Supplemental Material: Explorable Mesh Deformation Subspaces from Unstructured 3D Generative Models

A EXPERIMENTAL AND IMPLEMENTATION DETAILS

A.1 Latent code optimization

The given meshes in $\mathcal{M}$ are projected into $\mathcal{G}$'s latent space by minimizing the Chamfer distance of a random point sampling of each mesh, $\mathbf{P}$, to the output of the generator with respect to a latent code $\mathbf{z}_m$, making the objective $\arg\min_{\mathbf{z}_m} \mathrm{CD}(\mathcal{G}(\mathbf{z}_m), \mathbf{P})$. This optimization is done in a coarse-to-fine manner, where the sampling progressively gets denser; starting from 2^{11} and growing to 2^{15} points, doubling every 800 iterations. We used a point sampling schedule of $2048, 4069, 8192, 16384, 32768$, resampling every 800 iterations. The coarse-to-fine procedure was critical for avoiding local minima in the optimization. We visualize all projections of our shapes in each exploration space in Figures 1, 2, and 3.

A.2 Embedding of meshes into $\mathcal{E}$

The K nearest neighbor graph used to embed our landmarks into $\mathcal{E}$ uses $k = 5$ and the similarity between landmarks is determined by the dense correspondence distance given by SP-GAN's resulting point clouds. That is, the distance metric is the sum of Euclidean distances between points in $\mathcal{G}(\mathbf{z}_1)$ and $\mathcal{G}(\mathbf{z}_2)$ for two landmarks $\mathbf{z}_1, \mathbf{z}_2$.

We embed the input meshes $\mathcal{M}$ into the exploration space via a two-stage optimization process. First we optimize the embedding $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ using the triplet loss formulation given in Section 4.4, after 600 optimization iterations, we compute a Delaunay triangulation, $\mathcal{T}$, of $\mathbf{X}$. With this triangulation, we begin to impose ℓ_2 losses on the areas of triangles in $\mathcal{T}$ as well as the minimum interior angles of each triangle. In a sense, this process iteratively makes the embedding "more Delaunay." This process alone may cause the embedding to shrink to a single point, hence, we impose a constraint by "pinning" the convex hull of $\mathbf{X}$ before beginning stage two—these points do not get updated during optimization. In order to prevent the triangulation from becoming entangled (i.e. overlapping edges), we use a much smaller learning rate in the second stage of this process. We use Adam optimizer with a learning rate of 0.1 for the first stage, and 0.005 for the second stage. Finally, we execute an adjustment step that snaps points in $\mathbf{X}$ to the nearest point on the convex hull, if they are within a distance threshold from the hull. This prevents large sliver triangles from forming along the hull.

Note that alternative methods for the first stage can act as a drop-in replacement. That is, one could use, say, t-SNE or UMAP for the first stage of this pipeline (the second stage is unaffected).

A.3 Discretization of $\mathcal{E}$

We discretize the exploration space using the CAL-FEM Python package. Each facet in the Delaunay triangulation is densely subdivided into uniform elements, and the number of elements is determined by the area of the facet. The exploration spaces *chairs-100, tables-50,* and *airplanes-25* were discretized into approximately $18 \cdot 10^3$, $13 \cdot 10^3$ and $15 \cdot 10^3$ facets each.

A.4 Computation of boundary conditions

Similar to the procedure suggested by [Laine 2018], we discretize the geodesics paths into polylines, and the objective is then given by

$$\arg\min_{\mathbf{z}_1,\ldots,\mathbf{z}_{n-1}} \textstyle\sum_{i=0}^{n} \|\mathcal{G}(\mathbf{z}_i) - \mathcal{G}(\mathbf{z}_{i+1})\|^2$$

where $\mathbf{z}_{0\ldots n}$ are nodes along a polyline with n nodes. We employ a coarse-to-fine optimization of our polylines in $\mathcal{Z}$. The polylines are initialized as straight lines with 8 nodes connecting the source and target latent codes. The polyline is subdivided every 100 iterations, doubling the number of nodes until the polyline has 64 nodes.

