TextR3Mesh: Semantically-Guided Neural Remeshing

Anonymous CVPR submission

Paper ID *****

Figure 1. Best Results showing task/feats

Abstract

We present TextR3Mesh, a method for remeshing an input mesh guided by intuitive text prompts. We also introduce vertex density controls and novel remeshing operations that adaptively add and remove vertices to improve stylization resolution. The learning is guided by CLIP similarity between input text prompts or images and 3D renders of the mesh. This method does not require any pre-training or datasets and we demonstrate its ability to handle a wide array of input meshes, anisotropic texture prompts, and abstract mesh-prompt combinations.

1. Introduction

The process of producing or editing visual data is one of iteration, beginning with some prior and continually adjusting until an agreed-upon result is achieved. Over time, the resulting product should be able to diverge from the prior, but shifts in parameterization and form must remain controlled so as to not corrupt the result. This tension between freedom and control in automated modification of 3D data has been prevalent in the computer vision and graphics spaces. To enable substantial editing of an input mesh, we give our system the freedom to predict vertex motion in any direction, recolor vertices, and change the mesh's geometric parametrization via edge split, collapse, and flip operations. We control this process to ensure quality results by regularizing vertex displacement predictions, slowing integration of color, and creating strict geometrically-informed criteria for remeshing operations.

When describing the input and output meshes, we consider mesh *geometry* to be the topology and uncolored form of the uncolored mesh. Mesh *texture* refers to the depth and feature detail added by surface coloring. Mesh *resolution* is the level of fine-grain geometric detail achievable with the current set of vertices, which is typically proportional to the number of vertices and how uniformly they are distributed across the mesh surface. Intuitively, a mesh with tens of thousands of vertices well distributed across its surface can achieve more fine-grain geometric and texture detail than an 8 vertex cube. *Style* refers to the visual affect of the combination of mesh geometry and color. The resolution of style is analogously how small or fine-grain the details on the mesh are. A mono-color smooth cylindrical vase has low style resolution while a rough tactile carpet with tiny flower patterning has high style resolution.

Many works aim to integrate user-friendly interfaces for controlling said modification process to speed up workflows and lower barriers to entry. We achieve intuitive an user interface by leveraging CLIP, a joint image-text semantic embedding space. By comparing the embeddings of text to 2D projections of the 3D mesh, we can allow users to provide natural language prompts which provide strong signals that guide optimization. This more abstract representation of style does not pin the system to a set output. What's more, it allows users to play with word choice to find prompts that best evoke the desired style. Because CLIP allows for embedding of images, users can also use image prompts to guide optimization. The ability to use text and detailed imagery gives many intuitive entry points for mesh styling with TextR3Mesh.

