2007-04-24

Manifold Sculpting

Michael S. Gashler
Brigham Young University - Provo

Manifold Sculpting

by

Mike Gashler

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Mike Gashler

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Tony Martinez, Chair
Date	Dan Ventura
Date	Robert P. Burton

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Mike Gashler in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Tony Martinez
Chair, Graduate Committee

Accepted for the Department

Parris K. Egbert
Graduate Coordinator

Accepted for the College

Thomas W. Sederberg
Associate Dean, College of
Physical and Mathematical Sciences

ABSTRACT

Manifold Sculpting

Mike Gashler

Department of Computer Science

Master of Science

Manifold learning algorithms have been shown to be useful for many applications of numerical analysis. Unfortunately, existing algorithms often produce noisy results, do not scale well, and are unable to benefit from prior knowledge about the expected results. We propose a new algorithm that iteratively discovers manifolds by preserving the local structure among neighboring data points while scaling down the values in unwanted dimensions. This algorithm produces less noisy results than existing algorithms, and it scales better when the number of data points is much larger than the number of dimensions. Additionally, this algorithm is able to benefit from existing knowledge by operating in a semi-supervised manner.

ACKNOWLEDGMENTS

Assistance and contributions from the following individuals is greatly appreciated: Tony Martinez, Dan Ventura, Desiré Gashler, Vic Bunderson, Doran K. Wilde, and Cathleen Wilde.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Large dimensionality is a significant problem for many machine learning algorithms (1). Dimensionality reduction algorithms address this issue by projecting data into fewer dimensions while attempting to preserve as much of the informational content in the data as possible.

Dimensionality reduction is a two-step process: 1) Transform the data so that more information will survive the projection, and 2) project the data into fewer dimensions. The more relationships between data points that the transformation step is required to preserve, the less flexibility it will have to position the points in a manner that will cause information to survive the projection step. Due to this inverse relationship, dimensionality reduction algorithms must seek a balance that preserves information in the transformation without losing it in the projection. The key to finding the right balance is to identify where the majority of the information lies.

Nonlinear dimensionality reduction (NLDR) algorithms seek this balance by assuming that the relationships between neighboring points contain more informational content than the relationships between distant points. Although non-linear transformations have more potential than do linear transformations to lose information in the structure of the data, they also have more potential to position the data to cause more information to survive the projection. In this process, NLDR algorithms expose patterns and structures of lower dimensionality (manifolds) that exist in the original data. NLDR algorithms, or manifold learning algorithms, have potential to

make the high-level concepts embedded in multidimensional data accessible to both humans and machines.

This paper introduces a new algorithm for manifold learning called *Manifold Sculpting*, which discovers manifolds through a process of progressive refinement. The algorithm has several unique characteristics including the ability to extend the transformation of other manifold learning algorithms, and the ability to operate in a semi-supervised manner. Section 1.1 discusses some of the work that has been done in manifold learning. Section 2 (page 5) describes the Manifold Sculpting algorithm in detail. Section 3 (page 15) reports the results of several experiments that compare the capabilities of Manifold Sculpting with similar manifold learning algorithms. Section 4 (page 35) mentions some of the ongoing research with Manifold Sculpting, and discusses some areas with much potential for further research.

1.1 Related Work

Early NLDR algorithms, such as Nonlinear Multidimensional Scaling (2) and Nonlinear Mapping (3), have proven useful, but are unable to handle high nonlinearities in the data. Curvilinear Component Analysis (CCA) (4) uses a neural technique to solve many of these problems, and Curvilinear Distance Analysis (CDA) (5) takes it a step further by using distance on the manifold surface as a metric for preservation, but both algorithms are computationally demanding, and suffer from problems with local minima. Isomap (6) uses the same metric as CDA, but solves for the embedding into fewer dimensions with an algebraic technique. This approach is faster, but still computationally expensive as it requires solving for the eigenvectors of a large dense matrix, and has difficulty with poorly sampled areas of the manifold. (See Figure 1.A.) Locally Linear Embedding (LLE) (7) is able to perform a similar computation using a sparse matrix by using a metric that measures only relationships between vectors in local neighborhoods. Unfortunately it produces distorted results when the sample density is non-uniform. (See Figure 1.B.) L-Isomap, an improvement to the Isomap algorithm, was later presented that uses landmarks to reduce the amount of necessary computation (8). (See Figure 1.C.) Many other NLDR algorithms have been

presented, including Kernel Principal Component Analysis (9), Laplacian Eigenmaps (10), Manifold Charting (11), Manifold Parzen Windows (12), Hessian LLE (13), and others (14; 15; 16). Hessian LLE preserves the manifold structure better than the other algorithms but is, unfortunately, very computationally expensive. (See Figure 1.D.).

In contrast with these algorithms, *Manifold Sculpting* is robust to sampling issues and still produces very accurate results. Additionally, it can be used in either an unsupervised or semi-supervised manner. This algorithm iteratively transforms data by balancing two opposing heuristics, one that scales information out of unwanted dimensions, and one that preserves local structure in the data. Experimental results show that this technique preserves information into fewer dimensions with more precision than existing manifold learning algorithms. (See Figure 1.E.)

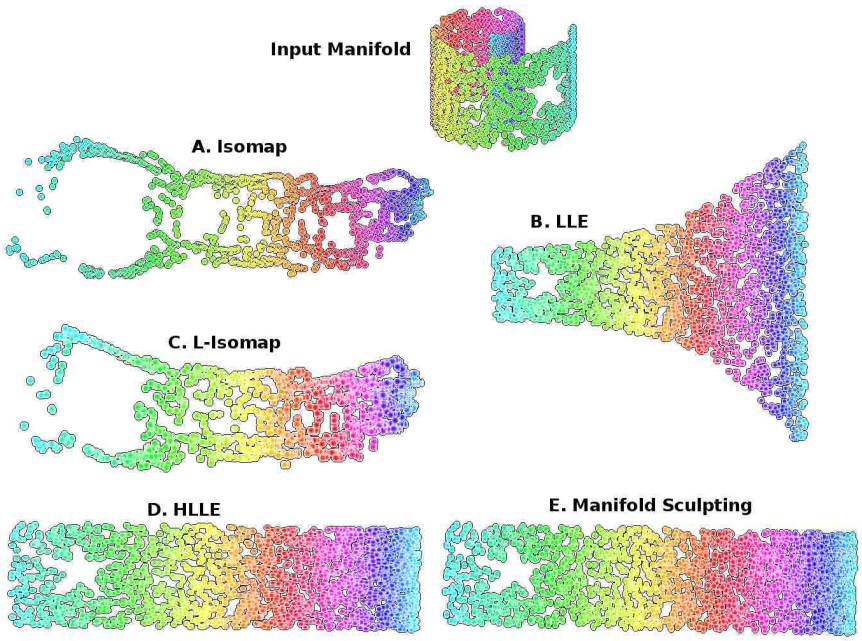


Figure 1.1: Comparison of several manifold learners on a Swiss Roll manifold. Color is used to indicate how points in the results correspond to points on the manifold. Isomap and L-Isomap have trouble with sampling holes. LLE has trouble with changes in sample density. HLLE and Manifold Sculpting both handle these problems well.

Chapter 2

The Manifold Sculpting Algorithm

An overview of the Manifold Sculpting algorithm is given in Table 2.1. To provide more depth, pseudo code is given in Figure 2.2. Let

P	$\equiv$ The set of all data points represented as vectors in $\Re^n$.
k	$\equiv$ The number of neighbors.
N	$\equiv$ A $\|P\| \times k$ matrix such that $n_{ij} \in N_i$ is the j^{th} neighbor of point p_i where $p_i \in P$.
D	$\equiv$ The set of dimensions in which the input data is represented.
$D_{preserved}$	$\equiv$ The set of dimensions preserved by the projection.
D_{scaled}	$\equiv$ The set of dimensions we throw out int the projection step. $(D = D_{preserved} \cup D_{scaled})$.
σ	$\equiv$ A constant scaling factor.
η	$\equiv$ The step size.

