NISTIR 7625

Three Dimensional Shape Retrieval using Scale Invariant Feature Transform and Spatial Restrictions

Lydia Lei

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 7625

Three Dimensional Shape Retrieval using Scale Invariant Feature Transform and Spatial Restrictions

Lydia Lei

National Institute of Standards and Technology
Gaithersburg, MD

August 2009

U.S. DEPARTMENT OF COMMERCE
Gary Locke, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLGY
Dr. Patrick D. Gallagher, Deputy Director

TABLE OF CONTENTS

DISCLAIMER

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST nor does it imply that the products mentioned are necessarily the best available for the purpose.

ABSTRACT

Three dimensional shape retrieval is a fairly new concept being studied all over the world. It is notable and essential because it can be applied to multiple disciplines, including: computer vision, CAD models, computer graphics, molecular biology, etc. For this project, 907 3D models from the Princeton Shape Benchmark (PSB) were rendered as depth images from 20 views. The models are categorized in 92 different classes, ranging from humans to houses. David Lowe's Scale Invariant Feature Transform (SIFT) algorithm was used in this project to normalize the images, find 'key points' on each of the views, and create a specific feature vector to describe its respective key point.

A comparison of these 907 objects was done by finding similarity between key points and their feature vectors on each of the objects' corresponding views. After using the SIFT algorithm, the results were further filtered using Euclidean distance differences and spatial restrictions. By adding the spatial restrictions, it prevented the code from matching a hand to a foot; this is because the x and y coordinates of the key points would not be in the same general 'spatial area'. Lastly, the overall similarity of two objects is calculated. The different objects were then ordered based on similarity and stored in a 'distance matrix'. The accuracy of the code and the results were evaluated by comparing the retrieval results to the results yielded from twelve other shape descriptors (not SIFT) using the PSB base classification.

ACKNOWLEDGEMENTS

The funding and support for this report was made possible by the Systems Integration for Manufacturing Applications (SIMA) and the Summer Undergraduate Research Fellowship (SURF) at the National Institute of Standards and Technology (NIST). My advisor, Afzal Godil, gave me his guidance and insight throughout the entire process of this research project, and I want to thank him for his consistent assistance and support. I would also like to thank my office mate, Helin Dutagaci, for being so patient and accommodating when helping me understand and adjust to the new project.

1. INTRODUCTION

1.1 Need for shape searching

In the area of shape retrieval, the computer is expected to read in a query object, compare the query object to the computer's database of objects, and retrieve the similar objects in decreasing order of similarity. In order for the retrieval system to work correctly, the internal programming of the shape retrieval algorithm must be accurate and efficient. The study of three dimensional shape retrieval is advancing, day to day, all over the world. Shape based retrieval of 3D data is included in disciplines such as computer vision, CAD images, object recognition, geometric modeling, computer-aided design and engineering, and chemistry. Specifically, the use of 3D models is increasing in manufacturing processes, especially injection molding and casting, where it is critical for the manufacturers to envision and comprehend the part accurately before spending a large amount of money building the tool [1]. In the field of computer vision, engineers are beginning to create products that use the computer vision algorithm to identify objects for the blind and handicapped.

3D shape retrieval is very similar to text retrieval. A well known example of text retrieval is the Google webpage. When a user wants to find a website regarding a certain keyword, after typing in that keyword, Google searches through its database and compares the keyword with the database of online websites, journal articles, image descriptions, etc. However, 3D shapes are not as easily retrieved as text is.

1.2 What is 'shape' and 'similarity'?

Three dimensional shape searching refers to 'determining similarities among shapes from a large database of 3D shapes' [1]. But, the definition of *similar* and of *shape* differs from person to person. Although there is no standard definition for shape, for the purpose of this paper, we will use Kendall's definition of shape: '…all the geometrical information that remains when location, scale, and rotational effects (Euclidean transformations) are filtered out from an object' [2]. To define the word 'similar', we will use the definition found in the Merriam-Webster Dictionary, 'having characteristics in common: strictly comparable' or 'not differing in shape but only in size or position'. In this paper, similarity is measured in terms of a similarity metric, and will be quantified as a metric in database terms. Metrics has not only been used in 3D shape retrieval, but has been used and applied in databases to find similar documents, images, audio, and movies [1].

1.3 Outline of this paper

In this paper, we will summarize which algorithms and databases were used in parallel with this project, as well as overview the process of writing an algorithm that performs efficient 3D shape retrieval using David Lowe's Scale Invariant Feature Transform (SIFT) algorithm.

2. SCALE INVARIANT FEATURE TRANSFORM

2.1 What is SIFT?

Scale Invariant Feature Transform is an algorithm, published by David Lowe[1] in 1999, to detect and describe local features in images. It is one of the most popular algorithms in the description and matching of 2D image features.

