Shape retrieval on the wavelet density hypersphere

Glizela Taino[1], Yixin Lin[2], Mark Moyou[3], and Adrian M. Peter[4]

[1]Department of Computer Science and Engineering, Washington University in St. Louis; glizelataino@gmail.com
[2]Department of Computer Science and Mathematics, Duke University, Durham, North Carolina; yixin.lin@duke.edu
[3]Department of Engineering Systems, Florida Institute of Technology, Melbourne, Florida; mmoyou@my.fit.edu
[4]Department of Engineering Systems, Florida Institute of Technology, Melbourne, Florida; apeter@fit.edu

July 2016

Abstract

This paper explores shape retrieval, comparison, and classification through the general framework of wavelet density estimation. First, we review the motivation and formal problem of shape retrieval, along with an extended review of historical and contemporary shape feature signatures and retrieval paradigms. Then we explain the paradigm of wavelet density estimation, along with potential intermediary feature extraction techniques such as the Laplace-Beltrami Operator, which brings us to the unit hypersphere geometry. We begin our investigation with different potential clustering schemes on this unit hypersphere, especially delving deeply into hierarchical clustering, before investigating a custom version of divisive hierarchical shape retrieval. We also prove the runtime of divisive and agglomerative hierarchical clustering, based on some assumptions about the k-means algorithm used to construct the hierarchy. We test this on a multitude of datasets, including SHREC 2011 [**?**] and MPEG-7 [**?**], and demonstrate the practicality of this retrieval scheme. We then move to a deeper examination of our feature representation, first by explaining the rationale and mechanics behind wavelet density estimation, and then optimizing this by streamlining the density estimation code and also with the use of linear assignment-based shape deformation. We provide significant evidence that these optimizations enhance the performance of the density estimation significantly.

1 Introduction

There has recently been a significant increase in the number of 2D and 3D shapes, and corresponding developments have progressed our understanding of measuring, classifying, and retrieving these shapes. The problem of shape is significant for several reasons, both practical and theoretical: because shape is a uniquely discerning attribute for visual understanding (compared to e.g. color, texture, scale), it has many potential applications from 3D digital design to robot vision. Thus, shape is a fundamental problem in computer vision and shape retrieval is an important piece of the quest to make machines intelligent.

The framework of shape retrieval is generally to take an unknown 2D and 3D shape known as the *query* and return a list of *results*, a series of known shapes (correspondingly 2D or 3D) in order of similarity. This is generally done solely with the geometry of the shape, not with textual metadata or other contextual information.

Pratically, shape retrieval is generally broken up into three stages: shape representation, feature representation, and retrieval mechanics.

The shape representation can be 2D points, a 3D point cloud, or a 3D mesh. This shape representation undergoes preprocessing which extracts information and transforms the shape into a feature representation that can be subsequently used as a signature for clustering similar objects. This feature representation/signature is then operated on by retrieval mechanics, which stores the initial database of known shapes and, given an unknown query, retrieves the actual list of ordered known shapes.

We initially used the Laplace-Beltrami Operator (LBO) signature and wavelet density estimation for our feature representation. This mapping (and, in general, our framework of wavelet density estimation) results in a useful interpretation as geometry on a unit hypersphere manifold. Our next investigation was to find the best way to cluster on the multidimensional unit hypersphere's curved manifold, we executed an experiment with sample data constructed using the Von-Mises distribution (a generalization of the Gaussian, or normal, distribution, on a circle) on a sphere in 3D. This was verified the veracity and practicality of an implementation of spherical k-means on this sample data.

As an implementation detail, we investigated two possible implementations of computing the mean of points on a hypersphere. The first is a mathematically rigorous implementation of a differential geometry concept called the Karcher mean, while the second is an estimation of this that exploits the structure of the unit hypersphere. We verified that the two were effectively equal for our purposes.

After verifying that the clustering algorithm works as expected, we used it as a subroutine in our spherical hierarchical clustering investigation. The idea is to form hierarchies which capture different abstraction layers of the shapes, which may speed up the process of shape retrieval. We performed another experiment to decided between agglomerative and divisive hierarchical clustering. Next, its runtime, storage, and overall performance of both approaches was tested. We also analyzed both algorithms using standard complexity theory techniques. Finally, we tested these on the SHREC 2011 [**?**] and MPEG-7 [**?**] datasets, which are standard shape datasets. This demonstrates that the hierarchical retrieval works as intended, and requires only $O(\log n)$ for a single query shape, instead of $O(n)$.

We then turn our attention to optimizing our feature representation. There was a significant amount of optimization to our wavelet density estimation which enabled a multiple-order-of-magnitude increase in speed across all dimensions and resolutions. This was possible by exploiting the grid-like structure of the wavelet density estimation process and utilizing parallelization. This enables the processing of much larger datasets, including the Princeton ModelNet datasets (the full version which contains 127,915 CAD models).

Finally, we investigated a clever optimization of the shape dissimilarity metric (the hypersphere distance metric) by first warping closer the shapes being compared. The intuition is that the "difficulty" of warping similar shapes is much smaller than warping two different shapes, and such a process will make the distance between two similar shapes smaller without affecting the distance between different shapes. This process utilizes a solution to classic combinatorial optimization problem of *linear assignment*, which proves key for this warping to be effective and tractable. We then test this modified dissimilarity metric to the original and expand on the results.

The layout of this paper is as follows: First, we provide a broad introduction to the standard shape retrieval framework, with motivations, applications, and a brief overview of previous work in the field. We proceed to delve deeply into the wavelet density estimation paradigm, specifically by detailing our promising work on optimizing the coefficient estimation process. This section also includes a in-depth primer on wavelet theory and justification for using wavelets for probability density estimation purposes. We proceed to lay some of the groundwork for our use of hierarchical clustering on the unit hypersphere, which includes some practicality testing of several differential geometry packages as well as an overview on k-means on a sphere. We then provide an exposition on our implementation of spherical hierarchical clustering, including proofs of algorithmic complexity on both agglomerative and divisive hierarchical clustering. We conclude with the Shape L'Ane Rouge, or sliding wavelet coefficients, which aids in our distance metric, as well as a summary of our results in extending this technique to multiple resolutions. This requires a brief overview of the linear assignment problem, a classic in combinatorial optimization problems.

Part I
Introduction to shape classification and retrieval

Shape analysis is the fundamental computer vision problem of helping machines understand shape. Here, we focus primarily on the subproblem of shape retrieval: given a *query shape*, how can we best retrieve the most similar shapes (*search results*) from an already-known catalog of shapes? Shape retrieval is as similarly fundamental to the problem of shape analysis as search engines to was to the Internet.

The standard general framework for shape retrieval is to consume some shape representation (e.g. 2D points, 3D point-clouds, or 3D meshes), extract a feature representation that captures the essence of the shape, and use retrieval mechanics to output a list of shapes based on similarity.

2 Motivations

There has been several developments which make shape analysis both exciting and crucial in recent years. First, the number of available models (2D and 3D shapes) available over the Internet, partly due to ever-decreasing costs of creating such models and partly due to the concerted efforts of researchers to construct domain-specific databases. Second, our understanding of the principles behind computer vision have advanced greatly, due to both the necessity of making sense of enormous amount of data being generated every day [**?**], and meteoric advances in our ability to solve them [**?**].

It is interesting to note that, fundamentally, our understanding of images and objects comes directly from our ability to understand their shape. To see this most intuitively, consider the two versions of the same picture in figures 1 and 2 which we immediately recognize as containing the same semantic meaning, despite the stripping away of most attributes in 2. We could change the speed, scale, amount of distortion, and almost any other characteristic parameter and still retain the same meaning due to shape.

AMALTHEA REU Technical Report No. 2016–5; available at www.amalthea-reu.org. Please send your correspondence to Georgios C. Anagnostopoulos.

