Multimedia Retrieval

3D Model Search Engine Based on Lightfield Descriptors

Bojan Endrovski

Stefan Hospes

31-10-2009

1 Introduction

Most 3d similarity methods focus on similarity of geometric distributions of models rather than searching on visual similarity. In this paper we use an approach that uses the visual similarity of models to compare and compute the distance between these models. This method (Yu-Te Shen, 2003) focuses more on the visual perception of 3d models by humans.

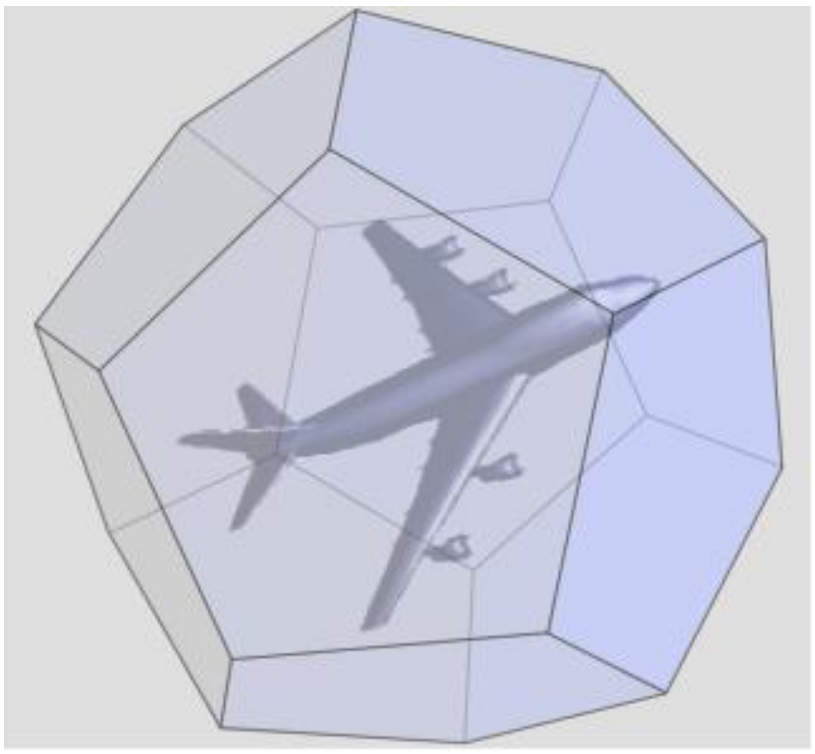
2 Method overview

The main idea of comparing models based on visual similarity is explained by the following quote: "If two 3D models are similar, they also look similar from all viewing angles". So essentially, the similarity of two 3D models can be defined by summing up the similarity of all viewing angles. However, sometimes 3D models are translated, rotated or scaled. In order to compare 3D models that are transformed using visual similarity, we use lightfield descriptors.

2.1 Lightfield descriptors

A lightfield descriptor is in short: a 4D representation of a 3D model using 2D images. Basically, these 2D images are rendered from an array of camera's which are distributed uniformly around the 3D model. With this render, all lights are turned off so that only silhouettes remain. The 3D model is now represented by a list of 2D image silhouettes. Due to performance limitations, the number of 2D images that represent the 3D model cannot be too high. Therefore we distribute the camera's on vertices of a regular dodecahedron, which has exactly 20 vertices. From these 20 vertices, there will be 10 unique rendered images. This is because in a dodecahedron all vertices will have another point lying exactly on the other side of the model. This will result in the same silhouette of the 3D model, only rotated. Therefore these images are not used, which leaves 10 images per lightfield.

Figure 1 - Dodecahedron around 3D Model



2.2 Sets of lightfields

A 3D model is not described by one lightfield, but by a set of lightfields in order to improve robustness against rotations of 3D models. Some models might be rotated around the axis, which will result in different silhouettes if only one lightfield is used. Therefore we use multiple lightfields which have a slightly rotated dodecahedron. All these lightfields have 10 images which contain the visual information of the model. Comparing two models based on sets of lightfields is now a problem of comparing sets of lightfields of each model against each other. The minimum distance between two models is the smallest distance between any lightfield of one model against each other.

2.3 Image processing

In order to compare two lightfields, 2D images have to be compared. The comparison has to be robust against rotation, transformation and image distortion. To best achieve this we extract feature vectors from the image which are transformation invariant. These feature vectors can be classified into region-based and boundary-based descriptors. Region based feature vectors can be computed by a Zernike transform. The result is a vector that describes the region features of a 2D image but neglects shape information. Boundary based descriptors can be computed by doing a Fourier Transform on the boundary of the shape. This results in a vector that describes the shape of a 2D image but ignores region information. It is shown in paper (Zhang et al, 2002) that if both vectors are combined, the comparison function performs better then situations where the vectors are used individually. Therefore we use a combination of both Zernike and Fourier feature vectors to represent the image. Now the distance between two images can be defined simply as computing the Euclidian distance between the combined feature vectors of both images.

