








Scalable Visual Exploration of 3D Shape Databases via Feature Synthesis and Selection

Xingyu Chen^{1,2} , Guangping Zeng¹ , Jiří Kosinka² ,
and Alexandru Telea³  

¹ University of Science and Technology Beijing, Beijing, China

² University of Groningen, Groningen, The Netherlands

³ Utrecht University, Utrecht, The Netherlands

a.c.telea@uu.nl

Abstract. We present a set of techniques to address the problem of scalable creation of visual overview representations of large 3D shape databases based on dimensionality reduction of feature vectors extracted from shape descriptions. We address the problem of feature extraction by exploring both combinations of hand-engineered geometric features and using the latent feature vectors generated by a deep learning classification method, and discuss the comparative advantages of both approaches. Separately, we address the problem of generating insightful 2D projections of these feature vectors that are able to separate well different groups of similar shapes by two approaches. First, we create quality projections by both automatic search in the space of feature combinations and, alternatively, by leveraging human insight to improve projections by iterative feature selection. Secondly, we use deep learning to automatically construct projections from the extracted features. We show that our three variations of deep learning, which jointly treat feature extraction, selection, and projection, allow efficient creation of high-quality visual overviews of large shape collections, require minimal user intervention, and are easy to implement. We demonstrate our approach on several real-world 3D shape databases.

Keywords: Content-based shape retrieval · Multidimensional projections · Feature selection · Deep learning · Visual analytics

1 Introduction

Recent advances in modeling, authoring, and scanning tools for 3D data have led to wealth of 3D models available to their interested users. As a consequence, this has led to the creation and deployment of specialized *shape databases* [9, 26] for managing the available content. A key challenge of these databases is to allow users to easily browse and search them to find models of interest.

As such shape databases grow in size and variability of the stored shapes, so does the users' effort required to explore them [33]. Typical techniques that support exploration include keyword based search, browsing the database along predefined hierarchies or taxonomies, and content-based shape retrieval (CBSR).

While certainly useful, all these techniques have limitations: Keyword search assumes that shapes are labeled with relevant keywords, and that users are familiar with these keywords. Hierarchy browsing is most effective when it matches the user’s mental model of how shapes are organized. Finally, CBSR works well when one wants to query for shapes similar to an example that one already avails of.

A particularly important use case which is not well covered by the above mechanisms involves users who want to first get a good overview of what a database contains. This helps understanding whether the database contains shapes of interest to the user – in which case, the user may decide to select a relevant subset thereof to explore in more detail via classical search mechanisms. Keyword, hierarchy, and CBSR techniques are not optimal for the overview task: They either show a small part of the database at a given time and/or ask the user to perform lengthy navigations to create a mental map of the database itself, much like when one navigates a web domain.

To address the overview task, our previous work [4] constructs a visual depiction of a full shape database, with shapes organized by similarity, using a dimensionality-reduction (DR) technique. This method offers *details-on-demand* mechanisms to enable users to control the separation quality of the similar-shape groups in the visual overview, understand what makes selected shapes similar (or different), and find features that have high, respectively little, value for creating the overview. This approach is simple to use, requires no prior knowledge of the organization of a shape database, nor does it require shapes to be labeled. The proposed visualization targets both end users (who aim to explore a shape database) and technical users (who aim to engineer features to create such overviews, or, further, to query or classify shapes in such databases).

However although visually effective, the above approach has a few limitations: (1) The hand-engineered features it uses may not always best capture shape similarity, and also are delicate to compute for poor-quality (non-watertight, self-intersecting, and/or variable-resolution meshes); (2) The feature extraction and DR steps are computationally quite slow, and cannot scale to real-world databases containing tens of thousands of shapes or more; (3) The underlying DR technique used is non-deterministic, meaning that overviews created from the same database (let alone databases where a few shapes change) will be different, which makes it hard for users to maintain their mental map.

In this paper, we improve the approach of [4] with the following contributions to all the above-mentioned limitations:

- We extract features from shapes using deep learning, thereby better capturing the shape’s similarities (1);
- We use deep learning for both feature extraction and DR, thereby being able to handle large shape databases efficiently (2);
- We use a deterministic DR projection, thereby ensuring consistent creation of the visual overviews even when the shape database changes (3).

This paper is structured as follows. Section 2 outlines related work in exploring 3D shape databases. Section 3 details our first proposed pipeline, based on hand engineered features which can be interactively selected to construct custom overviews. Section 4 presents results of this pipeline. Section 5 presents our second proposed pipeline, which uses deep learning for jointly addressing the

problems of feature extraction and dimensionality reduction, and illustrates the advantages of this approach. Section 6 discusses our overall proposal. Finally, Sect. 7 concludes the paper.

2 Related Work

Several mechanisms for searching and exploring 3D shape databases exist. Most such databases often provide only a subset of them rather than supporting them all. We describe below the most frequently-met such mechanisms.

Keyword search allows users to search for shapes with annotation labels by text words. It is the most popular method for searching for digital data – that is, not only 3D content, but also images and other multimedia types. As such, it is a familiar tool for most users. It is also simple to provide, therefore many 3D databases, such as Aim@Shape [1] and TurboSquid [37] support searching shapes by keywords. On the upload side, shapes can be saved in the database with associated keywords – either picked from a predefined ontology or freely provided by users – to be next searched. Yet, keyword search is not always accurate. One reason is that keyword lists are defined by humans. In many databases, uploaders make the keyword lists of shapes long and/or redundant to increase exposure rate of their favorite content. This mechanism works better for specialized databases, such as [16], which contains only shapes related to space exploration. In this case, keywords are restricted to a predefined dictionary of specialized terms which is easier to use when uploading and/or searching. Another approach to handle keyword search is to allow users to type in search terms freely, and then process these to extract semantic vectors which are used as the actual search keywords. This is the technology underlying several of Google’s search mechanisms. While this allows users to search without being aware of the actual ontology used to organize shapes (or other searchable content), topic extraction is typically non-transparent for users and can cause both false positives and false negatives during search, as anyone using text-base search techniques is aware of.

Overall, keyword search is widespread and works well for users familiar with a database’s organization (in particular, the ontology defining associated keywords), but also needs large manual effort and is less effective for overall exploration.

Hierarchical exploration systems use a predefined taxonomy of 3D shapes to organize databases. Taxonomies typically come in the form of hierarchies of shape types and included subtypes, which allow the user to directly explore both the shape collection and associated organization of shape types. Such systems often use thumbnail galleries to allow one to explore the database. Users can browse the hierarchy of such databases just like browsing a computer file system. Many databases such as the Princeton Shape Benchmark [29], Aim@Shape [1], and the ITI 3D search engine [9] support hierarchy browsing. However, hierarchical exploration has several limitations. First, it typically only allows one to explore a single *path* in the hierarchy at a given time, and hence is less suited for providing a global overview of a full database. Secondly, unlike keyword indexing – where a shape could be indexed by, and thus searched via, multiple keywords – hierarchical exploration typically only allows using a single such hierarchy

at a time. Indeed, it would be confusing and complex to allow one to visually browse multiple interlinked hierarchies. As such, the effectiveness of this method is linked to how well the provided hierarchy matches the user’s mental map of the shape universe under exploration.

Content based shape retrieval (CBSR) frees the user from the task of specifying keywords or navigating along the constraints of a predefined (hierarchical) taxonomy. Rather, users specify *content* directly, either in terms of a query shape or a shape proxy, such as a simplified version of the actual shape to search for, for example, a 2D sketch thereof. Next, shapes in the database are ranked based on a computed similarity function to the query, and the best matching ones are returned to the user. Unlike keyword search and hierarchical exploration, CBSR frees users from having to know how shapes and/or their keywords are organized in the database. Many CBSR methods are proposed by previous research [3, 33]. At a high level, all these methods extract a high-dimensional feature vector from both the shapes in the database and the query, and then use a suitable distance metric in the descriptor space to find the most similar shapes to the query. Many methods to extract such shape descriptors exist and each has its own advantages. *Geometric* descriptors aim to capture the actual geometry or form of the shape. They can be divided into global descriptors, such as shape elongation, eccentricity, and compactness; and local descriptors, such as saliency, curvature, shape contexts, and shape thickness. Global descriptors are easy and fast to compute, but cannot separate shapes which differ only at a local levels. As such, they are typically used in a pre-filtering phase to accelerate the search. Local descriptors [24, 27, 31, 34] capture more fine-grained, local, aspects of the shape, and are used to refine the search process. Topological descriptors, such as using 3D curve skeletons [11] or 3D surface skeletons [8] capture the part-whole structure of a shape, and are particularly good for queries which require pose invariance. Finally, view-based descriptors [5, 28] describe shapes based on views thereof taken from multiple viewpoints. As such, they do not require shapes to be described by high-quality meshes (which are often required for computing the other aforementioned descriptors). Moreover, view-based descriptors can also integrate additional properties such as color, reflectance, and texture, when these are deemed interesting to drive the search. Descriptors of several of the above types can be combined in the search process [12]. Apart from such hand-engineered descriptors, deep learning solutions have shown to be very effective in all tasks related to processing shape databases by means of extracted feature vectors [22, 32]. These include several tasks such as shape classification (which can be used standalone or as a help in designing labels for taxonomy creation) and also shape database querying. One salient method in this area is PointNet [22] which was used to obtain highly accurate classifications of complex 3D shapes represented as point clouds. We discuss this method, and its adaption to our goals, further in Sect. 5.

As already mentioned, CBSR makes searching easier than keyword search or hierarchical exploration. However, CBSR typically can only be used if one already avails of a query instance to search for. As such, CBSR is not designed to support the more general task of exploration of a shape database. Indeed, while CBSR shows which database shapes are similar to a *given* query, exploration should show how *all* database shapes are similar to *each other*.

Concluding the above, keyword search, hierarchical exploration, and CBSR are complementary tools for exploring a shape database, and optimize each for different tasks and use cases. They can be easily combined in a 3D database exploration system. However, even when combined, none of these methods does offer a compact and detailed overview of an entire database showing how all its shapes relate to each other in terms of similarity. Such an overview functionality is essential in contexts where one does not know what to search for, as it helps ‘bootstrapping’ the search by telling the user upfront what is in the database and how things are organized. Separately, we note that the above mechanisms do not aim to explain *why* a set of shapes – *e.g.*, the ones returned by a query – are found similar. Such explanations are useful for shape-database engineers who want to understand how the search process works to *e.g.* fine tune its underlying feature extraction and/or feature comparison (similarity function) to optimize its accuracy or computational speed. This has been shown by Rauber *et al.* [23] in the context of classification of 2D image databases. Our work in Sect. 4.3 uses similar interactive feature selection mechanisms as Rauber *et al.*. However, as we will discuss there, our goal is entirely different – that is, the creation of customized overviews of 3D shape databases rather than the optimization of an image classifier’s accuracy.

3 Feature Selection Method

We detail our first approach to creating visual overviews of shape databases, which uses hand engineered features which can be interactively selected by users to create custom overviews. For simplicity of reference, we call this approach the *feature selection method* next.

Let us introduce a few notations. A shape is described as a mesh $m = (V = \{\mathbf{x}_i\}, F = \{f_j\})$, *i.e.*, a collection of vertices $\mathbf{x}_i \subset \mathbb{R}^3$ and triangular faces f_j . Hence, a shape database is a set of meshes $M = \{m_k\}$. Shapes can be of different kinds, sampling resolutions, and require no extra organization or annotations, *e.g.*, class or hierarchy labels or keywords.

