Proceedings of the ASME 2016 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2016
August 21-24, 2016, Charlotte, USA

IDETC2016-60112

HOW DESIGNS DIFFER: NON-LINEAR EMBEDDINGS ILLUMINATE INTRINSIC DESIGN COMPLEXITY

Wei Chen
Dept. of Mechanical Engineering
University of Maryland
College Park, Maryland 20742
Email: wchen459@umd.edu

Jonah Chazan
Dept. of Computer Science
University of Maryland
College Park, Maryland 20742
Email: jchazan@umd.edu

Mark Fuge*
Dept. of Mechanical Engineering
University of Maryland
College Park, Maryland 20742
Email: fuge@umd.edu

ABSTRACT

This paper shows how to measure the complexity and reduce the dimensionality of a geometric design space. It assumes that high-dimensional design parameters actually lie in a much lower-dimensional space that represents semantic attributes. Past work has shown how to embed designs using techniques like autoencoders; in contrast, this paper quantifies when and how various embeddings are better than others. It captures the intrinsic dimensionality of a design space, the performance of recreating new designs for an embedding, and the preservation of topology of the original design space. We demonstrate this with both synthetic superformula shapes of varying non-linearity and real glassware designs. We evaluate multiple embeddings by measuring shape reconstruction error, topology preservation, and required semantic space dimensionality. Our work generates fundamental knowledge about the inherent complexity of a design space and how designs differ from one another. This deepens our understanding of design complexity in general.

INTRODUCTION

Products differ among many design parameters. For example, a wine glass contour can be designed using coordinates of B-spline control points: for 20 B-spline control points, a glass would have at least 40 design parameters. However, we cannot arbitrarily set these parameters because the contour must still look like a wine glass. Therefore, high-dimensional design parameters actually lie on a lower-dimensional manifold—what we call a *semantic space*—that encodes semantic design features, such as roundness or slenderness.

Past researchers, both within and beyond design, have proposed many algorithms for mapping high-dimensional spaces to lower dimension manifolds and back again—what we call a *design embedding*. But how do you choose which of these methods you should use for a given design space? What properties should a good design embedding possess? What, if anything, do embeddings tell us about the broader nature of a design space?

This paper answers those questions by proposing methods to study design embeddings. Specifically, by comparing three types of design embeddings—Principal Component Analysis (PCA), Kernel PCA, and Neural Network Autoencoders—along several factors that matter to design: accurately reconstructing embedded designs, preserving a design space's topology, and distributing points well within the semantic space. We demonstrate our approach on synthetic superformula examples [1] with varying non-linearity and a real world example of glassware.

Our approach helps novice designers rapidly and intelligently create new shapes by selecting appropriate embeddings that capture the complexity of design spaces. More importantly, our work deepens our understanding of design complexity in general by illuminating how design embeddings model a design space, how to judge the inherent complexity of a design space, and how designs differ from one another.

*Address all correspondence to this author.

RELATED WORK

Our hypothesis is that while shapes are represented in a high-dimensional design space, they actually vary in a lower-dimensional manifold, or surface within the larger design space [2, 3]. One would then navigate a shape space by moving across this manifold—like an ant walking across a surface of a sphere—while still visiting all valid designs. This raises several questions: How does one find the manifold (if it exists)? How does one "jump back" to the high-dimensional space (raw geometry) once one has wandered around on the manifold? How does one choose a manifold that has similar topological structure to the high-dimensional space?

Design Space Exploration and Shape Synthesis

Past work has explored and synthesized new shapes. MacDonald *et al.* [4] used conjoint analysis to design wine bottle shapes that conveyed semantic messages of wine flavor. Yumer *et al.* [5] proposed a method to allow continuous shape editing by using semantic attributes. Yumer *et al.* [6] used an autoencoder to enable continuous exploration of the high dimensional procedural modeling spaces within a lower dimensional space representing shape features. Ulu *et al.* [7] trained a neural network to map between the loading configurations and the optimal topologies, and estimate the optimal topologies for novel loading configurations. Ulu *et al.* [8] proposed a biologically inspired growth algorithm to automatically generate support structures based on the input structures. Unlike past work, this paper directly investigates the inherent complexity of a design space.

Manifold Learning Methods

Past researchers have taken four complementary approaches to project high-dimensional data set onto a manifold: 1) Using linear transformation like Principal Component Analysis (PCA) to find low-dimensional subspace that retains maximal variance [9]; 2) Creating Embeddings that minimize nearest-neighbor graph or distance quantities, including approaches like Principal Surfaces [10, 11], Matrix Factorizations [12, 13], and variants of Local Linear Embedding and Multi-Dimensional Scaling [5]; 3) Using embeddings within a Reproducing Kernel Hilbert Space (called *Kernel Methods*) [14, 15]; or 4) using multi-layer Neural Networks such as autoencoders to extract low-dimensional features from high-dimensional data [16, 6]. Our approach builds upon PCA, Kernel PCA, and the autoencoder and introduces new ways to compare and contrast those manifolds—or embeddings—to better understand how they capture a design space.

Original Design Space Reconstruction

To "jump down" to the lower-dimensional manifold is not enough; to explore high-dimensional design space (raw geometry) using the low-dimensional semantic attributes, it is also essential to "jump up" (to the high-dimensional raw geometry)—most techniques do one direction well, but not the other [17]. To solve this, past research has either: used two (separately trained) manifold learning approaches (one high-to-low, another low-to-high) [5, 18], or have used models that train both directions simultaneously [6, 19]. The methods used in our work, *i.e.*, PCA, Kernel PCA, and the autoencoder, map both directions ("jump-down" and "jump-up") simultaneously—the first two methods use the learned parameters to do inverse transform; while an autoencoder simultaneously trains an encoder and a decoder so that it can both encode high-dimensional inputs to a low-dimensional set of central network activations and then back out by decoding these activations to the original inputs.

Topology Preservation in Manifold Learning

For a manifold learning method, the topological structure of the original high-dimensional space should be preserved in the projected lower-dimensional space [20]—points far away or near to each other in the original space should likewise be far or near in the projected space. In our case, two objects with similar shapes should also have similar semantic features, and vice versa. One way to evaluate this topology preservation is by measuring the difference between the nearest neighbors graph of high-dimensional data set and its low-dimensional projection [21, 22]. Another way is to measure the preservation of pairwise distances between points in the two spaces [23, 24]. Persistent homology [25] can also be used for evaluating topology preservation of dimensionality reduction methods with high robustness against noise [26].

