Multimedia Retrieval

Diego Renders (5740894) and Florian-Claudiu Gheorghica (6924123)

October 2019

1 Introduction

Problem statement.

2 Part 1

2.1 File reading

The files from the Princeton Shape Benchmark (PSB) were all provided in the OFF file format and an OFF reader was initially considered as a starting point. However, as a PLY format reader was likely to be necessary in the long-term, an OFF-to-PLY converter was written in order to enforce PLY as the default format for the application. This converter simply creates a PLY header using data from the OFF file, then copies the vertex and face data over. While not a scalable approach for more complex OFF files, the converter was able to successfully process the entire PSB database.

2.2 Database evaluation

The database contains around 1800 models in OFF file format. All of the models contain only triangles as faces. The distribution of faces is shown in figure ?. The average number of vertices is 4221, the minimum 10, and the maximum 160940. The histogram, along with the average, shows that most of the models have a vertex count of smaller than twenty thousand. In order to successfully perform the upcoming steps of feature extraction the models need to have approximately the same number of vertices. Two actions can be undertaken to achieve this, supersampling and subsampling. Supersample the models with a low vertex count, and subsample the models with a large vertex count. A target number of vertices has to be chosen for this. One solution could be to choose the average. However because several of the meshes have a very high resolution (a hundred thousand vertices), subsampling this to reach the four thousand required vertices would ruin the original shape. Therefore the target vertices is chosen to be higher than the average, so that the higher resolution models retain their characteristics. The only downside for choosing a higher target number of vertices is an increase in computation time. The target number of vertices is set at forty thousand, and the methodology for supersampling and subsampling is explained in the following two sections.

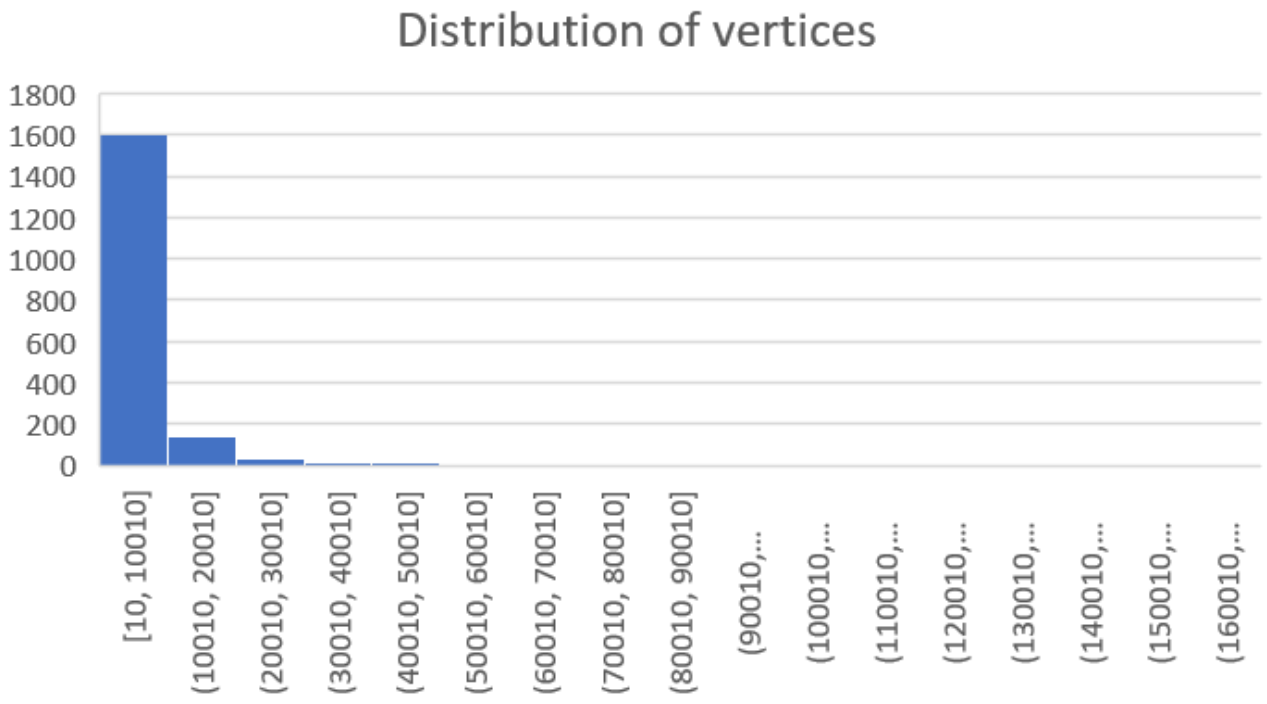


Figure 1: Histogram showing the distribution of vertices in the PSB database

2.3 Supersampling

Supersampling is done in this implementation by splitting a larger triangle into 4 smaller ones via adding the midpoints of the original triangle's edges as vertices and using them to form new triangles with the original

vertices. An example is shown below:

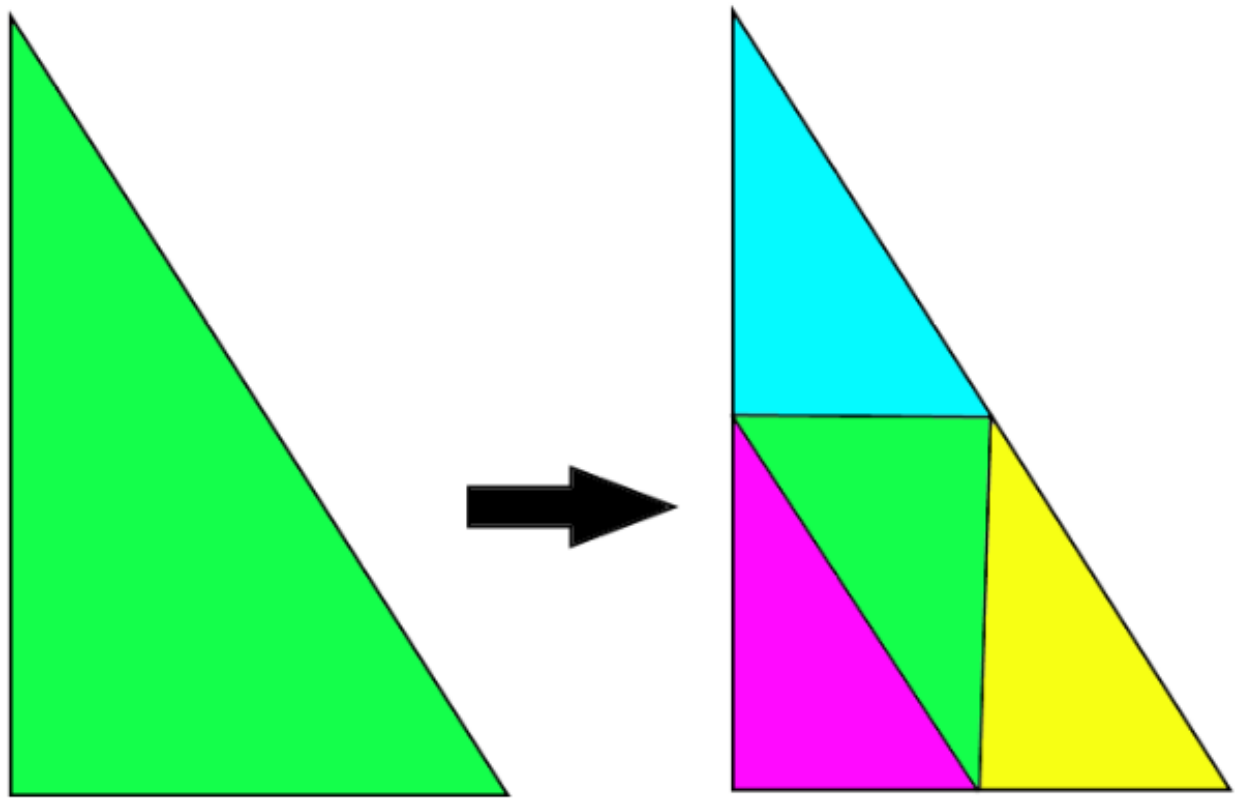


Figure 2: Supersampling Example

To ensure the triangles obtained like this are not outliers in terms of their size, the cells in the model were first sorted by their area inside a vector, from smallest to largest. The area of each cell was calculated using the following formula:

$$S = \frac{|AB||AC|sin(\theta)}{2} \tag{1}$$

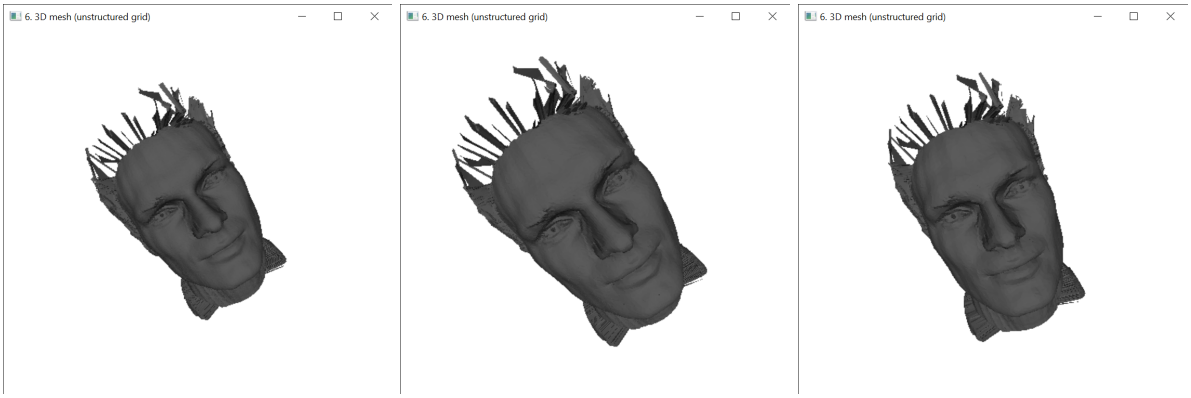
Thus, at each step, only the triangle with the largest surface is split up into smaller triangles. The original triangle is removed from the vector, and the smaller triangles are inserted at positions that do not disturb the ordered property of the data structure.

2.4 Subsampling

For the task of subsampling the Polygon Mesh Processing (pmp) library is used. The SurfaceSimplification class is used to perform mesh decimation, taking as input the mesh and a target number of vertices. To demonstrate that this algorithm works correctly for our models a pre and post visualisation is shown in figure 2. This is the largest model in the dataset with 316.498 faces and 160.940 vertices. The post decimation model contains 78.473 faces and 40.000 vertices in the middle picture and 20.000 vertices in the right picture. Even though the model only has a fourth of its original vertices in the second case, it clearly retains its characteristics. And even when the vertex count is brought down to 20.000 vertices it is hard to distinguish them. Therefore we can conclude that the algorithm is a suitable tool for this project.

2.5 Four step normalization

Next each mesh will go through the four step normalization pipeline so that it is ready to be used in upcoming tasks. Figure 1 shows a visualization of what each step does, the red, green, and blue represent the x, y, and z axises respectively from zero to one. The first step is to center on the Barycenter, than in step 2 PCA is done



(a) 160k vertices (b) 40k vertices (c) 20k vertices

Figure 3: Results of mesh decimation for model m303

to orient the object in an intuitive way. Step 3 performs a fliptest which result in the majority of the mass in the object to be located in the negative side of the axis. Finally step 4 normalizes the model, which is excluded in the figure because the model was already normalized and thus would show no difference.

2.5.1 Center on the barycenter

For the first step the barycenter b is calculated, which is the average x, y, and z coordinates of all vertices in the mesh. Next, each vertex v is translated, by subtracting these averages from each of its points resulting in the new vertex position u.

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} - \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

The result can be verified by calculating the average coordinates again, which should be 0 after the normalization.

2.5.2 PCA

Prinicipal component analysis is done using the arglib library. The three eigenvectors e_1, e_2, and e_3 are returned by the algorithm. A rotation matrix M is than created using the vectors x, y, and z, that represent the axises of the normal coordinate system.

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} M = \begin{bmatrix} x \cdot e_1 & x \cdot e_2 & x \cdot e_3 \\ y \cdot e_1 & y \cdot e_2 & y \cdot e_3 \\ z \cdot e_1 & z \cdot e_2 & z \cdot e_3 \end{bmatrix}$$

The new vertex u is obtained by multiplying this matrix with the old vertex v.