We aim to create a visual *overview* of M in which every shape m_k is depicted by a thumbnail rendering thereof. In this overview, similar shapes should be placed close to each other. Detail views can be invoked interactively via the thumbnails to show specific shape details. This combination of overview and details follows Shneiderman’s visual exploration mantra [30] to enable both free and targeted exploration of the shape database along the use-cases outlined in Sect. 2.

We create our visual overview follows. First, we preprocess all meshes in M to normalize their sampling resolution and size (Sect. 3.1). Secondly, we extract local features from each mesh, thereby capturing the geometry of the respective shapes (Sect. 3.2). Next, we extract a fixed-length feature vector from each shape by aggregating the above-mentioned local features (Sect. 3.3). Finally, we use dimensionality-reduction to project the shapes, encoded by their feature vectors, to create a 2D scatterplot (Sect. 3.4). We describe all these steps next.

3.1 Preprocessing

Shapes in a database M can, in general, have any sampling resolution, orientation (pose), and scale. Such variations are typically irrelevant for shape similarity and also induce unwanted variability when computing shape descriptors [3]. To alleviate this, we first remesh all shapes in M to a target edge-length of 1% of m 's bounding-box diagonal. Next, we translate and uniformly scale the remeshed shapes to tightly fit in the $[-1, 1]^3$ cube.

3.2 Local Feature Computation

We characterize shapes by several so-called *local features*. These features describe the shape in the neighborhood of every vertex $\mathbf{x}_i \in m$ and are hence good at capturing local geometry characteristics. We compute seven local features, as follows.

Gaussian Curvature (Gc): Gaussian curvature describes the overall deviation from flatness of a shape at a given point. We compute the Gaussian curvature of every vertex $\mathbf{x} \in m$ as

$$Gc(\mathbf{x}) = 2\pi - \sum_{f \in F(\mathbf{x})} \theta_{\mathbf{x},f}, \quad (1)$$

where $F(\mathbf{x})$ is the set of faces in F containing \mathbf{x} and $\theta_{\mathbf{x},f}$ is the angle of the two edges of f that contain \mathbf{x} . A perfectly flat triangle-fan around \mathbf{x} will have $Gc(\mathbf{x}) = 0$. Conversely, an infinitely sharp spike on m at \mathbf{x} will have $Gc(\mathbf{x}) = 2\pi$.

Average Geodesic Distance (Agd): Let $d(\mathbf{x}, \mathbf{y})$ be the length of the geodesic curve located on the surface of m between a pair of vertices \mathbf{x} and \mathbf{y} of m . Given this, we estimate the average geodesic distance of a vertex \mathbf{x} as

$$Agd(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in V} d(\mathbf{x}, \mathbf{y})}{|V|}. \quad (2)$$

Intuitively, Agd tells how close to a ‘tip’ or protrusion (or its concave equivalent) a vertex \mathbf{x} is. For example, for a hand model, points on the finger tips will have high Agd values, whereas points on the palm will have lower Agd values. Agd can thus discriminate shapes which have many protrusions (thus, high variations of Agd over their points), like the hand model, from overall rounder shapes (thus, very small variations of Agd over their points), like a ball.

We approximate the geodesic distance $d(\mathbf{x}, \mathbf{y})$ as the geometric length of the shortest path in the edge connectivity graph of m between \mathbf{x} and \mathbf{y} . This length can be easily and efficiently estimated using Dijkstra’s shortest-path algorithm with A* heuristics and edge weights equal to edge lengths. More accurate estimations of the geodesic distance between two points on a polygonal mesh exist, including computing the distance transform $DT(\mathbf{x})$ of \mathbf{x} over F and tracing a streamline in $-\nabla DT(\mathbf{x})$ from \mathbf{x} until it reaches \mathbf{y} [20]; minimization of the length of a cut created by a rotating slice plane passing through \mathbf{x} and \mathbf{y} [10]; or hybrid search techniques [38]. While more accurate than the Dijkstra approach we use, these methods are *considerably* more complex to implement, slower to

run (except the GPU-based method in [10]), and require careful tuning and/or specialized platforms (GPU support). For a detailed comparison of geodesic estimation methods on polygonal meshes, we refer to [10]. In our case, the added value of accurate geodesic computation is not needed. Indeed, as discussed next in Sect. 3.3, we only use aggregates (histograms) of all Agd values computed over an entire mesh. Hence, less accurate, but fast and easy to compute Dijkstra length estimation is sufficient for our purposes.

Normal Diameter (Nd): This descriptor aims to capture the local thickness of a shape at a given vertex. For this, we first estimate the surface normal at a vertex \mathbf{x} as

$$\mathbf{n}(\mathbf{x}) = \frac{\sum_{f \in F(\mathbf{x})} \mathbf{n}(f) \theta_{\mathbf{x},f}}{2\pi}, \quad (3)$$

where $\mathbf{n}(f)$ is the outward normal of face f . Let \mathbf{r} be a ray starting at \mathbf{x} and advancing in the direction $-\mathbf{n}(\mathbf{x})$. The normal diameter $Nd(\mathbf{x})$ is then the distance along \mathbf{r} from \mathbf{x} to the face $f \in F \setminus F(\mathbf{x})$ that \mathbf{r} intersects. Note that, besides this ray tracing approach, local shape thickness can be computed by other methods, such as medial surfaces [35] or view-based approaches [25]. However, these approaches are not suitable in our context since they require voxel models rather than meshes [35] or require a complex and expensive view-based computation pipeline [25]. As for the Agd estimation, our ray-based Nd estimation is arguably less accurate than alternative approaches but, since ultimately aggregated via histograms, strikes a good balance between quality and computation ease and speed.

Normal Angle (Na) and Point Angle (Pa): These features describe how vertices $\mathbf{x} \in V$ are distributed over the shape’s surface. Let \mathbf{e}_1 be the major eigenvector of the shape covariance matrix given by all vertices V . As known, \mathbf{e}_1 gives the direction in which V spreads the most. For every vertex $\mathbf{x} \in V$, we define the normal angle $Na(\mathbf{x})$ as the dot product between \mathbf{e}_1 and the surface normal $\mathbf{n}(\mathbf{x})$; and the point angle $Pa(\mathbf{x})$ as the dot product between \mathbf{e}_1 and $\mathbf{c} - \mathbf{x}$, where \mathbf{c} is the barycenter of m , respectively.

Shape Context (Sc): The shape context descriptor [2] is a 2D histogram that describes how distances and orientations of all vertices in V to a fixed, given, vertex $\mathbf{x} \in V$, vary. To compute Sc , we first build a local coordinate system at every vertex $\mathbf{x} \in V$, using the eigenvectors of the shape covariance matrix in the neighborhood of \mathbf{x} . This aligns the local coordinate system with the shape, making one of its axes coincide with the normal $\mathbf{n}(\mathbf{x})$ and the two other axes tangent to the surface of m at \mathbf{x} . Next, we discretize the orientations around \mathbf{x} into the eight octants of the local coordinate system. We also discretize distances using a set of distance ranges (t_i, t_{i+1}) defined by a distance-set $T = \{0, t_1, t_2, \dots, t_n, 1\}, n \in \mathbb{N}_+$. In practice, we use $T = [0, 0.1, 0.3, 1]$, given that our shapes are normalized in $[0, 1]^3$. Hence, for each vertex \mathbf{x} , $Sc(\mathbf{x})$ is a vector with $8 \times 3 = 24$ elements.

Point Feature Histogram (PFH): PFH [24] is a complex descriptor that captures the local shape geometry in the neighborhood of a vertex. Given a pair of vertices \mathbf{y} and \mathbf{y}' , where \mathbf{y}' is a neighbor of \mathbf{y} , we first define a local coordinate frame $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ as

$$\mathbf{u} = \mathbf{m}, \quad \mathbf{v} = (\mathbf{y}' - \mathbf{y}) \times \mathbf{u}, \quad \mathbf{w} = \mathbf{u} \times \mathbf{v}, \quad (4)$$

where \mathbf{m} is the vertex normal at \mathbf{y} . Next, the variation of the shape geometry between \mathbf{y} and \mathbf{y}' is measured by three polar coordinates

$$\alpha = \mathbf{v} \cdot \mathbf{m}', \quad \phi = \mathbf{u} \cdot \frac{\mathbf{y}' - \mathbf{y}}{\|\mathbf{y}' - \mathbf{y}\|}, \quad \theta = \arctan2(\mathbf{w} \cdot \mathbf{m}', \mathbf{u} \cdot \mathbf{m}'), \quad (5)$$

where \mathbf{m}' is the vertex normal at \mathbf{y}' . We build three histograms to capture the distributions of α, ϕ, θ for a given vertex \mathbf{x} by considering all pairs $(\mathbf{y}, \mathbf{y}') \in N_{\mathbf{x},k} \times N_{\mathbf{x},k}$ in the k -nearest neighbors $N_{\mathbf{x},k}$ of \mathbf{x} . In practice, we set $k = 30$ and use 5 bins for each histogram. This delivers, for each vertex \mathbf{x} , a PFH feature vector of $5^3 = 125$ entries.

Fast Point Feature Histogram (FPFH): While PFH models a neighborhood $N_{\mathbf{x},k}$ by all its point-pairs, the Simplified Point Feature Histogram (SPFH) models $N_{\mathbf{x},k}$ by the pairs $(\mathbf{x}, \mathbf{y}) | \mathbf{y} \in N_{\mathbf{x},k}$. That is, PFH considers k^2 pairs, whereas SPFH considers only k pairs. To compute FPFH, we proceed analogously to binning the α, ϕ, θ distributions (Eq. 5) in three 11-bin histograms, obtaining a feature vector of $3 \times 11 = 33$ elements. With this vector, we finally compute the FPFH value of a vertex \mathbf{x} following [24] as the distance-weighted average of the SPFH values over $N_{\mathbf{x},k}$ as

$$FPFH(\mathbf{x}) = SPFH(\mathbf{x}) + \frac{1}{k} \sum_{\mathbf{y} \in N_{\mathbf{x},k}} \frac{SPFH(\mathbf{y})}{\|\mathbf{x} - \mathbf{y}\|}. \quad (6)$$

3.3 Feature Vector Computation

The eight features introduced in Sect. 3.2 take different values for every mesh vertex $\mathbf{x} \in V$, as they indeed aim to capture the shape characteristics close to \mathbf{x} . To compare entire meshes to each other, we need to abstract from these local descriptors. We do this by computing, from all local descriptors of a shape, a single global descriptor, or feature vector, which (1) has the same length for all shapes, regardless of their vertex count, and (2) is invariant on the ordering (numbering) of vertices in the mesh. For this, we aggregate the values of every local descriptor, at all vertices of a mesh, into a fixed-length (10 bin) histogram. Note that some descriptors are by definition high-dimensional—for instance, the shape context Sc has $d = 24$ dimensions. For such d -dimensional descriptors, we compute a histogram having $10d$ bins, thus using 10 bins per dimension. Table 1 shows the local features, their dimensionality, and the number of bins used to quantize each. Summarizing, we reduce every shape m to a 1870-dimensional feature vector \mathcal{F} .

3.4 Dimensionality Reduction

The feature extraction process described so far essentially reduces a shape database M to a set of $|M|$ 1870-dimensional feature vectors. We next achieve our goal of creating a visual overview of the database by projecting these vectors in 2D using the well-known t-SNE dimensionality reduction technique [13].

