Normal-Driven Spherical Shape Analogies

Hsueh-Ti Derek Liu and Alec Jacobson
University of Toronto

Abstract
This paper introduces a new method to stylize 3D geometry. The key observation is that the surface normal is an effective instrument to capture different geometric styles. Centered around this observation, we cast stylization as a shape analogy problem, where the analogy relationship is defined on the surface normal. This formulation can deform a 3D shape into different styles within a single framework. One can plug-and-play different target styles by providing an exemplar shape or an energy-based style description (e.g., developable surfaces). Our surface stylization methodology enables Normal Captures as a geometric counterpart to material captures (MatCaps) used in rendering, and the prototypical concept of Spherical Shape Analogies as a geometric counterpart to image analogies in image processing.

1. Introduction
Analogies of the form $A : A' :: B : B'$ is a reasoning process that conveys $A$ is to $A'$ as $B$ is to $B'$. This formulation has become a core technique for creating artistic 2D digital content, such as image analogies [HJO*01] in Photoshop [Ado21] for image stylization and the Lit Sphere [SMGG01] (a.k.a. MatCap) in ZBrush [Pix20] for non-photorealistic renderings. However, leveraging analogies to stylize 3D geometry is still at a preliminary stage because defining the analogy relationship on surface meshes requires dealing with irregular discretizations, curved metrics, and different topologies.

In this paper, we introduce a step towards a more general 3D shape analogies, named spherical shape analogies. We consider a specific case where $A$ is a unit sphere. This restriction enables us to operate on an input mesh $B$ with arbitrary topologies, boundaries, and geometric complexity. While not fully general, because $A$ is restricted to be a sphere, we demonstrate that this formulation can immediately achieve different geometric styles within a single framework. In Fig. 1, we show that by providing different target style shapes $A'$ to the algorithm, we can turn the input shape $B$ into different styles. In addition to stylization, our method can encompass many existing applications, such as developable surface approximation and PolyCube deformation.

One key observation in our spherical shape analogies is that the surface normal is an effective instrument to capture geometric styles. Thus, we define the analogy relationship based on normals: we optimize a stylized shape $B'$ such that the relationship between the surface normals of $B$ and $B'$ is the same as the relationship between the surface normals of $A$ and $A'$.

We realize this by casting it as a simple and effective normal-driven shape optimization problem which aims at deforming the input shape towards a set of desired normals. However, such an optimization problem is often difficult due to the nonlinearity of unit normals. We draw inspiration from previous works and apply a change of variables to accelerate the computation: instead of di-
directly optimizing the vertex positions, we optimize a set of rotations that rotate the normals of the input mesh to the set of desired normals. Our simple formulation with the change of variables results in a generic stylization algorithm that runs at interactive rates.

2. Related Work

Our work shares similar motivations to computer-assisted image stylization pioneered by Haeberli [Hae90]. But since our outputs are stylized 3D geometries, we focus the discussion on geometric stylization and geometric deformation methods.

**Analogy-based Geometric Stylistation**

Many generative models have been proposed for creating stylized 3D objects, such as collage art [GSP*07; TRAS07], manga style [SLHC12], cubic style [LJ19], and neuronal homunculus [RRS12]. However, these methods are tailor-made for only a specific style.

**Analogy A : A′ ∼ B : B′** is a powerful idea to achieve different stylization results within a single framework. This idea has inspired several design tools for images [HJO*01], non-photorealistic renderings [SMGG01; FJL*16], and curves [HOCS02]. Beyond 2D data, the idea of analogy has also been used for transferring 3D geometric details from one shape to another. We omit the discussion on methods that are not based on analogies, such as mesh cloning [ZHWH*06; TSS*11] and geometric learning [KLC*20; HHGC20; WAK*20; CKP*21; LZ21], and focus on analogy-based techniques. Ma et al. [MHS*14] propose a method for 3D style transfer based on patch-based assembly. However, their method cannot handle free-form deformations and requires the source and the exemplar shape to share a similar structure in order to compute high-quality correspondences. Bhat et al. [BIT04] propose a voxel-based texture synthesis method for transferring geometric details encoded in the volumetric grid. Berkiten et al. [BHS*17] use metric learning for details represented as displacement maps. These methods are designed for high-frequency details (e.g., wrinkles on the surface). In contrast, our spherical shape analogies focuses on larger scale free-form deformations. Albeit limited — in our analogies A is restricted to the unit sphere — our method enables a first step in this exciting direction.

