Supplementary Material for Stroke Transfer: Example-based Synthesis of Animatable Stroke Styles

Hideki Todo*
Aoyama Gakuin University (AGU)
Kanagawa, Japan
htodo@cs.takushoku-u.ac.jp

Kunihiko Kobayashi*
Aoyama Gakuin University (AGU)
Kanagawa, Japan
kuni.koba.one-piece@outlook.com

Jin Katsuragi
Aoyama Gakuin University (AGU)
Kanagawa, Japan
jincgcg@gmail.com

Haruna Shimotahira
Aoyama Gakuin University (AGU)
Kanagawa, Japan
suansushui@gmail.com

Shizuo Kaji
Kyushu University
Fukuoka, Japan
skaji@imi.kyushu-u.ac.jp

Yonghao Yue
Aoyama Gakuin University (AGU)
Kanagawa, Japan
yonghao@it.aoyama.ac.jp

ACM Reference Format:
Hideki Todo, Kunihiko Kobayashi, Jin Katsuragi, Haruna Shimotahira, Shizuo Kaji, and Yonghao Yue. 2022. Supplementary Material for Stroke Transfer: Example-based Synthesis of Animatable Stroke Styles. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings (SIGGRAPH '22 Conference Proceedings), August 7–11, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3528233.3530703

1 DETAILED STEPS

1.1 Shading Model for Reference Renderings

Our reference rendering computes the (pre-tone-mapping) colors $\boldsymbol{I}$ (represented in the regular R, G, and B channels) lit by a few point light sources, each with an intensity $\boldsymbol{L}_j$, using a modified (normalized) Blinn-Phong model [Gotanda 2010]. For simplicity, we have omitted the Fresnel term from the BRDF, and disabled the fall off due to the inverse-square law (to avoid regions getting dark too quickly). The j-th light source gives rise to the j-th diffuse color $\boldsymbol{I}_{j,d}$ as

$$\boldsymbol{I}_{j,d} \coloneqq \boldsymbol{L}_j \odot \frac{\boldsymbol{\rho}_d}{\pi} \cos\theta_{l_j}, \tag{1}$$

as well as the specular color $\boldsymbol{I}_{j,s}$ as

$$\boldsymbol{I}_{j,s} \coloneqq \boldsymbol{L}_j \odot \frac{\boldsymbol{\rho}_s(n+2)}{4\pi(2-2^{-n/2})} \frac{\cos^n\theta_{h_j}\cos\theta_{l_j}}{\max(\cos\theta_{l_j},\cos\theta_e)}, \tag{2}$$

where $\odot$ is the component-wise multiplication, and $\boldsymbol{\rho}_d$ and $\boldsymbol{\rho}_s$ respectively represent the diffuse and specular reflection coefficients. A non-negative real number n describes the magnitude of the glossiness (the larger n, the closer to the perfect mirror), and θ_{l_j} (resp. θ_{h_j}, and θ_e) is the angle between the surface normal and the j-th lighting direction (resp. the j-th half vector, and the viewing direction). Obtuse angles are clumped to the right angle. Of course, the user can use their own customized lighting/shading model instead.

We define the total diffuse color $\boldsymbol{I}_d$, the total specular color $\boldsymbol{I}_s$, and the total color $\boldsymbol{I}$ as the following sums:

$$\boldsymbol{I}_d \coloneqq \sum_j \boldsymbol{I}_{j,d}, \tag{3}$$

$$\boldsymbol{I}_s \coloneqq \sum_j \boldsymbol{I}_{j,s}, \tag{4}$$

and

$$\boldsymbol{I} \coloneqq \boldsymbol{I}_d + \boldsymbol{I}_s. \tag{5}$$

The resulting colors $\boldsymbol{I}_d$, $\boldsymbol{I}_s$, and $\boldsymbol{I}$ are stored in a high dynamic range format to avoid excess quantization or saturation (so that we can take the derivatives later on).

1.2 Tone Mapping

The raw intensity value $\boldsymbol{I}_*$ (referring to $\boldsymbol{I}_d$, $\boldsymbol{I}_s$, or $\boldsymbol{I}$) may have an extremely large value, due to the existence of strong highlighting, a particular choice of the shading model, or the use of a light probe image as the light source (instead of the point one). Hence, a tone mapping should usually be used to produce a low dynamic range of intensities for display.

For the transfer of the stroke styles, we ask 1) the tone mapped values to be *bounded*, because otherwise we would need to (unrealistically and impractically) acquire style data covering a range of intensity too wide. We also ask 2) the tone mapping to be designed such that we can *consistently* assign styles across scenes. This consistency might be interpreted differently depending on the actual use of our stroke transfer. The user may want to tie the styles with a) the tone mapped intensities *relative* to the maximum intensity, or b) the *absolute* tone mapped values. Both cases refer to the displayed colors, but the canonical white value is scene-dependent (or even frame-dependent) in the former one. For the former case, the user can use an existing tone mapping operator, like the one by Reinhard et al. [2002]. For the latter case, we design a simple, customized tone mapping such that it 1) produces a bounded intensity and 2) preserves the intensity values in the usual low dynamic range (between 0 and 1) as much as possible.

We first convert the raw intensity $\boldsymbol{I}_*$ into the CIE 1976 $L^*a^*b^*$ (or simply CIELAB) color space. Let L^* be its luminance, and a^*

*Co-first authors — authors contributed equally.

SIGGRAPH '22 Conference Proceedings, August 7–11, 2022, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9337-9/22/08…$15.00
https://doi.org/10.1145/3528233.3530703

and b^* be the color channels. Then, our tone map T converts L^* to $L^{*\prime} = T(L^*)$. The color channels a^* and b^* are left unchanged. We obtain the final color by converting $L^{*\prime}$, a^*, and b^* back into the RGB color space.

We design the tone mapping T in the form of a sigmoid function. Specifically, we use the following transformed logistic function

$$T(x; L_M, \Theta_T) := L_M \left(2 \frac{\exp(\Theta_T x)}{\exp(\Theta_T x) + 1} - 1 \right) \tag{6}$$

with two parameters L_M, which adjusts the range of values to $(-L_M, L_M)$, and Θ_T, which controls the speed approaching the limiting values L_M or $-L_M$. Note that $T(x; L_M, \Theta_T)$ is anti-symmetric (i.e., $T(-x; L_M, \Theta_T) = -T(x; L_M, \Theta_T)$), so in particular, it maps $L^* = 0$ to $L^{*\prime} = 0$. Moreover, its second-order derivative is negative for $x \in (0, \infty)$, meaning the speed of increase of $L^{*\prime}$ slows down *monotonically*.

The parameters L_M and Θ_T are chosen to maximize the compression ratio while keeping a low deviation of the converted value $T(x; L_M, \Theta_T)$ from the raw value x in the low dynamic range $[0, 1]$. More precisely, we solve the following optimization problem:

$$(L_M, \Theta_T) = \underset{\tilde{L}_M, \tilde{\Theta}_T}{\operatorname{argmin}} \tilde{L}_M \quad \text{s.t.} \quad \max_{x \in [0,1]} \| x - T(x; \tilde{L}_M, \tilde{\Theta}_T) \| \leq 0.05. \tag{7}$$

Numerical search finds $(L_M, \Theta_T) = (1.39, 1.67)$.

1.3 Exemplar and Annotations

The final color converted from $L^{*\prime} = T(L^*; L_M, \Theta_T)$, a^*, and b^* are given to the artist to provide an exemplar (by overdrawing the strokes atop the reference rendering). Although regions with an exceeding luminance will saturate when displayed on a low dynamic range display, such regions only appear at a specular highlight, and a human is pretty good at extrapolating the luminance at such a region.