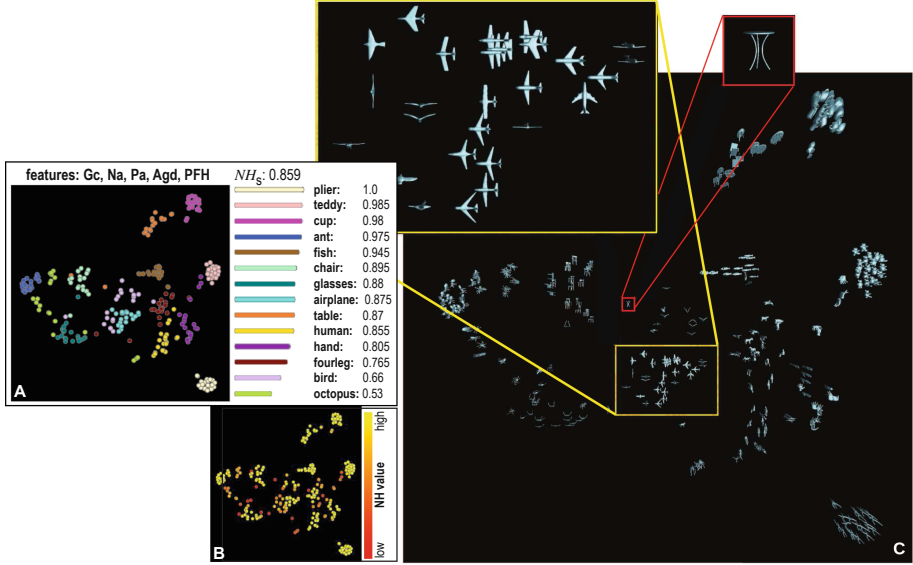


Fig. 1. Three views of the optimal projection scatterplot for the Princeton Shape Database, depicting classes and their NH_c values and the overall plot quality NH_s (A), per-shape NH values (B), and actual shape thumbnails (C). Figure taken from [4].

That is, t-SNE constructs a scatterplot $P(M) = \{P(m_k)\}$, where every shape $m_k \in M$ is mapped to a point $P(m) \in \mathbb{R}^2$, so that the distances between scatterplot points reflect the similarities of their feature vectors. Next, for visual clarity, we render these points $P(m)$ to show thumbnails, or other data attributes, of their respective shapes m .

As outlined in Sect. 1, we aim to use this visual representation to explore the database M . Hence, $P(M)$ should accurately reflect the similarities computed via feature vectors. Many metrics exist that compute the quality of projections – for a detailed overview, we refer to [17]. However, most of these are not applicable

Table 1. Local features, their dimensionalities, and their binning. Table taken from [4].

Name	Dimensionality	Bins
Gaussian curvature (Gc)	1	10
Average geodesic distance (Agd)	1	10
Normal diameter (Nd)	1	10
Normal angle (Na)	1	10
Point angle (Pa)	1	10
Shape context (Sc)	24	240
Point Feature Histogram (PFH)	125	1250
Fast Point Feature Histogram (FPFH)	33	330
Total		1870

to our context, since they assume the feature vector data as ground truth, *i.e.*, correct and accurate. In our case, we actually extract such data from the actual 3D meshes. Hence, we address projection quality computation by using class (label) information from the shapes, as follows. We assume that each class m has a categorical label $c(m) \in C$, where C is a set of categories (*e.g.*, keywords describing the different shapes in a database). Next, we define the neighborhood hit $NH(m)$ as the proportion of the k -nearest neighbors of $P(m)$ that have the same label $c(m)$ as m itself [19]. In practice, we set $k = 10$, following related applications that gauge projection quality [19]. With this, we can next define the neighborhood hit of an entire class $c \in C$ as

$$NH_c(c) = \frac{\sum_{m \in M: c(m)=c} NH(m)}{|m \in M : c(m) = c|}. \quad (7)$$

Finally, at the highest aggregation level, we define the neighborhood hit for an entire scatterplot $P(M)$ for a shape database M as

$$NH_s(M) = \frac{\sum_{m \in M} NH(m)}{|M|}. \quad (8)$$

The intuition behind the above metrics is as follows. $NH(m)$ describes how uniform the projection is, in terms of class labels, around the projection of mesh m . We do not use this metric directly in our evaluation, but only as a means to define the more aggregated NH_c and NH_s metrics. NH_c shows whether a group of points (in the projection) representing same-class meshes is well separated from point groups representing meshes of different classes. This is desirable, since we want next to use the scatterplot to answer questions like ‘‘How many shape classes are in a database, and how similar are they to each other?’’. Finally, NH_s shows how well a whole scatterplot can represent an entire shape database, and is thus a simple metric to use to compare the quality of different scatterplots. Both NH_c and NH_s range between 0 and 1, with higher values indicating better class separation, which is preferred. Note, importantly, that we do not use the class label information for anything related to the *construction* of the visual overview – as mentioned earlier, we can construct such projections using only unlabeled data. We only use class labels as a proxy to gauge the *quality* of the projection for typical tasks involving reasoning about the different types of shapes in a database.

4 Applications

We next illustrate our visual exploration on a subset of the Princeton Shape Benchmark [29] (280 meshes from 14 shape classes, 20 meshes from each class).

4.1 Optimal Scatterplot Creation

As mentioned in Sect. 3.4), creating a high-quality scatterplot is important for all exploration tasks it addresses next. To gauge this, we answer next the following questions:

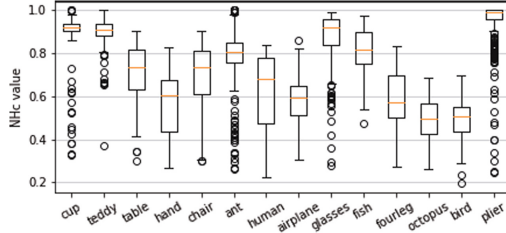


Fig. 2. NH_c statistics for the 14 classes in the shape database for all 255 projection scatterplots. Figure taken from [4].

- Q1: How can we create a good projection?
 Q2: Which features are best for grouping similar shapes (and separating different shapes) in the projection?
 Q3: Which is the minimal set of features needed to generate a good-quality projection?

The easiest way to proceed would be to create a t-SNE projection using the full 1870-dimensional feature vectors we extracted (Sect. 3.3). However, we do not know that all our features effectively capture shape similarity well; some may be redundant, thus only add computational complexity with no extra value; others may be even confusing, that is, decrease the projection quality. Moreover, using high-dimensional feature vectors makes the t-SNE projection task harder [39]. Hence, we first explore the idea of projecting *subsets* of the 1870 feature vector. We have 8 feature types (Table 1), so one idea would be to create all $2^8 - 1 = 255$ possible projections using combinations of these 8 feature types. We compute all these projection and next select the one having the highest NH_s quality. Note that this is related to the well-known scagnostics principle [36, 40] of generating a superset of all possible scatterplots from a multivariate dataset and next select for inspection the ones which show interesting details. In our case, however, we do the selection based on the projection quality NH_s .

Figure 1 shows three views of the optimal projection scatterplot. Image A shows the scatterplot with points (shapes $m \in M$) colored by their class value $c(m)$. The title above this image shows the feature subset used for this optimal scatterplot (highest $NH_s = 0.859$ value), namely (Gc, Na, Pa, Agd, PFH). The bar chart in image A shows the NH_c values for all classes, with high values (well separated classes) at the top. We see that *pliers* are perfectly separated from all other classes ($NH_{pliers} = 1$), while *octopus* is least well separated ($NH_{octopus} = 0.53$). Image B shows the optimal scatterplot colored by NH values for all shapes, ranging between red (low NH) to yellow (high NH). Red points show shapes which are not projected well—that is, placed close to different-class shapes. We see that there are such points in a variety of classes. Finally, image C shows the optimal scatterplot with shapes depicted by thumbnails. We see here, better than in image A, that pliers, teddies, cups, ants, and fishes are well projected; but birds are mixed with airplanes, and fourlegs are mixed with humans and hands. The octopus class is visually split in the projection into several parts. While this optimal scatterplot is not perfect, it is the best one we can create by

combinations of our 8 available features. While class separation is not perfect, closely-projected shapes are still similar. For example, ants are surrounded by octopuses, which is arguably logical, since both shape types have many thin and spread legs, *i.e.*, a high variability of *Agd*. Similarly, airplanes and birds are close to each other; both have wings and are quite flat.

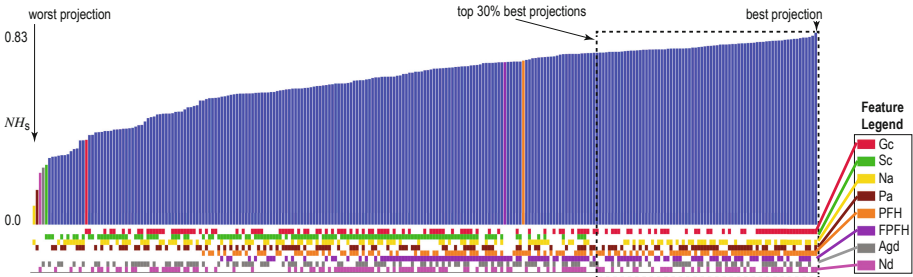


Fig. 3. Bar chart showing the NH_s scores of 255 projections, sorted on increasing value (best projections to the right, worst ones to the left). The color blocks under a bar show which features are used for that projection (the feature color legend on the right). Bars which are not blue only use one feature, whose identity colors the bar. Scanning the color matrix below the bars row-wise tells us which projections use which features. We see that PFH (orange) and FPFH (purple) are good features since their blocks are close to the right. Conversely, Sc (green) is not a very useful feature since its blocks are spread to the left. Figure taken from [4]. (Color figure online)

Besides seeing the optimal projection (Fig. 1), we also want to understand how far are all other 254 projections from this optimum. Figure 2 shows this by whisker plots of the NH_c values for all 255 projections, grouped per class. We see here that some classes (cup, plier) have a low NH_c variance around a very high value. Hence, optimal projection selection is not that relevant for these classes – they would be well separated in virtually any projection computed by any feature combination. However, for other classes (ant, hand), the variance is larger. Hence, computing an optimal projection is important if we want to separate these classes well from the others. We also see that birds and octopuses have quite low NH_c values. This strengthens the insights obtained from Fig. 1, telling that it is hard to separate these classes in any projection.

To address Q2 and Q3, Fig. 3 shows how features affect the quality of all produced projections. Each bar represents one of the 255 projections, with the bar length encoding that projection's NH_s value. Bars are sorted on this value left to right, so best projections are shown right. The matrix plot under the bar chart shows which features (color coded as in the legend at the right) are used in each projection. We also highlight this in the top bar chart: Projections using more than one feature have blue bars; projections that use only one feature have bars colored by that feature. This figure tells us several stories: (1) The difference between the best and worst projections is significant (NH_s 0.831 vs 0.38). (2) Some features, *e.g.* PFH (orange) and FPFH (purple), are crucial for high quality, since they appear frequently to the right of the matrix plot; other

features actually decrease quality, *e.g.* Sc (green) which appears only in the left half of the matrix plot. (3) The right of the matrix is denser than its left part, *i.e.*, using more features yields better projections, although the relation is not monotonic. (4) The highest-quality projections (roughly, right third of the bar chart) consistently use the same feature mix (Gc , Na , Pa , PFH , $FPFH$, Agd , Nd). (5) Different features have different patterns in the matrix plot, meaning there are no redundant features in the considered feature set.

Algorithm 1. Computing near-optimal feature sets.