**Surface Normals in Shape Deformation**

A key insight of our spherical shape analogies is to leverage surface normals to capture geometric styles. The surface normal is a fundamental geometric quantity and is ubiquitous in geometry processing. A representative example is in the PolyCube deformation [THCM04] where the goal is to optimize surface normals to be axis-aligned. Gregson et al. [GSZ11] and Zhao et al. [ZLL*17] use the closest rotation from the surface normal to an axis-aligned direction to drive the PolyCube deformation. Huang et al. [HJS*14] and Fu et al. [FBL16] propose to minimize energies defined on normals to create PolyCube shapes. In architectural geometry design, surface normals are a main ingredient to characterize polygon meshes with planar faces. The methods proposed by Deng et al. [DPW11] and Poranne et al. [POG13] utilize norms to formulate a distance-from-plane constraint to encourage planarity. Tang et al. [TSG*14] use the dot product between a face normal and its adjacent edge vectors to determine whether the vertices of a polygon are coplanar. Characterizing whether a mesh can be flattened to 2D without stretching or shearing, a.k.a. developability, also relies on surface normals. Stein et al. [SGC18] characterize discrete developability based on the 1-ring face normals, and pro-
pose an algorithm to compute piecewise developable surfaces. Sellán et al. [SAJ20] reformulate the developable energy into a convex semidefinite program for finding piecewise developable height-fields. In addition to these examples, deforming shapes into the cubic style [LJ19; FMR20], constructing shape abstractions [Ale21], surface parameterization [ZSL+20], and interactive mesh editing [YZX+04; SCL+04] are all related to surface normals. Many more examples can be found in the design of geometric filters, such as the Guided filter [ZDZ+15], the Shock filter [PK15], the Bilateral normal filter [ZFAT11], and the Total Variation mesh denoising [ZWZD15].

Our method can be adapted to these normal-based deformations. Compared to the PolyCube method [FBL16], we achieve comparable quality (see Fig. 2), but we can further generalize to polytopes (see Fig. 18). Compared to [LJ19] in cubic stylization (see Fig. 3), we can achieve similar performance, but we can further generalize to many styles other than the cubic style (see Fig. 1). In developable surface approximation, in contrast to the method by Sellán et al. [SAJ20], our method can be applied to surface triangle meshes (see Fig. 4) and is significantly faster than the method by Stein et al. [SGC18] (see Fig. 5).

3. Spherical Shape Analogies

Our main idea is to use surface normals to capture the style of 3D objects: if two shapes share a similar normal “profile”, we consider them to exhibit the same geometric style. Centered around this observation, as discussed in Sec. 1, we propose an analogy-based stylization method to translate the relationship between the normals of $A'$ and $A$ to create a stylized output shape $B'$ (see Fig. 6). Throughout the paper, we use green color to denote the target style shape $A'$, gray color to denote the input shape $A$, and blue color to denote the output stylized shape $B'$.

Figure 7: Our algorithm defines the analogous relation based on the surface normals. We first map the normals of the style shape $N_A$ to a unit sphere to obtain $\tilde{N}_A$ (top row), transfer the relationship between $N_A$ and $\tilde{N}_A$ to the input shape to obtain the target normals $T$ (middle row), then optimize the input shape $B$ so that the actual output normals are aligned with the target normal $T$ (bottom row).

3.1. Generating $N_{A'}$

Depending on the provided style shape $A'$ or user preferences, we consider three ways to get a set of target normals on a sphere $\tilde{N}_{A'}$.

