

Content-Based Shape Retrieval

Sanna Heesakkers

Utrecht University

3422453

s.h.m.heesakkers@students.uu.nl

Arjen Simons

Utrecht University

1393170

a.simons1@students.uu.nl

Jasper de Winther

Utrecht University

1326236

j.dewinther@students.uu.nl

ABSTRACT

Retrieving multimedia objects can be a hard task. This paper aims to create a Content-Based Shape Retrieval (CBSR) system which is able to return 3D shapes that are similar to a given query shape. We show how to pre-process shape databases [10][3] in order to extract the features used to compare different shapes. The comparison results in a distance between the shapes computed using the Euclidean or Earth mover's distances between individual features. An understandable visualisation of the dataset and its distances is presented using a t-SNE plot, which clearly shows which shapes are considered similar to one another. We show that the resulting CBSR system can accurately retrieve objects that have a lot of similar geometric attributes. To calculate the sensitivity and specificity of the presented pipeline, a Receiver Operating Characteristic (ROC) curve is computed for every class. Over the entire Labeled PSB dataset [3], the system has an average Area Under the ROC (AUROC) of 0.82. Some shape classes have an average AUROC up to 0.96.

KEYWORDS

MMR, Multimedia Retrieval, 3D Shape retrieval, 3D shape matching

1 INTRODUCTION

When trying to find information on a specific subject, the Google search engine can be used, which provides multiple sources when provided with a few words. This technique can also be applied to different data formats like images. In this report we describe a pipeline which enables us to use this kind of fuzzy search for 3D models. The pipeline is given a shape and returns an given number of 3D models that it determines to be the most similar.

The database used to provide the shapes in the project is the Princeton Shape Benchmark (PSB) [10] in combination with the Labeled PSB dataset [3]. We simply combine these datasets to get a broader range of shapes.

In Section 2, the pipeline setup is discussed. The section explains the rendering and system architecture design, used for the rest of the pipeline.

The following section (Section 3) covers the preprocessing steps. All shapes must be re-meshed and normalised to have certain non discriminating properties such as scale and position normalised. This section describes which properties are normalised, how they are normalised and how the distribution of these properties change after normalising.

Section 4 aims to retrieve features that can be used to discriminate between different shapes. It focuses on both global features that can be represented as a single real value and shape property features which are distributions that can be represented as histograms.

The proceeding section (Section 5) describes the querying stage of the pipeline. The feature ranges will be normalised and different distance functions will be discussed. It also covers the retrieval of a specified number of shapes whose distances are closes to the query shape.

Section 6 covers the pipeline's scalability. It discusses an improved data structure that allows for efficient querying over a large dataset. It also displays an overview of the dataset in the form of t-SNE plots.

The next section (Section 7) discusses the quality of the pipeline. It investigates what quality measure to use to evaluate the system as a whole. Afterwards, the results are shown and reflected upon.

Lastly (Section 8), a discussion of the results is presented to conclude the entire process.

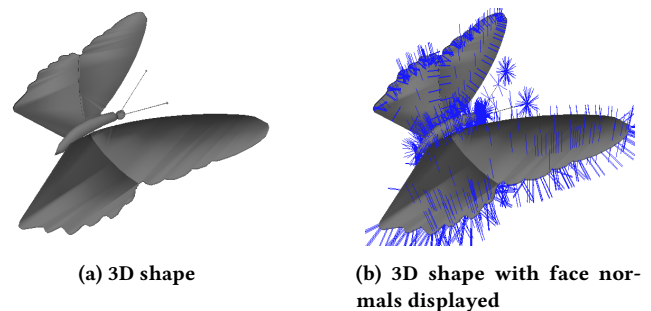


Figure 1: 3D rendering with Pyrender

2 STEP 1: WORKING WITH 3D SHAPES

The first step in rendering shapes is to read the shape's data. To achieve this, the Trimesh [1] library is used. This is a native python library for 3D mesh processing. This library is chosen because it is python native, which makes interfacing with it easier.

We decided upon an architecture where the entire dataset is loaded into memory at all times, and is stored in a standard list containing our custom *MeshData* class. This *MeshData* includes file location, Trimesh data, and multiple other computed features of the mesh. Luckily, our dataset is approximately half a gigabyte in size when loaded in RAM so this should not be a bottleneck.

To efficiently work with our data, the result of an entire processing step (the list of *MeshData* classes) is cached in a pickle file. As a result, when loading the mesh data, the processing time went from at least 20 seconds, to sub one second on consecutive runs. Reading many different files causes a lot of system level overhead which is mostly negated in our implementation after the first run. Consequently, when performing multiple operations that work on the entire dataset, the performance is bound by the operations applied to the meshes and not the time it takes to prepare the input. For

example, not all meshes need to individually be loaded and parsed when applying remeshing, but a single data-blob can be read and interpreted as the dataset.

The actual rendering of the 3D meshes is performed using Pyrender [5], a pure Python library for physically-based rendering. It is an easy to use library that comes with intuitive controls to view the models from different angles and distances. In addition, the library helps to compute face and vertex normals, and supports the visualisation of various 3D mesh attributes such as the mesh wireframe and the face/vertex normals (see Figure 1).

Due to the significant computational cost of the later steps in the pipeline, multithreading is an obvious first step in improving computation times. However, Python does not natively allow multithreading due to the Global Interpreter Lock (GIL). To circumvent this, the multithreading package was used which starts new Python instances to bypass the GIL, which does come with a cost memory wise. Unfortunately, the scaling also is not 1 to 1 per added core. With a many (8) core system there was a significant speedup.

3 STEP 2: PREPROCESSING

In order to discriminate between models, certain features must be extracted. The model’s bounding box size, translation, rotation, vertex and face count are not relevant discriminatory factors. They do however, influence relevant features such as surface area. To ensure that all relevant features can be extracted fairly on every model, these impartial features must be normalised.

In the first subsection, certain model features will be normalised and determined. The distribution of these features in the dataset will be investigated. The following section covers the normalisation of the translation, orientation and scale. First, the translation will be normalised, because the normalisation of the models’ orientation is dependent on this. After that, the models’ orientation will be normalised before their bounding box scales to ensure that only the axis-aligned bounding box (AABB) has to be computed. Subsequently, the last subsection covers the normalisation of the meshes their face and vertex count. This mesh resolution normalisation step is independent from any other normalisation steps. It does not matter whether the other normalisation steps are performed prior to, or after the mesh resolution step.

3.1 Determine model features

For every shape in the dataset, the class, the number of faces and vertices, the type of faces and the axis-aligned 3D bounding box are determined. This data is used to create an overview of the shapes and how they compare to one another.

Both the face and vertex counts have some extreme outliers. This will become problematic when trying to calculate features which can have $\Theta(n^2)$ or even $\Theta(n^3)$ time complexity. In table in Figure 2b we see that there are multiple objects with too high of a value to realistically run any $\Theta(n^3)$ algorithm on. The distribution of the face and vertex counts can be seen in Figure 2a.

To measure the variance of the bounding boxes, two variables are used: the length of the bounding box diagonal and the volume of the bounding box. The table in Figure 2b shows that the median of both variables is relatively close to the smallest value; meanwhile the mean is a few order of magnitudes bigger. This means that most

bounding boxes do not deviate a lot from one another, with a few very large outliers.

3.2 Normalising the meshing resolution

The variance in number of vertices and faces makes it hard to compare models in the future. Not only from a correctness point of view, but the computational complexity will also explode when there are too many vertices. Because of this, we decided to repeatedly subdivide each face into four until there are at least 1,000 faces. This method of subdividing triangles into four, instead of three, will keep face shapes more uniform, and minimises the number of very thin and elongated triangles. To reduce the number of vertices to computationally acceptable levels, quadric mesh simplification is being performed (using the Python Fast Quadric Mesh Reduction library [9]) as can be seen in Figure 3. This algorithm attempts to reduce the number of faces below 5,000 but can deviate from that on very complex meshes. This way most meshes will have between 1,000 and 5,000 faces.



(a) Plane mesh with 10,796 faces

(b) Normalised plane mesh with 5,000 faces

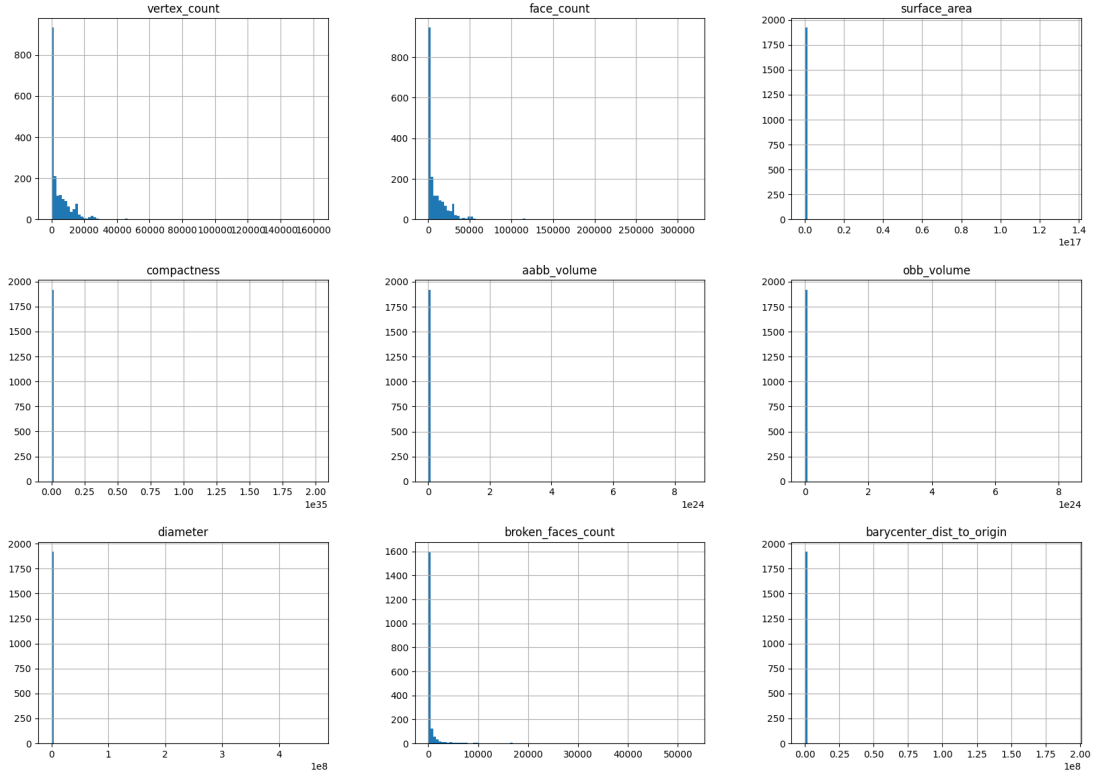
Figure 3: Mesh normalisation being applied to a plane mesh

3.3 Invalid mesh removal

Some meshes have errors in them. This could be a watertight issue, which can be resolved most of the time, or something more serious. When determining specific features, a few models were faulty in some way which caused their volume or surface area to be infinite, NaN or unreasonable. Fortunately, this only happened in very few cases, so we simply removed those meshes from our pipeline. Another problem with our remeshing is that it does not work all the time. For example, it is very hard to reduce the number of faces of a tree containing many leaves without destroying its shape. These models which cannot be simplified further often have many holes in them already, and thus mesh decimation would not be a good fit. Because of this, we decided to refine our dataset by throwing away 10% of the models with the highest ratio of non-connected faces, and the top 5% of the models with the most faces. This way the most problematic outliers are removed, while still keeping a large enough dataset.

3.4 Normalising shape translations

As stated earlier, the table in Figure 2b shows that the dataset contains some huge outliers in terms of distance from the shape’s barycenter to the origin of the reference coordinate frame. To easily



(a) Histograms of dataset features.

	No. vertices	No. faces	Surface area	Compactness	AABB volume	OBB volume	Diameter	No. broken faces	Distance to origin
Min	10	16	0.024	1.21e-4	2.75e-4	2.71e-4	0.19	0	2.24e-4
Max	160,940	316,498	1.35e+17	2.00e+35	8.55e+24	8.30e+24	4.63e+8	52,723	1.92e+8
Mean	5452.03	10,396.24	7.00e+13	1.84e+32	4.45e+21	4.32e+21	240,735.49	576.79	101,232.94
Median	1802	3378	1.39	31.10	0.22	0.19	1.01	42	0.61
Std	9305.39	17,883.77	3.07e+15	5.59e+33	1.95e+23	1.89e+23	1.06e+7	2291	4.37e+6

(b) Dataset statistics.

Figure 2: Dataset (1923 models) characteristics before mesh normalisation. Respectively they show the number of vertices, number of faces, surface area, compactness, axis-aligned bounding box volume, oriented bounding box volume, diameter, the number of broken faces and the distance from the barycenter to the origin of the reference coordinate frame.

perform other normalisation, that for example rotates the object to normalise its orientation, the object must be translated such that its barycenter c is located at the origin of the reference frame. To do this, every object point p_i must be updated using formula 1.

$$p_i^{\text{updated}} = p_i - c \quad (1)$$

After translating the shapes in the database, the table in Figure 8b

shows that the largest distance to origin is $7.75e-13$. This is very close to zero. No barycenter seems to have a distance of exactly zero. This seems to be caused by precision errors rather than the barycenters actually having a small offset.