Figure 1: Original photograph of still life

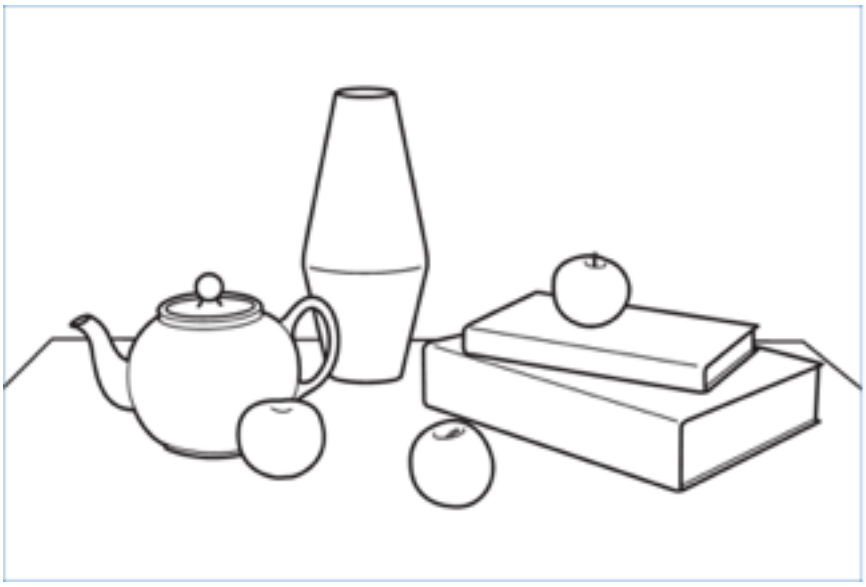

Figure 2: Same scene with purely shape information

3 Applications

Shape retrieval has a plethora of potential applications, which stems from how fundamental shape analysis is to the problem of computer vision. Solving shape retrieval has as far ranging effects as helping robots quickly make sense of their environment, helping build billion-dollar 3D animation franchises, and becoming an enabler of information retrieval for commerce and virtual/augmented reality on the scale that search engines did for text-based websites. Ultimately, fully understanding the implications of solving this problem may be impossible because the problem is so basic.

4 Previous work

Shape analysis, classification, and retrieval has been developed quite extensively, especially in last decade, in part due to a rise in the number of available models and techniques to process them. In order to effectively exploit this rise in usable shapes, the development of a comparison and retrieval system is necessary, prominently in the 3D model search engine developed at Princeton [**?**], as well as at the National Research Council of Canada [**?**], the National Taiwan University [**?**], and others [**?**, **?**, **?**].

There is a broad swath of literature dedicated to 3D retrieval shape retrieval, with many different methods with their particular strengths and weaknesses. All of these must be evaluated with respect to several salient attributes: foremost, robustness and discrimination, but also efficiency, necessity of preprocessing, ability to partially match, and strictness of requirements for data (e.g. mesh versus point cloud, closed meshes, etc.). Evaluating robustness and performance can be done with several standard datasets, including Princeton's Shape Benchmark [**?**] and the MPEG-7 dataset [**?**, **?**].

4.1 Global features

One category of papers that aid in 3D retrieval are those which use global features, i.e. features which look at the whole shape.

For example, Zhang and Chen [**?**] proposes an algorithm to extract global features like volume, moments, and Fourier transform coefficients from a mesh representation. It does this by simply summing up the features on each of the elementary shapes that form this mesh. Though efficient, it uses non-sophisticated features and requires a mesh shape representation. They then extend [**?**] this with a clever trick with manual annotation, which selects the least-known shape (based on their feature representation) and chooses that as the next annotation.

Other global features are explored by Corney et al. [**?**], who create an online (Internet-based) 3D shape search engine named ShapeSifter. Their coarse filters, which are used to filter and classify before more fine distinctions and distances are made, are based on a novel application of convex hulls. The three indices they use to do a preliminary filtering of candidate 3D shapes include *hull crumpliness* (the ratio of the object's surface area to the surface area of its convex hull), *hull packing* (the percent of the convex hull volume not occupied by the original object), *hull compactness* (the ratio of the convex hull's surface area cubed over the convex hull's volume squared). Though extremely quick, this work is at best a rough first approximation that require the use of more differentiating methods.

In the same vein of speedy similarity measures, Kolonias et al. [**?**] propose the use of the following shape descriptors: the aspect ratio, a binary 3D shape mask, and (more complex) the edge paths of the models. The set of paths that outline the shape of the 3D object is extracted using a proposed algorithm which extracts those geometric properties from an input VRML file. These sets are compared between 3D objects by first checking the similarity of angles and lengths, and then finding some metrics on corresponding paths.

A method which is only superficially similar to ours is that of Horn [**?**], who represent the shapes of surfaces by mapping each polygon of the 3D shape to the corresponding unit normal point on the Gaussian (unit) sphere, weighted by the surface area. This extended Gaussian image uniquely defines a convex polyhedron, up to a translation. This is extended by Kang and Ikeuchi [**?**], which simply stops discarding the distance to the origin of the polygon by encoding it onto the imaginary part of the complex sphere. However, these both require pose normalization.

One famous and highly cited innovation in using global feature for shape retrieval involves the concept of a global feature distribution, which uses the distribution of features rather than the features directly. Osada et al. [**?**] explores this concept by measuring properties like distance, area, volume, and angle between random points of the relevant shape, constructing a distribution, and using this to compare to other shapes. The most effective is the D2 shape distribution, which is the distribution of random points on the surface of the shape.

Shape distributions are quite good at interclass differentiation (i.e. between classes of shapes), though not as good at intraclass differentiation [**?**] (i.e. between shapes which belong to the same broad class but still require distinction) because they require finer representations.

4.2 Spatial features

Spatial features distinguish themselves by somehow capturing and using the spatial location of the 3D model for use in comparisons. In general, they requires pose normalization to be effective since they aren't rotationally invariant.

One early approach in spatial features is proposed by Ankerst et al. [**?**], who simply use 3D shape histograms. This was applied to molecular surfaces by taking a histogram of the points in concentric shells (and subdividing those shells into sectors) around a 3D shape's centroid.

Kazhdan et al. [**?**] propose an approach that can transform any rotation dependent shape descriptors to rotation independent one through the use of spherical harmonics. They essentially calculate spherical harmonics on concentric spheres of the shape and then use frequency analysis to construct a descriptor indexed only by radius and frequency. This is invariant to rotations.

Another highly cited paper is the the work by Novotni and Klein [**?**], which uses 3D Zernike descriptors instead of spherical harmonics to obtain rotational invariance. This is an improvement over spherical harmonics because spherical harmonics cannot detect internal rotations of the object.

4.3 Local features

An approach that has some similarities with our work is that of Shum et al., which presents a way to measure similarity between 3D shapes. They map local curvatures a distribution on a sphere, and then construct a distance metric between these spherical approximations of 3D shapes. However, this requires closed surfaces.

Ohbuchi et al. [**?**] use SIFT features extracted from range images and put into a bag-of-features representation. The range images are sampled uniformly from the view sphere.

Shilane and Funkhouser [**?**] proposed a scheme for selecting the distinctive local feature descriptors of a set of such feature descriptors, find the ones that help distinguish them the best, and finally using only those descriptors (as opposed to the full set), which leads to faster performance at a low cost.

A paper similar in approach is by Bai et al. [**?**], which combines contour, skeleton, global, and local features as a feature. This achieves good results on MPEG-7 without requiring many training samples.

4.4 View-based features

The idea of view-based features is that much of intrinsic information of a 3D shape can be encoded into multiple 2D views from different angles of that shape. Therefore, it's possible to compare two shapes by simply comparing the views.

A recent paper by Zhu et al. [**?**] uses deep learning to learn a feature representation of a shape using autoencoders on 2D views of a 3D shape. They combine this with the local descriptors by the previous Ohbuchi et al. [**?**] to obtain state-of-the-art results.

Part II
Optimization of wavelet density estimation

5 Introduction to wavelet density estimation

Wavelet density estimation was a promising technique for shape matching, as explored by Peter and Rangarajan in their 2008 paper [**?**]. Generally, the goal is to estimate a probability density function over the shape points and use that representation to compare shapes. The density is expanded expansion into a wavelet basis form, which has a theory with strong mathematical foundations and provides several advantages over competing function bases (e.g. the Fourier basis, or simple histograms).

This wavelet density estimation framework can be applied for any dimensional data (e.g. 1D, 2D, and 3D shapes) by using the tensor product (multidimensional) forms of the 1D wavelet bases. The coefficients of this functional basis (of the probability density function) form our feature representation, and also conveniently map to the unit hypersphere due to constraints we impose during density estimation. This manifold geometry allows us to use a simple, efficient, and well-defined distance metric as our shape dissimilarity measure.

