Multimedia Retrieval Project Report

Leo Zeches 6856705, Jagmeet Singh 6848370, Daan van Laar 5518741

November 2020

Contents

1 Introduction

The following report outlines the implementation of a Multimedia Retrieval System. The goal of this system is to analyze and retrieve 3D shapes from a database, given a 3D shape specified by the user.

3D shapes come in the form of *off* or *ply* files. These file formats are very similar to each other; they both contain entries for vertices, their coordinates and faces. A vertex is a point in a 3D coordinate system, with values for the x, y and z coordinate. A face is a flat surface made up of either three (triangle) or four (quad) vertices. A face is determined by the indices of vertices that make up the face.

This report is structured into different steps, with each step adding additional features and implementations while building on top of the previous steps.

In section 2, a basic 3D model viewer is implemented to read and view the shapes in the database. In section 3, the 3D shapes are normalized so that they have a somewhat similar structure and so that features can be extracted correctly and efficiently. In section 4, the actual features are calculated and extracted from the 3D shapes.

In section 5, the approach used to perform queries is outlined, in addition to distance functions and first results which are also explained. In section 6, a different approach, called approximate-nearest-neighbour (ANN) is implemented and used to scale the system and perform queries faster for large databases. Additionally, a dimensionality reduction program is explained and implemented in order to visualize the multi-dimensional feature vectors in a 2D space. In section 7, the system is evaluated using different metrics and differences between implementations can be analyzed and highlighted.

Finally, the whole system is discussed in section 8. The highlights and drawbacks are outlined, and possible improvements and future work are also touched upon.

2 Step 1: Reading and Viewing the Data

The first part of creating a Multimedia Retrieval System is having a way of displaying and reviewing the data. In our case, the data consists of off-files from the "Labeled PSB Dataset" [KHS10], either in the "off" or "ply" format. In order to display these files, a very basic C++ program was created in Visual Studio, using OpenGL [Gro17] to create a window and displaying the connected vertices and faces of the 3D objects.

First, the off-files are read by the program by extracting the relevant information from the header, creating a grid object and populating it with vertex- and face data. For each vertex, this data consists of the x, y and z coordinates, and for the faces this data consists of the number of vertices included (3 for triangles, 4 for quads) and the indices of these vertices.

If the file is not in the "off" format, but instead in the "ply" format, the program uses a similar process and creates a off-file from the extracted ply-file data. This is rather straight forward, since these two file formats are structurally very similar.

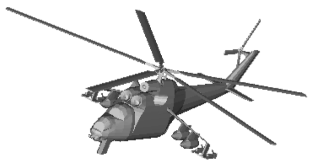


Figure 1: An example of a .ply file [Bur] displayed by the viewer.

Finally, basic OpenGL functions are used to draw the objects into the window. For this process, a tutorial for OpenGL was used which involves creating and drawing unstructured grids [Tel08]. Additionally, this tutorial also provided a guide for implementing basic functionality for visually inspecting the shape, namely rotating, panning and zooming. Using the keyboard, the program also allows to change how the shape is drawn, i.e. drawing only vertices, only outlines of faces, shading the faces, and drawing the shaded faces with a face grid on top.

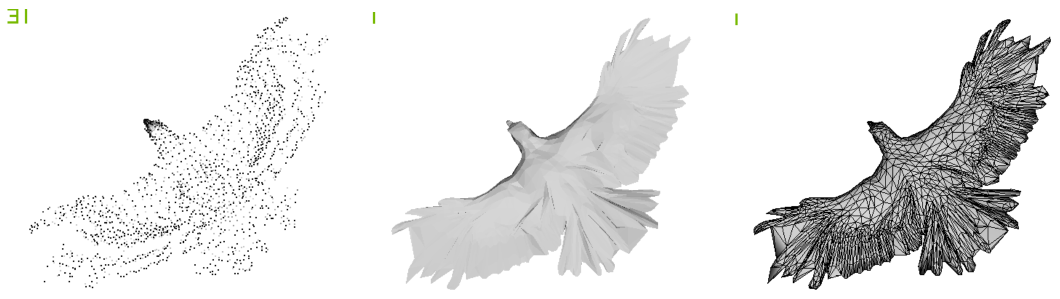


Figure 2: Example of different drawing styles. From left to right: Only vertices, shaded faces, shaded faces with grid.

3 Step 2: Preprocessing and Cleaning

For the second step, a reduced sample database was created to facilitate testing and debugging. This sample database has 188 entries, with about 10 different shapes per class. The shapes were inspected using the custom model view program and MeshLab[CNR20]. Some shapes in the labeled PSB database had problems which could not easily be fixed, like holes or overlapping faces. These shapes were not selected for the reduced sample database.

Afterwards, the program was extended to employ a simple filter which, for a given shape, outputs the class of the shape, the number of faces and vertices of the shape, the type of faces (triangles, quads or a mix) and the axis aligned bounding box of the shape. This information is then written into a csv file which can be easily accessed, extended or inspected. The filter uses the folder in which a shape is located to determine its class. The other properties of a shape are determined by scanning through the shape file and making note of the properties.

On starting the program, the csv file is located and read into program memory. Every shape that is then loaded is either added or updated to the filter. With a custom command, the current filter output will be saved back to the csv file. It is also possible to load the csv file manually.

Another command that was implemented scans the entire available database and extracts the data of each shape found to be saved. This is done by scanning from a folder in the directory, which makes this implementation work correctly with a growing database.

The data that is gathered by the filter contains the class of each shape, the number of faces and vertices, the types of those faces and the smallest box by which the shape is bound. The class of each shape is determined by the label given to it by the labeled PSB. The shapes are stored in folders of which the folder name is the name of the class. The amount of faces and vertices are read directly from the OFF file. The types of the face are determined while iterating through the faces of each individual shape. If they are all made up of 3, or 4 vertices, then the types are triangles and quads respectively . Otherwise, there's a mix. Lastly, the bounding box is determined by keeping track of the smallest and biggest coordinate of each axis. These form the 3D bounding box of each shape.

Next, the data needs to be normalized. Since there is a lot of variation between the different individual shapes in the database, these shapes need to be normalized to be used effectively later on. The order of normalization is as follows:

1. Translation
2. Scaling
3. Remeshing
4. PCA decomposition and alignment
5. Moment test

The order of these steps is not random, and choosing one order over another will bring some benefits and drawbacks to the later steps. Translation and scaling are done first in this case. Translating the shape to the origin will result in the barycenter coinciding with the origin of the coordinates (meaning the origin (0,0,0)) which will simplify steps like PCA decomposition. This results in the shapes being translation invariant.

The scaling normalization will not change the following steps, so it could in theory be done at any point of the normalization process. The same goes for remeshing.

The PCA rotation and moment test will come last and in this order. The moment test will ensure that the shapes will have consistent orientations, since PCA rotation will only enforce the correct directions of the shape, and not necessarily consistent orientations.

3.1 Translation

The shape is translated so that the barycenter of the shape (the point that is the average of all vertices in the shape; also called center of mass) coincides with the coordinate-frame origin. The barycenter is calculated by taking the average of each vertex coordinate:

$$x_{bary} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \text{ with } n = \text{ number of vertices in the shape, for each coordinate } x, y, z.$$

To translate the barycenter to the origin, the coordinate of the barycenter c are subtracted from each vertex p:

$$p_i^{updated} = p_i - c \text{ for every point } p_i \text{ in the shape.}$$

3.2 Scaling

The shape is scaled so that it will tightly fit into a unit-sized cube. For this, we define a factor as the maximum of the difference between the minimum and maximum x, y and z for each shape: factor $f = max((maxX - minX), (maxY - minY), (maxZ - minZ))$. Then each vertex p has its three coordinates p_x, p_y, p_z divided by this factor f:

$$p_{i,x}^{updated} = \frac{p_{i,x}}{f} \text{ for each coordinate } p_i \text{ and each coordinate } x, y, z$$

.

This results in shapes where their largest coordinates are on the limit of a unit-sized cube, with the rest of the shape fitting tightly into that cube.

These two initial normalization steps are performed on all of the shapes in the database, saving the normalized files in a new folder. A second csv file is generated, with the adapted values of the normalized files. Figures 3 and 4 show the effects of the two first normalization steps.

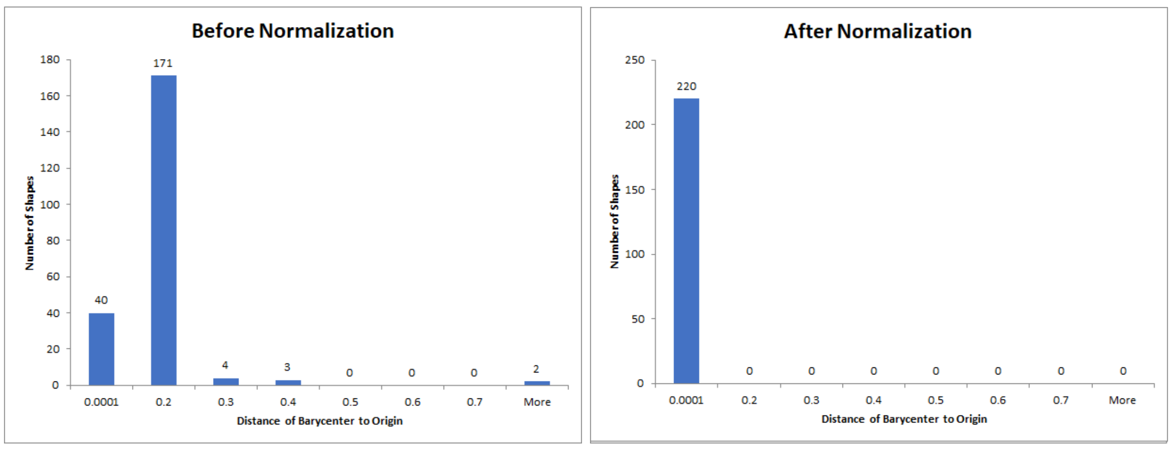


Figure 3: Results of the translation to the origin. The histogram shows the number of shapes with a certain distance to the origin. Left before normalization, right after.

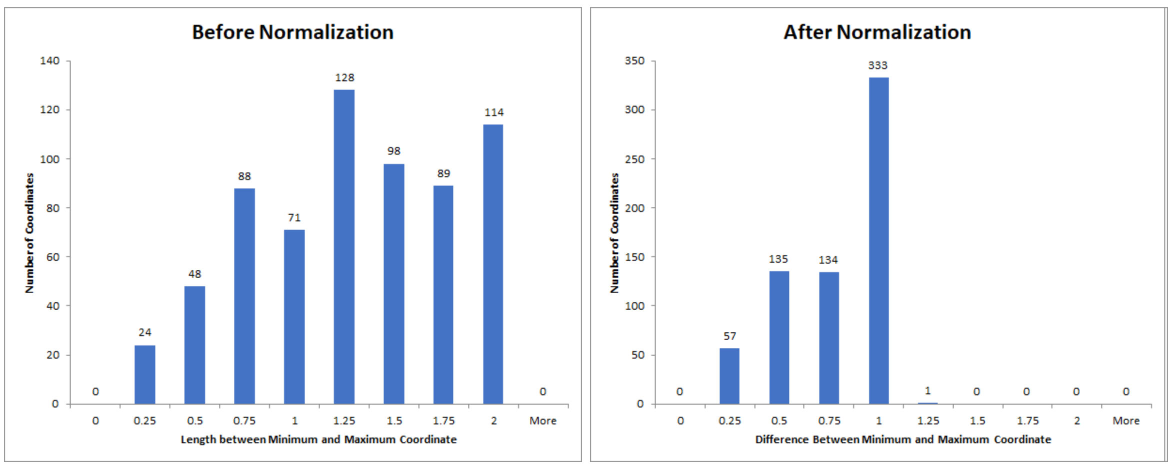


Figure 4: Results of the scaling into a unit-sized cube. The histogram shows the number of coordinates with a certain euclidian distance to the origin. Left before normalization, right after.

Next, outliers which have either too many or too little vertices and/or faces will have to be addressed. For feature extraction later on, it is important that the shapes have not too few, but also not too many vertices/faces. Too few faces will result in imprecise features, while too many vertices/faces will take a very long time to process. For this reason, we need to perform remeshing on outliers. We define a shape as an outlier if it has either less than 1000 vertices or more than 10000 vertices.