2.2 What is a feature vector?

When the original SIFT algorithm is applied to an object, multiple feature vectors (also known as shape descriptors) are created specific to different key points on that object. The goal of the shape descriptor is to uniquely characterize the shape of the object. Before a feature vector is formed, there are a series of steps SIFT performs to calculate the vector. These series of steps give the algorithm its name because it transforms image data into scale-invariant coordinates relative to local features. The following are the major stages of computations used to generate the set of image features [3]:

1. **Scale-space extreme detection:** The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.
2. **Key point localization:** At each candidate location, a detailed model is fit to determine location and scale. Key points are selected based on measures of their stability.
3. **Orientation assignment:** One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that have been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.
4. **Key point descriptor:** The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

A pro to this approach is the ability of SIFT to generate large numbers of features that densely cover the image over the full range of scales and locations. For a typical image of size 500x500 pixels, this scale invariant approach will produce around 2000 stable features (varies with image content and various parameters) [3].

By using the SIFT algorithm, the result will be two matrices; the first matrix stores location of the feature vector (includes x, y, and z coordinates), and the second matrix stores the actual feature vector.

[1] David Lowe is a Canadian computer scientist and a professor in the computer science department at the University of British Columbia. Lowe is a world renowned researcher in computer vision and is the patented author of the scale invariant feature transform algorithm.

2.3 SIFT matching algorithm

For this project, we not only used SIFT to create the various feature vectors for each image, we also used the SIFT matching algorithm. The matching algorithm matches images by individually comparing features from the query image to the features of all the images stored in a database. It finds matching features using the Euclidean distance of their feature vectors. The smaller the distance, the more similar the two images are.

The matching algorithm will yield two matrices: the matches and the distance between the matches. When finding the features on each image, SIFT numbers each feature vector. After using the matching algorithm, the results will look similar to
Figure 1:

Matches	0	4	5	23	12	45	7	1	9
	0	8	4	12	4	5	1	6	21

Scores	0	3434	4223	12342	23288	93458	23223	42358	24350

Fig 1: Example of the matches and scores matrices resulted from using the vl_ubcmatch function

In the 'matches' matrix, the number of columns represents the number of 'matches' the matching algorithm found between feature vectors of two images. The first row stores the feature vector number of the first image, and the second row is the corresponding 'matched' feature vector number from the second image. So, in this example, it shows that feature vector '4' (also known as keypoint number '4') from the first image matched with feature vector '8' from the second image, and their Euclidean distance is 3434. Figure 2 shows an example of this matching. Note that the drawing is not drawn to scale.

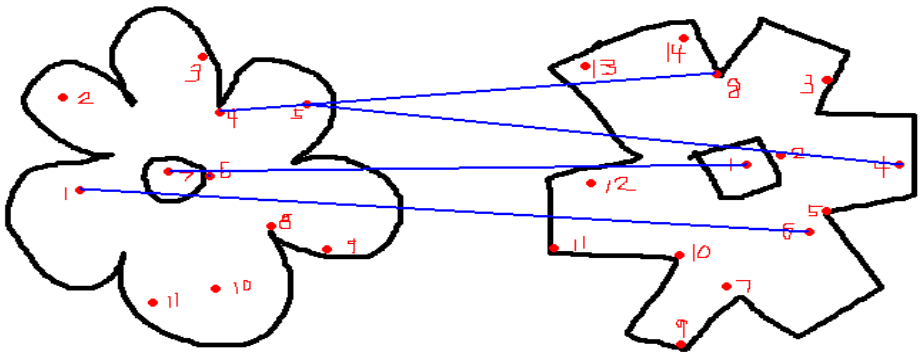


Figure 2: visual image of the matching algorithm and the numbered key points. Drawing not drawn to scale.

3. PRINCETON SHAPE BENCHMARK

3.1 What is the Princeton Shape Benchmark?

The Princeton Shape Benchmark (PSB) is a publically available database of polygonal models collected from the World Wide Web. It is a suite of tools for comparing shape matching and classification algorithms. There have been many algorithms written specifically for 3D shape retrieval. The PSB provides a way to compare and evaluate the algorithms. The main purpose of the PSB is to promote the use of standardized data sets and evaluation methods for research in matching, classification, clustering, and recognition of 3D models.

Before the PSB was created, there were no standard benchmarks available for matching 3D polygonal models representing a wide variety of objects. Recently, many new benchmarks have been made available in addition to the PSB benchmark created by faculty and students of Princeton University. Benchmarks from the University of Konstanz, Utrecht University, and Purdue University have also been open to the public for use [4]. The PSB database contains 1,814 models downloaded from the web, subdivided into a training set and a test set. Each set has 907 models each, each evenly classified into 90 and 92 classes respectively. The creators of the benchmark collected over 20,000 models from more than 2,000 websites, went through each of the 20,000 models individually, and hand picked the 1,814 models that are included in the final benchmark.