6 Wavelet density estimation

Our feature representation is formed by estimation of the probability density function of the shape, which we model as sample points from a distribution.

First, we must explain the use of wavelets as the foundation of our density estimation framework. An L^2 function, also known as a square-integrable function, is a real- or complex-valued function whose square integrates to a finite number. Probability density functions fall under this function space, therefore allowing us to use wavelets for density estimation using a linear combination of these basis functions. In other words, wavelet density estimation as a paradigm is built on strong mathematical foundations.

The equation for 1D wavelet density estimation is

$$p(x) = \sum_{j_o,k} \alpha_{j_o,k} \phi_{j_o,k}(x) + \sum_{j \geq j_o,k}^{j_1} \beta_{j,k} \psi_{j,k}(x) \tag{1}$$

where $\phi(x)$ and $\psi(x)$ are the scaling and wavelet basis functions, also known as the "father" and "mother" wavelets, respectively. The functions can also be set to some starting resolution level that determines its dilation and contraction. To extend to multiple resolutions, we incorporate the wavelet basis function with a starting (j_o) and stopping (j) resolution level. In order to form a basis, we need to make multiple copies of the function over different translates, k. These functions are scaled by their scaling and wavelet basis function coefficients, $\alpha_{j_o,k}$ and $\beta_{j,k}$. To easily

retain properties of true densities, such as non-negativity and integration to one, we compute the square root of the probability density function

$$\sqrt{p(x)} = \sum_{j_o,k} \alpha_{j_o,k}\phi_{j_o,k}(x) + \sum_{j \geq j_o,k}^{j_1} \beta_{j,k}\psi_{j,k}(x) \tag{2}$$

Since we optimized for 2D wavelet density estimation, we consider the second dimension with $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ and $\mathbf{k} = (k_1, k_2) \in \mathbb{Z}^2$. We also extend the equation to

$$\sqrt{p(\mathbf{x})} = \sum_{j_o,\mathbf{k}} \alpha_{j_o,\mathbf{k}}\phi_{j_o,\mathbf{k}}(\mathbf{x}) + \sum_{j \geq j_o,\mathbf{k}}^{j_1} \sum_{w=1}^{3} \beta_{j,\mathbf{k}}^{w}\psi_{j,\mathbf{k}}^{w}(\mathbf{x}) \tag{3}$$

where we calculate for the basis functions by using the tensor product method

$$\begin{aligned}
\phi_{j_o,\mathbf{k}}(\mathbf{x}) &= 2^{j_o}\phi(2^{j_o}x_1 - k_1)\phi(2^{j_o}x_2 - k_2) \\
\psi_{j,\mathbf{k}}^{1}(\mathbf{x}) &= 2^{j}\phi(2^{j}x_1 - k_1)\psi(2^{j}x_2 - k_2) \\
\psi_{j,\mathbf{k}}^{2}(\mathbf{x}) &= 2^{j}\psi(2^{j}x_1 - k_1)\phi(2^{j}x_2 - k_2) \\
\psi_{j,\mathbf{k}}^{3}(\mathbf{x}) &= 2^{j}\psi(2^{j}x_1 - k_1)\psi(2^{j}x_2 - k_2)
\end{aligned} \tag{4}$$

A mathematical advantage in using wavelets is its compact support, meaning they are nonzero on a small domain. Any point falling outside of the basis function's support does not contribute to the basis function value at that translation.

Intuitively, the scaling and wavelet basis function coefficients determine the height of its basis function. These coefficients thus uniquely define the probability density function of a shape, which means we can use them as our feature representation in shape retrieval. In order to initialize coefficients to a reasonable first approximation, we use a histogram approach. This is shown in the following equations:

$$\alpha_{j_o,\mathbf{k}} = \left(\frac{1}{N}\right) \sum_{i=1}^{N} \frac{\phi_{j_o,\mathbf{k}}(\mathbf{x})}{\sqrt{p(\mathbf{x})}} \tag{5}$$

$$\beta_{j_o,k} = \left(\frac{1}{N}\right) \sum_{i=1}^{N} \frac{\psi_{j,\mathbf{k}(\mathbf{x})}}{\sqrt{p(\mathbf{x})}} \tag{6}$$

As we estimate the coefficients (alpha and beta) of our probability density function, we use a loss function to determine whether our coefficients are as close to optimal as possible. To do this we try to minimize a negative loglikelihood objective function with respect to the coefficients

$$-\log p(X; \{\alpha_{j_o,k}\beta_{j,k}\}) = -\frac{1}{N}\sum_{i=1}^{N} \log \left[\sum_{j_o,k} \alpha_{j_o,k}\phi_{j_o,k}(x_i) + \sum_{j \geq j_o,k}^{j_1} \beta_{j,k}\psi_{j.k}(x_i)\right]^2 \tag{7}$$

where $X = \{x_i\}_{i=1}^{N}$. To determine the direction of the gradient descent, we compute the following expression:

$$-2\left(\frac{1}{N}\right) \sum \frac{\phi_{j_o,\mathbf{k}}(\mathbf{x})}{\sum \alpha_{j_o,\mathbf{k}}\phi_{j_o,\mathbf{k}}(\mathbf{x})} \tag{8}$$

In order for maintain essential properties of density estimation, we constrain the coefficients to one

$$\sum_{j_o,k} \alpha_{j_o,k}^2 + \sum_{j \geq j_o,k}^{j_1} \beta_{j,k}^2 = 1 \tag{9}$$

This constraint conveniently maps our feature representation into a unit hypersphere which allows us to take advantage of its geometric properties to signify similarity and dissimilarity between shapes in shape retrieval.

7 Implementation and our optimization

The wavelet density estimator code by Peter and Rangarajan accurately constructs the coefficient vectors (the alphas and betas as seen in **??**) and thus estimates the densities, but because of the sheer complexity of the procedure, there was significant room to improve the runtime. A full estimation over the standard shape databases can take multiple days even with relatively low resolution. For instance, the Brown database [**?**] has 99 shapes, the Swedish Leaf database [**?**] has 1125 shapes, and the MPEG7 database [**?**, **?**] has 1400 shapes, which are quite large already; the Princeton ModelNet database is orders of magnitude larger, at 127,915 shapes. At these scales, the inefficiency of the Peter-Rangarajan code is obvious (i.e. ModelNet estimation becomes intractable). Optimization is thus a primary priority.

The Peter-Rangarajan follows the full mathematical implementation, which means there exist superfluous calculations. For instance, they loop over every sample point and evaluate every basis function individually. We added an optimization of only calculating the basis functions for relevant sample points under specified basis functions with the hope to cut time required for estimation.

8 Wavelet density estimation optimization

Now that we have an overview of wavelet density estimation and the equations used, we can delve into our optimization. We focus our efforts on the 2D form of the density estimation on a given point set shape representation. We then use the coefficients that uniquely characterize the density function as our feature vector.

The resolution levels and domain size controls the number of translations. Originally, for a database such as MPEG7, which has 1400 shapes, an estimation process with a domain of [-0.5,0.5] in the both x and y directions, resolution levels 2 to 3, and 1344 translates would take roughly 16.8 hours. This is a particularly low resolution, and if raised we would expect an even slower runtime. To optimize this code for speed, we focus on the bottleneck: the calculation of the initial coefficients and negative log likelihood cost value with its gradient.

8.1 Initializing Coefficients

This function calculates equations (5) and (6) to compute initial coefficients that will be updated later through a loss-minimizing optimization process. The time spent for this calculation with the same parameters mentioned above is roughly around 1.8 of the 16.8 hours.

Given a point set shape representation, the wavelet density estimation framework covers the shape with basis functions at each translation. Each function has a coefficient that needs to be calculated. The bottleneck exists because of the way the code structures the computation: it code computes equations (4) for a *single point over all translations* using a Kronecker tensor product [**?**]. This means that if we have 1344 translations, it would perform 1344 operations for a single sample point. If a shape is made up of 4007 sample points, then it would be performing a total of 5,385,408 operations for a single shape. This calls for an excess amount of redundant computations since most of the scaling and wavelet basis functions for a single point over all translations return the value of zero because the observed sample point does not contribute to most basis functions, i.e. does not exist under the compact support of the basis function at specific translations. The solution is to change the structure of the looping:

Instead of looping through each point and finding which basis functions it falls under, we looped through basis functions along the horizontal direction, evaluated its basis function value (4) based on the points that fall under its support, and placed them into a matrix that holds these values. We then store this matrix to be used for later optimized computation (for the negative log likelihood cost value and gradient). Once we have this matrix of values, we can evaluate the basis function coefficients with the relevant points under each translation using equation (5) and (6).