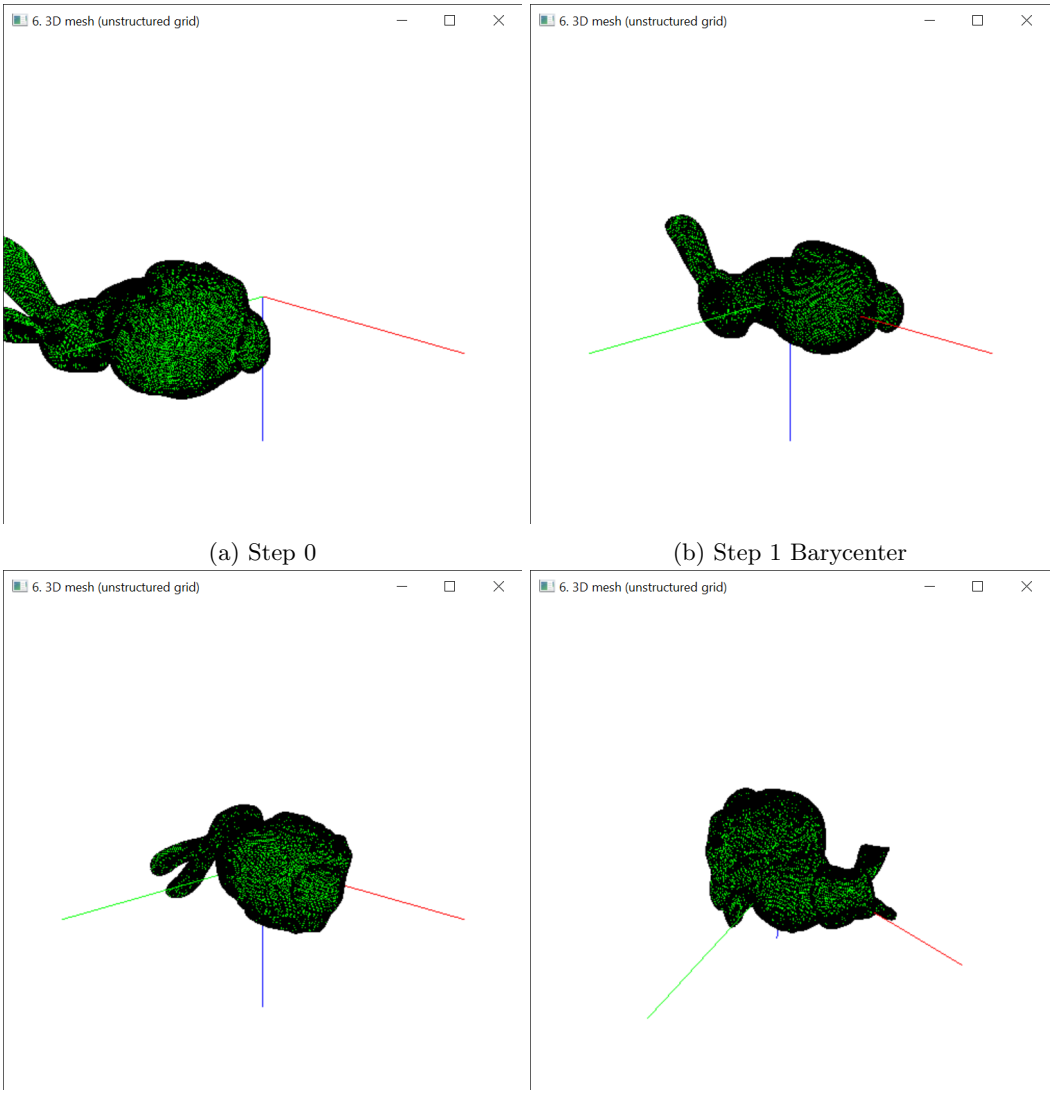
$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = M \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

The result of the translation can easily be verified. Doing PCA again on the translated mesh returns three eigenvectors that now correspond with the x, y, and z axis.

2.5.3 Fliptest

The eigenvectors used for the translation in the previous step are unoriented and thus give no information about to which side the model should be directed. Using the fliptest it is ensured that the majority of the mass resides in the negative half-space. Mass in this case is not indicated by the number of the vertices but also takes momentum into consideration, i.e. vertices farther away from the origin have a higher weight. Three variables are introduced: w_x, w_y, and w_z that indicate the total weight of each of three coordinates.

$$w_i = \sum sign(C_{t,i})(C_{t,i})^2 \tag{2}$$



(a) Step 0

(b) Step 1 Barycenter

(c) Step 2 PCA

(d) Step 3 Fliptest

Figure 4: Visualization of the Four step normalization pipeline.

where $C_{t,i}$ is the i^{th} coordinate of triangle t ($i \in x, y, z$). The latter part of the summation gives coordinates far away from the origin an higher weight, while the former part gives it either a negative or positive weight. These values are than used for a new scaling matrix M. Which flips the coordinates if necessary, in case the mesh is already properly oriented M will simply be the identity matrix.

$$M = \begin{bmatrix} sign(w_x) & 0 & 0 \\ 0 & sign(w_y) & 0 \\ 0 & 0 & sign(w_z) \end{bmatrix}$$

2.5.4 Normalization

The last step scales the model in the unit volume, i.e. it can fit in a unit cube. First the min and max of the x,y, and z coordinates of the axis-aligned bounding box is found. Next the largest distance δ between the min and max of these coordinates is used for the scaling factor $s = \frac{1}{\delta}$. Finally each vertex v is multiplied with this factor to obtain the new vertex u.

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \cdot s$$

A visualization is excluded, however the effectiveness of this step can easily be verified by looking at the resulting bounding box of the model.

2.6 Verifying the normalization

Next for each model in the database the following steps are performed: 1. if the model has ¿ 20.000 vertices use mesh decimation with target number of vertices set to 20.000. 2. if the model has ¡ 20.000 vertices use supersampling with target number of vertices set to 20.000. 3. Center the model on its barycenter. 4. Perform PCA and use align the eigenvectors with the x, y, and z axis. 5. Flip the model if necessary. 6. Normalize in a unit cube. In total 21 files were unable to processed, leaving a total of 1796 models in the new dataset. To verify that all the steps were performed correctly a new overview of the database can be created. The minimum number of vertices is 20.000, the maximum is 22963, the average is 200006, and standard deviation 116.61. These numbers show that the subsampling and supersampling worked correctly. There are a few models with a higher than average vertex count, because they were unable to processed by the *pmp* mesh decimation algorithm. However they are still included in the final dataset because their vertex count is close enough to 20.000.

3 Feature Extraction

There are two different types of features used to describe the meshes. The first are features which consist of a single float. And the second are the histogram features, which for a histogram with n bins contain n float values representing the fraction of values contained in each bin.

The definition of these features is discussed in the following subsections.