For each exemplar, we let an annotator (can be the artist themselves) to give annotations for the orientations, lengths, and widths of the strokes for (usually) a small *subset* of the strokes. To simplify the interactions for the annotations, we let the annotator to approximate each annotated stroke with a sequence of connected line segments. The annotator is also asked to give the width (a single scalar per stroke, measured in the screen space) for each stroke.

Note that because our targets are thick painting media, many strokes are only partially visible (the occluded or overdrawn regions are completely invisible). Assigning the length and/or width of a stroke only accounting for its visible region may not correctly obey the artist's intention; if possible, the length and width for its entire region should be provided instead.

1.4 Preparation for Regression

Prior to regression using the exemplar and annotations, we prepare the necessary data. These include the 1) stroke data, 2) features, and 3) canonical sections. Again, because we focus on thick painting media, we only care about visible styles: an extreme case would be that there are completely different styles underneath the visible parts; our model for thick media should not be influenced by such invisible data. A simple way to incorporate this is to process the exemplar per visible pixel (i.e., if a stroke spans over multiple visible pixels, we will have a piece of data for every such pixel; this will also naturally weigh a stroke more if its visible region is larger), leading to our error functional (4) in the main paper or its discrete forms (18), (22), or (24) discussed later.

1.5 Stroke Data

For each annotated stroke, we first fit a spline curve to its annotated sequence of line segments. Its arc length in the screen space is given as the stroke's length. While the colors are already available per pixel, we need to interpolate the orientations, lengths, and widths from the sparsely given annotations.

We first sample seed points on each fitted spline curve. At each seed point, we assign the annotated width and the arc length (both measured in the screen space) as the width and length data. For the orientations, we first compute the tangent vectors in the screen space $\boldsymbol{t}^{\mathrm{sc}}_{q_k}$ along the spline curves at the seed points q_k. These tangent vectors $\boldsymbol{t}^{\mathrm{sc}}_{q_k}$ are then pushed forward to the object space vectors $\boldsymbol{t}^{\mathrm{obj}}_{q_k}$ lying in the tangent space of the object surface (i.e., $\boldsymbol{t}^{\mathrm{obj}}_{q_k} = \Pi^{-1}(\boldsymbol{t}^{\mathrm{sc}}_{q_k})$[1]). This push forward operator is well-defined and one-to-one. Please see Section 1.7 for how to compute $\boldsymbol{t}^{\mathrm{obj}}_{q_k}$.

We choose to define the orientation in the object space to respect the underlying 3D geometry, while the width and length in the screen space to respect the dimension of the brush and the motion of the hand relative to the screen (but these can also be configured in the object space depending on the taste of interest).

From the quantities $\bar{B}_k$ (these are any of the orientations, widths, and lengths) defined on the seed points $\bar{q}_k$, we use radial basis functions to interpolate the values $B(q)$ at any location q on the screen (in a specified subregion for the case of partial exemplar, or the entire region excluding the background otherwise) as:

$$B(q) = \sum_k B_k \Psi(\|q - q_k\|; \Theta_\Psi), \tag{8}$$

where $\|q - q_k\|$ is the distance in the screen space. We choose the bases $\Psi(\|q - q_k\|; \Theta_\Psi)$ to be *multiquadric* given by

$$\Psi(x; \Theta_\Psi) = \sqrt{1 + \frac{x^2}{\Theta_\Psi^2}}, \tag{9}$$

where the parameter Θ_Ψ is the average of the screen space distances of all pairs of seed points (the default setting of SciPy). For the orientations, we normalize them again after interpolation. In our prototype implementation, we employ the interpolation to find the quantities at pixel centers.

We verify the validity of the interpolated orientations, lengths, and widths (most often only visually) to make sure that the annotations are sufficient and representative. The validity could be violated if the annotations are too sparse. For such a case, the annotator is asked to add more annotations.

[1] We are using the operator Π^{-1} (and also Π) to refer two related things. The first is the point correspondence: when we write $p = \Pi^{-1}(q)$, this means the inverse projection of the point q on the screen to the point $p \in M$. The second is the correspondence between tangent vectors: when we write $\boldsymbol{t}^{\mathrm{obj}}_{q_k} = \Pi^{-1}(\boldsymbol{t}^{\mathrm{sc}}_{q_k})$, this means the push forward of the tangent vector. The second is naturally induced from the first. It should be clear from the context which one we are referring to.

1.6 Extracting Features

As the features at a pixel q, we extract

- diffuse intensity $I_d(q) := T(L^*(\boldsymbol{I}_d(q))$, where $L^*(\boldsymbol{I}_d(q))$ is the luminance of $\boldsymbol{I}_d(q)$ computed from (3),
- specular intensity $I_s(q) := T(L^*(\boldsymbol{I}_s(q)))$, likewise from (4),
- apparent (diffuse) intensity gradient $I_d^{\nabla_2}(q) := \|\nabla_2 I_d(q)\|$ (we are excluding the gradient of specular because a highlight is usually localized in a small region and its gradient is not so informative), computed by taking finite differences,
- distance from silhouettes $d_S(q)$, computed using opencv [OpenCV 2015],
- Gaussian curvature $\kappa(\Pi^{-1}(q))$, computed using libigl [Jacobson et al. 2018]
- mean curvature $H(\Pi^{-1}(q))$, again computed using libigl [Jacobson et al. 2018], and
- apparent normal $(\tilde{n}_x(q), \tilde{n}_y(q), \tilde{n}_z(q)) := \boldsymbol{M}^{\mathrm{mv}}(\boldsymbol{n}(\Pi^{-1}(q)))$, where $\boldsymbol{M}^{\mathrm{mv}}$ is the model-view matrix; $\tilde{n}_z(q) = 1$ if the surface at $\Pi^{-1}(q)$ is facing toward the viewer and $\tilde{n}_z(q) = 0$ if it is facing sideways.

1.6.1 Encoding Features. For a raw feature value $\hat{f}$ (a scalar), we encode it into an internal representation f (again, a scalar), accounting for the following properties.

- Ideally, a real world data $\hat{f}$ and its internal representation f should be tied with an adequate one-to-one correspondence, meaning that those should be regarded as different (resp. the same) in the example styles should also be regarded as different (resp. the same) in the internal representation[2]. This suggests a design of an appropriate *relativization* operator discussed later and that there should usually be only one single relativization operator commonly applied to all data such that transfer (between different objects) makes the intended sense.
- Practically, it is important that the range of features in the exemplar covers (at least a large fraction of) the range of features in the target. It is absolutely fine that the former range is larger than the latter, but if the latter is much larger than the former, there can be a considerable number of queries requiring *extrapolation*; an output with too much extrapolation may not work in the intended way and should be avoided.
- We need a bit care in treating the Gaussian and mean curvatures. First, it is the pair of these curvatures describing the local shape (i.e., their ratio is a useful information); they should follow the same encoding procedure and should not be treated separately. Second, we need to be aware of differences in units. Because the curvature along a curve has a unit of inverse of length (e.g., m^{-1}), Gaussian curvature has a unit like m^{-2}, whereas mean curvature m^{-1}; a change of unit e.g., from meters to centimeters would change the ratios between these quantities.
- To account for the balance between different features, we might want to design an adequate *normalization*.