8.2 Negative log likelihood

The second area where the bottleneck occurs is in computing for the negative log likelihood cost value (7) and the function's gradient (8). We use these equations to check whether we have found the optimal coefficient values and which direction in which to move to optimize. The original time spent for this computation was around 13.9 of the 16.8 hours.

The original code actually replicated much of the work done in initialize coefficients. It would take a single point and calculate the basis function over all translations, resulting in needless computation. It does this to attain a matrix of the basis function values to perform the appropriate operations to solve for the cost value and gradient.

Since we already performed most of the heavy computation to attain this matrix in initializing coefficients, our solution was to simply pass in the needed matrix of basis function values for each translation and perform the proper computations. This effectively solves for the negative log likelihood cost value and gradient while ultimately eliminating loops.

9 Results

As mentioned above, our goal is to estimate a density function of a 2D shape using wavelets. This allows us to extract the coefficients as our feature representation to be used in shape retrieval on a unit hypersphere. However, with MPEG7 containing 1400 shapes, domain [-0.5,0.5], resolution level 2 to 3, and 1344 translates, it would take about 16.8 hours to calculate the coefficients. Our optimization yields significant improvements over the original code.

The original time it took the code to calculate for only the initial coefficients was about 1.8 hours. After our optimization, the run time to compute the initial coefficients cut all the way down to 0.13 hours, or about eight minutes. Ultimately, the computation runs about 100 times faster. As for calculations for the negative log likelihood cost value and gradient, our optimization shaved the lines of code from 54 lines to 21 lines, ridding it of loops. We essentially do away with 61% of lines of code. The run time was promising as well. To calculate the cost value and gradient for seven iterations per shape, computation time went from 13.9 hours to 0.089 hours, or around 5 minutes for the entire dataset. The optimized computation runs 100 times faster. Overall, to estimate the wavelet densities of a database of 2D shapes with the optimized code under the specified parameters would take 0.3 hours, or 17.8 minutes, running a little under 100 times faster.

With this optimization, we hope that researchers using the wavelet density framework can achieve full estimation of the coefficient values quickly, and spend more time in developing new and better methods of shape retrieval.

Part III
Clustering on a sphere

Traditional clustering is done in Euclidean space, but this fails for a variety of reasons on a manifold. For instance, traditional k-means fails on a hypersphere because Euclidean means will inevitably leave the manifold (for example, taking the mean of two vectors at an obtuse angle will result in a mean that exists inside the sphere).

We thus use differential geometry techniques to investigate clustering on a hypersphere, which is the manifold upon which our feature representations live. First, we investigate ways to generate Gaussian-distributed data on spherical surfaces, using the Von Mises distribution (later generalized to the Von Mises-Fisher distribution for n-dimensional spheres). Then we extend k-means to hypersphere manifolds, which first requires the investigation of hypersphere distances and means.

10 Von Mises distribution

Intuitively, the Von Mises distribution [**?**] is a simple approximation for the normal distribution on a circle (known as the *wrapped normal distribution*). The Von Mises probability density function is defined by the following equation:

$$P(x) = \frac{e^{b\cos(x-a)}}{2\pi I_0(b)}$$

where $I_0(x)$ is the modified Bessel function of the first kind; the Von Mises cumulative density function has no closed form.

The mean $\mu = a$ (intuitively, the angle that the distribution clustered around), and the circular variance $\sigma^2 = 1 - \frac{I_1(b)}{I_0(b)}$ (intuitively, b is the "concentration" parameter). Therefore, as $b \to 0$, the distribution becomes uniform; as $b \to \infty$, the distribution becomes normal with $\sigma^2 = 1/b$.

11 Von Mises-Fisher distribution

The Von-Mises Fisher distribution is the generalization of the Von Mises distribution to n-dimensional hyperspheres. It reduces to the Von-Mises distribution with $n = 2$. The probability density function is defined by the following equation:

$$f_p(\boldsymbol{x}; \boldsymbol{\mu}, \kappa) = C_p(\kappa)\exp(\kappa \boldsymbol{\mu}^T \boldsymbol{x})$$

where

$$C_p(\kappa) = \frac{\kappa^{p/2-1}}{(2\pi)^p I_{p/2-1}(\kappa)}$$

and intuitively is an approximation for the normal distribution on the hypersphere.

Example code for constructing random (Von Mises-Fisher distributed) points on a sphere can be found in the Appendix, at section 24.

12 Hypersphere means

A basic operation in Euclidean space is the act of taking the mean of a set of points. For example, k-means, which is a fundamental clustering algorithm, depends on this operation. However, this simple averaging procedure must be adapted to work on manifolds. To see intuitively why this is true, imagine two vectors on the hypersphere. The straight Euclidean mean of the two vectors actually ends up inside the sphere, not on the surface! Taking the Euclidean mean almost certainly results in a vector not on the manifold, so we require some tools from differential geometry to find means on manifolds.

12.1 Karcher mean

The Karcher mean is a geometric mean of various matrices and an extension of the well known geometric mean of two matrices. It is also referred to as the Riemannian geometric mean. In our investigation, we used it as a way to compute the average distance between two points on a hypersphere while obeying the curvature of the manifold. The process is stated in Algorithm 1. In order to do this we must incorporate a Logarithm map and Exponential map on the manifold.

The Logarithm map is defined as a function that takes a point, ρ_2, on the manifold and maps its projection vector, γ, on the tangent plane at an origin point, T_{p_1}.

$$\rho_2 = Exp_{\rho_1}(\gamma) = cos(|\gamma|)\rho_1 + sin(|\gamma|)\frac{\gamma}{|\gamma|} \tag{10}$$

On the first iteration, ρ_1 is the origin point of the tangent space.

The Exponential map can be thought of as the inverse function of the Logarithm map. It is a function that takes in a vector on the tangent plane at origin point and maps it on the hypersphere. As the difference between the updating vectors on the tangent space begin to show little change, it is thought as the optimal mean between the two points on the manifold.

$$\gamma = Log_{\rho_1}(\rho_2) = \tilde{\rho}\frac{cos^{-1}(\langle\rho_1, \rho_2\rangle)}{\sqrt{\langle\tilde{\rho}, \tilde{\rho}\rangle}} \tag{11}$$

where $\tilde{\rho} = \rho_2 - \langle\rho_2, \rho_1\rangle\rho_1$. Karcher mean, as we have mentioned before, is calculated in a way to obey the curvature of the hypersphere.

12.2 Spherical mean

Unlike the Karcher mean, the spherical mean does not go along the curvature of the hypersphere. In the assignment step of this algorithm, where we adjust the cluster centers, we find the mean by summing up all the vectors in a cluster and normalizing it back onto the manifold.

$$x_l = \frac{\sum_{i \in X_l} \rho_i}{||\sum_{i \in X_l} \rho_i||}, l = 1 \dots K$$

where X_h are the center of each K number of clusters. This is the normalized gravity center of the new cluster.

12.3 Practical comparison

To find the best approach in calculating a cluster center, we put Karcher mean and Spherical K-mean to the test. We tested them in five types of situations in where the angles are either identical, orthogonal, opposite, two random angles, or multiple random angles as shown in Figure 3.

We found that Karcher mean and Spherical k-means produced the same results as long as the step size for Karcher mean was set to one. But both approaches failed in the case where the angles were opposite.

13 k-means clustering

k-means clustering is a simple and popular algorithm for the *unsupervised clustering problem*, the task of grouping a set of observations so that a group is "similar" within itself and "dissimilar" to other groups. k-means partitions n observations into k clusters, with each observation belonging to the nearest mean of the cluster. This problem is NP-hard in general, but there are heuristics which guarantee convergence to a local optimum.