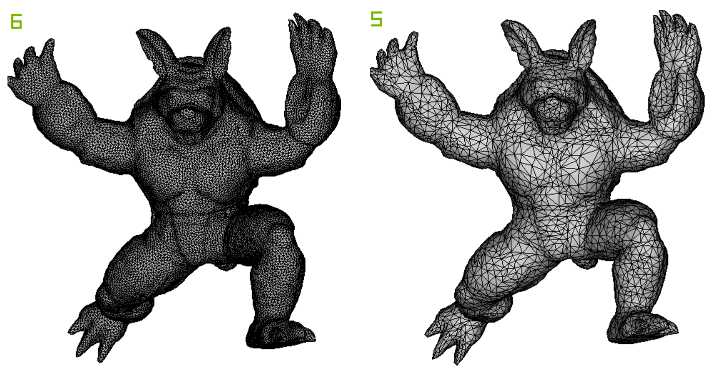


Figure 5: Example of the results of remeshing. The figure on the left has 25'273 vertices and 50'542 faces. The figure on the right has 5002 vertices and 10'000 faces.

At this stage, the remeshing is performed manually using *MeshLab* [CNR20]. The outliers are manually remeshed to have a number of vertices which are around 5000. In our sample database, 98 shapes have more than $10'000$ vertices, and are classified as outliers to be remeshed manually. Figure 6 shows the results of the remeshing process. If for any reason a mesh could not be remeshed manually or was still not watertight at this point, it was replaced with another shape or removed from the sample database entirely.

Finally, our sample database has the following shape: A total of 188 shapes distributed over 19 shape classes, with an average of 9.89 shapes per class. The shape of the final sample database is given in more detail in figure 7.

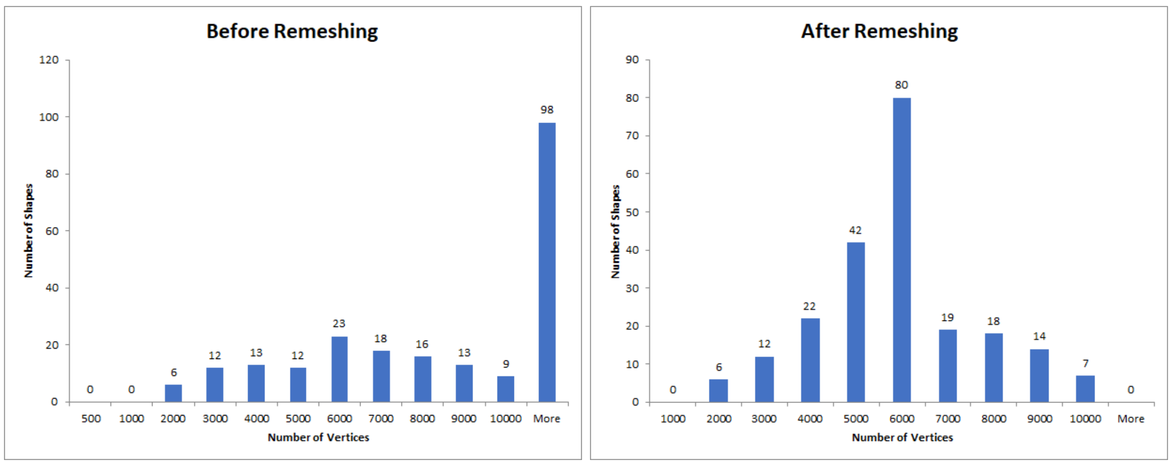
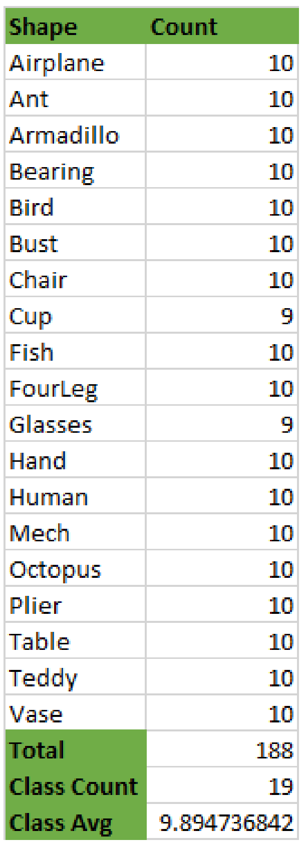


Figure 6: Results of the remeshing. The number of vertices of the outliers are reduced to around 5000.



Shape	Count
Airplane	10
Ant	10
Armadillo	10
Bearing	10
Bird	10
Bust	10
Chair	10
Cup	9
Fish	10
FourLeg	10
Glasses	9
Hand	10
Human	10
Mech	10
Octopus	10
Plier	10
Table	10
Teddy	10
Vase	10
Total	188
Class Count	19
Class Avg	9.894736842

Figure 7: The final shape of the sample database.

4 Step 3: Feature Extraction

In this step, the final steps of the normalization process are performed, before the features can be extracted in section 4.2. These features are computed on the normalized sample database. Each shape file is read in and has a number of features extracted (total of 75 individual feature values). They are stored in a csv file, together with additional relevant values (like standard deviations, means, and min/max values). These additional relevant values will be used in section 4.4 to normalize the features.

4.1 Normalization continued

First of all, the remaining nomalization steps are performed. These steps are the following:

1. PCA alignment
2. Flipping (Moment) Test

These two steps are performed to bring all of the shapes to a (more or less) equal orientation and thus result in rotation invariance.

4.1.1 PCA decomposition and alignment

The goal of the PCA decomposition is to rotate and align the shape in direction of the spread of the vertices. In other words, the direction of the spread, also known as the eigenvectors of the vertex cloud, should be aligned with the axes of the coordinate system. To compute this, the covariance matrix C is calculated:

$$C = \begin{bmatrix} \sigma(x,x) & \sigma(x,y) & \sigma(x,z) \\ \sigma(y,x) & \sigma(y,y) & \sigma(y,z) \\ \sigma(z,x) & \sigma(z,y) & \sigma(z,z) \end{bmatrix}$$

$$\text{with } \sigma(x,y) = \frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - \overline{x}) \cdot (y_i - \overline{y}), \text{where } \overline{x} \text{ resembles the average value of } x$$

.

After computing the covariance matrix, Eigen [JG18], a C++ template for linear algebra, is used to compute the eigenvectors. Using these vectors, the new coordinates for each vertex are obtained by calculating the dot product between the old vertex with respect to the shape barycenter (in this case the origin (0,0,0), since the shape has already been translated), and each eigenvector separately. This will result in the update x,y,z coordinates of the rotated shape:

$$p_{i,x}^{upated} = p_i \cdot eig_1$$
$$p_{i,y}^{upated} = p_i \cdot eig_2$$
$$p_{i,z}^{upated} = p_i \cdot eig_3$$

4.1.2 Moment Test

Because the previous pose alignment does not take into account orientation, we normalize the orientation in this step. We want the "bulkiest" part of the shape to be in the positive octant. For this, the second order moment of the shape's volume is calculated and the shape is flipped along x, y or z as needed. The formula to compute the transform matrix F is as follows:

$$F = \begin{bmatrix} sign(f_x) & 0 & 0 \\ 0 & sign(f_y) & 0 \\ 0 & 0 & sign(f_z) \end{bmatrix}$$

$$\text{with } f_i = \sum_t sign(C_{t,i}) \cdot (C_{t,i})^2 \text{ and } C_{t,i} \text{ being the i-th coordinate of the centroid of the face t.}$$

Each coordinate of each point p_i of the shape is multiplied with F, flipping the shape along the desired axis:

$$p_{x,i}^{updated} = p_{x,i} \cdot sign(f_x)$$
$$p_{y,i}^{updated} = p_{y,i} \cdot sign(f_y)$$
$$p_{z,i}^{updated} = p_{z,i} \cdot sign(f_z)$$

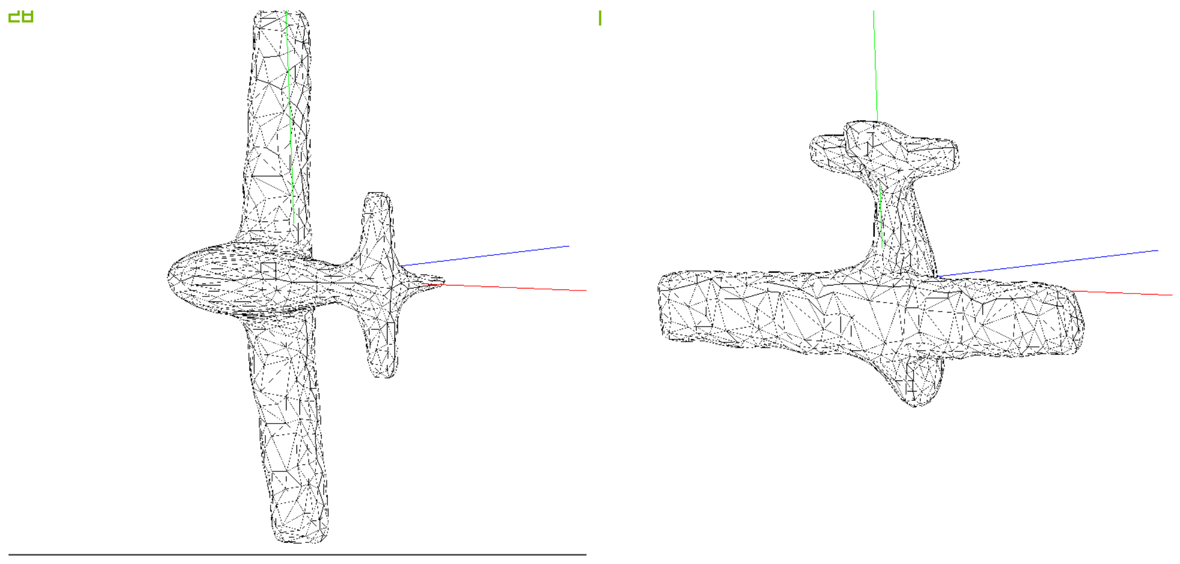


Figure 8: Results of the rotation and the moment test. The x-axis is red, y-axis is green, and z-axis is blue. The second figure has his eigenvectors aligned along the main axes, and is then flipped so that the main "mass" of the shape lies left of the respective axis.

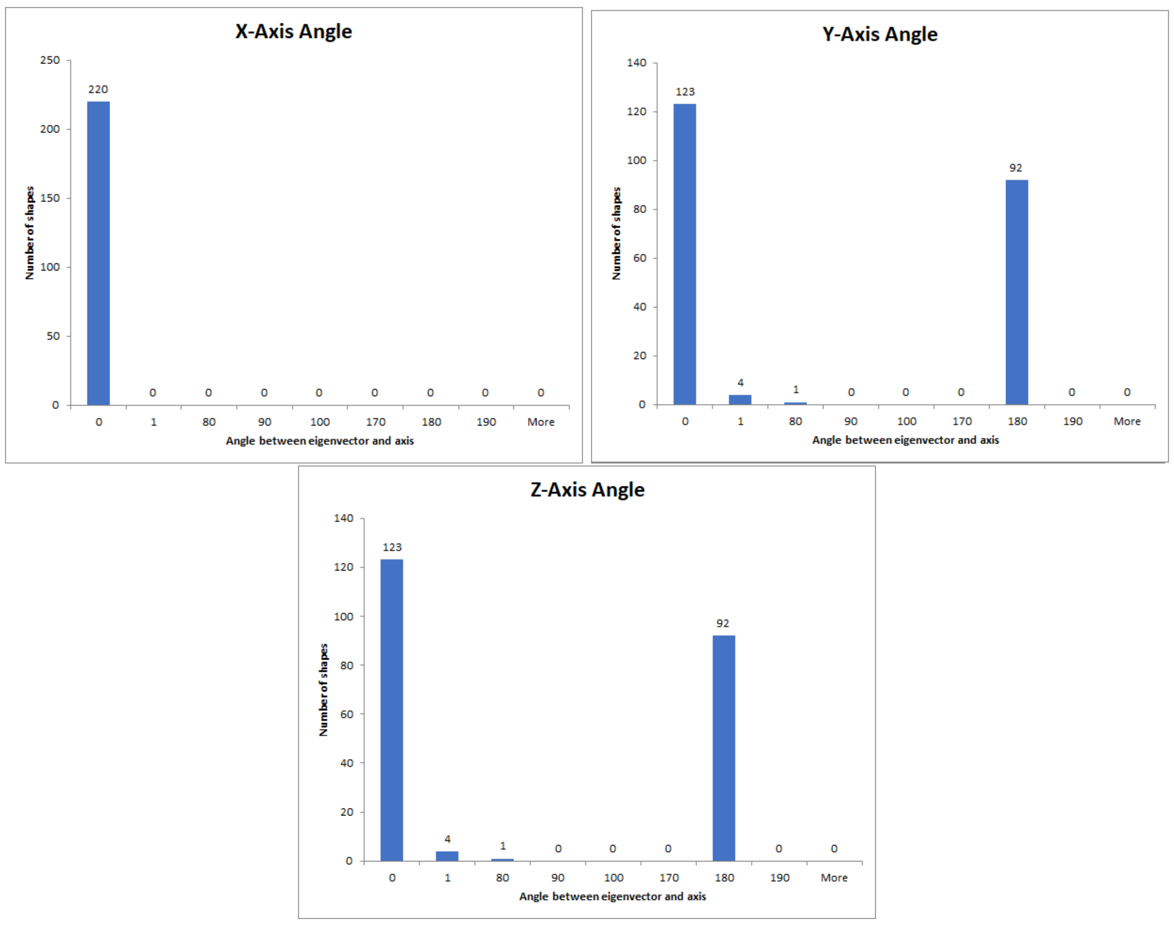


Figure 9: Histograms of the angles between the eigenvectors and the x-, y- and z-axis. It is clear that most of these have been aligned with their respective axis. The few outliers have been examined manually, and these shapes are mostly symetrical and round along the y-z axes, so the PCA alignment will change somewhat randomly, since these eigenvectors are very similar. However these shapes are in that case also aligned.