2.1 Steps 1 and 2: Compute local relationships

For each data point p_i in P, find the k-nearest neighbors N_i. For each j (where $0 <= j < k$) compute the Euclidean distance δ_{ij} between p_i and each $n_{ij} \in N_i$. Also compute the angle θ_{ij} formed by the two line segments (p_i to n_{ij}) and (n_{ij} to m_{ij}), where m_{ij} is the most colinear neighbor of n_{ij} with p_i. (See Figure 2.1.) The most colinear neighbor is the neighbor point that forms the angle closest to π. The values

Table 2.1: A high-level overview of the Manifold Sculpting algorithm.

1	Find the k nearest neighbors of each point.
2	Compute a set of relationships between neighboring data points.
3	Optionally align axes with principal components.
4	While the stopping criteria has not been met...
	a. Scale the data in the non-preserved dimensions with a constant factor σ, where $\sigma < 1$.
	b. Restore the relationships computed in step 2 by adjusting the data in the preserved dimensions.
5	Project the data.

of δ and θ are the relationships that the algorithm will attempt to preserve during transformation. The global average distance between all the neighbors of all points δ_{ave} is also computed.

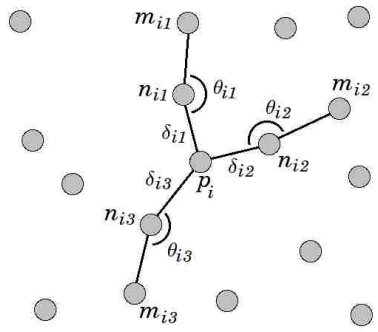


Figure 2.1: δ and θ define the relationships that Manifold Sculpting attempts to preserve.

2.2 Step 3: Optionally preprocess the data

The data may optionally be preprocessed with the transformation step of Principal Component Analysis (PCA). Manifold Sculpting will work without this step; however, preprocessing can result in significantly faster convergence. To the extent that there is a linear component in the manifold, this will move the information in the

function ManifoldSculpting

1) for each $p_i \in P$:
   $N_i \leftarrow$ the k nearest neighbors of p_i
2) for i from 1 to $|P|$:
   for j from 1 to k:
   $\delta_{ij} \leftarrow \text{distance}(p_i, n_{ij})$
   $m_{ij} \leftarrow \text{argmax}_{0<l\leq k}\, \text{correlation}(n_{ij} - p_i, n_{jl} - n_{ij})$
   $c_{ij} \leftarrow \text{correlation}(n_{ij} - p_i, m_{ij} - n_{ij})$
   (Note: correlation computes the cosine
   of the angle between two vectors)
   $\delta_{ave} \leftarrow \frac{1}{k|P|} \sum_{0<i\leq|P|,0<j\leq k} \delta_{ij}$
   $\eta \leftarrow \delta_{ave}$
3) Call AlignAxesWithPrincipalComponents(P)
4) Until the stopping criteria is met, do:
   a) for each $p_i \in P$:
   for each $j \in D_{scaled}$:
   $p_{ij} \leftarrow p_{ij}\sigma$
   while $\frac{1}{k|P|} \sum_{0<i\leq|P|,0<j\leq k} \delta_{ij} < \delta_{ave}$:
   for each $p_i \in P$:
   for each $j \in D_{preserved}$:
   $p_{ij} \leftarrow p_{ij}/\sigma$
   b) add p_r to a queue, where p_r is a random element of P
   $steps \leftarrow 0$
   while the queue is not empty, do:
   pop p_{cur} from the queue
   if p_{cur} has not been adjusted yet:
   $steps \leftarrow steps + \text{AdjustPoints}(p_{cur}, \eta)$
   add each neighbor of p_{cur} to the queue
   if $steps \geq |P|, \eta \leftarrow \eta/.9$ else $\eta \leftarrow \eta * .9$
5) Project the data by dropping D_{scaled}

Figure 2.2: Pseudo code for the Manifold Sculpting algorithm. Note that pseudo code for the AlignAxesWithPrincipalComponents function is given in Figure 2.3 (page 9), and pseudo code for the AdjustPoints function is given in Figure 2.5 (page 12)

data into as few dimensions as possible, thus leaving less work to be done in step 4 (which handles the non-linear component). Although small generated manifolds (like the Swiss Roll manifold) may benefit very little from this step, natural manifolds in much larger dimensional space are very likely to have a significant linear component.

Efficient PCA algorithms generally compute only the first few principal components and combine the transformation step with the projection step. Since we want only the transformation portion of PCA, and since all the dimensions of data are needed to maintain stability in step 4, these algorithms are not suitable for this purpose. We therefore present a new algorithm that aligns data with only the first few principal components, but preserves correct values in all dimensions. We start with the standard set of basis vectors: $\{\hat{i}, \hat{j}, \hat{k}, \ldots\}$. For each $d_i \in D_{preserved}$, we rotate the set of basis vectors to align the i^{th} basis vector with the i^{th} principal component (which we compute with a variation of the algorithm presented in (17)). Then we remaps all the data into the new vector space. Pseudo-code for this algorithm is given in Figure 2.3.

2.3 Step 4: Transform the data

The data is iteratively transformed as shown in Figure 2.4. This transformation continues until some stopping criterion has been met. One effective technique is to stop when the sum change of all points during the current iteration falls below a threshold. The best stopping criteria depend on the desired quality of results – if precision is important, the algorithm may iterate longer; if speed is important it may stop earlier.

2.3.1 Step 4a: Scale the values in D_{scaled}.

All the values in D_{scaled} are scaled down by multiplying by a constant factor σ, where $0 < \sigma < 1$ ($\sigma = 0.99$ was used in this paper). Over time, the values in D_{scaled} will converge to 0. When D_{scaled} is dropped by the projection (step 5), there will be very little informational content left in these dimensions. The values in $D_{preserved}$ are scaled up to keep the average neighbor distance equal to δ_{ave}.

function AlignAxesWithPrincipalComponents(P)

(center data around origin)

$\mu \leftarrow mean(P)$
for each $p_i \in P$:
    $p_i \leftarrow p_i - \mu$
$Q \leftarrow P$
$G \leftarrow \{\hat{i}, \hat{j}, \hat{k}, \ldots\}$ such that $|G| = |D|$
for k from 1 to $|D_{preserved}|$

(find the next principal component)

    $c \leftarrow$ a random vector with $|D|$ elements
    do 10 times:
        $t \leftarrow$ zero vector with $|D|$ elements
        for each $q_i \in Q$:
            $t \leftarrow t + (q_i \cdot c)q_i$
        $c \leftarrow \frac{t}{|t|}$

(subtract out the component)

    for each $q_i \in Q$:
        $q_i \leftarrow q_i - (c \cdot q_i)c$

(find a safe plane of rotation)

    $a \leftarrow G_k$
    $b \leftarrow \frac{c-(a \cdot c)a}{|c-(a \cdot c)a|}$

(rotate the basis vectors)

    $\phi \leftarrow \arctan(\frac{b \cdot c}{a \cdot c})$
    for j from k to $|D|$
        $u \leftarrow a \cdot G_j$
        $v \leftarrow b \cdot G_j$
        $G_j \leftarrow G_j - ua$
        $G_j \leftarrow G_j - vb$
        $r \leftarrow \sqrt{u^2 + v^2}$
        $\theta \leftarrow \arctan(\frac{v}{u})$
        $u \leftarrow r\cos(\theta + \phi)$
        $v \leftarrow r\sin(\theta + \phi)$
        $G_j \leftarrow G_j + ua$
        $G_j \leftarrow G_j + vb$

(transform the data)

for each $p_i \in P$:
    for j from 1 to $|D|$:
        $p_{ij} \leftarrow p_i \cdot G_j + \mu_j$

Figure 2.3: Pseudo code for the AlignAxesWithPrincipalComponents function. This performs nearly the same function as the axis rotation step of principal component analysis, except this algorithm only aligns with the first $|D_{preserved}|$ principal components, and it is much more efficient when the number of dimensions is large.