3.1 Features

3.1.1 Surface area

The surface area of the mesh is the sum of all the face areas. In the dataset all the faces are triangles, therefore the surface area is the sum of all these triangles. The area of a triangle is calculated using the $triangle_area$ function provide by the *pmp* library. The area of a triangle t with vertices u, v, and w is defined as followed:

$$area(t) = 0.5 * N((t_v - t_u) \times (t_w - t_u)) \tag{3}$$

Where $N(x)$ is the Euclidean norm of a vector x. The surface area feature F_{SA} for a given mesh M is than

$$F_{SA}(M) = \sum_i area(t_i). \tag{4}$$

where t_i is the ith triangle of mesh M. The minimum value this feature can have is 0, but does not have a clear theoretical maximum. The maximum as is displayed in table ? is 32.96, with an average of 1.54, and standard deviation of 1.65, which shows that this feature has large outliers. Therefore the feature cannot be normalized by extent normalization, and instead standardization has to be used.

3.1.2 Bounding Box Volume

The bounding box of a mesh is a rectangle that can be represented as two points, the opposite corners *bot* and *top*. The volume V is easily calculated as the $width\dot{l}ength\dot{h}eight$

$$bot = \begin{bmatrix} x_{min} \\ y_{min} \\ z_{min} \end{bmatrix} \; top = \begin{bmatrix} x_{max} \\ y_{max} \\ z_{max} \end{bmatrix}$$

$$V(bot, top) = (x_{max} - x_{min}) \cdot (y_{max} - y_{min}) \cdot (z_{max} - z_{min}) \tag{5}$$

The bounding box volume feature F_{BBV} for a mesh M with its bounding box opposite corners M_{bot} and M_{top} is

$$F_{BBV}(M) = V(M_{bot}, M_{top}) \tag{6}$$

The minimum value a bounding box can have is 0 when the mesh has no faces, and 1 when the bounding box is equal to the unit cube. Which is verified by the numbers in table ? for the BBV feature.

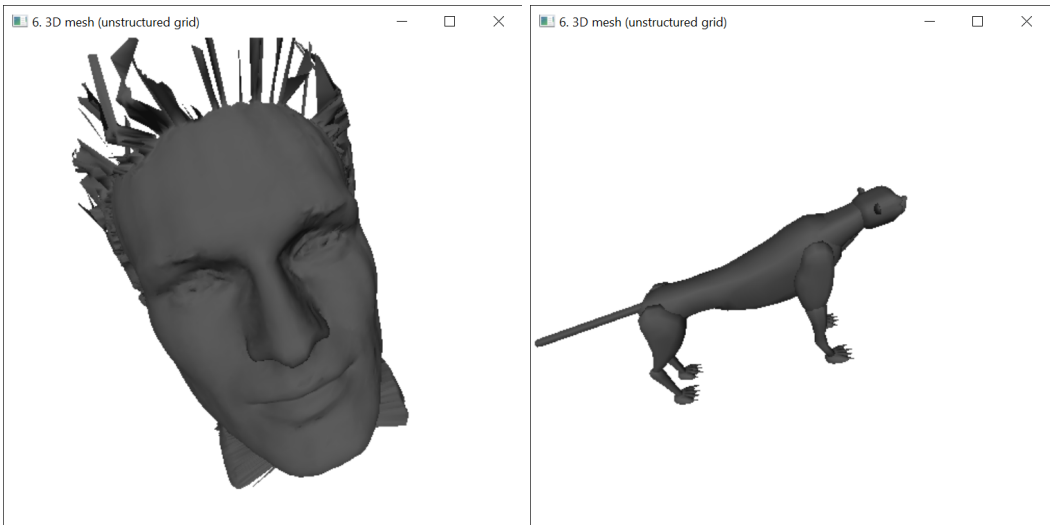
	min	Max	Avg	Stddev
SA	0.02194	32.961956	1.53768275	1.648700958
BBV	0.001874	0.986091	0.261523862	0.2203638
ECC	0.000004	0.956016	0.148089259	0.182812364

3.1.3 Eccentricity

The eccentricity of a mesh the ratio of the biggest eigenvalue of the covariance matrix to the smallest. The alglib library is once again used to do PCA and obtain the eigenvalues of the shape covariance matrix. For the eigenvalues e_1, e_2, and e_3 where e_1 is the largest value and e_3 the smallest, the eccentricity is

$$F_{EC}(M) = M_{e_3}/M_{e_1} \tag{7}$$

Where M_{e_3} and M_{e_1} are the smallest and largest eigenvectors of mesh M respectively. This describes the relation between the biggest and smallest eigenvectors, or also between the number of points spread in the x direction and the z direction. An object like a sphere or a head will have a high eccentricity, because their vertices are not spread out in a single direction. The picture of the head in figure 5 has an eccentricity of 0.72, while the leopard on the right has an eccentricity of 0.03.



(a) m303 (b) m94

Figure 5: Two models with different eccentricity values

The eccentricity can go close to 0 for long objects or reach 1 for a perfect spherical form. Furthermore, this feature could also be extended to also describe the relation ratio between the other eigenvalues, M_{e_2} / M_{e_1} or M_{e_3} / M_{e_2}.

3.1.4 Circularity and Compactness

Circularity is the inverse of compactness, a measure which represents how compact a shape is. From this perspective, circularity shows how "circular" a shape is. In 2D, compactness is minimized by a perfect circle, where its value is 1. The formula for 2D compactness is constructed as the dimensionless ratio between a shape's perimeter (P) and area (A), with constants added to the denominator such that it produces the value 1 for a circle. The formula is the following:

$$C_{2D} = \frac{P^2}{4\pi A} \tag{8}$$

Because the system requires such a metric to be used for 3D shapes, a similar formula has been derived. We begin by replacing perimeter and area with surface area (S) and volume (V), their equivalent 3D metrics:

$$\frac{S}{V} \tag{9}$$

As the result must be dimensionless, these measures must be raised to adequate powers so that their units of measurement cancel out:

$$\frac{S^3}{V^2} \tag{10}$$

Lastly, the constant (x) for the denominator must be determined by ensuring the derived formula will give 1 as the result if the shape in question is a perfect sphere:

$$\frac{(4\pi R^2)^3}{x(\frac{4}{3}\pi R^3)^2} = 1 \tag{11}$$

Solving for x, we obtain the result 36π. Thus, the final compactness formula is:

$$C_{3D} = \frac{S^3}{36\pi V^2} \tag{12}$$

As mentioned at the start of this question, we use the inverse metric of compactness, circularity, due to the fact that it produces values only between 0 and 1; 1 if the shape is a sphere, and very close to 0 if the shape has a very uneven contour.

$$Circularity = \frac{36\pi V^2}{S^3} \tag{13}$$

In the program, the two models in *Figure 5* above return expected results for circularity. For the human face (a), the computation returns a value of 0.311646, much larger than the one for the leopard (b), 0.059514.