Our current encoding makes use of the following relativization and normalization stages. The actual design should adapt to each particular use case, and our design should be treated as an example of such a recipe to help the user designing their own customized encoding procedure.

[2]What this means exactly depends on the actual use case.

1.6.2 Relativization. We relativize the raw feature values as follows.

- For the distance from silhouettes d_S, we linearly scale it so that the minimum and maximum values become 0 and 1, respectively. This is done per frame and is for relatively treating the closeness from the boundary and to the innermost region.
- For the Gaussian κ and mean H curvatures, we intend to treat them in a relative sense (like d_S), with the relativeness meaning that the style in a region with a curvature higher than others in a frame is tied with the style in such a region in another frame, and we do not care about comparing their absolute values between frames. We also assume that the artist will not use unperceivable hidden curvature information to determine the style on the visible region. These considerations lead to our following design for the relativization of the curvatures. For each frame, we first perform a (relative) non-dimensionalization. We choose the representative length l^{obj} to be the average of the side lengths of the bounding box of the object, and multiply $(l^{\mathrm{obj}})^2$ (resp. l^{obj}) to κ (resp. H), i.e., the non-dimensionalized quantity becomes $\kappa^{\mathrm{nd}} = \kappa(l^{\mathrm{obj}})^2$ (resp. $H^{\mathrm{nd}} = Hl^{\mathrm{obj}}$). Then, we remove outlier values (due to numerical inaccuracies for the computation of the discrete curvatures). Outlier removal is done for κ and H separately by first enumerating the curvature values in the visible region and detecting the upper and lower 1% quantiles. Then, we obtain the minimum and maximum of the rest (middle) 98% quantile. Any value outside of this middle quantile is clamped to the minimum or maximum value. Let the clamped values be $\kappa^{\mathrm{nd,\,c}}$ and $H^{\mathrm{nd,\,c}}$. Next, for points q in the visible region $\Pi(M)$, we compute their maximum absolute curvature $C^{\max}$ as $C^{\max} = \max_{q\in\Pi(M)} \max(|\kappa^{\mathrm{nd,\,c}}(q)|, |H^{\mathrm{nd,\,c}}(q)|)$. Note that the comparison between the Gaussian and mean curvatures only makes sense when they are non-dimensionalized. Then, we set the relativized Gaussian κ^{rel} and mean H^{rel} curvatures to $\kappa^{\mathrm{rel}} = \frac{\kappa^{\mathrm{nd,\,c}}}{C^{\max}}$ and $H^{\mathrm{rel}} = \frac{H^{\mathrm{nd,\,c}}}{C^{\max}}$. With this relativization, $\kappa^{\mathrm{rel}}, H^{\mathrm{rel}} \in [-1, 1]$.
- For the apparent diffuse intensity gradient $I_d^{\nabla_2}$, we again only care about their relative magnitudes for comparison between frames. For each frame, we compute its mean $\mu_{I_d^{\nabla_2}}$ and standard deviation $\sigma_{I_d^{\nabla_2}}$ within the visible region, then we clamp outliers with values larger than $\mu_{I_d^{\nabla_2}} + 2\sigma_{I_d^{\nabla_2}}$ (because $I_d^{\nabla_2} \geq 0$ by construction, we do not clamp the lower side). Then, we linearly scale and shift the range $[0, \mu_{I_d^{\nabla_2}} + 2\sigma_{I_d^{\nabla_2}}]$ to $[-1, 1]$.
- We leave the diffuse I_d and specular I_s intensities, as well as the normal features $\tilde{n}_x$, $\tilde{n}_y$, and $\tilde{n}_z$ unchanged.

1.6.3 Normalization. After relativization, I_d and I_s are within $[0, L_M]$; $I_d^{\nabla_2}$, κ, H, $\tilde{n}_x$, and $\tilde{n}_y$ are within $[-1, 1]$; and d_S and $\tilde{n}_z$ are within $[0, 1]$. We rescale and shift them such that all of them are within

$[-1, 1]$ (i.e., $[0, L_M]$ for I_d and I_s becomes $[-1, 1]$, and $[0, 1]$ for d_S and $\tilde{n}_z$ becomes $[-1, 1]$).

1.7 Computing Canonical Sections

The 6 canonical sections involved in our method are

- apparent intensity gradient $\boldsymbol{I}_{d\parallel}^{\nabla_2} = \Pi^{-1}(\mathcal{N}(\nabla_2 I_d))$, where $I_d = T(L^*(\boldsymbol{I}_d(q)))$, and $\mathcal{N}(\boldsymbol{x}) := \boldsymbol{x}/\|\boldsymbol{x}\|$ is the normalization operator,
- its 90° rotation $\boldsymbol{I}_{d\perp}^{\nabla_2} = \Pi^{-1}(\mathrm{Rot}_{\pi/2}(\mathcal{N}(\nabla_2 I_d)))$, where the rotation is performed in the image space (also applies for other rotated canonical sections),
- silhouette guided direction $\boldsymbol{o}_{s\parallel} = \Pi^{-1}(\mathcal{N}(\boldsymbol{o}_{2,s\parallel}))$,
- its 90° rotation $\boldsymbol{o}_{s\perp} = \Pi^{-1}(\mathrm{Rot}_{\pi/2}(\mathcal{N}(\boldsymbol{o}_{2,s\parallel})))$,
- apparent gradient of the normal tilt $\boldsymbol{n}_{\parallel} = \Pi^{-1}(\mathcal{N}(\nabla_2 \tilde{n}_z))$, and
- its 90° rotation $\boldsymbol{n}_{\perp} = \Pi^{-1}(\mathrm{Rot}_{\pi/2}(\mathcal{N}(\nabla_2 \tilde{n}_z)))$.

On a silhouette line, $\boldsymbol{o}_{2,s\parallel}(p)$ aligns with the line. The choice of its orientation does not matter as long as it is coherent between frames; the negation of the sign is simply reflected in the negation of the corresponding weight function. We have defined all of these canonical sections $\hat{A}_i$ from 2D orientations (i.e., all of $\hat{A}_i$ have the form $\hat{A}_i = \Pi^{-1}(\hat{A}_i^{\mathrm{sc}})$, where $\hat{A}_i^{\mathrm{sc}}$ is a direction in the 2D screen space, and recall that the rotations are performed in the image space); this is due to our assumption that the orientations are determined on the basis of the perceived 2D information. The push forward operator Π^{-1} is applied to these 2D orientations to lift them to the 3D object space for latter convenience of vector field smoothing.