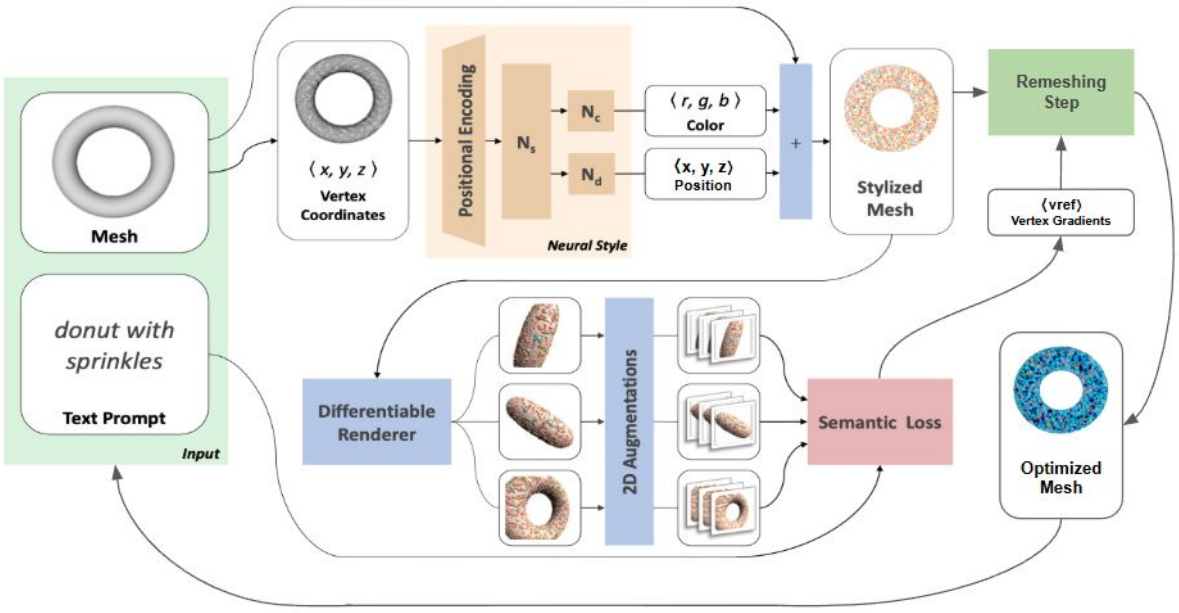

Figure 2. High level image of our method starting from the input mesh

The editing of the mesh is handled by a neural networked dubbed the *neural style field* (NFF), which is optimized to predict vertex displacements and and color shifts that best correspond to the input text prompt. The NSF carries with it a natural regularization of our geometry and color, as the network's spectral smoothing leads to locally consistent displacements and coloring which get more fine-grain over optimization. The displacement outputs are regularized such that they are projected to the vertex normals at the beginning of optimization, and slowly allowed to be more directly applied to vertices. This again slows the introduction of complexity so the system can learn how to leverage manipulation freedom to achieve the best results.

Between applications of NSF predicted vertex displacements and color shifts we perform a set of remeshing operations to adjust the mesh resolution. The primary operation is edge split, which adds new vertices in regions of the mesh that are developing more complex geometry, the assumption being that these regions will need the extra resolution to continue to optimize. This functionality works in concert with a mesh density tracker, which adjusts vertex displacement outputs from the NSF to pull vertices along the mesh surface to regions with high geometric complexity. The split operation takes over where this adaptive mesh density fails, a.k.a. when the number of vertices cannot be adequately redistributed to meet the needs of style resolution. Lastly, we layer in an edge flip operation to ensure rational vertex connectivity as mesh geometry and resolution change and an edge collapse on regions with very low geometric complexity.

In summary, our method takes advantage of CLIP to grant users intuitive input options for semantic guidance of mesh stylization. We slowly give the system more freedom of modification by using the regularized outputs of a neural network to adjust vertex positions and colors. Our pipeline tracks vertex density to adjust local mesh resolution as needed for stylization. This is done in concert with edge split, flip, and collapse operations that take over to introduce and adjust mesh resolution when needed over optimization so the output mesh can achieve the best style quality.

2. Related Work

2.1. Mesh Stylization

Stylization of 3D meshes is a multifaceted problem that relies on updates to their geometry and texture. For the objective of mesh texturing, mesh parameterization is a core approach in computer graphics for mapping textures onto triangular surfaces [1, 2, 3]. This problem requires different parameterization approaches depending on the mesh quality and the intended outcomes for the process. As a result, many parameterization techniques place certain restrictions on the qualities of input meshes that cannot be guaranteed for "in the wild" meshes [4]. Text2Mesh sidestepped the parameterization problem by using a neural style field to predict per vertex color values and displacements for an explicit mesh representation, offering compatibility for a larger pool of meshes [5, 6]. Still, Text2Mesh requires input meshes to have high resolutions and well-distributed vertices to produce meaningful results since it cannot create vertices adaptively.

Updating the style of 3D shapes involves geometric style transfers from a source model to a target one without compromising the structure of the target model [7]. Approaches have included part-based shape analysis, category-specific style transfer, and image-driven mesh stylization [8, 9, 10]. Text2Mesh aimed for general geometric style transfers through the use of guidance from flexible text prompts enabled by the representational power of CLIP [6]. However, Text2Mesh lacks the ability to interpret and visualize certain text prompts accurately.

Our paper builds upon Text2Mesh's framework, adopting the use of its neural style field and text-driven approach for mesh stylization. With these features, TextR3Mesh bypasses the parameterization problem and achieves general geometric style transfers. With improved text-driven stylization and remeshing procedures, TextR3Mesh surpasses its predecessor Text2Mesh in its ability to style meshes for text prompts and meshes of varying complexity and quality.