3.5 Normalising shape orientations

In the next step, the orientation of the models is normalised. In order to normalise the orientation, the eigenvectors that represent the

spread of the object must be computed. The goal of the orientation normalisation is to align the major eigenvector with the x-axis of the reference coordinate system and the respective medium eigenvector with the y-axis of the reference coordinate system. This means that after normalisation, the axis on which the largest spread of points on the mesh is located, is the same as the x-axis.

To compute the eigenvectors representing the spread of points on the mesh, Principle Component Analysis (PCA) is used. This is done using the NumPy Python library [2]. First the `numpy.cov` method [6] is used to compute the covariance matrix of the shapes' vertices. Thereafter, the covariance matrix is used as input for the `numpy.linalg.eig` method [7], which computes the major, respective medium and minor eigenvectors, and their corresponding eigenvalues.

To align the shape with the reference coordinate system axis, projections are used. With p_i being the unaligned shape coordinate and the major eigenvector e_1 and respective medium eigenvector e_2 , the formula that updates their position is:

$$\begin{aligned} x_i^{\text{updated}} &= p_i \cdot e_1 \\ y_i^{\text{updated}} &= p_i \cdot e_2 \\ z_i^{\text{updated}} &= p_i \cdot (e_1 \times e_2) \end{aligned} \quad (2)$$

The barycenter is not part of the formula, because all shapes are normalised to have their barycenter at the origin of the reference coordinate frame.

After normalisation most meshes align properly. Figure 4 shows an example of a mesh of which the orientation is aligned properly after normalisation. This is however not always the case. Some meshes are not properly aligned using this method. Figure 5b shows an example of this. This issue could be the result of the meshes not having uniformly distributed vertices. Figure 6a shows the mesh of an object that was normalised properly when using the vertices as input for the covariance matrix. It manifests a mesh that is fairly evenly distributed in terms of vertex positions. Figure 6b on the other hand displays a mesh that does not have a uniform distribution of vertices on the mesh. It has two very dense areas where the eyes are located. This causes the the PCA to return this as the most distributed axis as shown in Figure 5b.

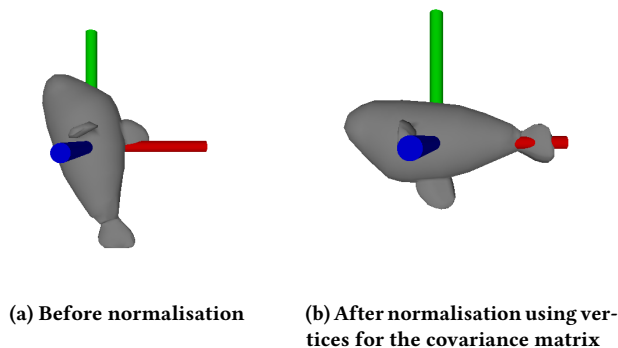


Figure 4: Object m58 orientation normalisation results

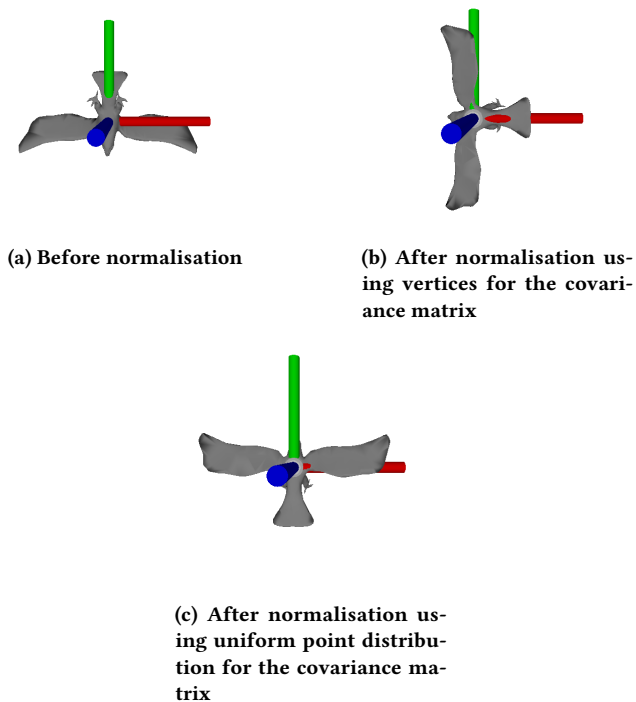


Figure 5: Object m45 orientation normalisation results

To solve the issue that meshes with non uniformly spread vertices cause when computing the eigenvectors, the mesh can be sampled uniformly. Each triangle can be sampled uniformly with a number of points that correspond to the size of the triangle. Let v_0, v_1, v_2 , be the three vertices of an arbitrary triangle t . Let r_1 and r_2 be random floats in the range $[0, 1]$. The formula [8] that uniformly generates a random point on a triangle is:

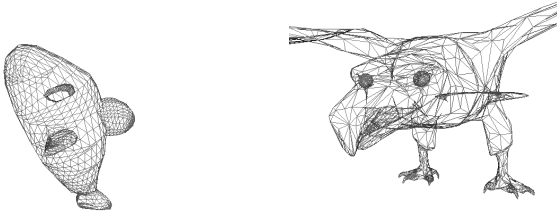
$$p = (1 - \sqrt{r_1})v_0 + (\sqrt{r_1}(1 - r_2))v_1 + (r_2\sqrt{r_1})v_2 \quad (3)$$

To generate the new eigenvectors, the uniformly distributed points on the mesh are used as input for the covariance matrix. As a result, a lot of meshes with the previously described issues, align properly when orientated using these newly computed eigenvectors. Figure 5c shows how the model that was not normalised properly is now aligned perfectly with the axis of the reference frame.

After using uniformly distributed points some meshes still could not be aligned properly. Figure 7c shows an example of this. Figure 6c shows that this mesh contains a lot of overlapping faces. Since the surface of the mesh is uniformly sampled, these areas are still sampled more than other areas, causing the spread of points to not be properly computed.

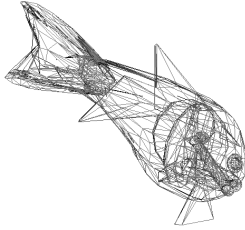
Having some meshes not aligned properly should not impact future work too much. It can impact the computation of the oriented bounding box (OBB) and the eccentricity for these objects. This effect will not be too high since the values seem to still be close to the expected value. Only if the orientation is way off, the deviating results will cause bigger issues. Another minor issue could arise in terms of user friendliness of the program, when the shapes

are displayed and the object is oriented such that it cannot be recognised immediately.



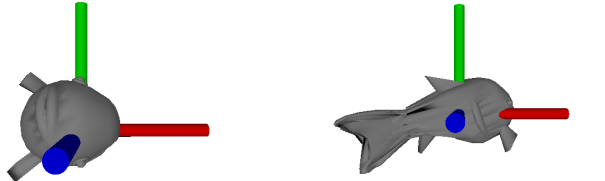
(a) Wireframe of object m58

(b) Wireframe of object m45



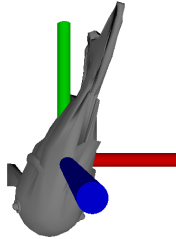
(c) Wireframe of object m66

Figure 6: Mesh wireframes



(a) Before normalisation

(b) After normalisation using vertices for the covariance matrix



(c) After normalisation using uniform point distribution for the covariance matrix

Figure 7: Object m66 orientation normalisation results

3.6 Flipping test

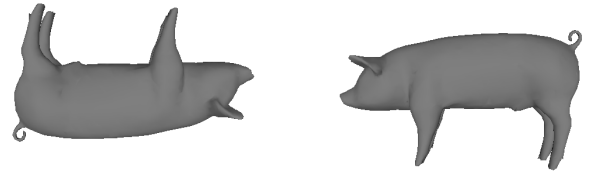
To make sure all shapes are oriented the same, some shapes have to be flipped. This is important when the shapes are displayed. When we take horses, visually, it makes most sense when all legs point downwards. To accomplish this, for each axis, the side with most mass is computed. When most mass resides on the negative side of the axis, the shape is mirrored with respect to that axis. During the flipping test, a variable f_i is computed along each axis, where $i = 0$ equals x , $i = 1$ equals y , and $i = 2$ equals z . The two values of f_i can either be -1 when the object needs to be mirrored along the respective axis, or 1 , when the object does not need to be mirrored with respect to that axis. This value is computed using the following formula:

$$f_i = \sum_t \text{sign}(C_{t,i})(C_{t,i})^2 A_t \quad (4)$$

where the summation goes over all the mesh triangles t , $C_{t,i}$ is the i^{th} coordinate of triangle t its centroid, and A_t is the area of triangle t . To mirror the mesh when needed, the mesh can be scaled along all the axis using f_i :

$$\begin{aligned} x_i^{\text{updated}} &= x_i \text{sign}(f_0) \\ y_i^{\text{updated}} &= y_i \text{sign}(f_1) \\ z_i^{\text{updated}} &= z_i \text{sign}(f_2) \end{aligned} \quad (5)$$

Figure 9 shows object 392 before and after being flipped correctly.



(a) Before flipping

(b) After flipping

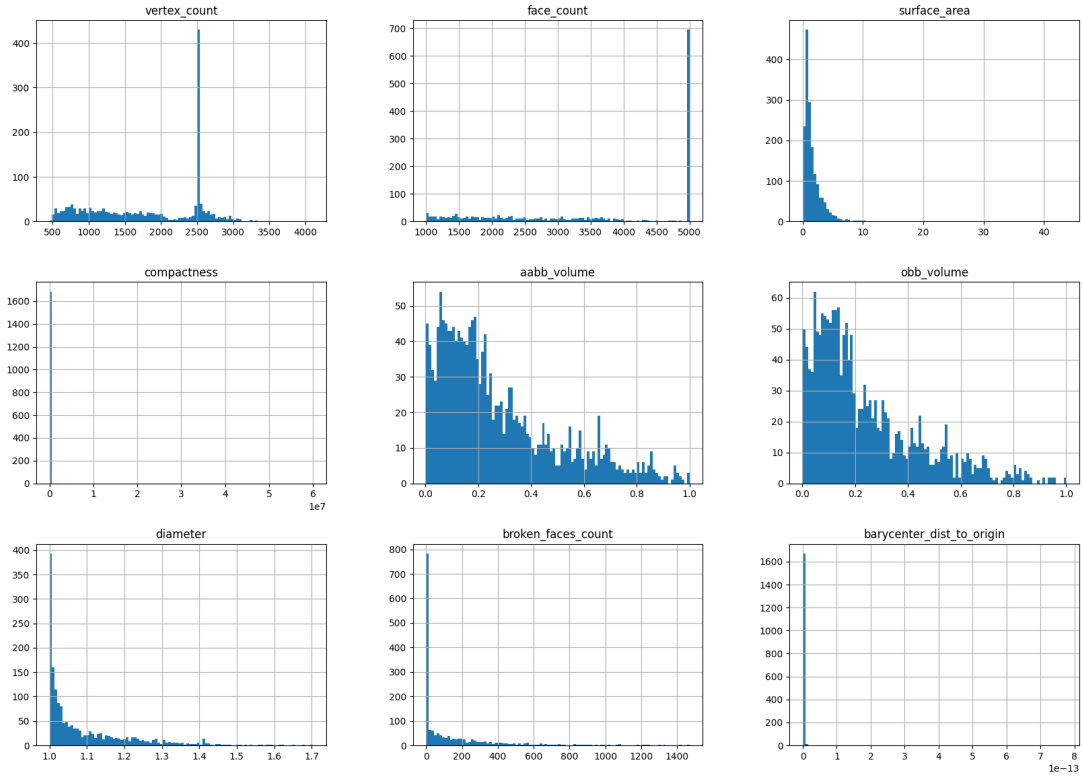
Figure 9: Object 392 flip result

3.7 Normalising shape scales

The scale of the model's axis-aligned bounding box must also be normalised. If this is not the case, metrics such as surface area cannot be used to accurately distinguish different types of models. Currently, the dataset contains some big outliers in terms of scale. The table in Figure 2b shows that the smallest AABB volume is 28 orders of magnitude smaller than the largest AABB box volume.

To normalise the scale, every model can be scaled to fit inside of a unit cube, where its axis-aligned bounding box sizes D_x , D_y , D_z have a length smaller or equal to one. To accomplish this, every object point p_i must be multiplied by a scaling factor σ as shown in formula 6.

$$\begin{aligned} D_{\max} &= \max(D_x, D_y, D_z) \\ \sigma &= 1/D_{\max} \\ p_i^{\text{updated}} &= \sigma \cdot p_i \end{aligned} \quad (6)$$



(a) Histograms of dataset features after normalising and mesh removal.