The Princeton Shape Benchmark not only provides the test set of models for standardizing the evaluation of 3D shape retrieval algorithms, they also provide several evaluation tools. The evaluation tools are as followed [4]:

- **Best Matches:** a web page for each model displaying images of its best matches in rank order. The associated rank and distance value appears below each image, and the models in the query model's class are highlighted with a thickened frame.
- **Precision-recall plot:** a plot describing the relationship between precision and recall in a ranked list of matches. For each query model in class C and any number K of top matches, "recall" (the x axis) represents the ratio of models in class C returned within the top K matches, while "precision" (the y axis) indicates the ratio of the top K matches that are members of class C. A perfect retrieval results in a straight horizontal line at the top of the graph, precision = 1.0.
- **Distance image:** an image of the distance matrix where the lightness of each pixel (*i* , *j*) is proportional to the magnitude of the distance between models *i* and *j*. The smaller the magnitude, the more similar the two objects are.
- **Tier image:** an image visualizing nearest neighbor, first tier, and second tier matches. This image is often more useful than the distance image because the best matches are clearly shown for every model, regardless of the magnitude of their distance values.

- **Nearest Neighbor (NN):** the percentage of the closest matches that belong to the same class as the query. An ideal score would be 100 %, and higher scores represent better results.
- **First-tier and Second-tier (FT and ST):** the percentage of models in the query's class that appear within the top K matches, where K depends on the size of the query's class. For a class with $|C|$ members, $K = |C| - 1$ for the first tier, and $K = 2 * |C| - 1$ for the second tier. These statistics are similar to the "Bulls Eye Percentage Score". An ideal matching result gives score of 100 %, and higher values represent better results.
- **E-measure (E-M):** a composite measure o the precision and recall for a fixed number of retrieved results. In general, a user of a search engine is more interested in the first page of query results than in later pages. This measure considers only the first 32 retrieved models for every query and calculates the precision and recall over those results. The maximum score is 1.0, higher values indicate better results.
- **Discounted Cumulative Gain (DCG):** a statistic that weights correct results near the front of the list more than correct results later in the ranked list under the assumption that a user is less likely to consider elements near the end of the list. Higher numbers are better.

3.2 How the PSB was used in this project

Programmers, from all over the world, use different shape descriptors in their 3D shape retrieval algorithm and use the Princeton Shape Benchmark to evaluate and assess their 3D shape retrieval algorithm. By using the PSB in this project, it gives us the opportunity to compare our results to the results of other algorithms using different shape descriptors. Some other shape descriptors, not including SIFT, are [4]:

- D2 Shape Distribution (D2)
- Extended Gaussian Image (EGI)
- Complex Extended Gaussian Image (CEGI)
- Shape Histogram (SHELLS)
- Shape Histogram (SECTORS)
- Shape Histogram (SECSHEL)
- Voxel
- Spherical Extent Function (EXT)
- Radialized Spherical Extent Function (REXT)
- Gaussian Euclidean Distance Transform (GEDT)
- Spherical Harmonic Descriptor (SHD)
- Light Field Descriptor (LFD)

In this project specifically, we mainly evaluated our algorithm based on the following tools: nearest neighbor, first-tier, second-tier, E-measure, and DCG. We compared our results (NN, FT, ST, E-M, and DGC) with the results of other algorithms.

4. SIMILARITY DISTANCE

4.1 What is similarity distance?

As mentioned above in section 2, the Euclidean distance is used in SIFT to compute the difference between two feature vectors. However, finding the distance between two feature vectors does not automatically transfer to how similar the two objects are. Because every image has several feature vectors, 3D shape retrieval algorithms need to be able to compare all the matched feature vectors of an image and create an overall similarity distance for two objects.

For this project, each object had 20 views. In our algorithm, when trying to match two objects, we had to compare all 20 corresponding views, find the matching feature vectors for each view, and calculate an overall similarity distance for the two objects; the smaller the similarity distance, the more similar the objects are.

4.2 How the similarity distance was computed using SIFT matching

Throughout the process of the project, many different drafts of the similarity distance equation were created. The distance equation began as a fairly simple equation; the similarity distance was just the inverse of the number of matches (using vl_ubcmatch) between image 1 and image 2 over the average number of key points on image 1 and 2.