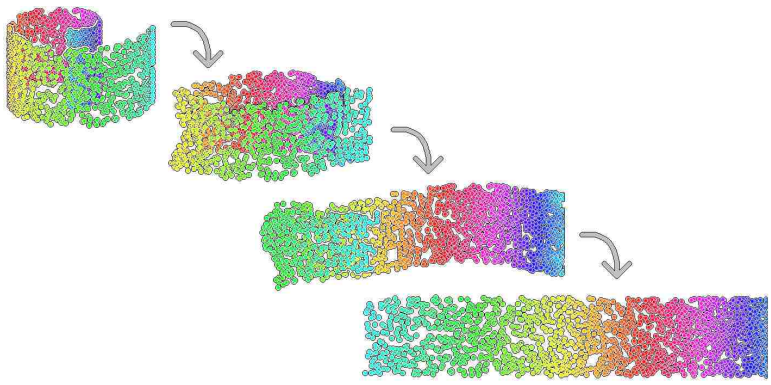


Figure 2.4: A Swiss Roll manifold shown at four stages of the iterative transformation. This experiment was performed with 2000 data points, $k = 14$, $\sigma = 0.99$, and iterations = 300.

2.3.2 Step 4b: Restore original relationships.

For each $p_i \in P$, the values in $D_{preserved}$ are adjusted to recover the relationships that are distorted by scaling. Intuitively, this step simulates tension on the manifold surface. A heuristic error value is used to evaluate the current relationships among data points relative to the original relationships:

$$\epsilon_{p_i} = \sum_{j=0}^{k} \left(\frac{\delta_{ij_0} - \delta_{ij}}{2\delta_{ave}} \right)^2 + \left(\frac{\max(0, \theta_{ij_0} - \theta_{ij})}{\pi} \right)^2 \tag{2.1}$$

where δ_{ij} is the current distance to $n_i j$, δ_{ij_0} is the original distance to $n_i j$ measured in step 2, θ_{ij} is the current angle, θ_{ij_0} is the original angle measured in step 2 These denominator values were chosen because the value of the angle term can range from 0 to π, and the value of the distance term will tend to have a mean of about δ_{ave} with some variance in both directions. These value therefore normalize the terms of the equation such that they each carry approximately equal weight. We adjust the values in $D_{preserved}$ for each point in order to minimize this error value. Unfortunately the equation for the true gradient of the error surface defined by this heuristic is complex, and is in $O(|D|^3)$. We therefore use the simple hill-climbing technique of adjusting in each dimension in the direction that yields improvement. Pseudo-code for this

technique is given in Figure 2.5.

Many algorithms that attempt to minimize an error heuristic over a non-convex error surface may be compared to rolling a ball down a hill and hoping that it finds its way to a globally optimal position. This algorithm, on the other hand, is more analogous to a ball following at the feet of a person walking across a trampoline. It begins at a globally optimal position (with an error value of zero), and seeks to remain in a basin while the error surface slowly changes.

At least three factors help increase the likelihood that the algorithm will be able to maintain a globally optimal position: First, the PCA pre-processing step often tends to move the whole system to a state somewhat close to the final state. Even if a local minimum exists so close to the final optimal state, it may have a sufficiently small error as to be acceptable. Second, every point has a unique error surface. Even if one point becomes temporarily stuck in a local minimum, its neighbors are likely to pull it out, or change the topology of its error surface when their values are adjusted. Very particular conditions are necessary for every point to simultaneously find a local minimum. Third, as long as it stays fairly close to the currently optimal state while the values in D_{scaled} slowly change, it unlikely for the error surface to be able to change in a manner that permanently separates it that optimal state. (This is why all the dimensions need to be preserved in the PCA pre-processing step.) And perhaps most significantly, our experiments show that Manifold Sculpting generally tends to converge to very good results.

To speed convergence, we adjust the step size η to keep the total number of steps taken in each pass approximately equal to the total number of data points. If the number of steps is less than the number of data points, then $\eta \leftarrow \eta * .9$, otherwise $\eta \leftarrow \eta / .9$. Experimentally this technique was found to converge significantly faster than using a constant or constantly decaying value for η. The order in which points are adjusted also has an impact on the convergence rate. Best results were obtained by employing a breadth-first neighborhood graph traversal from a randomly selected point. (A new starting point was randomly selected for each iteration.) Intuitively this may be analogous to how a person smoothes a crumpled piece of paper by starting at

an arbitrary point and smoothing outward. Higher weight is given to the component of the error contributed by neighbors that have already been adjusted in the current iteration. Thus the error equation is

$$\epsilon_{p_i} = \sum_{j=0}^{k} w_{ij} \left(\left(\frac{\delta_{ij_0} - \delta_{ij}}{2\delta_{ave}} \right)^2 + \left(\frac{\max(0, \theta_{ij_0} - \theta_{ij})}{\pi} \right)^2 \right) \tag{2.2}$$

where $w_{ij} = 1$ if n_i has not yet been adjusted in this iteration, and $w_{ij} = c$, where $c > 1$, if n_{ij} has been adjusted. Large values for c yield better performance, while small values yield greater stability. If c is very large and the number of neighbors k is small, the manifold may become twisted and therefore require additional iterations before the system will correct itself and converge. A value of $c = 10$ was used for all the experiments reported in this paper.

```
function AdjustPoints(p_cur, η)
 s ← −1
 improved ← true
 while improved = true, do:
        increment s
        improved ← false
        error ← ComputeError(p_cur)
        for each d ∈ D_preserved:
              p_cur_d ← p_cur_d + η
              if ComputeError(p_cur) > error:
                    p_cur_d ← p_cur_d − 2η
                    if ComputeError(p_cur) > error:
                          p_cur_d ← p_cur_d + η
                    else
                          improved ← true
              else
                    improved ← true
 return s
```

Figure 2.5: Pseudo code for the AdjustPoints function. Note that the ComputeError function is equation 2.2.

2.4 Step 5: Project the data

At this point D_{scaled} contains only values that are very close to zero. The data is projected by simply dropping these dimensions from the representation.

Chapter 3

Validation

We tested the properties of Manifold Sculpting with a wide variety of experiments and a range of manifolds. Section 3.1 shows the results of experiments designed to test accuracy. These results indicate that Manifold Sculpting is more accurate than Isomap, LLE, and HLLE with most problems. Section 3.1.1 reports an experiment in which accuracy is measured for each algorithm with varying number of neighbors. Section 3.1.2 reports the accuracy of each algorithm with varying sample densities. Section 3.1.3 reports accuracy with an entwined spirals manifold. Section 3.1.4 shows results with two image-based manifolds. Finally, section 3.1.5 measures accuracy with each of these algorithms using image-based manifolds with known topologies such as images of a rotating object, images from a panning camera, etc. Section 3.2 gives a complexity analysis of the Manifold Sculpting algorithm and reports results of an experiment to measure the scalability of running time. Section 3.3 reports results on an experiment using non-image-based manifolds to show that Manifold Sculpting is useful in other domains as well.

3.1 Accuracy

Figure 1.1 (page 3) shows that Manifold Sculpting appears visually to produce results of higher quality than LLE and Isomap with the Swiss Roll manifold, a common visual test for manifold learning algorithms. Quantitative analysis shows that it also yields better results than HLLE. Since the actual structure of this manifold

is known prior to using any manifold learner, we can use this prior information to quantitatively measure the accuracy of each algorithm.

3.1.1 Accuracy with varying number of neighbors.

We define a Swiss Roll in 3D space with n points (x_i, y_i, z_i) for each $0 \leq i < n$, such that $x_i = t\sin(t)$, y_i is a random number $-6 \leq y_i < 6$, and $z_i = t\cos(t)$, where $t = 8i/n + 2$. In 2D manifold coordinates, the point is (u_i, v_i), such that $u_i = \frac{\sinh^{-1}(t)+t\sqrt{t^2+1}}{2}$ and $v_i = y_i$.