	No. vertices	No. faces	Surface area	Compactness	AABB volume	OBB volume	Diameter	No. broken faces	Distance to origin
Min	457	1000	0.041	1.09	2.03e-3	2.03e-3	1	0	8.58e-18
Max	4102	5000	43.70	2.20e+5	1	1	1.70	1470	7.75e-13
Mean	1810.98	3481.58	1.66	402.18	0.27	0.24	1.09	154.14	1.59e-15
Median	1865	3587.5	1.09	24.68	0.20	0.17	1.04	28	7.51e-16
Std	761.28	1474.44	1.90	6588.66	0.22	0.20	0.12	258.12	1.90e-14

(b) Dataset statistics after normalising and mesh removal.

Figure 8: Dataset (1686 models) characteristics after mesh normalisation and mesh removal. Respectively they show the number of vertices, number of faces, surface area, compactness, axis-aligned bounding box volume, oriented bounding box volume, diameter, the number of broken faces and the distance from the barycenter to the origin of the reference coordinate frame.

After normalisation, the AABB volumes seem to be distributed well. The table in Figure 8b displays a well distributed AABB volume and diameter. The table shows that the maximum AABB volume equals to one, which is the volume of a unit cube. The table also shows that the minimum and maximum diameters are 1 and 1.7. This is smaller than $\sqrt{3}$ (diagonal of a unit cube), which means that every shape fits inside of a unit cube. It is also bigger or equal to

one, which means that the objects with an AABB smaller than a unit cube, are also enlarged correctly.

4 STEP 3: FEATURE EXTRACTION

In this section, features that can be used to discriminate between different shapes will be extracted. The first subsection aims to extract global properties of the shape. These properties can be represented as a single real value. The proceeding subsection targets

shape property descriptors. These descriptors are distributions that can be represented as histograms.

4.1 Global properties

Variable	Equation	Definition
Volume	$\frac{1}{6} \sum_{t_i} (v_1 \times v_2) \cdot v_3 $	Calculation of the volume inside the mesh by adding the volumes of every tetrahedron formed by the triangle's vertices and the barycenter.
Compactness	$S^3 / (36\pi V^2)$	The relation between the surface area and the volume of an object.
3D rectangularity	V/V_{OBB}	Division of the volume of the object by its oriented bounding box volume.
Eccentricity	$ \lambda_1 / \lambda_3 $	The major eigenvector divided by the respective minor eigenvector.

Table 1: Calculations of the global descriptors.

The 3D shapes can be categorised by extracting the global properties, such as the surface area, compactness, 3D rectangularity, diameter and eccentricity. The equations used to calculate these descriptors are shown in Table 1. To check if the feature extraction was successful, two shapes which look very dissimilar are used to compare their global descriptors. The objects that are chosen to be compared are m100 and m702, see Figure 10, due to the expected dissimilarities in their global descriptors. The calculated values of these are displayed in Table 2.

The surface area is obtained by using the Trimesh library. It is expected that object m100 has a higher value compared to m702, which is confirmed in the table.

In the second column of the table, the compactness is displayed. The compactness of an object indicates how close its shape is to a sphere. The closer the compactness of an object is to 1, the more similar the object is to a sphere with the same volume. Because m702 is a long and thin object, the expectation is that the compactness will be higher than the compactness of m100, because it has a rounder shape. This is in line with the values in the table.

The calculation of the 3D rectangularity is performed to check how similar the volume of the object is to its oriented bounding box. To compute the OBB volume, the Trimesh library is used. Because the volume of the OBB will always be higher than the volume of the objects, the rectangularity will have a value between 0 and 1. In the table it is apparent that the rectangularity of m100 is higher than the value of m702, which means that the volume of m100 is closer to its OBB volume than the volume of m702. The reason for this is that there is more empty space in the OBB of m702 in comparison to the OBB of m100.

To obtain the diameter of the shape, the distances between all vertices is calculated and the longest distance between two points on the surface of the mesh is the diameter. Because the objects are

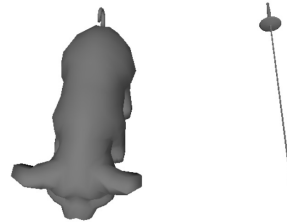


Figure 10: Object m100 and m702 (left and right respectively), which are used to compare the values of their descriptors.

both uniformly scaled, the diameters have a value within the range $[1, \sqrt{3}]$.

Finally, the eccentricity is calculated to obtain the spread of the objects. The eccentricity of m702 is higher than m100, which means that the differences between the major and minor eigenvalues are higher for m702.

	Surface area	Compactness	3D rectangularity	Diameter	Eccentricity
m100	1.28	3.12	0.32	1.06	3.87
m702	0.041	121.65	0.0067	1.00	26.80

Table 2: Global descriptors of object m100 and m702.

4.2 3D shape property descriptors

After obtaining the five global properties for all models, five shape property descriptors are computed. These descriptors are used to compare all meshes

- **A3** The angle between three random vertices
- **D1** The distance between the barycenter and a random vertex
- **D2** The distance between two random vertices
- **D3** The square root of area of a triangle given by three random vertices
- **D4** The cube root of volume of a tetrahedron formed by four random vertices

To calculate these descriptors, we use 10,000 samples for D1 and 100,000 for the other descriptors using random vertices for the calculations. The calculation of D1 is performed using less samples, because every object has a maximum of 5,000 vertices.

The values of A3 are computed using:

$$\theta = \arccos (v_1 \cdot v_2) \quad (7)$$

where θ is the angle and v_1 and v_2 the vectors from one random vertex to two other randomly chosen vertices. For 100,000 samples, this means that three random vertices are considered 100,000 times. In total, 300,000 vertices are considered whilst computing A3.

To compute D1, a distance method is used to obtain the Euclidean distance of 10,000 random vertices to the barycenter of the models.

D2 is measured by randomly selecting two vertices on the object and calculating the distance between them. Since 100,000 samples are taken, this means that in total 200,000 vertices are considered.

For the calculation of the square root area, equation 8 is used.

$$A = \sqrt{s(s - |v_1|)(s - |v_2|)(s - |v_3|)} \quad (8)$$

$$s = \frac{1}{2}(|v_1| + |v_2| + |v_3|) \quad (9)$$

with A the area, s the semiperimeter and v_1 , v_2 and v_3 the vectors between three randomly chosen vertices. For this calculation a total of 300,000 vertices are considered for 100,000 samples.

Finally, the cube root of volume of a tetrahedron is obtained using equation 10.

$$V = \sqrt[3]{\frac{1}{6}|(v_1 - v_4) \cdot ((v_2 - v_4) \times (v_3 - v_4))|} \quad (10)$$

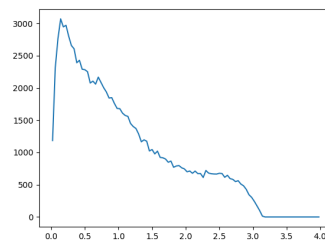
In this equation V is the volume of the mesh and v_1 , v_2 , v_3 and v_4 are the vectors which define the tetrahedron. Because 100,000 tetrahedra are considered, a total of 400,000 random vertices are chosen.

A histogram of every shape class containing all objects per class is created for every shape descriptor. The histograms are displayed in Appendix A.

To check whether the histograms are good, we compared the A3 histogram of two geometrically similar shapes and one geometrically different shape. The similar shapes are chosen from the Airplane and Bird class and a different shape from the Mech class, as displayed in Figure 11. As expected, the histograms of the plane and bird are very similar, due to their geometric similarities. The histogram of the mech has a very different shape, because the angles between three random points in a cube will be bigger compared to the angles in the plane and bird.

5 STEP 4: QUERYING

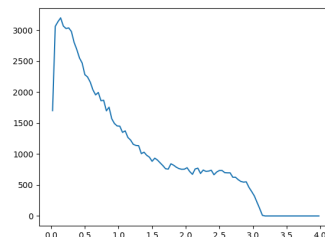
In this section, a Content-Based Shape Retrieval (CBSR) system is implemented, in which for a given query shape the best matching shapes are found. In the first subsection we describe how to normalise the ranges of all features, then the custom distance function is explained, and lastly we compare two distance functions. Before actually querying anything, a feature vector for the query shape is computed, using the global and shape descriptors. This means that the elements in the vector consists of the surface area, compactness, rectangularity, diameter, eccentricity, and the histograms of the shape descriptors A3, D1, D2, D3 and D4. Next, the feature vectors of the remaining shapes in the database are compared to the vector of the query shape. This is done by computing the distance between the query vector and the other vectors. In the last subsection, a comparison is given between the query results of our distance metric and a metric using only Euclidean distance.



(a) Histogram of the A3 shape descriptor of a shape in the Airplane class.



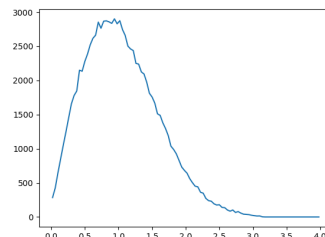
(b) A shape from the Airplane class.



(c) Histogram of the A3 shape descriptor of a shape in the Bird class.



(d) A shape from the Bird class.



(e) Histogram of the A3 shape descriptor of a shape in the Mech class.



(f) A shape from the Mech class.

Figure 11: Both plane and bird have roughly the same A3 shape descriptor histogram, and the mech is quite different, as expected.

5.1 Normalising feature ranges

As mentioned, the extracted shape features are used as components of the distance metric that indicates how dissimilar two shapes are from one another. In order to do this, the extracted features must be normalised. To account for large outliers, the global scalar features are normalised using standardisation. Let F be a scalar feature of a given shape. Then \bar{f} is the mean of that feature computed over the entire dataset and σ is the standard deviation of that feature. The normalised feature value of a given shape is then computed using the following formula:

$$F_{\text{normalised}} = \frac{F - \bar{f}}{\sigma} \quad (11)$$

The histogram features are normalised differently. Every bin b of a given histogram h is normalised to be within the range $[0, 1]$.

The normalised bin of a histogram is computed using the following formula:

$$b_{\text{normalised}} = \frac{b}{\int h} \quad (12)$$

where $\int h$ is the summation of all the bins of h .

5.2 Computing distances

After normalising the vectors, the distance between every element can be calculated. For the first five elements, containing global descriptors, a simple Euclidean distance (L_2) calculation is used:

$$L_2 = \sqrt{\sum_{i=1}^5 |v_1(i) - v_2(i)|^2} \quad (13)$$

where v_1 and v_2 are the two feature vectors that are compared. For the last five elements, containing the shape descriptor histograms, an Earth mover's distance (EMD) calculation is performed:

$$\text{EMD} = \frac{\sum_{ij} f_{ij} d_{ij}}{\sum_{ij} f_{ij}}, \text{ with } d_{ij} = |i - j| \quad (14)$$

with f_{ij} being the flow between the bins i and j of two histograms over distance d_{ij} .

After calculating the distances between the query vector and the feature vector of the other shapes in the database, these distances can be sorted from low to high. The lower the distance, the more similar a shape is to the query shape. Four examples of query shapes for which their most similar shapes are found, are presented in Figure 12. For the first two query shapes (teddybear and fish) our distance metric shows mostly good results with five of the six shapes being in the same class as the query shape. For the Cup class the results are even better since all the returned shapes are cups.

However, for the last query shape (octopus) the most similar objects are not in the same class. This is because the extracted features can not accurately describe an octopus. An octopus has a shape for which concavity should be included in the features, which has not been done.

5.3 Distance metric comparison

In Section 5.1 a combination of Euclidean distance and EMD was used to find similar shapes to a query shape. To compare the results of our "custom" method with the plain Euclidean norm, a query table is created using the same query shapes. The results are presented in Figure 13. The L_2 method looks worse in comparison with our distance metric used in Figure 12. For example, our distance metric for the first shape (teddybear) results in five other shapes which are in the same shape class as the query shape. Using the L_2 method results in only two shapes which are in the same shape class as the query shape. Furthermore, these two correct results do not have the shortest distance to the query shape. An explanation for why our results perform better than the L_2 results, is because a calculation of the distances between the shape descriptors is more accurate using EMD compared to Euclidean distance. This is because the EMD tries to fit the shape descriptor histograms (see Appendix A) of the query object with the other, by using the curvature of the distribution, instead of the exact pairwise bin levels. This negates issues when two histograms have the same shape, but are shifted slightly. The EMD is a more accurate measure for our purposes than the Euclidean distance since the shape of the histogram is more important than the shift.




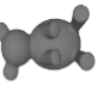


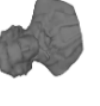

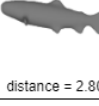
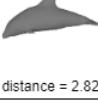
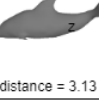
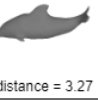
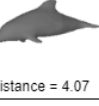
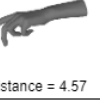














Query shape	1	2	3	4	5	6
	 distance = 2.09	 distance = 3.17	 distance = 4.49	 distance = 5.07	 distance = 5.65	 distance = 6.69
	 distance = 2.80	 distance = 2.82	 distance = 3.13	 distance = 3.27	 distance = 4.07	 distance = 4.57
	 distance = 2.83	 distance = 4.00	 distance = 6.12	 distance = 6.50	 distance = 8.89	 distance = 9.61
	 distance = 4.89	 distance = 5.62	 distance = 6.27	 distance = 6.49	 distance = 6.55	 distance = 6.63

Figure 12: Query results using our "custom" distance metric.





