4.2 Step 3A: 3D shape descriptors

In this step the first features of the 3D shapes are extracted. For this shape-retrieval system, we use 3D shape descriptors for the elementary (single-value) feature descriptors, rather than using 2D appearance-based feature descriptors. This is simply due to the fact that these 3D descriptors are generally simpler to implement, without being less descriptive or effective for the querying process outlined in section 5.

The system starts by iterating over the off-files in the sample database. Each file is read and a grid object is created. Then, the different features are computed on the shape vertices. At the end of the process, the extracted features for each file are stored in a csv file.

The extracted features are the following:

4.2.1 Surface Area

To calculate the surface area of a 3D shape, the individual areas of the triangle faces are calculated and summed up. The formula for the are of a face i with three vertices is:

$$\text{Area } a_i = \sqrt{s \cdot (s - d_1) \cdot (s - d_2) \cdot (s - d_3)}$$

with $s = \dfrac{d_1 + d_2 + d_3}{2}$ and d_k with $k \in \{1, 2, 3\}$being the three lengths of the triangle face.

The total surface area A of a 3D shape with n faces is therefore

$$A = \sum_{i}^{n} a_i$$

4.2.2 Volume

To calculate the volume of a 3D shape, we apply a simple formula:

$$\text{Volume } V = \frac{1}{6} \left| \sum_{i}^{n} (p_{i,1} \times p_{i,2}) \cdot p_{i,3} \right| \text{ for each triangle face } i$$

The formula gives the volume if the barycenter coincides with the origin of the coordinate system, which has been done in the translation step of the normalization process.

Furthermore, for this formula to be correct, we need the 3D shapes to be watertight (no holes) and to have consistently oriented faces. Volume is not used as a comparative feature in our implementation, but is still used to calculate the compactness of a shape.

4.2.3 Compactness / Sphericity

The compactness and sphericity calculate how close a 3D shape is to a perfect sphere. For our feature, we are using the compactness value. The smaller this value is, the more compact the vertices of a shape are.

The compactness c is calculated by the formula

$$c = \frac{A^3}{36\pi V^2} \text{ where } A = \text{ surface area and } V = \text{ volume.}$$

$$\text{The sphericity s is equal to } s = \frac{1}{c}.$$

4.2.4 Axis-aligned Bounding Box Volume

After the normalization step, the bounding box is automatically aligned with the axes. The formula to calculate the volume v of the bounding box b is:

$$v_b = l_x \cdot l_y \cdot l_z$$

where l_k are the lengths of the bounding box for $k = \{x, y, z\}$, with

$$l_x = (max_x - min_x), l_y = (max_y - min_y), l_z = (max_z - min_z)$$

4.2.5 Diameter

To calculate the diameter, the longest distance between two vertices in the shape needs to be found. To compute this, the distance between every vertex and the barycenter (origin) is computed:

$$max\left(\sqrt{p_{i,x}^2 + p_{i,y}^2 + p_{i,z}^2}\right) \text{ for every vertex } p_i \text{ in the shape.}$$

Then the length between this first furthest vertex p_f and every other vertex in the shape is compared, with the longest being the diameter:

$$diam = max\left(\sqrt{(p_{i,x} - p_{f,x})^2 + (p_{i,y} - p_{f,y})^2 + (p_{i,z} - p_{f,z})^2}\right)$$

4.2.6 Eccentricity

The eccentricity denotes how much the vertices of a shape are spread out along the major and minor eigenvectors. The larger this value is, the more a shape is elongated along one axis. For a unit-sized cube, the eccentricity would be exactly 1.

The eccentricity of a 3D shape is calculated by:

$$\frac{|\lambda_1|}{|\lambda_3|}$$

Where λ_1 is the major eigenvalue and λ_3 is the minor eigenvalue. From this we can easily deduce why a cube has a eccentricity of 1. This is because the cube has equal eigenvectors in all directions, as the vertices in a cube are spread uniformly in all directions.

4.2.7 Example

File	Class	Surface Area	Volume	Compactness	Bounding Box Volume	Diameter	Eccentricity
Cube.off	-	6	1	1.912	1	1.73	1
61.off	Airplane	0.492	0.009	11.627	0.107	0.996	28.157
104.off	Chair	0.816	0.01	43.478	0.244	1.109	4.434

Table 1: Table with the computed features for 3 different 3D shapes. The first shape is a 1x1x1 cube. Features like surface area and volume have been double checked using MeshLab. The remaining features all make sense for the given shapes and compared to the cube.

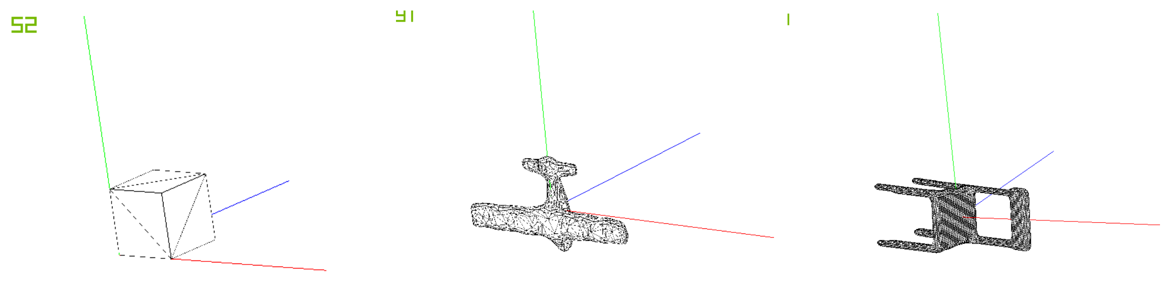


Figure 10: The three files for which the first features have been calculated and compared.

4.3 Shape Property Descriptors

In addition to the previous elementary descriptors, 5 shape property descriptors are also computed for each 3D shape. These descriptors are distributions, not single values. Since for most of these descriptors, the possible number of combinations is very high ($O(N)$ to $O(N^4)$ possible combinations of N vertices). The actual space of computations n for each of the descriptors except D1 in section 4.3.2 is set to 1000000.

The data is gathered and represented using histograms with 14 bins each. The bin size is defined by the maximum possible value divided by 14.

4.3.1 A3: Angle between three random vertices

Calculating the angle between 3 random vertices in a given 3D shape with N vertices gives a N^3 space of possible combinations. This space is supsampled to be $n = 1000000$. Using three different for loops over $k = \sqrt[3]{n}$ iterations each, the angles Θ between three randomly sampled vertices p_1, p_2, p_3 are calculated as follows:
First, the two vectors $a = vec(p_1, p_2)$ and $b = vec(p_2, p_3)$ are created:

$$a = \{p_{1,x} - p_{2,x}; p_{1,y} - p_{2,y}; p_{1,z} - p_{2,z}\}$$
$$b = \{p_{2,x} - p_{3,x}; p_{2,y} - p_{3,y}; p_{2,z} - p_{3,z}\}$$

These vectors are normalized before the dot product between them is calculated using their magnitudes:

$$a_{mag} = \sqrt{(a_x)^2 + (a_y)^2 + (a_z)^2}$$
$$b_{mag} = \sqrt{(b_x)^2 + (b_y)^2 + (b_z)^2}$$
$$a_n = \left\{\frac{a_x}{a_{mag}}; \frac{a_y}{a_{mag}}; \frac{a_z}{a_{mag}}\right\}$$
$$b_n = \left\{\frac{b_x}{b_{mag}}; \frac{b_y}{b_{mag}}; \frac{b_z}{b_{mag}}\right\}$$

Finally, the dot product *dot* between the normalized vector is calculated and the angle is recovered:

$$dot = a \cdot b = (a_{n.x} \cdot b_{n.x} + a_{n.y} \cdot b_{n.y} + a_{n.z} \cdot b_{n.z})$$
$$\Theta = arccos(dot) \cdot \frac{180}{\pi}$$

4.3.2 D1: Distance between a random vertex and the barycenter

For the distance between a random vertex and the barycenter, all of the possible N vertices are analyzed, and there is no need for subsamplingl, because the 3D shapes have already been normalized and each have at most 10000 vertices. Because of the translation step during the normalization process, the barycenter of each shape coincides with the coordinate origin (0,0,0). Because of this, the distance to the barycenter d of a vertex p_i is defined as the euclidian distance, namely:

$$d_i = \sqrt{(p_{i,x})^2 + (p_{i,y})^2 + (p_{i,z})^2} \text{ for every vertex } p_i$$

.

4.3.3 D2: Distance between two random vertices

Calculating the distance between two random vertices in a given 3D shape with N vertices gives a N^2 space of possible combinations. This space is supsampled to be $n = 1000000$. Using two different for loops over $k = \sqrt{n}$ iterations each, the distance d between two randomly sampled vertices p_1, p_2 are calculated as follows:

$$d = \sqrt{(p_{1,x} - p_{2,x})^2 + (p_{1,y} - p_{2,y})^2 + (p_{1,z} - p_{2,z})^2}$$

4.3.4 D3: Square root of area of triangle given by three random vertices

Calculating the square root of the area of the triangle between three random vertices in a given 3D shape with N vertices gives a N^3 space of possible combinations. This space is supsampled to be $n = 1000000$. Using three different for loops over $k = \sqrt[3]{n}$ iterations each, the area a between three randomly sampled vertices p_1, p_2, p_3 is calculated as follows:

$$d_1 = \sqrt{(p_{1,x} - p_{2,x})^2 + (p_{1,y} - p_{2,y})^2 + (p_{1,z} - p_{2,z})^2}$$
$$d_2 = \sqrt{(p_{2,x} - p_{3,x})^2 + (p_{2,y} - p_{3,y})^2 + (p_{2,z} - p_{3,z})^2}$$
$$d_3 = \sqrt{(p_{1,x} - p_{3,x})^2 + (p_{1,y} - p_{3,y})^2 + (p_{1,z} - p_{3,z})^2}$$
$$s = \frac{d1 + d2 + d3}{2}$$

and

$$a = \sqrt{s \cdot ((s - d_1) \cdot (s - d_2) \cdot (s - d_3))}$$

4.3.5 D4: cube root of volume of tetrahedron formed by four random vertices

Calculating the cube root of the volume of the tetrahedron between four random vertices in a given 3D shape with N vertices gives a N^4 space of possible combinations. This space is supsampled to be $n = 1000000$. Using three different for loops over $k = \sqrt[4]{n}$ iterations each, the volume v between four randomly sampled vertices p_1, p_2, p_3, p_4 is calculated as follows:

$$v = \frac{(p_1 - p_4) \cdot ((p_2 - p_4) \times (p_3 - p_4))}{6}$$

4.3.6 Histograms

The features were extracted with 1000000 data points per descriptor, divided up into 14 bins and normalized. Histograms of similar shapes and shapes of the same class should look very similar to each other, while histograms of different classes should be noticeably distinct. In Fig.11, the histograms of the five extracted shape property descriptors of the classes "Ant" and "Cup" are shown as an example. The similar shapes all have similar histograms for the most part, indicated by the lines that are tightly overlapping and depicting a distinctive shape. The shapes of the histograms are different for both classes, which is to be expected since "Ant" and "Cup" look different and are expected to have different 3D features.