A.5 Computing switch points & remapping boundary conditions

Our boundary conditions are given as polylines with 64 nodes that connect latents $\mathbf{z}_i$ to $\mathbf{z}_j$. In order to compute the point in which we switch from mesh $\mathbf{M}_i$ to mesh $\mathbf{M}_j$, we first compute two deformation sequences: one from $\mathbf{M}_i$ to $\mathbf{M}_j$, and the other $\mathbf{M}_j$ to $\mathbf{M}_i$. Given these two sequences of meshes, we can identify the time t^* where the chamfer distance between the meshes is minimal. We find the optimal switch point in a subsection of the deformation sequence centered around $t = 0.5$, i.e. we do not take a switch point to be, say, $t = 0.01$, rather we only consider t values in $[0.35, 0.65]$. This is to prevent overly distorting the boundary conditions.

Remapping the polyline is done by dilating both sides of the polyline such that t^* is mapped exactly to $t = 0.5$. Hence all of the switch points' boundaries can be visualized by a standard Voronoi diagram.

A.6 The energy-minimizing map Φ

Training. Our map Φ from exploration space to $\mathcal{Z}$ is implemented as a four layer multi-layer perceptron with sinusoidal positional encoding [Mildenhall et al. 2020], where $L = 5$

$$\gamma(\mathbf{x}) = \left(\sin(2^0\pi\mathbf{x}), \cos(2^0\pi\mathbf{x}), ..., \sin(2^L\pi\mathbf{x}), \cos(2^L\pi\mathbf{x})\right).$$

The MLP is optimized via stochastic gradient descent with a learning rate of $3 \cdot 10^{-4}$ as per the objective in Equation 2. We use batch sizes of 256 with gradient accumulation. Our training times vary from 12 to 24 hours for our smallest and largest exploration spaces.

Inference. Our formulation of the submanifold objective (Section 4.2) assumes that $\mathcal{Z}$ is piecewise linear, hence in order to do inference on Φ properly, we must query it in a similar fashion. At inference time, a point in exploration space $\mathbf{x} \in \mathcal{E}$ is decoded into primal space by first identifying which FEM facet $\mathbf{x}$ resides in. Then we compute a barycentric interpolation of the functional values of the facet's vertices lifted into primal space. Additionally, if any of these vertices lie on our boundary, we "swap" out the functional value with the boundary value, which is the same procedure that we use during training.

A.7 Deformation module

For a given path connecting source and target points in exploration space, we accumulate the vertex displacements via the discrete Euler method in Equation 4. For all rendered results in the paper, we integrate over 180 samples along the given path. We employ a custom GPU-accelerated smoothing radial basis function interpolator that has a throughput of 72 steps per second, which facilitates real-time interaction. The smoothing parameter is chosen for each exploration space independently: 15, 50, 15 for the *airplanes-25*, *tables-50*, *chairs-100* spaces respectively. Preliminaries for RBF interpolation are in Appendix B.

A.8 Obtaining shapes for exploration spaces

Our exploration spaces: *chairs-100*, *tables-50*, *airplanes-25* were populated using a semi-random shape retrieval routine. We start with a set of N hand-picked shapes ($N = 5$ in our case), then iteratively select shapes from the dataset based on similarity. In particular, at each step in this process we sample a random shape from the top-K nearest shapes in ShapeNet. We vary the K parameter across experiments to accommodate for lack of diversity in certain shape categories. In particular, we used $K = 2000$, $K = 1500$, and $K = 5000$ for chairs, tables, and airplanes respectively. A temperature parameter γ was used to control a similarity-weighted softmax over the retrieved K shapes. Finally, the distance metric used to measure shape similarity is the dense-correspondence distance given by SP-GAN, as mentioned in Section 6.

B RBF INTERPOLATION PRELIMINARIES

Given a set of data samples $f(\mathbf{x}_0), \dots, f(\mathbf{x}_n)$ and *radial basis functions* centered those samples, $\phi(\mathbf{x}_k) = \hat{\phi}(\|\mathbf{x} - \mathbf{x}_k\|)$, RBF interpolation represents an interpolant as a weighted combination of the bases

$$\begin{aligned} S(\mathbf{x}) &= \sum_{i=0}^{n} w_i \cdot \phi(\|\mathbf{x} - \mathbf{x}_i\|), \\ \text{s.t.} \quad S(\mathbf{x}_i) &= f(\mathbf{x}_i)\ \forall i \end{aligned}$$

whose weights can be solved via the linear system $\mathbf{A}\mathbf{w} = \mathbf{b}$, where

$$\begin{bmatrix} \phi(\|\mathbf{x}_0 - \mathbf{x}_0\|) & \dots & \phi(\|\mathbf{x}_n - \mathbf{x}_0\|) \\ \phi(\|\mathbf{x}_0 - \mathbf{x}_1\|) & \dots & \phi(\|\mathbf{x}_n - \mathbf{x}_1\|) \\ \vdots & \ddots & \vdots \\ \phi(\|\mathbf{x}_0 - \mathbf{x}_n\|) & \dots & \phi(\|\mathbf{x}_n - \mathbf{x}_n\|) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} f(\mathbf{x}_0) \\ f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_n) \end{bmatrix}$$