Query shape	1	2	3	4	5	6
	 distance = 0.23	 distance = 0.29	 distance = 0.37	 distance = 0.38	 distance = 0.38	 distance = 0.40
	 distance = 0.15	 distance = 0.15	 distance = 0.16	 distance = 0.16	 distance = 0.16	 distance = 0.17
	 distance = 0.17	 distance = 0.21	 distance = 0.25	 distance = 0.31	 distance = 0.35	 distance = 0.37
	 distance = 0.12	 distance = 0.13	 distance = 0.14	 distance = 0.15	 distance = 0.16	 distance = 0.17

Figure 13: Query results using Euclidean distance.

6 STEP 5: SCALABILITY

In order to find the most similar shapes we could calculate the distance between every shape any time there is a new query. However this would be slow with a large enough dataset. To circumvent this issue, we find the k-Nearest Neighbours (k-NN) using the Python library SciPy [11]. This is a fast implementation of the k-NN algorithm using a KDTree. In our relatively small dataset this did not matter that much as queries went from 0.18s to 0.11s on average when going from brute force to k-NN respectively. On larger datasets the KDTree structure should result in large performance increases since the query time complexity goes from $O(n)$ to $O(\log n)$.

6.1 Dimensionality reduction

To get greater insight into our retrieval pipeline we decided to generate t-SNE plots [4] (see Figure 14). Because the PSB consists of broad shape classes, we decided to continue our analysis and visualisation with the Labeled PSB (see Figure 15). Every class has been given a different colour to easily differentiate between them. If the shapes of the same class are clustered together, this means that likely a good query result will be given. While standard t-SNE plots could give insight into the quality of our distance metric by visualising clusters, it does not give an intuitive overview as to what the shapes look like, in these plots. So a t-SNE plot of all 3D rendered shapes of the Labeled PSB was also created. This plot is interactive, and a snapshot can be seen in Figure 16. The result of t-SNE has not been used as a pre-processing step before querying. This is done because performance was not an issue in the first place.

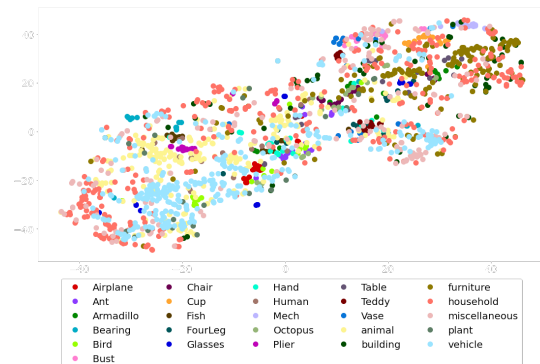


Figure 14: A t-SNE plot showing clusters of both the labeled dataset and the PSB. There is no distinction being made between the subclasses, so vehicle contains cars, hot air balloons, planes and more. That is why this t-SNE plot is messier than Figure 15.

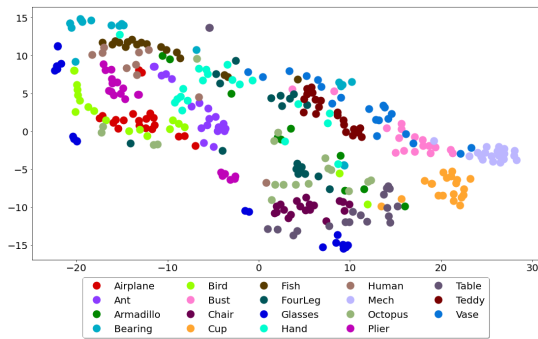


Figure 15: A t-SNE plot showing clusters of the labeled dataset.

7 STEP 6: EVALUATION

In this section, an evaluation of the CBSR system is given. To do this, the shapes' labels are used as ground truth for our classification problem. The PSB dataset did not come with very specific class labels. Both cars, helicopters and hot air balloons reside in the vehicle class. This means that returning a car when querying for a helicopter would result in a correct result if only the class labels are used to evaluate the results. This should not be the case since a car and a helicopter would be considered to be quite distinct from one

another. Contrary to the Princeton Shape Benchmark dataset, the Labeled PSB dataset does have specific labels with exactly twenty shapes per label. It contains, for example, a bird and octopus class instead of a generic animal class. The fact that every class has the same number of shapes makes for a fair, per class, evaluation.

7.1 Quality analyses

To evaluate the quality of the CBSR system, sensitivity and specificity will be used as quality measures, which have the following calculation:

$$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (15)$$

$$\text{Specificity} = \frac{\text{True Negative}}{\text{False Positives} + \text{True Negatives}} \quad (16)$$

These quality measures explain all the basic metrics - True Positives, False Positives, True Negatives and False negatives - in an way that can more easily be interpreted. An extended version of this, accounts for the fact that the query size in the CBSR system can change. This extended version is the Receiver Operating Characteristic (ROC) curve. The ROC curve also allows for a visual interpretation of the results. To allow for quick comparisons in quality between different parts of the system the Area Under the ROC (AUROC) will also be used.

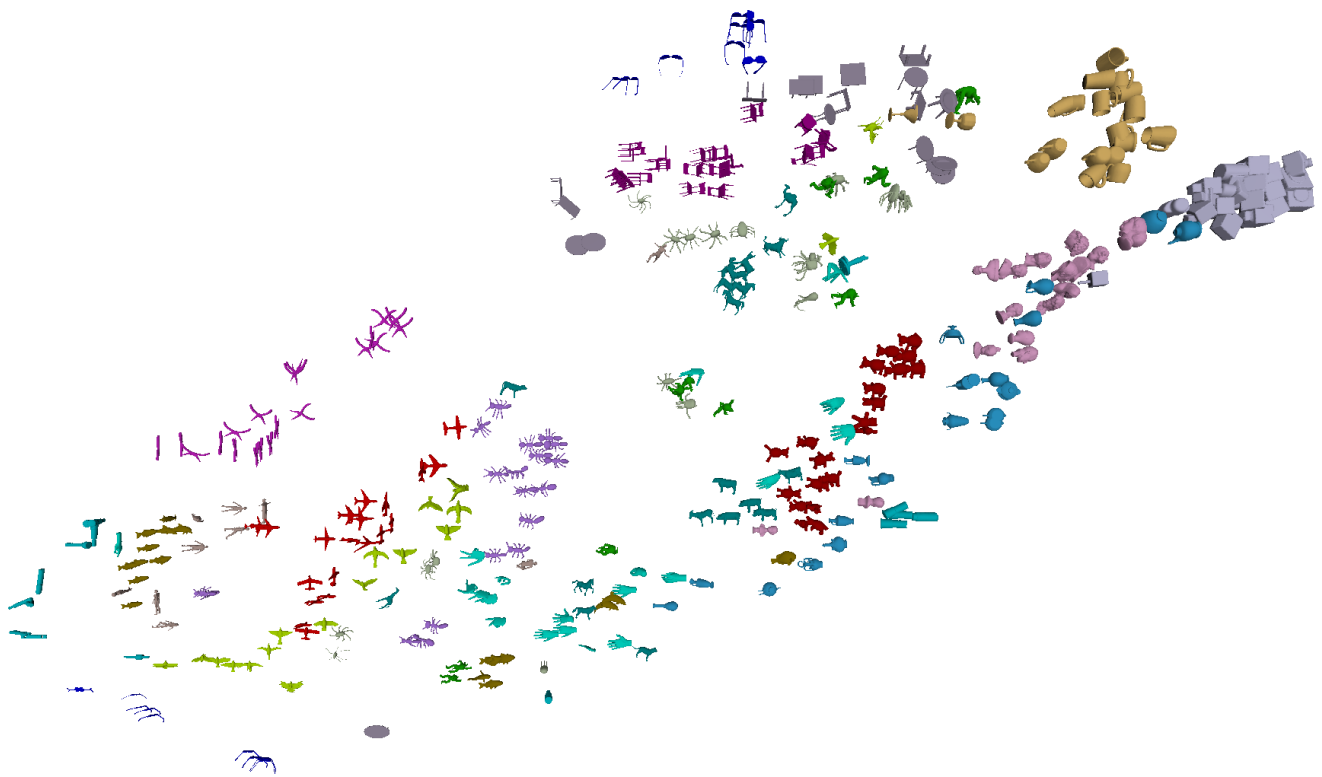


Figure 16: A t-SNE plot showing 3D shapes. Most notably, mechs (lilac/grey) are very clustered in the top-right, glasses (dark blue) are at the top-center and bottom-left, and octopuses (grey/green) are quite distributed throughout the dataset.

Figure 18 shows that the AUROC of the CBSR system as a whole is 0.82 (Table 3). The ROC curve indicates that the system favours specificity over sensitivity since the number of False Positives is minimised more than the number of False Negatives. The t-SNE plot in Figure 16 also shows that a lot of shapes from similar classes such as ants, cups and fishes, are clustered together.

Figure 17a shows that the Mech class performs really well with an average AUROC of 0.96. The shapes in the Mech class are all very similar in a topological sense. They are cubes with a tube coming out at one side. The Mech shape that performs worse than the others has a tube that is way thinner than the tubes of the other Mech classes. The t-SNE plot in Figure 16 also shows that all Mech shapes are located on the top right, with one Mech shape being located a bit more to the bottom left next to the shapes from the Bust class.

The CBSR system does not perform as well on the Glasses class (Figure 17b). Table 3 shows the AUROC to be 0.57. The ROC curve reflects this, as some glasses perform worse than returning random shapes for some values of k . When we study the t-SNE plot in Figure 16, two clusters of shapes from the Glasses class can be seen; one in the top middle and one in the bottom left. Figure 19a shows all the glasses in the dataset. It can be seen that some glasses have arms that almost align with the front frame, whilst other glasses have arms at an angle close to 90° . This causes a huge difference in eccentricity shown in Figure 19b. This difference also explains the two clusters in the t-SNE plot, where glasses with arms that are angled in a similar way are clustered together.

Shape	AUROC
Airplane	0.86
Ant	0.86
Armadillo	0.83
Bearing	0.60
Bird	0.69
Bust	0.90
Chair	0.91
Cup	0.75
Fish	0.87
Fourleg	0.86
Glasses	0.57
Hand	0.79
Human	0.76
Mech	0.96
Octopus	0.81
Plier	0.83
Table	0.82
Teddy	0.96
Vase	0.85
average	0.82

Table 3: The mean AUROC for each shape class from the Labeled Princeton dataset, with the global average.

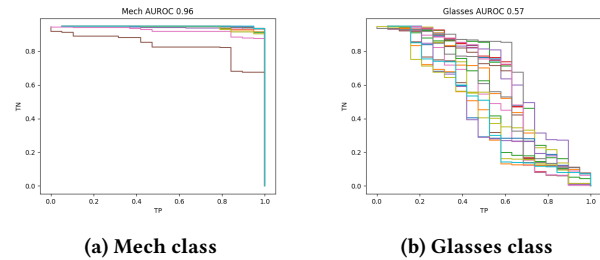


Figure 17: The class with the highest and lowest AUROC score, left and right respectively.

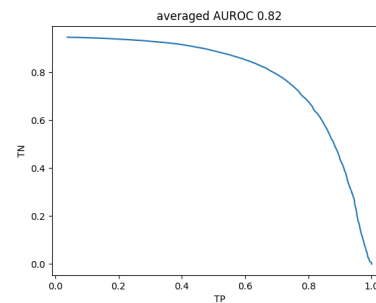


Figure 18: The shape of the mean ROC taken over all the shapes.

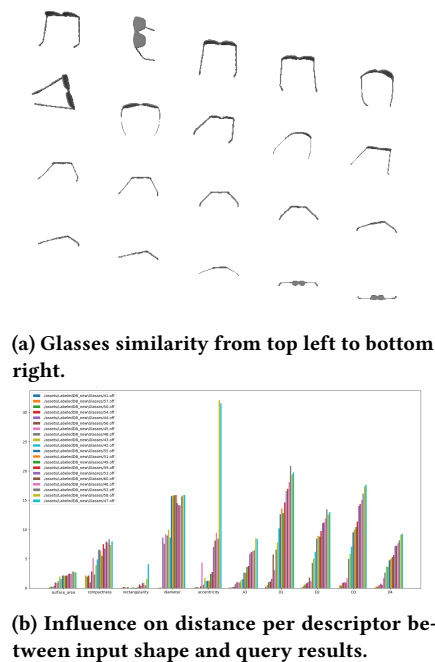


Figure 19: In subfigure (a) we see the results of a query, and in subfigure (b) the descriptors which influenced this query.

7.2 Distance visualisation

Some classes are significantly better than others as can be seen in Figure 17. Additional visualisations are made, specifically to find the causes of observed issues, for example, with the Glasses class. Namely, it is useful to see which features contribute to the final distance between two objects, this way it is possible to tweak features and gain insight in the results.