Require: Set of features \mathcal{F} ; maximal size s , $1 \leq s \leq |\mathcal{F}|$, of feature-set to search for,
Ensure: Near-optimal feature set \mathcal{C} ,
1: $\mathcal{C} := \emptyset, \mathcal{C}_{new} := \emptyset$;
2: **repeat**
3: $\mathcal{C} := \mathcal{C}_{new}$
4: **for each** $\mathcal{F}_{sub} \subseteq \mathcal{F}, |\mathcal{F}_{sub}| \leq s$ **do**
5: $\mathcal{C}_{temp} := (\mathcal{C} \cup \mathcal{F}_{sub}) - (\mathcal{C} \cap \mathcal{F}_{sub})$
6: **if** $NH_s(\mathcal{C}_{temp}) > NH_s(\mathcal{C}_{new})$ **then**
7: $\mathcal{C}_{new} := \mathcal{C}_{temp}$;
8: **end if**
9: **end for**
10: **until** $(\mathcal{C}_{new} = \mathcal{C})$;
11: **return** \mathcal{C} ;

4.2 Fast Computation of Near-optimal Projection Scatterplot

When the feature set \mathcal{F} is large, computing all possible projections to select the optimal one (Sect. 4.1) is expensive. We accelerate this by a greedy algorithm (Algorithm 1). The parameter s gives the maximum size of the feature-set to search for. For every search iteration, $\binom{s}{|\mathcal{F}|}$ feature combinations are examined, retaining the one yielding the highest NH_s value. Better solutions in terms of NH_s are obtained for larger s values, at the expense of higher search times. In the limit, when $s = |\mathcal{F}|$, Algorithm 1 compares all possible $2^{|\mathcal{F}|}$ feature combinations. From our tests, good NH_s values can be obtained by setting $s = 1$. For this setting, the time complexity of our algorithm is $O(|\mathcal{F}|^2)$.

Table 2 shows timing results of our search algorithm, executed 5 times, to account for t-SNE’s stochastic nature. For every round, we show the time taken by exhaustive search *vs* our greedy search, and also the number of t-SNE projections being evaluated. We see that our greedy search yields practically the same NH_s as exhaustive search, but is roughly 5 times faster.

4.3 User-Driven Projection Engineering

Section 4.2 showed how to automatically compute a good-quality projection (that separates different-class shapes well) by automatic feature selection. We saw that, even when testing all possible 8-feature combinations, we cannot obtain an ideal projection – some classes are easier to separate than others (see also Fig. 2). This is not surprising, given that our automatic search selects, or discards, all features of the same *type*, *e.g.*, the 24 shape-context Sc features are either all used, or all ignored, when constructing the projection. We next aim to address the creation of a good projection differently. The key observation here is that

Table 2. Performance of the greedy algorithm for near-optimal projection construction. Table taken from [4].

Round	Search method	NH_s	Time (secs)	t-SNE runs
1	Exhaustive	0.831	459.74	255
	Greedy	0.831	103.49	56
2	Exhaustive	0.830	452.12	255
	Greedy	0.830	84.98	48
3	Exhaustive	0.829	453.70	255
	Greedy	0.820	70.24	40
4	Exhaustive	0.832	445.47	255
	Greedy	0.832	111.71	64
5	Exhaustive	0.824	447.66	255
	Greedy	0.824	97.39	55

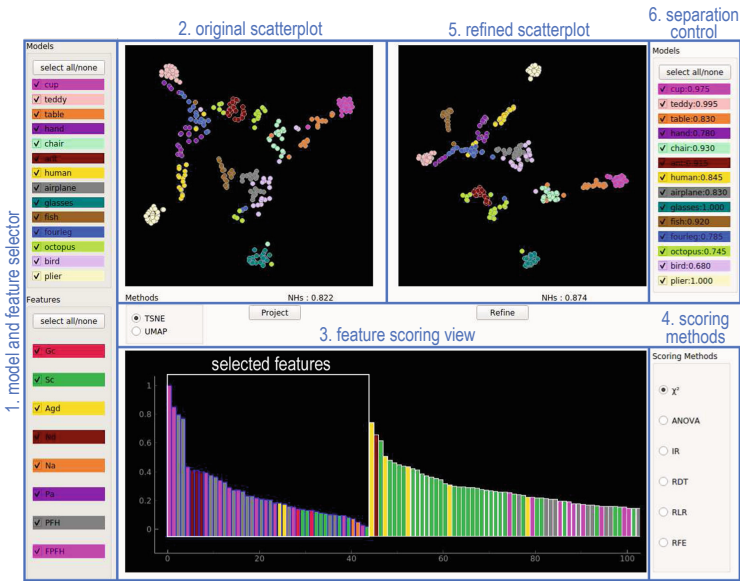


Fig. 4. User-driven projection engineering tool and its six views (Sect. 4.3). Figure taken from [4].

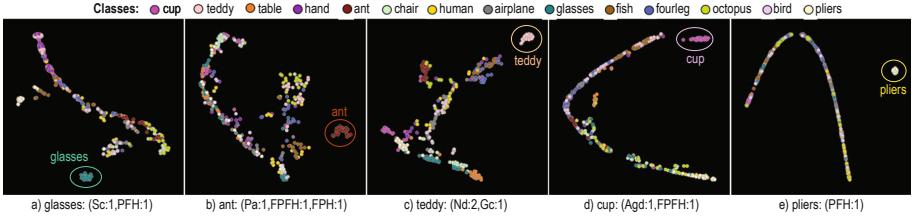


Fig. 5. Finding the minimal number of feature-bins able to separate five shape classes from the rest of the database. Notation *name:i* indicates that *i* bins of feature *name* are used. Figure taken from [4].

there are cases when one wants to optimize for separation of certain classes, depending on use-case specifics. Hence, *user input* in deciding which feature combination leads to a good projection is crucial.

We thus rephrase question Q2 as: How can we pick ‘good’ feature-bins (from the total set of 1870 bins) that separate classes in the way we desire *in a specific context*? To do this, we propose an interactive tool based on feature scoring (Fig. 4) which contains several views (1–6) that allow one to explore how features drive separation of shape classes and also select feature subsets to lead to a customized projection. These views support an overview-and-details-on-demand workflow, as follows:

Model and Feature Selector (1): The user starts by selecting the shape *classes* and feature *types* of interest in this view. If users are interested only in a few classes, these can be selected here; if one wants to separate equally well all

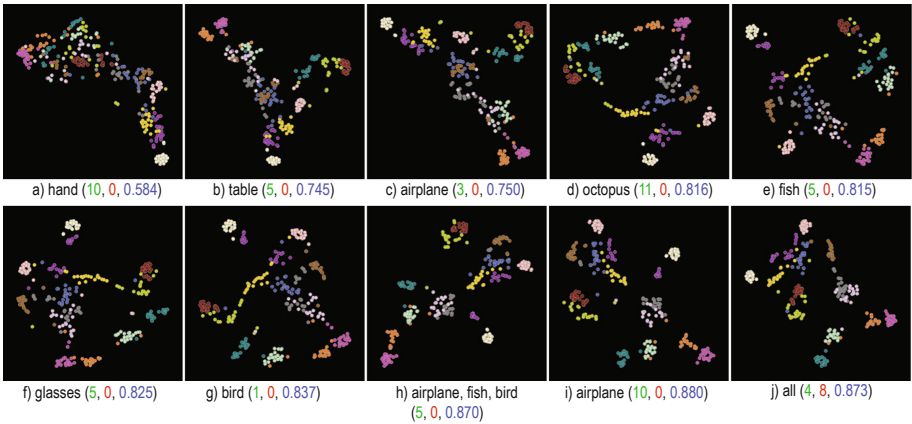


Fig. 6. Incremental creation of high-quality projection scatterplot that separates all classes well. In each step, a few feature-bins (having high scores, count indicated in green) are selected to separate one or several classes from the rest, and a few feature-bins (having low scores, count indicated in red) are removed from the selection. NH_s at each step are rendered blue. Figure taken from [4]. (Color figure online)

classes, then all should be selected. For instance, from our earlier experiments (Sect. 4.1), we saw that birds are hard to separate from airplanes. The user can then select only these two classes in view (1) to explore how to increase their separation. One can also select feature types (from the 8 computed ones) to use for creating the projection. This helps examining, or debugging, the effect of each feature type. Classes are categorically color-coded, and the same colors are used in the scatterplots (2, 5). Feature types are also categorically color-coded with the same colors used in the feature scoring view (3).

Original Scatterplot (2): This view shows a scatterplot using all shape classes *vs* all feature types chosen in the selector (1). It is used to gauge the effects of the selection performed in view (1) in terms of class separation or overall suitability of the scatterplot to the task at hand. The projection shown here can be next refined to *e.g.* increase separation between desired classes or instances (shapes) using the feature scoring views (3, 4) discussed next. Scatterplots in view (2) are computed either with the t-SNE or UMAP [15] projection methods. t-SNE spreads similar points better over the available 2D space, but is slower. UMAP creates more compact clusters, but is faster than t-SNE. A detailed comparison of these techniques is presented in a recent survey [7].

Feature Scoring Views (3, 4): One can use the feature selector (1) to toggle on or off every feature and gauge its effect on the projection (2). However, this process can be opaque and requires sustained trial and error until a desired effect is obtained. To alleviate this, the barchart (3) shows the *discriminative score* of every element f_i of the 1870-dimensional feature vector, *i.e.*, how much f_i contributes to separating class c_i from a few or from all other classes $c_j \neq c_i$ selected in view (1), depending on the separation control (6, discussed below). Colors show to which feature types the elements f_i belong. For instance, the several purple bars in Fig. 4(3) correspond to the 330 bins of the FPFH feature (purple in Fig. 4(1)). Scores are computed with six scoring methods [23]: chi-squared, one-way ANOVA, Randomized Decision Trees (RDT), Randomized Linear Regression (RLR), iterative relief (IR), and Recursive Feature Elimination (RFE), which are commonly used for assessing classifier performance. The desired scoring method can be chosen by the user in panel (4). This barchart supports two tasks: First, it shows how the many bins of each feature contribute to the separation power of that feature. Secondly, it allows fine-grained examination of the effect of each such bin on class separation: Users can freely *select* specific bins (from the 1870 ones) to create a new projection. Selected bins are outlined with a blue border and listed, in decreasing score order, before the unselected ones, in the barchart. The new projection created by the user-selected bins is shown in view (5).

Refined Scatterplot (5): This projection shows instances from the classes selected in view (1), projected using the feature-bins selected in the barchart (3). This is thus a *refined view* of the original projection (1). By comparing the refined and original scatterplots one can see how the fine-grained selection of each of the 1870 features improves (parts of) the projection. In other words, obtaining an optimal projection is achieved in two steps: First, one selects feature types (in view 1). This is similar to the search process described in Sect. 4.1, except that it is driven by the user rather than automatic. Upon obtaining a suitable

projection by this selection, one *refines* it by (de)selecting individual bins for the used feature types. This corresponds to considering or ignoring *ranges* of the values of the features under exploration.

Separation Control (6): As mentioned, feature scoring gauges how well selected features separate a class c_i from other classes $c_j \neq c_i$. The view (6) allows controlling this. The view shows all shape classes c_i in the database. If all classes are selected in (6), scoring measures how well each class c_i is separated from *each* other class $c_j \neq c_i$. If only one class c_i is selected in (6), scoring measures the separation of c_i from *all* other classes $\cup_{j \neq i} c_j$. This allows one to flexibly measure the separation of arbitrary *groups* of classes rather than only the separation of individual classes themselves.

4.4 Use-Cases

We demonstrate our user-driven projection engineering (Sect. 4.3) by answering two practical questions. More use-cases are described in the original paper [4].

A. How to find the smallest number of features that separate a given shape class from all others? (Q3, Sect. 4.1)

Figure 5 answers this question for classes glasses (a), ant (b), teddy (c), cup (d), and pliers (e). We select each of the aforementioned classes in the model selector (Fig. 4(1)) and use the feature scoring view (Fig. 4(3)) and separation control (Fig. 4(6)) to find feature values (bins of the 1870-dimensional feature vector) that best separate this class from the other 13 ones. We gauge separation via the refined projection (Fig. 4(5)) and its NH_s score. For the specific class examples listed here, Fig. 5 shows that these can be separated very well from the rest of the database by *maximally three*, and sometimes just one, feature bin(s) of the 1870 computed ones.

