u

EL-GY 9143
3D Computer Vision: Techniques and Applications

NAMAN PATEL

N17075820

nkp269@nyu.edu

Project 2:
Shape Search Engine

Due 21st October, 2015

[17 Pages]

Objective:

To design a shape search engine for the TOSCA dataset to retrieve relevant 3D shapes for given query in MATLAB.

Introduction:

3D models represent a physical body using a collection of points in 3D space, connected by various geometric entities such as triangles, lines, curved surfaces, etc. Being a collection of data (points and other information), 3D models can be created by hand, algorithmically (procedural modeling), or scanned.

Shape analysis is the mainly automatic analysis of geometric shapes, for example using a computer to detect similarly shaped objects in a database or parts that fit together. For a computer to automatically analyze and process geometric shapes, the objects have to be represented in a digital form.

Once the objects are given, either by modeling (computer-aided design), by scanning (3D scanner) or by extracting shape from 2D or 3D images, they have to be simplified before a comparison can be achieved. The simplified representation is often called a shape descriptor. These simplified representations try to carry most of the important information, while being easier to handle, to store and to compare than the shapes directly.
A complete shape descriptor is a representation that can be used to completely reconstruct the original object.

3D Shape Descriptors can be classified into distributed Shape descriptor and Spectral Shape Descriptor. Examples of some advanced Distributed shape descriptors are Euclidean Shape Descriptor, Geodesic Shape Descriptor, Diffusion shape descriptor and Dictionary coded shape descriptors. Examples of Spectral Shape Descriptors are Fourier series based Shape Descriptor and Laplace-Beltrami operator based Spectral Shape Descriptor.

Our objective is to design a 3D Shape search engine which will compare the query image with the whole dataset and output images similar to the query images.

These are the steps involved in designing the search engine:

- Load the dataset
- Calculate and save the descriptors for all the shapes in the dataset
- Design a GUI for getting the Query shape
- Compare the Query shape with all the shapes in the dataset using an appropriate similarity metric
- Retrieve and display the most similar shapes in the GUI

There are additional tasks to be performed for testing the reliability of the descriptor which are as follows:

- Calculate the mean average precision for a given descriptor against the entire dataset
- Calculate the precision and recall for all the shapes in the dataset and plot the average curve of all individual Precision recall curve

- Generate 3 noise shapes with different noise levels for each given 3D shape in the dataset
- Test the search engine with the new dataset which include clean as well as noisy shapes.
- Also calculate the mean average precision and the Average precision recall curve for the new dataset.
- Generate 3 incomplete 3D shapes with different level of incompleteness for each given clean 3D shape in the dataset.
- Test the search engine with the new dataset which include clean as well as incomplete shapes.
- Also calculate the mean average precision and the Average precision recall curve for the new dataset.

We'll be performing the above objectives for Euclidean Shape descriptor and Geodesic Shape Descriptor.

Description of Files:

Euclidean_Distance.m: Function for calculating Euclidean distance between each point of two matrices

Euclidian_Descriptor.m: Function for Euclidean Distance based Shape Descriptor

Geodesic_Descriptor.m: Function for Geodesic Distance based Shape Descriptor

Shortest.m: Function for calculating shortest distance from a source vertex to all the vertices in a graph

addGaussianNoise.m: Function for Generating Gaussian noise on 3D shapes

Clean_descriptor_generator.m: Program for Generating Euclidean and Geodesic Descriptors for all the shapes in the dataset

Cleannoise_descriptor_generator.m: Program for Generating Euclidean and Geodesic Descriptors for all the shapes in the dataset and shapes with added noise at different level

Cleanincomplete_descriptor_generator.m: Program for Generating Euclidean and Geodesic Descriptors for all the shapes in the dataset and incomplete shapes at different level.

PRCurve_MAP_Clean_Generator.m: Program for Generating PR Curve and Calculating Mean Average Precision for the clean dataset using Euclidean and Geodesic Descriptor.

PRCurve_MAP_Cleannoise_Generator: Program for Generating PR Curve and Calculating Mean Average Precision for the clean+noise dataset using Euclidean and Geodesic Descriptor.

PRCurve_MAP_Cleanincomplete_Generator.m: Program for Generating PR Curve and Calculating Mean Average Precision for the clean+incomplete dataset using ED and GD

Assignment_GUI.m: Function for shape search from clean, clean+noise and clean+incomplete dataset using Euclidean and Geodesic Descriptors.

Assignment_GUI.fig: File containing the GUI figure

EDMAT_clean.mat: Euclidean Descriptors for the clean dataset

EDMAT_cleannoise.mat: Euclidean Descriptors for the clean+noise dataset

GDMAT_clean.mat: Geodesic Descriptors for the clean dataset

GDMAT_cleannoise.mat: Geodesic Descriptors for the clean+noise dataset

EDMAT_cleanincomplete.mat: Euclidean Descriptors for the clean+incomplete dataset

GDMAT_cleanincomplete.mat: Geodesic Descriptors for the clean+incomplete dataset

Other than these files, we have files for generating labels and filenames for making all the programs faster.

Euclidean_Distance.m:

Function Arguments:

Input: Matrix of points between which the Euclidean Distance has to be calculated.

Output: Euclidean Distance

Function Implementation:

- Initialize the Euclidean distance matrix to zero with size rows of 1st coordinate matrix x size of 2nd coordinate matrix
- Take the sum of square of difference of distances between each coordinate and store it in their corresponding position

Euclidian_Descriptor.m:

Function Arguments:

Input: Path of the 3D shape .off file whose descriptor is to be evaluated.

Output: Normalized Euclidean Descriptor

Function Implementation:

- Open the 3D shape file
- Read the first line and remove all the new line characters
- Get the number of vertices and triangles from the second line of the .off file
- Read and store all the vertices given in the file
- Calculate the Euclidean distance of all the vertices with respect to each vertex and store it in a matrix
- Generate a histogram based on the frequency of similar Euclidean distances
- Normalize the histogram

Shortest.m:

Function Arguments:

Input: Matrix representing the distances between neighboring vertices

Source: Source vertex from which the distance is to be calculated

Output: Distances of each vertex from the source vertex.