8 DISCUSSION

The CBSR system performs quite well overall, with an AUROC of 0.82. It seems to favour specificity over sensitivity. The averaged ROC curve over the whole system shows that the number of False Positives is minimised more than the number of False Negatives.

It can be seen that the CBSR system performs really well on shape classes that contain lots of shapes that are, in a topological sense, very similar. Examples are the Mech and Teddy classes. All shapes in these classes are very similar, as opposed to classes such as Glasses. Both the Mech and the Teddy class have an AUROC of 0.96 which means they almost perform perfectly on the tested dataset.

The CBSR system also has a few limitations. One of them is the discrimination between shapes that we would perceive as ‘different’, but have lots of topological similarities. An example of this is the Bird class. It deems a lot of planes to be more similar than other birds. The system currently does not take very refined features into account. Which could limit the quality of these search results.

Another limitation has the opposite problem. These are shapes that would be perceived to be similar by humans whilst having more topological differences. The Glasses class is an example of this. Some glasses are very elongated resulting in a high distance when compared to other glasses that could be perceived as similar. This seems to not only be a problem in our CBSR system, but a general problem MMR systems tend to have. A feature that in this specific problem could differentiate more between birds and planes is global curvature. Planes are all very similar in terms of curvature, which would result in different values for the birds.

The last major limitation comes from shapes that have topological properties which are not used as part of the distance function. The Octopus class is an example of this. These shapes have lots of concavities, which is a shape property that is not taken into account. Therefore shapes such as the octopuses tend to have poor query results.

8.1 Conclusion

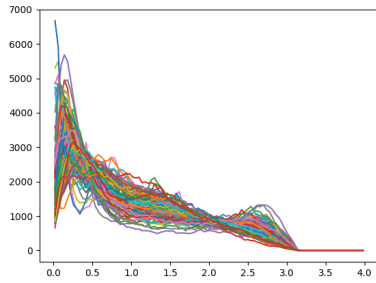
We presented a Content-Based Shape Retrieval system that manages to retrieve a given amount of shapes that are most similar to the query shape. For objects that have very distinct geometric features, the system performs really well. For other shapes that are topologically very similar to shapes from other classes or have features that the system doesn’t account for, the system does not yet perform optimally.

REFERENCES

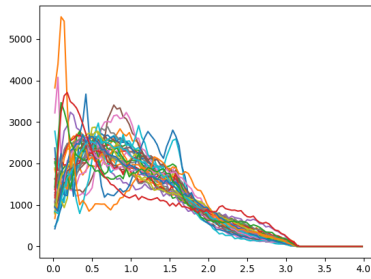
- [1] Michael Dawson-Haggerty. *Trimesh*. 2022. URL: <https://trimsh.org/>.
- [2] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. doi: 10.1038/s41586-020-2649-2. URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- [3] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. “Learning 3D Mesh Segmentation and Labeling”. In: *ACM Transactions on Graphics* 29.3 (2010).
- [4] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: <http://www.jmlr.org/papers/v9/vandermaaten08a.html>.
- [5] Matthew Matl. *Pyrender*. 2018. URL: <https://pyrender.readthedocs.io/>.
- [6] Numpy. *numpy.cov*. 2022. URL: <https://numpy.org/doc/stable/reference/generated/numpy.cov.html>.
- [7] Numpy. *numpy.linalg.eig*. 2022. URL: <https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html>.
- [8] Robert Osada et al. “Shape Distributions”. In: *ACM Transactions on Graphics* 21.4 (Oct. 2002), pp. 807–832.
- [9] *pyfqmr : Python Fast Quadric Mesh Reduction*. 2022. URL: <https://github.com/Kramer84/pyfqmr-Fast-Quadric-Mesh-Reduction>.
- [10] “The Princeton Shape Benchmark”. In: *Proceedings of the Shape Modeling International 2004*. SMI ’04. USA: IEEE Computer Society, 2004, pp. 167–178. ISBN: 0769520758.
- [11] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. doi: 10.1038/s41592-019-0686-2.

Appendices

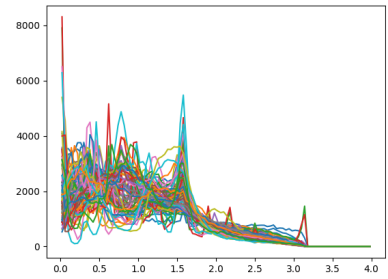
A SHAPE DESCRIPTORS



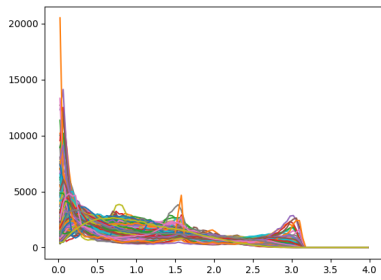
(a) Animal class



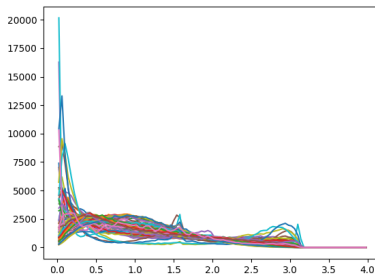
(b) Building class



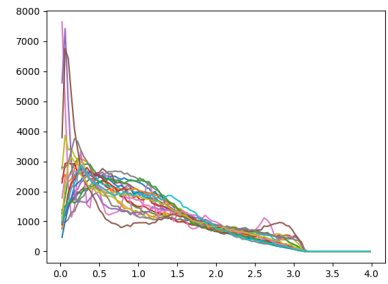
(c) Furniture class



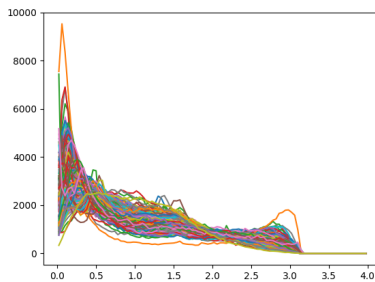
(d) Household class



(e) Miscellaneous class



(f) Plant class



(g) Vehicle class

Figure 20: A3 histograms of every sample class of the Princeton Shape Benchmark

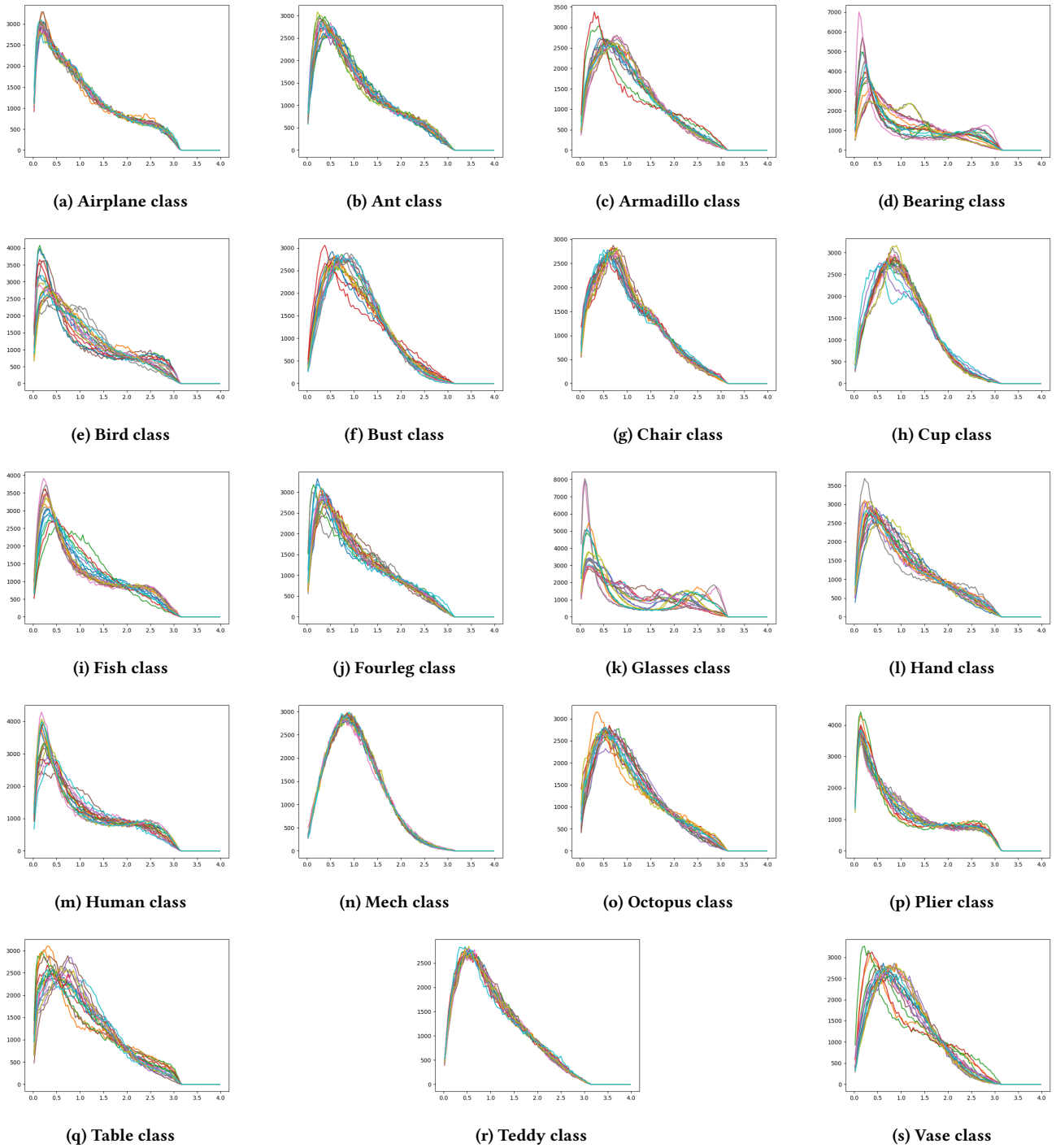
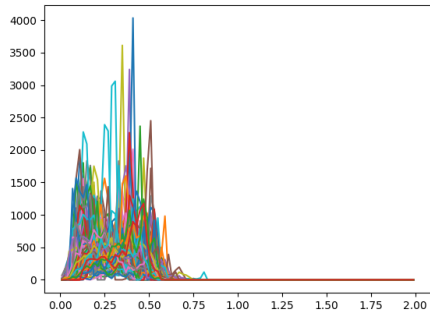
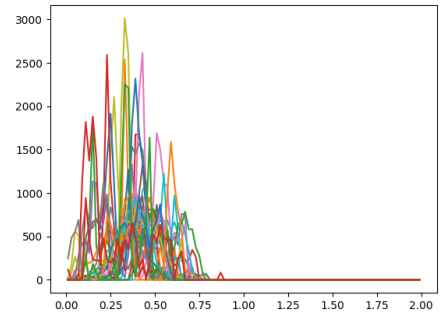


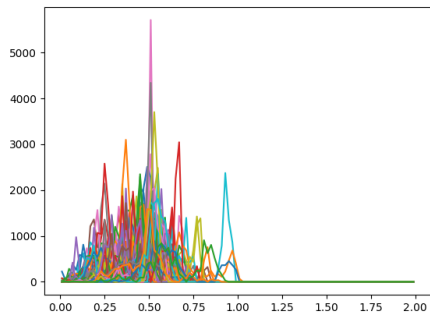
Figure 21: A3 histograms of every sample class of the Labeled Princeton Shape Benchmark