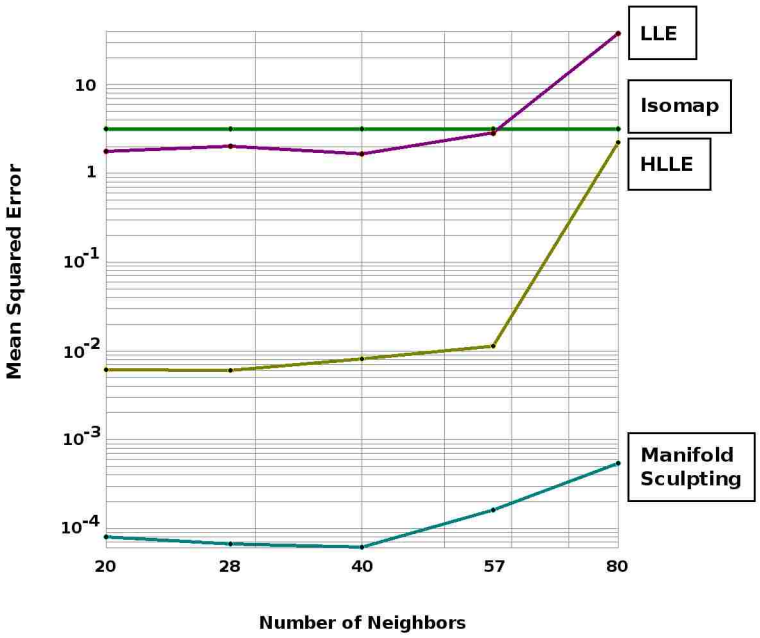


Figure 3.1: The mean squared error of four algorithms with a Swiss Roll manifold using a varying number of neighbors k. In this example, when $k > 57$, neighbor paths cut across the manifold. Isomap is more robust to this problem than other algorithms, but HLLE and Manifold Sculpting still yield better results. These results are shown on a logarithmic scale.

We created a Swiss Roll with 2000 data points and reduced the dimensionality to 2 with each of four algorithms. Next we tested how well this output aligns with the

expected output values by measuring the mean squared distance from each point to its expected value. (We rotated, scaled, and translated the output values as required to obtain the minimum possible error measurement for each algorithm.) Figure 3.1 shows the average squared Euclidean distance between each transformed point and its expected value. These results are consistent with a qualitative assessment of Figure 1.1. Results are shown with a varying number of neighbors k. In this example, when $k = 57$, local neighborhoods begin to cut across the manifold. Isomap is more robust to this problem than other algorithms, but HLLE and Manifold Sculpting still yield better results. The additional sub-pixel precision that can be obtained with Manifold Sculpting may be superfluous for visualizations, but can be critical for applications that rely on the precision of the data.

3.1.2 Accuracy with varying sample densities.

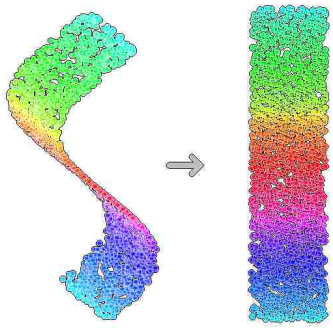
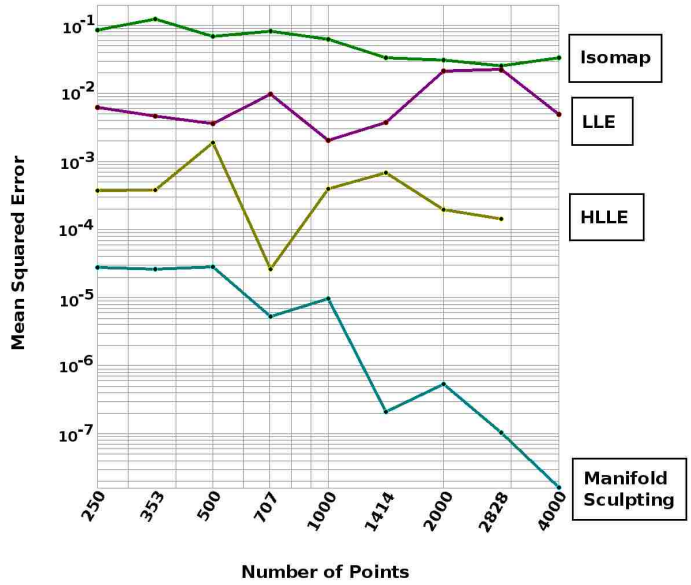
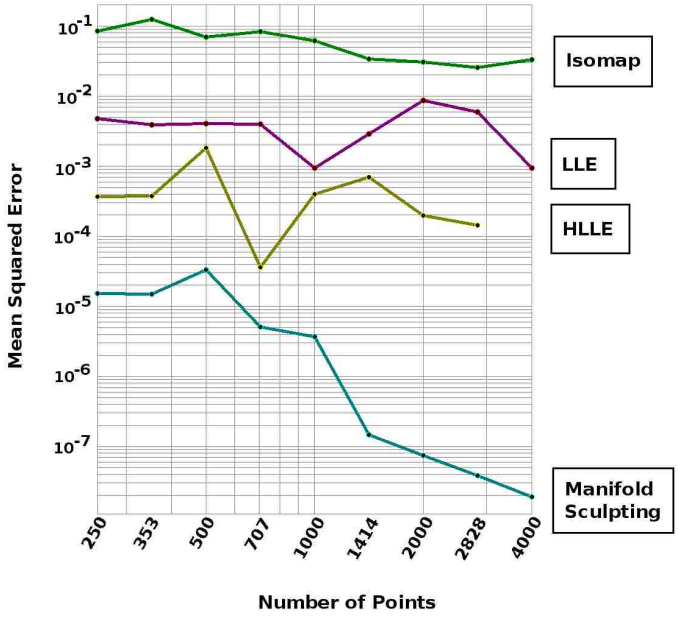


Figure 3.2: A visualization of an S-Curve manifold.

A similar experiment was performed with an S-Curve manifold as shown in Figure 3.2. We defined the S-Curve points in 3D space with n points (x_i, y_i, z_i) for each $0 \leq i < n$, such that $x_i = t$, $y_i = sin(t)$, and z_i is a random number $0 \leq z_i < 2$, where $t = \frac{(2.2i-0.1)\pi}{n}$. In 2D manifold coordinates, the point is (u_i, v_i), such that $u_i = \int_0^t \left(\sqrt{\cos^2(w) + 1}\right) dw$ and $v_i = y_i$.



(a) $k = 14$



(b) $k = 20$

Figure 3.3: The mean squared error of points from an S-Curve manifold for four algorithms with a varying number of data points. Manifold Sculpting shows a trend of increasing accuracy with an increasing number of points. The experiment shown in (a) was performed with 14 neighbors, while the experiment shown in (b) was performed with 20 neighbors. Results are not significantly influenced by this difference. Results are shown on a logarithmic scale. Due to the demanding memory requirements of the HLLE algorithm, we only tested HLLE with up to 2828 data points.

Figure 3.3 shows the mean squared error of the transformed points from their expected values using the same regression technique described for the experiment with the Swiss Roll problem. We varied the sampling density to show how this affects each algorithm. A trend can be observed in this data that as the number of sample points increases, the quality of results from Manifold Sculpting also increases. This trend does not appear in the results from other algorithms. Results are shown on a logarithmic scale, using 14 and 20 neighbors.

One drawback to the Manifold Sculpting algorithm is that convergence may take longer when the value for k is too small. This experiment was also performed with 6 neighbors, but Manifold Sculpting did not always converge within a reasonable time when so few neighbors were used. The other three algorithms do not have this limitation, but the quality of their results still tend to be poor when very few neighbors are used.