A common choice for the radial function is the *thin plate spline*, which is given by $\phi(r) = r^2 \ln r$, and is our choice of basis function.

The smoothing variant of RBF interpolation requires that we add a polynomial term to the interpolant. That is, we add to $S(\mathbf{x})$ a sum of weighted monomials up to a specified degree, and require that the interpolant evaluates to a be *exactly* polynomial if the data itself comes from a polynomial (this additional condition is necessary to specify a unique solution). The linear system eventually becomes

$$\begin{bmatrix} \mathbf{A} + \lambda\mathbf{I} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}$$

where $\mathbf{P}$ is a matrix of monomial terms and $\mathbf{d}$ is their coefficients. Finally, the smoothing parameter λ (which is mentioned in our method), is incorporated by adding a constant term to the diagonal of $\mathbf{A}$. For further reading, please refer to [Anjyo et al. 2014].

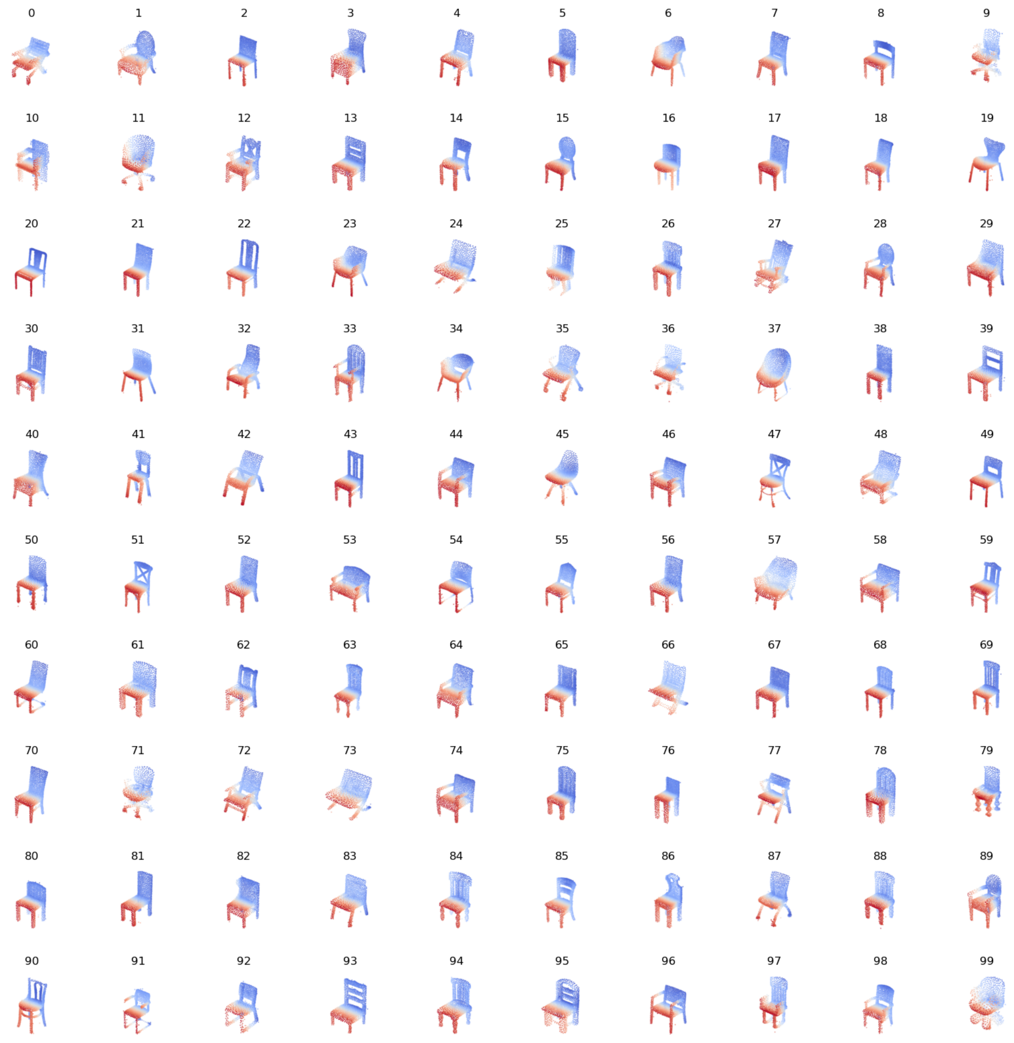

Figure 1: Point cloud visualizations of the chairs in *chairs-100* after being projected into SP-GAN's latent space.

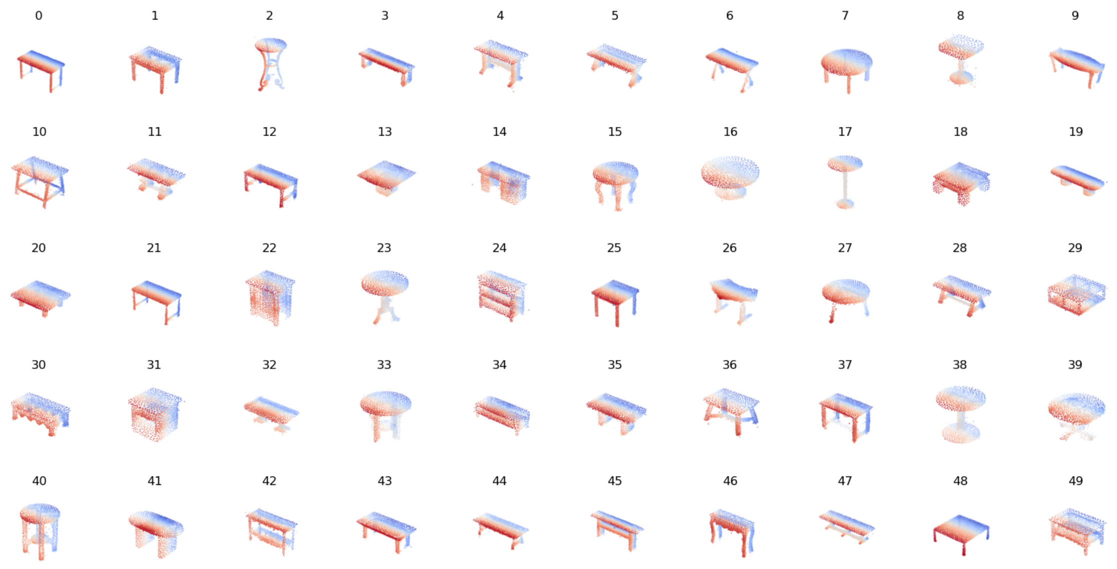

Figure 2: Point cloud visualizations of the tables in *tables-t0* after being projected into SP-GAN's latent space.

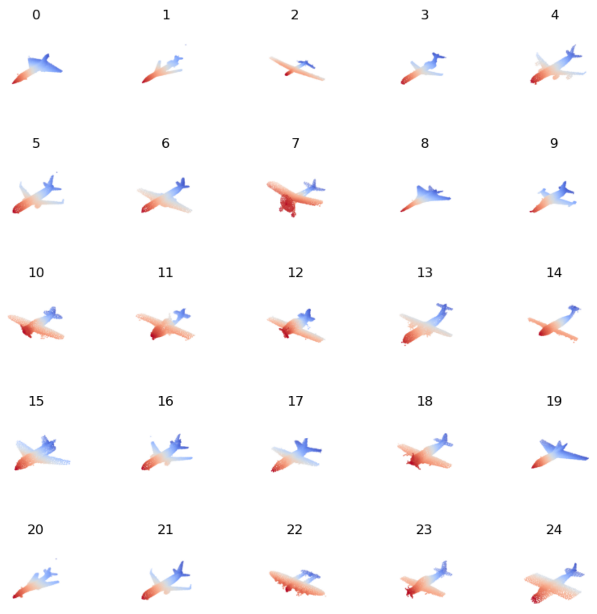

Figure 3: Point cloud visualizations of the airplanes in *airplanes-25* after being projected into SP-GAN's latent space.