SAMPLES AND DESIGN SPACE CONSTRUCTION

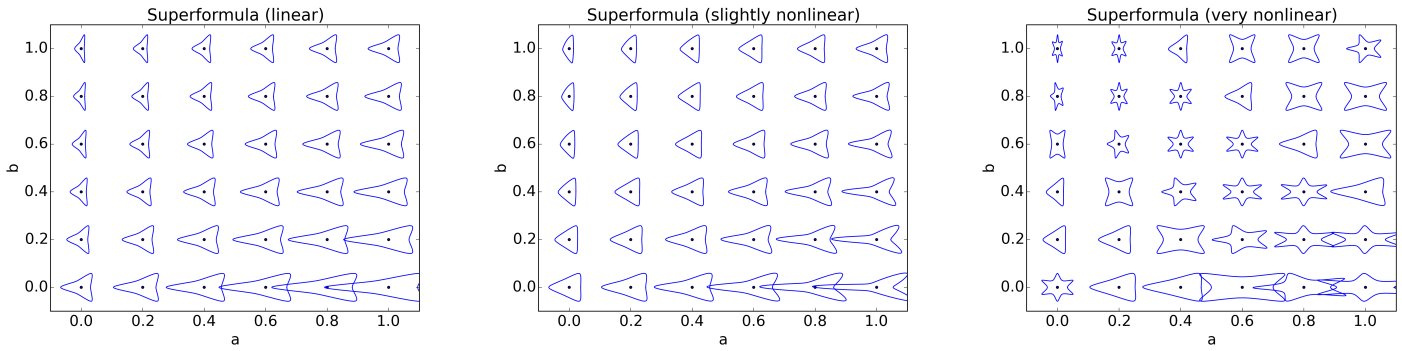
To create a high-dimensional design space $\mathcal{X}$, we generate a set of design parameters or shape representations $X \in \mathcal{X}$ as training and testing data. We fit sample shapes with B-spline curves and use coordinates of B-spline control points as X. The sample shapes come from two sources: 1) the superformula [1] as synthetic example to control and compare the complexity of design spaces; and 2) glassware contours as a real world example.

Superformula Examples

The superformula generalizes the ellipse. With a radius (r) and an angle (θ), the superformula can be expressed in polar coordinates:

$$r(\theta) = \left(\left| \frac{\cos(\frac{m\theta}{4})}{\alpha} \right|^{n_2} + \left| \frac{\sin(\frac{m\theta}{4})}{\beta} \right|^{n_3} \right)^{-\frac{1}{n_1}} \quad (1)$$

According to Gielis, two-dimensional superformula shapes have a multidimensional parameter space in $\mathbb{R}^6$ with the various parameters $(\alpha, \beta, m, n_1, n_2, n_3)$ [1]. For ease of visualization,



(a) Linear superformula (b) Slightly nonlinear superformula (c) Highly nonlinear superformula

FIGURE 1: THE ORIGINAL *a*-*b* SPACE OF SUPERFORMULA EXAMPLES.

we look at two-dimensional subspaces of the superformula parameter space. To artificially control the dimensionality of the design space, however, we fit B-splines to the generated shapes and use the coordinates of the spline control points as our high-dimensional features. Our examples consist of three different cases in which we control two variables a and b such that the spline control points—the design space—vary at different levels of non-linearity.

The first example (the "linear example") is the set of linear scalings of one particular case of the superformula (Fig. 1a). The spline points (x,y) vary linearly up to noise from the fitting process, since the shape is simply stretched.

$$(x,y) = (a \cdot r(\theta)\cos\theta, b \cdot r(\theta)\sin\theta)$$
$$(\alpha,\beta,m,n_1,n_2,n_3) = (1,1,3,7,18,18)$$

The second example (the "slightly nonlinear example") involves manipulating n_1, n_2, and n_3. These parameters change whether the shape is circumscribed or inscribed in the unit circle as well as otherwise altering the shape. Thus a shape can either be convex or concave (Fig. 1b). The points of the spline therefore do not vary exactly linearly in a and b. This means that the changes in the spline points are not linear, and the space is two-dimensional. These changes, however, are fairly small compared to that of stretching, so a linear transformation can still explain much of the variation.

$$(x,y) = (a \cdot r(\theta)\cos\theta, b \cdot r(\theta)\sin\theta)$$
$$(\alpha,\beta,m,n_1,n_2,n_3) = (1,1,3,7-a+b,12+a,12+b)$$

The third example (the "highly nonlinear example") is also in a two-dimensional space, but involves manipulating the symmetry of the resulting shapes (m), so that the splines vary drastically as parameters change (Fig. 1c). A linear approximation should therefore explain far less of the variation.

$$(x,y) = (a \cdot r(\theta)\cos\theta, b \cdot r(\theta)\sin\theta)$$
$$(\alpha,\beta,m,n_1,n_2,n_3) = (1,1,3+(a+b) \mod 4,7,12+a,12+b)$$

We randomly choose parameters a and b to generate training samples and testing samples for the above three cases. Fig. 1a, Fig. 1b and Fig. 1c shows the generated shapes according to parameters a and b for linear, slightly nonlinear and highly nonlinear cases, respectively. We call such space the *original a-b space*, and compare it with the 2-dimensional semantic space $\mathcal{F}$ to evaluate an embedding.

Real World Glassware Example

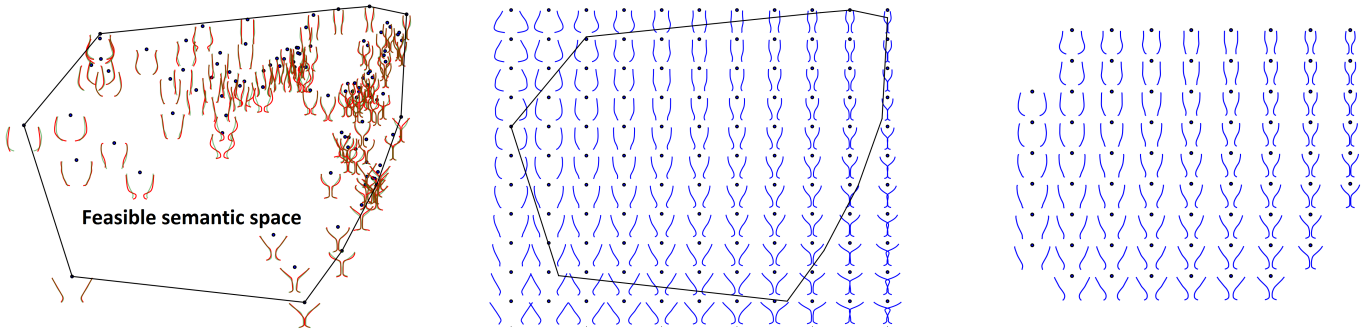
Glassware is a good real world example because 1) B-splines shape representation (design parameters) smoothly fits the glass boundary contours, and 2) we can interpret the semantic features of glassware—*e.g.*, roundness, slenderness, type of drink, *etc.*We use 115 glass image samples, including wine glasses, beer glasses, champagne glasses, and cocktail glasses. We fit each glass contour with a B-spline curve, and build the design space $\mathcal{X}$ using the coordinates of B-spline control points.