1. Closest normals. The simplest case is when the style shape $A'$ is a simple convex shape with only few distinct face normals (e.g., icosahedron). We compute $\tilde{N}_{A'}$ simply via snapping the normals of the sphere $N_{A'}$ to the nearest normal in the style shape $N_A$.

2. Spherical parameterization. For a generic genus-0 shape (e.g., smooth or concave), we compute its parameterization to a sphere using, for example, formalized mean curvature flow [KSB12]. Then $\tilde{N}_{A'}$ can be computed from the spherical parameterization.

3. Normal Capture. If one desires more control, one can manually specify $\tilde{N}_{A'}$, in the spirit of how MatCap (material capture
3.2. Generating $T$

Generating target normals $T$ on the input shape $B$ using analogy requires the correspondences between $A$, $B$. We compute the map using the Gauss map, leveraging the fact that our $A$ is always a unit sphere (see the inset, where we use colors to visualize the correspondences). Specifically, the unit normal vector of each element (e.g., vertex or face) on the input shape $B$ can be equivalently interpreted as a point on the unit sphere $A$. Thus, we can easily map signals from $A$ back to $B$. Once the correspondences are obtained via the normals of input shape $B$, we can trivially compute $T$ by “pasting” $\hat{N}_A$ on top of $B$.

3.3. Generating $B'$

After obtaining a set of target normals $T = \{t_k\}$ for each vertex $k$, our goal is to obtain a deformed output shape $B'$ whose surface normals approximate $T$. Let $V$ be a matrix of vertex locations with size $|V|$-by-3 and $F$ be the face list with size $|F|$-by-3 of the input shape $B$. Our output shape $B'$ is a deformed version of the input shape and we use $V'$ to denote the $|V|$-by-3 matrix of the deformed vertex locations. We formulate the normal-driven deformation as an energy optimization in the following form:

$$\min_{V' \in \mathbb{R}^{3 \times |V'|}} \sum_{k \in V} E_k(v_k, u_k) + \lambda_1 \|\hat{n}_{k}(V') - t_k\|^2_2,$$

(1)

where $E_k$ denotes a regularizing energy to preserve the details of the input mesh, and the second part measures the squared distance from the output unit surface normal $\hat{n}_{k}(V')$ to the target output normal $t_k$ at vertex $k$. We use $u_k$ to denote the Voronoi area of the vertex $k$, $\lambda$ is a weighting parameter to control the balance between the two terms, and $v_k (u_k)$ is the input (output) location of vertex $k$. In Fig. 9, we can observe that using a small $\lambda$, the method preserves the input shape $B$. Using a bigger $\lambda$, the method favors in deforming the shape more into the style of $A'$. The choice of $E_k$ depends on the user’s intent. One can apply different regularizations to obtain different results. For the purposes of this exposition, we introduce our optimization based on ARAP regularization in Sec. 3.4. We discuss how to extend to other regularizations in Sec. 4.1.

3.4. Normal-Driven Optimization with ARAP

We use $e_{ij} := v_j - v_i \in \mathbb{R}^3$ to denote the edge vector between vertices $i, j$ on the original mesh, and $e'_{ij} := v'_j - v'_i$ for the edge vectors on the deformed mesh. We can write down the energy that uses ARAP regularization as

$$\min_{V' \in \mathbb{R}^{3 \times |V'|}} \sum_{k \in V} \sum_{i \in N_k} w_{ij} ||R_k e_{ij} - e'_{ij}||^2_2 + \lambda_2 ||\hat{n}_{i}(V') - t_k||^2_2.$$ 

(2)

We use $N_k$ to denote the edge vectors of the spikes and rims at vertex $k$ (see the inset) [CPSS10]. $R_k \in SO(3)$ to denote a 3-by-3 rotation matrix defined on $k$, and $w_{ij}$ is the cotangent weight of edge $i, j$ [PP93]. However, this energy is difficult to optimize because the term $\hat{n}_{i}(V')$ is non-linear in $V'$.