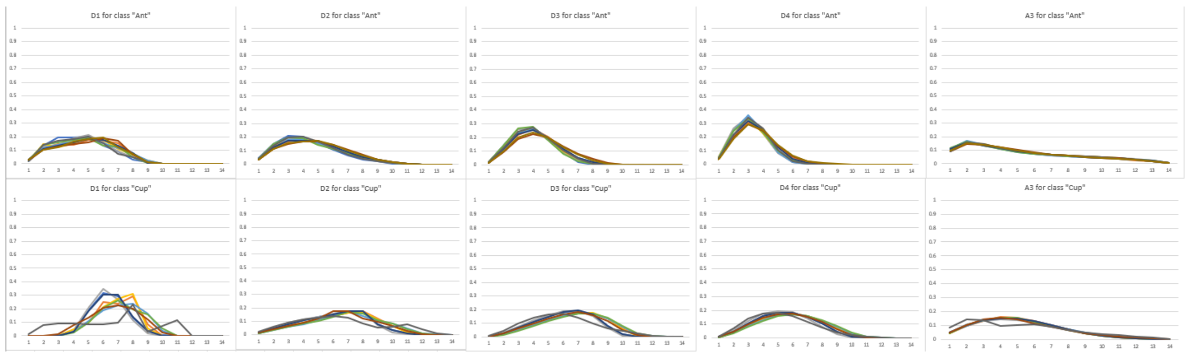


Figure 11: Feature histograms for the classes "Ant" and "Cup". Each line represents one of the 10 shapes per class.

4.4 Standardization

To be able to compare the 3D shape descriptors, they first need to be normalized. For this, standardization is performed on the extracted global features of the database and the query shape. After extracting the feature values f_i with $i = \{1, ..., 5\}$, this is done by using the average μi and standard deviation σi. Depending on the size and number of outliers, it might be beneficial to standardize the database by a certain size. In our case, we divide the database by $\sigma_i \cdot 2$.

$$f_i^{new} = \frac{f_i - \mu_i}{\sigma_i \cdot 2}$$

The feature histograms are also normalized. Each value which accounts for the number of features in one of the 14 bins is divided by the total number of elements in the histogram. This returns a histograms which values are between 0 and 1 (relative frequencies) with a combined total of 1. For a histogram h with n elements and 14 bins, each value is defined as:

$$h_i^{new} = \frac{h_i}{n}, \text{ for } i \in \{0, ..., 13\}$$

5 Step 4: Querying

In this step, the program is extended as to be able to perform a query on the database. A 3D off-file is given as a query input from which the features are extracted. The feature csv file is loaded in and the program compares these features with the query features. Finally, the K closest matches are returned using the windows console, depending on the user input for K, with a minimum of 5. OpenGL is used to create 6 windows and to display the query shape and the 5 closest shapes returned by the query using our custom shape viewer from section 2.

5.1 Distance Metrics

The actual differences between two 3D shapes can be computed using different metrics. Each of these work differently and can work better or worse for different types of data. There are different distance metrics for distinct shape descriptors and histograms. The differences are calculated pairwise between the feature vectors of the database and the query shape.

5.1.1 Euclidean Distance

The simplest distance metrics is the Euclidean Distance. It can be used to quickly compare distinct feature values (for example surface area). The difference (distance) d between two feature vectors v_a, v_b of size n is calculated as follows:

$$d_{a,b} = \sqrt{\sum_{i=1}^{n} (v_{a,i} - v_{b,i})^2}$$

5.1.2 Euclidean Distance for Histograms

The simple Euclidean Distance can also be used to compare histograms. Here, the difference is computed by comparing the pairwise bins B of the histograms h_a, h_b:

$$d_{a,b} = \sqrt{\sum_{i=1}^{B} (h_{a,i} - h_{b,i})^2}$$

5.1.3 Cross-Bin Distance

A more accurate way to compare histograms is using a Cross-Bin Euclidean Distance. The bins are not only pairwise compared, but also to adjacent bins. This is done by using predefined weights $w_{i,j}$ which denote how much a certain bin comparison contributes to the overall distance. The weights used in the later example results are for example 0.8 for $i = j$ and 0.1 for $abs(i - j) = 1$:

$$d_{a,b} = \sqrt{\sum_{i=1}^{B} \sum_{j=1}^{B} w_{i,j} \cdot (h_{a,i} - h_{b,j})^2}$$

The cross-bin distance is generally higher for shapes that might be similar or equal to each other than a pure Euclidean distance. That is because the histogram features are sampled randomly, and some amount of noise is reduced and captured by comparing different histogram bins. This distance metric is useful to capture certain histogram "shapes" instead of only focusing on individual bin values. Shapes with similar histogram shapes will have a smaller distance to each other than histograms with different histogram shapes, even of those different histograms might have very similar values for individual bins.

5.2 First Results

The first results are produced by first extracting the features from the normalized database, tracking the average values and standard deviations of each feature and writing them in a database csv file. The features are normalized using standardization, and the histograms are constructed using 14 bins per histogram and normalizing them based on the histogram element count.

To perform a query, the user can specify if they want to reload a different (normalized) database or enter a (normalized) query shape to start the query.

The database csv file is loaded in and converted into feature vector objects. The features of the query shapes are then computed, normalized using min/max normalization, and the histograms are created using the same process as for the database. Then, the query shape is compared to the shapes in the database by taking each pair of feature vectors and performing different distance computations on them.

The first five features (outlined in 4.2) are compared using Euclidean Distance. Then, each of the 5 different histograms are pairwise compared to each other using the Cross-Bin Euclidean Distance with the weights $w_{i,j} = 0.8$ for $i = j$ and $w_{i,j} = 0.1$ for $abs(i - j) = 1$.

At this point, no custom weights are used yet. The 5 global descriptors and 5 histograms get equal weights for the overall distance: $w_f = 0.5$ for the combined global features and $w_h = 0.1$ for the cross-bin distances for each of the five histograms, so that the total weight will be equal to 1.

As first examples, three files are used as query shapes. The results are shown in the tables 5 to 7 and figures 21 to 23.

Name	Shape	Distance
61.off	Airplane	0.202892
9.off	Human	0.339447
8.off	Human	0.388362
63.off	Airplane	0.435189
68.off	Airplane	0.435676
69.off	Airplane	0.44643
64.off	Airplane	0.467666
5.off	Human	0.47134
62.off	Airplane	0.471975
65.off	Airplane	0.504467

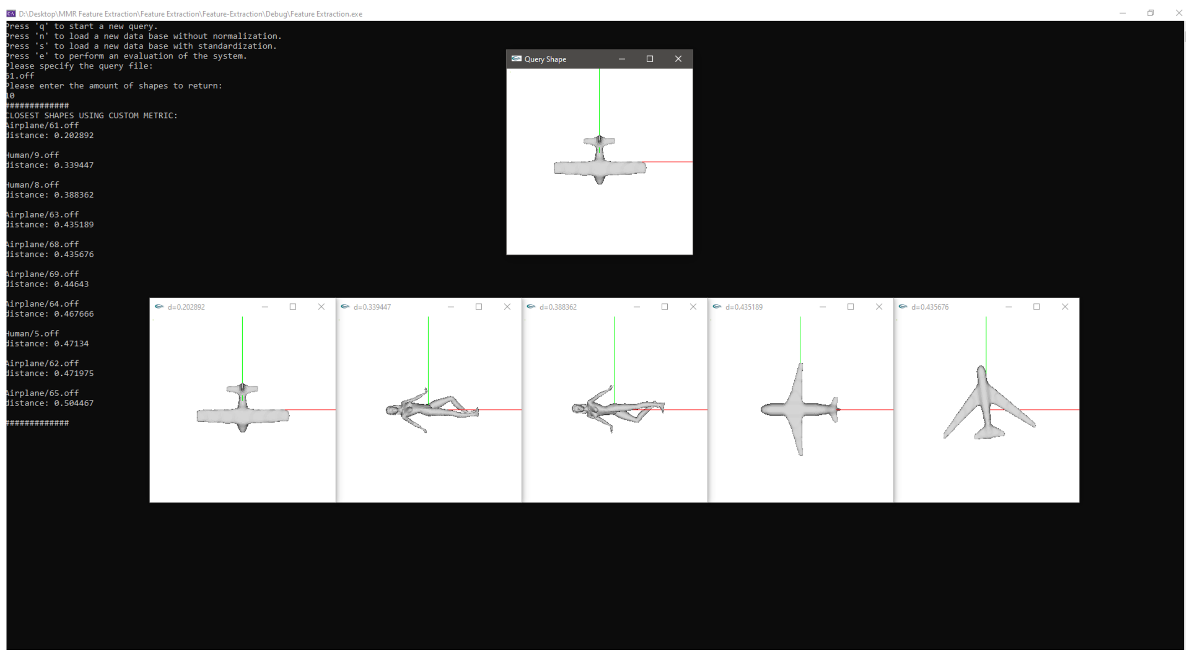
Table 2: Results of the first query of the file "61.off"

Name	Shape	Distance
104.off	Chair	0.179716
105.off	Chair	0.293163
110.off	Chair	0.436589
101.off	Chair	0.569196
107.off	Chair	0.629043
108.off	Chair	0.650517
44.off	Glasses	0.762517
141.off	Table	0.783513
103.off	Chair	0.817342
48.off	Glasses	0.861277

Table 3: Results of the query of the file "104.off"

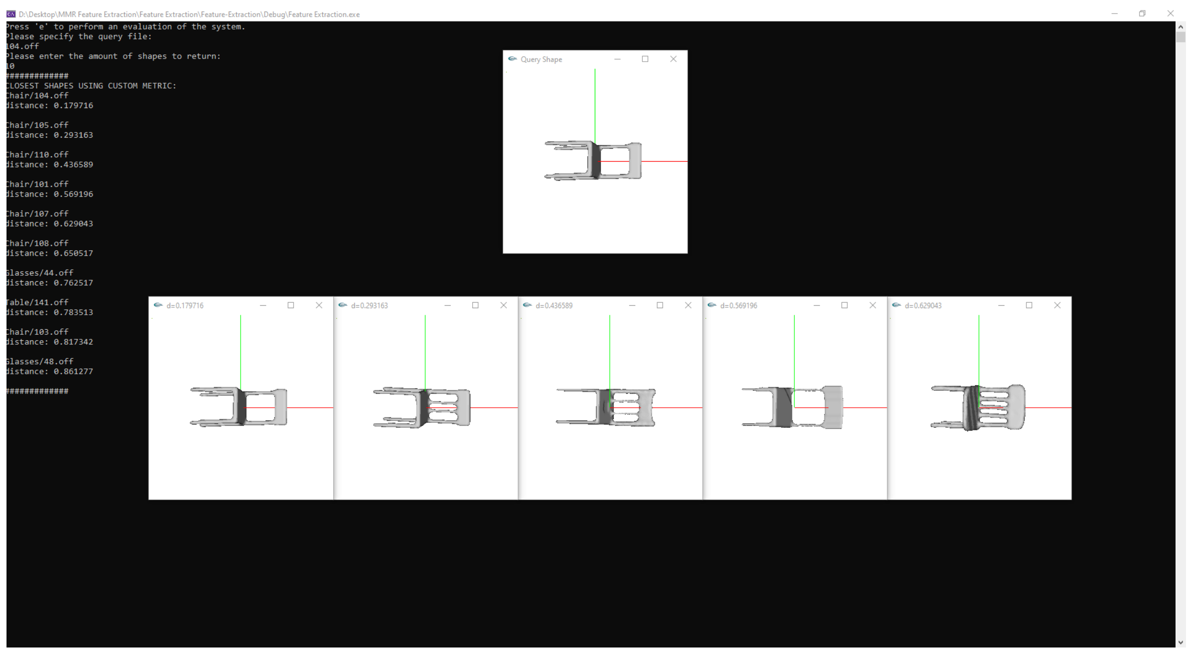
Name	Shape	Distance
282.off	Armadillo	0.214867
285.off	Armadillo	0.492644
389.off	FourLeg	0.513487
166.off	Teddy	0.524122
362.off	Vase	0.532779
170.off	Teddy	0.557944
167.off	Teddy	0.558985
182.off	Hand	0.565201
161.off	Teddy	0.578729
169.off	Teddy	0.596093

Table 4: Results of the query of the file "282.off"


D:\Desktop\MMR Feature Extraction\Feature Extraction\Feature-Extraction\Debug\Feature Extraction.exe
Query Shape
d=0.202892
d=0.339447
d=0.388362
d=0.435189
d=0.435676