For 20 views, (x) being the view number:

$$S = S + \frac{\text{\# of matches on view (x) between image 1 and image 2}}{\text{average of key points on image 1 and image 2, view (x)}}$$

Finally, at the end of the algorithm:

$$D = \frac{1}{S}$$

However, after testing this equation on several different images, we discovered that the vl_ubcmatch code does not give the same results when we switch the order of the images. When using the vl_ubcmatch function, the format of the code looks like this:

[matches, scores] = vl_ubcmatch(**image1**, **image2**);

Left of the equal sign are the two matrices that will store the matches as well as the Euclidean distance of the matches between image1 and image2. However, the order of the images on the right side of the equation is crucial; by switching the format to:

[matches, scores] = vl_ubcmatch(**image2**, **image1**);

it will change the results of the matches and scores. We decided to add in an additional equation to our algorithm to make sure we included all the matches from the two images; the final matches between two images would resulted from comparing image 2 to image

1, as well as the matches resulted from comparing image 1 to image 2. So, the similarity distance equation evolved into:

For 20 views, (x) being the view number:

$$S = S + \frac{\#\text{ of matches on view (x) between image 1 and image 2}}{\text{average of key points on image 1 and image 2, view (x)}}$$

$$P = P + \frac{\#\text{ of matches on view (x) between image 2 and image 1}}{\text{average of key points on image 1 and image 2, view (x)}}$$

Finally, at the end of the algorithm:

$$D = \frac{1}{\left(\frac{S+P}{2}\right)}$$

4.3 How the similarity distance compared to other results

Before any filters were put on the matches yielded from the SIFT matching algorithm, $D = \frac{1}{\left(\frac{S+P}{2}\right)}$ was used when testing our algorithm using the PSB base classification.

By only using the above equation in our algorithm, the results are close to the results of the D2 shape descriptor. Figure 3 shows where our results rank with the other 12 shape descriptors using the PSB base classification:

Shape Descriptor	Discrimination				
	Nearest Neighbor	First Tier	Second Tier	E-Measure	DCG (%)
LFD	65.7%	38.0%	48.7%	28.0%	64.3
REXT	60.2%	32.7%	43.2%	25.4%	60.1
SHD	55.6%	30.9%	41.1%	24.1%	58.4
GEDT	60.3%	31.3%	40.7%	23.7%	58.4
EXT	54.9%	28.6%	37.9%	21.9%	56.2
SECSHEL	54.6%	26.7%	35.0%	20.9%	54.5
VOXEL	54.0%	26.7%	35.3%	20.7%	54.3
SECTORS	50.4%	24.9%	33.4%	19.8%	52.9
CEGI	42.0%	21.1%	28.7%	17.0%	47.9
EGI	37.7%	19.7%	27.7%	16.5%	47.2
SIFT (our algorithm)	**31.6%**	**19.5%**	**28.8%**	**16.9%**	**48.4**
D2	31.1%	15.8%	23.5%	13.9%	43.4
SHELLS	22.7%	11.1%	17.3%	10.2%	38.6

Fig 3: Comparing our results with the results of the other 12 shape descriptors using the PSB base classification

The algorithms used in our results table (Figure 3) are all similar in that they all include three main steps. The differences between the algorithms lie mainly in the details of their shape descriptors. The three main steps are as follows:

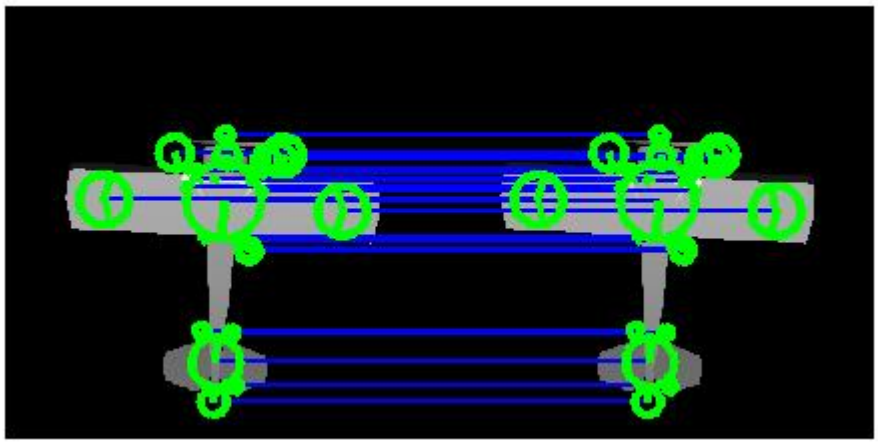
1. **Normalization**: normalize the models for differences in scale and possibly translation and rotation
2. **Shape Descriptor**: generates a shape descriptor for each model
3. **Distance**: computes the distance between every pair of shape descriptors

5. FILTERING SIFT MATCHES USING EUCLIDEAN DISTANCE

5.1 Irrelevant Comparison using the SIFT matching algorithm

Because our results were not as high as expected, we started adding in filters to the SIFT match results. When using the SIFT matching algorithm, the algorithm goes through each feature vector on image 1 and compares them to all the feature vectors, one by one, on image 2. When graphing the images and their matches, we saw that although the comparing is an extremely thorough process, it sometimes makes irrelevant comparisons. For example, when matching two images of a plane, the SIFT matching algorithm might mistakenly match the head of the plane in image 1 to the tail of the plane in image 2. Figure 4 shows an example of a perfect match as well as an example of a poor match.