We adapt the observation made in [LJ19] that the space of unit vectors can be captured by rotations. Thus, we can perform a change of variables by replacing $\hat{n}_{i}(V')$ with the rotated unit normal of the input mesh $R_k \hat{n}_{k}$ as

$$\min_{V' \in \mathbb{R}^{3 \times |V'|}} \sum_{k \in V} \sum_{i \in N_k} w_{ij} ||R_k e_{ij} - e'_{ij}||^2_2 + \lambda_2 ||R_k \hat{n}_{k} - t_k||^2_2.$$ 

(3)

where $\hat{n}_{k}$ is the $k$th unit vertex normal of the input mesh computed via area-weighted average of face normals, which is constant throughout the optimization. This $R_k \hat{n}_{k}$ can be perceived as an approximation of the area-weighted vertex normals of the output mesh $\hat{n}_{k}(V')$. In Fig. 10, we visualize the difference between the output normals $\hat{n}_{k}(V')$ and the rotated input normals $R_k \hat{n}_{k}$. We can notice that $R_k \hat{n}_{k}$ is a decent approximation of the output vertex normals computed via area-weighted average. We can observe that error tend to concentrate on high-curvature regions because...
discrete vertex normals are less accurate along on those regions and the ARAP regularization encourages smooth deformation. This change of variables allows us to solve for $R_s$ in parallel and make this energy quadratic in $V'$. In addition, the fact that $R_s$ is shared across the ARAP term and the normal term enables us to jointly consider both the regularization and the normal terms when obtaining the deformed vertex locations $V'$. We minimize this energy via the local/global strategy [SA07], where the local step involves solving a set of small orthogonal Procrustes problems and the global step amounts to a linear solve. For the sake of reproducibility, we reiterate the local-global steps for our energy in App. A, B. Non-linear methods, such as Newton’s method, could be applied to our scenario. It is however far slower than the local-global optimization since a single iteration of the Newton’s method could be more expensive than 100 iterations of the local-global iterations (see [LBK17]). Thus, it is less suitable for our interactive applications. Further accelerating our solver using other optimization methods (e.g., [KGL16; PDZ*18; ZBK18]) should be possible, but is left as future work.

4. Extensions & Analysis

In this section, we introduce its extensions to different regularizations and how to handle cases where target normals $T(B')$ are a function of output geometry.

4.1. Different Regularizations

In addition to $E_{ARAP}$, the normal-driven optimization supports different regularization energies for different modeling intents. One could use ARAP when the goal is to produce a smooth deformation that preserves surface details. If one wants to produce a non-smooth deformation (e.g., sharp creases) while preserving local rigidity, one could instead use a face-only ARAP energy $E_{FARAP}$ discussed in [ZG16; LG15] which consists of only the membrane term. If one is interested in preserving the textures and allowing local scaling, one could use an as-conformal-as-possible energy $E_{ACAP}$ [BDS*12].

Face-only ARAP. The core idea is to remove the bending term from ARAP and only measure the membrane term [TPBF87], so that two adjacent triangles can bend freely. We achieve this by applying the idea from [ZG16; LG15] which only measures the ARAP energy over the three edge vectors of a face $f_k$ (see the inset), instead of the spokes and rims $N_f$. Precisely, we can write this “face-only” ARAP regularization $E_{FARAP}$ as

$$E_{FARAP}(V', R) = \sum_{k \in F} \sum_{i,j \in f_k} w_{ij} ||R_k(e_{ij} - e'_{ij})||^2.$$  \hfill (4)

As-conformal-as-possible. If the goal is to create novel geometric details, it is crucial to allow non-rigid deformations. However, an arbitrary deformation may lead to undesirable behaviors, such as badly shaped triangles. Thus constraining the angle preservation, a.k.a. conformity, will be a suitable regularization. Specifically, we use the ACAP energy $E_{ACAP}$ in [BDS*12] as our regularization

$$E_{ACAP}(V', R, s) = \sum_{k \in F} \sum_{i,j \in N_k} w_{ij} ||s_k R_k e_{ij} - e'_{ij}||^2,$$  \hfill (5)

where $s_k$ is a scalar representing the scaling of local patch. One can compute the optimal $s_k$ analytically via the method by Schönemann et al. [SC70] (see App. C).