3.1.3 Accuracy with entwined spirals manifold.

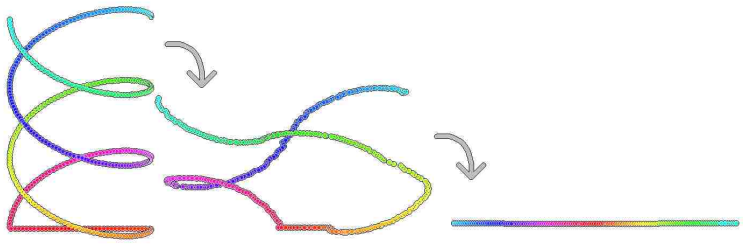


Figure 3.4: A visualization of an Entwined Spirals manifold.

A test was also performed with an Entwined Spirals manifold as shown in Figure 3.4. In this case, Isomap was able to produce better results than Manifold Sculpting (see Figure 3.5), even though Isomap yielded the worst accuracy in previous problems. This can be attributed to the nature of the Isomap algorithm. In cases where the manifold has an intrinsic dimensionality of exactly 1, a path from neighbor

to neighbor provides an accurate estimate of isolinear distance. Thus an algorithm that seeks to globally optimize isolinear distances will be less susceptible to the noise from cutting across local corners. When the intrinsic dimensionality is higher than 1, however, paths that follow from neighbor to neighbor produce a zig-zag pattern that introduces excessive noise into the isolinear distance measurement. In these cases, preserving local neighborhood relationships with precision yields better overall results than globally optimizing an error-prone metric. Consistent with this intuition, Isomap is the closest competitor to Manifold Sculpting in other experiments that involved a manifold with a single intrinsic dimension, and yields the poorest results of the four algorithms when the intrinsic dimensionality is larger than one.

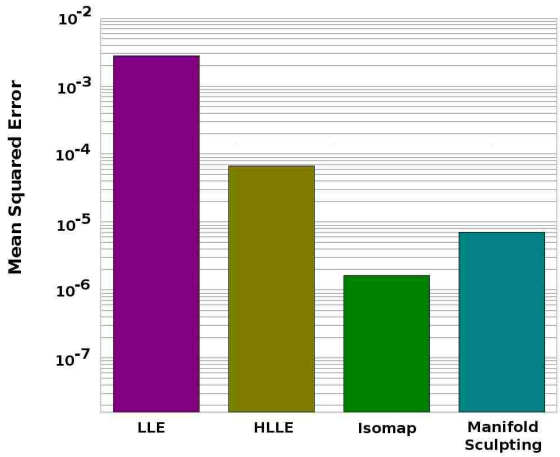


Figure 3.5: Mean squared error for four algorithms with an Entwined Spirals manifold.

3.1.4 Image-based manifolds.

The accuracy of Manifold Sculpting is not limited to generated manifolds in three dimensional space. We also performed several experiments to demonstrate its capabilities with real-world problems. Figure 3.6 shows several frames from a video sequence of a person turning his head while gradually smiling. Each image

Figure 3.6: Images of a face reduced by Manifold Sculpting into a single dimension. The values are are shown here on four wrapped lines in order to fit the page. The original image is shown above each point.

was encoded as a vector of $1,634$ pixel intensity values. No single pixel contained enough information to characterize a frame according to the high-level concept of facial position, but this concept was effectively encoded in multi-dimensional space. This data was then reduced to a single dimension. (Results are shown on four separate lines in order to fit the page.) The one preserved dimension could then characterize each frame according to the high-level concepts that were previously encoded in many dimensions. The dot below each image corresponds to the single-dimensional value in the preserved dimension for that image. In this case, the ordering of every frame was consistent with the ordering in the video sequence.

Figure 3.7 shows eleven images of a hand. These images were encoded as multidimensional vectors in the same manner. The high-level concept of "hand openness" was preserved into a single dimension. In addition to showing that Manifold Sculpting can work with real-world data, this experiment also shows the robustness of the algorithm to poorly sampled manifolds because these eleven images were the only images used for this experiment. Again, the ordering of every frame was consistent with the ordering in the video sequence.

3.1.5 Controlled manifold topologies.

Figure 3.8 shows a comparison of results obtained from a manifold generated by translating an image over a background of random noise. Nine of the 400 input images are shown as a sample, and results with each algorithm are shown as a mesh. Each vertex is placed at a position corresponding to the two values obtained from one of the 400 images. For increased visibility of the inherent structure, the vertexes are connected with their nearest input space neighbors. Because two variables (horizontal position and vertical position) were used to generate the dataset, this data creates a manifold with an intrinsic dimensionality of two in a space with an extrinsic dimensionality of 2,401 (the total number of pixels in each image). Because the background is random, the average distance between neighboring points in the input space is uniform, so the ideal result is known to be a square. Note that the distortions produced by Manifold Sculpting tend to be local in nature, while the distortions produced by other algorithms tend to be more global. This explains why the results from Manifold Sculpting tend to fit the ideal results with much lower total error (as shown in Figure 3.1 and Figure 3.3). Perhaps more significantly, it also tends to keep the intrinsic variables in the dataset more linearly separable. This is particularly important when the dimensionality reduction is used as a pre-processing step for a supervised learning algorithm.

For contrast, Figure 3.9 shows the results of using Manifold Sculpting to reduce the dimensionality of a manifold generated by sliding a window over a larger picture of the Mona Lisa. In this case the final results are not square because some parts of the image change at a different rate than other parts. Because the manifold learner has no knowledge of this fact, it operates with the assumption that the ratio between

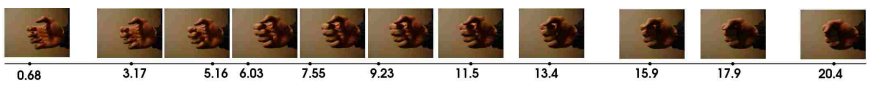


Figure 3.7: Images of a hand reduced to a single dimension. The original image is shown above each point.

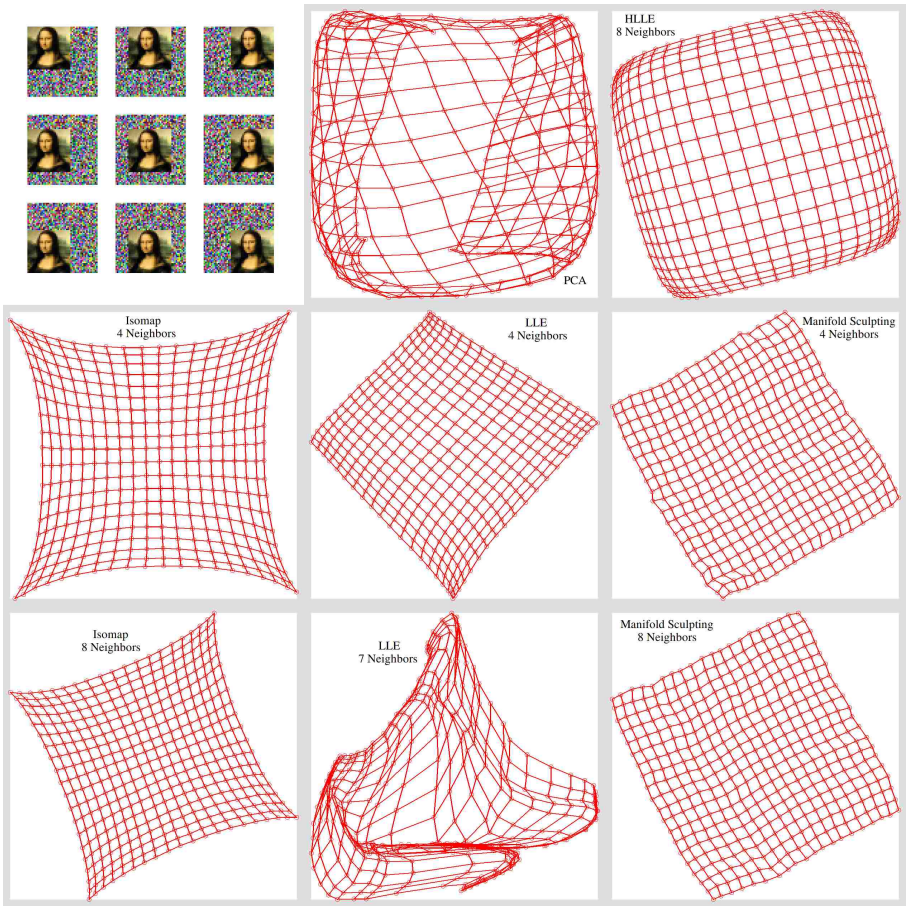


Figure 3.8: A comparison of results with a manifold generated by translating an image over a background of noise. Manifold Sculpting tends to produce less global distortion, while other algorithms tend to produce less local distortion. Each point represents an image. HLLE did not work with 4 neighbors, and LLE did not work with 8 neighbors.