B. How to create a projection that separates well all classes? (Q1, Sect. 4.1)

Figure 6 shows a typical workflow for answering this question. We start with a default projection that uses all 1870 features. Next, we search, using the feature scoring view (Fig. 4(3)), for feature-bins that are most discriminatory, *i.e.*, have highest scores, for each of the classes in our database, starting with the hand class (we can start from any other class). As we study additional classes, we keep adding feature-bins that are discriminatory for them. When we have visited all classes, we have a candidate feature-set. We next clean (reduce) this set by removing from it features that have low scores, *i.e.*, have low discrimination power or even work adversely. The entire process can be done in a few minutes. The images in Fig. 6 show us how the projection quality NH_s almost continuously improves as we add more feature-bins when considering new classes. During this process, we can visit a given class several times (*e.g.* airplane), as features that score high for it can appear several times during the exploration as we study other classes. The final result (Fig. 6j) contains all 14 classes, has a quality score $NH_s = 0.873$, and uses only 51 feature-bins of the total of 1870 ones. It is important to note that our final NH_s value is *higher* than the one found by exhaustive search ($NH_s = 0.831$, Table 1). Indeed, our manual search is more fine-grained, as it allows us to select or discard individual *feature-bins* (of the 1870 ones); in contrast, automatic search only considered entire *features*

(of the 8 in total). Obtaining a similar-quality result to the one manual search led to, by using exhaustive search, would be prohibitively expensive, as it would involve searching all 2^{1870} feature-bin combinations.

5 Feature Learning Method

Our proposal so far showed how we can construct good-quality projections for exploring 3D shape databases, either by exhaustive search or by user-driven projection design. However, our solution has several limitations:

- **input quality:** Computing the features in Sect. 3.2 involves many constraints. For instance, computing *Agd* requires the meshes to have a single connected component; computing *Gc* requires meshes to be manifold and water-tight. Overall, poor-quality meshes (containing self-intersections, holes, and/or non-uniform sampling) cause serious problems for feature computation;
- **user effort:** The feature selection process (Sect. 4.1), although able to lead to good-quality projections, is time consuming for the user and involves a non-negligible amount of trial and error;
- **replicability:** The used projections (t-SNE and UMAP) are non-parametric. That is, projecting the same (let alone slightly changed) shape database will lead to different scatterplots, thereby not helping users to maintain their mental map of the database;
- **scalability:** For large databases (more than a few hundred shapes), the feature extraction and projection takes considerable amounts of time; moreover, if we consider using more features, both the greedy search and the user-driven projection engineering become slower and more complex to execute;
- **ease of use:** Implementing and setting up the extraction of hand-engineered features (Sect. 3.2) is a highly involved process. Adding more features to the set of existing ones only complicates this process further.

We address all the above issues jointly by using a deep learning approach for both feature extraction (also next feature learning, following deep learning terminology) and projection. Concretely, we adopt PointNet [22] for feature extraction and NNproj [6] for projection. We next outline this approach and its two main components.

PointNet is a deep-learning model used to classify 3D shapes represented as point clouds with very high accuracy [22] (see Fig. 7, blue part). For our visualization goals, and as a replacement of the hand-engineered features described in Sect. 3.2, we use the latent features extracted by PointNet (see Fig. 7, yellow part), after training it for its original classification task using the labels present in the shape database. NNproj is also using deep learning to create high-quality DR projections of arbitrary high-dimensional data [6]. It is trained by providing it with several 2D scatterplots of corresponding feature vectors, created by any desired DR technique, *e.g.*, t-SNE. NNproj can create projections of nearly the same quality as the ground-truth ones it learns from, is thousands of times faster than t-SNE, has a very simple implementation, and is not sensitive to parameter settings or small changes in the input data. We use NNproj to replace t-SNE and UMAP in our visualization construction.

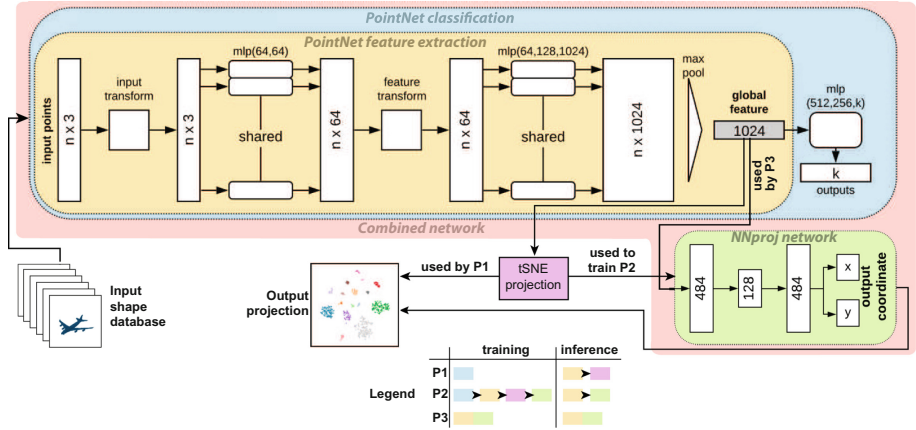


Fig. 7. Architecture of proposed networks P1, P2, P3.

Our framework proposes three pipelines (P1, P2, P3) to create shape-database overviews, as follows. The legend in Fig. 7 shows which models (networks) are part of the training, respectively, inference, of each pipeline. P1 includes PointNet feature-extraction followed by standard t-SNE projection thereof. P1 can already create overviews, but these do not support incremental updating, since t-SNE is non-parametric. Hence, we use P1 mainly for training P2 and P3, as outlined next. P2 runs P1, then trains NNproj to imitate the thus-constructed t-SNE projections, and then uses the trained NNproj instead of t-SNE to create the final projection. P3 drops the classification part of PointNet and trains the joint PointNet-NNProj network. To train P3, we use projections for the training-set shapes created with P1 or P2. In other words: The three pipelines are not different solutions for the same end goal. Rather, P1 is a lower-level pipeline, needed to train PointNet for feature extraction; while P2 and P3 are, functionally identical pipelines that users can choose to use for the final projection construction. The difference between P2 and P3 is simply whether feature extraction and projection are learned separately (P2) or jointly (P3).

For training all models described above, we use the ShapeNet [26] database, which has 14921 shapes from 16 classes. We divide it into a training set (12137 shapes) and a test set (2784 shapes). As this database is quite large, and we have several models to train, we conducted experiments to find how large the training sets of all our deep learning models need to be for sufficient projection accuracy. Specifically, these are:

- NP : the number of shapes for training PointNet,
- NN : the number of feature vectors to train NNProj, and
- NC : the number of shapes for training P3.

We tested 13 values for each of NP , NN and NC , ranging from 320 to 12137. All networks were trained with 250 epochs and early stopping.

5.1 Experiments and Results

We now present the results of our three pipelines (P1, P2, P3) introduced above and how these depend on the sizes of their respective training sets NP , NN , and NC . We evaluate results both qualitatively (by examining the output projections) and quantitatively, by the NH metric (Sect. 3). Since our scatterplots are now larger than those discussed in Sect. 4, we use now a correspondingly larger value $k = 20$ to compute NH . We structure our evaluation along several points, as follows.

Results: Figure 8 shows the overview projection created by P2, with shape icons added to a subset of the database shapes, to limit occlusion. The full projection is shown in the top-left inset as a scatterplot colored by class label. We see that the projection matches our overall expectations: Shapes from different classes are separated well, and similar shapes are close to each other. As with any feature extractor, including the original PointNet, some anomalies exist however. For example, we see a green chair model (A) surrounded by laptop shapes; and a purple lamp model (B) surrounded by table models. This clearly happens since these two shapes are geometrically very similar to the respective classes. Separately, the overview helps us seeing structure *within* classes. For instance, the tables class appears to be visually split into four-legged (FL), round (RO), and bureaus or desks (BU). Note that this information is not available in the original labels of the shape database; it is only the projection that helped us find it.

How much data (NP) is needed to train PointNet? Table 3 shows the test accuracy AC of PointNet for different training-set sizes NP . As NP increases,

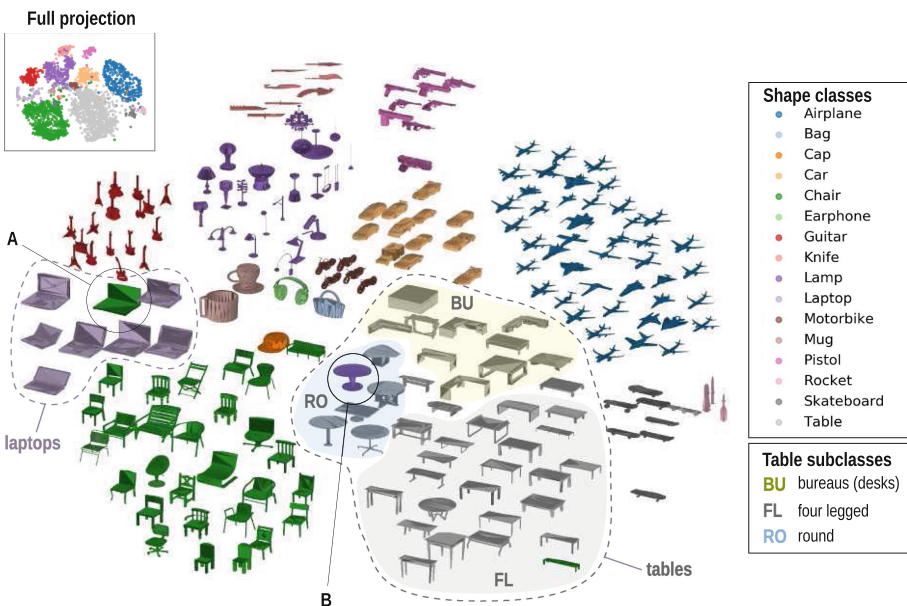


Fig. 8. Overview projection created by pipeline P2.

Table 3. PointNet training accuracy (AC) and NH values for pipeline P1 when training ($P1Train$) and testing ($P1Test$). The two color legends at the bottom show accuracy (green shades) and NH values (yellow-red) respectively in this table and the following ones.

NP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
AC	0.825	0.935	0.954	0.958	0.962	0.963	0.971	0.976	0.976	0.976	0.976	0.976	0.976
P1Train	0.934	0.976	0.985	0.997	0.996	0.999	0.998	0.985	0.992	0.993	0.992	0.991	0.987
P1Test	0.848	0.915	0.931	0.938	0.942	0.934	0.949	0.932	0.945	0.945	0.94	0.943	0.945
low AC	0.825	0.842	0.859	0.875	0.892	0.909	0.926	0.942	0.959	0.976	0.992	0.999	0.999
low NH	0.703	0.736	0.769	0.801	0.834	0.867	0.9	0.932	0.965	0.998	0.998	0.998	0.998

AC also increases until reaching a local maximum ($AC = 97.1\%$) for $NP = 6000$ shapes. The global maximum $AC = 97.6\%$ is achieved, as expected, when using all $NP = 12137$ shapes in the training set. We also see that for $NP = 2000$ we already get a very good accuracy $AC = 95.4\%$, sufficient for our visualization goals. With the trained PointNet, we next extract features and project them using t-SNE (pipeline P1). Table 3, row $P1Train$ shows the NH projection quality metric for P1 on its training sets of various sizes NP . Next, row $P1Test$ shows the same NH metric, this time for the test set. While NH is slightly higher for the training set (as expected), the NH values for the test set are also quite high, indicating that P1 produces good quality projections.