Deploying these regularizations $E_{FARAP}, E_{ACAP}$ requires only a few changes in the optimization steps. Deploying $E_{FARAP}$ only involves changing the incidence matrix. Deploying $E_{ACAP}$ only requires adding one more line of code in the local step to solve an isotropic orthogonal Procrustes problem [SC70]. We detail such changes in App. C. In Fig. 11, we apply the same deformation to a sheet but with different regularizations. We can perceive that different regularizations favor drastically different solutions.

Our framework allows one to easily plug-and-play different regularizations. Specifically, we use $E_{ARAP}$ for applications that favor smooth deformation (e.g., Fig. 13), $E_{FARAP}$ for creating sharp creases (Fig. 18, 20), and $E_{ACAP}$ when one wants to manipulate geometric details such as in Fig. 22.

4.2. Dynamic Target Normals

Our method converges to a local minimum. Empirically, we observe that treating the target normal $T$ as a constant throughout the optimization may work fine perceptually in many cases (see the left pair in Fig. 12). However, constant $T$ may lead to an undesirable local minimum due to a sub-optimal assignment of $T$ (see the right pair in Fig. 12). Inspired by Projective Dynamics [BML*14], a simple solution to avoid such local minima is to treat $T$ as a function of $B'$ ($N_{B'}$ specifically), and update $T$ at every iteration. We...
Algorithm 1: Normal-driven optimization

**Input:** A triangle mesh \( V, F \) and a weight \( \lambda \)

**Output:** Deformed vertex positions \( V' \)

1. compute \( \tilde{N}_A' \) // step 1, Sec. 3.1
2. compute \( T \) from \( \hat{n}(V) \) // step 2, Sec. 3.2
3. \( \hat{n} \leftarrow \hat{n}(V) \) // compute input surface normals
4. \( Q, K \leftarrow \text{precompute}(V, F) \) // see App. B
5. \( V' \leftarrow V \)
6. while not converge do
7. \( \hat{R} \leftarrow \text{local-step}(V', \hat{n}, T, \lambda) \) // App. A
8. \( V' \leftarrow \text{global-step}(\hat{R}, Q, K) \) // App. B
9. compute \( T \) from \( \hat{n}(V') \) // (optional) for dynamic \( T \)

summarize the pseudo code in Alg. 1. If \( T \) is a constant throughout the optimization, one can simply skip the optional step at line 9.

In terms of convergence, in the case where \( T \) is constant, the convergence behaves the same as the original ARAP [SA07], where the energy decreases monotonically. In the case where \( T \) is dependent to \( B' \), we do not guarantee a monotonic decrease in energy, but the optimization still converges in our experiments. In the inset, we visualize the convergence plot for examples in Fig. 12.

We implement our algorithm in C++ with Eigen [GJ10] and evaluate our method on a MacBook Pro with an Intel i5 2.3GHz processor. Our method runs 24 iterations per second for a mesh with around 20k vertices. We report a complete picture of our runtime in the inset. The local step will be the computation bottleneck for meshes with less than 20k vertices, but further acceleration can be achieved via the method by Zhang et al. [ZJA21]. Typically, within the first 10 iterations, our method can achieve a visually similar result compared to the converged solution. This property enables us to build an interactive tool for users to play with different style shapes \( A' \) or artistic controls.
5. Applications

The major benefit of our analogy-based stylization method is that one can plug-and-play different style shapes to obtain different results. When one provides convex primitives with few distinct face normals, we can simply use the method discussed in Sec. 3.1 to turn an input shape into the style of the primitive (see Fig. 13, 14). In Fig. 15, we also quantitatively show that our method can effectively reduce the difference between mesh normals and the normals of a primitive. If the provided style shape is smooth or non-convex, where the simple closest normal may fail to capture the style, one could use a spherical parameterization described in Sec. 3.1 to achieve the stylization. If desiring more user controls, one could “paint” the desired surface normals on a unit sphere (see Sec. 3.1), and then transfer the style of the painted normals directly to the input (see Fig. 17).