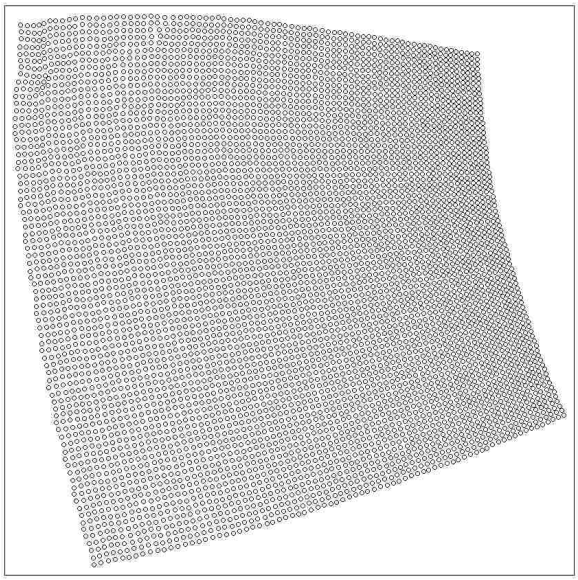


Figure 3.9: Manifold Sculpting was used to reduce the dimensionality of a manifold generated from a collection of 5776 images each with 1225 pixels. These images were generated by sliding a window over a larger picture of the Mona Lisa. In this case the results are correctly not square because the sliding window creates a bigger change in input distance when sliding over some areas than others.

extrinsic and intrinsic distance in one local neighborhood of the data is equal to the ratio between extrinsic and intrinsic distance in every other local neighborhood. The "squareness" of the shape represented by these results shows the extent to which that assumption is valid for this data set. In order to fully recover the intrinsic variables of an arbitrary data set, a manifold learner would additionally need a mapping that would enable it to determine the intrinsic distances from the extrinsic values. But even with the assumption of constant distance ratios, Manifold Sculpting is able to recover the intrinsic variables of this dataset with a high level of separation and obvious continuity. This test used 5776 images each with 1225 pixels. All the other algorithms required an unreasonable amount of memory to operate on a dataset of this size, so results are only shown for Manifold Sculpting.

Unfortunately, the manifold structure represented by most real-world problems is not known *a priori*. The accuracy of a manifold learner, however, can still

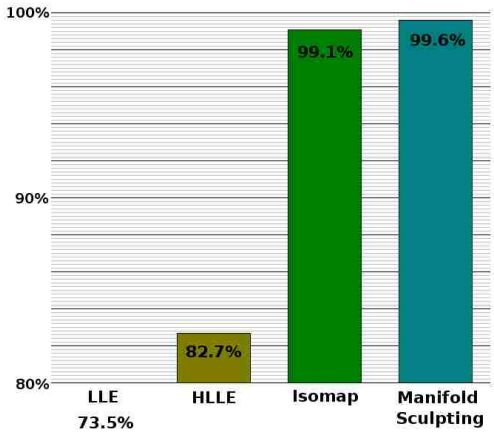
be estimated when the problem involves a video sequence by simply counting the proportion of frames that are sorted into the same order as the video sequence. We selected several video sequences to show various types of manifolds and measured the accuracy of each manifold learning algorithm.

Figure 3.10(a) shows a sample of frames from a video sequence of a rotating stuffed animal. The manifold produced by the vector encoding of such a video sequence can be characterized with a few observations. Notice that the elements in each vector that represent background pixels will remain relatively constant across the manifold. Elements that represent pixels on an outer edge of the depicted object will change with seemingly random values as new colors rotate into view. Elements that represent pixels that are interior to the object will typically aquire subsequent values from their neighbors, resulting in a change in manifold topology proportional to the change in contrast with neighboring pixels. Because this video sequence represents several possible manifold characteristics, we believe it should be serve as a good general model for testing a manifold learning algorithm. In order to verify our results, and to test the susceptibility of the algorithms to small changes in the data, we made a second video of the same scene. Results are shown in Figure 3.10(b) and Figure 3.10(c) as the percentage of frames that were correctly ordered. Notice that HLLE, Isomap, and Manifold Sculpting yield consistent results, while LLE is less stable with respect to this manifold.

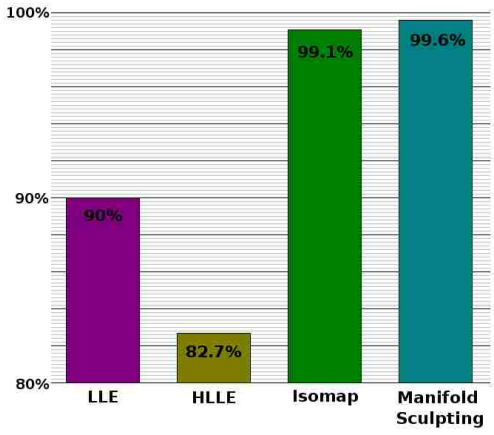
Figure 3.11(a) shows a sample of frames from a video sequence made by moving a camera down a hallway on a dolly. Notice that pixels near the center of the frame remain relatively constant, while pixels near the edges change quite rapidly. This results in a manifold that contains a wide range of topological variability. Again we made two videos of this same scene. Results are shown in Figure 3.11(b) and Figure 3.11(c). Interestingly, LLE yields consistent results with this problem, while the results from HLLE fluctuate. In one of the two cases, Isomap produced slightly better results than Manifold Sculpting.

Figure 3.12(a) shows a sample of frames from a video sequence made by panning across a scene of stuffed animals. Unlike the video sequence of the rotating fox,

(a) Video 1: Sample frames from the rotating fox movie

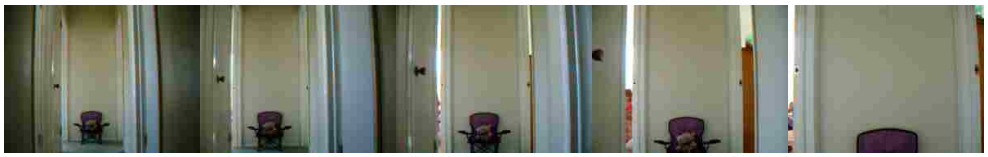


(b) Frame order accuracy 1

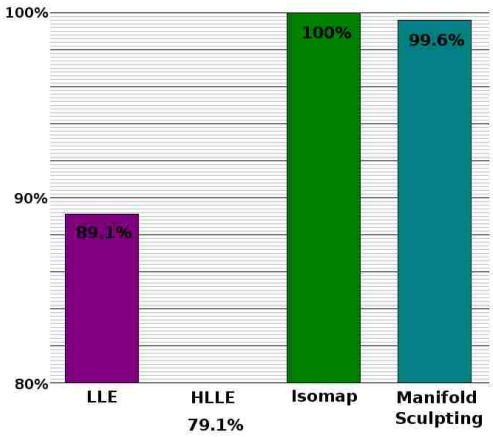


(c) Frame order accuracy 2

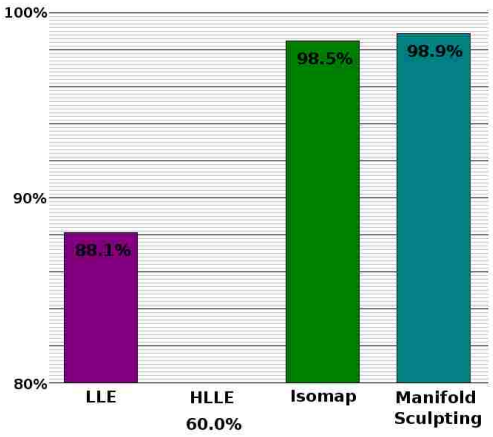
Figure 3.10: (a) A sample of frames from a rotating stuffed animal. (b) The percentage of frames correctly ordered by four manifold learning algorithms. (c) The same measurements from another filming of the same scene. In this case, LLE appears to be less stable with respect to small changes in the data.