2.3.1 Zernike transform

Zernike complex moments are constructed using a set of Zernike polynomials. Zernike polynomials are a sequence of orthogonal polynomials on the unit disk and they can be expressed as (Weizhuo Wanga, 2008)

$$V_{n,m}(x,y) = V_{n,m}(\rho,\vartheta) = R_{n,m}(\rho)e^{im\vartheta}$$

where $= \sqrt{-1}$, *n* is the order of the radial polynomial and *m* a positive and negative integers subject to constrains $n - |m|$ even, $|m| \leq n$, representing the repetition of the azimuthal angle. From *x* and *y* we obtain ρ and ϑ by simple conversion to polar coordinates. The radial polynomial $R_{n,m}(\rho)$ is defined as

$$R_{n,m}(\rho) = \sum_{s=0}^{(n-|m|)/2} (-1)^s \frac{(n-s)!}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n+|m|}{2}-s\right)!}\rho^{n-2s}$$

From the definition itself we can see that $R_{n,m}(\rho) = R_{n,-m}(\rho)$. We will use this property later in the implementation. As we are concerned only with calculating the Zernike moments for a digital image we will skip directly to the formulation of the moments in adequate form

$$Z_{n,m} = \frac{n+1}{\pi}\sum_x \sum_y I(x,y)V^*_{n,m}(x,y), x^2 + y^2 \leq 1$$

With rotation by an angle α we get $I(\rho, \vartheta) = I(\rho, \vartheta - \alpha)$ and by simply substituting that into the equation we get

$Z^r_{n,m} = Z_{n,m}e^{-im\vartheta}$ and $\left| Z^r_{n,m} \right| = \left| Z_{n,m} \right|$

Which indicates that the modulo of the rotated image is the same as the original, thus using the modulo as a feature descriptor will yield rotation invariance and make the method more robust. As suggested by the original paper we use 36 unique coefficients from the Zernike moments up to order 10.

2.3.2 Fourier transform

The Fourier transform uses boundary information to compute the feature vector. First the boundary of the silhouette has to be found. Then, we can compute the center of mass of the boundary points. We simply take the average of all x and y coordinates and define the result as the x and y center of mass. Then, the boundary can be traced and the distance from the boundary point to the center of mass can be computed using the Euclidian distance. The resulting list is a 1 dimensional function that describes the boundary of the object. This list is rotation invariant because distances are expressed as distances from the center of mass, which are rotation invariant.

Now a Discrete Fourier Transform is computed with the 1D signal as input. We use the following function as described in [Zhang]:

$$a_n = \frac{1}{N}\sum_{t=0}^{N-1} r(t) \exp\left(\frac{-j2\pi nt}{N}\right), \quad n = 0, 1, .., N-1$$

This function will return a number of coefficients a_n, n = 0,1,.., N – 1, that describe the input signal as a combination of frequencies. When all frequencies are added together the 1D input signal will appear again. The lower coefficients describe overall features of the shape whilst the higher coefficients describe more precise features of the shape. We use only a set of lower coefficients to represent an image because we are only interested in comparing global features of a shape. Furthermore the coefficients are normalized to make the Fourier Transform scale invariant.

$$f = \left[\frac{|a_1|}{|a_0|}, \frac{|a_2|}{|a_0|}, \ldots, \frac{|a_{N/2}|}{|a_0|}\right]$$

This resulted in a set of coefficients which we use to compare models. We computed 10 coefficients to describe the image. To compare two sets of Fourier coefficients we used the Euclidian distance.

2.4 Comparing

2.4.1 Two 3D models

As previously stated we use multiple lightfields to account for different orientations of the models. The measuring of the dissimilarity (distance) between two models is done by comparing all the lightfields from the first model with all the model of the second model and taking the orientations with minimal distance.

2.4.2 Two lightfields

Each image is now described by a set of Fourier and Zernike descriptors which have been computed from the 2D images. These two sets of descriptors can be combined using the following function:

$$fz=\{\alpha\cdot FD_1,\ \alpha\cdot FD_2,\ \ldots,\ \alpha\cdot FD_m,\ \beta\cdot z_1,\ \beta\cdot z_2,\ \ldots,\beta\cdot z_n\}$$

Alpha and beta are weighting factors for both descriptors. The resulting descriptor describes the 2D image. Comparing two descriptors is as simple as taking the Euclidian distance. Now a lightfield consists of 10 of those descriptors. While the image metric itself is rotation invariant in image space, we do have to take into account the different possible orientation of the model relative to the dodecahedron used for the lightfield. For all lightfields the images have been rendered in the same order from the different vertices on the dodecahedron. The distance between two lightfields is defined as the best of the 60 possible, 20 vertices each with three neighboring vertices, orientations

$$D_A = min_i \sum_{k=1}^{20} d(I_{1k}, I_{2k}), \qquad i = 1..60$$

Where I_{1k}, I_{2k} are the corresponding images under the i-th rotation.