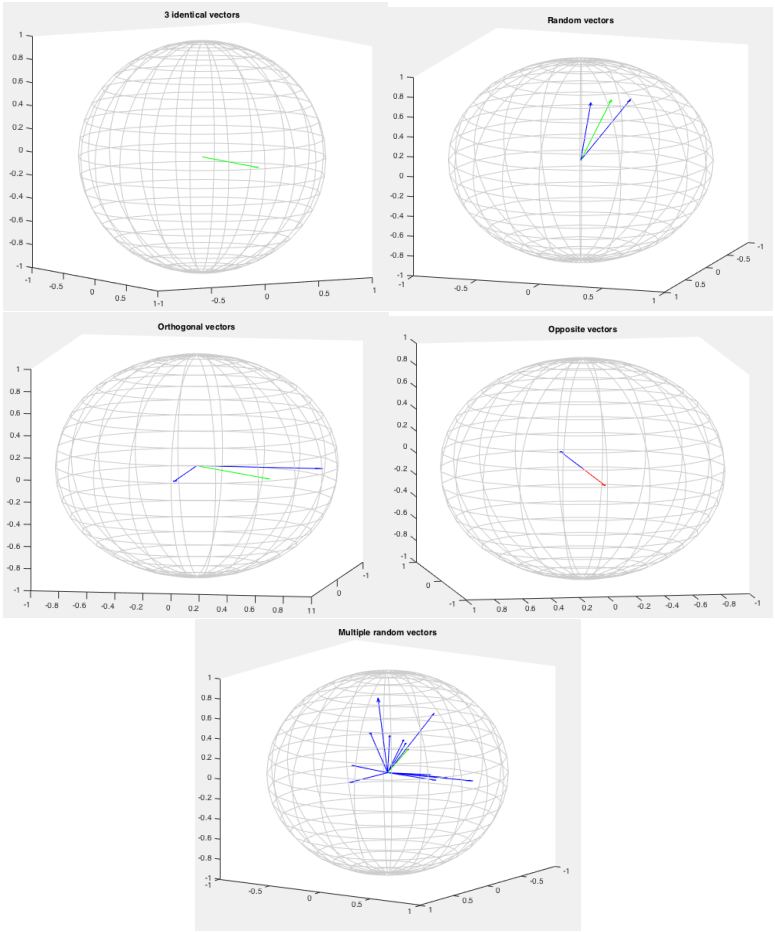

Figure 3: Different test cases comparing accuracy of Karcher mean and spherical means. Blue vectors are the vectors being averaged, green are Karcher means, red are spherical means.

Algorithm 1 Lloyd's algorithm for k-means clustering

1: generate an initial set of k means
2: **while** not converged **do**
3: assign all data points to nearest Euclidean-distance mean
4: calculate new means to as the centroids of the observations in the cluster
5: **end while**

The standard heuristic (known as *Lloyd's algorithm* [**?**]) is the following:

There is a choice of initialization method. The *Forgy method* randomly picks k observations as initial means, while the *Random Partition method* randomly picks a cluster for each observation.

The Lloyd's algorithm is a heuristic, so it does not guarantee a global optimum. Furthermore, there exists sets of points in which it converges in exponential time. However, it has been shown to have a smoothed polynomial running time, and in practice converges quickly.

14 Spherical k-means clustering

Spherical k-means clustering is the same idea, however, the points are not on a flat euclidean space but rather on the hypersphere. We investigated a MATLAB implementation by Nguyen [**?**, **?**], which required a mean-and-norm-normalized dataset located on a hypersphere. Important aspects of this implementation include:

- When there exists an empty cluster, the largest cluster is split
- Use the dot product as "negative distance", which leverages the fact that observations are unit vectors on the hypersphere
- Use the normalized sum of observations as a centroid/mean, which leverages the fact that observations are unit vectors on the hypersphere. Note that this fails on pathological cases where the sum of observations is zero.

For example, if you have two vectors that are opposite of each other, their sum is the zero vector.

14.1 Our investigation

In order to test how well the spherical k-means clustering algorithm worked, we constructed a random data set using the Von Mises distribution random sampling function from the MATLAB Circular Statistics Toolbox [**?**]. We constructed 70 clusters of 10 points each, using random points on the sphere as means and with step size $\kappa = 100$, and then applied the spherical k-means clustering and visualized the results on the unit three dimensional sphere, color coded for each cluster and with estimated means labeled as red X's and true means labeled as black X's.

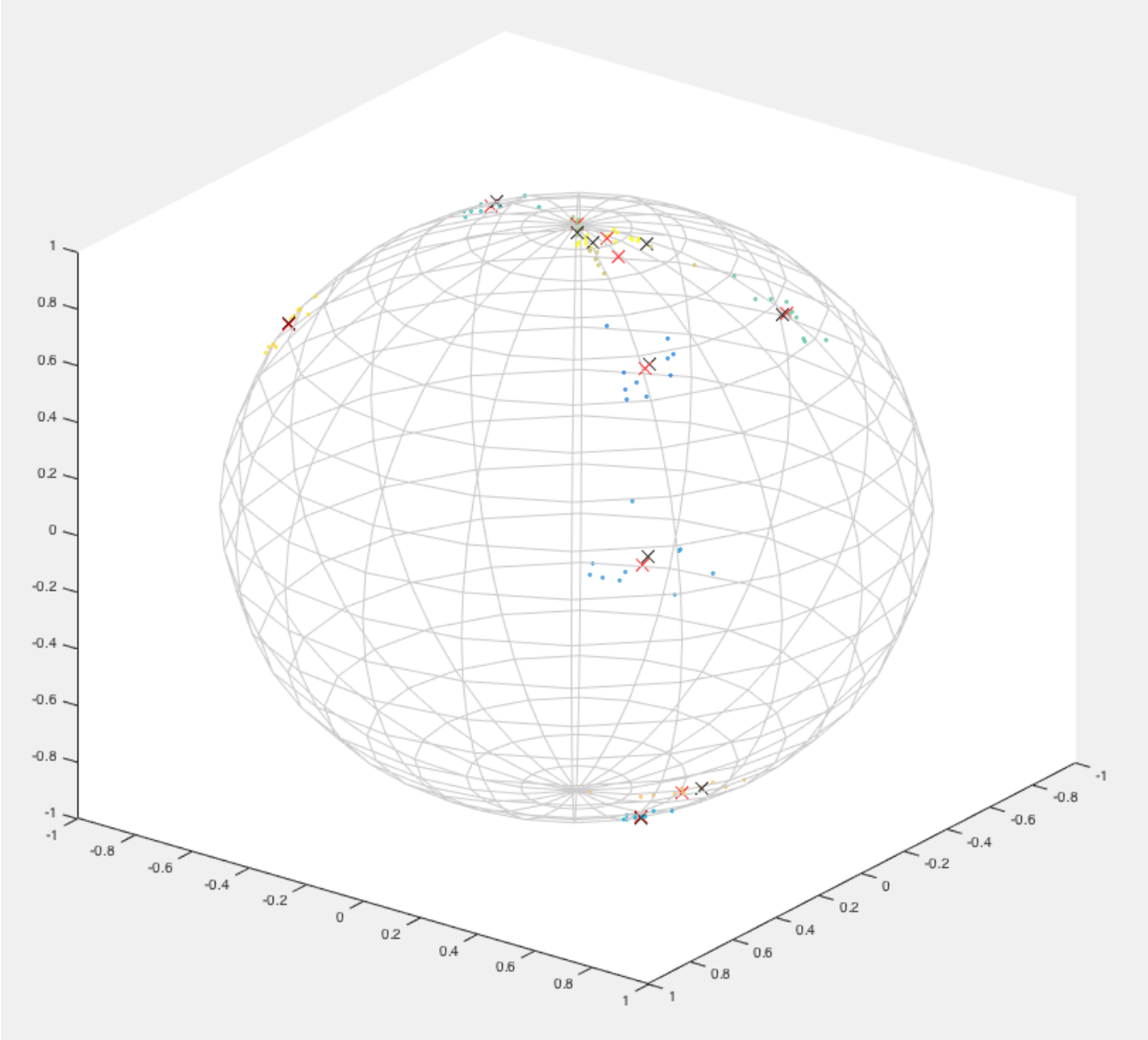

Figure 4: Plot of 10 clusters and means on a 3-D sphere. Red x's are computed means, black x's are actual means; because they line up fairly well, the implementation is effective.