2.2. Text Driven Manipulation

TextR3Mesh uses text controls for driving mesh deformations. CLIP coordinates visual and textual representations into a shared embedding space [11]. Other works in the field have utilized CLIP for text-driven image editing and synthesis [12, 13, 14]. Text2Mesh further leveraged CLIP's representational abilities through controls for text-driven 3D mesh stylizations and mesh rendering [6].

Our work extends upon Text2Mesh through the integration of state-of-the-art diffusion models. Diffusion models have become popular generative models that leverage Markov chains to convert samples from a noise distribution to samples from the data distribution. This process is undertaken in the forward process (noising process) and the reverse process (generative process) [15]. Recent works have harnessed diffusion guidance, steering the generative process based on certain conditions [16, 17, 18]. Specifically, DeepFloyd IF is a text-to-image pixel diffusion model that has been used to great success for purposes of text-driven image generation [19]. By using DeepFloyd IF guidance, TextR3Mesh improves over Text2Mesh through better text-driven guidance. Specifically, DeepFloyd IF possesses a deeper understanding of the image and text prompt space than CLIP through its large language model text encoder T5-XXL and increased number of effective parameters.

2.3. Remeshing and Inverse Rendering

Remeshing processes change the mesh topology of a 3D model to facilitate mesh repair, adaptive vertex densities, and mesh optimizations [20, 21]. In Continuous Remeshing for Inverse Rendering, Palfinger offers a refined remeshing algorithm with steps for edge collapse, edge split, edge flip, smoothing, and relocating [22]. Further works seek to adjust mesh topology during optimization so that geometrically complex regions of meshes gain resolution [23]. At each iteration of TextR3Mesh's optimization pipeline, remeshing steps inspired from these works are used to heal mesh defects and adaptively change the number vertices in certain regions for better stylization.

In Text2Mesh, 2D images of the stylized input mesh are rendered from various views. Using CLIP, the similarity between these 2D images and the text prompt guide the mesh stylization process [6]. However, Text2Mesh's guidance heavily relies on the quality and resolution of the initial source mesh to produce results since the vertex count is constant and vertices are only displaced along the normal direction. While TextR3Mesh also uses inverse rendering for mesh stylization, our system leverages remeshing operations to iteratively improve the mesh topology for optimization. This improvement offers more flexibility in the topology of the initial source mesh and ultimately upgrades the final mesh resolution.

3. Method

An illustration of our method can be seen in Fig. 2. The input object being modified is a triangular mesh M parameterized by vertices $V \in \mathbb{R}^{n\times 3}$ and Faces $F \in \{0, ..., n-1\}^{m\times 3}$. As we remesh over optimization we denote each as V_i and F_i to indicate vertices and faces of mesh M_i at optimization step $i \in \{1, ..., \text{max iterations}\}$. The NSF maps vertex positions to the RGB color shift and (x, y, z) displacement that results in the stylized mesh M^S with the best semantic correspondence with the input text prompt t. We render multiple views of M^S, apply 2D augmentations, and then compute the similarity between their average CLIP embedding and that of the text prompt. We then calculate the average distance between each vertex and its one-ring neighbors. Vertices in regions of high curvature are penalized for this distance between each other and areas of low curvature are encouraged to have higher distances. This adaptive mesh density energy and the semantic similarity between the 2D renders and text prompt provide the signal to update the NSF weights. Finally, we calculate the current edge lengths of M_i^S and compare them to the target lengths of M_{i-1}^S. This information is fed into the remeshing criteria to inform the remeshing step, adjusting V_{i+1} and F_{i+1}.