3.1.5 Normalizing features

The three features bounding box volume, eccentricity, and circularity all fall in the range [0,1]. If the surface area feature is also to be normalized in this range, extent normalization has to be used by using its minimum and maximum value. However this feature has large outliers. For example the minimum of 1 and maximum of 36 is used to normalize, the distance between a surface area of 3 and 1 would become 0.0571. Even though these 2 meshes would be very different in terms of surface area, their distance does not reflect this. Therefore standardization is used for this feature, and to be consistent, it will also be used for all the other features. Each new feature value will be calculated with the following formula

$$f'_{i,j} = \frac{f_{i,j} - \mu_i}{\delta_i}, \quad i \in \{SA, BBV, ECC, DI, C\} \tag{14}$$

Where μ_i is the average for feature i, δ_i the standard deviation, and $f_{i,j}$ the feature value for the jth mesh. This standardization changes the average of a feature to 0 and the standard deviation to 1. Meaning that if two different feature values differ one standard deviation from each other, there distance will be one. The table below shows the new description values of the different features.

	min	Max	Avg	Stddev
SA	-0.919356	19.06002	0	1
BBV	-1.178278	3.28805	0	1
ECC	-0.810042	4.419431	0	1

3.2 Histogram features

For extracting the histogram features random numbers will be generated. This is done by using the c++ function rand(). For the features that require multiple random numbers it is possible to generate duplicates, which would create large outliers. Therefore the numbers are re-sampled if there are any duplicates. Furthermore for obtaining the values of the histograms 20.000 of these samples will be taken for all histogram features. For each histogram feature they will first be generated using a minimum and maximum value based on their theoretical maximum. These results will than be used to adjust the minimum and maximum so that there wont be any bins that are universally unused. The number of bins used bins that will finally be used is set at 12.

3.2.1 D1 - distance to barycenter

This feature calculates the distance between a randomly generated point and the barycenter. The distance D between two vectors u and v is calculated as follows

$$D(u, v) = \sqrt{\sum_{i=1}^{3}(u_i - v_i)^2} \tag{15}$$

The minimum distance is obviously 0, and the maximum distance is equal to the diameter of a unit cube, $\sqrt{3}$, which is an unrealistic scenario. In order the evaluate the quality of the features, the histogram distributions of different classes will be plotted on a graph. The blue lines represent the histogram distributions of models m93, m94, m95, m96, and m97, which are all leopard like animals. And the yellow lines are models m1120, m1121, m1122, m1123, and m1124, which are air planes. These graphs will be made for each of the histogram features.

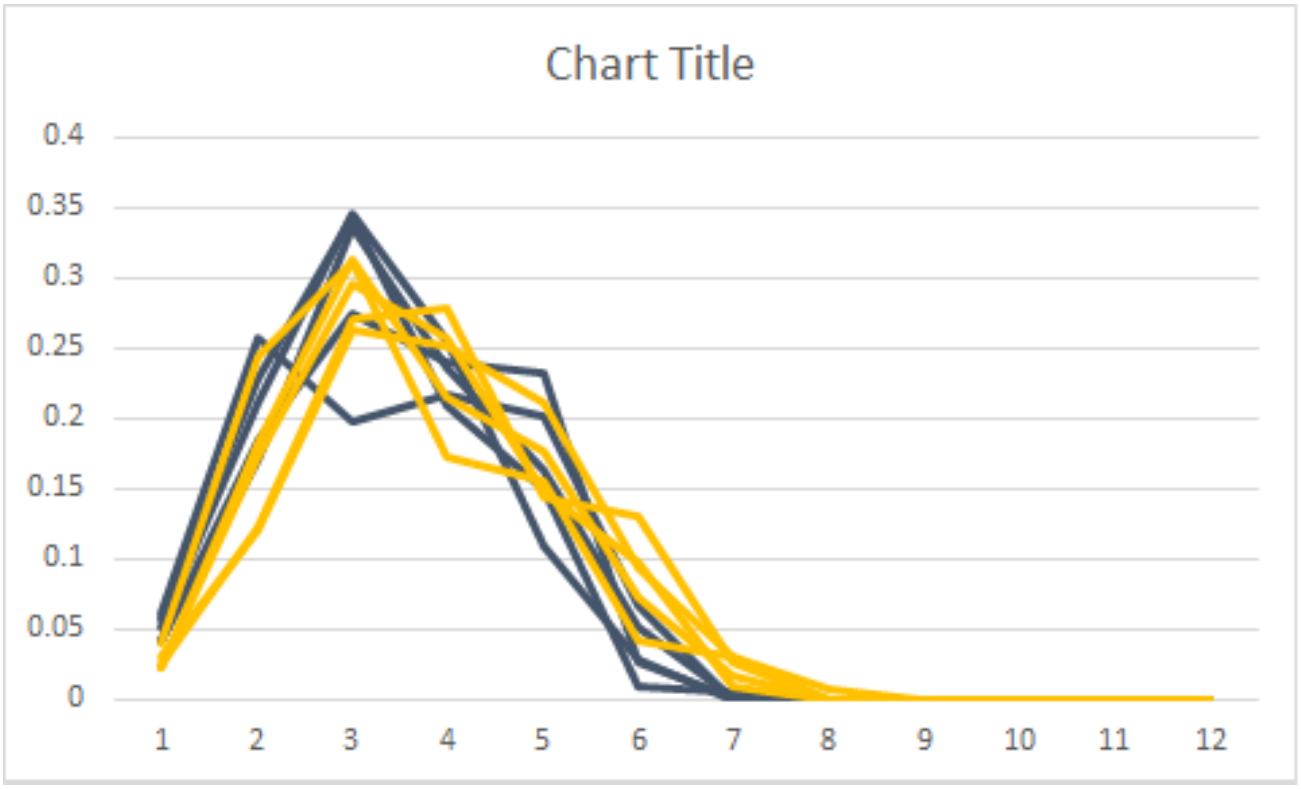


Figure 6: Histogram distributions for feature D1 for different classes. Yellow = air planes, Blue = leopards

3.2.2 D2 - distance between two points

This time two random points are generated and the same distance function as for the D1 feature is used. The min and maximum distances are the same as the previous feature, all though the distances will generally be higher.