Function Implementation:

- Initialize all the distances from source vertex to other vertices as infinity

- Distance to source vertex is 0
- Initialize a vector containing the visited nodes and add the source node to that vector
- While all the nodes are not visited do the following:
    - For all nodes not visited, update the distance from source to the minimum of it present value and the sum of distance from source to its nearest neighbor and distance to its nearest neighbour.
    - Next find the distance which is infinity and make the smallest indice the new source
    - Update the visited node matrix to the node whose distance is updated
- Run the loop until there is no change in distances and all distances are less than infinity.

Geodesic_Descriptor.m:

Function Arguments:

Input: Path of the 3D shape .off file whose descriptor is to be evaluated.

Option: Method to use to evaluate. 1 for Dijkstra 2 for inbuilt function.

Output: Normalized Geodesic Descriptor

Function Implementation:

- Open the 3D shape file
- Read the first line and remove all the new line characters
- Get the number of vertices and triangles from the second line of the .off file
- Read and store all the triangles given in the file. The first number in each line is the number of vertices connected. The other three are the points which constitute the triangle.
- Store the vertices of each triangle
- Calculate the Euclidean distance between each vertex of the triangle and store it in the matrix G where the element G_{ij} is the Euclidean distance between points with index i and j.
- The matrix G represents a graph. Next, calculate the shortest path in the Graph which represents the Geodesic distance using option 1 or option 2.
- Option 1 runs the shortest function for all the vertices and option 2 uses the graphallshortestpath function.
- Generate a histogram based on the frequency of similar Geodesic distances
- Normalize the histogram

addGaussianNoise.m

Function Arguments:

Input: Path of the 3D shape .off file whose noise shape is to be evaluated.

Path where the 3D shape with noise is to be saved

Gaussian Noise to be applied to the shape

Clean_descriptor_generator.m

- Initialize the path to the path of the dataset

- Initialize a cell array with name of the shapes and an array with the number of shapes
- Evaluate the Geodesic and Euclidean descriptor for all of the shapes and store it in matrices **GDMAT_clean.mat** and **EDMAT_clean.mat** respectively.

Cleannoise_descriptor_generator.m

- Initialize the path to the path of the dataset
- Initialize a cell array with name of the shapes and an array with the number of shapes
- Evaluate the Geodesic and Euclidean descriptor for all of the shapes and store it in matrices **GDMAT_cleannoise.mat** and **EDMAT_cleannoise.mat** respectively
- Generate noises at different levels (low, medium and high) and evaluate the Geodesic and Euclidean descriptor and store it in **GDMAT_cleannoise.mat** and **EDMAT_cleannoise.mat** respectively

Cleanincomplete_descriptor_generator.m

- Initialize the path to the path of the dataset
- Initialize a cell array with name of the shapes and an array with the number of shapes
- Evaluate the Geodesic and Euclidean descriptor for all of the shapes and store it in matrices **GDMAT_cleanincomplete.mat** and **EDMAT_cleanincomplete.mat** respectively
- Generate incomplete shapes at different levels (low, medium and high using Meshlab) and evaluate the Geodesic and Euclidean descriptor and store it in **GDMAT_cleanincomplete.mat** and **EDMAT_cleanincomplete.mat** respectively

PRCurve_MAP_Clean_Generator.m

- Load **EDMAT_clean.mat** and **shapeindex.mat**
- Calculate the Euclidean distance between each Euclidean descriptors of all the shapes.
- Sort by minimum Euclidean distances and store its indices.
- Retrieve the shapes from the indices and shapeindex and store it in a retrieval matrix
- Now compare the 1st retrieval (Original Image) for each shape with all the other retrievals. If there is a match then increment the variable k (AUB) and also add k/number of queries to average precision variable which is initially 0.
- Obtain the precision by dividing k obtained by number of queries
- Obtain the recall by dividing k by the total number of shapes which is queried in the dataset.
- Do this for all the shapes
- Take the mean of average precision variable to get the mean average precision
- Take the mean of the precision variable to obtain an average precision for all the shapes
- Take the mean of the recall to obtain an average recall for all the shapes
- Plot the average precision-recall curve

- Hold on the obtained plot
- Load **GDMAT_clean.mat** and **shapeindex.mat**
- Calculate the Euclidean distance between each Geodesic descriptors of all the shapes.
- Sort by minimum Geodesic distances and store its indices.
- Retrieve the shapes from the indices and shapeindex and store it in a retrieval matrix
- Now compare the 1st retrieval (Original Image) for each shape with all the other retrievals. If there is a match then increment the variable k (AUB) and also add k/number of queries to average precision variable which is initially 0.
- Obtain the precision by dividing k obtained by number of queries
- Obtain the recall by dividing k by the total number of shapes which is queried in the dataset.
- Do this for all the shapes
- Take the mean of average precision variable to get the mean average precision
- Take the mean of the precision variable to obtain an average precision for all the shapes
- Take the mean of the recall to obtain an average recall for all the shapes
- Plot the average precision-recall curve
- Compare both the curves obtained

PRCurve_MAP_Cleannoise_Generator.m

- Load **EDMAT_cleannoise.mat** and **shapeindexnoise.mat**
- Calculate the Euclidean distance between each Euclidean descriptors of all the shapes.
- Sort by minimum Euclidean distances and store its indices.
- Retrieve the shapes from the indices and shapeindex and store it in a retrieval matrix
- Now compare the 1st retrieval (Original Image) for each shape with all the other retrievals. If there is a match then increment the variable k (AUB) and also add k/number of queries to average precision variable which is initially 0.
- Obtain the precision by dividing k obtained by number of queries
- Obtain the recall by dividing k by the total number of shapes which is queried in the dataset.
- Do this for all the shapes
- Take the mean of average precision variable to get the mean average precision
- Take the mean of the precision variable to obtain an average precision for all the shapes
- Take the mean of the recall to obtain an average recall for all the shapes
- Plot the average precision-recall curve
- Hold on the obtained plot
- Load **GDMAT_cleannoise.mat** and **shapeindexnoise.mat**
- Calculate the Euclidean distance between each Geodesic descriptors of all the shapes.
- Sort by minimum Geodesic distances and store its indices.