A tangent vector $\boldsymbol{t}_p^{\mathrm{obj}}$ at $p \in M$ in the object space pushed forward from $\boldsymbol{t}_q^{\mathrm{sc}} \in \mathbb{R}^2$ at q in the screen space lies in the tangent space T_pM of the object surface (i.e., $\boldsymbol{t}_p^{\mathrm{obj}} \in T_pM$), hence can be parameterized by the two scalar coefficients α_p and β_p used for linearly combining the two basis vectors $\boldsymbol{e}_p^{(1)}$ and $\boldsymbol{e}_p^{(2)}$ of the tangent space:

$$\boldsymbol{t}_p^{\mathrm{obj}} = \alpha_p \boldsymbol{e}_p^{(1)} + \beta_p \boldsymbol{e}_p^{(2)}. \tag{10}$$

The projection operator Π at p linearly relates the two tangent spaces T_pM and $\mathbb{R}^2$, hence can be represented using a matrix $\boldsymbol{\Pi}_p$, and we have the relation between $\boldsymbol{t}_p^{\mathrm{obj}}$ and $\boldsymbol{t}_q^{\mathrm{sc}}$ via

$$\boldsymbol{t}_q^{\mathrm{sc}} = \boldsymbol{\Pi}_p \boldsymbol{t}_p^{\mathrm{obj}}. \tag{11}$$

Substituting (10), we have

$$\boldsymbol{t}_q^{\mathrm{sc}} = \boldsymbol{\Pi}_p(\alpha_p \boldsymbol{e}_p^{(1)} + \beta_p \boldsymbol{e}_p^{(2)}) = \begin{pmatrix} \boldsymbol{\Pi}_p \boldsymbol{e}_p^{(1)} & \boldsymbol{\Pi}_p \boldsymbol{e}_p^{(2)} \end{pmatrix} \begin{pmatrix} \alpha_p \\ \beta_p \end{pmatrix}. \tag{12}$$

Letting $\mathbf{P}_p := \begin{pmatrix} \boldsymbol{\Pi}_p \boldsymbol{e}_p^{(1)} & \boldsymbol{\Pi}_p \boldsymbol{e}_p^{(2)} \end{pmatrix}$, we have $\mathbf{P}_p \in \mathbb{R}^{2\times 2}$, and

$$\begin{pmatrix} \alpha_p \\ \beta_p \end{pmatrix} = \mathbf{P}_p^{-1} \boldsymbol{t}_q^{\mathrm{sc}} \tag{13}$$

is well-defined for visible points p. Hence, we arrive at

$$\boldsymbol{t}_p^{\mathrm{obj}} = \begin{pmatrix} \boldsymbol{e}_p^{(1)} & \boldsymbol{e}_p^{(2)} \end{pmatrix} \begin{pmatrix} \alpha_p \\ \beta_p \end{pmatrix} = \begin{pmatrix} \boldsymbol{e}_p^{(1)} & \boldsymbol{e}_p^{(2)} \end{pmatrix} \mathbf{P}_p^{-1} \boldsymbol{t}_q^{\mathrm{sc}}, \tag{14}$$

meaning that the matrix representation $\boldsymbol{\Pi}_q^{-1}$ for the push forward operator Π^{-1} at q is given by

$$\boldsymbol{\Pi}_q^{-1} = \begin{pmatrix} \boldsymbol{e}_p^{(1)} & \boldsymbol{e}_p^{(2)} \end{pmatrix} \mathbf{P}_p^{-1}. \tag{15}$$

1.8 Regression for Orientations

From (15), we can verify that $\Pi \circ \Pi^{-1} = id$, where id is the identity operator, via

$$\boldsymbol{\Pi}_q \boldsymbol{\Pi}_q^{-1} = \boldsymbol{\Pi}_q \begin{pmatrix} \boldsymbol{e}_p^{(1)} & \boldsymbol{e}_p^{(2)} \end{pmatrix} \mathbf{P}_p^{-1} = \mathbf{P}_p \mathbf{P}_p^{-1} = \mathbf{I}. \tag{16}$$

We represent each canonical section $\hat{A}_i(p)$ as a column vector $\hat{\boldsymbol{A}}_{i,p} \in \mathbb{R}^3$ and $\hat{A}(p)$ as a matrix $\hat{\mathbf{A}}_p \in \mathbb{R}^{3\times N_A}$. The output of the weight model $W_{\boldsymbol{u}}(\phi_{L,V}(p))$ can then be represented as a column vector $\boldsymbol{W}_{\boldsymbol{u},p} \in \mathbb{R}^{N_A}$, and $W_{\boldsymbol{u}}(\phi_{L,V}(p)) \cdot \hat{A}(p)$ becomes $\hat{\mathbf{A}}_p \boldsymbol{W}_{\boldsymbol{u},p}$.

Because in our construction, all of the canonical sections $\hat{A}_i(p)$ have the form $\hat{A}_i(p) = \Pi^{-1}(\hat{A}_i^{sc}(q))$, $\hat{\mathbf{A}}_p$ can be expressed as $\hat{\mathbf{A}}_p = \boldsymbol{\Pi}_q^{-1} \hat{\mathbf{A}}_q^{sc}$, where $\hat{\mathbf{A}}_q^{sc} \in \mathbb{R}^{2\times N_A}$ consists of N_A column vectors $\hat{\boldsymbol{A}}_{i,q}^{sc}$ representing $\hat{A}_i^{sc}(q)$. Hence, $\Pi(W_{\boldsymbol{u}}(\phi_{L,V}(\Pi^{-1}(q))) \cdot \hat{A}(\Pi^{-1}(q)))$ in the error functional (4) in our main paper becomes $\boldsymbol{\Pi}(\boldsymbol{\Pi}_q^{-1} \hat{\mathbf{A}}_q^{sc}) \boldsymbol{W}_{\boldsymbol{u},\Pi^{-1}(q)}$, which simplifies to $\hat{\mathbf{A}}_q^{sc} \boldsymbol{W}_{\boldsymbol{u},\Pi^{-1}(q)}$. With this, our error functional becomes

$$\mathcal{E}_{\boldsymbol{u}}(\boldsymbol{W}_{\boldsymbol{u}}) = \int_{\Pi(M)} \left\| \boldsymbol{u}_0(q) - \hat{\mathbf{A}}_q^{sc} \boldsymbol{W}_{\boldsymbol{u},\Pi^{-1}(q)} \right\|^2 dQ. \tag{17}$$

We discretize the integral as a summation over pixels for the visible regions of the surfaces as

$$\mathcal{E}_{\boldsymbol{u}}(\boldsymbol{W}_{\boldsymbol{u}}) \approx \mathcal{E}_{\boldsymbol{u}}^{\mathrm{dis}}(\boldsymbol{W}_{\boldsymbol{u}}) = \sum_{q \in \Pi^{\mathrm{dis}}(M)} \left\| \boldsymbol{u}_0(q) - \hat{\mathbf{A}}_q^{sc} \boldsymbol{W}_{\boldsymbol{u},\Pi^{-1}(q)} \right\|^2 \Delta Q, \tag{18}$$

where $\Pi^{\mathrm{dis}}(M) \subset \Pi(M)$ is the discrete set of points corresponding to the pixel centers, and ΔQ is the area of a pixel (a constant).

If $W_{\boldsymbol{u}}(\phi_{L,V}(p))$ is a linear model, we can further represent $\phi_{L,V}(p)$ as a column vector $\boldsymbol{\phi}_p \in \mathbb{R}^{N_F}$, and $W_{\boldsymbol{u}}(\phi_{L,V}(p))$ as the multiplication of an instance-independent matrix $\mathbf{W}_{\boldsymbol{u}}^{(1)} \in \mathbb{R}^{N_A \times N_F}$ with $\boldsymbol{\phi}_p$ plus a constant vector $\mathbf{W}_{\boldsymbol{u}}^{(0)} \in \mathbb{R}^{N_A}$:

$$\boldsymbol{W}_{\boldsymbol{u},p} = \mathbf{W}_{\boldsymbol{u}}^{(0)} + \mathbf{W}_{\boldsymbol{u}}^{(1)} \boldsymbol{\phi}_p, \tag{19}$$

where we are stretching our use of sans-serif symbol for a vector $\mathbf{W}_{\boldsymbol{u}}^{(0)}$ here for later simplicity of the generalized form (23). So, the regression for the linear model returns the learned weights $\check{\mathbf{W}}_{\boldsymbol{u}}^{(0)}$ and $\check{\mathbf{W}}_{\boldsymbol{u}}^{(1)}$ via