METHODOLOGY

To understand design embeddings, we first train three dimensionality reduction models that map from a high-dimensional design space $\mathcal{X}$ to a lower-dimensional semantic space $\mathcal{F}$, and back again. Then given new points in $\mathcal{F}$ (*i.e.*, semantic attributes), we can reconstruct designs using the trained model, and vice versa. To compare design embeddings, we use two metrics—reconstruction error and topology preservation. Our overall approach consists of four steps: model training, shape reconstruction, model evaluation, and confidence prediction.

Model Training

We compare three different dimensionality reduction methods: PCA, kernel PCA, and an Autoencoder. These methods all simultaneously learn a mapping from the design space $\mathcal{X}$ to the semantic space $\mathcal{F}$ ($f : \mathcal{X} \to \mathcal{F}$), and from the semantic space back to the design space ($g : \mathcal{F} \to \mathcal{X}$).



(a) Create a convex hull of the training set in the semantic space

(b) Copy the boundary of the convex hull to the grid

(c) Remove samples outside the boundary

FIGURE 2: RECONSTRUCTED SHAPES IN THE FEASIBLE SEMANTIC SPACE.

PCA performs an orthogonal linear transformation on the input data. Kernel PCA is an extension of PCA which achieves nonlinear transformation using the kernel trick [27]. An autoencoder is an artificial neural network nonlinearly maps $\mathcal{X}$ and $\mathcal{F}$ when it uses nonlinear activation functions. According to Hornik et al. [28], a "shallow" neural network with only one arbitrarily large hidden layer can approximate a mapping to any level of precision. Thus an autoencoder should embed designs well when you possess sufficient data to train it.

We split any design data into a training set and testing set. We further split the training data via 10-fold cross validation to optimize the hyper-parameters using Bayesian optimization [29]. After optimizing any hyper-parameters, the model trains each example with the entire training set. We test each model with the testing set. For those who wish to fully reproduce our results, we have put our full experiment code, including the settings for kernel PCA and the autoencoder on GitHub [1].

New Shape Reconstruction

After mapping from design space $\mathcal{X}$ to semantic space $\mathcal{F}$, we also need to map back to the original $\mathcal{X}$ from $\mathcal{F}$—*i.e.*, given certain semantic attributes, we want to generate new shapes. To visualize this mapping $g : \mathcal{F} \to \mathcal{X}$, we uniformly sample from $\mathcal{F}$, reconstruct those samples in $\mathcal{X}$, and then plot the reconstructed shapes in Fig. 2b. Some of those reconstructed shapes, however, may not be valid real world shapes—*e.g.*, glasses on the right bottom of Fig. 2b, where the glass contours on the two sides intersect. We call these *infeasible shapes*. To limit $\mathcal{F}$ to only feasible shapes, we take the convex hull of the training samples in $\mathcal{F}$ as shown in Fig. 2a. We sample $\mathcal{F}$ inside the convex hull creating the glass shape distribution—the feasible semantic space—shown in Fig. 2c. Because training samples all have feasible shapes, the created samples lying between any two training samples should show valid designs if the semantic space preserves the topology of the original design space (*i.e.*, the space is not highly distorted). This method does take some risks of removing potentially innovative designs. However, whether the generated shapes are infeasible, absurd, or innovative is difficult to tell since by definition designs off the manifold are "unusual." This paper's main focus is exploring the complexity of an existing design space rather than recommending innovative designs, though this could be an interesting topic for future research.

Evaluation

To evaluate embeddings, we consider these questions:

1. Given known semantic attributes (*e.g.*, roundness and slenderness), can the embedding precisely restore the original design parameters that created it?
2. Do samples with similar shapes (design parameters) have similar semantic attributes?
3. Do shapes in the semantic space ($\mathcal{F}$) vary similarly compared to the original shapes in the design space ($\mathcal{X}$)?

To answer these questions, we propose two metrics for comparing embeddings: 1) reconstruction error and 2) topology preservation.

Reconstruction Error. Reconstruction error measures how the actual B-spline control points differ from the reconstructed control points:

$$\varepsilon = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \| r_j^{(i)} - s_j^{(i)} \| \tag{2}$$

where m is the number of designs, n is the number of spline control points, $r_j^{(i)}$ is the ith reconstructed control point for the jth

[1] https://github.com/IDEALLab/design_embeddings_idetc_2016

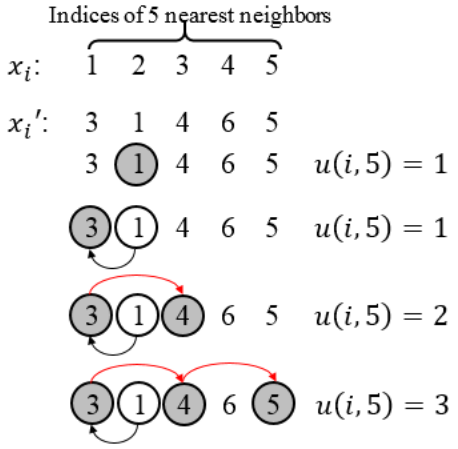


FIGURE 3: ALGORITHM FOR COMPUTING TOPOLOGICAL ALIGNMENT.

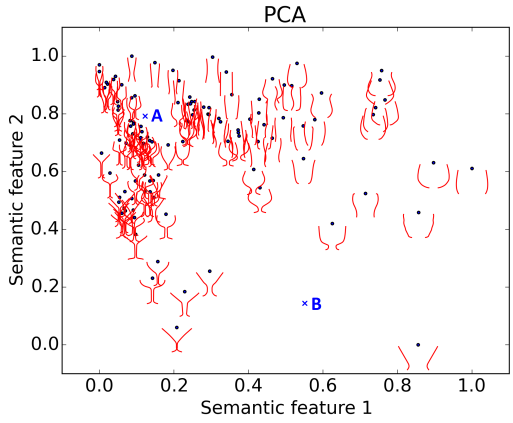
design, and $s_j^{(i)}$ is the ith original control point for the jth design.

Topology Preservation. To answer the second and third question, we have to compare the topologies of the high-dimensional design space $\mathcal{X}$ and the low-dimensional semantic space $\mathcal{F}$. Specifically, we compare the nearest neighbors of each sample. If the nearest neighbors of each point in $\mathcal{F}$ are also its nearest neighbors in $\mathcal{X}$, similar shapes will remain close in $\mathcal{F}$. Furthermore, we prefer that shapes that vary continuously in $\mathcal{X}$ also do so in $\mathcal{F}$. Thus each design in either space should have similar neighbors in similar proximity. If an embedding satisfies those two conditions, it preserves the topology of $\mathcal{X}$ after projecting into $\mathcal{F}$.