5.1. PolyCube Deformation

If one is interested in PolyCube maps [THCM04], we can adapt normal driven editing to create PolyCube maps, following the observation in [ZLL*17]. Specifically, we need to use a cube as a style shape and move the pre-computation step in Alg. 1 to the optimization loop. Moving the pre-computation in the loop would no longer preserve the original details, which is desirable for creating PolyCube shapes. This modification may also lead to badly shaped triangle. When these faces appear, a quick solution is to move the vertex towards the 1-ring average by a small amount to improve triangle quality. For the sake of comparison, we use the same PolyCube segmentation as in [FBL16] and show that we can achieve comparable results in Fig. 2. We can further generalize the PolyCube method (left) to create polygonal boxed maps.

Figure 18: By using different sets of normals, we can generalize the PolyCube method (left) to create polygonal boxed maps.

5.2. Developable Surface Approximation

So far we have only considered an explicit shape or a set of painted normals as our style shape. Here we further extend our method to support an energy that describes a certain style. In particular, we consider the target normal $T$ is computed via an optimization

$$T = \arg \min_T f(B'), \quad (6)$$

and, similar to the case where $T$ is dependent to $B'$, we update $T$ at every iteration in the local/global solve.

We evaluate this extension via setting $f$ to be the discrete developability energy proposed in [SGC18], with details provided in App. E. Compared to the original method, our approach contains a regularization term in addition to the developable energy, thus our optimization requires no remeshing and results in the faster optimization (see Fig. 5). In Fig. 19, we further show that our framework enables one to control the number of creases in the results. With our framework one can interactively create a variety of piece-wise developable shapes (see Fig. 20). In Fig. 21, we evaluate our results by visualizing the discrete Gaussian curvature before and after running our developable flow. We can observe that the Gaussian curvature concentrates along the creases and results in a piece-wise developable surface. In the inset, we quantitatively demonstrate that our method effectively increases the developability of the mesh in Fig. 21.
6. Limitations & Future Work

Our method draws inspiration from Projective Dynamics [BML*14] to handle the case where target normals $T$ are a function of output shape $B'$ (e.g., Fig. 12, 20). Although being fast and suitable for our intended interactive applications, it often struggles to converge to a highly accurate solution. Extending our optimization to, for example, Newton’s method would be desirable for applications that desire highly accurate solutions.

Our approach is restricted to a sphere as our reference shape $A$, and uses the Gauss map to determine the correspondences between $A$ and the input $B$. As the Gauss map purely relies on surface normals to determine the map, the resulting map is ignorant to area distortion. This characteristic is beneficial to handle input shapes $B$ that are very different (e.g., different genus) from a sphere because in these cases it is challenging to obtain a map with low area distortion. However, the price we have to pay is that we cannot support structured and high-frequency patterns (e.g., geometric texture synthesis). Thus, if one is interested in stylizing shapes with detailed textures, we suggest to first synthesize target normals on the surface directly [WLKT09] then perform the normal-driven optimization (Sec. 3.4). In Fig. 22 we demonstrate this alternative by unbaking an existing normal map for manufacturing purposes (see the inset) and synthesizing normal textures from an image.

Our method currently supports manifold triangle meshes. Extending to non-manifold meshes, polygon meshes, volumetric meshes, and point clouds could be beneficial to handle real-world geometric data. Not every shape or normal capture sphere is valid to serve as the style shape of our algorithm. Discovering the validity of a style shape is important to understand the behavior of these novel modeling methods. Removing the assumption about the source shape being a sphere could lead to a more general analogy-based shape editing. Based on the observation that surface normals are a promising geometric quantity to capture the style of a shape. Developing a better categorization of styles based on normals or exploring learning-based techniques on normals (instead of vertices) could lead to novel stylization methods.