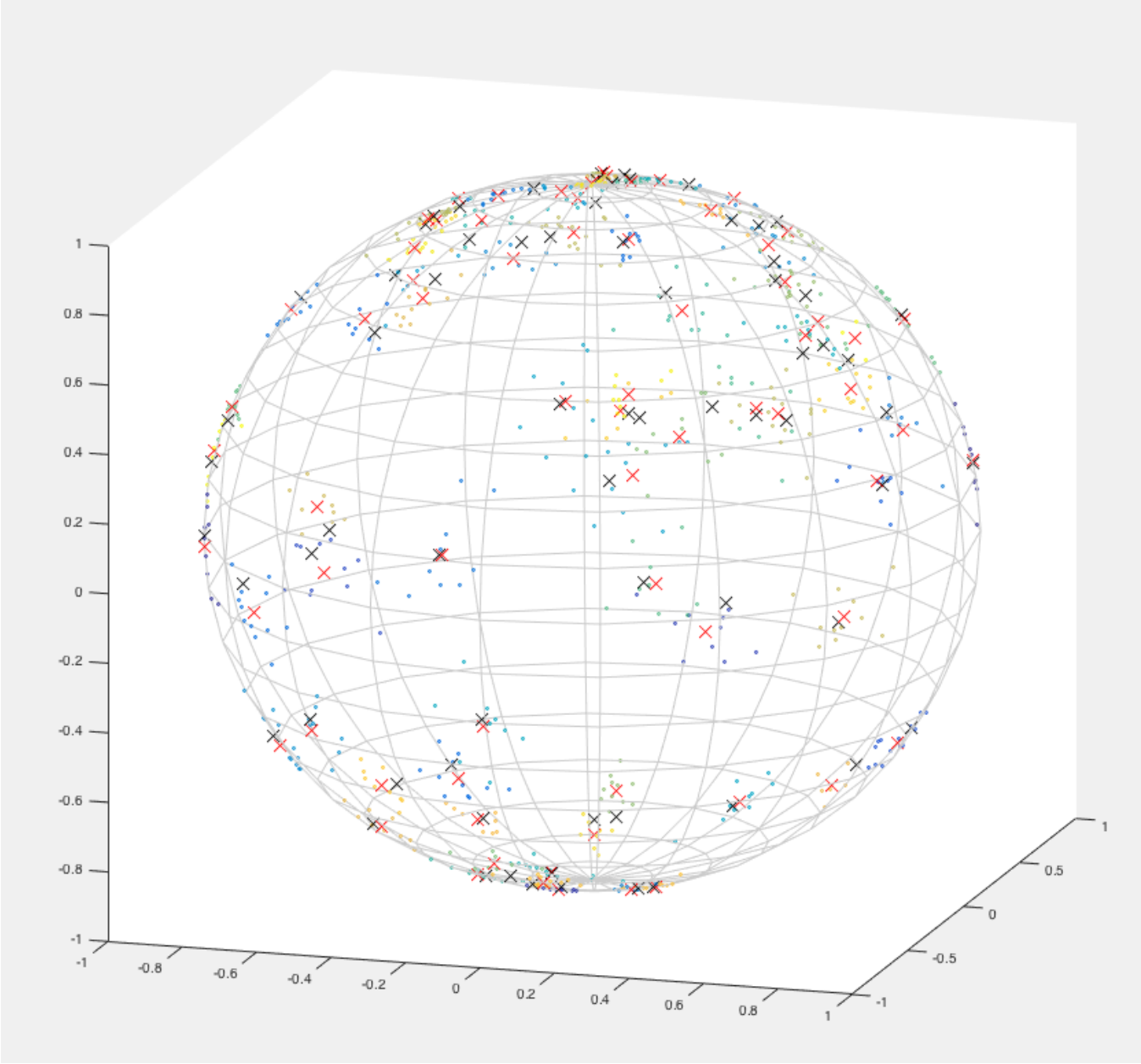

Figure 5: Same experiment with 70 clusters number of clusters.

Part IV
Hierarchical clustering

We want to develop the most efficient way to group data based on a measure of dissimilarity. We explored the topic of hierarchical clustering which is used in data mining and statistics. Hierarchical clustering can be implemented in two ways: divisive and agglomerative. We will elaborate on these two variants in this paper to compare the its algorithmic complexity, or how much resources are needed when computed, and the different types of cluster linkage criterion. Further information on cluster analysis can be found in [**?**].

15 Introduction to hierarchical clustering

The idea behind hierarchical clustering is to capture the different abstraction layers in a hierarchy, i.e. moving from the broadest levels of differences down to finer distinctions. The idea is to not only speed up the retrieval process (which drops from a linear amount of comparisons to logarithmic) but also to increase the accuracy of retrieval. The standard approach is to cluster at different levels of dissimilarity (generally by using Euclidean distance) and build a

tree from this. We explore an alternative approach by generalizing $k-$means to a hypersphere, though many of the paradigms of hierarchical clustering also apply.

16 Divisive and agglomerative clustering

Divisive clustering [**?**] is also known as the "top down" method. This takes in a data set as one large cluster. Then as it moves down the hierarchy, it recursively splits the data until it reaches the leaves of the tree, i.e. each observation is its own singleton cluster.

Agglomerative clustering can be thought to do the opposite of divisive clustering and is known as a "bottom up" strategy. It takes in a data set and it initially looks at each observation as its own singleton cluster. Then, based on its linkage criterion, it will merge recursively to form new clusters until all the data is in one large cluster.

A dendrogram [**?**] can be used in both cases to record and visualize the hierarchical nature of the clusters and their distance to each other.

17 Cluster dissimilarity measurements

In forming clusters, a measurement must be made to determine which clusters split or merge in regards to divisive or agglomerative clustering, respectively. There are many different ways to measure dissimilarity but this paper will explore single linkage, complete linkage, and centroid linkage. Each of the measurements below refer to ways in measuring distances between k number of clusters $c_{i=1...k}$ in relationship with specific points $x_{i=1...k}$ in those clusters. Because different linkage patterns will result in different clusters being formed, it is an important parameter to vary in hierarchical clustering.

17.1 Single linkage

Single linkage takes the distance between the closest observations, or minimum distance, in each cluster and merges those with the shortest distance. This works well for data sets that desire to cluster together long chained data points. Let $D(c_i, c_j)$ be the distance between two centroids for $i, j \in \{1, 2, ...k\}$, and $D(x_i, x_j)$ be the distance between two data points in $i, j \in \{1, 2, ...n\}$. Then single linkage defines the following relationship:

$$D(c_1, c_2) = \min_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2) \tag{12}$$

17.2 Complete linkage

Complete linkage is similar to single linkage. It takes the distance between the farthest observations, or the maximum distance, in each cluster. The clusters that have the shortest measured distance are merged together. This measurement is ideal for data that is spherically organized relative to each other. The relationship is defined as

$$D(c_1, c_2) = \max_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2) \tag{13}$$

17.3 Centroid linkage

Using centroid linkage involves taking the average of all points in a cluster and assigning that value as the mean centroid. Then the centroids with the shortest distances are merged together. Centroid linkage is defined as

$$D(c_1, c_2) = D\left(\left(\frac{1}{|c_1|}\sum_{x \in c_1} \vec{x}\right), \left(\frac{1}{|c_2|}\sum_{x \in c_2} \vec{x}\right)\right) \tag{14}$$

18 Clustering implementations

We developed both hierarchical divisive and agglomerative clustering functions to explore which approach clustered more effectively and efficiently (i.e. algorithm time complexity). In our particular application, instead of flat clustering, we had to cluster over the curved surface of a sphere. In order to do this, we utilized the spherical k-means function [**?**]. We also chose our linkage criterion to be based on centroid linkage.

Essentially, we developed the divisive clustering by following the intuition mentioned above. For our agglomerative clustering implementation, we did a slightly different variation hoping its run time would be slightly better than the traditional method. In our code, intuitively, we initiated each point as its own cluster with its own centroid. Then we recursively merged two clusters together based on centroid linkage to form a new cluster with its own cluster centroid each time. This process stops when there is a single point to represent the entire data's average centroid.

We tested with a relatively small, three dimensional data set of 150 points. In both divisive and agglomerative, we input different cluster values into spherical k-means which determines how often the function gets called. In the divisive method, we asked spherical k-means to return two clusters. This means the number of times we call spherical k-means is based on how many nodes we have which is $m = 2n - 1$ where n is the number of data points. For the agglomerative case, we asked spherical k-means to return the $n/2$ clusters. This means means we had to call spherical k-means at every level which is $l = \lceil \log_2 n \rceil$.