How much data (NP, NN) is needed to train P2? Training P2 requires training PointNet with NP shapes and next training NNproj with NN feature vectors (Fig. 7). So, P2’s quality depends on both NP and NN . Table 4 (a–c) shows this dependency. In detail: Table 4(a) shows the NH value of t-SNE when projecting different sizes NN of feature vector sets extracted by PointNet. We notice that there are some NH fluctuations on the second row where $NN = 320$. However, when NP and NN are both greater than 1000, the t-SNE projections all yield good NH values (above 94%). The highest NH value (99.8%) appears for $NP = NN = 6000$. We also see that the colors in the upper-right triangle half of Table 4(a) are darker than in the lower-left triangle half: NH is slightly higher when $NP \geq NN$. Indeed, when $NP \geq NN$, the input data of t-SNE is a subset of PointNet’s training data. In contrast, when $NP < NN$, t-SNE runs with some shapes that are not in PointNet’s training set.

After creating the ground-truth scatterplots by t-SNE, we use them to train NNProj. After this, P2 is ready to be used. Table 4(b) shows the NH of P2 trained with different NN and NP values, when projecting NNproj’s training-data. The values in Table 4(b) are very close to their counterparts in Table 4(a), being roughly 0.1% to 2% lower. This means that NNproj was trained successfully, so P2 can project well its training data.

Table 4(c) shows NH values when using P2 to project test data (2784 shapes). Although the NH values are slightly lower than those in Tables 4(a) and (b), all of them, except the first one ($NP = 320, NN = 320$) outperform those delivered by our earlier feature engineering. Also, we see that $NN = 2000$ and $NP = 3000$ are already enough to deliver sufficiently high NH values, thus, high-quality projections. The overall highest NH is obtained for $NP = 6000$, same as in Tables 4(a, b). An interesting phenomenon happens when NP is small (320 or 1000). In this case, we train PointNet with few shapes. We can see

Table 4. NH projection quality for (a) ground-truth of pipeline P2; (b) P2 training; (c) P2 testing; and (d) P3, for different values of the respective training-set sizes NP , NN , and NC . Color mapping follows the one in Table 3.

(a)	P2 ground-truth	NNNP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
		320	0.934	0.867	0.889	0.923	0.905	0.824	0.932	0.785	0.866	0.873	0.858	0.843	0.812
		1000	0.888	0.576	0.977	0.951	0.994	0.961	0.997	0.942	0.974	0.961	0.969	0.952	0.957
		2000	0.897	0.96	0.985	0.996	0.997	0.982	0.996	0.97	0.985	0.982	0.983	0.978	0.981
		3000	0.9	0.954	0.974	0.997	0.996	0.986	0.997	0.977	0.99	0.987	0.988	0.982	0.983
		4000	0.901	0.953	0.969	0.99	0.996	0.989	0.998	0.981	0.99	0.989	0.989	0.985	0.983
		5000	0.905	0.953	0.965	0.982	0.989	0.99	0.997	0.981	0.992	0.991	0.99	0.987	0.985
		6000	0.907	0.953	0.963	0.978	0.983	0.985	0.998	0.984	0.992	0.99	0.991	0.987	0.984
		7000	0.907	0.954	0.961	0.976	0.982	0.981	0.993	0.985	0.993	0.991	0.992	0.99	0.983
		8000	0.911	0.952	0.961	0.974	0.98	0.979	0.988	0.982	0.992	0.991	0.991	0.989	0.982
		9000	0.912	0.952	0.963	0.975	0.98	0.977	0.987	0.98	0.99	0.993	0.992	0.989	0.983
		10000	0.915	0.953	0.963	0.974	0.979	0.977	0.985	0.981	0.989	0.991	0.992	0.991	0.985
		11000	0.919	0.955	0.966	0.976	0.98	0.978	0.986	0.982	0.99	0.991	0.992	0.991	0.986
12137	0.916	0.955	0.967	0.975	0.978	0.976	0.985	0.981	0.989	0.988	0.991	0.99	0.987		
(b)	P2 training	NNNP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
		320	0.935	0.865	0.882	0.923	0.906	0.81	0.932	0.782	0.858	0.872	0.858	0.843	0.812
		1000	0.886	0.572	0.971	0.991	0.992	0.958	0.996	0.936	0.963	0.963	0.963	0.939	0.951
		2000	0.893	0.956	0.983	0.996	0.996	0.964	0.994	0.967	0.978	0.978	0.981	0.971	0.976
		3000	0.892	0.951	0.973	0.997	0.992	0.982	0.997	0.97	0.99	0.994	0.982	0.977	0.969
		4000	0.894	0.947	0.965	0.991	0.995	0.986	0.998	0.978	0.984	0.984	0.986	0.979	0.974
		5000	0.897	0.95	0.961	0.978	0.987	0.984	0.996	0.975	0.985	0.986	0.988	0.984	0.98
		6000	0.893	0.948	0.956	0.976	0.979	0.981	0.996	0.978	0.988	0.986	0.984	0.982	0.974
		7000	0.897	0.945	0.955	0.973	0.979	0.978	0.992	0.978	0.986	0.988	0.989	0.98	0.978
		8000	0.9	0.943	0.952	0.967	0.974	0.975	0.986	0.974	0.987	0.988	0.988	0.982	0.976
		9000	0.897	0.943	0.951	0.97	0.968	0.968	0.983	0.968	0.987	0.986	0.988	0.98	0.977
		10000	0.897	0.939	0.952	0.963	0.971	0.973	0.982	0.963	0.985	0.982	0.987	0.983	0.978
		11000	0.896	0.94	0.955	0.966	0.966	0.968	0.978	0.974	0.983	0.986	0.984	0.981	0.975
12137	0.893	0.939	0.952	0.967	0.972	0.965	0.981	0.968	0.985	0.982	0.982	0.975	0.974		
(c)	P2 testing	NNNP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
		320	0.766	0.829	0.857	0.89	0.858	0.847	0.905	0.812	0.877	0.901	0.88	0.866	0.854
		1000	0.809	0.877	0.886	0.9	0.919	0.911	0.927	0.898	0.918	0.914	0.924	0.914	0.924
		2000	0.813	0.878	0.888	0.909	0.923	0.911	0.923	0.899	0.916	0.924	0.919	0.916	0.923
		3000	0.816	0.887	0.897	0.904	0.909	0.922	0.93	0.908	0.925	0.919	0.928	0.92	0.916
		4000	0.823	0.897	0.897	0.905	0.912	0.919	0.925	0.916	0.924	0.919	0.926	0.921	0.912
		5000	0.827	0.897	0.897	0.905	0.918	0.919	0.93	0.905	0.92	0.923	0.929	0.925	0.924
		6000	0.83	0.897	0.903	0.908	0.91	0.917	0.924	0.907	0.922	0.917	0.916	0.923	0.909
		7000	0.83	0.896	0.904	0.908	0.922	0.925	0.929	0.906	0.917	0.923	0.92	0.923	0.917
		8000	0.835	0.893	0.908	0.906	0.914	0.916	0.932	0.914	0.917	0.928	0.918	0.917	0.926
		9000	0.833	0.895	0.905	0.916	0.913	0.916	0.925	0.911	0.929	0.929	0.927	0.917	0.919
		10000	0.836	0.885	0.898	0.901	0.915	0.916	0.925	0.906	0.919	0.929	0.913	0.92	0.918
		11000	0.84	0.89	0.911	0.911	0.898	0.905	0.919	0.913	0.919	0.923	0.922	0.931	0.915
12137	0.825	0.895	0.894	0.904	0.92	0.914	0.935	0.912	0.926	0.922	0.915	0.916	0.916		
(d)	P3	NC	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
		P1Train	0.934	0.976	0.985	0.997	0.996	0.99	0.998	0.985	0.992	0.993	0.992	0.991	0.987
		P1Test	0.848	0.915	0.931	0.938	0.942	0.934	0.949	0.932	0.945	0.945	0.94	0.943	0.945
		P2Train	0.935	0.972	0.983	0.997	0.995	0.984	0.996	0.978	0.987	0.986	0.987	0.981	0.974
		P2Test	0.766	0.877	0.888	0.904	0.912	0.919	0.924	0.906	0.917	0.929	0.913	0.931	0.916
		P3Train	0.927	0.963	0.973	0.993	0.993	0.979	0.99	0.975	0.981	0.988	0.986	0.982	0.979
P3Test	0.703	0.887	0.879	0.912	0.913	0.921	0.909	0.922	0.918	0.93	0.921	0.92	0.919		
NH	0.703	0.736	0.769	0.801	0.834	0.867	0.9	0.932	0.965	0.998					

that this does not yield high NH values. However, when next using more shapes to train NNproj (NN increases), NH also increases. That is, we can use a small labeled dataset to train PointNet, and then use a larger *unlabeled* dataset to improve P2’s performance.

How much data (NC) is enough to train P3? Table 4(d) shows the NH results of P3, trained using P1 for different training set sizes NC , and compares them with those of P1 and P2. To ease comparison, the values in rows *P2Train* and *P2Test* in Table 4(d) come from the diagonals of Tables 4(b, c) for $NC = NN = NP$. Rows *P1Train* and *P1Test* show the NH values of P1 projecting its training, respectively test, data. Rows *P3Train* and *P3Test* show the NH values for P3 on training, respectively test, data. From Table 4(d), we see that P3 performs similarly to P2 on both training and test data, with good NH values when having at least $NC = 3000$ training shapes.

How does PointNet’s accuracy influence P1 and P2? As explained, PointNet was originally designed for *classification*. However, we use here PointNet’s

feature vectors for *projection*. So, it is interesting to see if the classification-related accuracy (AC) and projection-related quality (NH) are correlated. Separately, we ask ourselves if there is a relationship between the NH of ground truth t-SNE and the NH of projections created with P2 and P3. To explore this, we draw scatterplots of these values and compute their Pearson Correlation Coefficients (PCC). Figure 9(a) shows the AC vs P1 NH and AC vs P2 NH scatterplots. The plot contains 13 point-groups, one for P1 (blue), and the other 12 for an NN setting of P2 each (green color-coded on NN). These point-groups are visually indicated by different colors and also connected by lines for easing reading. The dotted line shows ideal perfect correlation, for reference. We see that the PCCs of all these lines—each representing an instance of P1 or P2—are close to 1, so P1 and P2’s NH metric directly correlates with PointNet’s accuracy.

Next, Fig. 9(b) shows scatterplots of P2 and P3’s NH vs t-SNE’s NH values. The plot contains 13 point-groups, one for P3 (red), and the other 12 for a NN setting of P2 each (green color-coded on NN). The PCCs of these lines are also close to 1, so P1 and P2’s NH quality is directly correlated with the ground-truth (t-SNE)’s quality. A similar correlation—albeit for a different deep learning model for performing projections—was mentioned in [6], but not formally assessed by means of PCC. The data for P3 (red line) may seem at first sight far worse than that for P2 (green lines). However, this is due to a single point for the lowest t-SNE NH value. For all other values, the red line is practically in the same area as the green lines, telling that P2 and P3 are very similar from the perspective of t-SNE vs deep-learning-network produced NH values.