Acknowledgements

Our research is funded in part by NSERC Discovery (RGPIN2017–05235, RGPAS–2017–507938), New Frontiers of Research Fund (NFRF–201), the Ontario Early Research Award program, the Canada Research Chairs Program, the Fields Centre for Quantitative Analysis and Modelling and gifts by Adobe Systems, Autodesk and MESH Inc. We thank Sheldon Andrews, Abhishek Madan, Silvia Sellán, Oded Stein, Li-Yi Wei for helps on experiments. We thank members of Dynamic Graphics Project at the University of Toronto; Sarah Kushner, Abhishek Madan, Silvia Sellán, Letícia Mattos Da Silva, Towaki Takikawa for proofreading; John Hancock for the IT support. We thank all the artists for sharing a rich variety of 3D models.

References


Given a fixed \( V' \), we obtain the optimal rotation for each vertex \( k \) by solving the following minimization problem

\[
R_k = \arg \min_{R \in \text{SO}(3)} \sum_{i,j \in N_k} w_{ij} \| R_k e_{ij} - e'_k \|^2 + \lambda \| R_k t_k - t_k' \|^2
\]

The above optimization is an instance of the orthogonal Procrustes which finds the best rotation matrix \( R_k \) to map a set of vectors \( (e_{ij}, \hat{e}_k) \) to another set of vectors \( (e'_k, t_k) \). We can re-write it into a more compact expression as:

\[
R_k^* = \arg \max_{R \in \text{SO}(3)} \text{Tr}(R_k X_k)
\]

\[
X_k = [E_k \quad \hat{e}_k] \begin{bmatrix} W_k & \lambda \mu_k \lambda_k \frac{e'_k'}{t_k'} \end{bmatrix}.
\]

Where \( W_k \) is a \( |V_k| \times |V_k| \) diagonal matrix of the cotangent weights \( w_{ij} \), \( E_k \) and \( E'_k \) are \( 3 \times |V_k| \) matrices concatenating the edge vectors of the face one-ring at the rest and deformed states, respectively. One can then derive the optimal \( R_k \) from the SVD of \( X_k = U_k \Sigma_k U_k^T \).}

\[ R_k = \text{argmax}_{R \in \text{SO}(3)} \text{Tr}(R_k X_k).
\]

up to changing the sign of the column of \( U_k \) so that \( \text{det}(R_k) > 0 \).

\[
R_k = \text{argmax}_{R \in \text{SO}(3)} \text{Tr}(R_k X_k).
\]
to solving the following problem
\[ V^* = \arg \min_{V'} \sum_{k \in E} \sum_{i,j} w_{ij} \| R_k e_{ij} - \hat{e}_{ij} \|_2^2 \]

We can expand this energy as
\[
\sum_{k \in E} \sum_{i,j} w_{ij} \| R_k e_{ij} - \hat{e}_{ij} \|_2^2 \\
= \sum_{k \in E} \sum_{i,j} w_{ij} \hat{e}_{ij}^T e_{ij} - 2w_{ij} \hat{e}_{ij}^T R_k e_{ij} + \text{constant}
\]

It is often convenient to express the summation in terms of matrices.

We introduce a directed incidence matrix \( A_k \) with size \(|V|\)-by-\(|N_k|\) to represent the edge vectors in \( N_k \) as \( V^T A_k \), and we use \( M_k \) to represent a \(|N_k|\)-by-\(|N_k|\) diagonal matrix of the weights \( w_{ij} \). Then we can re-write the energy in terms of matrices as
\[
\sum_{k \in E} \text{Tr}(M_k A_k^T V^T V^T A_k) - 2 \text{Tr}(M_k A_k^T V R_k V^T A_k) \\
= \sum_{k \in E} \text{Tr}(V^T A_k M_k A_k^T V') - 2 \text{Tr}(R_k V^T A_k M_k A_k^T V') \\
= \text{Tr} \left( V^T \left( \sum_k (A_k M_k A_k^T) V' \right) \right) - 2 \text{Tr} \left( \left( \sum_k R_k V^T A_k M_k A_k^T \right) V' \right) \\
= \text{Tr} \left( V^T QV' \right) - 2 \text{Tr}(RKV'),
\]

where \( R = \{ R_k \} \) is the concatenation of all the rotations, \( Q \) is a \(|V|\)-by-\(|V|\) symmetric matrix, and \( K \) is a \(|V|\)-by-\(|3V|\) matrix stacking the constant terms which can be computed during the precomputation. We can then find the optimal \( V' \) by solving a linear system
\[ QV' = K^T R^T \]

As we know from [SA07], \( Q \) is the cotangent Laplacian [PP93]. We can pre-factorize \( Q \) to speed up performance. With these pieces in hand, we can minimize our energy Eq. 3 by iteratively performing the local and the global steps (see Alg. 1).