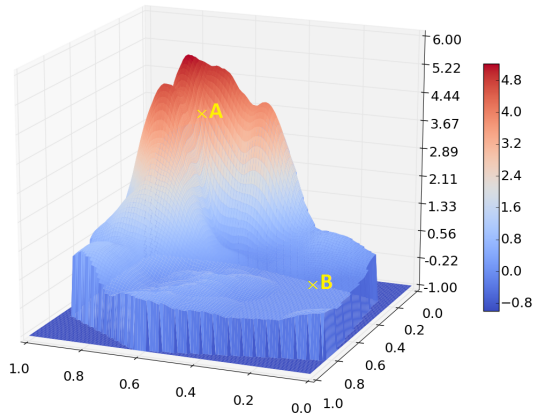
We measure how well an embedding preserves a neighborhood ordering using the ordered neighborhoods coincidence (ONC) index [22]:

$$o(k) = \frac{1}{pk}\sum_{i=1}^{p} u(i,k) \tag{3}$$

where k defines the neighborhood size around each point; p is the number of designs in the set; i is a design's index in that set; and $u(i,k)$ compares the index for each point x_i in $\mathcal{X}$ compared to a corresponding point x_i' in $\mathcal{F}$. It computes the the number of neighbors arranged in the same topological ordering in both $\mathcal{X}$ and $\mathcal{F}$. We get this topology alignment number $u(i,k)$ using the algorithm shown in Fig. 3. For a point x_i in $\mathcal{X}$, we find a list of its neighborhood indices $L = [n_1^{(i)}, n_2^{(i)}, ..., n_k^{(i)}]$. Then given $f : \mathcal{X} \rightarrow \mathcal{F}$, we get $x_i' = f(x_i)$ and its corresponding neighborhood indices $L' = [n_1'^{(i)}, n_2'^{(i)}, ..., n_k'^{(i)}]$ in $\mathcal{F}$. We look for items of L one by one in L' and skip any items which do not exist in L.



(a) Distribution of training samples in semantic space



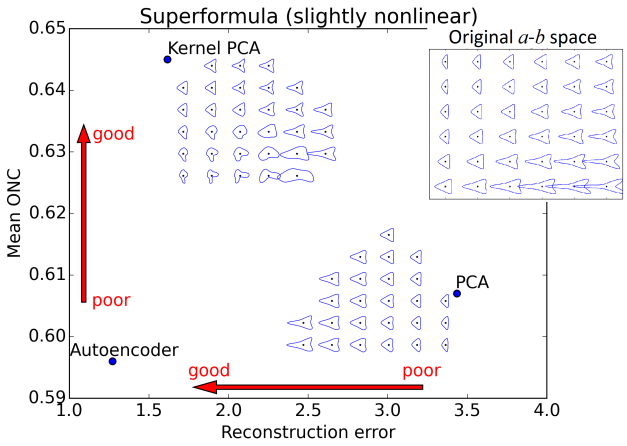
(b) Kernel density estimation using training samples

FIGURE 4: TRAINING SAMPLE SPARSITY IN SEMANTIC SPACE.

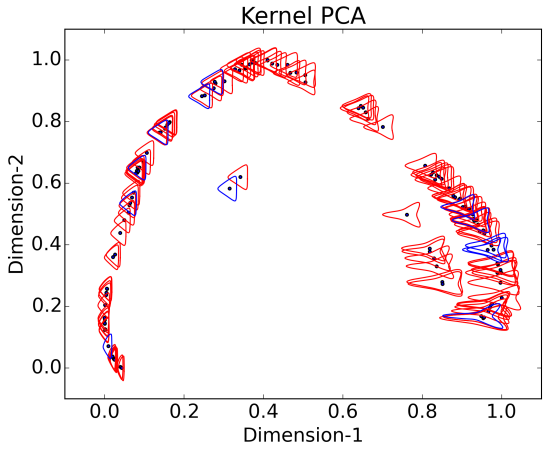
To determine the topological alignment, we add directed edges in L' to connect two successive neighbors in L. For example, if $n_j = n_p'$ and $n_{j+1} = n_q'$, we add an edge: $n_p' \rightarrow n_q'$. The topology alignment number $u(i,k)$ is obtained by counting the number of edges pointing forward. When k is small, the index $o(k)$ indicates the local topology preservation; as k grows, $o(k)$ indicates global topology preservation.

Confidence of Shape Reconstruction

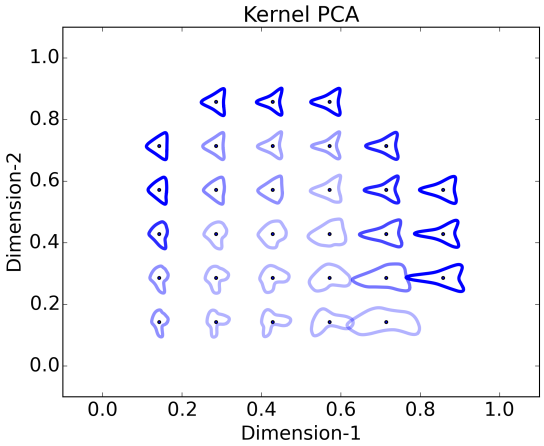
Reconstruction error and topology preservation evaluate an embedding over training and testing sets. But the semantic space $\mathcal{F}$ may have areas devoid of training or testing samples (a sample-sparse area). How do we evaluate new shapes reconstructed in those areas? If samples are evenly distributed then reconstruction error can adequately indicate correctness. In re-



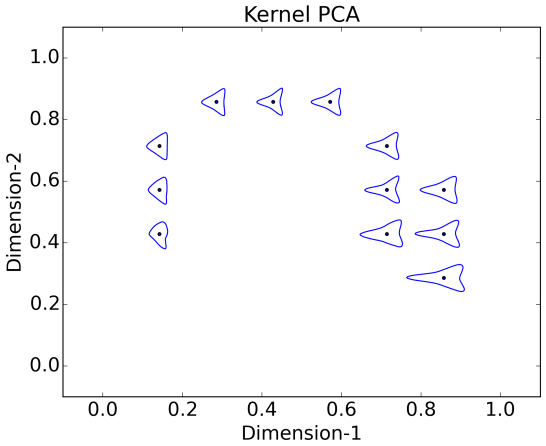
(a) Reconstruction error and mean ONC (a method with the best performance should be on the upper left corner)



(b) Distribution of training and testing samples in semantic space (red: training samples; blue: testing samples)



(c) Density evaluation for new reconstructed shapes (darker color means higher density)



(d) Remove shapes with low density evaluation

FIGURE 5: IRREGULAR SEMANTIC SPACE IN THE SUPERFORMULA EXAMPLE.

ality, however, training and testing samples may concentrate in certain areas of $\mathcal{F}$ (*e.g.*, Fig. 5b), making it hard to measure the reconstruction accuracy in sample-sparse areas.

Figure 4a shows a distribution of training samples in $\mathcal{F}$. Most training samples concentrate at the top left and we are confident that a new shape near that region will reconstruct correctly—provided the embedding has low reconstruction error and good topology preservation. But what about the reconstructed shapes far away from the training samples? For example, compare the model's prediction at A and B in Fig. 4a: we would have higher confidence in the reconstruction at A, compared to B.