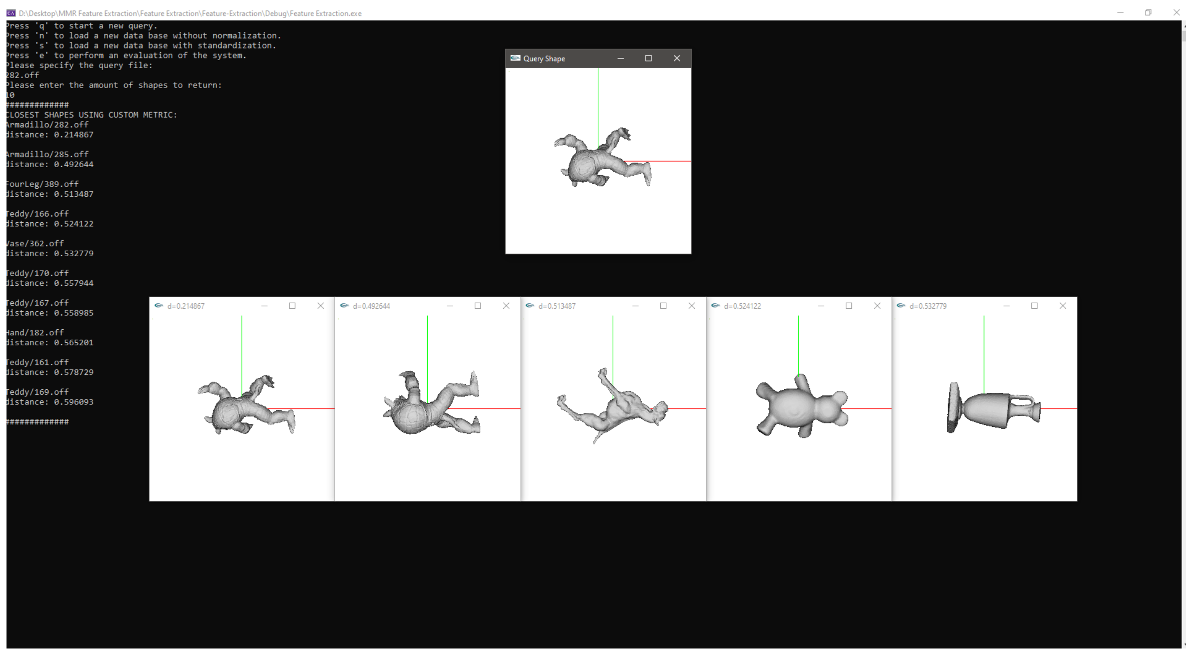
```
Press 'q' to start a new query.
Press 'n' to load a new data base without normalization.
Press 's' to load a new data base with standardization.
Press 'e' to perform an evaluation of the system.
Please specify the query file:
61.off
Please enter the amount of shapes to return:
10
############
CLOSEST SHAPES USING CUSTOM METRIC:
Airplane/61.off
distance: 0.202892

Human/9.off
distance: 0.339447

Human/8.off
distance: 0.388362

Airplane/63.off
distance: 0.435189

Airplane/68.off
distance: 0.435676

Airplane/69.off
distance: 0.44643

Airplane/64.off
distance: 0.467666

Human/5.off
distance: 0.47134

Airplane/62.off
distance: 0.471975

Airplane/65.off
distance: 0.504467

############
```

Figure 12: Output of the first query example for the query file ”61.off”.


D:\Desktop\MMR Feature Extraction\Feature Extraction\Feature-Extraction\Debug\Feature Extraction.exe
Query Shape
d=0.179716
d=0.293163
d=0.436589
d=0.569196
d=0.629043

```
Press 'e' to perform an evaluation of the system.
Please specify the query file:
104.off
Please enter the amount of shapes to return:
10
############
CLOSEST SHAPES USING CUSTOM METRIC:
Chair/104.off
distance: 0.179716

Chair/105.off
distance: 0.293163

Chair/110.off
distance: 0.436589

Chair/101.off
distance: 0.569196

Chair/107.off
distance: 0.629043

Chair/108.off
distance: 0.650517

Glasses/44.off
distance: 0.762517

Table/141.off
distance: 0.783513

Chair/103.off
distance: 0.817342

Glasses/48.off
distance: 0.861277

############
```

Figure 13: Output of the first query example for the query file ”104.off”.


D:\Desktop\MMR Feature Extraction\Feature Extraction\Feature-Extraction\Debug\Feature Extraction.exe
Query Shape
d=0.214867
d=0.492644
d=0.513487
d=0.524122
d=0.532779

```
Press 'q' to start a new query.
Press 'n' to load a new data base without normalization.
Press 's' to load a new data base with standardization.
Press 'e' to perform an evaluation of the system.
Please specify the query file:
282.off
Please enter the amount of shapes to return:
10
############
CLOSEST SHAPES USING CUSTOM METRIC:
Armadillo/282.off
distance: 0.214867

Armadillo/285.off
distance: 0.492644

FourLeg/389.off
distance: 0.513487

Teddy/166.off
distance: 0.524122

Vase/362.off
distance: 0.532779

Teddy/170.off
distance: 0.557944

Teddy/167.off
distance: 0.558985

Hand/182.off
distance: 0.565201

Teddy/161.off
distance: 0.578729

Teddy/169.off
distance: 0.596093

############
```

Figure 14: Output of the first query example for the query file ”282.off”.

In the first example, there are a number of shapes from an incorrect shape class. In the first 5 returned and displayed query results, two "Human" shapes are returned. In the full query result of size 10, there are a total of 7 correctly returned "Airplane" shapes and 3 incorrect "Human" shapes.

For the second example, the system returns five correct "Chair" shapes as the first query result. In the full query result, 7 shapes are correct and 3 shapes are incorrect, of which one shape is of class "Table", which is very close to the shape "Chair".

In the third example, only 2 shapes are returned correctly in the first five results. In the full query, only 2 shapes are of the correct shape Armadillo", with 8 shapes being incorrect. Of these incorrect shapes, 5 shapes are "Teddy", which could be considered similar to "Armadillo".

Overall, the system manages to always return the query shape itself as the first and closest shape. Besides that, the accuracy of the query result varies for different classes. For every example. the system manages to return correct or similar classes. These results can most likely be improved by adding custom weights to the different distances. A full evaluation with custom weights is explained in detail in section 7.

Note that while the first result of the queries is the shape itself, the distance is not equal to 0. This would be preferred as it is a property of a metric. However, the reason why the distance is not 0 is due to the noise in the histograms and the cross-bin matching. Due to the cross-bin matching attempting to compensate for noise, a bit of extra distance is introduced. This should not be a problem as this is roughly the same for all shapes.

6 Step 5: Scalability

In order to improve the running time and to facilitate visualization for large databases, certain refinements can be done to the shape retrieval system. There are two distinct mechanics:

- Approximate-Nearest-Neighbor (ANN)
- Dimensionality Reduction

6.1 Approximate-Nearest-Neighbor

To efficiently query a set of data points in a d-dimensional space, the Approximate-Nearest-Neighbor (ANN) algorithm works similar to the k-NN algorithm. The data set is pre-processed into a search structure in order to be able to return the k (approximately) nearest data points to a query point q efficiently.

First, the data (in our case extracted features) is stored using a search structure called *kd-tree*. This search structure can then be queried for a single object (in this case a query shape), using an approximate nearest neighbor search algorithm to return a defined number (denoted k) of closest shapes in the search structure. Using this method, the time complexity can be reduced, from the previous linear querying time $O(N)$ to a logarithmic querying time $O(k \cdot \log N)$ for querying a database of size N.

The distance between two shapes can be defined by the Minkowski metrics, and in our case the Euclidean Distance is used. ANN trades accuracy for speed compared to a traditional k-NN algorithm and is able to work well with a high number of dimensions d. In our case the dimensions are the individual features ($d = 75$).

The algorithm is implemented using the ANN library for C++ [MA10]. Firstly, an ANN kd-tree is created from the database, using the shape features as point coordinates. The query shape features are extracted and used as query point coordinates and a query on the kd-tree is performed, returning the k closest points including their distances.

Since the distance is calculated using Euclidean Distance for every feature without the use of individual weights by default, the returned results are only slightly different from the custom distance function outlined in section 5.2. The distances themselves are smaller, since there is no cross-bin-distance function. However, the example query results of the ANN implementation are almost equal to the custom distance implementation for the standard weights despite not using the cross-bin distance. The results of these examples are listed in the appendix A. The evaluation of the ANN implementation in terms of accuracy and other metrics is outlined in section 7.3.

6.2 Dimensionality Reduction

Dimensionality Reduction denotes the process of reducing a multi-dimensional space into a lower-dimensional space, often done for visualizing purposes. In our case, the feature vectors of 75 dimensions can be reduced into the 2D space in order to be plotted and visualized.

There are a number of approaches and algorithms that are possible for this procedure. In this implementation, the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm is used. This technique is well suited for the visualization of high-dimensional data sets and is often used in image processing. The dimensionality reduction function is implemented in Python, as the t-SNE is already implemented as a part of the *sklearn* package. It is run with a PCA initialization, a perplexity of 21 and a learning rate of 180. These values have been found by iterating over a number of possible combinations and comparing the 2D plots. The results are plotted using a colored legend. Additionally, a hover function is implemented, allowing closer inspection by hovering over the individual data points in the plot and inspecting individual file names and their respective classes.

The goal of this 2D visualization is to show the grouping of the different classes using the extracted features. The closer two points are in the plot, the closer their features are to each other. In theory, well separated and tightly grouped clusters should represent each class, and a clear distinction between classes and their features should be easily visible on the 2D plot.

In figure 15, the result of the dimensionality reduction is shown. There are some visible clusters for some shapes, for example for the class "Mech". However, there are a number of outliers of other classes which make a clear distinction harder. This could indicate that the features themselves are not representative enough, or that the t-SNE implementation in our program is not optimized enough to be able to cluster the points more effectively.

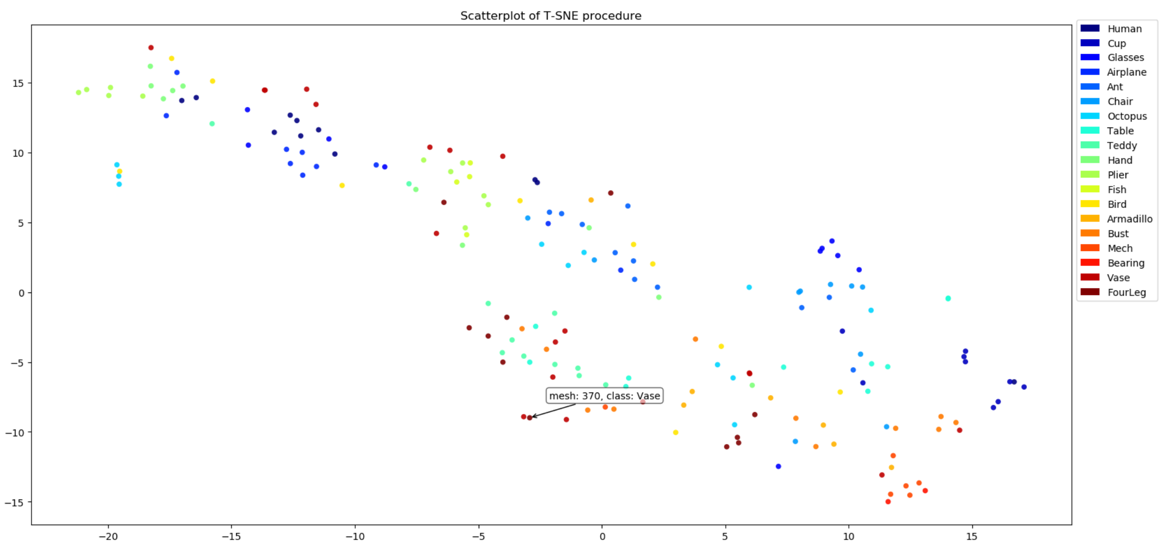


Figure 15: Screenshot of the dimensionality reduction output with hover functionality.

7 Step 6: Evaluation

In order to be able to assess the practicality and functionality of any shape-retrieval system, an evaluation needs to be performed. Different quality metrics can be considered for this step, as there are a multitude of different ways to quantify accuracy for such systems. First, the different metrics are outlined and explained. Then, the details and results of the evaluation of the shape-retrieval system are listed and explained.

7.1 Evaluation Metrics

To evaluate and assess the performance of a shape-retrieval system, a number of different metrics can be calculated. Generally speaking, a single metric will not encapsulate the full performance of a shape-retrieval system, as they only describe certain aspects of a query result. To be able to fully evaluate (and compare) different systems, a combination of evaluation metrics should be used.