As a benchmark, we also used the MATLAB `linkage` function to construct a pairwise-distance based tree (merging nodes together by single-linkage) and compared the results.

18.1 Implementation details

In k-means, there is a choice of initial mean initialization. This affects the initial distribution of means and therefore potentially the number of iterations until convergence, but also may be expensive computationally. For instance, a good setting of the initial means will reduce the likelihood of empty clusters found, which is dealt with by splitting the largest cluster and is therefore computationally expensive.

For the agglomerative algorithm, it makes sense to spend time initializing the means (we chose to maximize the distance between the initial means), since the number of calls to spherical k-means is small and the number of clusters for each call is high. For the divisive algorithm, it doesn't make sense to incur this expense when spherical k-means is called many times with only the branching-factor number of clusters (generally a small constant). This optimization and lack thereof, respectively, reduces the running time for both algorithms substantially.

18.2 Algorithm complexity analysis

Let n be the number of data points, k be the number of clusters (number of means), d be the dimensionality of the data, and i be the number of iterations until convergence. The runtime of the k-means algorithm (including spherical k-means) is $O(nkdi)$.

Let us assume that the cost of initializing the means and splitting empty clusters is negligible, i.e. $O(1)$. In practice, the two are related: we find that explicitly picking good starting means for the agglomerative algorithm (e.g. randomly, or by maximizing distance between the means) makes the probability of empty clusters small, whereas it becomes excessive and unnecessary for the divisive algorithm. See 18.1 for more discussion.

We will also assume that d and i are constants. Though in degenerate data sets i can be an exponential function of n, in practice i is a small constant on datasets with clustering structuring.

Let us also assume that we choose a branching factor of 2, i.e. a binary hierarchical tree structure. The following analysis does not depend on the branching factor, assuming it is constant.

18.2.1 Agglomerative complexity

Theorem.

$$T_{aggl}(n) = \Theta(n^2 di)$$

Proof. Our agglomerative algorithm has a recurrence relation of the following form:

$$T_{aggl}(n) = T(\frac{n}{2}) + nkdi$$

for some constant c, since we divide the problem size into two for every recursive call. But we choose $k = \frac{n}{2}$ each time, since the number of means we choose to cluster with is also half of n.

We use the master theorem [**?**] to solve this recurrence relation:

Master theorem variable	Value
a	1
b	2
$f(n)$	$\frac{n^2 di}{2}$
c	2
$log_b(a)$	0

This satisfies case 3 of the master theorem:

$$f(n) \in \Omega(n^c) \text{ s.t. } c > log_b(a)$$

since $f(n) \in \Omega(n^2)$ s.t. $c = 2 > log_b(a) = 0$, and

$$af(\frac{n}{b}) \leq k_0 f(n) \text{ for some } k_0 < 1 \text{ and sufficiently large } n$$

since when $k_0 = \frac{1}{4}$, $\left\{ f(\frac{n}{2}) = (\frac{1}{4})\frac{n^2 di}{2} \right\} \leq \left\{ k_0 f(n) = (\frac{1}{4})\frac{n^2 di}{2} \right\}$
and so, by the master theorem: $T_{aggl}(n) = \Theta(n^2 di)$. □

18.2.2 Divisive complexity

Theorem.

$$T_{div}(n) = \Theta(n \log(n) di)$$

Proof. Our divisive algorithm has a recurrence relation of the following form:

$$T_{div}(n) = 2T(\frac{n}{2}) + nkdi$$

for some constant c, since we divide the problem size into two for every recursive call and we have two subproblems each time. But we choose $k = 2$ each time, since the number of means we choose to cluster with is always the branching factor.

Problems of this form are solvable using a highly prominent and useful algorithm called the master theorem [**?**], which (given certain assumptions about the corresponding recurrence relation) gives the time complexity of many common recursive algorithms.

We use the master theorem to solve this recurrence relation:

Master theorem variable	Value
a	2
b	2
$f(n)$	$n(2)di$
c	1
$log_b(a)$	1

This satisfies case 2 of the master theorem:

$$f(n) \in \Theta(n^c \log_{k_0} n) \text{ s.t. } c = log_b(a)$$

by letting $k_0 = 0$, since $f(n \log^0(n)) = f(n) \in \Theta(n)$ s.t. $c = 1 = log_b(a) = 1$.
and so, by the master theorem: $T_{div}(n) = \Theta(n \log(n) di)$. □

18.2.3 Conclusion

As we demonstrate with our complexity proofs, the divisive complexity algorithm is actually faster (linearithmic versus quadratic), which guides our use of the divisive algorithm in our experiments.

19 Hierarchical results

Our unsupervised clustering algorithm had reasonable but unpromising results. However, using a supervised implementation which guides the clustering using the correct labels produced quite promising results, summarized in table 8.

Figure 6: Supervised hierarchy results (accuracy measured in bullseye metric)

	SHREC11	MPEG7	BROWN
Without hierarchy	20.7%	75.3%	51.8%
With hierarchy	99.7%	99.3%	83.0%

Part V
Shape L'Ane Rouge: sliding wavelets

20 Sliding wavelet coefficients

We use a feature representation that incorporates the use of a density estimation using wavelets, which brings us to a d-dimensional hypersphere. However, the original density estimation was over the 2- or 3-dimensional space, i.e.

each coefficient of represents the amplitude of the wavelet in a region of 2D or 3D space. We then compute distances between the probability density functions through use of the angle or arc between the points on the hypersphere; intuitively, this distance metric measures how different the corresponding regions are in amplitude.

However, if the shapes are not aligned (specifically, if there are non-rigid transformations between them), there may be mistakes in using this distance metric. The goal of sliding the wavelet coefficients, as described in Shape L'Ane Rouge [**?**], is to reduce non-rigid differences between the densities before taking the distance between them.

20.1 Review of wavelet density estimation

Our goal is to compute a robust distance metric between shapes such that the essential information inherent in the shape is captured while the irrelevant information is reduced. This follows the paradigm where similarities between shapes are considered after removing certain classes of transformations [**?**, **?**].

We first take the problem of representing shapes in such a way. Given a 2D or 3D point cloud, we attempt to measure it either directly as a probability density function or after applying certain transformations (e.g. the Laplace-Beltrami operator). We estimate $\sqrt{p(x)}$ and then obtain $p(x) = (\sqrt{p(x)})^2$ in order to sidestep many of the problems associated with estimating probability density functions.

Let $\boldsymbol{x} \in \mathbb{R}^2$, j_1 be some stopping scale level for multiscale decomposition, and $\boldsymbol{k} \in \mathbb{Z}^2$ be an index into the spatial location of the basis. In 2D, the wavelet expansion of densities is given as follows:

$$\sqrt{p(x)} = \sum_{j_0,\boldsymbol{k}} \alpha_{j_0,\boldsymbol{k}} \phi_{j_0,\boldsymbol{k}}(\boldsymbol{x}) + \sum_{j \geq j_0,\boldsymbol{k}}^{j_1} \sum_{w=1}^{3} \beta_{j,\boldsymbol{k}}^{w} \psi_{j,\boldsymbol{k}}^{w}(\boldsymbol{x}) \tag{15}$$

with the coefficients $\alpha_{j_0,\boldsymbol{k}}$ and $\beta_{j,\boldsymbol{k}}^{w}$ representing "how dense" the probability density function is at that spatial point.

Because of the unit integrability of probability density functions, we can interpret a coefficient as a point on the unit hypersphere, and can use the corresponding distance metric (i.e. angle or arc length between any two points).

20.2 Motivation for sliding

Suppose there are two shapes which are identical except for a translation such that none of the points overlapped. If we take the above density estimation technique, the wavelet coefficients will have a high distance on the unit hypersphere (i.e. the dot product between the two vectors is 0, so we get a distance of $\frac{\pi}{2}$). However, these shapes are the same and our goal is to have a low distance between similar shapes. In other words, we want to reduce the effect of these transformations that don't inform us about the intrinsic difference between shapes. If we had a method to match up corresponding points with each other and then take the hypersphere distance, we could have a much more robust distance metric to use.

In Shape L'Ane Rouge, we attempt to warp one density into the shape of another before taking the hypersphere distance between them, using linear assignment.