So how do we quantify this "reconstruction certainty"? If we assume that the learned model can only guarantee accurate reconstruction near training samples, we can use an Epanechnikov kernel density estimation (KDE) to approximate the probability density function of shapes in the semantic space:

$$K(x;h) \propto 1 - \frac{x^2}{h^2} \tag{4}$$

where h is the bandwidth of the kernel. Fig. 4b shows the kernel density estimate for the training set in Fig. 4a. The KDE outside the feasible semantic space is not taken into consideration. Based on the KDE, the probability of correct shape reconstruction at B is much lower than that at A.

Fig. 5 shows an example where the low reconstruction error and high topology preservation cannot guarantee the correct-

ness of the shapes far away from training samples. As shown in Fig. 5a, although kernel PCA has better performance than PCA based on the reconstruction error and topology preservation, some of the new shapes reconstructed by kernel PCA seem obviously unexpected (compared to the original designs in Fig. 1b), while shapes reconstructed by PCA are more reasonable. Those unreasonable shapes reconstructed by kernel PCA lie in a semantic cavity created by training samples (Fig. 5b). After applying Epanechnikov kernel density estimation (Fig. 5c) and removing the shapes with low density evaluation, we obtain shapes with high probability to be correct, as shown in Fig. 5d.

To avoid sample-sparse areas where reconstructed shapes have low certainty, we would prefer to spread training and testing samples throughout the entire semantic space—even at the cost of higher overall reconstruction error. Our future work will address this issue by finding a way to enhance the regularity of training sample distribution in the projected semantic space.

RESULTS AND DISCUSSION

We compare design embeddings across both synthetic and real-world examples, breaking our results into four sections:

1. How we constructed the samples we used for our results.
2. How well the embeddings capture known semantic features.
3. How well embeddings capture the known intrinsic dimensionality of the design space.
4. How well embeddings respond to changes in design space complexity—as expressed by non-linearity.

We expect that 1) as the non-linearity of the design parameters increases, embeddings might struggle to correctly capture any semantic features; 2) PCA should do well when learning the semantic features in the linear case, but fail to do so as nonlinearity increases, since it is only a linear transformation; and 3) Kernel PCA and Autoencoders with optimized hyperparameters should improve over PCA in non-linear cases, though they may transform the topology of the space or create semantic cavities more so than PCA.

Do Embeddings Capture Semantic Features?

Figures 6a–6c show the superformula shape distribution in the learned two-dimensional semantic space $\mathcal{F}$ for the linear superformula case. All the shapes shown in the figures are scaled to the same height. Thus the aspect ratio will matter instead of the height and the width and we expect an embedding to capture this dependence on aspect ratio. By looking at the semantic space learned by PCA (Fig. 6a) and kernel PCA (Fig. 6b), widths of superformula patterns change along the horizontal axis, whereas widths do not vary. In the autoencoder case, the learned semantic feature also captures the aspect ratio along the direction of the arrow in Fig. 6c. All three embeddings successfully capture the semantic features of the original *a*-*b* space for the linear superformula example (Fig. 1a). We expected this because in a linear design space both linear and non-linear embeddings should do well.

In the slightly nonlinear case, the original *a*-*b* space (Fig. 1b) demonstrates that convexity changes along with the aspect ratio (*i.e.*, as the aspect ratio increases, the convexity decreases). Thus an embedding should capture these two features. PCA (Fig. 7a) and the autoencoder (Fig. 7c) do this well. We did not expect PCA to capture these features; the non-linearity may not have been enough to prevent PCA from capturing all the semantic features well. For kernel PCA (Fig. 7b), new predicted shapes with high certainty (darker colored) match shapes shown in the original *a*-*b* space; while shapes with low certainty (lighter colored) do not—despite low reconstruction error and high topology preservation (Fig. 7d). Compared to PCA and the autoencoder, kernel PCA predicts unstable new shapes—that is, proposes good shapes around training and testing samples but bad shapes elsewhere in $\mathcal{F}$.

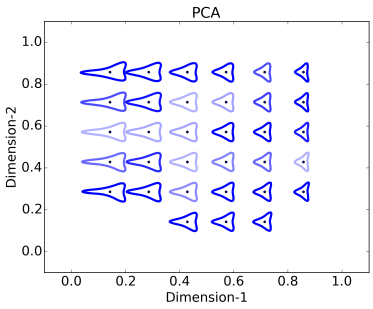
In the highly nonlinear case, more complex features vary in the original *a*-*b* space, such as the number of angles. PCA (Fig. 8a) and kernel PCA (Fig. 8b) struggle to embed such complex features—both have large reconstruction error and thus cannot guarantee accurate reconstruction of training samples, let alone new shapes. For the autoencoder (Fig. 8c), the reconstructed shapes compare well with the original *a*-*b* space.

For the glass example (Fig. 9), the bottom radius and the top radius generally differentiate the training examples. All three embeddings fully capture these main semantic features. Kernel PCA (Fig. 9b), for example, places champagne glasses at the top right corner of the 2-dimensional semantic space, cocktail glasses at the bottom right, and beer glasses at the top left.

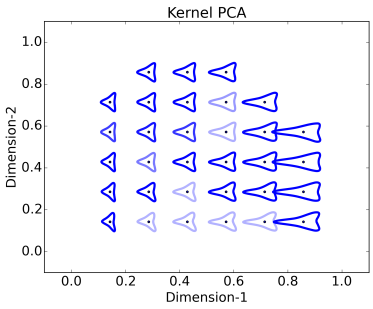
Intrinsic Dimensionality

The intrinsic dimensionality D is the minimum number of dimensions required to precisely reconstruct the design space $\mathcal{X}$. For example, we use parameters a and b as two variables when constructing superformula shapes. Although $\mathcal{X}$ is a high-dimensional space that contains twenty B-spline control points, its intrinsic dimensionality is still two ($D = 2$). Ideally, an embedding should approximate the mapping $f : (x_1, y_1, x_2, y_2, ...) \rightarrow (a, b)$ where (x_i, y_i) is the coordinate of the ith B-spline control point. Or more generally, an embedding should capture how designs differ using as few semantic dimensions (d) as possible. For PCA and Kernel PCA, d is the number of principal components; and for autoencoders, d is the number of nodes in the deepest hidden layer. As d grows, we should expect all embeddings to better describe the design manifold, reconstructing the original design data more precisely.