(a) Video 2: Sample frames from the hallway movie



(b) Frame order accuracy 1

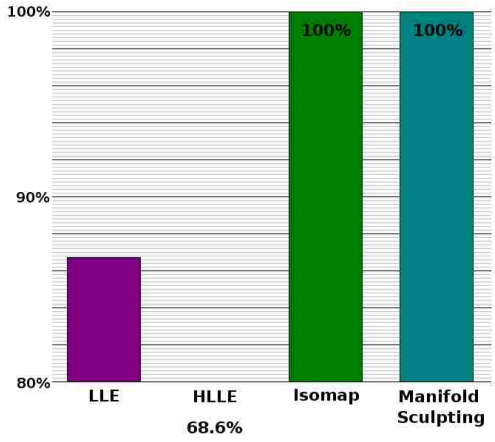


(c) Frame order accuracy 2

Figure 3.11: (a) A sample of frames from a camera travelling down a hallway on a dolly. (b) The percentage of frames correctly ordered by four manifold learning algorithms. (c) The same measurements from another filming of the same scene. Isomap and Manifold Sculpting both yield good results with this problem.

there are no background pixels that remain constant. Instead, every pixel obtains its subsequent values from its neighbors. Despite this potential to make a more complex manifold, both Isomap and Manifold Sculpting are able to sort all of the frames in this sequence into the correct order.

(a) Video 3: Sample frames from the panning movie



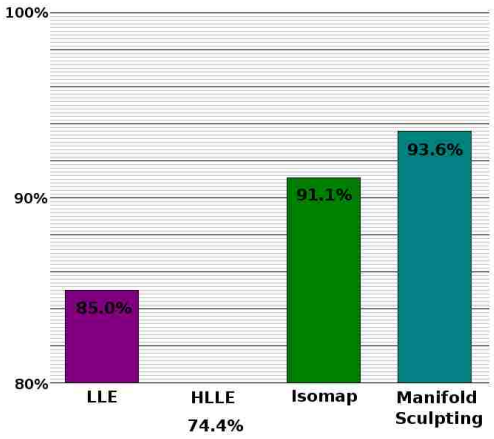
(b) Frame order accuracy

Figure 3.12: (a) A sample of frames from a video sequence that pans across a scene of stuffed animals. (b) The percentage of frames correctly ordered by four manifold learning algorithms. Both Isomap and Manifold Sculpting get all of the frames correct in this case.

Figure 3.13(a) shows a sample of frames from a video sequence of another rotating stuffed animal. Even though the transformation (rotation) is the same as that in the video sequence with the fox, the high-contrast texture of this stuffed animal creates a much more difficult manifold. As the black spots shift across the pixels, a manifold is created that swings wildly in the respective dimensions. Due to

the large hills and valleys in the topology of this manifold, the nearest neighbors of a vector frequently create paths that cut across the manifold. These results, therefore, show how rubust each algorithm is to this situation.

(a) Video 4: Sample frames from another movie of a rotating stuffed animal



(b) Frame order accuracy

Figure 3.13: (a) a sample of frames from video sequence of a rotating stuffed animal. The high-contrast texture on this toy creates a manifold with large topological hills and valleys. (b) the percentage of frames correctly ordered by four manifold learning algorithms.

3.2 Computational Complexity

The computational complexity of the algorithm can be broken down as follows:

Step 1: $O(dkn\log(n))$ where $n = |P|$, and $d = |D|$

Step 2: $O(k^2n)$

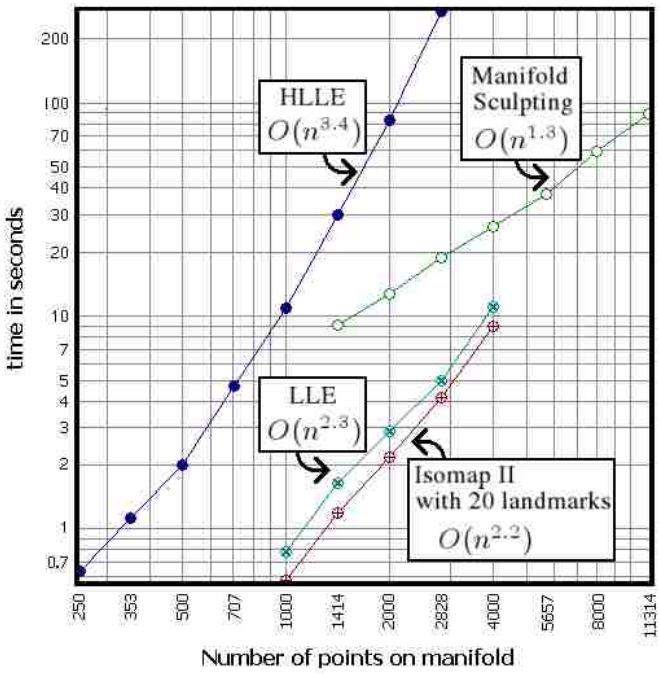


Figure 3.14: The performance of Manifold Sculpting does not degrade as much with additional data points as do other algorithms. Results are shown on a logarithmic scale. Only the values used to compute scalability are shown. Values for fewer than 1000 points were not used for Manifold Sculpting because it did not consistently converge with so few points. Values with more than 4000 points were not used with LLE or Isomap due to their demanding memory requirements. Values with more than 2828 points were not used with HLLE because it took so long.

Step 3: $O(d^2n)$

Step 4: $O(jdkn)$ where j is the number of iterations necessary to converge

Step 5: $O(dn)$

If k is assumed to be a small constant, then the overall complexity is bounded by $O(jdn\log(n))$. The value j seems to remain relatively constant when a given manifold is sampled with varying densities. Unfortunately, a precise value for j is difficult to estimate without actually performing the algorithm. Its value is related to the complexity of the manifold. It is believed that this value remains nearly constant because once the major features of a manifold are sampled, additional samples tend to have little effect on the topological complexity. However, a solid theoretical foundation for evaluating manifold complexity remains to be discovered. Consequently, experimental results provide a better framework for evaluating the computational complexity of the Manifold Sculpting algorithm.

The scalability of Manifold Sculpting was compared experimentally with LLE, L-Isomap, and HLLE using a Swiss Roll manifold sampled with a variable number of points. Results are shown in Figure 3.14. (We used L-Isomap for comparing runtime performance because it scales better than regular Isomap. We used regular Isomap when comparing accuracy, because it yields more accurate results than L-Isomap.) Unfortunately, the scalability shown in Figure 3.14 is only valid when the number of input dimensions is already small. In higher dimensional space, Step 1 becomes $O(dkn^2)$ because a more efficient method for finding neighbors is not known. Neighbor finding can be done efficiently with a kd-tree as long as the input space is well sampled, but when the number of dimensions is large, an exponentially larger dataset is necessary for this technique to remain efficient.

To compute scalability, we generated a Swiss Roll manifold as defined in section 3.1.1 with a range of sample densities. We measured the running time with each algorithm, and then used least squares to regress the function $t = an^b + c$ to fit the experimental data. (t represents time, and n represents the number of sampled points.) The value of b was then reported as the estimated order of scalability for

the algorithm. For Manifold Sculpting, the values $k = 40$, and $\sigma = 0.99$ were used. The PCA preprocessing step was omitted. For Isomap II, 20 landmarks were used in every case. For LLE and HLLE, the value $k = 14$ was used to improve their respective scalability. It should also be noted that the available implementation of LLE uses an inefficient technique for finding neighbors, which we did not attempt to fix.