- Retrieve the shapes from the indices and shapeindex and store it in a retrieval matrix
- Now compare the 1st retrieval (Original Image) for each shape with all the other retrievals. If there is a match then increment the variable k (AUB) and also add k/number of queries to average precision variable which is initially 0.
- Obtain the precision by dividing k obtained by number of queries
- Obtain the recall by dividing k by the total number of shapes which is queried in the dataset.
- Do this for all the shapes
- Take the mean of average precision variable to get the mean average precision
- Take the mean of the precision variable to obtain an average precision for all the shapes
- Take the mean of the recall to obtain an average recall for all the shapes
- Plot the average precision-recall curve
- Compare both the curves obtained

PRCurve_MAP_Cleanincomplete_Generator.m

- Load **EDMAT_cleanincomplete.mat** and **shapeindexnoise.mat**
- Calculate the Euclidean distance between each Euclidean descriptors of all the shapes.
- Sort by minimum Euclidean distances and store its indices.
- Retrieve the shapes from the indices and shapeindex and store it in a retrieval matrix
- Now compare the 1st retrieval (Original Image) for each shape with all the other retrievals. If there is a match then increment the variable k (AUB) and also add k/number of queries to average precision variable which is initially 0.
- Obtain the precision by dividing k obtained by number of queries
- Obtain the recall by dividing k by the total number of shapes which is queried in the dataset.
- Do this for all the shapes
- Take the mean of average precision variable to get the mean average precision
- Take the mean of the precision variable to obtain an average precision for all the shapes
- Take the mean of the recall to obtain an average recall for all the shapes
- Plot the average precision-recall curve
- Hold on the obtained plot
- Load **GDMAT_cleanincomplete.mat** and **shapeindexnoise.mat**
- Calculate the Euclidean distance between each Geodesic descriptors of all the shapes.
- Sort by minimum Geodesic distances and store its indices.
- Retrieve the shapes from the indices and shapeindex and store it in a retrieval matrix
- Now compare the 1st retrieval (Original Image) for each shape with all the other retrievals. If there is a match then increment the variable k (AUB) and also add k/number of queries to average precision variable which is initially 0.

- Obtain the precision by dividing k obtained by number of queries
- Obtain the recall by dividing k by the total number of shapes which is queried in the dataset.
- Do this for all the shapes
- Take the mean of average precision variable to get the mean average precision
- Take the mean of the precision variable to obtain an average precision for all the shapes
- Take the mean of the recall to obtain an average recall for all the shapes
- Plot the average precision-recall curve
- Compare both the curves obtained

AssignmentGUI.m

- Build the shape searching GUI containing 6 plots which includes 1 plot for the original image and the other 5 for retrieved image, a browse button and text box for loading the query image, a pop up box for selecting the descriptor to use, a pop up box for the dataset to use and the retrieve button for searching similar images.
- Get the descriptor that was chosen from the call back function of the descriptor pop up menu
- Get the dataset that was chosen from the call back function of the Dataset pop up menu
- Get the input shape path from the call back function of the browse button
- Load the appropriate descriptor dataset and file index according to the descriptor and dataset chosen.
- Load the query image and evaluate the descriptor selected by the user for the shape
- Calculate the Euclidean distance between descriptors of all the shapes and the query shape
- Sort the distances according to the minimum Euclidean distances and get their indices.
- Retrieve the shapes from the indices using the shapefile index file
- Get the coordinates of the first 6 shapes retrieved and plot it on each of the plots.
- The first shape is the original image and the other 5 are the retrieved images.

Running the Program:

For Precision-Recall Curve of the Clean Dataset:

- Open **PRCurve_MAP_Clean_Generator.m**
- Make sure **EDMAT_clean.mat**, **GDMAT_clean.mat** and **shapeindex.mat** are in your working directory.
- Run the program

For Precision-Recall Curve of the Clean and Noise Dataset:

- Open **PRCurve_MAP_Cleannoise_Generator.m**
- Make sure **EDMAT_cleannoise.mat, GDMAT_cleannoise.mat** and **shapeindexnoise.mat** are in your working directory.

- Run the program

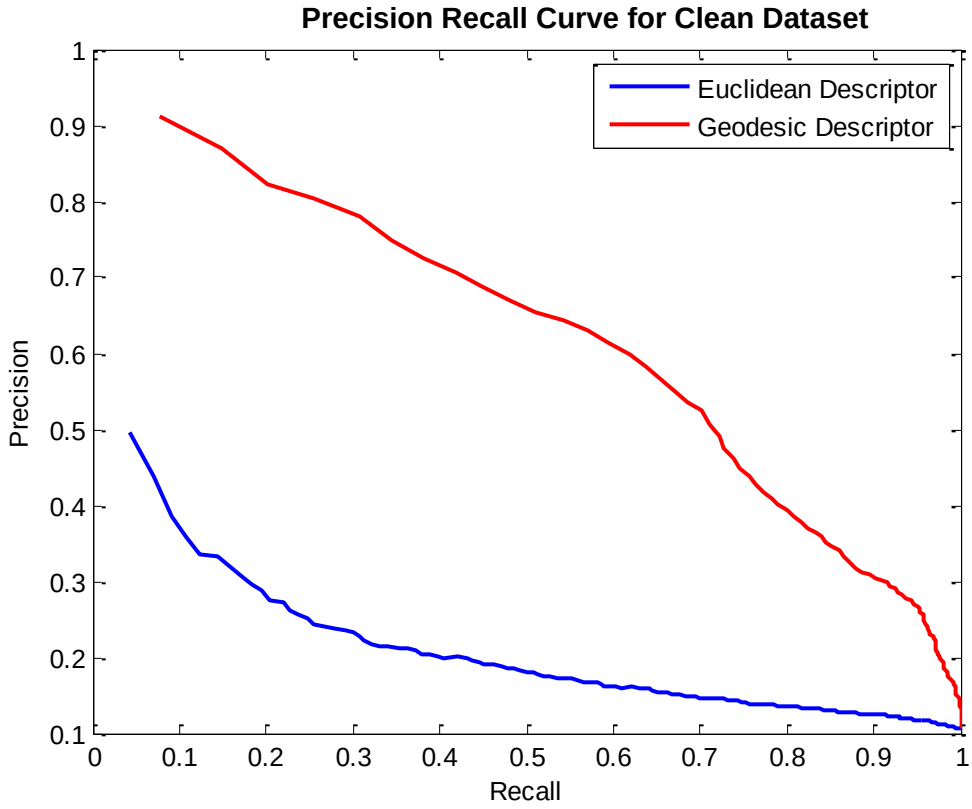
For Precision-Recall Curve of the Clean and Incomplete Dataset:

- Open **PRCurve_MAP_Cleanincomplete_Generator.m**
- Make sure **EDMAT_cleanincomplete.mat, GDMAT_cleanincomplete.mat** and **shapeindexnoise.mat** are in your working directory.
- Run the program

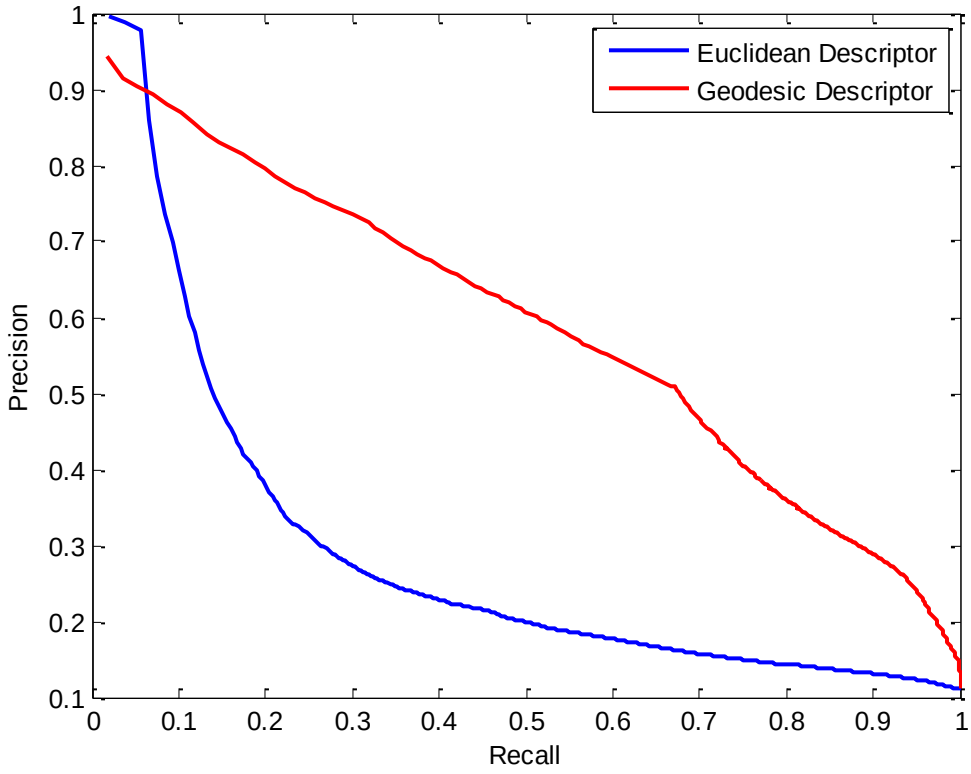
For Running the Shape Search Engine:

- In **AssignmentGUI.m** change the path on line 148 to your path of the dataset.
- Make sure all the datasets and index labels are in the working directory
- Run the program
- Select your choice of the descriptor
- Select your choice of the dataset
- Press browse and select the query image and then press Retrieve.

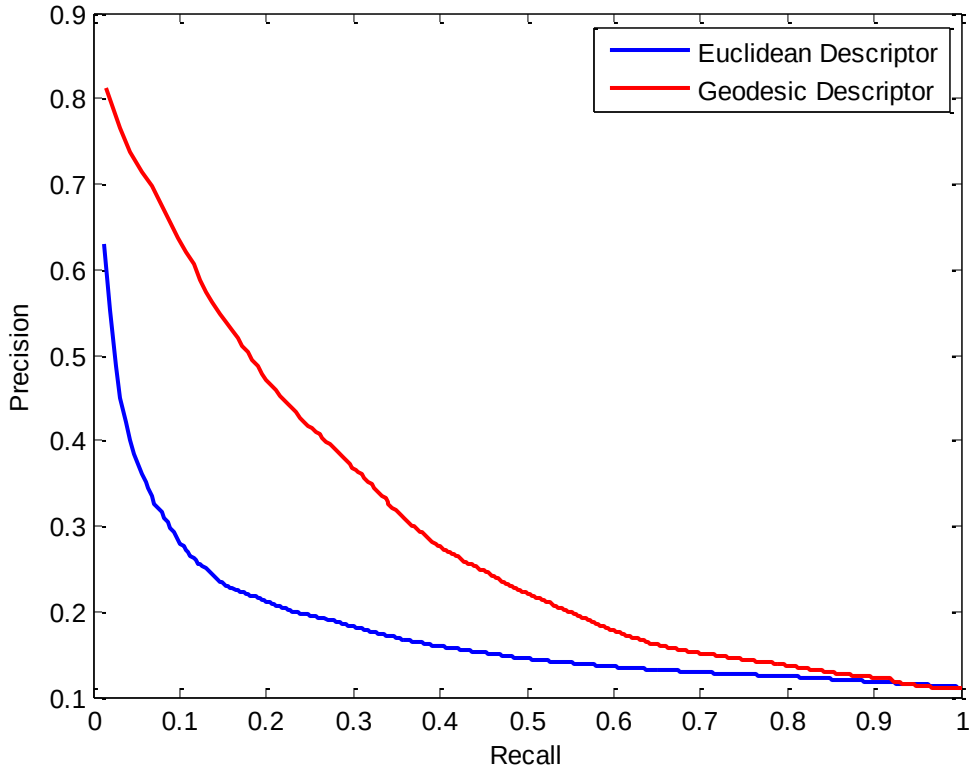
Experimental Results:



Precision Recall Curve for Clean+Noise Dataset



Precision Recall Curve for Clean+Incomplete Dataset



Mean Average Precision:

Mean Average Precision for clean dataset with Euclidean Descriptor: 0.291736

Mean Average Precision for clean dataset with Geodesic Descriptor: 0.70937

Mean Average Precision for clean and noise dataset with Euclidean Descriptor: 0.334681