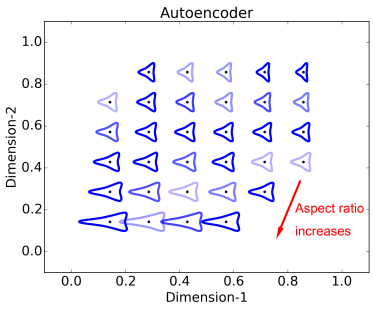
When the semantic space dimensional reaches the intrinsic dimensionality ($d = D$), we would expect reconstruction error to drop sharply and flatten out. PCA should excel at indicating the intrinsic dimensionality D when the design space is linear. In non-linear design spaces, Kernel PCA and autoencoders should



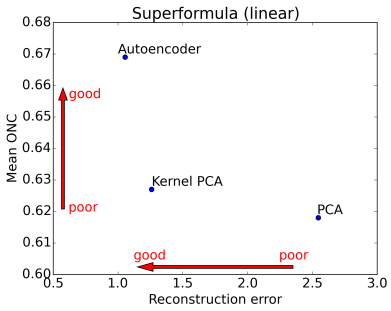
(a) Semantic space learned by PCA



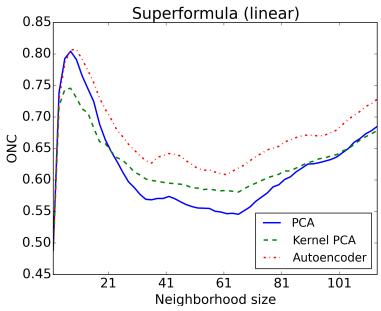
(b) Semantic space learned by Kernel PCA



(c) Semantic space learned by Autoencoder

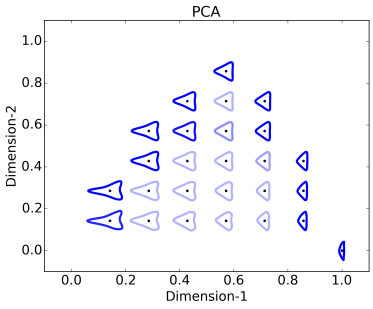


(d) Reconstruction errors and mean ONC indices

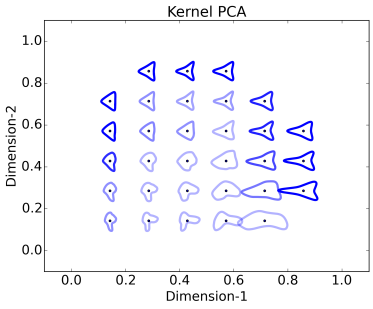


(e) ONC index

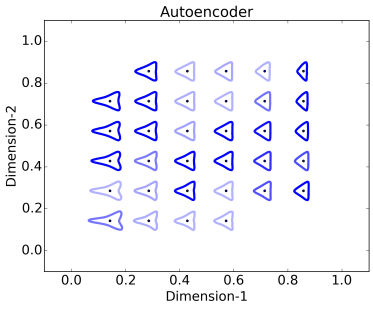
FIGURE 6: THE LINEAR SUPERFORMULA EXAMPLE (SEMANTIC SPACE DIMENSIONALITY = 2).



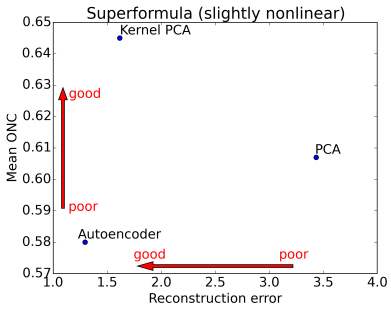
(a) Semantic space learned by PCA



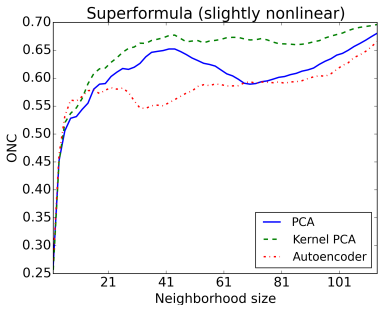
(b) Semantic space learned by Kernel PCA



(c) Semantic space learned by Autoencoder

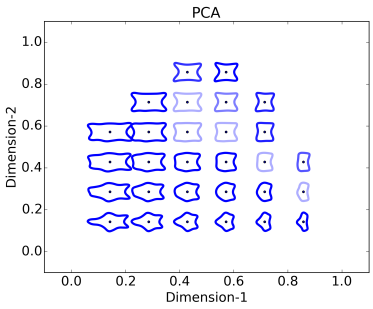


(d) Reconstruction errors and mean ONC indices

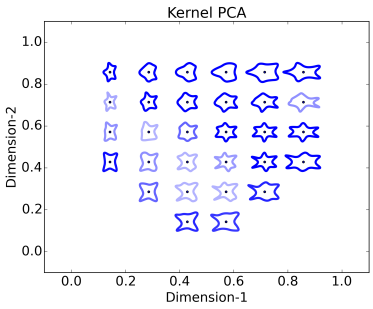


(e) ONC index

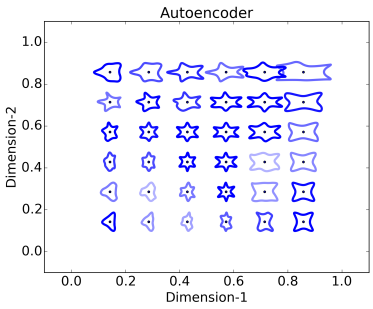
FIGURE 7: THE SLIGHTLY NONLINEAR SUPERFORMULA EXAMPLE (SEMANTIC SPACE DIMENSIONALITY = 2).



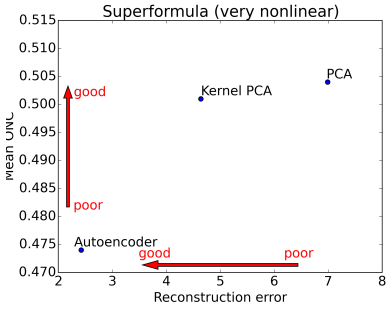
(a) Semantic space learned by PCA



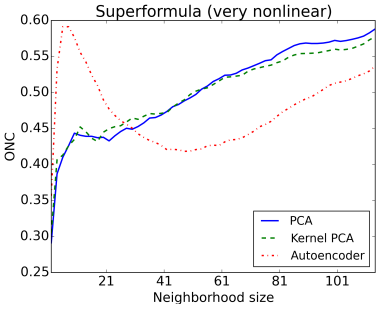
(b) Semantic space learned by Kernel PCA



(c) Semantic space learned by Autoencoder

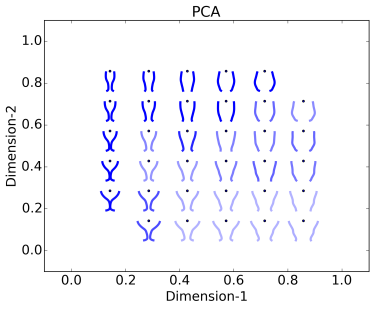


(d) Reconstruction errors and mean ONC indices

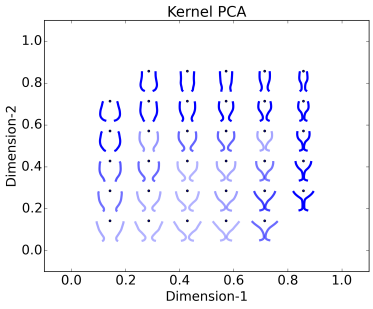


(e) ONC index

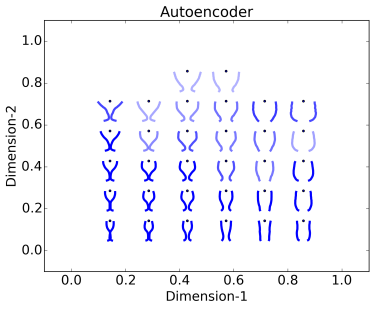
FIGURE 8: THE HIGHLY NONLINEAR SUPERFORMULA EXAMPLE (SEMANTIC SPACE DIMENSIONALITY = 2).