$$\begin{aligned} (\check{\mathbf{W}}_{\boldsymbol{u}}^{(0)}, \check{\mathbf{W}}_{\boldsymbol{u}}^{(1)}) &= \operatorname*{argmin}_{\mathbf{W}_{\boldsymbol{u}}^{(0)}, \mathbf{W}_{\boldsymbol{u}}^{(1)}} \mathcal{E}_{\boldsymbol{u}}^{\mathrm{dis}}(\mathbf{W}_{\boldsymbol{u}}^{(0)}, \mathbf{W}_{\boldsymbol{u}}^{(1)}) \\ &= \operatorname*{argmin}_{\mathbf{W}_{\boldsymbol{u}}^{(0)}, \mathbf{W}_{\boldsymbol{u}}^{(1)}} \sum_{q \in \Pi^{\mathrm{dis}}(M)} \left\| \boldsymbol{u}_0(q) - \hat{\mathbf{A}}_q^{sc}(\mathbf{W}_{\boldsymbol{u}}^{(0)} + \mathbf{W}_{\boldsymbol{u}}^{(1)} \boldsymbol{\phi}_{\Pi^{-1}(q)}) \right\|^2 \Delta Q. \end{aligned} \tag{20}$$

Likewise, a second-order model can be represented by

$$\boldsymbol{W}_{\boldsymbol{u},p} = \mathbf{W}_{\boldsymbol{u}}^{(0)} + \mathbf{W}_{\boldsymbol{u}}^{(1)} \boldsymbol{\phi}_p + \frac{1}{2} \mathbf{W}_{\boldsymbol{u}}^{(2)} : (\boldsymbol{\phi}_p \boldsymbol{\phi}_p^{\top}), \tag{21}$$

where $\mathbf{W}_{\boldsymbol{u}}^{(2)} \in \mathbb{R}^{N_A \times N_F \times N_F}$ is the second-order coefficients (tensor), and the operator ':' represents a contraction. The coefficients $\mathbf{W}_{\boldsymbol{u}}^{(0)}$,

$\mathbf{W}_{\boldsymbol{u}}^{(1)}$ and $\mathbf{W}_{\boldsymbol{u}}^{(2)}$ are all instance-independent. The regression now becomes

$$(\check{\mathbf{W}}_{\boldsymbol{u}}^{(0)}, \check{\mathbf{W}}_{\boldsymbol{u}}^{(1)}, \check{\mathbf{W}}_{\boldsymbol{u}}^{(2)}) = \underset{\mathbf{W}_{\boldsymbol{u}}^{(0)}, \mathbf{W}_{\boldsymbol{u}}^{(1)}, \mathbf{W}_{\boldsymbol{u}}^{(2)}}{\operatorname{argmin}} \mathcal{E}_{\boldsymbol{u}}^{\mathrm{dis}}(\mathbf{W}_{\boldsymbol{u}}^{(0)}, \mathbf{W}_{\boldsymbol{u}}^{(1)}, \mathbf{W}_{\boldsymbol{u}}^{(2)})$$
$$= \underset{\mathbf{W}_{\boldsymbol{u}}^{(0)}, \mathbf{W}_{\boldsymbol{u}}^{(1)}, \mathbf{W}_{\boldsymbol{u}}^{(2)}}{\operatorname{argmin}} \sum_{q \in \Pi^{\mathrm{dis}}(M)} \left\| \boldsymbol{u}_0(q) - \hat{\mathbf{A}}_q^{sc} \left(\mathbf{w}_{\boldsymbol{u}}^{(0)} + \mathbf{w}_{\boldsymbol{u}}^{(1)} \boldsymbol{\phi}_{\Pi^{-1}(q)} + \frac{1}{2} \mathbf{w}_{\boldsymbol{u}}^{(2)} : (\boldsymbol{\phi}_{\Pi^{-1}(q)} \boldsymbol{\phi}_{\Pi^{-1}(q)}^{\top}) \right) \right\|^2 \Delta Q. \tag{22}$$

In general, an n-th order model can be represented by

$$\boldsymbol{W}_{\boldsymbol{u},p} = \sum_{k=0}^{n} \frac{1}{k!} \mathbf{W}_{\boldsymbol{u}}^{(k)} : (\underbrace{\boldsymbol{\phi}_p \otimes \cdots \otimes \boldsymbol{\phi}_p}_{k \text{ times}}), \tag{23}$$

where $\mathbf{W}_{\boldsymbol{u}}^{(k)} \in \mathbb{R}^{N_A \times \overbrace{N_F \times \cdots \times N_F}^{k \text{ times}}}$, and 0 times of tensor products between $\boldsymbol{\phi}_p$ is interpreted as a scalar of 1.

In essence, the regression (for a model $W_{\boldsymbol{u}}(\phi_{L,V}(\Pi^{-1}(q))$ of any order) will become a standard least square problem:

$$(\check{\mathbf{W}}_{\boldsymbol{u}}^{(0)}, \ldots, \check{\mathbf{W}}_{\boldsymbol{u}}^{(n)}) = \underset{\mathbf{W}_{\boldsymbol{u}}^{(0)}, \ldots, \mathbf{W}_{\boldsymbol{u}}^{(n)}}{\operatorname{argmin}} \mathcal{E}_{\boldsymbol{u}}^{\mathrm{dis}}(\mathbf{W}_{\boldsymbol{u}}^{(0)}, \ldots, \mathbf{W}_{\boldsymbol{u}}^{(n)})$$
$$= \underset{\mathbf{W}_{\boldsymbol{u}}^{(0)}, \ldots, \mathbf{W}_{\boldsymbol{u}}^{(n)}}{\operatorname{argmin}} \sum_{q \in \Pi^{\mathrm{dis}}(M)} \left\| \boldsymbol{u}_0(q) - \hat{\mathbf{A}}_q^{sc} \sum_{k=0}^{n} \frac{1}{k!} \mathbf{W}_{\boldsymbol{u}}^{(k)} : (\underbrace{\boldsymbol{\phi}_{\Pi^{-1}(q)} \otimes \cdots \otimes \boldsymbol{\phi}_{\Pi^{-1}(q)}}_{k \text{ times}}) \right\|^2 \Delta Q. \tag{24}$$

1.9 Generating Orientation Field

Once we have learned the model, the initial orientation $\boldsymbol{u}(t,p)$ at a point p on the object surface at time t can be estimated as

$$\boldsymbol{u}(t,p) = \hat{\mathbf{A}}_p \boldsymbol{W}_{\boldsymbol{u},p}. \tag{25}$$

For an n-th order model, this is

$$\boldsymbol{u}(t,p) = \hat{\mathbf{A}}_p \sum_{k=0}^{n} \frac{1}{k!} \mathbf{W}_{\boldsymbol{u}}^{(k)} : (\underbrace{\boldsymbol{\phi}_p \otimes \cdots \otimes \boldsymbol{\phi}_p}_{k \text{ times}}). \tag{26}$$

We evaluate $\boldsymbol{u}(t,p)$ at each vertex of the surface mesh of the object, which requires the evaluation of $\hat{\mathbf{A}}_p$ and $\boldsymbol{\phi}_p$ at the vertex, for also occluded ones and back-facing ones. Ideally, all of these quantities can still be computed by using a vertex shader (for d_S or $\boldsymbol{o}_{2,s}$, we can use the data at the visible point p^{vis} on the line of sight between the viewpoint and p). In our prototype implementation, we just use the data at p^{vis} for all of them instead, admitting that this work around (though simple) would introduce unwanted alternation to the resulting orientation field (though such alternation seems to be mitigated by our vector field smoothing).