3.1. Neural Style Field Network

As discussed above, the NSF network maps vertex points of M_i to style attributes $c_i, d_i \in \mathbb{R}^{n\times 3}$, where each row of matrix c_i corresponds to a per-vertex RGB shift and each row of d_i corresponds to a per-vertex (x, y, z) displacement vector in $\mathbb{R}^3$. The low-dimensional input position vectors raises concerns over spectral smoothing, which could prevent the network from earning high-enough frequency rules to achieve the desired style fidelity. To address this, we pass

as input a point p's positional encoding $\gamma(p)$ pulled from Fourier feature mapping:

$$\gamma(p) = [cos(2\pi \mathbf{B}p), sin(2\pi \mathbf{B}p)]^T \quad (1)$$

where each element of $\mathbf{B} \in \mathbb{R}^3$ is randomly drawn from $\mathcal{N}(0, \sigma^2)$, making $\mathbf{B}$ a random Guassian matrix. σ is a hyperparameter which allows users to control the frequency of the NSF mapping function.

In implementation, points $p \in V$ are normalized to lie within a unit bounding box so we can use constant render parameters over optimization. Vertices' positional encodings $\gamma(p)$ are then passed into the base MLP N_b. Outputs from N_b are passed into displacement N_d and color shift N_c residual MLPs. N_c maps to a color shift vector $c_p \in [0, 1]^3$ which is added to each vertex's base grey coloring vector $[0.5, 0.5, 0.5]^T$. N_d maps to a vertex displacement vector $d_p \in \mathbb{R}^3$. We observe CLIP to occasionally give extremely strong gradients during optimization, so to control the pace of modification we clamp the L_2 norm of d_p to be between $(-0.1, 0.1)$.

At optimization step i, the displacement vector δ_p applied to each vertex is:

$$\delta_p = \lambda \vec{n}_p + (1 - \lambda) d_p \quad (2)$$

$$\text{where } \lambda = (1 - r)^{\frac{i-1}{100}} \quad (3)$$

Over the course of optimization δ_p is a weighted average between d_p and its projection to the normal $\vec{n}_p$. A note that we project d_p to whichever is closer between the positive and negative of the vertex normal as determined by the sign of the cosine similarity between $\vec{n}_p$ and d_p. The weighting is controlled by λ such that at iteration $i = 1$, $\delta_p = \vec{n}_p$ and approaches $\delta_p = d_p$ according to λ's exponential schedule shown above. We've experimentally landed on $r = 0.4$ so $\delta_p \approx d_p$ at $i = 1,000$ iterations. This regularization of the outputs of N_d provides two benefits. (1) By biasing early displacement vectors to the normal, we constrain the optimization problem the NSF is solving and allow it to learn more complex 3D displacements later. (2) forcing early displacements to be along the normal introduces much bolder topological changes which helps overcome texture bias in optimization.

3.2. Visual Losses

The main guidance for the NSF is provided by a semantic loss between renders of mesh M^S and input text prompt t. We use a differentiable rasterizer (Kaolin) to generate renders from virtual camera positions parameterized by horizontal (azim) and vertical (elev) view angles on a sphere centered on M^S, where the radius of the sphere controls camera distance from M^S. In each iteration, we generate three sets of n_θ renders of M^S. (1) I_θ^{full} have a fixed radius of 1.5 and capture the entire shape, (2) I_θ^{local} has renders with radii randomly sampled from a uniform distribution $\mathcal{U}(1, 1.5)$ to provide focused views of surface texture, and (3) I_θ^{displ} contain renders of M^S from similarly sampled radii where the color of vertices has been switched to the default grey to obtain a geometric loss that counteracts texture bias and only updates N_d. We apply a perspective shift augmentation $\psi_{ps}(\cdot)$ to renders in I_θ^{full} to combat degenerate views.

To extract semantic information for comparison, we take the average CLIP embedding, given by $E(\cdot)$, of each of the three sets of renders and separately embed the text prompt:

$$\hat{S}^{full} = \frac{1}{n_\theta} \sum_\theta E(\psi_{ps}(I_\theta^{full})) \in \mathbb{R}^{512} \quad (4)$$

$$\hat{S}^{local} = \frac{1}{n_\theta} \sum_\theta E(I_\theta^{local}) \in \mathbb{R}^{512} \quad (5)$$

$$\hat{S}^{displ} = \frac{1}{n_\theta} \sum_\theta E(I_\theta^{displ}) \in \mathbb{R}^{512} \quad (6)$$

$$\phi_{target} = E(t) \in \mathbb{R}^{512} \quad (7)$$

The visual similarity loss is then the weighted sum of the cosine similarities $Sim_c(a, b)$ between each render set's average CLIP embedding and the target embedding of the text prompt for each render set in $\hat{S} = \{\hat{S}^{full}, \hat{S}^{local}, \hat{S}^{displ}\}$:

$$\mathcal{L}_{sim} = \sum_{\hat{S},\ s \in |\hat{S}|} w_s Sim_c(\hat{S}, \phi_{target}) \quad (8)$$