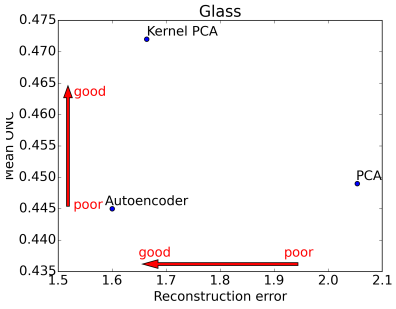
(a) Semantic space learned by PCA



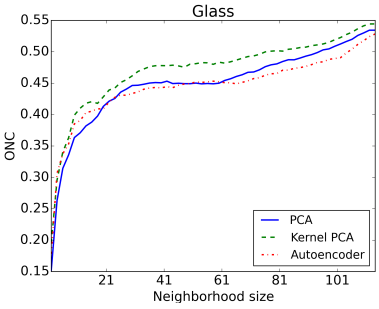
(b) Semantic space learned by Kernel PCA



(c) Semantic space learned by Autoencoder

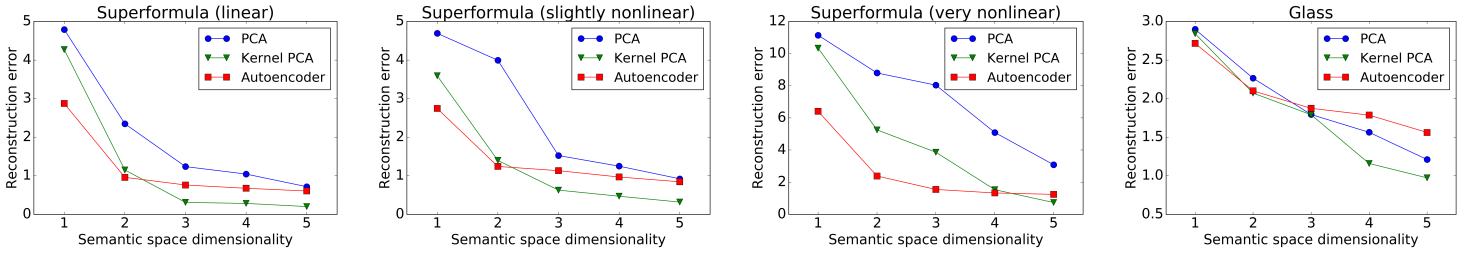


(d) Reconstruction errors and mean ONC indices



(e) ONC index

FIGURE 9: THE GLASS EXAMPLE (SEMANTIC SPACE DIMENSIONALITY = 2).



(a) Linear superformula (b) Slightly nonlinear superformula (c) Highly nonlinear superformula (d) Glass

FIGURE 10: RECONSTRUCTION ERROR VERSUS SEMANTIC SPACE DIMENSIONALITY.

achieve a drop in reconstruction error near the intrinsic dimensionality compared to PCA, which would require more semantic dimensions than necessary to achieve similar reconstruction error. For all superformula examples, we know that $d = 2$—subject to some minor fitting noise—while we do not know the intrinsic dimensionality for the glass example. Figure 10 shows the reconstruction error of testing samples versus the semantic space dimensionality for the three superformula examples and the glass example. The reconstruction errors are computed using the mean reconstruction error of two different subsets of our samples as training and test set.

Design Space Complexity

The three superformula examples progressively increase the nonlinearity of the design set. This allows us to test how embeddings perform under increasing design space complexity.

As shown in Fig. 10, since PCA is a linear transformation, it needs higher semantic space dimensionality d to achieve low reconstruction error when the nonlinearity of the superformula increases. If PCA performs well, the design space likely varies linearly, and PCA's reconstruction error should drop sharply from $d = 1$ to $d = 2$; while if the design space varies nonlinearly, PCA will require more dimensions to reconstruct all the features of the design (Fig. 10a – 10c). Kernel PCA shows similar regularity with PCA, but can achieve much lower reconstruction error when d is high enough.

This differs in the real world case. Glasses possess not only primary features (such as top/bottom radii), but also various minor features (*i.e.*, small details, curvature of the bowl and base, *etc.*). Therefore, when the semantic space dimensionality d reaches two, all embeddings can capture the primary features, resulting in low reconstruction error. As d increases, PCA and kernel PCA continue to capture minor details, decreasing the reconstruction error.

In all the examples, the autoencoder's reconstruction error stabilizes as d increases, and does not outperform kernel PCA when d is large enough. This may be due to hyperparameter selection. Compared to kernel PCA, the autoencoder has more hyperparameters to set and takes more time to train. So optimizing its hyperparameters is much harder. This may cause kernel PCA to perform better as semantic space dimensionality goes up. However, unlike autoencoders, when the semantic space dimensionality goes down to a certain point, kernel PCA can no longer capture the main features of shapes regardless of its hyperparameter setting, thus the autoencoder outperforms kernel PCA in these cases.

Design space complexity also affects new predictions far away from training samples (*i.e.*, those with low certainty). By looking at the reconstructed shapes in Fig. 6–9, we can see that sometimes our trained model can propose reasonable shapes in areas with low certainty, like in the linear superformula case (Fig. 6) and the slightly nonlinear superformula case (Fig. 7a and 7c). But sometimes embeddings propose unreasonable shapes, as in kernel PCA's slightly nonlinear (Fig. 7b) or highly nonlinear superformula case (Fig. 8). As the design space complexity increases, embeddings struggle to propose reasonable shapes far away from their training samples. Future work could explore methods to segment or model highly non-linear design spaces.

CONCLUSION

We introduced an approach to evaluate two-way mappings between a design parameter space and semantic space. We evaluated PCA, kernel PCA, and autoencoders with respect to reconstruction error and topology preservation. As expected, we found that 1) as design space complexity increased, embeddings sacrificed reconstruction error and topology preservation—though non-linear models (kPCA and autoencoders) faired much better than linear models (PCA); 2) increasing the semantic dimensionality helped demarcate the intrinsic dimensionality of the design space; and 3) autoencoders performed fairly robustly compared to PCA and kPCA, though future work could explore ways to increase both topology preservation and more even embedding of training samples throughout the semantic space.