Appendix C: Generalize to \( E_{\text{FARAP}} \) and \( E_{\text{ACAP}} \)

Changing the regularization from \( E_{\text{FARAP}} \) to the membrane-only regularization \( E_{\text{FARAP}} \) (Eq. 4) requires to re-define \( R \) on each face and change the set of edge vectors to the three edge vectors of a triangle. These changes would lead us to replace the \( E_k, E_k' \) in the local step Eq. 7 to the three edge vectors of a face, and \( a_k \) to the face area. In the global step, one only needs to update the incidence matrix \( A_k \) in Eq. 10 to a \(|V|\)-by-\(|f_k|\) matrix containing the three edge vectors information.

Deploying the as-conformal-as-possible regularization \( E_{\text{ACAP}} \) (Eq. 5) changes the local step to solve an instance of the isotropic orthogonal Procrustes problem, where an analytical solution has been derived in [SC70]. In short, one can obtain the optimal rotation the same way as Eq. 9, and compute the optimal scaling \( s_k \) analytically as
\[
s_k = \frac{\text{Tr}(W_k E_k'^T R_k E_k) + \lambda a_k \hat{n}_k^T t_k}{\text{Tr}(W_k E_k'^T E_k) + \lambda a_k \hat{n}_k^T t_k}.
\]

When assembling the matrices for the global step, using \( E_{\text{ACAP}} \) would require replacing \( R_k \) with \( s_k R_k \).

Appendix D: Projective Dynamics for Dynamic Target Normals

We draw inspiration from projective dynamics [BML*14] to handle cases where the target normal \( T \) is a function of output geometry \( B' \). Let us first define
\[ T = \arg \min_{V'} E_N(V') \]
as a minimizer of an energy \( E_N \) defined on the output shape. In our cases, \( E_N \) could be the distance to the closest normals or the developable energy [SGC18]. With this definition, we re-write Eq. 3 as
\[
\min_{V', R} \sum_{k \in E} \sum_{i,j} w_{ij} \| R_k e_{ij} - \hat{e}_{ij} \|_2^2 + \lambda a_k \| R_k \hat{e}_k - t_k \|_2^2,
\]
subject to \( T = \arg \min_{E_N(V')} \)

This reformulation allows us to directly deploy the projective dynamics solver by first projecting \( T = \{ t_k \} \) to the “constraint” \( E_N \), fixing \( t_k \), and solving the original problem as Eq. 3 via the local/global solver to get \( V' \) at the next iteration. We then iterate this procedure (see Alg. 1) until convergence. This expression enables us to plug-and-play different \( E_N \) for different modeling objectives.

Appendix E: Normal Driven Developable Surfaces

Our normal-driven editing can be used to create developable surfaces by specifying a set of target normals that are developable. Stein et al. [SGC18] propose a characterization of discrete developability based on face normals of a vertex one-ring. In short, if all the one-ring face normals correspond to a common plane or two planes, then this local one-ring is piecewise developable.

With this characterization, we can easily get a set of “developable” face normals by (1) visiting all the one-ring faces of a vertex, (2) performing a small principle component analysis on the face normals for each one-ring, and (3) projecting the normals to one or two common planes by zeroing out the components corresponding to the smallest eigenvalues. By using a different threshold to decide whether to zero out the smallest or the smallest two components, we can control the amount of creases in the developable approximation (see Fig. 19). As each face will receive three (possibly) different developable normals from the previous procedure, we simply average them to get the target face normals. We perform this developable normal computation at each iteration in parallel, which corresponds to the Line 9 of Alg. 1.