How can we use classification accuracy to interpret projections? In P1 and P2, we trained PointNet for classification. Besides a feature vector, this delivers a *confidence* value for the classification. We can use this value for our visual exploration goal, as follows. Figure 10 (top) shows the training-set projected by P2, with point luminances encoding their classification confidences (dark = low, bright = high confidence). We immediately see that confidence is high within same-class clusters and low on the cluster borders. Also, as we increase the number of training samples NP for the PointNet classifier, we see how the confidence nears 1.0 for most samples; although, the lowest-confidence ones still remain on

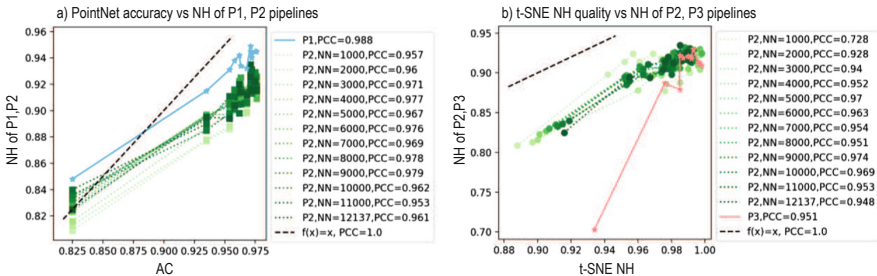


Fig. 9. a) Correlation of PointNet’s accuracy AC with the NH quality of pipelines P1 and P2. b) Correlation of t-SNE’s NH quality with the NH quality of pipelines P2 and P3.

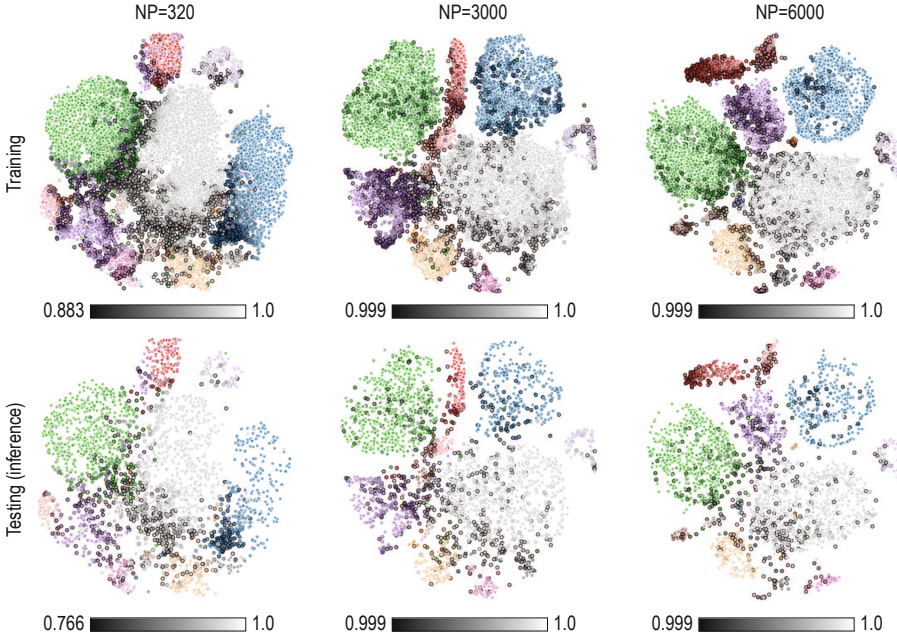


Fig. 10. Classification confidence values (dark = low, bright = high confidence), during training (top) and testing (bottom) for the P2 pipeline trained with three NP values.

cluster borders. Figure 10 (bottom) shows the same visualization for a test-set projected by the trained P2. For a small test set $NP = 300$, we see that confidence is significantly lower than on the training set. For $NP \geq 3000$, test set confidence is basically the same (nearly one) as training set confidence. Importantly, we see that confidence is relatively lowest on the cluster borders also for the test data. We can use these visualizations (on the test data) to assess how confident we are that the shape database projection indeed faithfully reflects the similarities of the underlying shapes. As expected, shapes close to cluster borders are harder to classify, thus, have less discriminant feature vectors and are in turn harder to project well. Upon seeing such images, users can decide to *e.g.* further explore additional information concerning shapes having low classification confidence. This type of insight is crucial when interpreting projections as it is well known that such methods cannot always place all their input data correctly [7, 14, 17].

5.2 Computational Performance

We discuss next the computational performance of our three pipelines P1-P3. For this, we split effort into *setup* effort, *i.e.*, the time needed to perform all operations required to have the pipeline ready for inference; and *inference* effort, *i.e.*, the time a pipeline needs to create the projection of a shape database.

Table 5. Setup time, pipelines P1–P3 (top). Setup time of P2 as function of NP and NN (bottom).

Setup time P1-P3	NC	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
	P1	347.4	1045.8	2058.2	3003.1	4049.0	5046.0	6027.3	6927.6	8040.8	8912.8	9836.0	11849.7	13187.9
	P2	361.2	1073.1	2118.0	3068.5	4128.7	5117.4	6157.8	7074.9	8205.5	9057.0	10027.2	12033.6	13403.8
	P3	301.7	886.8	1743.7	2607.2	3475.7	4349.5	5198.4	6176.8	7221.8	7890.5	8758.4	9557.1	10586.9
	P3'	649.1	1932.5	3802.0	5610.2	7524.8	9395.5	11225.7	13104.4	15262.6	16803.3	18594.4	21406.8	23774.8
Setup time details for P2	NNINP	320	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000	12137
	320	361.2	1062.4	2082.4	3027.8	4079.4	5068.9	6066.9	6963.5	8073.4	8947.0	9876.5	11903.7	13228.5
	1000	360.9	1073.1	2085.4	3030.1	4084.7	5098.2	6051.1	6952.0	8085.3	8941.4	9868.8	11887.4	13223.7
	2000	397.4	1097.5	2118.0	3062.3	4128.3	5073.5	6085.1	6992.2	8087.6	8946.3	9936.7	11914.2	13259.0
	3000	406.1	1111.8	2155.8	3068.5	4104.0	5141.1	6119.8	7022.1	8135.2	8976.9	9912.0	11961.5	13242.5
	4000	432.0	1126.7	2174.8	3093.7	4128.7	5117.1	6120.3	7044.2	8135.0	9002.4	9949.1	11957.8	13301.7
	5000	470.6	1178.6	2180.8	3098.0	4134.9	5117.4	6125.6	7025.4	8110.5	9077.5	9951.4	12030.3	13324.4
	6000	444.5	1189.7	2172.0	3161.5	4192.6	5204.1	6157.8	7028.0	8151.5	9059.0	9963.2	12007.2	13324.8
	7000	482.3	1179.0	2198.5	3134.6	4234.4	5235.1	6195.6	7074.9	8142.5	9105.8	10035.5	11973.4	13316.8
	8000	497.8	1194.9	2214.5	3174.8	4236.6	5241.8	6141.0	7061.3	8205.5	9077.1	10018.9	12021.3	13383.4
	9000	517.7	1215.4	2205.5	3226.3	4264.8	5152.2	6238.7	7161.2	8223.3	9057.0	9884.9	12024.7	13432.3
	10000	543.1	1276.0	2262.5	3142.5	4211.5	5200.3	6249.9	7181.3	8295.5	9051.6	10027.2	12044.8	13356.8
	11000	451.9	1209.1	2236.9	3238.2	4220.6	5253.2	6210.6	7108.2	8321.5	9165.8	10040.2	12093.6	13362.3
	12137	566.1	1331.4	2324.1	3250.0	4237.2	5204.9	6271.7	7123.1	8225.2	9233.7	10078.8	11983.9	13403.8
time (secs)	801	2910	5518	8126	10734	13342	15950	18559	21167	23775				

Setup Time: Table 5(top) shows the setup time for all three pipelines. Columns N indicate the training set size for the three pipelines, *i.e.*, NP for PointNet, NN for NNproj, and NC for P3, respectively. Row $P1$ shows the setup time for P1, identical to PointNet’s training time. Row $P2$ shows the setup time for P2 when $NN = NP$ (Table 5 (bottom) gives more detailed information, see next). The setup time for P2 includes training PointNet, feature extraction, ground truth generation (t-SNE), and training NNproj. Comparing the first two rows in Table 5 (top), we see that training PointNet is dominating the setup of P2. Row $P3$ shows the setup (training) time of P3 when we already have a ground truth projection. We see that training P3 is slightly faster than training P1 since P3’s network is slightly simpler. Finally, row $P3'$ shows the setup time for P3 when we use P1 to create the ground truth needed for training it. Table 5 (bottom) shows the setup time of P2 for all combinations of NP and NN in our experiments. Values in this table increase rapidly with NP and slightly with NN . That is, training NNProj is negligible compared to training PointNet.

Inference Time: Figure 11 shows the projection (inference) time for three pipelines as a function of how many shapes they need to project. We see that all three pipelines have linear time complexity. P1 and P2 are really close and they are about 5 times faster than P1. The high relative cost of P1, and its deviations from a perfect line, are explained by the fact that P1 uses t-SNE, whose cost (a) is high and (b) varies depending on its stochastic initialization, this explaining the wiggles in the blue line in Fig. 11 (for a related analysis, see [7]). In contrast, P2 and P3 show a perfect linear relation with the shape database size, as these are purely deep-learning model executions.

These three pipelines are all much faster than our feature selection method which takes about 127h to extract the 8 features we listed in Table 1 for 320 shapes.

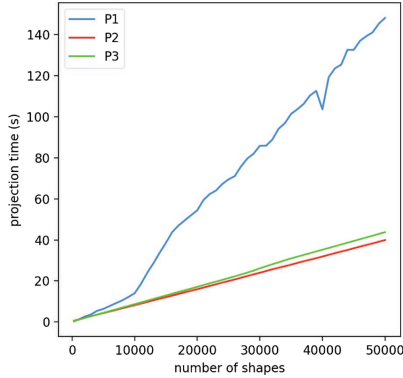


Fig. 11. Projection time comparison for pipelines P1, P2, P3.

6 Discussion

We discuss next several aspects of our proposal, as follows.

Feature Selection vs Feature Learning: We have presented two approaches for creating shape-database projections: *selecting* features from a pre-computed set based on feature engineering (Sect. 3) vs using an automatically *learned* feature vector using deep learning (Sect. 5). We call these next the feature selection (FS) and the feature learning (FL) approaches. Both approaches share the same aims, listed as Q1-Q3 in Sect. 4.1. From this viewpoint, each approach has its own advantages and limitations. As mentioned at the beginning of Sect. 5, FS has some clear limitations with respect to input shapes, user effort, replicability, scalability, and ease of use – thus, it does not fully address Q1. The FL approach scores very highly on all these points: It accepts any point cloud shape as input, so has no constraints on mesh quality (input shape independence); it works fully automatically, not requiring any specific user input (low user effort); it creates projections deterministically, thus stably upon small-to-medium input changes (replicability); it scales linearly with the input size, being 4 orders of magnitude faster than FS; and it is very easy to deploy, being based on standard deep learning libraries [18]. Also, FL addresses Q2 (Sect. 4.1, *i.e.*, which is the minimal feature-set needed for a good projection, in a different way than FS: Its deep learning approach does not care about feature *selection*, but rather *synthesizes* features which are best for good projection creation. The results in Sect. 5.1 show quite clearly that FL can create high-quality projections this way, without having to worry about feature selection. In contrast, FS *must* consider feature selection, since it is by construction restricted to a fixed number of predefined features.

However, FS has two interrelated advantages upon FL: First, it allows users to see and select how features affect the projection (Q2, Sect. 4.1). The bar chart view (Fig. 3) allows users to find and select features to optimize the projection quality. Secondly, the FL feature scoring view (Fig. 4) interacts with the projection view to enable users to select specific features that explain the similarity of shapes in a group and/or the separation of several groups, and, more importantly,

to generate custom projections in which separation of specific classes is favored. Using these views for the same tasks with the FL features is not straightforward, since NNproj is trained on an *entire* feature set and would need re-training if this set changes. Also, the FL features are *abstract*, *i.e.*, they do not have a concrete meaning for users, thereby making reasoning about them extremely challenging. Hence, globally put, FL is better when one wants a full control (and understanding) of how features drive the projection creation; whereas FS is better when one wants a fully automatic, easy to use, and scalable method that creates overall good separation.