Unexpectedly, we found that non-linear embeddings, particularly Kernel PCA, created cavities in the semantic space

that created unpredictable shapes, despite low reconstruction error and high topology preservation near training samples. This led us to the main limitation of non-linear design embeddings: the semantic space obtained by design embeddings may contain sample-sparse areas, where it cannot guarantee correct new shape reconstruction. For future work, we plan to enhance the regularity of training sample distribution in the projected semantic space. We also plan to identify and separate possible competing manifolds in highly nonlinear samples so as to possibly decrease the non-linearity of the design space.

While this paper addressed geometric design spaces, our approach would extend to any type of design embeddings, including those that reduce dimensionality of design text, or even combinations of words and geometry. It could apply to both improving interfaces that help novices explore designs as well as helping model consumer preferences in high-dimensional design spaces. Our work's main implication is that choosing a design embedding carries with it important choices about what you value in your semantic space: Should it reconstruct designs consistently? Should it preserve local or global topology? Should the semantic space maintain regularity? Choosing an embedding with the properties you want is not straightforward; our approach provides a principled way to compare and contrast embeddings—to help you navigate those options and identify useful properties of both the embedding and your design space in general. Ultimately, our work illuminates how designs differ from one another by unpacking and studying the inherent complexity of design spaces; by providing a deeper understanding of what we design and how we design.

REFERENCES

[1] Gielis, J., 2003. "A generic geometric transformation that unifies a wide range of natural and abstract shapes". *American journal of botany,* ***90***(3), pp. 333–338.

[2] Van der Maaten, L., Postma, E., and Van den Herik, H., 2009. "Dimensionality reduction: A comparative review". *Technical Report TiCC TR 2009-005*.

[3] Bengio, Y., Courville, A., and Vincent, P., 2013. "Representation learning: A review and new perspectives". *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* ***35***(8), Aug, pp. 1798–1828.

[4] MacDonald, E., Lubensky, A., Sohns, B., and Papalambros, P. Y., 2008. "Product semantics and wine portfolio optimisation". *International Journal of Product Development,* ***7***(1-2), pp. 73–98.

[5] Yumer, M. E., Chaudhuri, S., Hodgins, J. K., and Kara, L. B., 2015. "Semantic shape editing using deformation handles". *ACM Transactions on Graphics (TOG),* ***34***(4), p. 86.

[6] Mehmet Ersin Yumer, Paul Asente, R. M. L. B. K., 2015. "Procedural modeling using autoencoder networks". In Proceedings of the 28th ACM User Interface Software and Technology Symposium, UIST '15, ACM.

[7] Ulu, E., Zhang, R., and Kara, L. B., 2015. "A data-driven investigation and estimation of optimal topologies under variable loading configurations". *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, pp. 1–12.

[8] Ulu, N. G., and Kara, L. B., 2015. "Generative interface structure design for supporting existing objects". *Journal of Visual Languages and Computing,* ***31, Part B***, pp. 171–183.

[9] Jolliffe, I., 1986. *Principal Component Analysis*. Springer, Heidelberg.

[10] Ozertem, U., and Erdogmus, D., 2011. "Locally defined principal curves and surfaces". *J. Mach. Learn. Res.,* ***12***, July, pp. 1249–1286.

[11] Hastie, T., and Stuetzle, W., 1989. "Principal curves". *Journal of the American Statistical Association,* ***84***(406), pp. 502–516.

[12] Souvenir, R., and Pless, R., 2005. "Manifold clustering". In Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, Vol. 1, IEEE, pp. 648–653.

[13] Elhamifar, E., and Vidal, R., 2011. "Sparse manifold clustering and embedding". In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds. Curran Associates, Inc., pp. 55–63.

[14] Wang, M., Sha, F., and Jordan, M. I., 2010. "Unsupervised kernel dimension reduction". In *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, eds. Curran Associates, Inc., pp. 2379–2387.

[15] Gisbrecht, A., Lueks, W., Mokbel, B., and Hammer, B., 2012. "Out-of-sample kernel extensions for nonparametric dimensionality reduction". In Proceedings of European Symposium on Artificial Neural Networks, ESANN, pp. 531–536.

[16] Reed, S., Sohn, K., Zhang, Y., and Lee, H., 2014. "Learning to disentangle factors of variation with manifold interaction". In Proceedings of the 31st International Conference on Machine Learning (ICML-14), T. Jebara and E. P. Xing, eds., JMLR Workshop and Conference Proceedings, pp. 1431–1439.

[17] Lawrence, N., 2005. "Probabilistic non-linear principal component analysis with gaussian process latent variable models". *The Journal of Machine Learning Research,* ***6***, pp. 1783–1816.

[18] Damianou, A., Ek, C., Titsias, M., and Lawrence, N., 2012. "Manifold relevance determination". In Proceedings of the 29th International Conference on Machine Learning (ICML-12), J. Langford and J. Pineau, eds., ICML '12, Omnipress, pp. 145–152.

[19] Zhu, Z., Luo, P., Wang, X., and Tang, X., 2014. "Multi-view perceptron: a deep model for learning face identity and view representations". In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds. Curran Associates, Inc., pp. 217–225.

[20] Lukasik, S., and Kulczycki, P., 2013. "Using topology preservation measures for multidimensional intelligent data analysis in the reduced feature space". In *Artificial Intelligence and Soft Computing*, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, and J. Zurada, eds., Vol. 7895 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 184–193.

[21] Lee, J. A., and Verleysen, M., 2007. *Nonlinear Dimensionality Reduction*. Springer, New York.

[22] de Medeiros, C., Costa, J., and Silva, L., 2011. "A comparison of dimensionality reduction methods using topology preservation indexes". In *Intelligent Data Engineering and Automated Learning - IDEAL 2011*, H. Yin, W. Wang, and V. Rayward-Smith, eds., Vol. 6936 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, pp. 437–445.

[23] Borg, I., and Groenen, P. J. F., 2005. *Modern Multidimensional Scaling: Theory and Applications*. Springer, New York.

[24] Sammut, C., and Webb, G. I., 2010. *Encyclopedia of Machine Learning*. Springer, New York.

[25] Edelsbrunner, H., and Harer, J., 2008. "Persistent homology-a survey". *Contemporary mathematics,* ***453***, pp. 257–282.

[26] Rieck, B., and Leitte, H., 2015. "Persistent homology for the evaluation of dimensionality reduction schemes". *Computer Graphics Forum,* ***34***(3), pp. 431–440.

[27] Schölkopf, B., Smola, A., and Müller, K.-R., 1998. "Nonlinear component analysis as a kernel eigenvalue problem". *Neural computation,* ***10***(5), pp. 1299–1319.

[28] Hornik, K., Stinchcombe, M., and White, H., 1989. "Multilayer feedforward networks are universal approximators". *Neural networks,* ***2***(5), pp. 359–366.

[29] Snoek, J., Larochelle, H., and Adams, R. P., 2012. "Practical bayesian optimization of machine learning algorithms". In Advances in neural information processing systems, pp. 2951–2959.