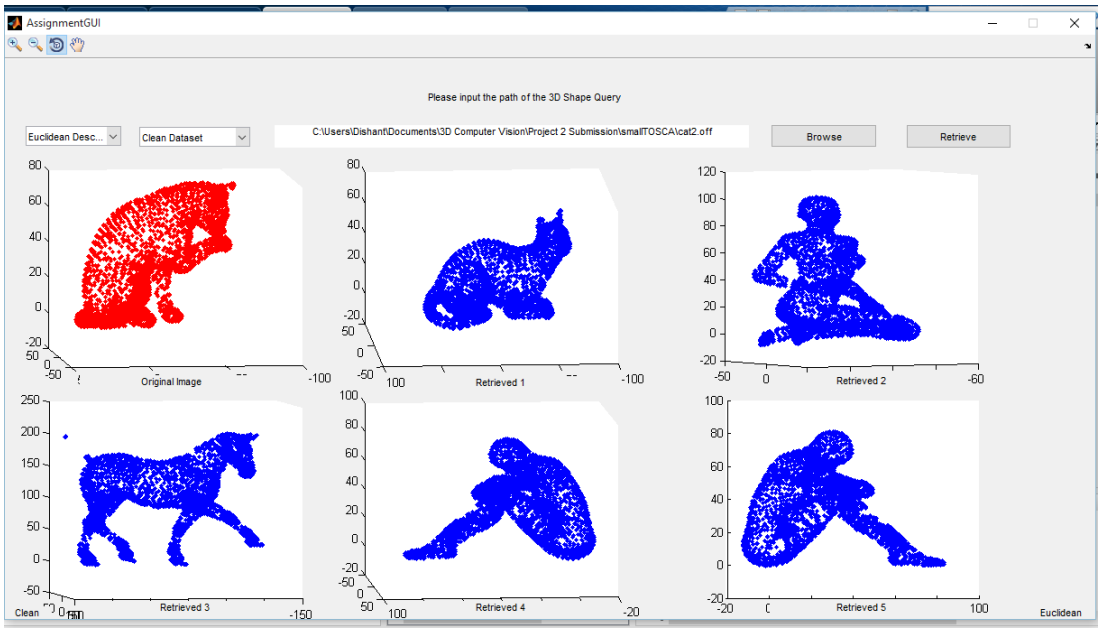
Mean Average Precision for clean and incomplete dataset with Geodesic Descriptor: 0.663546

Mean Average Precision for clean and incomplete dataset with Euclidean Descriptor: 0.213

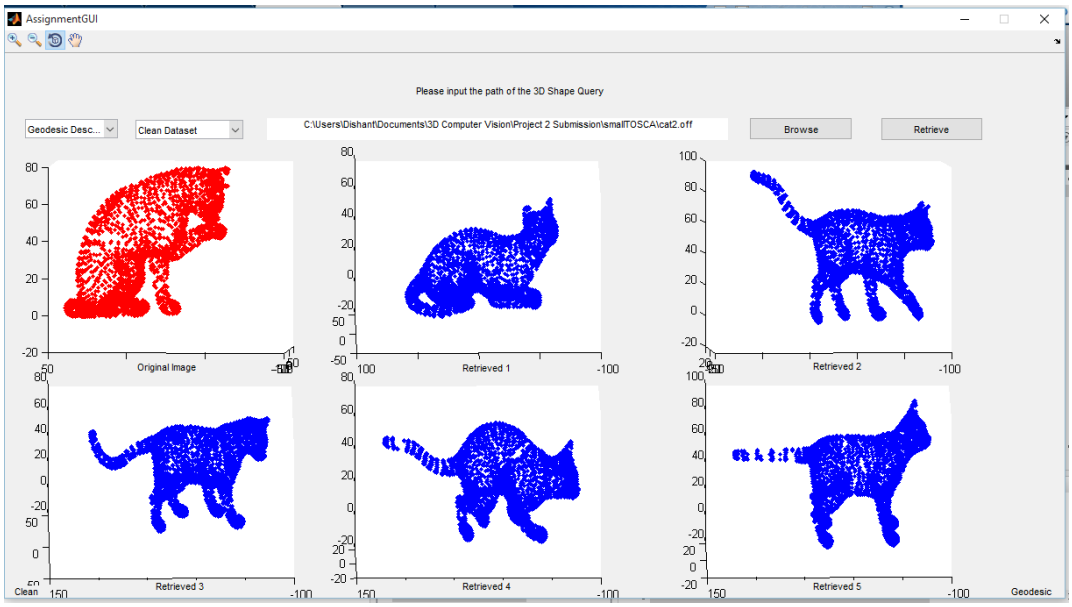
Mean Average Precision for clean and incomplete dataset with Geodesic Descriptor: 0.3598

GUI for Shape Search Engine:

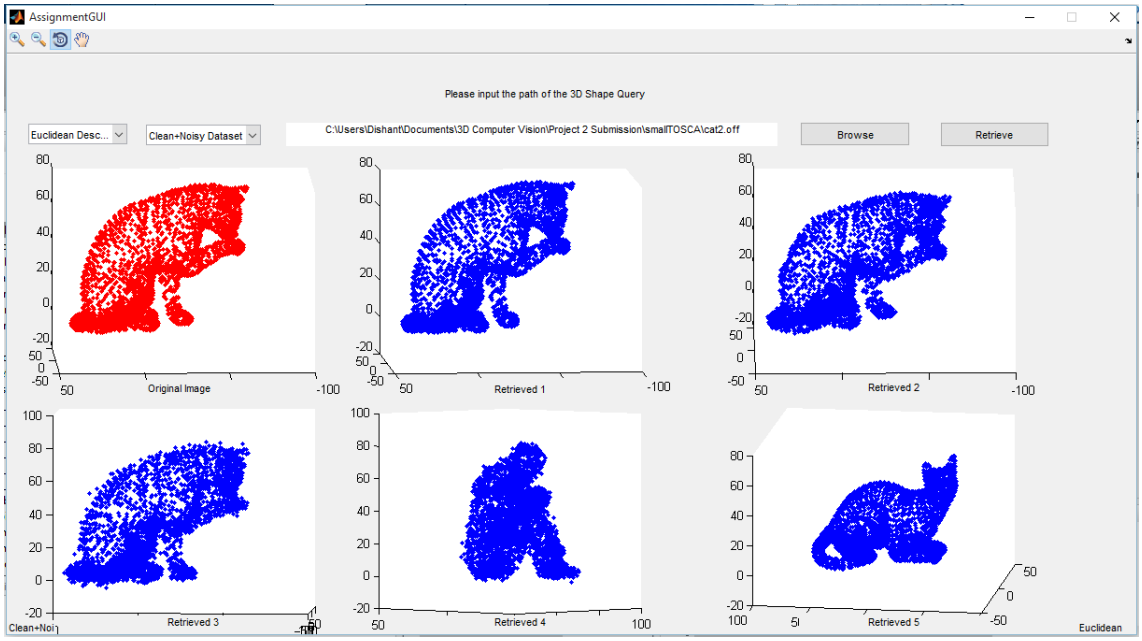
Euclidean descriptor for Clean Dataset:



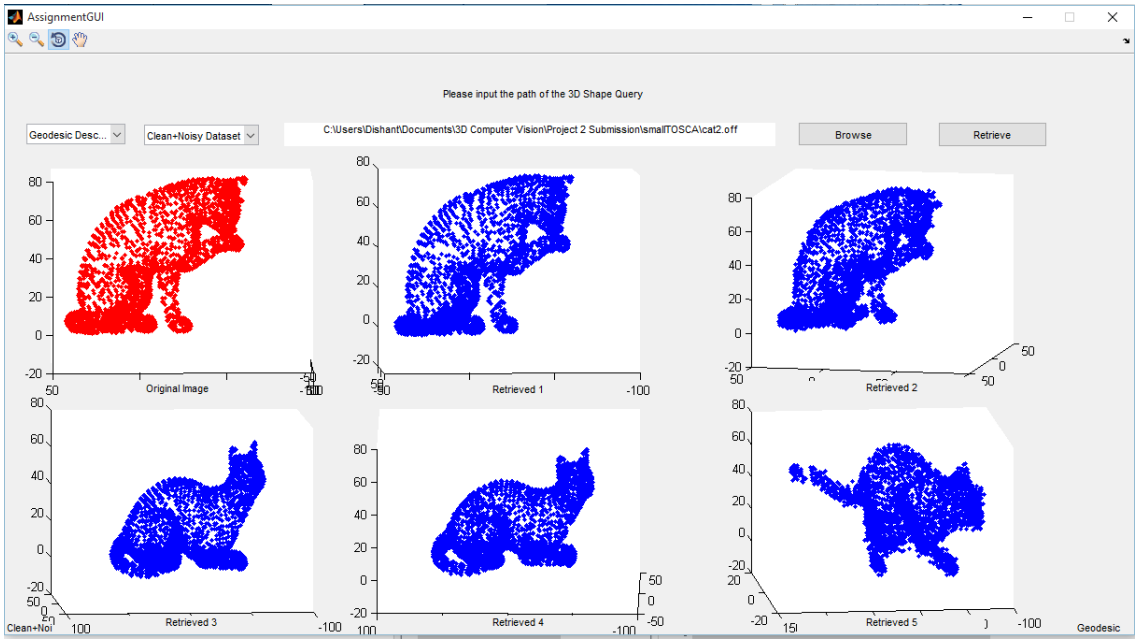
Geodesic descriptor for Clean Dataset:



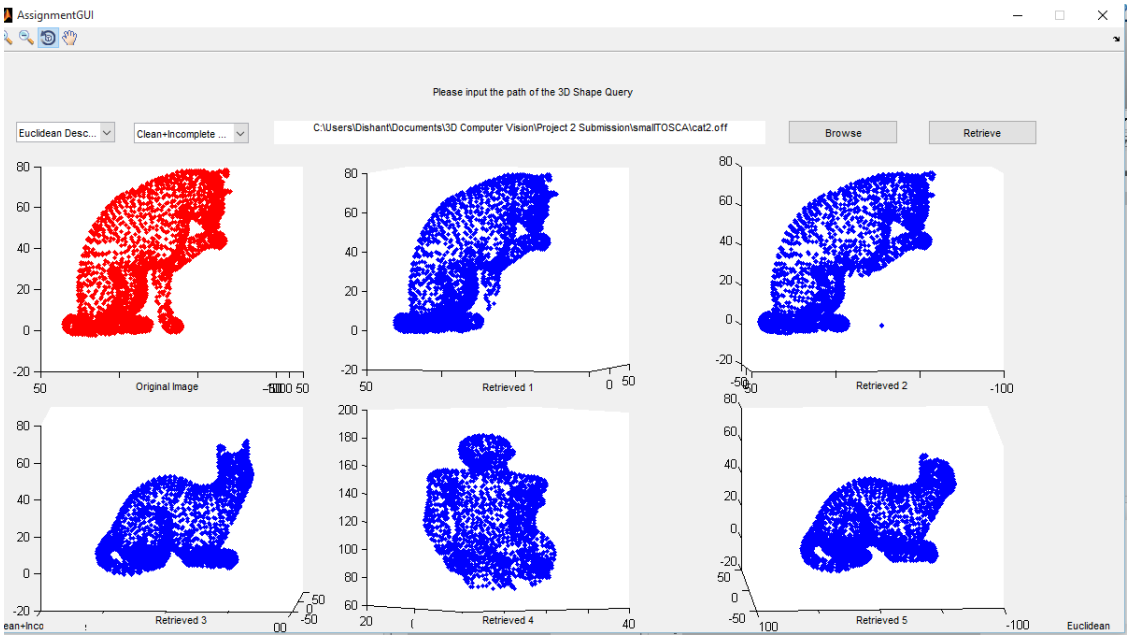
Euclidean Descriptor for Clean+Noisy Dataset:



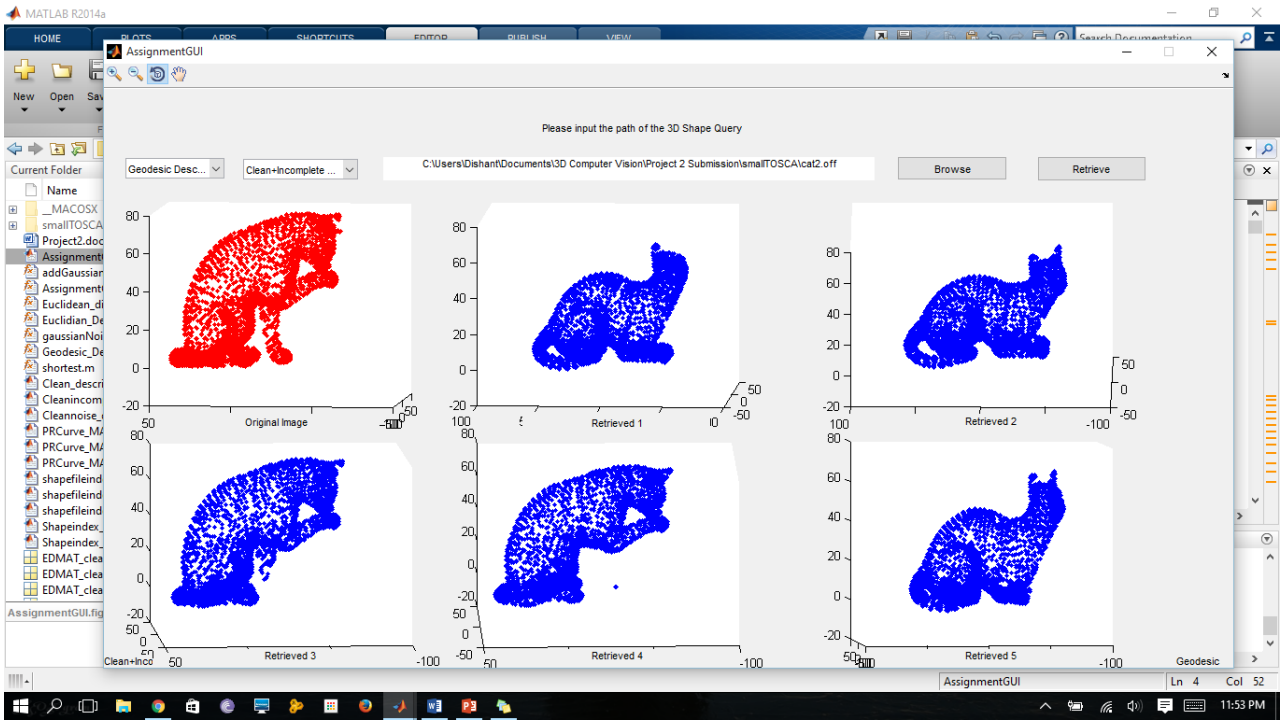
Geodesic Descriptor for Clean+Noisy Dataset:



Euclidean Descriptor for Clean+Incomplete Dataset:



Geodesic Descriptor for Clean+Incomplete Dataset:



New Descriptor Algorithm:

Angle Descriptor

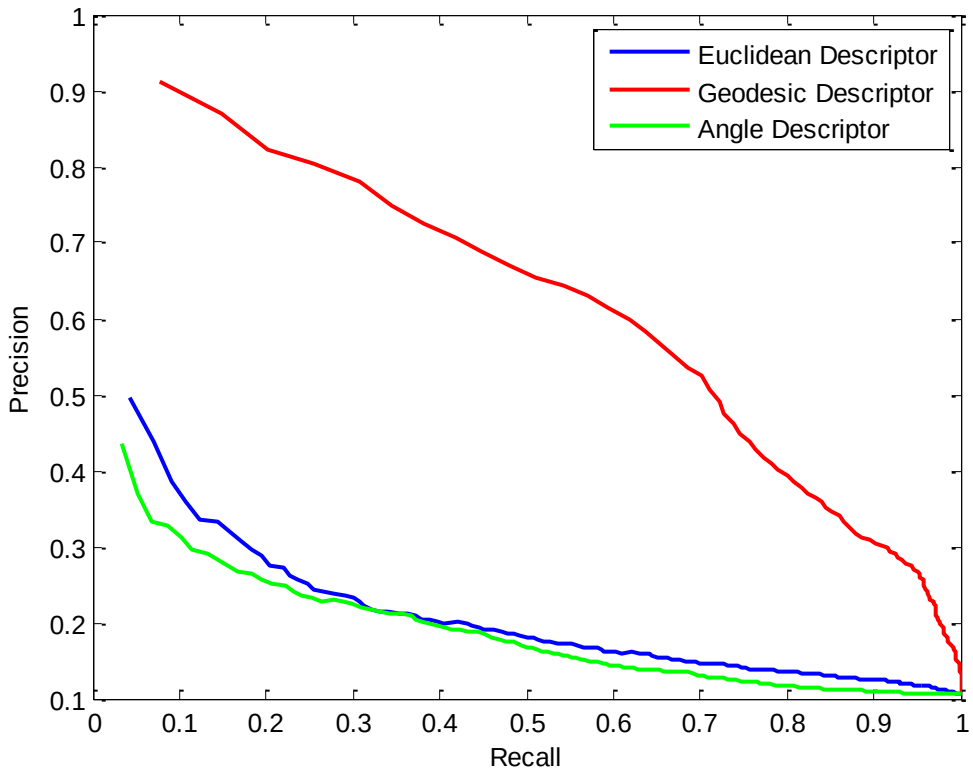
- Calculate the centroid of a shape
- Get the angle between the centroid and each vertex in the shape
- Get the histogram of the angles of each vertex with respect to the centroid
- Normalize the histogram to get the angle descriptor