3 Implementation

Our programs are written in C# (.NET) and we use a couple of shell scripts to execute these programs. There are 3 different programs.

The first program renders the lightfield descriptors using Direct X (XNA). This offloads the rendering to the GPU of the computer which speeds up the process. This program is called using a shell script which iterates through the model directories and computes 10 images for 10 lightfields.

The second program computes the Zernike and Fourier descriptors of each image and stores the result in the model directory. This program uses the basenames file to iterate through the model directory. With the features extracted the offline processing part is complete and new models can be queried against the database.

The third program computes the distance between all 3D models in the models directory. It takes one model and then computes the distance to every other model in the models directory and stores this. Every model is iterated.

3.1 Rendering

The rendering of the models is done in hardware using the XNA platform. The mesh is created as a pair of vertex buffer and index buffer on the GPU. We have already seen that the method does a lot of work to make the distance measure as rotation invariant as possible, so explicit normalization of the model is not necessary. Nevertheless the translation and scaling should be handled in some way. The centroid of the model is used as pivot for all the rotations, so carefully picking it is very important for the robustness of the whole system. The centroid of the model is calculated from its surface, taking the area of each triangle into account. This method gave much better results than using the average all the vertices, which can be biased towards areas modeled with higher detail. As previously stated we would like to use 10 images for the 20 points of view, for exactly this reason we use orthographic projection. The projection is simply constructed to fit the model into the disk in the center of the image, implicitly handling uniform scaling. While this doesn’t use the whole resolution of the image, it enables the whole shape to be used for the calculation of the Zernike moments.

The ten different orientations are calculated by creating a random vector, uniformly distributed over the unit sphere, and forming an orthonormal basis using the vector. As models are often modeled in a similar manner, we keep the first orientation to the original unrotated. In addition, for easier debugging we use pseudo random orientations, so that we get the same orientations for each model in consecutive runs.

Each time frame is used to render a single image in the back-buffer, with size 256x256 as suggested by the original paper, later resolved to a texture and saved to disk. This makes rendering extremely fast.

3.2 Zernike Moments

3.2.1 Implementation

For the extraction of the Zernike moments an implementation was created straight from the mathematical formulation. This execution speed of this naïve implementation made it unusable even for the database of 493 models. Analysis showed few bottlenecks in the execution. As multiple passes over image need, the first choice was to simply copy the data to the local memory, bypassing any image API and use a more cache coherent access. This resulted in up to five times faster execution. Despite that few early outs have reduced the execution time; still most of the time was being spent on calculation of the Zernike polynomials and their values. Caching of the Zernike polynomials and calculating the value using Horner's rule improved the performance significantly, but it would still take more than 8 hours for the whole database. The final improvement was to use all available CPU cores on the system by employing multiple threads. With the write access in the polynomial cache being the only block of code needing thread locking, the speedup scales linearly with the number of available cores. This finally led to processing time below 8 hours on a Core 2 Duo, so that processing can be done overnight.

3.2.2 Proof of correctness

Whether the calculation of the Zernike moments was correct is not easy conclude directly from calculated complex values even for simple inputs, like a rectangle or a triangle. Instead we try to reconstruct the original image from the coefficients. The image can be easily reconstructed as

$$I(\rho, \vartheta) = \sum_{n=0}^{N_{max}} \sum_{m} Z_{n,m} V_{n,m}\,(\rho, \vartheta)$$

where N_{max} is the order of the polynomials that ware used in feature descriptors and m is constrained as previously. The property $R_{n,m}(\rho) = R_{n,-m}(\rho)$ of the Zernike polynomials enables us to use the 36 coefficients (instead the full 55) to reconstruct the full image up to order 10. We can see the results for orders 10, 20 and 30