Once we have the initial orientations at vertices, we convert them to the discrete 1-forms stored at edges following Fisher et al. [2007]. Suppose that an edge e_{ij} has its start and end points p_i and p_j, with the corresponding positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, respectively, we assign to it the following discrete 1-form c_{ij} (a scalar):

$$c_{ij} = \frac{\boldsymbol{u}(t,p_i) + \boldsymbol{u}(t,p_j)}{2} \cdot (\boldsymbol{p}_j - \boldsymbol{p}_i). \tag{27}$$

1.10 Smoothing Orientation Field

Our prototype implementation solves the optimization (7) in our main paper sequentially. First spatially:

$$\check{\boldsymbol{c}}_s^k = \underset{\tilde{\boldsymbol{c}}_s^k}{\operatorname{argmin}} \left(\star_1 \left\| \tilde{\boldsymbol{c}}_s^k - \boldsymbol{c}^k \right\|^2 + \lambda_s (\tilde{\boldsymbol{c}}_s^k)^{\top} \boldsymbol{L}_1 \tilde{\boldsymbol{c}}_s^k \right), \tag{28}$$

and then temporally:

$$\check{\boldsymbol{c}}^k = \underset{\tilde{\boldsymbol{c}}_t^k}{\operatorname{argmin}} \left(\star_1 \left\| \tilde{\boldsymbol{c}}_t^k - \check{\boldsymbol{c}}_s^k \right\|^2 + \lambda_t \frac{\star_1}{\Delta t} \left\| \tilde{\boldsymbol{c}}_t^k - \tilde{\boldsymbol{c}}_t^{k-1} \right\|^2 \right), \tag{29}$$

where we set $\check{\boldsymbol{c}}_s^0 = \check{\boldsymbol{c}}_s^1$. For the construction of the operators $\star_1$ and $\boldsymbol{L}_1$ of discrete exterior calculus, we use the PyDec library [Bell and Hirani 2008, 2012]. Note that with this sequential approach, the spatial smoothing (28) can be solved per frame independently, and the temporal smoothing (29) can be solved per frame sequentially (only using the data from the previous frame $\tilde{\boldsymbol{c}}_t^{k-1}$ and the spatially smoothed current frame $\check{\boldsymbol{c}}_s^k$). This approach enables incremental processing of the animation data, and the result of earlier frames are not affected when additional frames are later added to the sequence.

1.11 Anchor Points

We predetermine our anchor points (initial points for strokes) such that

- the anchor points are more or less evenly spaced, and
- the anchor points form a hierarchy.

We build such a hierarchy of anchor points via *incremental* Poisson disk sampling. We first use point cloud utils library [Williams 2018], which implements a Poisson disk sampling algorithm by Bowers et al. [2010], to generate points at level 0 on the object surface with the initial radius r_0, set to 5% of the diagonal length of the bounding box of the object. When proceeding to the next level i, we first reduce the radius by a factor of 0.5, i.e., $r_i = 0.5 r_{i-1}$. To generate the points in the i-th level, ideally, we should first register all the points in the levels $0, 1, \ldots, i-1$ with the reduced radius r_i, and continue the Poisson disc sampling. In our prototype implementation, this step is instead approximated by first generating a set of points *from scratch* using the reduced radius r_i, then for each point in one of the previous levels $0, 1, \ldots, i-1$, we identify the closest point in the newly sampled point set and delete that point. We treat the rest of points as those in the level i. This process is repeated to create a hierarchy of levels 0, 1, 2, and 3.

1.12 Generating Strokes

We convert the discrete 1-form smoothed using (29) back to a vector field $\check{\boldsymbol{u}}$ using the Whitney bases according to (4) in Fisher et al. [2007]. Note that we do not care the length of the vectors queried from $\check{\boldsymbol{u}}$ in what follows.

We generate strokes by first computing integral curves of the vector field (also using the length queried from our model) to decide their center lines. Then, we generate polygons from the centerlines and the queried width. Finally, these polygons are drawn with the queried colors and the specified texture onto the screen.

To generate a center line, we start from an anchor point $p^{(0)}$. We use the superscript, e.g., (0), here to identify the vertex index of the discrete center line. That is, $p^{(0)}, p^{(1)}, \ldots$ forms the discrete

line strip of the center line, and $p = p^{(0)}$ is the anchor point or the start point of the line strip. For information that only depends on the start point (i.e., width, length, and color), we will simply use p. We perform nearest neighbor query to our model to obtain the width w_p, length l_p, and color C_p at $p = p^{(0)}$. Recall that w_p and l_p are measured in the screen space. If necessary, the user may multiply globally constant scalars to w_p and l_p for adjustment. For the orientation, we first randomly sample an angle offset ξ_p and query the unaltered initial tangent vector $\boldsymbol{u}_{p^{(0)}}$ from the smoothed vector field $\breve{\boldsymbol{u}}$. Then, we rotate $\boldsymbol{u}_{p^{(0)}}$ by ξ_p within the facet (i.e., a rotation along the normal of the current facet) to obtain the altered tangent $\hat{\boldsymbol{u}}_{p^{(0)}}$. We traverse along the facets of the surface mesh to construct the center line. We use a scalar l_p^{acc}, initialized to 0, to accumulate the projected length of the discrete line strip on the screen during the traversal.

Starting from $p^{(i)}$ ($i = 0$ initially), we move along $\hat{\boldsymbol{u}}_{p^{(i)}}$ until we first reach a point $p^{(i+1)}$ on an edge between facets. We compute the projected length $\Delta l^{i,i+1} = \|\Pi(p^{(i+1)}) - \Pi(p^{(i)})\|$ on the screen space and accumulate $\Delta l^{i,i+1}$ to l_p^{acc}. If $l_p^{acc} \geq l_p$, we terminate the traversal. Otherwise, we continue the traversal by moving onto the adjacent facet, querying the unaltered tangent $\boldsymbol{u}_{p^{(i+1)}}$ from $\breve{\boldsymbol{u}}$ at $p^{(i+1)}$ on the new facet, and performing a rotation along the normal of the new facet by the angle ξ_p to get the altered tangent $\hat{\boldsymbol{u}}_{p^{(i+1)}}$. We repeat this process until $l_p^{acc} \geq l_p$ is satisfied.

Note that there are cases where the above process does not terminate (about 9.8%). There are three possible cases (the first two are rare and the last one is most probable). The first is that the sampled tangent vector $\boldsymbol{u}_{p^{(i)}}$ is zero. Theoretically, many closed surfaces possess such singular points somewhere on the surface (the number depends on their topology). But in reality, the chance to hit such a singular point is rare. We have a fail safe for such a case to simply terminate the traversal there. The second and third cases are due to the fact that the vector field $\breve{\boldsymbol{u}}$ converted from the one form (using the Whitney bases) is not necessarily continuous across facets. In the second case, we may end up at repeatedly traversing among several facets at a vicinity of a mesh vertex and see almost no progress in terms of the accumulated projected length of the discrete line strip. For this case, we terminate the traversal if $\Delta l^{i,i+1}$ is smaller than a threshold (set to 10^{-5}) for a successive number of times (set to 10). In the third case, we may be forced to stay back in the same facet when trying to cross an edge. For such a case, we simply terminate at the point on the edge.