A)



B)

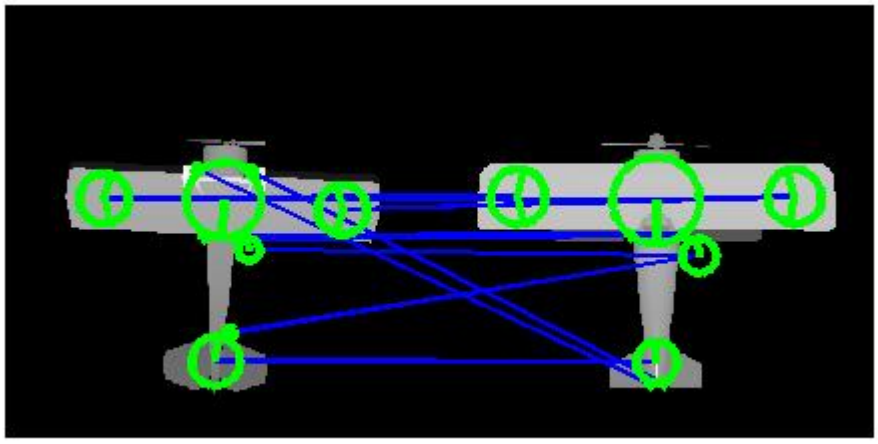


Fig 4.
A) An example of a perfect match
B) An example of an irrelevant comparison using the SIFT matching algorithm

After we received results from our first distance computation (section 4.2), we tried to filter out the results the SIFT matching algorithm returned. The first method we tried included sorting through the Euclidean distances stored in the 'scores' matrix from the SIFT matching function. We knew, from researching Euclidean distances, that the smaller the distance, the more similar the two feature vectors are. In distance measurements, there is a threshold value that yields the maximum and best results. We decided to find that threshold value through a long guess and check process.

5.2 Finding the Threshold Value

When we first started filtering the matches using the Euclidean distance, we took an educated guess for what the threshold value was. We first considered having the threshold be 10% of the size of the image. However, the units of the images and the Euclidean distances are not the same, so it would be hard to convert and compare the two sizes.

We then looked at a few 'scores' matrices, to see what the Euclidean distances were for matches that were very similar and what the distances were for matches that were very dissimilar. We observed that most of the matches that were irrelevant had distances over 30000; most of the irrelevant matches had at least 4 significant figures. From this observation, we made our first threshold value 30000. We decided that if the distance stored in the 'scores' matrix is over the threshold value, we would not count that as a 'match'.

This method is different from our first similarity distance computation because before, we included all the matches in the calculations (refer to section 4.2), relevant and irrelevant. Now, instead of having the numerator be the number of all the matches, it will now be the modified number of matches. Every time a Euclidean distance in the 'scores' matrix is over the thresh value, the number of matches will decrease by one. The pseudo-code is as follows:

```
Thresh_value = 30000;
---------------------------------------------

[matches, scores] = vl_ubcmatch (image 1, image 2);

Number of matches = original number of matches (without filtering);

Have a for-loop that goes through all the values in the scores matrix:
        If (value > thresh_value)
                Number_of_matches = Number_of_matches – 1;
        End

S = S + Number_of_matches [# of matches (after filter) between image 1 & 2];
                Average number of key points on image 1 and image 2
```

[matches2, scores2] = vl_ubcmatch (image 2, image 1);

Number_of_matches2 = original number of matches (without filtering);

Have a for-loop that goes through all the values in the scores2 matrix:
    If (value > thresh_value)
        Number_of_matches2 = Number_of_matches2 – 1;
    End

P = P + Number_of_matches2 [# of matches (after filter) btwn image 1 & 2];
        Average number of key points on image 1 and image 2

After comparing all 20 views of two images:

$$D = \frac{1}{\left(\frac{S+P}{2}\right)}$$

After implementing the filter using the Euclidean distances, with the threshold value being 30000, our results increased only minimally. However, after further narrowing down the threshold value, we concluded that the threshold value needs to be around 4000 to yield the best results. With the threshold value now being 4000, our results ranked just above the results of the Complex Extended Gaussian image (CEGI) shape descriptors (refer back to figure 3 in section 4.3). Yet, we still wanted our results to improve, so we began researching and reading about other algorithms that also used SIFT and some kind of additional filter.