Figure 2- Original image

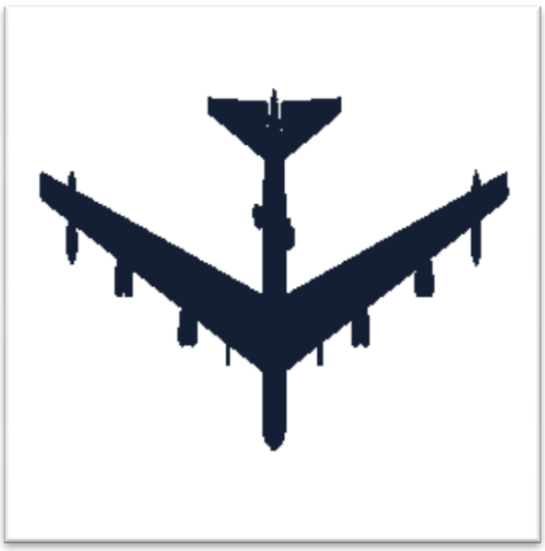


Figure 3 – Order 10

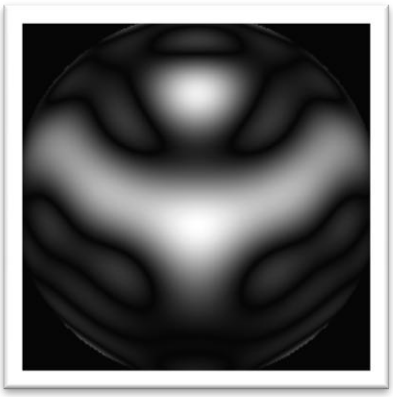


Figure 4 – Order 20

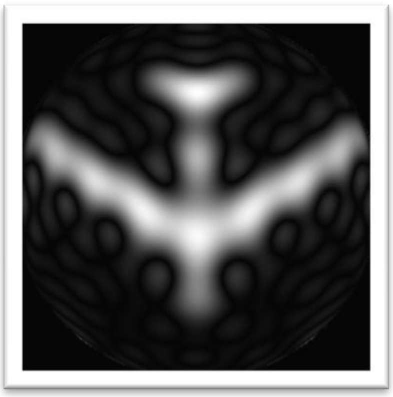
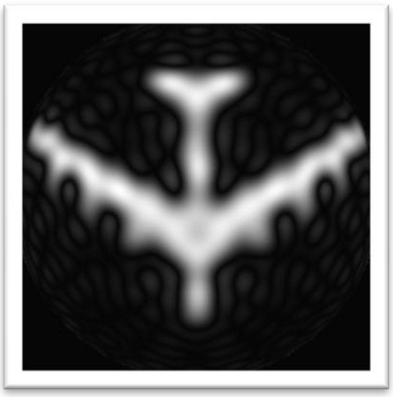


Figure 5 – Order 30



As the reconstruction takes equal amount of time as the extraction itself it's switched off by default. If needed for debugging purposes can be easily switched by defining a RECONSTRUCT symbol during compilation time.

3.3 Fourier Transform

3.3.1 Implementation

The first step in computing the Fourier transform of an image was extracting the boundary of the shape. The silhouette images itself had some holes and segmented parts in it. So we first performed a closing and opening on the image to filter away noise pixels and connect segmented parts. We used an external library (AForge) to do this image processing.

Now the boundary could be traced. A problem was finding the starting point on the boundary of the object. The first approach was iterating through the image file from the left top, row by row, and selecting the first image point as starting point. But because the image might still be separated, there was a possibility that just a small segment of the object was found instead of the complete object.

The second approach was trying to find the starting point from the center of the object instead from the left top. The image was traced upwards from the center of the image until the boundary had been found. Some images however did have a hole exactly in the center of the image, therefore the method had to be adjusted to also continue to iterate until the last appearance of an object pixel had been made. This resulted in a stable starting point selection.

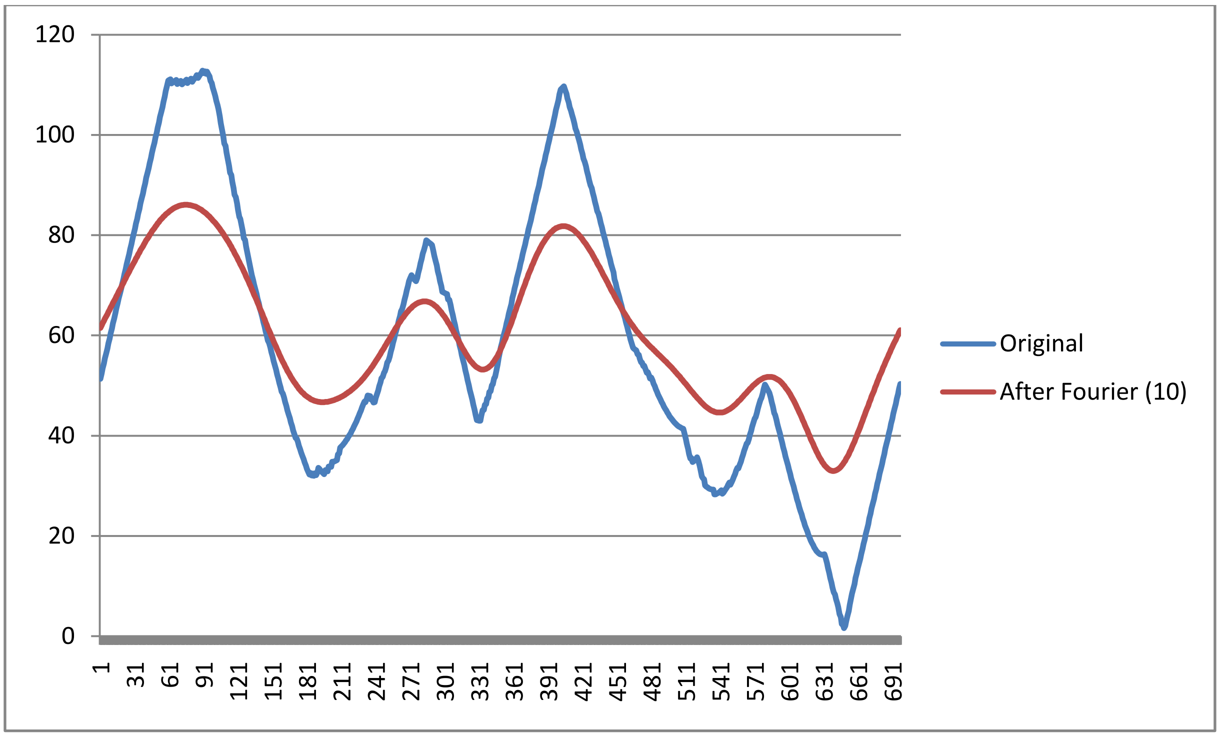
Then the boundary had to be traced, this was done using a simplified version of the method described in (Ren et al, 2002). This method used the boundary pixel and the previous boundary pixel to trace to boundary in the correct direction. Once the boundary had been traced, the average value of all x and y coordinates was computed in order to find the center of mass of the object. With this center of mass the centroid distance of the boundary was computed and the distance signal could be extracted. Using this signal the Fourier transform was computed.