Once we have the discrete line strip, we turn it into a polygon strip. At each vertex $p^{(i)}$ of the strip, we compute the local frame $\mathbf{F}^{(i)}$ formed by the tangent $\boldsymbol{F}_t^{(i)}$, normal $\boldsymbol{F}_n^{(i)}$, and bi-normal $\boldsymbol{F}_b^{(i)}$:

$$\mathbf{F}^{(i)} = \begin{pmatrix} \boldsymbol{F}_t^{(i)} & \boldsymbol{F}_n^{(i)} & \boldsymbol{F}_b^{(i)} \end{pmatrix}, \tag{30}$$

then extend the strip along the bi-normal as follows. If $p^{(i)}$ is the start or end point, the normal $\boldsymbol{F}_n^{(i)}$ and tangent $\boldsymbol{F}_t^{(i)}$ are simply taken to be the facet normal at $p^{(i)}$ and $\hat{\boldsymbol{u}}_{p^{(i)}}$, respectively. For each remaining point $p^{(i)}$ (on an edge), we use libigl [Jacobson et al. 2018] to compute the edge normal for $\boldsymbol{F}_n^{(i)}$. For the tangent vectors, we take the tangent vectors $\hat{\boldsymbol{u}}_{p^{(i-1)}}$ and $\hat{\boldsymbol{u}}_{p^{(i)}}$ of the adjacent line segments, and set $\boldsymbol{F}_t^{(i)}$ to be their normalized average: $\boldsymbol{F}_t^{(i)} = \mathcal{N}(\mathcal{N}(\hat{\boldsymbol{u}}_{p^{(i-1)}}) + \mathcal{N}(\hat{\boldsymbol{u}}_{p^{(i)}}))$, where $\mathcal{N}$ is the normalization operator: $\mathcal{N}(\boldsymbol{x}) \coloneqq \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$. Finally, we compute $\boldsymbol{F}_b^{(i)} = \boldsymbol{F}_t^{(i)} \times \boldsymbol{F}_n^{(i)}$.

To extend the line strip using the local frames, at each $p^{(i)}$ we generate two shifted points $p_-^{(i)}$ and $p_+^{(i)}$ along the bi-normal as: $p_-^{(i)} = p^{(i)} - \eta \boldsymbol{F}_b^{(i)}$ and $p_+^{(i)} = p^{(i)} + \eta \boldsymbol{F}_b^{(i)}$, where the unknown scalar η is determined such that the projected width $w^{(i)} = \|\Pi(p_+^{(i)}) - \Pi(p_-^{(i)})\|$ equals w_p.

Prior to the drawing of the stokes onto the screen, we sort their drawing order following the procedure discussed in our main paper. Here, we describe the details regarding the sorting operation. We convert the color C_p to the $L^*a^*b^*$ color space and use the L^* value as the key for sorting. We use the merge sort algorithm, for preserving the orders of strokes with the same L^* key from the previous frame. Inconsistent sorting algorithms, such as quick sort, would break the orders, and would thus degrade the coherency.

When rendering the strokes, we perform the depth test only against the underlying object (no depth tests between strokes). We first draw the underlying object with a tiny negative depth offset. Then, we set the depth buffer to read only and draw the strokes according to the sorted order, accounting for the their colors and the brush texture.

2 ADDITIONAL ANALYSES

2.1 Additional Regression Results

We have tested the first, second, and third order models in our regression for 6 different examples (see Fig. 1). The first order model tends to well fit the exemplar orientations for simpler geometries as in sphere and blobby examples shown in our main paper. Depending on the complexity of the exemplars and underlying geometries, the reconstruction error may increase due to the conflicts of the different orientations on similar features. Even though second and third order models would allow better alignment with the exemplar orientations, we chose the first order model since it is more stable and require less computation time and storage, thanks to its much smaller number of coefficients.

2.2 Different Factors

In terms of the orientations, we examined the dominance of the weights with respect to the three (lighting, silhouette, and normal) categories of the canonical sections, with dominance defined by the ratio between the sum of the weights (in absolute values) in the chosen category and that of all weights. As shown in Fig. 2, the blobby example (Fig. 5 of our main paper) has lighting-dominant orientations (corresponding weights dominate up to 58%) near the center, silhouette-dominant orientations (corresponding weights dominate up to 62%) at outer regions, and normal-dominant orientations (corresponding weights dominate up to 77%) at high curvature regions. In terms of the widths, for the monkey example (5:52 in our video), the strokes are thin at its nose but thick between its nose and mouth. This is mainly affected by the mean curvature. These tendencies agree with the artist's intention expressed in the exemplars.

Figure 1: Additional evaluations of our regression. We compare exemplar orientations (blue arrows) with reconstructed orientations (red arrows) for the first, second, and third order models on double-vortex sphere, spot, monkey, head, fertility, and gargoyle examples.

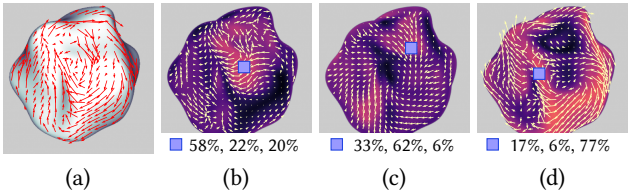


Figure 2: Dominance of weights for the three categories of canonical sections. For a frame of the blobby example, we show the orientation field on the surface in (a), and its components from lighting-based, silhouette-based, and normal-based canonical sections in yellow arrows in (b), (c), and (d), respectively. We used the same color map to encode the dominance (the ratio between the sum of the absolute magnitudes of the corresponding weights to that of all weights) in (b-d), where pure black and pure white indicate 0 and 100%, respectively. The blue markers in (b-d) indicate the locations where we observe the maximum dominance in the category, and the tuples of the percentages from left to right show the percentages of the lighting-based, silhouette-based, and normal-based dominance at the marker points.

2.3 Performance

The amortized cost per frame is approximately 1 minute for the stroke field generation (including spatial and temporal smoothing), and 7 minutes for synthesizing the strokes on a typical desktop PC. Our regression scales with the image resolution, stroke field synthesis scales with the mesh resolution and the number of frames, and synthesis of individual strokes scales with the numbers of anchor points and frames. They respectively take 4.4%-20%, 0.01%-0.2%, and 80.6%-94.5% of the overall computation time for animations of 150 frames.

2.4 Stroke Patterns

In Fig. 3, we summarized the prominent stroke patterns observed in our video. The gargoyle example (8:42 of our video) has strokes forming like a spiral towards the center of its right wing. The average ratio of length-to-width in the gargoyle example is 30. Strokes may make a sharp turn when there is a rapid variation in the vector field, as seen at the bottom part of the parent in the bottom layer of 'fertility' object (6:53 of our video). The superimposed result (7:38 of our video) gives more varieties of stroke patterns.

2.5 Nearest-neighbor Queries

We do not have a threshold for closeness. For Fig. 5 of our main paper, we see an average of 98% overlap between the ranges of the queried features and those of the exemplar ones for each dimension in the proxy space with the smallest (85%) overlap seen for the diffuse intensity as shown in Fig. 4. In the full proxy space, the diameter of the bounding sphere of the 90% percentile of the exemplar data is 4.16, while the average and standard deviation of the Euclidean distances between the query points and their nearest-neighbor exemplar points are 0.55 and 0.25, respectively. Guiding the gathering of the exemplars for a bounded closeness would be an interesting future work.