3.2.3 D3 - area of a random triangle

To get a random triangle three random points are generated. The area of this triangle is calculated the same way as was done for calculating the total surface area. The square root of this value is than added to the histogram. For the area of a triangle the minimum value is 0.

3.2.4 D4 - are of a random tetrahedron

Four random points a, b, c, and d are generated that together will make the tetrahedron. The formula used for calculating the volume is:

$$V(a,b,c,d) = \frac{|(a-d)\cdot((b-d)\times(c-d))|}{6} \tag{16}$$

3.2.5 A3 - angle between 3 random vertices

The last histogram features uses three random points u, v, and w to calculate the angle between them. These three points create a triangle that has three angles, however the order in which they are generated determines which angle is chosen. Therefore all three possible angles are calculates, and the maximum of those is chosen.

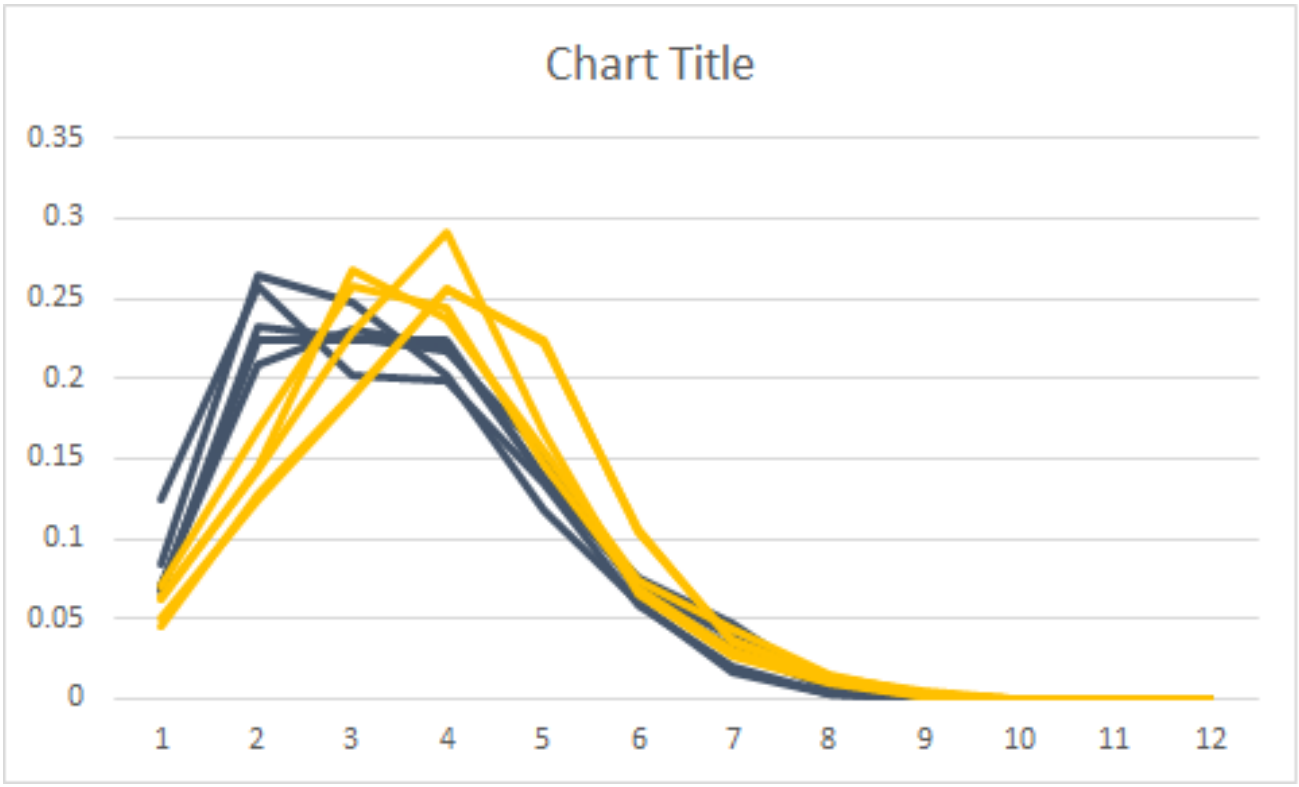


Figure 7: Histogram distributions for feature D2 for different classes. Yellow = air planes, Blue = leopards