20.3 Use of linear assignment

Linear assignment informally attempts to match up a set of objects with another set of objects (usually with the same cardinality) in such a way as to minimze cost. We use this to match up each coefficient in the vector with another in order to find the maximum overlap between the two. We penalize large shifts by using some multiple λ of the Euclidean distance between regions in the original point/point-cloud representation (otherwise, any shape would be able to fit any other shape).

20.4 Implementation

Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be the two coefficient vectors that we are taking the distance between. We attempt to match $\boldsymbol{c_1}$ with $\boldsymbol{c_2}$ using linear assignment, with a cost matrix

$$C = -(\boldsymbol{c_1} \otimes \boldsymbol{c_2}) + \lambda D \tag{16}$$

where $\boldsymbol{c_1} \otimes \boldsymbol{c_2}$ is the outer product between the two coefficient vectors, D is distance matrix with d_{ij} = the Euclidean distance between the ith and jth locations that the coefficients correspond to, and λ is some weight value that determines how important distance is to the cost.

The distance matrix is constructed by reconstructing the 2- or 3-dimensional grid under which the coefficients were formed using MATLAB's `ndgrid` function, finding the points `pts`, and then computing the pairwise distance using MATLAB's `squareform(pdist(pts))`.

21 Linear assignment

The assignment problem is a classical problem in graph theory and combinatorial optimization [**?**]. Informally, it deals with finding the most optimal (least-cost) way of assigning tasks to workers. It also has an equivalent graph theory formulation. We call such an assignment problem problem *linear* if there are equal numbers of tasks and workers and the total cost for all tasks is the sum of the individual costs incurred. This is a quite generalized problem that has many applications in resource allocation, including originally in transportation theory.

21.1 Motivations

This problem first arose in transportation theory, where the goal is to assign n mines to n factories which consume the ore that the mines produce. However, the problem arises whenever there is a need to assign tasks to workers with potentially different costs for each potential task-worker pair.

We investigate this problem in order to align the discrete probability distributions of two shapes and then measure the distance between them. This is explained in section 20.

21.2 Formal specification

Let X and Y be two sets of equal size. Let C be a cost function

$$C : X \times Y \to \mathbb{R}$$

The problem is to find a bijection $f : A \to T$ such that the cost function is minimized:

$$\sum_{x \in X} C(x, f(x))$$

The linear assignment problem is often given as a cost matrix

$$C \mid c_{ij} = \text{cost of moving element } i \text{ to } j$$

Equivalently, in graph theoretic terms: given a weighted bipartite graph, find the minimum weight perfect matching.

21.3 Algorithms and runtime

Because this problem can be formulated as a linear program, it can be solved with general linear program solving algorithms (e.g. simplex and variants). However, we can solve this more efficiently by exploiting the structure of the problem.

One famous algorithm is the famous Hungarian algorithm, which was proposed by Kuhn in the 1950s and originally had time complexity of $O(n^4)$, although this was improved by Lawler in 1976 to $O(n^3)$.

We use the Jonker-Volgenant algorithm [**?**] [**?**], which is notably faster in a practical sense.

22 Shape L'Ane Rouge results

There were fairly promising results for our multiresolution extension of the Shape L'Ane Rouge paradigm. We first validated the approach visually by choosing particular lambda values and reconstructing the shape using the permutation created from linear assignment.

Then we validated on the MPEG7 dataset. Our actual retrieval results weren't as high as we were hoping, which may be due to implementation details such as optimization of the lambda parameter. Improvement on retrieval results were negligible to negative.

Part VI
Conclusion

Shape retrieval, comparison, and classification is a significant problem that has far-ranging applications due to the problem's fundamental nature in the field of computer vision. In this paper, we provide a brief exposition of the wavelet density estimation framework, including the theoretical foundation behind wavelets. We then provide several contributions to the framework of wavelet density estimation-based shape retrieval, including a speedup of the density estimation code by two orders of magnitude for 2D shapes in a multiresolution wavelet basis, an improved shape

Figure 7: Shape L'Ane Rouge warping over several lambda

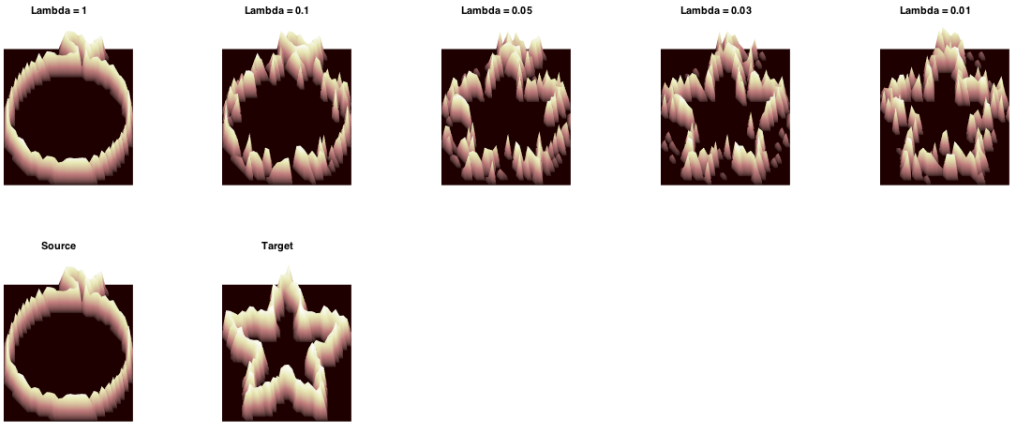

Figure 8: Supervised hierarchy results

λ values	No lin. assgn.	$\lambda = 0.036$	λ =1e-6	λ = 1e-8
Accuracy (bullseye metric)	80.2%	80.0%	70.9%	73.2%

similarity metric using linear assignment and multiresolution wavelets,, hierarchical clustering algorithms on high-dimensional unit hypersphere (with algorithmic time complexity proofs), and experimental validation across multiple datasets, with results competitive with state of the art.

23 Acknowledgements

This material is based on works/research, supported in part by the National Science Foundation under Grant No. 156035.

Part VII
Appendix

24 Implementation of random spherical data in MATLAB

By using the Circular Statistics Toolbox in MATLAB, we can construct random spherical data through exploitation of the of Von Mises (circular) distribution.

```
%-------------------------------------------------------------------------
% Function:    random_spherical_data
% Description: Draws a line on a sphere.
%
% Inputs:
%
% numClusters     - Number of clusters.
%
% numPoints       - Number of points per cluster.

% numPoints       - Kappa (concentration parameter);
%                     higher means closer clusters.

% means1, means2  - Optional vector of means.
%
%
% Outputs
%
% dataMatrix      - An nxd dataMatrix containing the random points.
% meanMatrix      - A matrix containing the means of each of the clusters.
% memMatrix       - An array of labels for each randomly generated datapoint.
%
%
% Usage: Used in hierarchical clustering on the unit hypersphere.
%
% Authors(s):
%   Mark Moyou - markmmoyou@gmail.com
% Yixin Lin - yixin1996@gmail.com
%   Glizela Taino - glizelataino@gmail.com
%
% Date: Monday 6th June, 2016 (2:34pm)
%
% Affiliation: Florida Institute of Technology. Information
%              Characterization and Exploitation Laboratory.
%              http://research2.fit.edu/ice/
% -------------------------------------------------------------------------



function [dataMatrix, meanMatrix, memMatrix] = random_spherical_data(...
  numClusters, numPoints, kappa, means1, means2)

if nargin <= 3
  means1 = nan(numClusters); means2 = nan(numClusters);
  for i = 1:numClusters
    means1(i) = rand_angle();
    means2(i) = rand_angle();
  end
end

if length(means1) ~= numClusters || length(means2) ~= numClusters
```

```
  error('Means vectors are not the right length!');
end

dataMatrix = []; meanMatrix = []; memMatrix = [];
for i = 1:numClusters
    mean1 = means1(i); mean2 = means2(i);
    % Create random angles using the von Mises distributions
    angles = [circ_vmrnd(mean1, kappa, numPoints) circ_vmrnd(mean2, kappa, numPoints)];
    % Convert to Cartesian coordinates
    [x ,y, z] = sph2cart(angles(:,1), angles(:, 2), 1);
    [muX, muY, muZ] = sph2cart(mean1, mean2, 1);

    meanMatrix = [meanMatrix; muX, muY, muZ];
    dataMatrix = [dataMatrix; [x y z]];
    memMatrix = [memMatrix; (i + zeros(numPoints, 1))];
end

end
```