The implementation was able to compute the Fourier transformation of an image in roughly 0.5 seconds, this was a lot less then the Zernike implementation but still not acceptable. Using the same methods as the Zernike implementation the transformation was done in 0.2 seconds. This was improved more by using threads to fully utilize the power of the CPU cores.

3.3.2 Proof of correctness

For the Fourier transformation we use the centroid distance. Therefore we cannot construct the same image using the coefficients. But we can construct the 1D signal from those coefficients, using the reverse Fourier transform. The following graph has the 1D signal first, and then the reconstructed signal using 10 coefficients.

Figure 6 - 1D signal of centroid distance



It is obvious that using 10 coefficients results in the reconstruction of only the low level features of the signal. The 10 coefficients can be used to compare the image with other images.

The Fourier transform also has to be translation invariant, this can be proven by computing the Fourier transform on a number of rotated images.

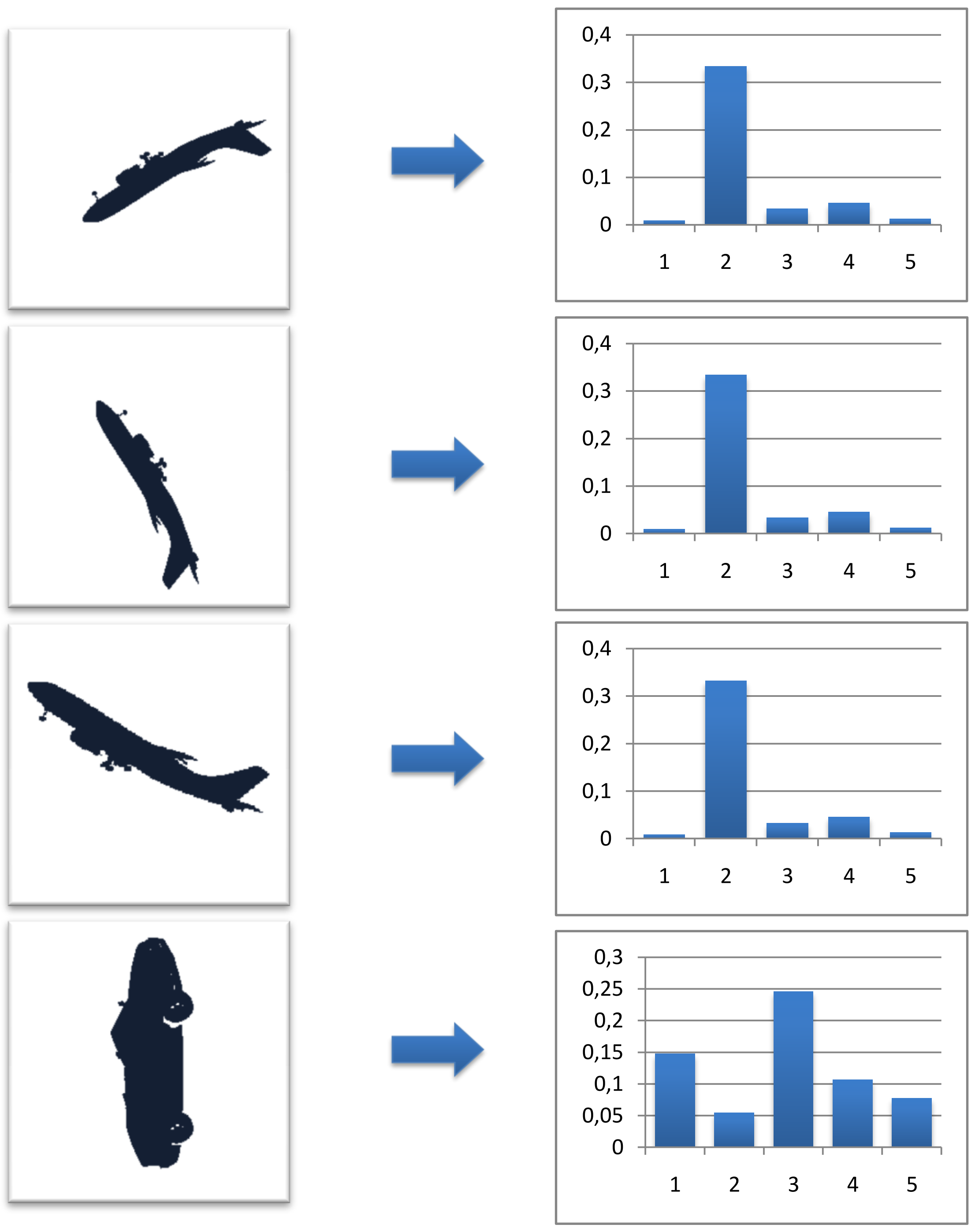


Figure 7 – Fourier coefficients of various silhouettes

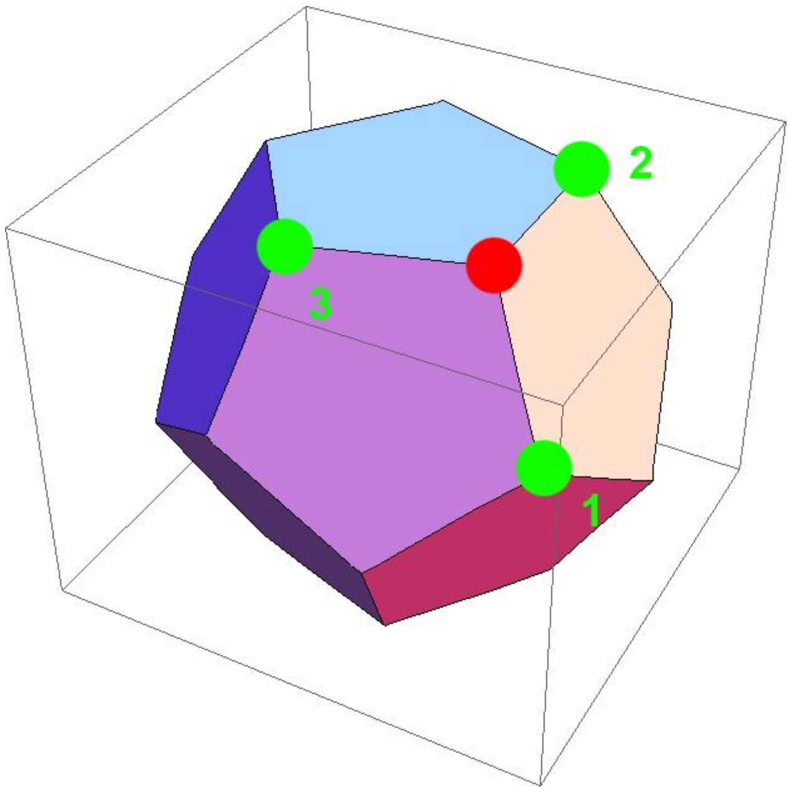
For every image the first 5 Fourier descriptors were computed and put into a graph. The Fourier transform of the rotated and scaled images is almost the same, the difference is not visible in the graph. A different shape returns completely different coefficients.

3.4 Comparison

Implementing the comparison between models is straightforward and follows the guidelines of the original paper. We compare all the lightfields of one model with all the lightfields of the other one and return the one with minimal distance. Each lightfield itself can be used to represent 60 rotations; the one with minimal distance is optimal.

The original paper does not cover method used for generating the 60 rotations. This is a combinatorial problem that we can solve by doing a breadth-first search on the dodecahedron graph. The technique can be illustrated in the figure bellow. We use a single ordering of vertices from the first dodecahedron that will be used for all comparisons. Than we choose one of the points on the second dodecahedron and choose an ordering of the three neighboring vertices. A breath-first search will yield a unique ordering. We can repeat this step by choosing a new pivot point and ordering of the neighboring points.