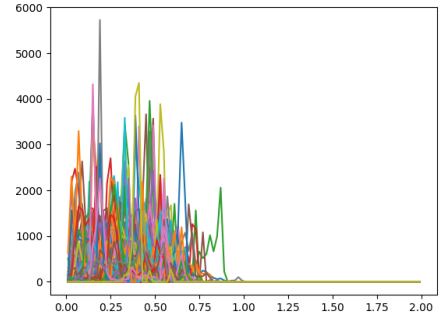
(a) Animal class



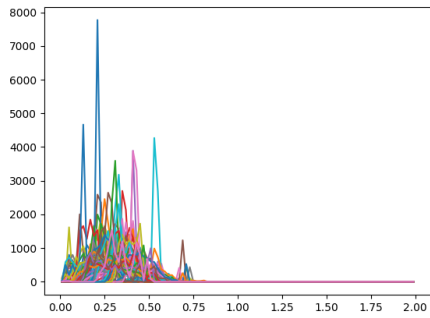
(b) Building class



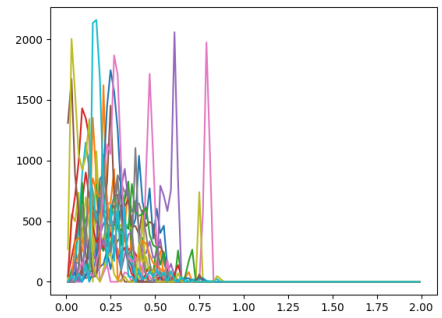
(c) Furniture class



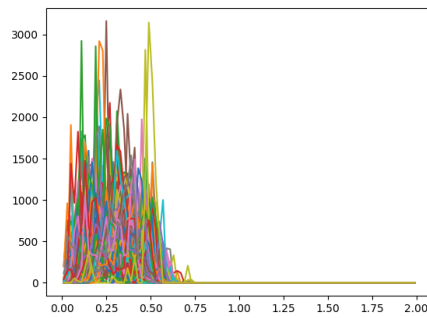
(d) Household class



(e) Miscellaneous class



(f) Plant class



(g) Vehicle class

Figure 22: D1 histograms of every sample class of the Princeton Shape Benchmark

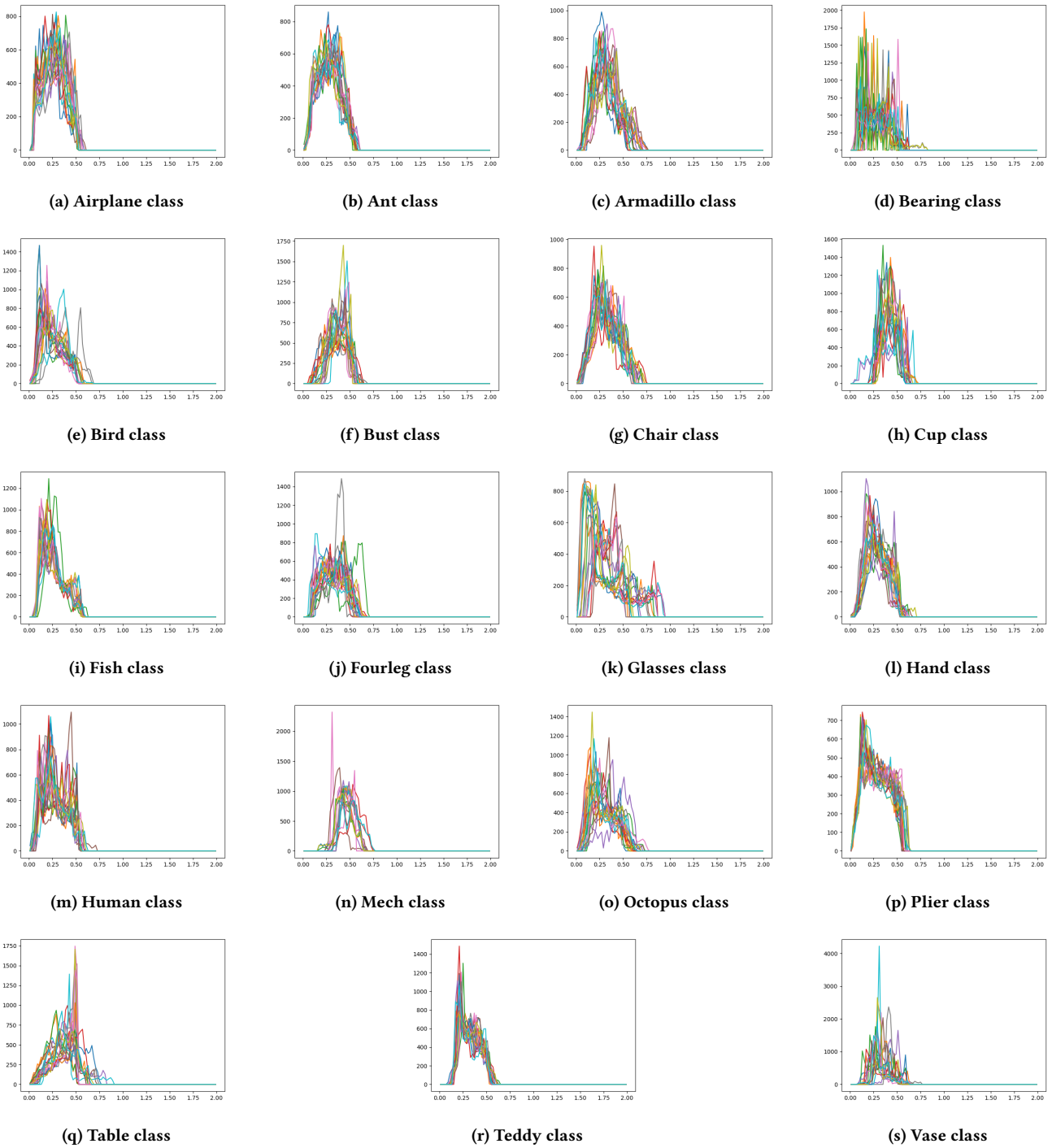
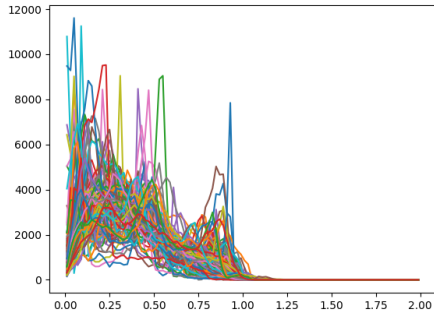
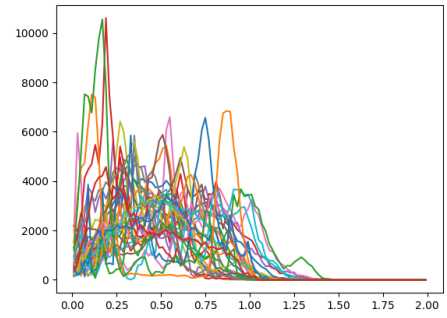


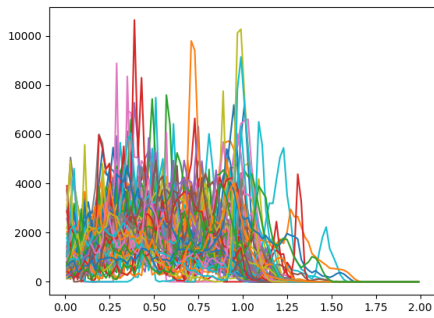
Figure 23: D1 histograms of every sample class of the Labeled Princeton Shape Benchmark



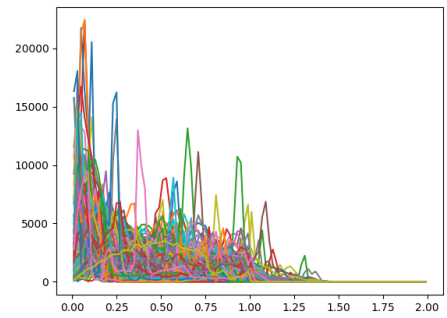
(a) Animal class



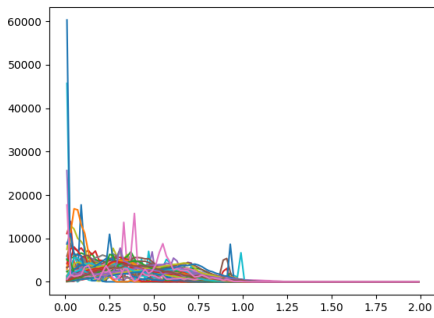
(b) Building class



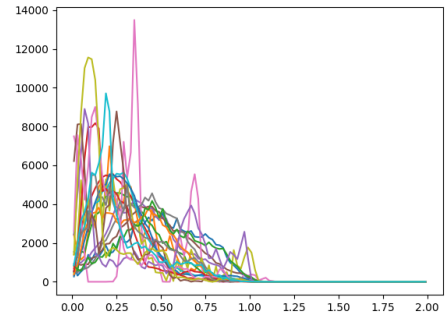
(c) Furniture class



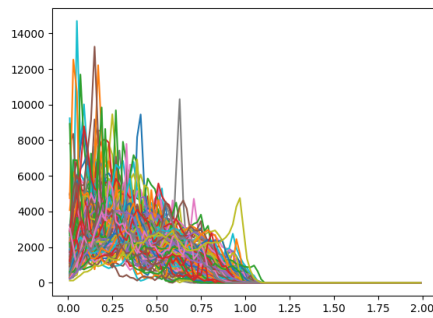
(d) Household class



(e) Miscellaneous class



(f) Plant class



(g) Vehicle class

Figure 24: D2 histograms of every sample class of the Princeton Shape Benchmark

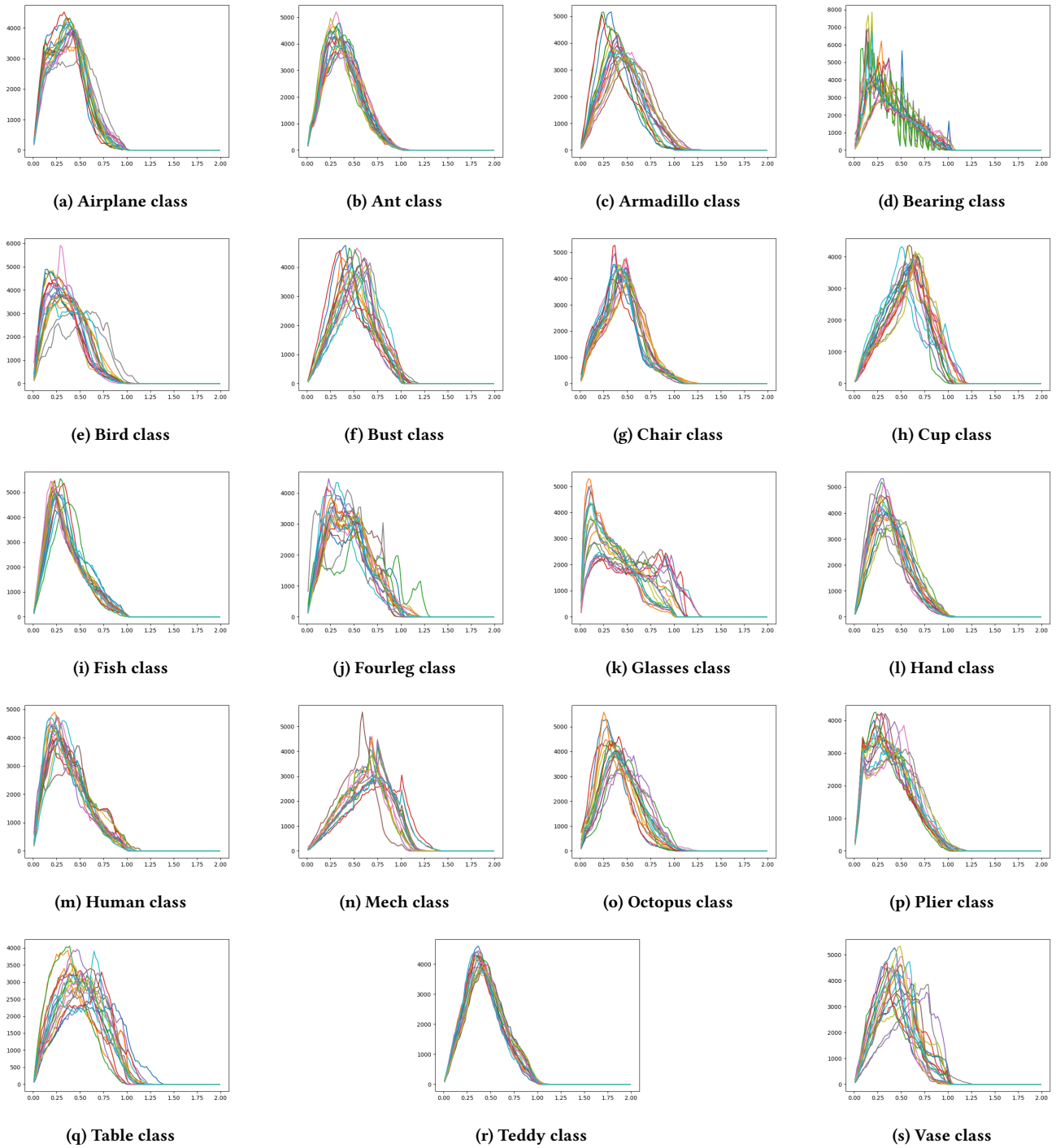
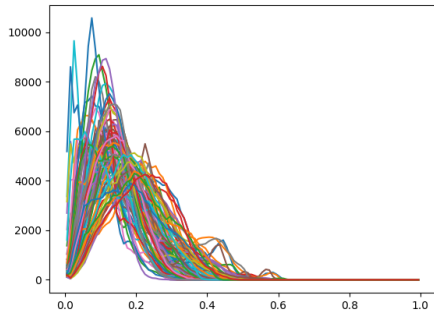
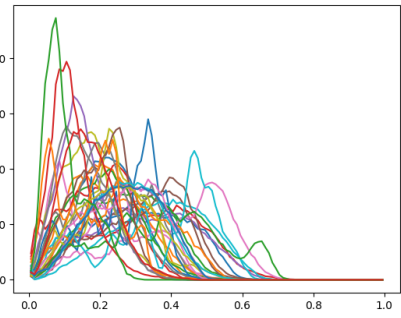