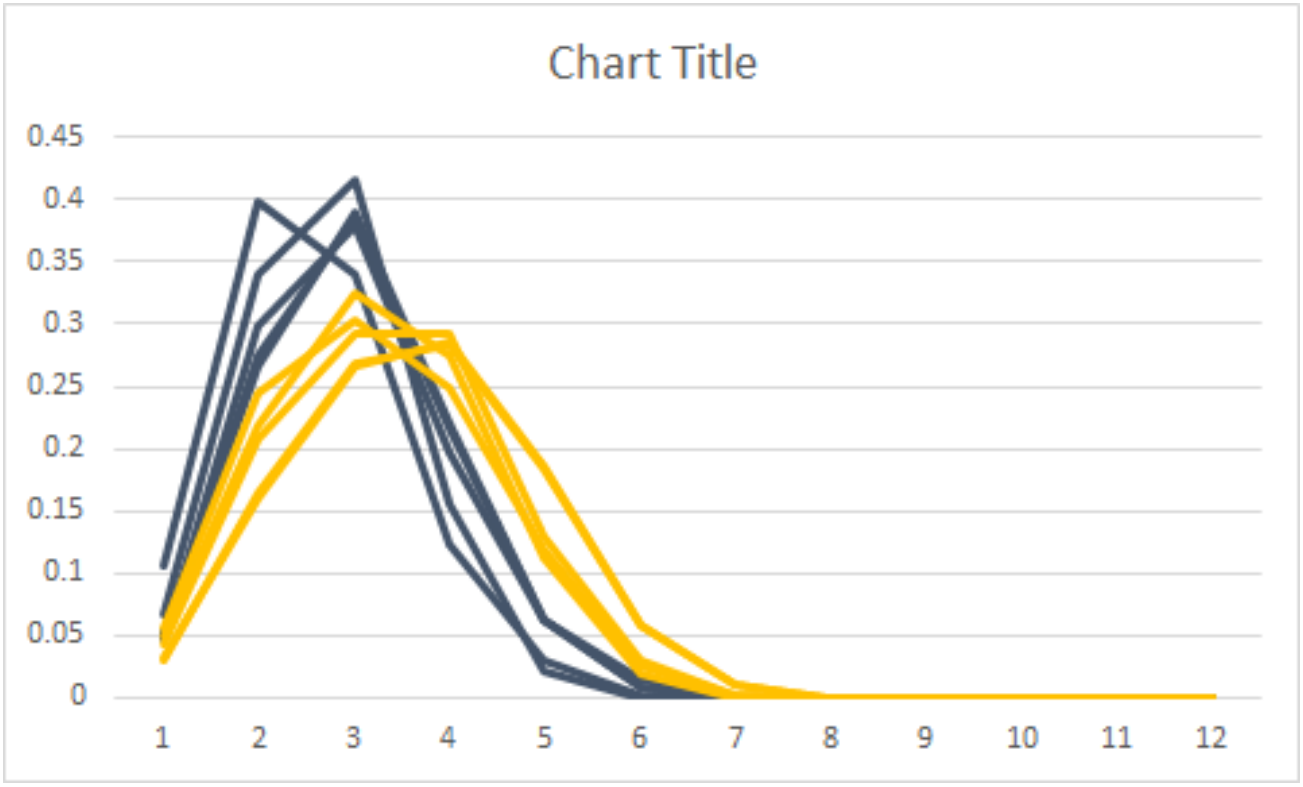


Figure 8: Histogram distributions for feature D3 for different classes. Yellow = air planes, Blue = leopards

For each angle the distance is chosen by using two vectors that share a similar point. For example, the vectors $a = v - u$ and $b = w - u$ give one possible angle of the triangle. The formula used for calculating an angle given

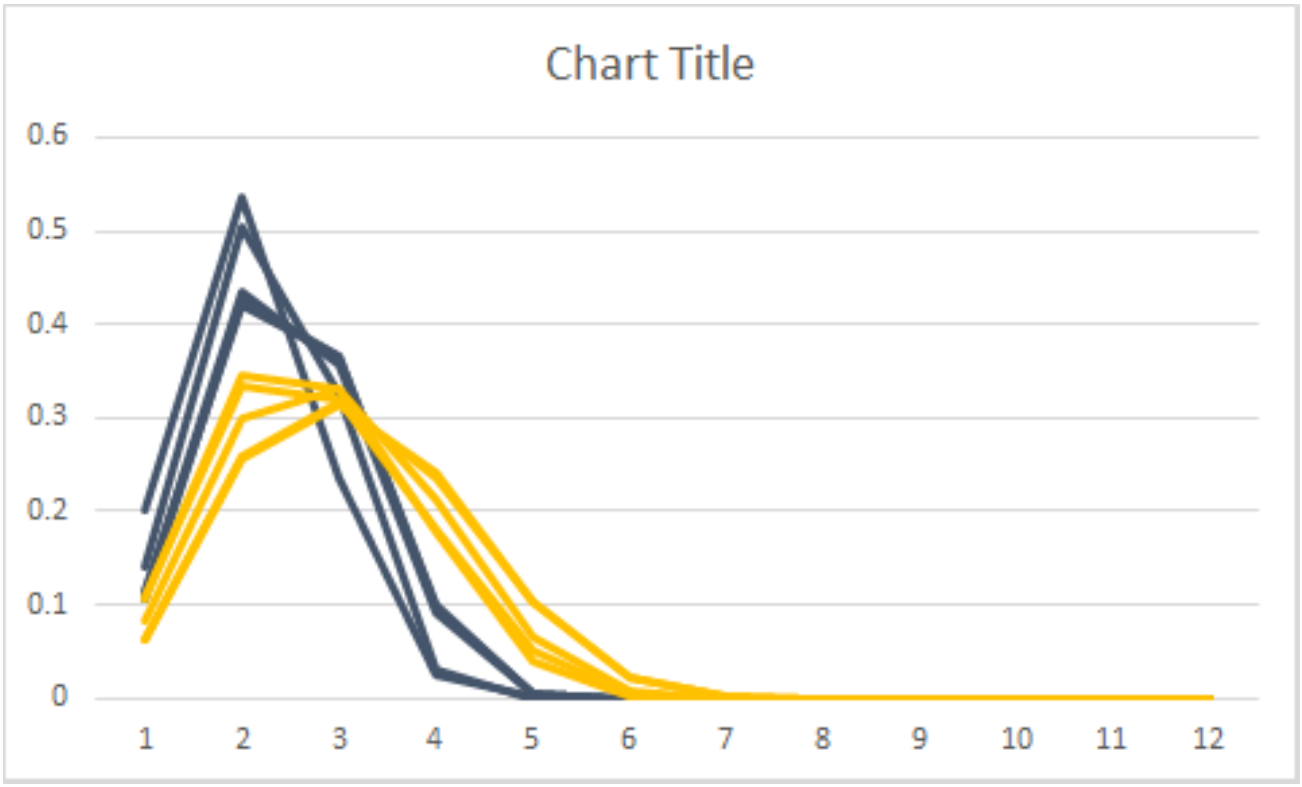


Figure 9: Histogram distributions for feature A3 for different classes. Yellow = air planes, Blue = leopards

two vectors is as follows:

$$Angle(a, b) = atan2(N(a \times b), (a \cdot b)) \tag{17}$$

Where N gives the euclidean norm of a vector, and atan2 the angle in the euclidean plane. The minimum value that the angle could be is 0, and the maximum is π.

4 Matching

All the features discussed in the previous section are calculated for each mesh and put in a single feature vector $x = (x_1, ..., x_n)$ where each x_i is a single feature. To compare different meshes the distance is to be calculated between their two feature vector. If feature vector x is compared with a different feature vector $y = (y_1, ...y_3$, each of their features will be compared individually and the sum of these distances will be regarded as the distance between the two feature vectors.

4.1 Distance calculation

Each feature has m values, for features $m = 1$ because it only has value, for histograms m is equal to the number of bins of that histogram. The distance function $L_(x, y)$ gives the total euclidean distance between two feature vectors x and y

$$L(x, y) = \left(\sum_{i=1}^{n} d(x_i, y_i)^2 \right)^{1/2} \tag{18}$$

Where d is a function that calculates the distance between two features

$$d(x_i, y_i) = \frac{\left(\sum_{j=1}^{m} |x_{i,j} - y_{i,j}|^2 \right)^{1/2}}{m} \tag{19}$$

Where $x_{i,j}$ is the jth value of the ith feature of feature vector x, the same case for y. The sum of all the distances between feature values is divided by m so that each feature has the same weight, i.e. a histogram with 8 values should not be weighted 8 time as much as a scalar feature which has only 1 value.