3.3. Adaptive Density Loss

Figure 3. Comparison with text2mesh - general results (maybe high-res)

As our system frequently remeshes, we want some way to adjust vertex density along the mesh's surface so regions that are more complex have more vertices around to work with. To accomplish this, we pull techniques from Jung et al. to calculate an adaptive mesh density energy term. We first consider the one-ring neighborhoods around each vertex as a local *region*. We intuitively want to allocate more vertices to be clustered in *regions* with complex geometric patterns to increase their fidelity. This allocation problem requires penalizing distance between vertices in high-complexity *regions*. With this in mind, Jung et al. use surface curvature as a proxy for *regional* geometric complexity. They leverage the fact that surface curvature is proportional to the norm of the Laplacian, and use the discrete Laplacian operator L to calculate and compare the relationships between *regional* complexity and average distance between each vertex and its one-ring neighbors.

Jung et al. begin by defining some function $l(M^S)$ which outputs a vector in $\mathbb{R}^n$ of the average L_2 distance between each vertex and its one-ring neighbors. This function can be used to create an energy term that penalizes the difference between $l(M^S)$ and some vector of target average distances $l' \in \mathbb{R}^n$:

$$E_a(M^S, l') = \frac{1}{n}\|l(M^S) - l'\|_2 \tag{9}$$

$$l(M^S)_p = \frac{1}{\mathcal{N}_p}\|(LM)_p\|_2 = \frac{1}{|\mathcal{N}_p|}\sum_{v\in\mathcal{N}_p}\|p - v\|_2 \tag{10}$$

Where $\mathcal{N}_p$ is the set of vertices in point p's one-ring neighborhood, L is the discrete Laplacian operator with a uniform weight, and $l(M^S)_p$ and $(LM)_p$ refer to the p-th row in $l(M^S) \in \mathbb{R}^n$ and $(LM) \in \mathbb{R}^{n\times 3}$ respectively.

The next step is defining per-*region* target average one-ring distances l'. Elements of l' should be larger for areas of low geometric complexity, defined as curvature, and smaller for areas of high complexity. But in order to calculate target average distances we need a prior mesh M' to evaluate. As such we define l' to be a function of M', which is the stylized mesh from a prior iteration. In implementation we calculate this prior and loss term every n_{iter} iterations making the prior $M' = M^S_{i-n_{iter}}$:

$$K_p = \frac{l(M')_p}{l_m(M')},\ K \in \mathbb{R}^{n'} \tag{11}$$

$$l_m(M') = \frac{1}{n'}\sum_{p\in M'} l(M')_p \tag{12}$$

Where $l_m(M')$ is the mean of the average one-ring distances of M' and n' is the number of vertices in mesh M'. K_p can intuitively be thought of as the local curvature, as defined by the Laplacian vertex's *region*, normalized by the average curvature of all *regions* in the mesh. We can use this normalized local complexity metric to up or down-weight the *regional* edge lengths of M' to get target *regional* edge lengths for M^S.

$$l'_k(M') = l(M') * clamp(\bar{K}/K),\ l'_k(M') \tag{13}$$

$$\bar{K} = \frac{1}{|K|}\sum_p K_p \tag{14}$$

Where $clamp(a)$ clamps the values of vector a to $[0, 1]$. $(\bar{K}/K) > 1$ for all vertices with below average local curvature and < 1 for vertices with above average local curvature. When multiplied by $l(M')$ is scales down the target lengths for *regions* of high curvature and leaves the targets untouched for *regions* of low curvature. Our adaptive density energy term can thus be re-written as:

$$E_a(M^S, l'_k(M')) = \frac{1}{n'}\|l(M^S) - l'_k(M')\|_2 \tag{15}$$

Note that we now divide by n' instead of n, since we must conform to the size of $l'_k(M')$. In implementation, we only calculate this energy term over vertices that exist in both M' and M^S. This energy term is weighted by a hyperparameter $w_E = 150$ and backpropogated to the NSF network parameters.