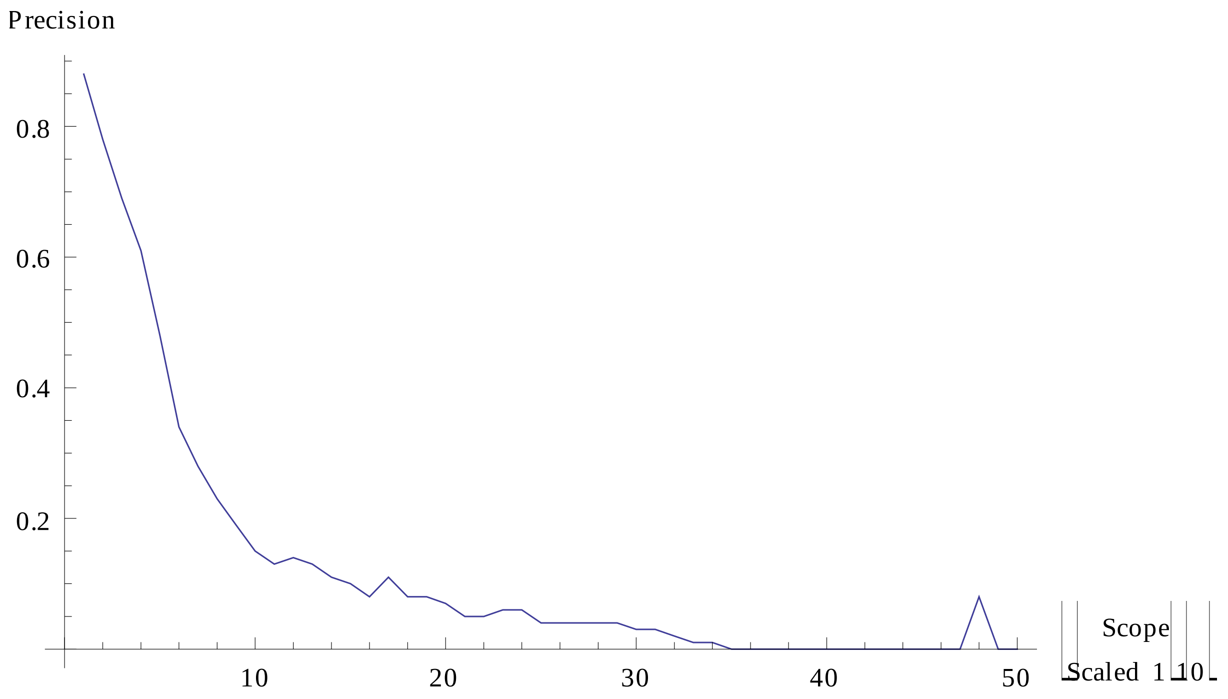
Figure 8 - Dodecahedron



4 Performance

The method gives opportunity for great discriminating power and it is also very adaptable due to the easy balancing of the Zernike and Fourier coefficients with the α and β are weighting factors. The original paper does not give any rule of thumb values for these factors, so multiple trail runs were concluded in order to derive values that will give good results. Best overall results for the given database were obtained for α = 4.0 and β = 1.0. Next is a graph of the average precision across all the models in the AIRCRAFT class as a function of the scope:

Figure 9 - Average precision for the aicraft class



Some of the classes of object have quite dissimilar objects within the class. Cars and aircrafts being one of the most uniform will be used in the next graphs. The fist one is a precision vs. recall graph using the AIRBUS model and the second one using AUDIAVUS as a query model.