6. OTHER METHODS USING SIFT

Several articles were read during the research portion of this project. There were two SIFT methods, in particular, that had results better than the LFD results. Those two methods were outlined in an essay titled ‘Salient Local Visual Features for Shape-Based 3D Model Retrieval’ [5]. The two methods are the Bag-Of-Features SIFT and the Individual Match SIFT.

6.1 Bag-Of-Features SIFT and Individual Match SIFT

The Bag-Of-Features SIFT (BF-SIFT) algorithm and the Individual Match SIFT (IM-SIFT) algorithm are fairly similar. Both compares 3D models using David Lowe’s original SIFT algorithm, but both add in additional step to the SIFT algorithm to filter the results to achieve more accurate matches. They differ in their ways to compute the similarity distance between two sets of visual features.

The BF-SIFT uses the 'bag of words' approach to compute the similarity distance. The algorithm renders range images of the models from several viewpoints and uses SIFT to find local features on each of the viewpoints. It then converts those local features into a 'visual word' using a 'visual codebook'. The quantized local features are then accumulated into a histogram, storing the frequencies of each of each 'visual word'. The result of using BF-SIFT is a global feature vector specific to the image, generated from the histogram. Kullback-Leibler divergence (KLD) is then used to calculate the dissimilarity among a pair of feature vectors.

The IM-SIFT algorithm is aimed at retrieving rigid models using local features. The IM-SIFT performs full pose normalization, including scale, position, and rotation. By having the entire image normalized and leveraging the positions of the features, the IM-SIFT algorithm tries to avoid irrelevant comparison of local features. The algorithm only compares features extracted from corresponding areas in the pose-normalized coordinate frame. Also, to improve the performance, the IM-SIFT compute two independent distances using two pose normalization methods, mass-PCA and normal-PCA [5]. At the last stage of the algorithm, it only takes the minimum of the two distances (the smaller the distance, the more similar) as the similarity distance. Figure 5 is a picture of how IM-SIFT compares the features. To learn more about the BF-SIFT or IM-SIFT method, refer to the article mentioned above.

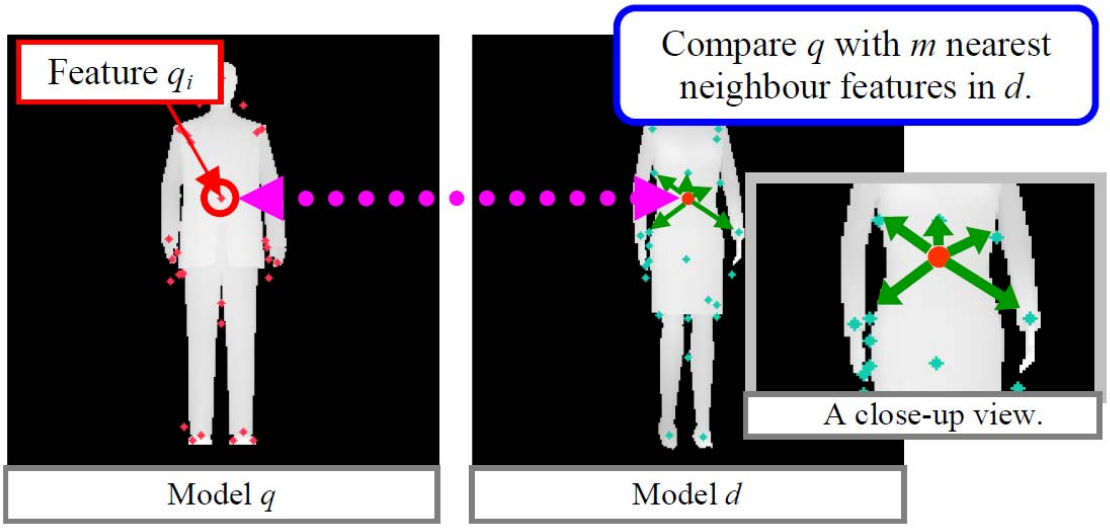


Fig 5. A feature in the model to the left is restricted to its proximity in the model to the right in the pose-normalized coordinate space. The correspondence assumes successful pose normalization.

7. FILTERING SIFT MATCHES USING SPATIAL RESTRICTIONS

7.1 Spatial Restriction Filtering

The original SIFT matching algorithm, as seen in section 5, does not account for spatial locations when matching feature vectors. Without spatial restrictions, the matching algorithm will make irrelevant matches. Although we cannot completely prevent the

program from making irrelevant matches, spatial restrictions will help filter out the significantly extraneous matches.

After reading about the other methods that used SIFT, the ideas used in the IM-SIFT technique seemed the easiest to modify and replicate. The algorithm we were writing already used the original SIFT algorithm to create local feature descriptors and matched the different images on corresponding viewpoints. Since the algorithm has all the feature vectors and their matches stored in memory, instead of rewriting the entire code to only match feature vectors in corresponding areas, we will use the 'matches' matrix and filter out the results that are too far, spatially, from each other.