3.4. Remeshing

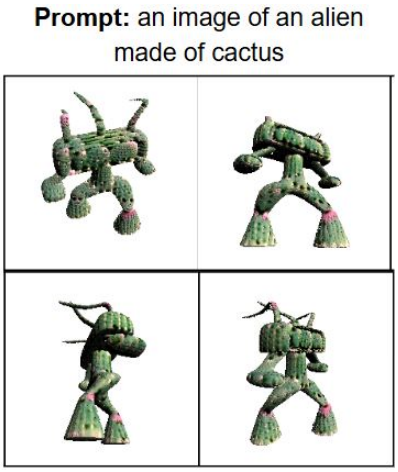

Figure 4. Low res cases

We perform three types of remeshing operations — edge split, collapse, and flip — every $n_r = 10$ iterations. Edge splits aim to add detail where the adaptive mesh density fails to sufficiently cluster enough vertices to meet style fidelity demands. Edge collapse is meant to counter-balance splits by periodically collapsing the smallest edge in *regions* with the greatest local geometric complexity. Lastly, edge flips are applied to encourage rational connectivity as geometry and vertex counts change. Below we introduce novel edge split and collapse criteria that build on the intuition developed above.

The adaptive mesh density energy term aims to subtly reallocate vertices to be clustered closer towards *regions* of high curvature. But sometimes this is not enough, and in order to add more details we must add more vertices. By identifying *regions* with abnormally high curvature, we can apply splits to all edges connecting the central vertex to its one-ring neighbors to enable further complexity to grow. The criterion for *regions* to split can be expressed using the $(\bar{K}/K)$ term from Jung et al.'s adaptive density energy.

As a refresher, $(\bar{K}/K)$ is centered around one, large for *regions* with high local curvature, and small for *regions* with low local curvature. Since we are focused on areas of high local curvature for both the split and collapse operations, our focus is on *regions* where $(\bar{K}/K) \in (0, 1)$. We set a cutoff ratio $\alpha \in (0, 1)$ and split all edges adjacent to vertices with $(\bar{K}/K) \leq \alpha$. Operationally, we place a new

vertex at the mean position of the two vertices that define the adjacent edge, then update the parameterization of the input mesh M such that the face and vertex matrices so that V can be fed into the NSF with the new vertex positions for displacement. The collapse operation works similarly. Using a predefined hyperparameter $\beta \in (0, 1)$ and $\leq \alpha$ we collapse the smallest adjacent edge for each vertex where $(\bar{K}/K) \leq \beta$. The parameters of the NSF input mesh M are updated similar to after the split operation.

Our implementation of the edge splitting operation and both the criteria and implementation of edge flip are adapted from Palfinger's 2022 work on continuous remeshing. We define the optimal degree of a vertex ω_p to be four for boundary vertices and six otherwise. Using this, we look to minimize the sum of squared vertex degree errors $\hat{\omega}_p$:

$$\Phi_\omega = \sum_p \hat{\omega}_p^2 \tag{16}$$

$$\hat{\omega}_p = \begin{cases} \omega_p - 4 \text{ if vertex is on boundary} \\ \omega_p - 6 \text{ otherwise} \end{cases} \tag{17}$$

For each possible edge connectivity flip, we can calculate its effect on Φ_ω via the following equation:

$$\Delta\Phi_\omega = \sum_{j \in \mathcal{V}} ((\hat{\omega}_p + \Delta\hat{\omega}_p)^2 - \hat{\omega}_p^2) \tag{18}$$

Where $\mathcal{V}$ is the set of four vertices considered in the flip (2 for current edge and 2 for new edge). All edge flip candidates are ranked by $\Delta\Phi_\omega$ and candidates that are within an edge of distance to another higher priority candidate are removed to avoid interference. As per Palfinger's method, the triangle face normals adjacent to the edge are checked after the flip to ensure they are not flipped compared to the mean normal of the original adjacent triangles.

References