All of these metrics use the following types of query results:

- True positives (TP): A shape is correctly returned in the query result
- True negatives (TN): A shape is correctly not returned in the query result
- False positives (FP): A shape is incorrectly returned in the query result
- False negatives (FP): A shape is incorrectly not returned in the query result

The metrics implemented for this evaluation are the following:

7.1.1 Accuracy

Accuracy is a very intuitive single-value metric which gives insight about the performance of the system as a whole. Most users have an idea about how to interpret accuracy, and will be able to assess the performance of a shape-retrieval system based on this metric. Accuracy evaluates how many of the database elements are identified correctly. This includes both correctly returned and correctly omitted elements (true positives and true negatives). For a database of size d, accuracy acc is defined as:

$$acc = \frac{(TP + TN)}{(TP + TN + FP + FN)} = \frac{(TP + TN)}{d}$$

The downside of the accuracy metric is that it might be deceptively high for large databases and small query sizes. If we imagine a database of 10 different classes with 10 shapes per class, and a query of size 10 with 0 true positives, the accuracy would be $\frac{0+80}{100} = 0.8$. Seeing that the query has returned not a single correct shape, this accuracy can seem very high, which highlights again the need to use a combination of different evaluation metrics.

7.1.2 Precision

Precision (also called Positive Prediction Value or PPV) is the number true positives in relation to the query size. A high precision means a large amount of the query result is the desired shape. A low precision means most of the returned elements are returned incorrectly. This metric gives a good evaluation about the correctness of the query results themselves, which is very relevant for most users. The precision PPV of a query result of size s is defined as:

$$PPV = \frac{TP}{(TP + FP)} = \frac{TP}{s}$$

7.1.3 Sensitivity

Sensitivity (also called True Positive Rate TPV) is the number of true positives in relation to the number of class elements in the database. A high sensitivity means many of the correct query shapes that are in the database are actually returned in the query result. A low sensitivity means that most items of the desired class are not returned. Compared to the the precision metric, the sensitivity gives more insight about the system performance from a database view point. The sensitivity TPV of a query on a database with c items in the desired class is defined as:

$$TPV = \frac{TP}{(TP + FN)} = \frac{TP}{c}$$

Together, precision and sensitivity allow for a good evaluation of the query performance of a shape-retrieval system.

7.1.4 Last Rank

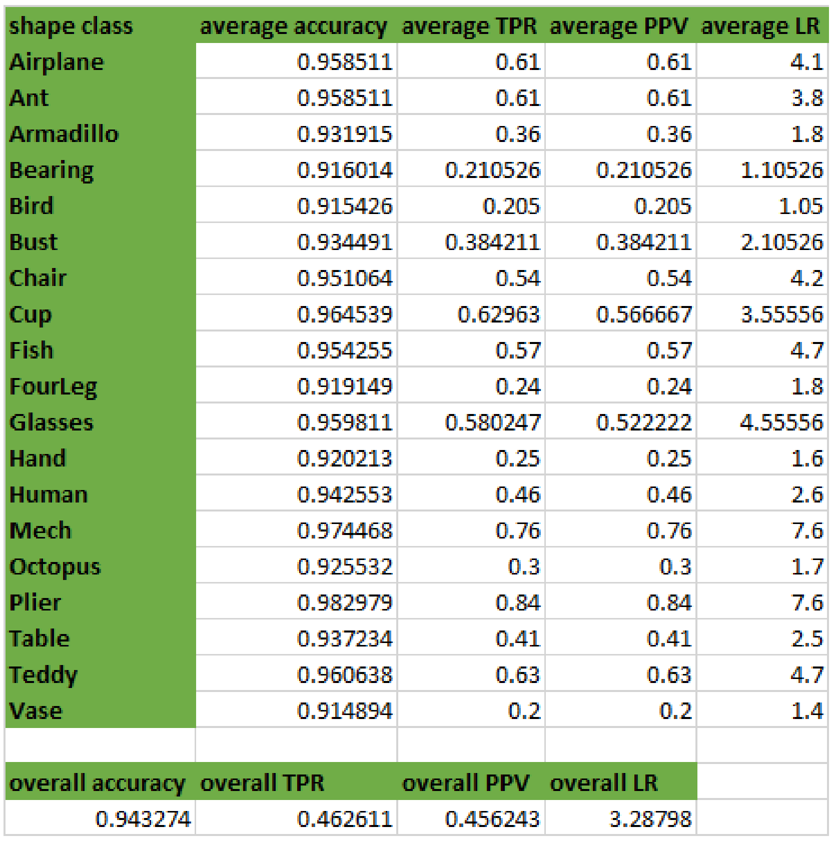
When performing a query, users might not interested in the actual number of true positives, or in the other relevant shapes in the database. A lot of times users will only look at the first few positive results of the query.

The Last Rank metric denotes how many first query results in sequence are true positives. The higher it is, the more true positives are returned before a false positive shows up in the query result. This metric is useful to evaluate how practical a system might be for the average user. Even when accuracy or precision is not perfect, last rank could still be relatively high and therefor indicate a usefulness of the system. The last rank LR is defined as the rank (position in the query) of the last true positive.

7.2 Evaluation Results

Using the aforementioned four evaluation metrics, the system is evaluated on the sample database. Each shape is individually used as a query item, performing a query and analyzing the results and tracking the metrics. For each class, the average accuracy, precision, sensitivity and last rank are computed and outputted into an evaluation csv file. An overall grand aggregate (averages over all the classes) is also computed.

In this step, the values for the custom weights in the querying function of the custom distance implementation are also determined. For both the global feature weight w_f and the histogram weight w_h, introduced in section 5.2, different values are tried out and compared. The best results were found using $w_f = 0.1$ and $w_h = 0.18$. This means that the histogram cross-bin distance has a higher impact on the distance. Figure 16 shows the result of the evaluation. Additional evaluation results for different weights are found in the appendix B.



shape class	average accuracy	average TPR	average PPV	average LR
Airplane	0.958511	0.61	0.61	4.1
Ant	0.958511	0.61	0.61	3.8
Armadillo	0.931915	0.36	0.36	1.8
Bearing	0.916014	0.210526	0.210526	1.10526
Bird	0.915426	0.205	0.205	1.05
Bust	0.934491	0.384211	0.384211	2.10526
Chair	0.951064	0.54	0.54	4.2
Cup	0.964539	0.62963	0.566667	3.55556
Fish	0.954255	0.57	0.57	4.7
FourLeg	0.919149	0.24	0.24	1.8
Glasses	0.959811	0.580247	0.522222	4.55556
Hand	0.920213	0.25	0.25	1.6
Human	0.942553	0.46	0.46	2.6
Mech	0.974468	0.76	0.76	7.6
Octopus	0.925532	0.3	0.3	1.7
Plier	0.982979	0.84	0.84	7.6
Table	0.937234	0.41	0.41	2.5
Teddy	0.960638	0.63	0.63	4.7
Vase	0.914894	0.2	0.2	1.4
overall accuracy	overall TPR	overall PPV	overall LR	
0.943274	0.462611	0.456243	3.28798	

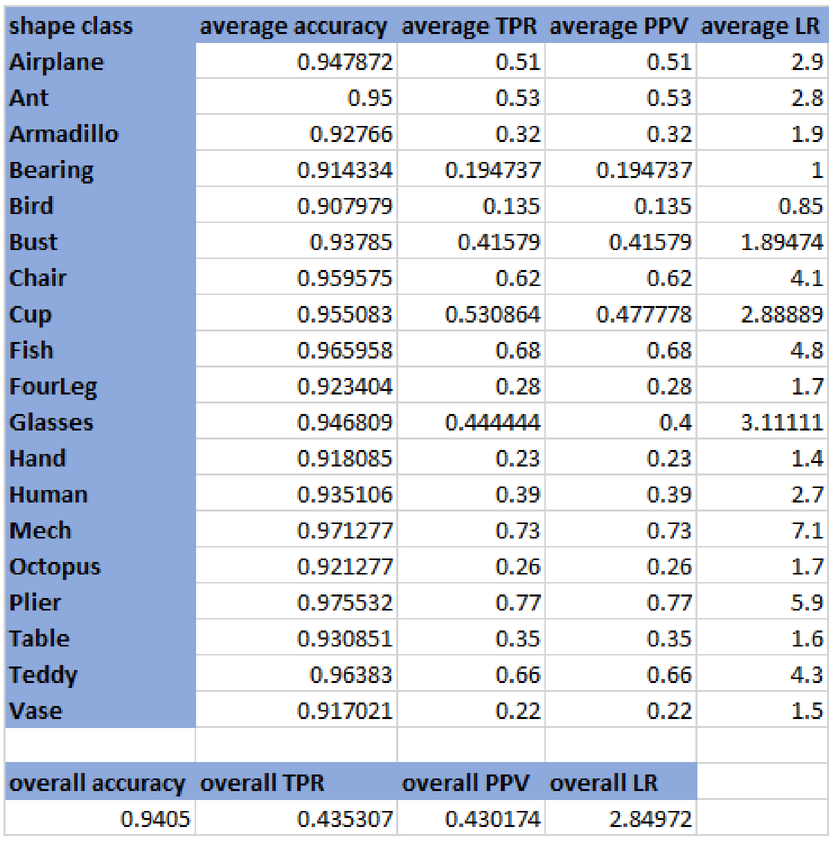
Figure 16: Results of the evaluation of the custom distance query implementation, with $w_f = 0.1$ and $w_h = 0.18$.

From the result of the evaluation, we can assess the usability and drawbacks of the system. Overall, almost half of the query results are correct in terms of query size and class size (TPR and PPV ≈ 0.45). The overall average Last Rank is 3.28, indicating that most of the time, more than 3 shapes are correctly returned for a query.

Additionally, the evaluation shows that the system performs better for certain shapes while performing worse for other shapes. For example, both the classes "Plier" and "Mech" have high accuracy and high Last Rank values. The system works particularly well for those kind of shapes. In contrast, shapes like "Bird" and "Vase" have very low values for all the metrics.

7.3 ANN evaluation

For the ANN implementation evaluation, the same approach is used. The evaluation metrics are the same in order to allow for correct comparison between the two implementations. The features are not individually weighted. Figure 17 shows the results of the evaluation.



shape class	average accuracy	average TPR	average PPV	average LR
Airplane	0.947872	0.51	0.51	2.9
Ant	0.95	0.53	0.53	2.8
Armadillo	0.92766	0.32	0.32	1.9
Bearing	0.914334	0.194737	0.194737	1
Bird	0.907979	0.135	0.135	0.85
Bust	0.93785	0.41579	0.41579	1.89474
Chair	0.959575	0.62	0.62	4.1
Cup	0.955083	0.530864	0.477778	2.88889
Fish	0.965958	0.68	0.68	4.8
FourLeg	0.923404	0.28	0.28	1.7
Glasses	0.946809	0.444444	0.4	3.11111
Hand	0.918085	0.23	0.23	1.4
Human	0.935106	0.39	0.39	2.7
Mech	0.971277	0.73	0.73	7.1
Octopus	0.921277	0.26	0.26	1.7
Plier	0.975532	0.77	0.77	5.9
Table	0.930851	0.35	0.35	1.6
Teddy	0.96383	0.66	0.66	4.3
Vase	0.917021	0.22	0.22	1.5
overall accuracy	overall TPR	overall PPV	overall LR	
0.9405	0.435307	0.430174	2.84972	

Figure 17: Results of the evaluation of the ANN implementation.

The evaluation shows that the ANN implementation is noticeably worse than the custom implementation in overall TPR, PPV, and LR. This is most likely due to the cross-bin distance function and the custom weights. Since ANN implements a pure Euclidean Distance in our case, it performs slightly worse. However, for larger databases the differences in accuracy might outweigh the need for a faster query.

8 Discussion

In this report, we outlined our implementation for a custom 3D-shape retrieval system. The outlined implementation and methods works reasonably well for some shapes, most notably "Plier" and "Mech". For other shapes, like "Airplane" or "Chair", the query results can be quite decent, returning very similar shapes (for example returning "Bird" instead of "Airplane", or "Table" instead of "Chair"), even if the evaluation metrics might not look too convincing.

The drawbacks of the low accuracies could be overcome by trying to create custom weights for different shape classes. It is very possible that different classes are more descriptive with global features, while others would benefit from taking the histograms more into account. The individual features and histograms themselves could also be weighted differently. A feature like surface area might be a lot more descriptive than a feature like diameter.

Another possible extension and future work could be implementing 2D shape descriptors and comparing them to the global 3D descriptors. Different shapes might benefit from their descriptive shapes and could be more distinct in that way. The best way would be to use a combination of both 3D and shape descriptors and weighing them differently to achieve the best accuracies.

