. Tao, C. Batty, E. Fiume, and D. I. W. Levin. 2019. Mandoline: Robust Cut-Cell Generation for Arbitrary Triangle Meshes. *ACM Transactions on Graphics* 38, 6, Article 179 (2019), 17 pages.

J.-M. Thiery, P. Memari, and T. Boubekeur. 2018. Mean Value Coordinates for Quad Cages in 3D. *ACM Transactions on Graphics* 37, 6, Article 229 (2018), 14 pages.

R. Vaillant, L. Barthe, G. Guennebaud, M.-P. Cani, D. Rohmer, B. Wyvill, O. Gourmel, and M. Paulin. 2013. Implicit Skinning: Real-Time Skin Deformation with Contact Modeling. *ACM Transactions on Graphics* 32, 4, Article 125 (2013), 12 pages.

R. Vaillant, G. Guennebaud, L. Barthe, B. Wyvill, and M.-P. Cani. 2014. Robust Iso-Surface Tracking for Interactive Character Skinning. *ACM Transactions on Graphics* 33, 6, Article 189 (2014), 11 pages.

W. Wang, B. Jüttler, D. Zheng, and Y. Liu. 2008. Computation of Rotation Minimizing Frames. *ACM Transactions on Graphics* 27, 1, Article 2 (2008), 18 pages.

Y. Wang and J. Solomon. 2021. Fast Quasi-Harmonic Weights for Geometric Data Interpolation. *ACM Transactions on Graphics* 40, 4, Article 73 (2021), 15 pages.

J. Wu, R. Westermann, and C. Dick. 2015. A Survey of Physically Based Simulation of Cuts in Deformable Bodies. *Computer Graphics Forum* 34, 6 (2015), 161–187.

Z. Xu, Y. Zhou, E. Kalogerakis, C. Landreth, and K. Singh. 2020. RigNet: Neural Rigging for Articulated Characters. *ACM Transactions on Graphics* 39, 4, Article 58 (2020), 14 pages.

Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. 2004. Mesh Editing with Poisson-Based Gradient Field Manipulation. *ACM Transactions on Graphics* 23, 3 (2004), 644–651.

R. Zayer, C. Roessl, Z. Karni, and H.-P. Seidel. 2005. Harmonic Guidance for Surface Deformation. *Computer Graphics Forum* 24, 3 (2005), 601–609.

Q. Zhou, T. Weinkauf, and O. Sorkine. 2011. Feature-Based Mesh Editing. In *Proc. Eurographics, Short Papers*.

A DISCRETE LAPLACIAN MATRIX

In this appendix, we revisit the construction of the discrete Laplacian operator for 3D polygons derived in [de Goes et al. 2020]. Consider the simple polygon associated with the cut-face f defined by the loop of n_f halfedges. We use the rows of the matrix $\mathbf{X}_f$ of size $n_f \times 3$ to indicate the 3D position of the polygon's corners ordered counter-clockwise in a reference configuration. We also define the matrix $\mathbf{D}_f$ of size $n_f \times n_f$ that computes the differences between consecutive corners in f, i.e., $\mathbf{D}_f^{i,i+1} = 1$, $\mathbf{D}_f^{i,i} = -1$, and zero otherwise. The edge vectors delineating f can then be expressed by $\mathbf{E}_f = \mathbf{D}_f\mathbf{X}_f$. Lastly, we denote the matrix $\mathbf{A}_f$ of size $n_f \times n_f$ that averages consecutive face-corners, i.e., $\mathbf{A}_f^{i,i+1} = \mathbf{A}_f^{i,i} = 1/2$, and zero otherwise.

Equipped with these matrices, we compute the polygonal vector area $\mathbf{a}_f$ integrated over f via $[\mathbf{a}_f] = \mathbf{E}^t\mathbf{A}_f\mathbf{X}_f$, where $[\mathbf{a}_f]$ indicates the 3×3 skew-symmetric matrix such that $[\mathbf{a}_f]\mathbf{u} = \mathbf{a}_f \times \mathbf{u}$ for any 3D vector $\mathbf{u}$. We then assign the area of the cut-face f to $a_f = \|\mathbf{a}_f\|$ and define $\mathbf{n}_f = \mathbf{a}_f / a_f$ as its constant normal vector. The discrete gradient operator of the cut-face f is assembled by the matrix

$$\mathbf{G}_f = (-1/a_f)[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f, \qquad (7)$$

thus mapping the n_f values sampled at the face-corners to a vector perpendicular to the polygonal normal $\mathbf{n}_f$. Note that crack edges within the cut-face f contribute to the discrete gradient operator $\mathbf{G}_f$ with a term that measures the difference of face-corner values averaged on both sides of the crack multiplied by the edge vector tracing the crack rotated by $\pi/2$ around $\mathbf{n}_f$ and divided by a_f.

Since the cut-face polygon is of arbitrary size, its matrix $\mathbf{G}_f$ can have a non-trivial null-space of dimension $n_f - 2$ that accounts for non-linear discrete functions expressed within f with zero gradient. To quantify this null-space, we introduce the projection operator

$$\mathbf{Q}_f = \mathbf{D}_f - \mathbf{E}_f\mathbf{G}_f, \qquad (8)$$

which is a matrix of size $n_f \times n_f$ that returns the difference of face-corner values per oriented edge subtracted by the alignment of the gradient vector relative to each edge vector. In particular, the projection operator identifies the colinear columns in $\mathbf{G}_f$ introduced by the duplicated crack edges in f. We also point out that the matrix $\mathbf{Q}_f$ corresponds to a restriction of the projection operator $\mathbf{P}_f$ for discrete one-forms proposed by de Goes et al. [2020] (see their Eq. 12) to discrete functions via $\mathbf{D}_f$, that is, $\mathbf{Q}_f = \mathbf{P}_f \mathbf{D}_f$.

Given a discrete function $\boldsymbol{\phi}_f$ of size n_f defined over the cut-face f, the Dirichlet energy evaluates the smoothness of $\boldsymbol{\phi}_f$ by penalizing the squared norm of its gradient vector as well as the edge differences for the non-linear part of $\boldsymbol{\phi}_f$, yielding

$$E_D(\boldsymbol{\phi}_f) = a_f \|\mathbf{G}_f \boldsymbol{\phi}_f\|^2 + \lambda \|\mathbf{Q}_f \boldsymbol{\phi}_f\|^2, \tag{9}$$

where $\lambda = 1$ as recommended by de Goes et al. [2020]. Finally, we expand the face-based Dirichlet energy into the quadratic form $E_D(\boldsymbol{\phi}_f) = \boldsymbol{\phi}_f^{\mathrm{t}} \mathbf{L}_f \boldsymbol{\phi}_f$, and obtain the discrete Laplacian operator for the cut-face f as the matrix of size $n_f \times n_f$ of the form

$$\mathbf{L}_f = a_f \mathbf{G}_f^{\mathrm{t}} \mathbf{G}_f + \lambda \mathbf{Q}_f^{\mathrm{t}} \mathbf{Q}_f. \tag{10}$$

As shown by de Goes et al. [2020], the Laplacian matrix $\mathbf{L}_f$ is symmetric, scale-invariant, positive semi-definite, linear precise in the planar domain, and reproduces the well-known cotan weights when the cut-face is a triangle. Moreover, this discretization handles polygons with cracks by including cut discontinuities in the gradient vector and filtering out spurious modes via the projection matrix.