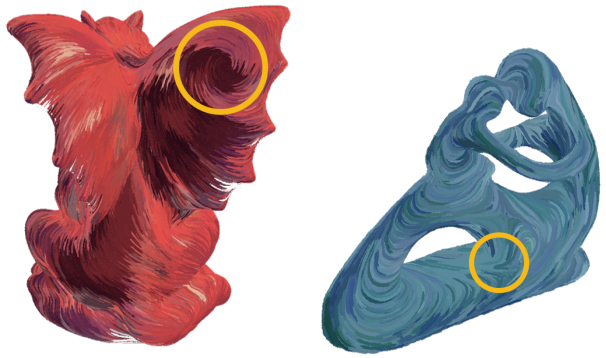
2.6 Higher-order Representations

A higher-order model is less stable, as illustrated in Fig. 5. For blobby (Fig. 5 of our main paper), the first- and second-order representations respectively resulted in absolute values of the weights with averages around 0.8 and 3.9 consistently over frames, and standard deviations around 0.15 and 1.9. The maximum values varied consistently from 1.3 to 1.5 over frames for the first-order representation, but largely (from 5.3 to 35) for the second-order one with larger values frequently encountered when synthesizing the back side.

3 MORE DETAILS FOR FUTURE WORK

We have designed the encoding of the features such that the transferred stroke styles would look reasonable even when we use a single exemplar. This is done by relativizing and normalizing the features in a common feature space across settings. When using a different encoding, it is important that the exemplars contain enough samples such that there will not be too much extrapolation during the transfer. It would be interesting to have an automatic approach that can tell if an enough variation of samples are collected during the exemplar preparation and annotation process.

It would be interesting to also learn the *correlation* among neighboring strokes for the colors, lengths, widths, and orientations. Currently, (though nice for a painterly rendering) neighboring strokes



(a) (b)

Figure 3: Stroke patterns observed in our results. (a) Spiral and long thin strokes in the gargoyle example. (b) Sharp turn in fertility example.

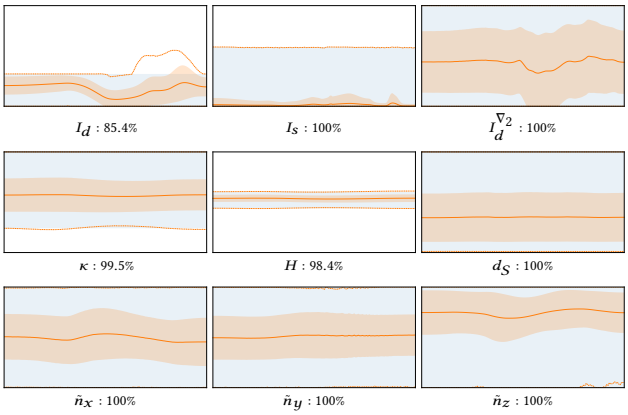
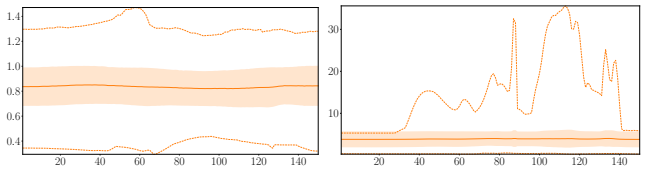


Figure 4: Data alignment between queried features and exemplar features. We have tested all 9 feature spaces in the plots; their horizontal axes show the frame count (left: 1, right: 150), and their vertical axes show the (relativized and normalized) feature value (bottom: −1, top: 1). The painted regions in light blue indicate the coverages of the exemplar. There are three orange lines in each plot: the center one shows the average value μ_{query} of the queried features, and the top and bottom orange lines show the maximum and minimum of the queried features, respectively. The numbers next to the feature labels show the percentage of the queried data (i.e., between the top and bottom orange lines) that reside in the range of the exemplar (i.e., the light blue region). The painted regions in light orange indicate the range of $\mu_{\text{query}} \pm \sigma_{\text{query}}$, where σ_{query} is the standard deviation of the queried features.

in our generated results may have a large variation in colors even when neighboring strokes in the exemplar have coherent colors. We could configure the sorting procedure or the color model to produce coherent colors, but we do not have a continuous control over this



(a) 1st-order (b) 2nd-order

Figure 5: Stability of regression model. The horizontal axes show the frame count, and the vertical axes show the weights (in absolute values) returned from the models. The three orange lines in each plot from top to bottom show the maximum, average ($\mu_{|w|}$), and minimum of the absolute returned weights, while the painted orange regions indicate the range of $\mu_{|w|} \pm \sigma_{|w|}$, where $\sigma_{|w|}$ is the standard deviation of the absolute returned weights. (a) First-order model exhibits stable weight variation over frames, while (b) second-order model has large weight values for the queries near the backside (1-st and 76-th frames correspond to the completely front and back sides).

correlation. We believe that a stochastic model for a generalized randomness would be an important future work.

While we have focused on the automatic generation of animatable strokes from exemplars, it would be also interesting to offer interactive artistic control during the generation of stroke field as well as the integral curves. Allowing for local edits or online learning from new exemplars would offer more flexibility in the generated styles. We have not done any optimization of our code written in python, and acceleration would be crucial for interactive control.

Although the way of overdrawing strokes is acceptable for opaque media, application to transparent media, such as water color, would require careful consideration for removing already generated strokes as well as their mixing. Extending our work for atmospheric participating media would be another interesting extension.

The generation of a temporally stable result relies on a spatially and temporally coherent orientation field. Our current way for the construction of the orientation field does not perform topological optimizations for the singular points of the orientations. This would be important if we want to incorporate additional canonical sections varying rapidly in time, such as the suggestive contours [DeCarlo et al. 2003]. Also, extending our work using tensor fields rather than vector fields would be attractive. We envision that bridging more advanced geometry processing tools with stroke generation would be an interesting future avenue to explore.

REFERENCES

Nathan Bell and Anil N. Hirani. 2008. PyDEC: A Python Library for Discrete Exterior Calculus. https://github.com/hirani/pydec Software made available on Google Code website..

Nathan Bell and Anil N. Hirani. 2012. PyDEC: Software and Algorithms for Discretization of Exterior Calculus. *ACM Trans. Math. Software* 39, 1, Article 3 (nov 2012), 41 pages. https://doi.org/10.1145/2382585.2382588

John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. 2010. Parallel Poisson Disk Sampling with Spectrum Analysis on Surfaces. *ACM Transactions on Graphics* 29, 6 (Proc. of SIGGRAPH ASIA 2010), Article 166 (dec 2010), 10 pages. https://doi.org/10.1145/1882261.1866188

Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. 2003. Suggestive Contours for Conveying Shape. *ACM Transactions on Graphics* 22, 3 (Proc. of SIGGRAPH 2003) (jul 2003), 848––855. https://doi.org/10.1145/882262.882354

Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Hugues Hoppe. 2007. Design of Tangent Vector Fields. *ACM Transactions on Graphics* 26, 3 (Proc. of SIGGRAPH 2007) (jul 2007), 56–65. https://doi.org/10.1145/1276377.1276447

Yoshiharu Gotanda. 2010. Practical implementation at tri-Ace. In *SIGGRAPH 2010 Course: Physically Based Shading Models in Film and Game Production*.

Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

OpenCV. 2015. Open Source Computer Vision Library.

Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. 2002. Photographic Tone Reproduction for Digital Images. *ACM Transactions on Graphics* 21, 3 (Proc. of SIGGRAPH 2002) (jul 2002), 267––276. https://doi.org/10.1145/566654.566575

Francis Williams. 2018. Point Cloud Utils: A Python library for common tasks on 3D point clouds and meshes. https://github.com/fwilliams/point-cloud-utils