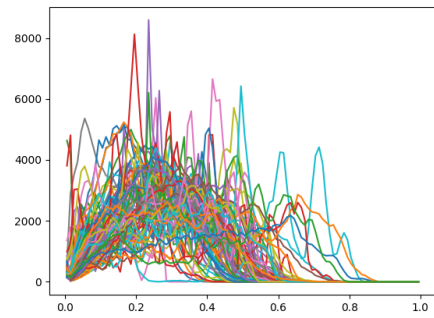
Figure 25: D2 histograms of every sample class of the Labeled Princeton Shape Benchmark



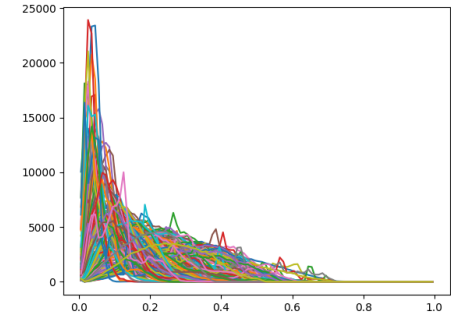
(a) Animal class



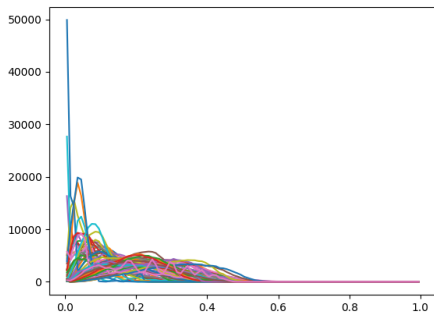
(b) Building class



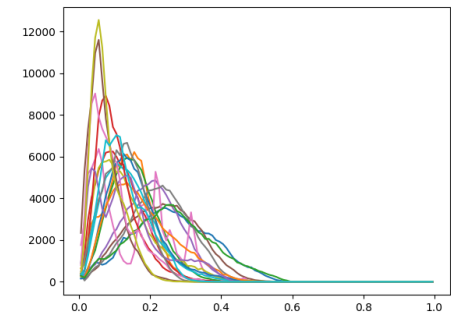
(c) Furniture class



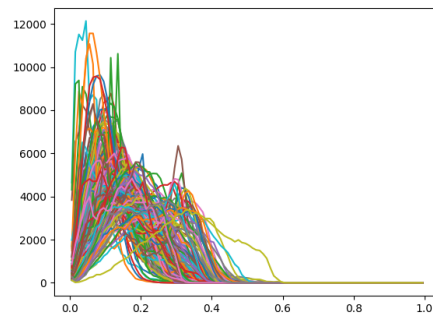
(d) Household class



(e) Miscellaneous class



(f) Plant class



(g) Vehicle class

Figure 26: D3 histograms of every sample class of the Princeton Shape Benchmark

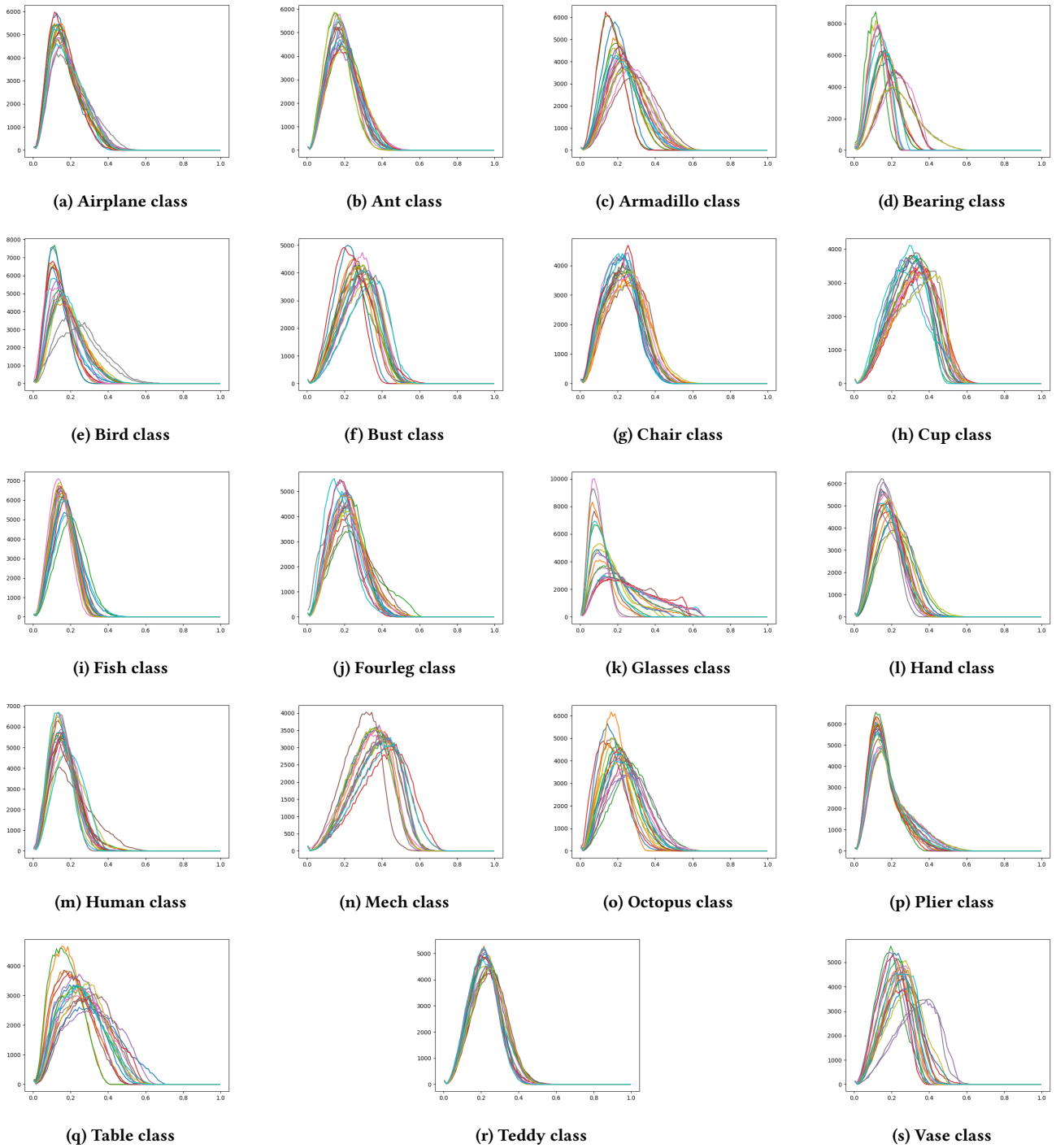
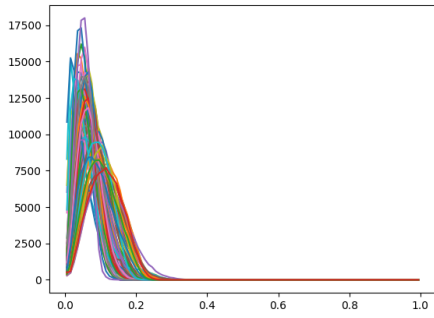
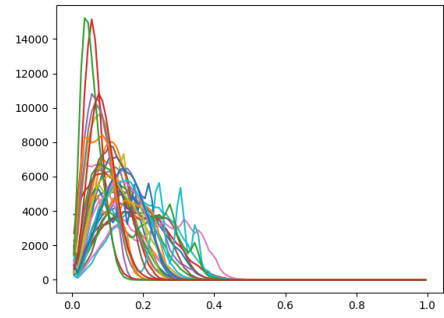


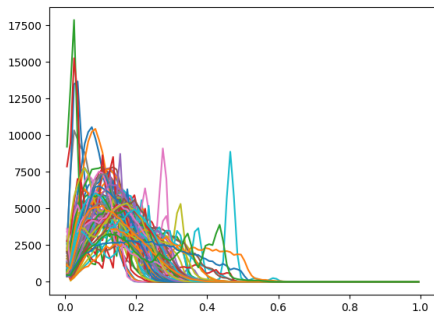
Figure 27: D3 histograms of every sample class of the Labeled Princeton Shape Benchmark



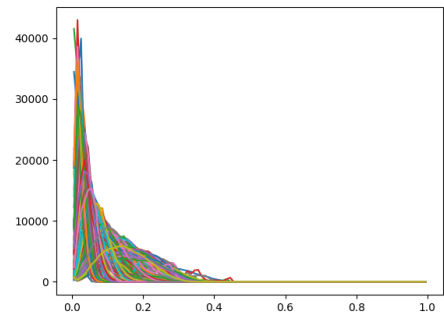
(a) Animal class



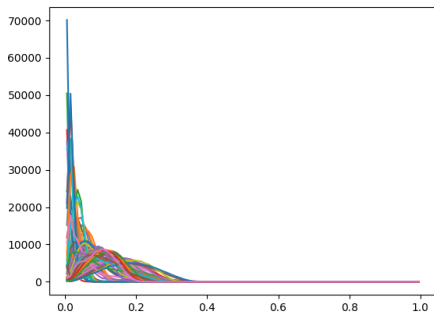
(b) Building class



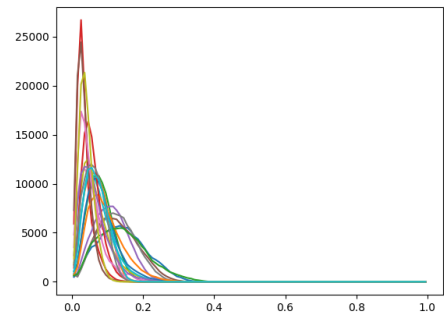
(c) Furniture class



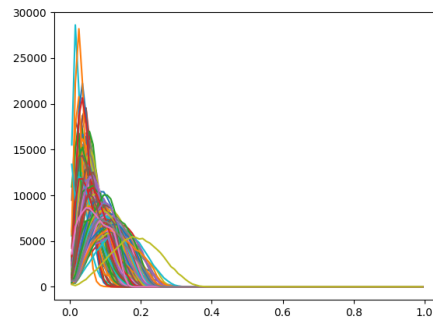
(d) Household class



(e) Miscellaneous class



(f) Plant class



(g) Vehicle class

Figure 28: D4 histograms of every sample class of the Princeton Shape Benchmark

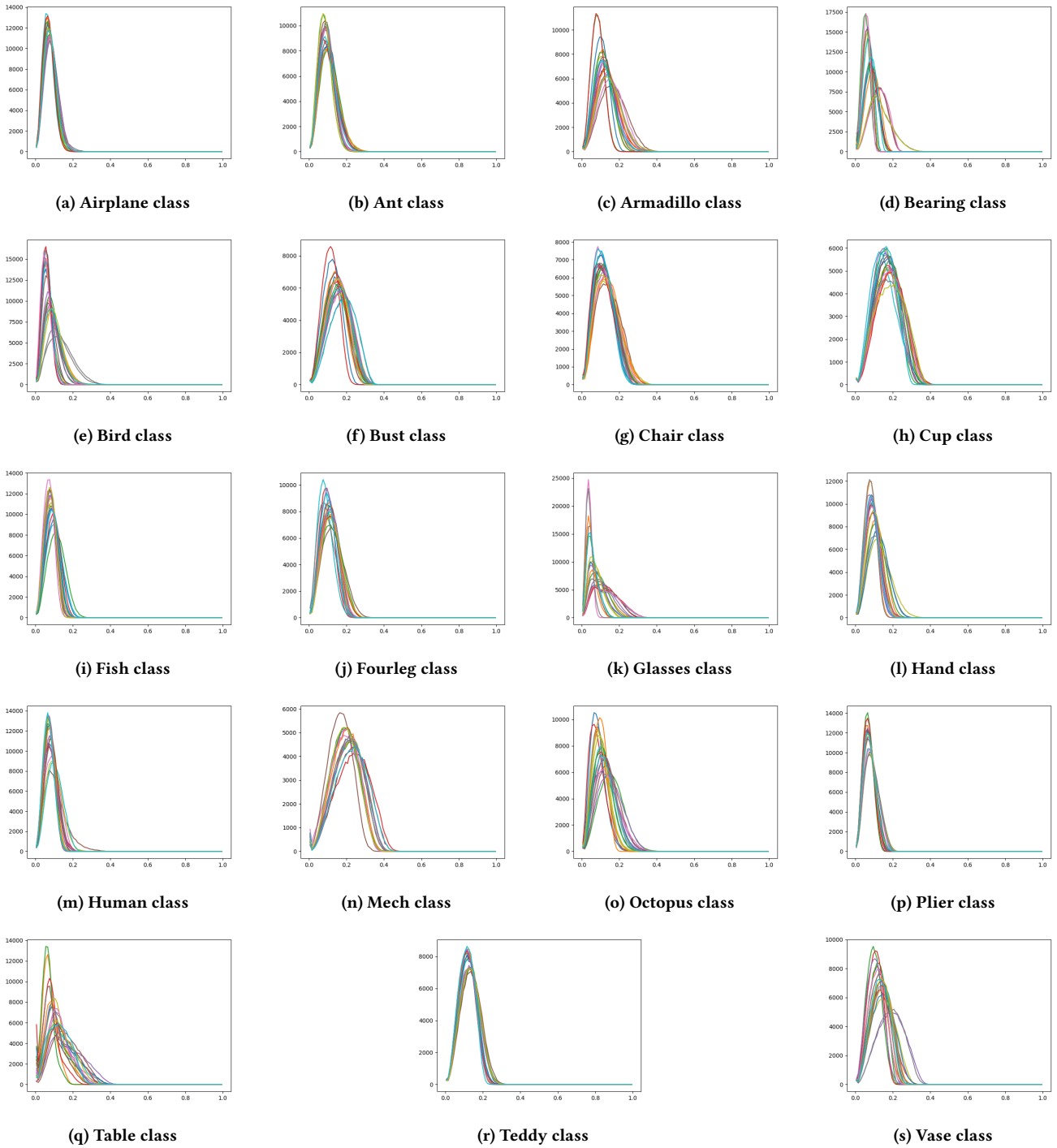
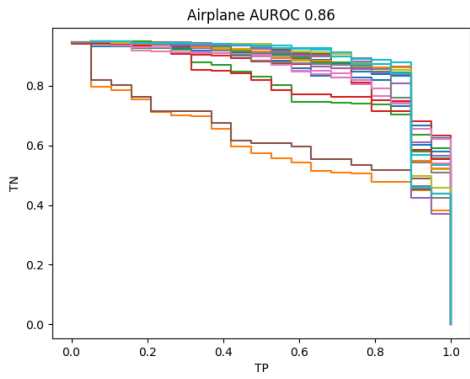
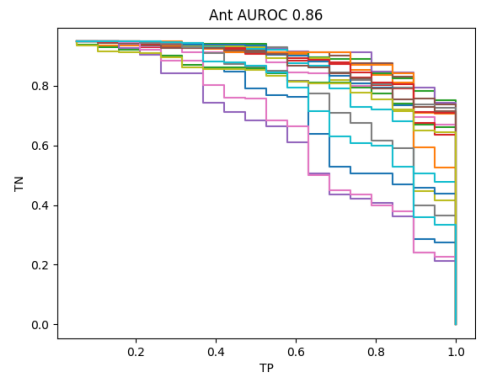