Besides those possible improvements, the system could be evaluated and fine-tuned better by doing more complex evaluations with additional metrics, using larger databases, having more consistency between the shapes (more similar vertex and faces count) and by building a more extensive feature extraction process (sampling over a larger number of points for certain features). Most of these improvements will come with a higher preprocessing or feature extraction effort and will take more time, which might not be desirable.

References

[CNR20] ISTI - CNR. *MeshLab 2020.09.* `https://www.meshlab.net/`. (2020).

[Gro17] Khronos Group. *OpenGL API Version 4.6.* `https://www.opengl.org/about/`. (2017).

[Tel08] Alexandru C. Telea. *Data Visualization: Principles and Practice Chapter 3 Tutorial.* `https://www.cs.rug.nl/svcg/DataVisualizationBook/Sp3`. (2008).

[JG18] Benoît Jacob and Gaël Guennebaud. *Eigen.* `http://eigen.tuxfamily.org/index.php?title=Main_Page`. (2018).

[KHS10] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. "Learning 3D Mesh Segmentation and Labeling". In: *ACM Transactions on Graphics* 29.3 (2010).

[MA10] David M. Mount and Sunil Arya. *ANN: A Library for Approximate Nearest Neighbor Searching.* `https://www.cs.umd.edu/~mount/ANN/`. (Version 1.1.2, January 27, 2010).

[Bur] John Burkardt. *PLY object files.* `https://people.sc.fsu.edu/~jburkardt/data/ply/`.

A Query Examples for ANN

Name	Shape	Distance
61.off	Airplane	9.45195e-12
9.off	Human	0.0409417
8.off	Human	0.0482179
68.off	Airplane	0.0618367
63.off	Airplane	0.0646004
62.off	Airplane	0.0716188
5.off	Human	0.0796521
64.off	Human	0.081181
69.off	Airplane	0.0848906
6.off	Human	0.105189

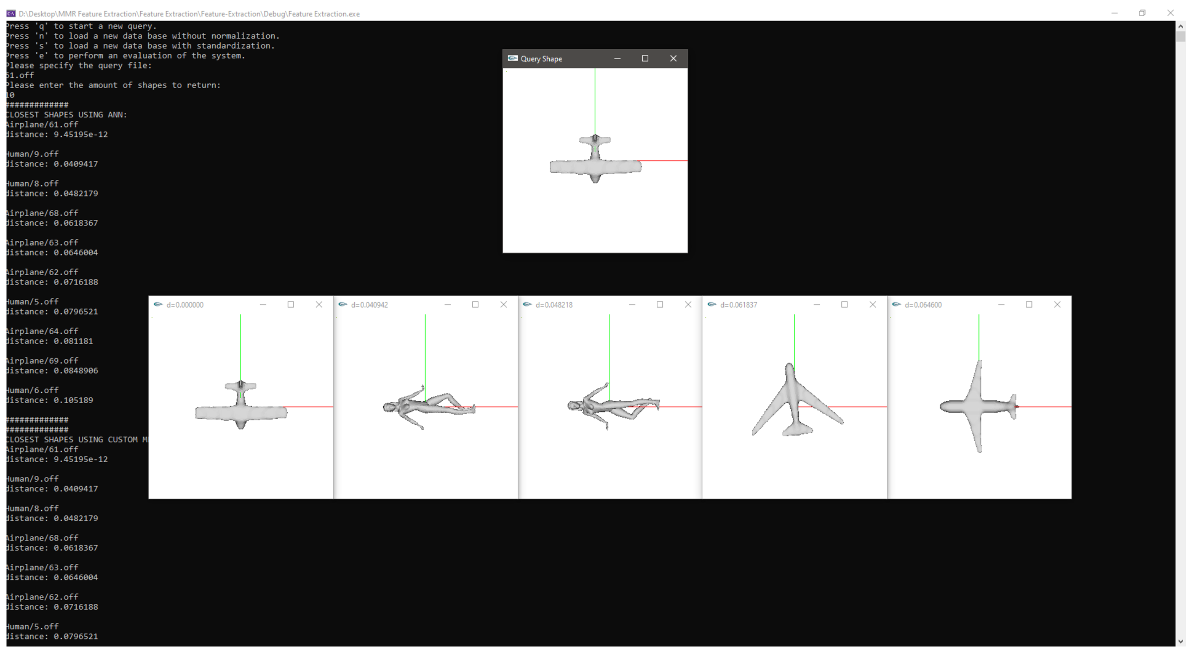
Table 5: Results of the first query of the file "61.off"

Name	Shape	Distance
104.off	Chair	0.000642388
105.off	Chair	0.0252774
110.off	Chair	0.10052
101.off	Chair	0.199977
107.off	Chair	0.311038
141.off	Table	0.317172
108.off	Chair	0.36633
44.off	Glasses	0.380021
48.off	Glasses	0.436767
106.off	Chair	0.478113

Table 6: Results of the query of the file "104.off"

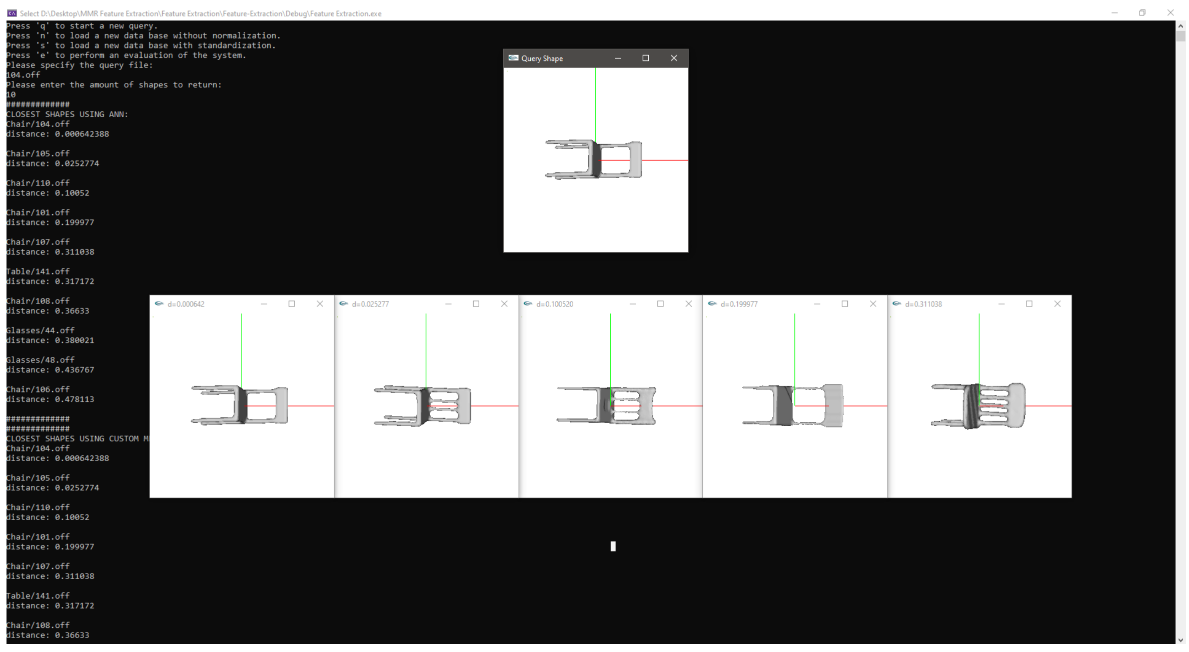
Name	Shape	Distance
282.off	Armadillo	0.000922214
285.off	Armadillo	0.102844
182.off	Hand	0.12582
170.off	Teddy	0.130113
166.off	Teddy	0.133634
389.off	FourLeg	0.134037
362.off	Vase	0.1356
167.off	Teddy	0.135936
169.off	Teddy	0.146344
168.off	Teddy	0.154255

Table 7: Results of the query of the file "282.off"



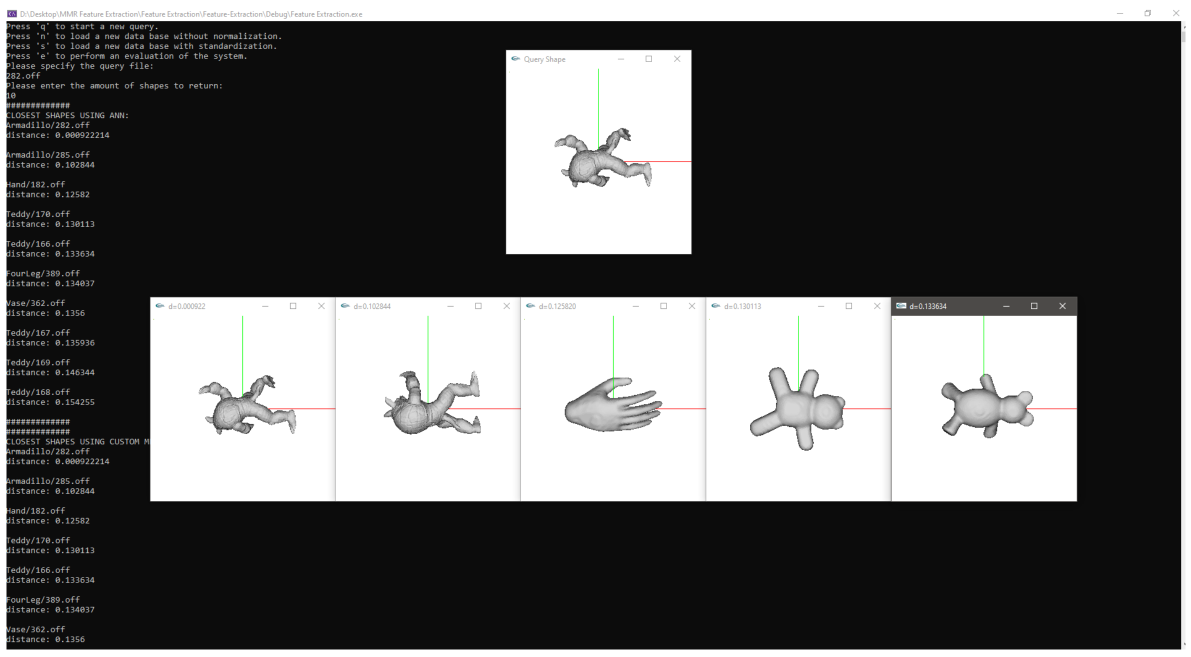
```
D:\Desktop\MMR Feature Extraction\Feature Extraction\Feature-Extraction\Debug\Feature Extraction.exe
Press 'q' to start a new query.
Press 'n' to load a new data base without normalization.
Press 's' to load a new data base with standardization.
Press 'e' to perform an evaluation of the system.
Please specify the query file:
61.off
Please enter the amount of shapes to return:
10
#############
CLOSEST SHAPES USING ANN:
Airplane/61.off
distance: 9.45195e-12

Human/9.off
distance: 0.0409417

Human/8.off
distance: 0.0482179

Airplane/68.off
distance: 0.0618367

Airplane/63.off
distance: 0.0646004

Airplane/62.off
distance: 0.0716188

Human/5.off
distance: 0.0796521

Airplane/64.off
distance: 0.081181

Airplane/69.off
distance: 0.0848906

Human/6.off
distance: 0.105189

#############
#############
CLOSEST SHAPES USING CUSTOM M
Airplane/61.off
distance: 9.45195e-12

Human/9.off
distance: 0.0409417

Human/8.off
distance: 0.0482179

Airplane/68.off
distance: 0.0618367

Airplane/63.off
distance: 0.0646004

Airplane/62.off
distance: 0.0716188

Human/5.off
distance: 0.0796521
```

Figure 18: Output of the first query example for the query file ”61.off”.



```
Select D:\Desktop\MMR Feature Extraction\Feature Extraction\Feature-Extraction\Debug\Feature Extraction.exe
Press 'q' to start a new query.
Press 'n' to load a new data base without normalization.
Press 's' to load a new data base with standardization.
Press 'e' to perform an evaluation of the system.
Please specify the query file:
104.off
Please enter the amount of shapes to return:
10
#############
CLOSEST SHAPES USING ANN:
Chair/104.off
distance: 0.000642388

Chair/105.off
distance: 0.0252774

Chair/110.off
distance: 0.10052

Chair/101.off
distance: 0.199977

Chair/107.off
distance: 0.311038

Table/141.off
distance: 0.317172

Chair/108.off
distance: 0.36633

Glasses/44.off
distance: 0.380021

Glasses/48.off
distance: 0.436767

Chair/106.off
distance: 0.478113

#############
#############
CLOSEST SHAPES USING CUSTOM M
Chair/104.off
distance: 0.000642388

Chair/105.off
distance: 0.0252774

Chair/110.off
distance: 0.10052

Chair/101.off
distance: 0.199977

Chair/107.off
distance: 0.311038

Table/141.off
distance: 0.317172

Chair/108.off
distance: 0.36633
```