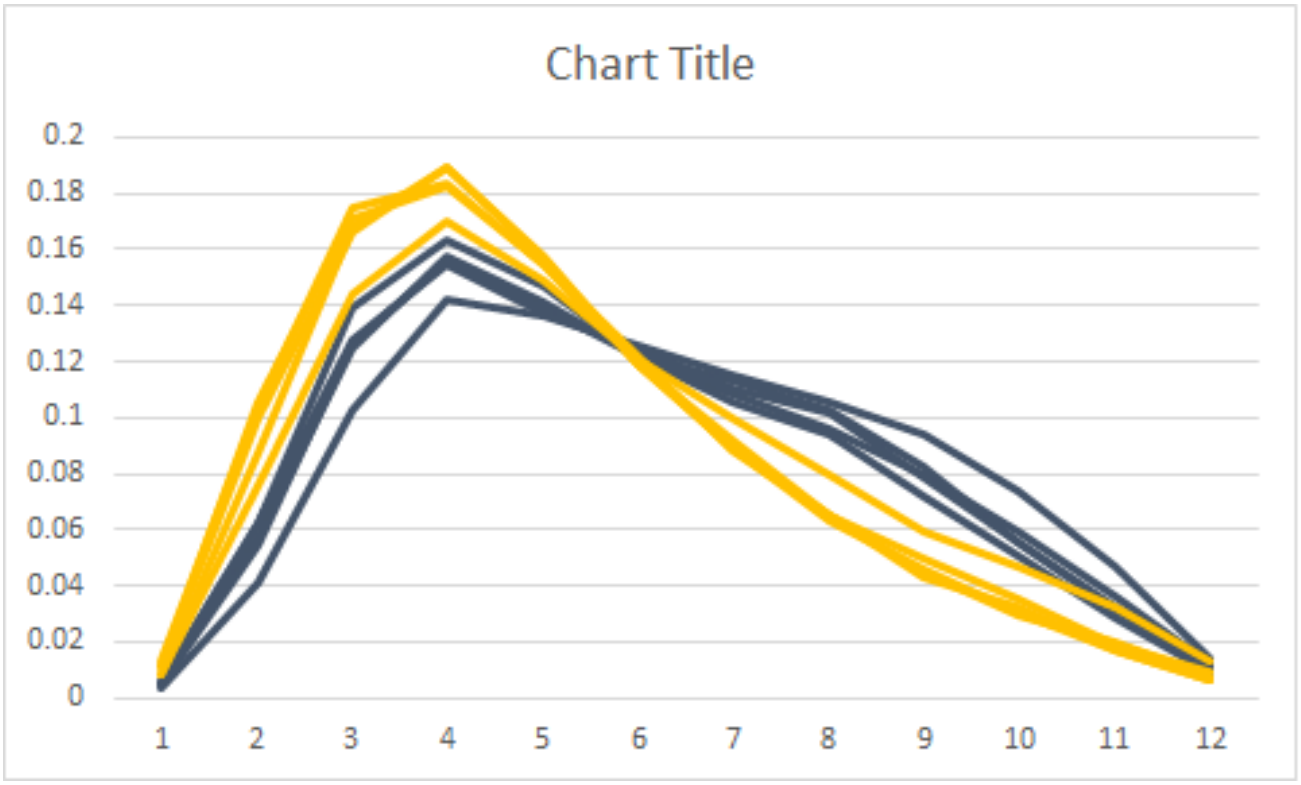


Figure 10: Histogram distributions for feature A3 for different classes. Yellow = air planes, Blue = leopards

4.2 Query result

These distances are calculated for each mesh in the database, obviously excluding the the query item, if it exists. The result is thus a list of distances for a query item i $L_i = d_1, ...d_k$, where k is the number of items in the database, and is ordered in ascending order. With an input query size s the first s elements of L are than returned as the query result.

5 Evaluation

The final step is to evaluate the performance of the retrieval system. This is done by analysing the query result Q which as a list of items $x_1, ..., x_s$, where s is the size of the query and each x is an item in the database. As was explained in the matching section, this list is ordered on ascending distance to the query item l. First of, there are several values that describe a query result, namely the true positives (TP), false positives (FP), true negatives (TN), and false negatives (FP). TP is the number of items with the correct class label in the query result.

$TP = count(\{y \in Q \mid c_y = c_l\})$
$FP = s - TP$
$TN = k - TP - FP - FN$
$FN = N_{c_l} - TP$

Where c_i is the class label of item i, k the number of items in the dataset, and N_{c_l} the number of items in the class of query item l.

5.1 Evaluation metric

There are many evaluation metrics that can be used. Furthermore, there are different aspects of the system for which the performance can be evaluated. For example the performance for a single item in the dataset, the average performance of a class, and the overall performance of the program. In the evaluation of this system the performance for each item will be calculated and, from those measurements, the class averages, and overall performance will be derived. Two interesting evaluation metrics are precision and recall. Precision shows how

many of the returned items in the query are of the correct class, while recall shows how many of the correct class are represented in the query. The problem with recall however is that it is highly dependent on the size of the query. A query of size 1 can never have a recall of 1, assuming that the class size is higher than 1. Therefore the recall is not a useful performance metric for this system, because it gives little insight. A large query size will result in a high recall and a low query size in low recall. The more interesting metric is precision whose definition is given below.

$$Precision = \frac{TP}{TP + FP} \tag{20}$$

Another problem that arrives is what query size the evaluation metric should be calculated for. A universal query size is undesirable, because there are large discrepancies between the sizes of different classes. A class with only five items will have a terrible performance when the query size is set to 100, because the maximum precision is $\frac{5}{100}$. Furthermore, even for a specific class, choosing the query size is an arbitrary decision. Therefore the Mean Average Precision (MAP) is used. Instead of choosing a single query size, $k = N_{c_l}$ different query sizes will be chosen. The query result that is returned by the matching algorithm is an ordered list of ascending distance values of size d, where d is the size of the database. Then, for each $i \in 1, ..., k$, the precision and accuracy are calculated for the first i elements in the query. The precision for a specific model is then the average of these values. The MAP quality metric is the mean of these averages across all models. Furthermore, the MAP can also be calculated for each individual class, giving insight into what types of models the retrieval system works best on.