Figure 29: D4 histograms of every sample class of the Labeled Princeton Shape Benchmark

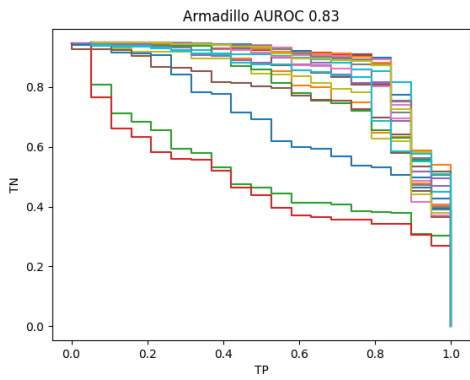
B RELATIVE OPERATING CHARACTERISTICS



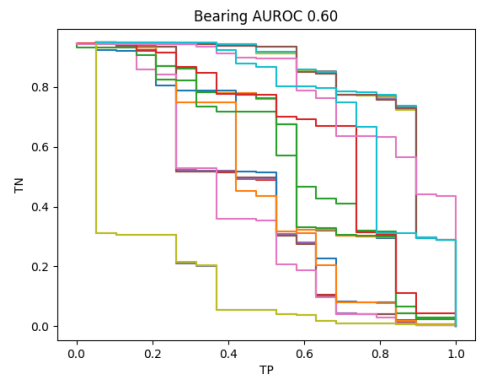
(a) Airplane class



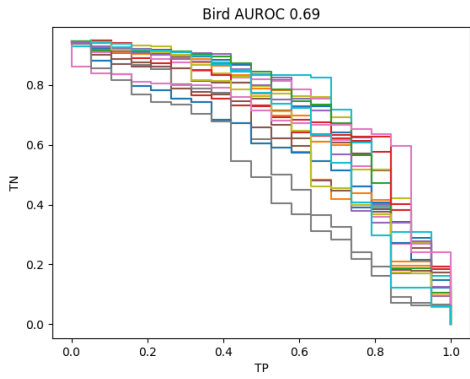
(b) Ant class



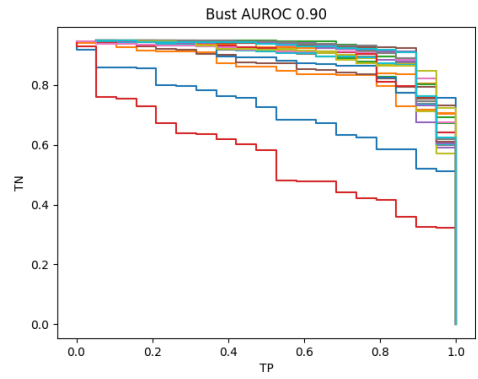
(c) Armadillo class



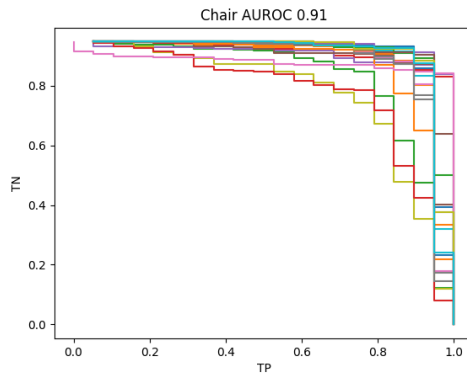
(d) Bearing class



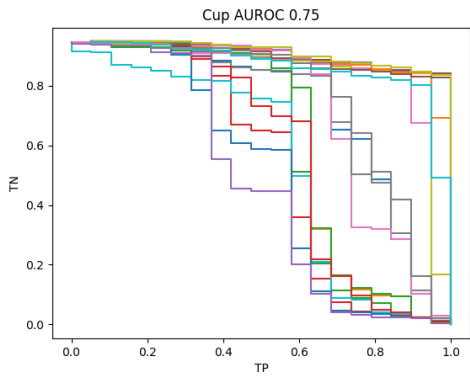
(e) Bird class



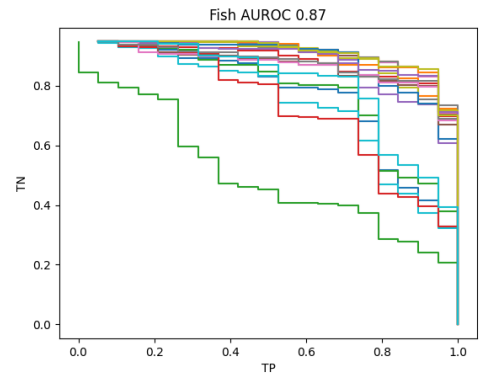
(f) Bust class



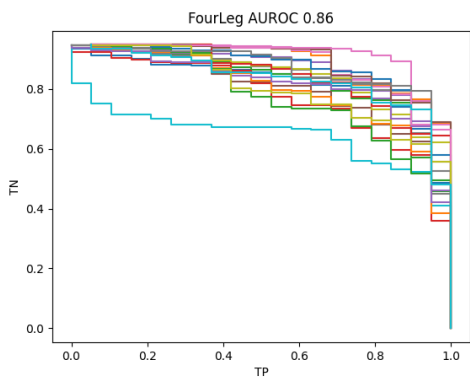
(g) Chair class



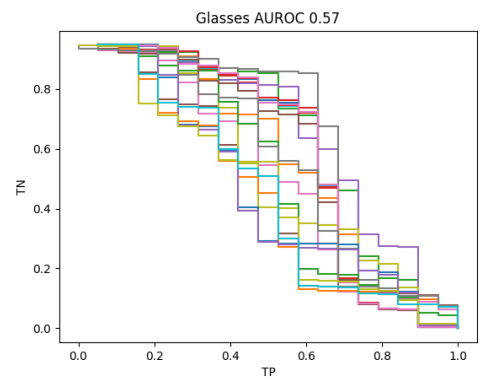
(h) Cup class



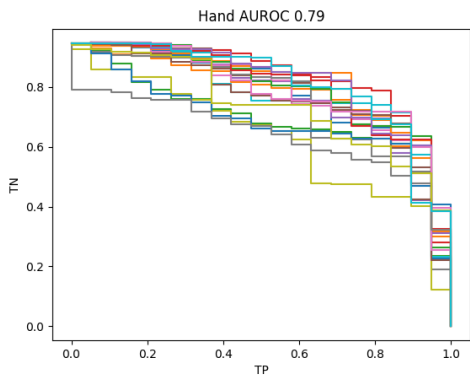
(i) Fish class



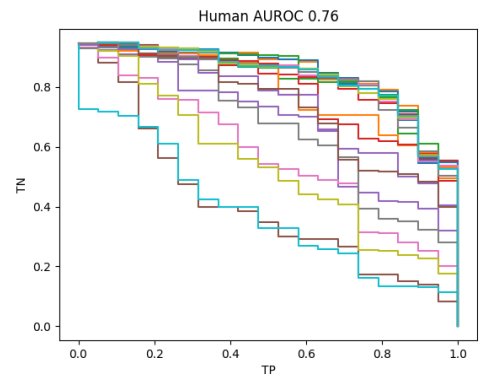
(j) Fourleg class



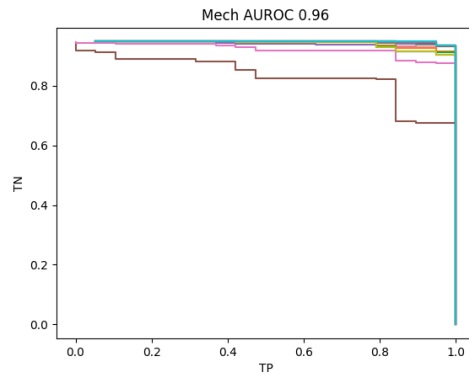
(k) Glasses class



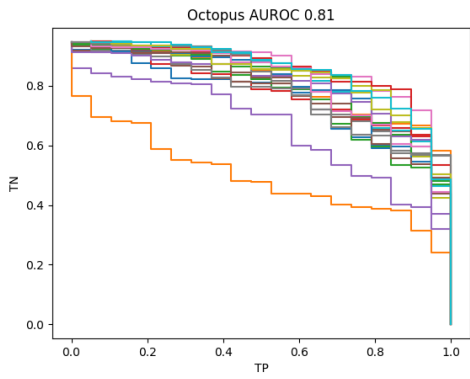
(l) Hand class



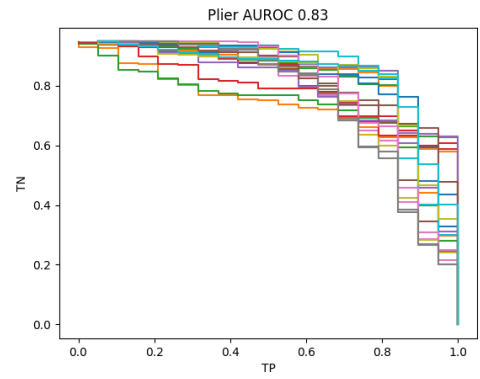
(m) Human class



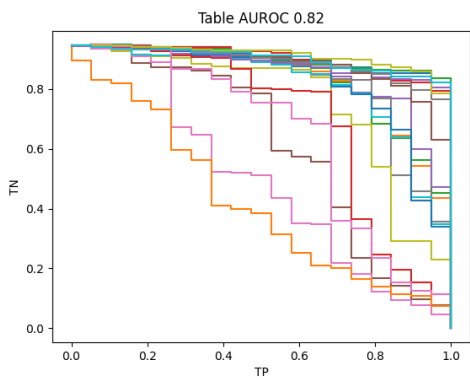
(n) Mech class



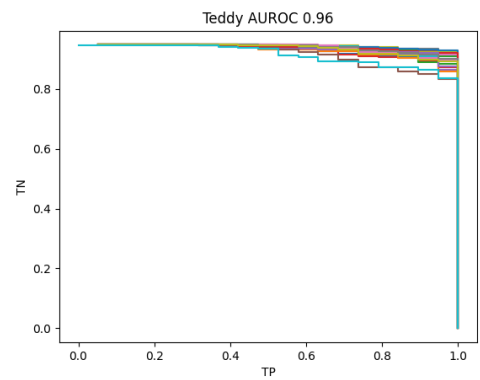
(o) Octopus class



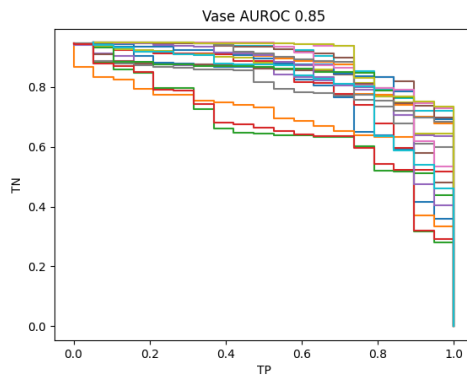
(p) Plier class



(q) Table class



(r) Teddy class



(s) Vase class

Figure 30: Relative Operating Characteristics of every sample class of the Labeled Princeton Shape Benchmark