7.2 How far is spatially 'too far'?

In order for this filtering to work accurately, we had to determine how large our 'spatial boundaries' will be. Just like how we analyzed the Euclidean distances in the 'scores' matrix, as explained in section 5, we now analyzed the x and y coordinates of each of the feature vectors. After studying the coordinates of the descriptors, we decided to make the 'area sections' +/- 15 units away, for the x and y coordinates, from the feature vector. This number was chosen through a process of guess and check after the first educated guess, after studying the coordinate of the descriptors. After the first estimated number, a series of distances were tested to see which distance yielded the best results. After several tries, the distance is chosen as 15 for x and y coordinates.

So what does this number, 15, mean when the algorithm is filtering the results? This numbers means that if a feature descriptor is at coordinate (63.9808, 150.9524), then they will only consider feature descriptors between (48.9808, 135.9524) and (78.9808, 155.9524) a match. We are able to do this simply because all 3D models were normalized in scale, position, translation, and rotation in the beginning of the algorithm. If the models were not normalized, this method would not work.

This process of filtering in our program is very similar to the IM-SIFT algorithm. IM-SIFT sets the spatial boundaries before the matching algorithm is run. In our algorithm, since the matching has already been completed and the matches are already been stored in memory, we just applied the spatial filter last.

7.3 How spatial filtering affects the similarity distance computation

By adding in spatial restriction filtering to our algorithm, it affected the similarity distance computation just like how the Euclidean distance filter did. For the spatial filtering, there was an if-statement in the program:

> In addition to the 'S' and 'P' variable calculations, we add in a 'Q' variable.
>
> ---

Num_of_matches3 = number of original matches using SIFT matching algorithm;

If (the 'matched' feature vector is not located inside the spatial boundaries)
Num_of_matches3 = num_of_matches3 – 1;
End

Q = Q + Num_of_matches3 [# of matches (after filtering) between image 1 & 2];
Average of key points on image 1 and image 2

At the end:

$$D = \frac{1}{\left(\frac{S+P+Q}{3}\right)}$$

8. RESULTS

8.1 Our Results compared to the results yielded from other shape descriptors

When we used the Princeton Shape Benchmark to evaluate our final algorithm, our results increased drastically after applying the two filters, the Euclidean distance filter and the spatial restriction filter. Before adding in the spatial restrictions, the results yielded near the CEGI results. However, by including a spatial boundary to our matching algorithm, our results now yielded above the EXT results. A visual comparison of the results can be seen in Figure 6.

Shape Descriptor	Discrimination				
	Nearest Neighbor	First Tier	Second Tier	E-Measure	DCG
LFD	65.7%	38.0%	48.7%	28.0%	64.3%
REXT	60.2%	32.7%	43.2%	25.4%	60.1%
SHD	55.6%	30.9%	41.1%	24.1%	58.4%
GEDT	60.3%	31.3%	40.7%	23.7%	58.4%
SIFT (Our algorithm)	**51.8%**	**30.7%**	**40.9%**	**24.4%**	**57.2%**
EXT	54.9%	28.6%	37.9%	21.9%	56.2%
SECSHEL	54.6%	26.7%	35.0%	20.9%	54.5%
VOXEL	54.0%	26.7%	35.3%	20.7%	54.3%
SECTORS	50.4%	24.9%	33.4%	19.8%	52.9%
CEGI	42.0%	21.1%	28.7%	17.0%	47.9%
EGI	37.7%	19.7%	27.7%	16.5%	47.2%
D2	31.1%	15.8%	23.5%	13.9%	43.4%
SHELLS	22.7%	11.1%	17.3%	10.2%	38.6%

Fig 6: Comparing our results, after adding in the Euclidean distance filter and the spatial restrictions, with the results of the other 12 shape descriptors using the PSB base classification

Although the nearest neighbor from my algorithm falls fairly low on the table (between SECTORS and VOXEL), the rest of the evaluation results are comparably higher than most of the results yielded from other shape descriptors. As stated in section 4, by using only the original SIFT algorithm and our similarity distance computation equation, the results yield around the results of the D2 shape descriptor. By adding in spatial filters, it doubles the percentages in the results.

The precision and recall graph, for the 13 shape descriptors, is shown in figure 7. As seen in the graph, SIFT graph falls near the EXT & GEDT coordinates.