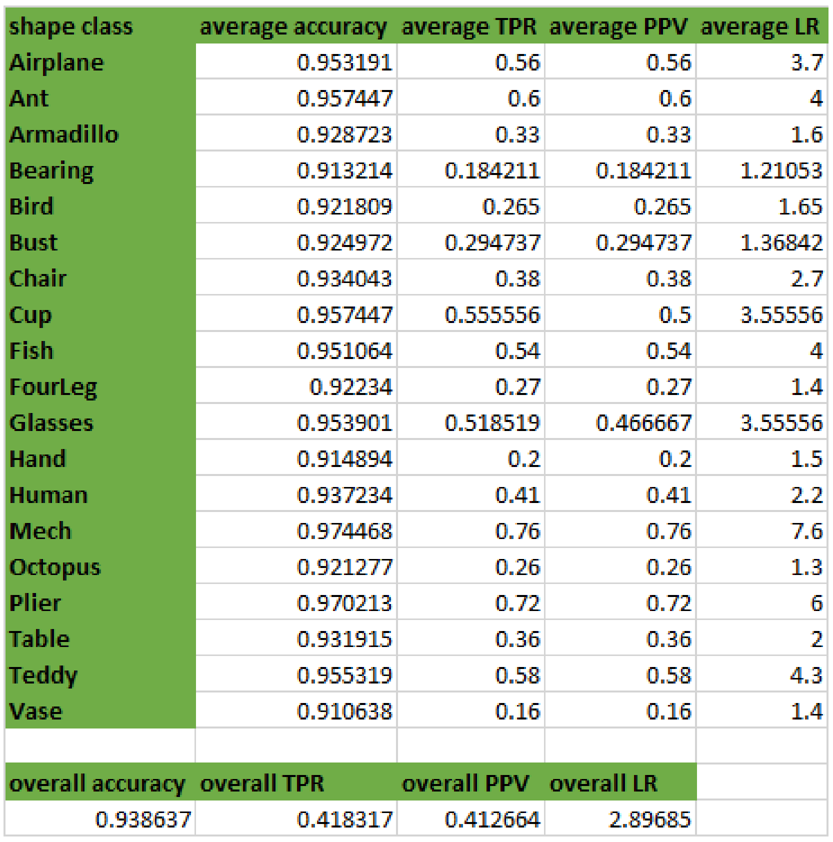
Figure 19: Output of the first query example for the query file ”104.off”.



```
D:\Desktop\MMR Feature Extraction\Feature Extraction\Feature-Extraction\Debug\Feature Extraction.exe
Press 'q' to start a new query.
Press 'n' to load a new data base without normalization.
Press 's' to load a new data base with standardization.
Press 'e' to perform an evaluation of the system.
Please specify the query file:
282.off
Please enter the amount of shapes to return:
10
#############
CLOSEST SHAPES USING ANN:
Armadillo/282.off
distance: 0.000922214

Armadillo/285.off
distance: 0.102844

Hand/182.off
distance: 0.12582

Teddy/170.off
distance: 0.130113

Teddy/166.off
distance: 0.133634

FourLeg/389.off
distance: 0.134037

Vase/362.off
distance: 0.1356

Teddy/167.off
distance: 0.135936

Teddy/169.off
distance: 0.146344

Teddy/168.off
distance: 0.154255

#############
#############
CLOSEST SHAPES USING CUSTOM M
Armadillo/282.off
distance: 0.000922214

Armadillo/285.off
distance: 0.102844

Hand/182.off
distance: 0.12582

Teddy/170.off
distance: 0.130113

Teddy/166.off
distance: 0.133634

FourLeg/389.off
distance: 0.134037

Vase/362.off
distance: 0.1356
```

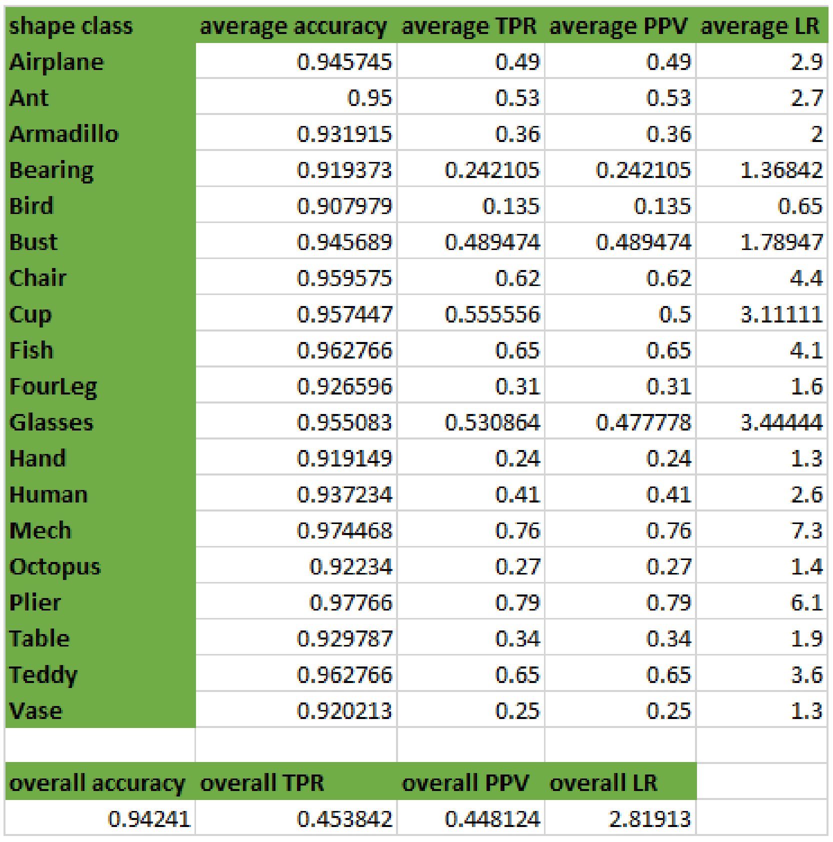
Figure 20: Output of the first query example for the query file ”282.off”.

B Additional Evaluation Results



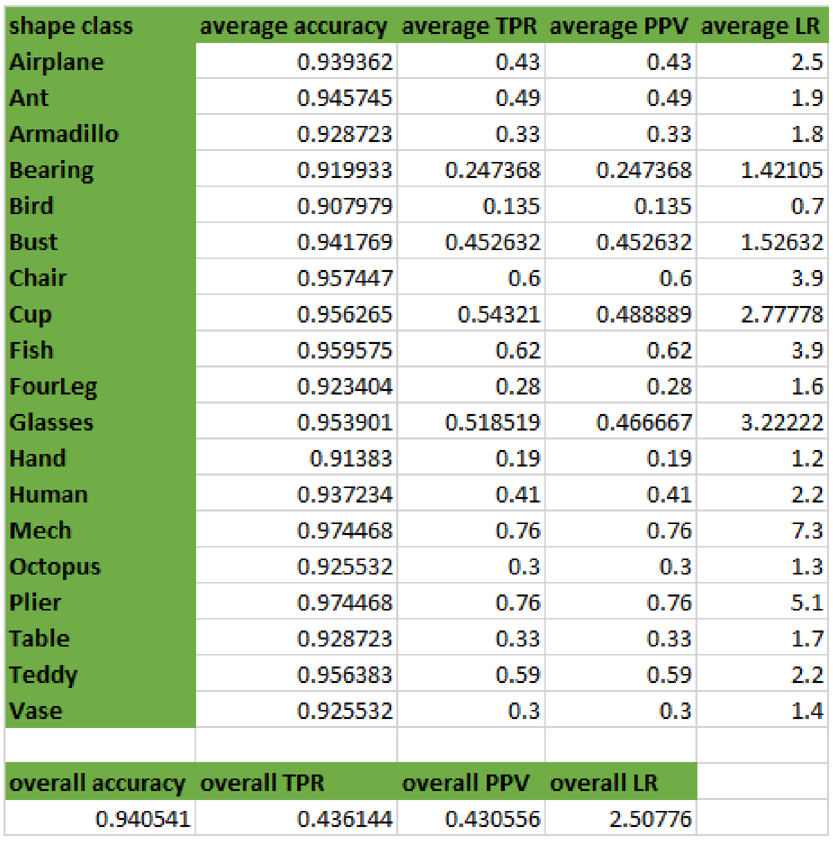
shape class	average accuracy	average TPR	average PPV	average LR
Airplane	0.953191	0.56	0.56	3.7
Ant	0.957447	0.6	0.6	4
Armadillo	0.928723	0.33	0.33	1.6
Bearing	0.913214	0.184211	0.184211	1.21053
Bird	0.921809	0.265	0.265	1.65
Bust	0.924972	0.294737	0.294737	1.36842
Chair	0.934043	0.38	0.38	2.7
Cup	0.957447	0.555556	0.5	3.55556
Fish	0.951064	0.54	0.54	4
FourLeg	0.92234	0.27	0.27	1.4
Glasses	0.953901	0.518519	0.466667	3.55556
Hand	0.914894	0.2	0.2	1.5
Human	0.937234	0.41	0.41	2.2
Mech	0.974468	0.76	0.76	7.6
Octopus	0.921277	0.26	0.26	1.3
Plier	0.970213	0.72	0.72	6
Table	0.931915	0.36	0.36	2
Teddy	0.955319	0.58	0.58	4.3
Vase	0.910638	0.16	0.16	1.4
overall accuracy	overall TPR	overall PPV	overall LR	
0.938637	0.418317	0.412664	2.89685	

Figure 21: Evaluation results with $w_f = 0.0$ and $w_h = 0.2$.



shape class	average accuracy	average TPR	average PPV	average LR
Airplane	0.945745	0.49	0.49	2.9
Ant	0.95	0.53	0.53	2.7
Armadillo	0.931915	0.36	0.36	2
Bearing	0.919373	0.242105	0.242105	1.36842
Bird	0.907979	0.135	0.135	0.65
Bust	0.945689	0.489474	0.489474	1.78947
Chair	0.959575	0.62	0.62	4.4
Cup	0.957447	0.555556	0.5	3.11111
Fish	0.962766	0.65	0.65	4.1
FourLeg	0.926596	0.31	0.31	1.6
Glasses	0.955083	0.530864	0.477778	3.44444
Hand	0.919149	0.24	0.24	1.3
Human	0.937234	0.41	0.41	2.6
Mech	0.974468	0.76	0.76	7.3
Octopus	0.92234	0.27	0.27	1.4
Plier	0.97766	0.79	0.79	6.1
Table	0.929787	0.34	0.34	1.9
Teddy	0.962766	0.65	0.65	3.6
Vase	0.920213	0.25	0.25	1.3
overall accuracy	overall TPR	overall PPV	overall LR	
0.94241	0.453842	0.448124	2.81913	

Figure 22: Evaluation results with $w_f = 0.5$ and $w_h = 0.1$.



shape class	average accuracy	average TPR	average PPV	average LR
Airplane	0.939362	0.43	0.43	2.5
Ant	0.945745	0.49	0.49	1.9
Armadillo	0.928723	0.33	0.33	1.8
Bearing	0.919933	0.247368	0.247368	1.42105
Bird	0.907979	0.135	0.135	0.7
Bust	0.941769	0.452632	0.452632	1.52632
Chair	0.957447	0.6	0.6	3.9
Cup	0.956265	0.54321	0.488889	2.77778
Fish	0.959575	0.62	0.62	3.9
FourLeg	0.923404	0.28	0.28	1.6
Glasses	0.953901	0.518519	0.466667	3.22222
Hand	0.91383	0.19	0.19	1.2
Human	0.937234	0.41	0.41	2.2
Mech	0.974468	0.76	0.76	7.3
Octopus	0.925532	0.3	0.3	1.3
Plier	0.974468	0.76	0.76	5.1
Table	0.928723	0.33	0.33	1.7
Teddy	0.956383	0.59	0.59	2.2
Vase	0.925532	0.3	0.3	1.4
overall accuracy	overall TPR	overall PPV	overall LR	
0.940541	0.436144	0.430556	2.50776	

Figure 23: Evaluation results with $w_f = 0.9$ and $w_h = 0.02$.