Selecting the Best Feature Learning Approach: We have studied and presented two approaches for jointly doing feature learning and projection, called P2 and P3. Which one is best? Our results (Sect. 5.1 shows that P2 and P3 produce very similar results, quality-wise, given the same (amounts of) training data. P3 exhibits slightly higher quality than P2, which makes sense, as P3 trains jointly for both feature extraction and projection. Separately, the results in Sect. 5.1 show very little variation in performance of all pipelines as a function of their training set sizes. In practice, as we discussed there, setting NP , NN , and NC around 3000 shapes gives good results for all pipelines, with only minimal improvements obtained when tweaking these training-set size values.

Learning Projections: Currently, we train NNproj with all the 1024 PointNet features to create projections. We could reduce this dimensionality to a lower value using an intermediate autoencoder stage, or alternatively using a feature-selection optimization technique as presented in Sect. 4.1. This would possibly make NNproj’s task of learning projections easier in terms of training set size required to obtain a certain class separation (NH) and/or epochs needed for convergence. Concerning the choice of projection techniques, we used t-SNE to train NNproj. However, learning other projection techniques such as UMAP may lead to ultimately better, easier to interpret, projections.

Scalability: Both the FS and FL approaches depend linearly on the number of *shapes* in the database to be explored and the number of *features* which are extracted from each shape. Yet, as explained already, FS is 4 orders of magnitudes slower. For handling real-world databases of tens of thousands of shapes, like ShapeNet, the FL approach is clearly more suitable. Note that both FS and FL approaches can be applied offline, *i.e.*, when shapes are changed and/or new shapes are added to the database. Shape databases do not change with a high frequency, so offline extraction can be done without highly affecting the performance for the end user. We compute the t-SNE projection using *scikit-learn*, which projects several hundreds of instances in a few seconds; the UMAP implementation, provided by the authors [15], works in real time for this dataset size. If needed, other, faster projections can be used [21]. From a visualization viewpoint, the scatterplot, barchart, and matrix plot metaphors we use scale well to hundreds of thousands of points (shapes) and tens of features.

Evaluation: One important aspect concerning our proposal is evaluating its effectiveness for different types of tasks and users. In detail, we identify *end users*, for whom tasks involve getting an overview of a shape database, finding similar groups of shapes, finding which features make two shape groups similar (or different), and finding outlier shapes; and *technical users*, for whom tasks

involve selecting a small set of features able to create effective visualizations for the first user group. We consider such evaluations to be part of future work.

7 Conclusion

We have presented a set of techniques for creating 2D visual representations for exploring 3D shape databases for CBSR applications. Our approach is based, first, on reducing shapes to feature vectors, followed by using dimensionality reduction to create, and explore, 2D scatterplots that encode the shapes' similarities. We support both above steps (feature extraction and projection creation) by two different mechanisms. First, we use a feature engineering approach, followed by a scagnostics approach to create near-optimal projections, and accelerate the automated search for good feature combinations using a greedy technique. To refine the created projections beyond what the automatic search can do, we propose a visual analytics workflow that enables users to customize the obtained projections in terms of separating specific classes or generating high-separation projections for all classes, based on the separation power of all available features. We show that our user-driven approach can create projections with better separation than the automatic one, and also helps finding discriminating features (to be used in a CBSR system) and confusing features (of little value for such systems). Our second approach uses deep learning approach to jointly cover feature extraction and dimensionality reduction. This makes the end-to-end pipeline automatic, easy to use, robust to database changes, and computationally scalable. Both our approaches can be applied to any 3D shape database, allowing CBSR engineers to streamline the process of designing and selecting effective features for shape classification and retrieval. We demonstrate our work on two real-world 3D shape databases.

This work can be extended in several directions, as follows. Performing a user study to measure how well our techniques can support typical exploration tasks related to 3D shape databases, either in opposition to, or as an addition to, existing exploratory tools for such databases, would be of evident added value. Secondly, our techniques are generic, being able to handle any data collections that can be described in terms of high-dimensional feature vectors. Hence, it is interesting to consider its deployment in supporting the exploration of other data types, such as image and/or text collections or scientific data collections.

References

1. Aim@Shape: Aim@shape digital shape workbench 5.0 (2019). <http://visionair.ge.imati.cnr.it>
2. Belongie, S., Malik, J., Puzicha, J.: Shape context: a new descriptor for shape matching and object recognition. In: Proceedings of the NIPS, pp. 831–837 (2001)
3. Bustos, B., Keim, D., Saupe, D., Schreck, T., Vranic, D.: Feature-based similarity search in 3D object databases. *ACM Comput. Surv.* **37**(4), 345–387 (2005)
4. Chen, X., Zeng, G., Kosinka, J., Telea, A.: Visual exploration of 3D shape databases via feature selection. In: Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP, pp. 42–53. INSTICC, SciTePress (2020). <https://doi.org/10.5220/0008950700420053>

5. Cyr, C.M., Kimia, B.B.: 3D object recognition using shape similarity-based aspect graph. In: Proceedings of the IEEE ICCV, pp. 254–261 (2001)
6. Espadoto, M., Hirata, N., Telea, A.: Deep learning multidimensional projections. *J. Inf. Vis.* (2020). <https://doi.org/10.1177/1473871620909485>
7. Espadoto, M., Martins, R., Kerren, A., Hirata, N., Telea, A.: Towards a quantitative survey of dimension reduction techniques. *IEEE TVCG* (2019). <https://doi.org/10.1109/TVCG.2019.2944182>
8. Feng, C., Jalba, A.C., Telea, A.C.: Improved part-based segmentation of voxel shapes by skeleton cut spaces. *Math. Morphol. - Theory Appl.* **1**(1) (2016)
9. ITI DB: The informatics & telematics institute database (2019). <http://3d-search.iti.gr/3DSearch/index.html>
10. Jalba, A., Kustra, J., Telea, A.: Computing surface and curve skeletons from large meshes on the GPU. *IEEE TPAMI* **35**(6), 783–799 (2013)
11. Jalba, A., Kustra, J., Telea, A.: Surface and curve skeletonization of large 3D models on the GPU. *IEEE TPAMI* **35**(6), 1495–1508 (2012)
12. Kalogerakis, E., Hertzmann, A., Singh, K.: Learning 3D mesh segmentation and labeling. *ACM TOG* **29**(4) (2010)
13. van der Maaten, L., Hinton, G.: Visualizing high-dimensional data using t-SNE. *J. Mach. Learn. Res.* **9**, 2579–2605 (2008)
14. Martins, R., Coimbra, D., Minghim, R., Telea, A.: Visual analysis of dimensionality reduction quality for parameterized projections. *Comput. Graph.* **41**, 26–42 (2014)
15. McInnes, L., Healy, J., Melville, J.: UMAP: uniform manifold approximation and projection for dimension reduction. [arXiv:1802.03426](https://arxiv.org/abs/1802.03426) (2018)
16. NASA: Nasa 3D resources (2019). <https://nasa3d.arc.nasa.gov>
17. Nonato, L., Aupetit, M.: Multidimensional projection for visual analytics: linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG* (2018). <https://doi.org/10.1109/TVCG.2018.2846735>
18. Paszke, A., et al.: Pytorch: an imperative style, high-performance deep learning library. In: Wallach, H., et al. (eds.) *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc. (2019). <http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
19. Paulovich, F.V., Nonato, L.G., Minghim, R., Levkowitz, H.: Least square projection: a fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG* **14**(3), 564–575 (2008)
20. Peyre, G., Cohen, L.: Geodesic computations for fast and accurate surface remeshing and parameterization. In: Bandle, C., et al. (eds.) *Elliptic and Parabolic Problems. PNLDE*, vol. 63, pp. 151–171. Springer, Heidelberg (2005). https://doi.org/10.1007/3-7643-7384-9_18
21. Pezzotti, N., Lelieveldt, B.P., van der Maaten, L., Höllt, T., Eisemann, E., Vilanova, A.: Approximated and user steerable t-SNE for progressive visual analytics. *IEEE TVCG* **23**(7), 1739–1752 (2017)
22. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: deep learning on point sets for 3D classification and segmentation. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017
23. Rauber, P.E., da Silva, R.R.O., Feringa, S., Celebi, M.E., Falcão, A.X., Telea, A.C.: Interactive image feature selection aided by dimensionality reduction. In: *Proceedings of the EuroVA*, pp. 19–23 (2015)
24. Rusu, R.B., Blodow, N., Beetz, M.: Fast point feature histograms (FPFH) for 3D registration. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3212–3217 (2009)
25. Schmidt, W., Sotomayor, J., Telea, A., Silva, C., Comba, J.: A 3D shape descriptor based on depth complexity and thickness histograms. In: *Proceedings of the SIBGRAPI* (2015)

26. ShapeNet: ShapeNet online repository (2019). <https://www.shapenet.org>
27. Shapira, L., Shamir, A., Cohen-Or, D.: Consistent mesh partitioning and skeletonisation using the shape diameter function. *Vis. Comput.* **24**(4), 249–262 (2008)
28. Shen, Y.T., Chen, D.Y., Tian, X.P., Ouhyoung, M.: 3D model search engine based on lightfield descriptors. In: *Eurographics 2003 - Posters*. Eurographics Association (2003). <https://doi.org/10.2312/egp.20031031>
29. Shilane, P., Min, P., Kazhdan, M., Funkhouser, T.: The Princeton shape benchmark. In: *Proceedings of the SMI*, pp. 167–178 (2004). <http://shape.cs.princeton.edu/benchmark>
30. Shneiderman, B.: The eyes have it: a task by data type taxonomy for information visualizations. In: *Proceedings of the IEEE Symposium on Visual Languages*, pp. 336–343 (1996)
31. Shtrom, E., Leifman, G., Tal, A.: Saliency detection in large point sets. In: *Proceedings of the IEEE ICCV*, pp. 3591–3598 (2013)
32. Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E.: Multi-view convolutional neural networks for 3D shape recognition. In: *Proceedings of the IEEE ICCV*, pp. 945–953 (2015)
33. Tangelder, J., Velkamp, R.: A survey of content based 3D shape retrieval methods. *Multimed. Tools Appl.* **39**(3), 441–471 (2008)
34. Tasse, F., Kosinka, J., Dodgson, N.: Cluster-based point set saliency. In: *Proceedings of the IEEE ICCV*, pp. 163–171 (2015)
35. Telea, A., Jalba, A.: Voxel-based assessment of printability of 3D shapes. In: Soille, P., Pesaresi, M., Ouzounis, G.K. (eds.) *ISMM 2011*. LNCS, vol. 6671, pp. 393–404. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21569-8_34
36. Tukey, J., Tukey, P.: Computer graphics and exploratory data analysis: an introduction. In: *The Collected Works of John W. Tukey: Graphics: 1965–1985* (1988)
37. TurboSquid Inc: Turbosquid shape repository (2019). <https://www.turbosquid.com>
38. Verma, V., Snoeyink, J.: Reducing the memory required to find a geodesic shortest path on a large mesh. In: *Proceedings of the ACM GIS*, pp. 227–235 (2009)
39. Wattenberg, M.: How to use t-SNE effectively (2016). <https://distill.pub/2016/misread-tsne>
40. Wilkinson, L., Anand, A., Grossman, R.: High-dimensional visual analytics: interactive exploration guided by pairwise views of point distributions. *IEEE TVCG* **12**(6), 1363–1372 (2006)