Figure 10 - Precision Recall graph using AIRBUS model

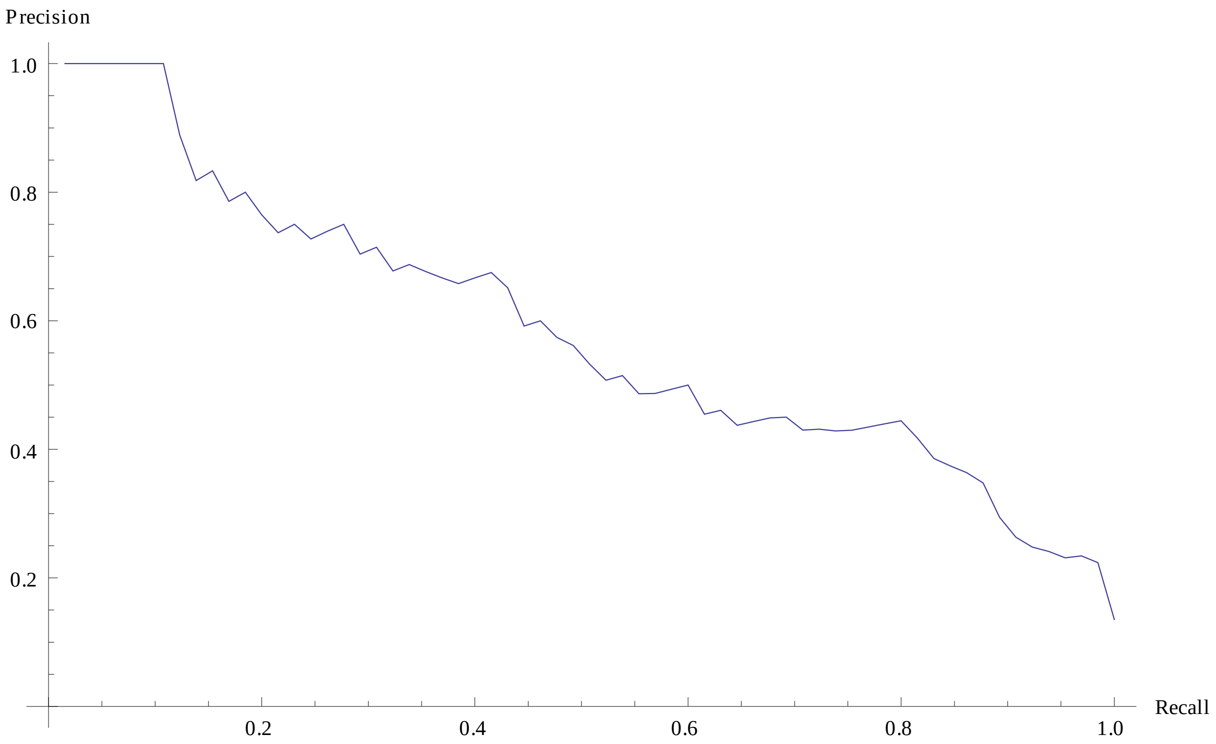
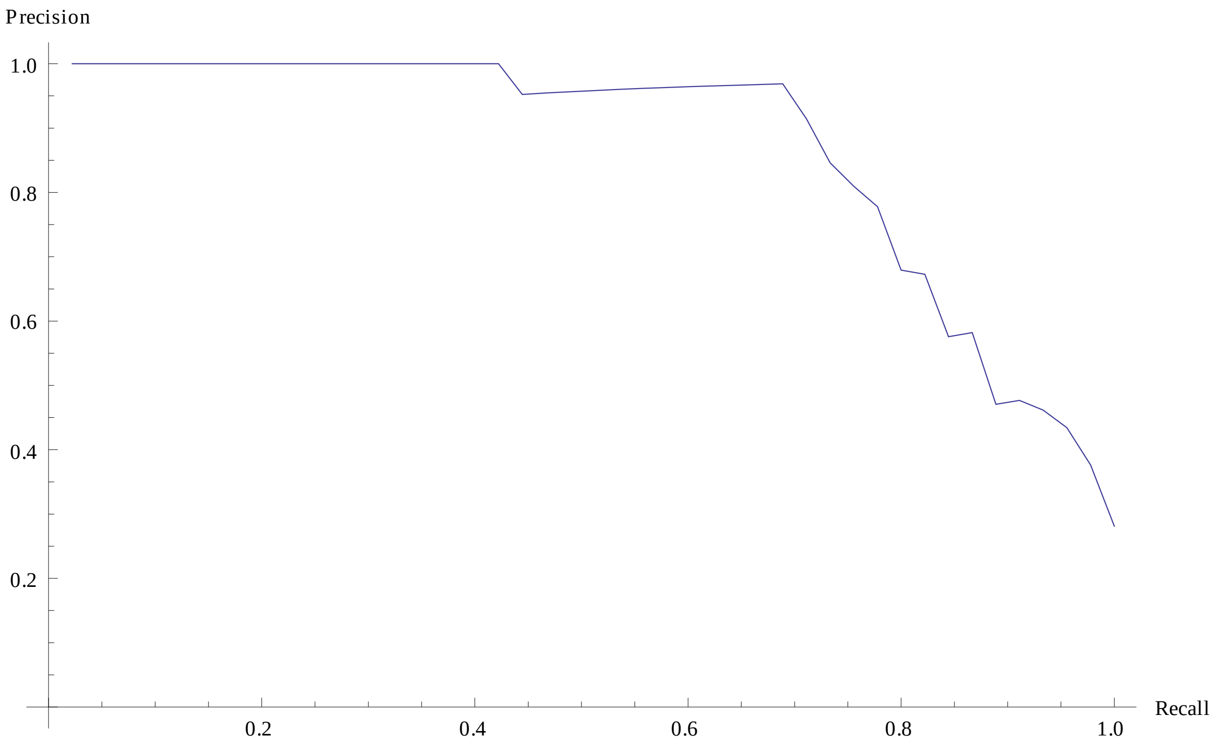


Figure 11 - Precision Recall graph using AUDIAVUS model



5 Conclusion

In this report an implementation of the method “3D Model Search Engine Based on Lightfield Descriptors” was elaborated. The implementation enables retrieval 3D models from a database using a query model. The original paper also includes a 2D drawing user interface for queering the database, which was omitted from this implementation. Much of the performance of the whole system heavily depends of the choice of the weighting factors, which can only be adjusted once the system is complete, therefore more time to run experiments and measure the performance will might further improve the system.

Bibliography

Yu-Te Shen, D.-Y. C.-P. (2003). 3D Model Search Engine Based on Lightfield Descriptors.

Weizhuo Wanga, J. E. (2008). Mode-shape recognition and finite element model updating using the Zernike moment descriptor.

Dengsheng Zhang and Guojun Lu (2002), An Integrated Approach to Shape Based Image Retrieval.

Aforge, http://code.google.com/p/aforge/

Mingwu Ren, Jingyu Yang and Han Sun (2002), Tracing boundary contours in a binary image.