For Search engine

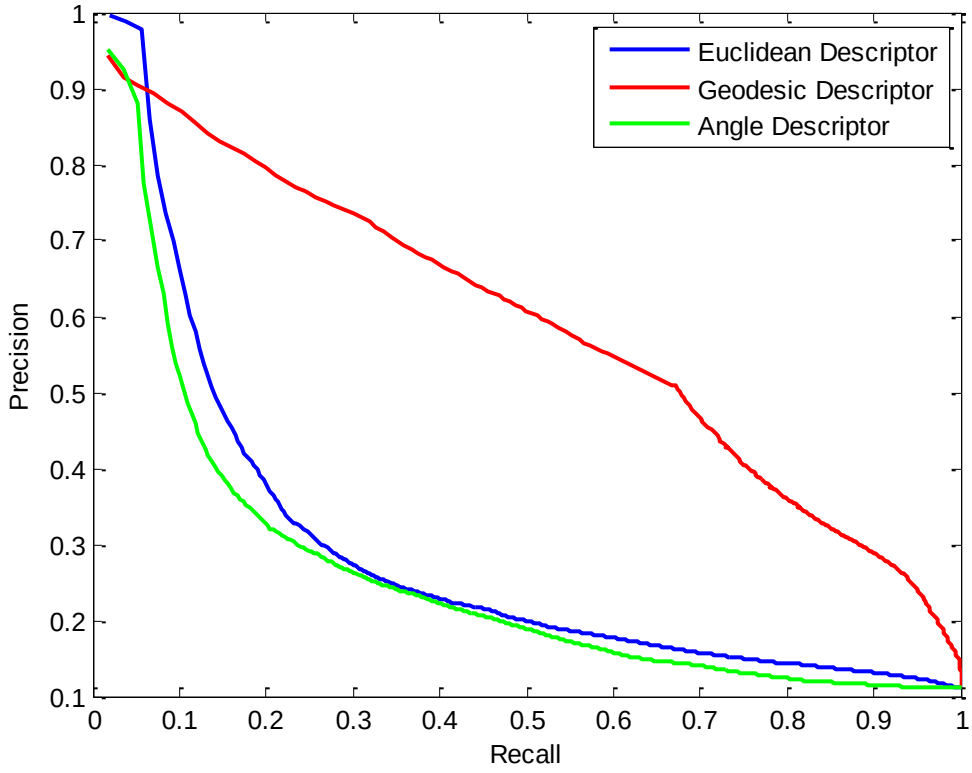
- Get the descriptor for all the vertices
- Get the Euclidean distance between the query vertex and all the vertices
- The shapes with minimum Euclidean distance will be the one which are most similar.

Results:

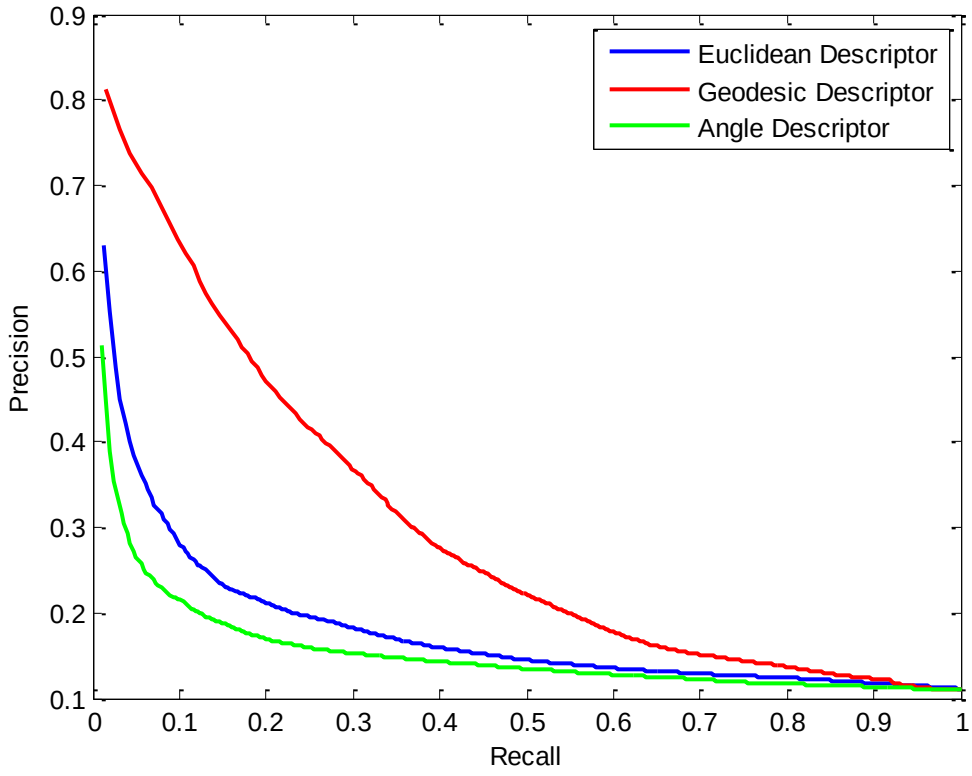
Precision Recall curve for clean dataset



Precision Recall curve for clean+noise dataset



Precision Recall curve for clean+incomplete dataset



Minimum average Precision

Mean Average Precision for clean dataset with Angle descriptor: 0.267833

Mean Average Precision for clean and noise dataset with Angle descriptor: 0.307391

Mean Average Precision for clean and incomplete dataset with Angle descriptor: 0.184832

Conclusion:

The proposed angle descriptor performs worse than the Euclidean as well as Geodesic Descriptor because it does not have any characteristic that will minimize the error for isometric transformation.

Eigen-Geodesic Descriptor

- Calculate Geodesic distance between each point of a shape and store it in a matrix
- Perform the above step for all the shapes and store it in a matrix X of dimension (number of shapes*(number of points)^2
- Get the mean shape Geodesic matrix from all the shapes Geodesic matrix
- Subtract mean shape from all the shapes Geodesic matrix
- Get the covariance matrix S of the new Geodesic matrix
- Get the eigenvalues and eigenvectors of matrix S
- Order the eigenvectors according to the maximum eigenvalues and store it in matrix W arranged by new sorted indices
- Multiply this matrix W with the mean shape Geodesic matrix to get eigen shape descriptors equal to the number of shapes
- Now pass each of these eigen shape descriptor through a denoised autoencoder and train this autoencoder until it is able to reconstruct the Eigen shape
- After the network is fully trained, the middle layer will be the descriptor layer which is able to reconstruct the original image

For Search Engine:

- Pass every shape through this denoised autoencoder network and store the descriptor obtained
- Now pass the query image through the autoencoder to get its descriptor
- Calculate the Euclidean distance between the descriptor obtained and descriptor obtained of all the other shapes
- The shapes having the minimum Euclidean distance are the shapes most similar to the query image

Due to time constrain, I was not able to test my descriptor. Hope to implement it shortly.