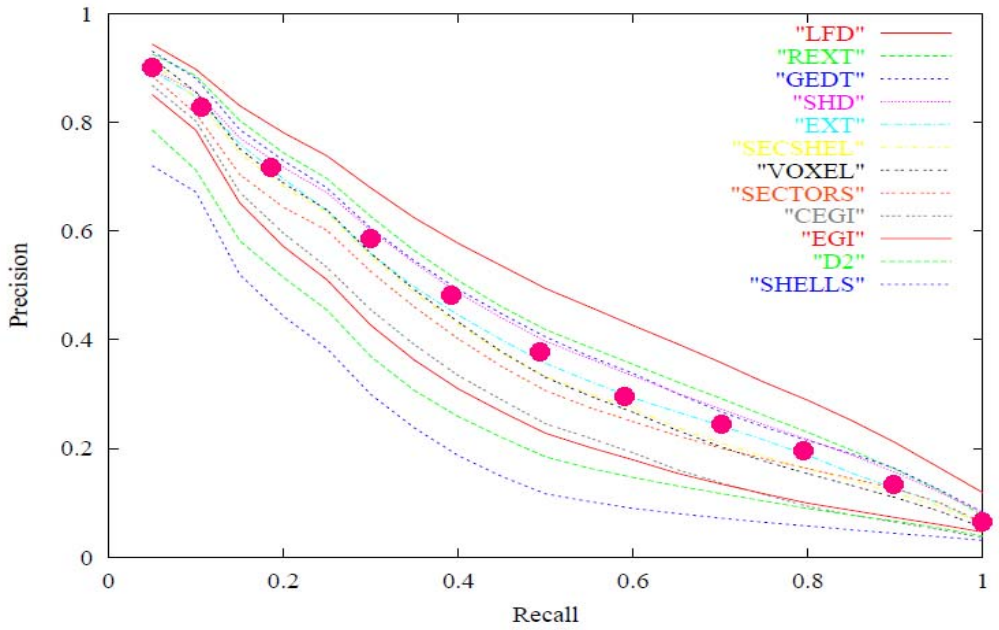


Fig 7. Precision-recall curves computed for 12 shape descriptors for tests with the PSB base classification. The results of the SIFT algorithm are represented by the dots, not connected by a line (not drawn to scale).

The precision and recall coordinates for the SIFT algorithm is in the table below, figure 8.

Recall	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
Precision	0.9022	0.8151	0.7293	0.6799	0.6337	0.574	0.5267	0.4803	0.4402	0.3975
	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1
	0.3659	0.3357	0.3074	0.2775	0.2456	0.2141	0.1802	0.1474	0.1148	0.0806

Fig 8. Table of the recall and precision coordinates for the SIFT algorithm.

9. CONCLUDING REMARKS

Overall, using David Lowe's Scale Invariant Feature Transform algorithm, to compare 3D shape models, yields acceptably good results. It can be concluded that using SIFT in parallel with further filtering and implementing spatial restrictions can produce results better than most other shape descriptors. Although the results from the SIFT algorithm did not achieve the highest results, it performed relatively well and ranked high among the results yielded from using other shape descriptors.

Despite the many advantages of the SIFT algorithm, there are still some things that can be improved in the algorithm. For the next algorithm, instead of setting the spatial restrictions after the comparison, it would be better to write the algorithm similar to the IM-SIFT algorithm. Although setting the restrictions after the matching saved energy from rewriting the matching algorithm, it did not reduce the cost of feature comparison. By setting the boundaries before the matching process, it would have saved time and memory. Instead of comparing all of the feature vectors, the program would only compare those shape descriptors inside the spatial boundaries.

Another part of the code that could be improved would be the similarity distance equation. The equation used in the SIFT algorithm was just the simple average of the variables 'S', 'P', and 'Q'. More time could be spent on finding a better distance equation so that the program could yield better results. Instead of finding the percentage of 'correct' matches (what the variables 'S', 'P', and 'Q' represented in this SIFT algorithm), the similarity distance could be, for example, related to the Euclidean distance of the shape descriptors.

In conclusion, there is one last limitation to be mentioned about the SIFT 3D shape retrieval algorithm. This algorithm assumes that the objects being compared are normalized in scale, position, translation and rotation. However, if this algorithm were applied to computer vision or geometric modeling, the objects used then may not be normalized, and thus the results of this algorithm would be skewed.

10. REFERENCES

[1] Iyer, N., Jayanti S., Lou, Kui., Kalyanaraman, Y., Ramani, K.: Three-dimensional shape searching: state-of-the-art review and future trends.

[2] Kendall DG. The diffusion of shape. Adv Appl Probab 1977; 9:428-30

[3] Lowe, David G.: Distinctive Image Features from Scale-Invariant Key points. January 5, 2004.

[4] Shilane, P., Min, P., Kazhdan, M., and Funkhouser, T.: The Princeton Shape Benchmark.

[5] Ohbuchi, R., Osada, K., Furuya, T., Banno, T.: Salient Local Visual Features for Shape-Based 3D Model Retrieval.

[6] Tangelder, Johan W.H., Velkamp, Remco C.: A survey of content based 3D shape retrieval methods. 2007.