3.3 Document Manifolds

The utility of manifold learning algorithms for image processing applications has recently been recognized, but this is certainly not the only field that deals with multi-dimensional data. The Vector Space Model (18), for example, is commonly used in the field of Information Retrieval to characterize web documents. Each web page is represented as a large vector of term weights. The number of dimensions in the vector corresponds to the number of unique word stems (about 57,000 in English), and the values in these dimensions correspond to a term weight computed from the number of occurrences of the term in a document. Search queries can be evaluated by finding the documents whose vector has the closest angle with the vector of the query. This representation bears some striking resemblances to the pixel representation used for processing images, so it seems likely that similar results could be obtained by applying manifold learning to this field.

To test this hypothesis, we implemented a simple application for refining the results obtained from a *Google* search. A typical search often yields many thousands of documents, but users rarely have patience to look at more than the first few. Our application downloads the first 100 documents, removes stop words, stems each term with the Porter stemming algorithm (19), and represents the documents with the Vector Space Model. Next it uses Manifold Sculpting to reduce the dimensionality of the vectors to a single dimension. It then divides this dimension into two halves at the point that minimizes the sum variance of the two halves. Finally it extracts three terms to represent each of the two clusters by summing the vectors in the cluster and picking the three terms with the highest total weight. In theory, these two groups of terms should reflect the most significant range of context found in the query results,

and a user should be able to refine his or her query by selecting which of the two halves is closer to the intended meaning of the query.

As an example, a query for the term "speaker" yielded for one set of terms "box", "amp", and "off", and for the other set "hit", "baseball", and "bat". The first interpretation of the term "speaker" probably comes from references to audio devices. Prior to performing this experiment we did not know that this term had anything to do with baseball, but a search for the refined query "speaker baseball" yields many documents with information about Tris Speaker who was elected to the baseball hall of fame in 1937. Not all queries yielded such distinctly separated results, but this is an area with much potential for further research.

Chapter 4

Semi-supervision

Manifold learning and clustering have many things in common. Clustering collections of multidimensional vectors such as images, for example, is more effective when manifolds in the data are taken into account (20). Manifold learning and clustering are both unsupervised operations. Unlike clustering however, which groups vectors into a discrete number of buckets, manifold learning arranges them into a discrete number of continuous spectra. Essentially, clustering is to manifold learning as classification is to regression. It seems intuitive, therefore, that techniques which benefit clustering algorithms may have a corresponding counterpart for manifold learning algorithms. Clustering algorithms can be greatly enhanced with partial supervision (21). Likewise, a small modification to the Manifold Sculpting algorithm makes it possible to perform semi-supervised manifold learning.

Semi-supervised clustering involves a subset of data points for which classification values or hints about those values are provided. Semi-supervised manifold learning likewise requires final values or estimated final values to be provided for a subset of the data points. During scaling iterations (step 4 of the Manifold Sculpting algorithm), these supervised points are moved to their final values. As these points are moved, they tend to recursively pull their neighbors along with them, resulting in more efficient and potentially more accurate manifold learning. Figure 4.1 shows the amount of time required to learn the manifold for each of the four videos that were used in section 3.1.5 with a varying percentage of supervised points. In these

experiments, most of the performance gain comes from the first 20% of supervised points. With the Swiss Roll problem, for example, computation time can be cut in half with fewer than 5% supervised points.

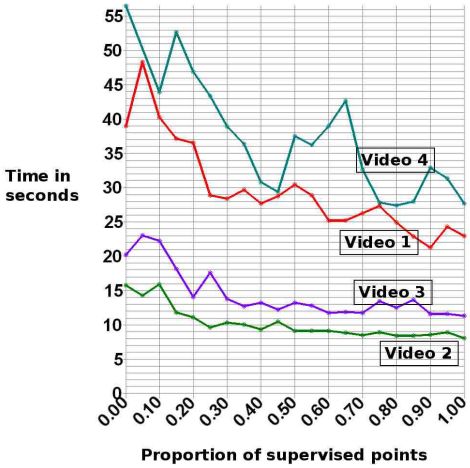


Figure 4.1: Manifold learning is faster when more points are supervised. (When most of the points are supervised, the only significant cost is the neighbor-finding step of the algorithm.)

In some cases data may not be available all at once. In such cases it may be desirable to learn the manifold incrementally (22). This is naturally achieved with semi-supervised manifold learning. Each time new data becomes available, the entire manifold is re-learned with all the old data acting as supervised values. With successive passes, a larger percentage of the points will be used in a supervised manner, thus causing each learning cycle to execute more quickly.

Often, semi-supervised clustering is performed by having human experts classify a subset of the available data. When real values are required, however, it may only be reasonable to ask human experts to provide vague estimates. For example, humans might be able to easily sort a collection of images according to relevance to a particular topic. The sorted images could be assigned sequential output values, and

these values could be used as estimates for the points in manifold coordinates. However, precise values may be difficult to obtain initially. Fortunately, even estimated output values can be useful in semi-supervised manifold learning. This benefit is exploited by using the estimated output values as a guide only during the initial scaling iterations when coarse refinements are being made. The points are then set free during later iterations when more precise refinements are performed on the data. The more confident the supervisor is about provided output values, the more iterations in which the supervision can provide benefit to the algorithm.

4.1 Conclusions

The experiments tested in this paper show that Manifold Sculpting yields more accurate results than other well-known manifold learning algorithms. Manifold Sculpting is robust to holes in the sampled area. Manifold Sculpting is more accurate than other algorithms when the manifold is sparsely sampled, and the gap is even wider with higher sampling densities. Manifold Sculpting has difficulty when the selected number of neighbors is too small but consistently outperforms other algorithms when it is larger.

Manifold Sculpting scales better than most manifold learning algorithms when the number of data points is much larger than the number of input dimensions. In the asymptotic case (with respect to data size) with a constant number of input dimensions, therefore, Manifold Sculpting performs better than other known manifold learning algorithms.

Manifold Sculpting benefits significantly when the data is pre-processed with the transformation step of PCA. Current research seeks to determine the extent to which other dimensionality reduction algorithms may work in tandem with Manifold Sculpting to produce high quality results in an efficient manner. Ongoing research also seeks to exploit the semi-superved capability of Manifold Sculpting to produce results that could not be obtained from a purely unsupervised manifold learning algorithm.

Bibliography

[1] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton, NJ, USA: Princeton University Press, 1961. 1

[2] R. N. Shepard and J. D. Carroll, "Parametric representation of nonlinear data structures (p. r. krishnaiah ed.)," in *International Symposium on Multivariate Analysis*, vol. 2. New York: Academic Press, 1965, pp. 561–592. 2

[3] J. W. Sammon, "A nonlinear mapping for data structure analysis," in *IEEE Transactions on Computers*, vol. C-18(5), 1969, pp. 401–409. 2

[4] P. Demartines and J. Hérault, "Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 148–154, January 1997. [Online]. Available: citeseer.ist.psu.edu/demartines96curvilinear.html 2

[5] J. Lee, A. Lendasse, N. Donckers, and M. Verleysen, "A robust non-linear projection method," in *ESANN 2000, European Symposium on Artificial Neural Networks*, Bruges (Belgium), 2000, pp. 13–20. 2

[6] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000. 2

[7] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, pp. 2323–2326, 2000. 2

[8] V. de Silva and J. B. Tenenbaum, "Global versus local methods in nonlinear dimensionality reduction." in *NIPS*, 2002, pp. 705–712. 2

[9] B. Schölkopf, A. J. Smola, and K.-R. Müller, "Kernel principal component analysis," *Advances in kernel methods: support vector learning*, pp. 327–352, 1999. 3

[10] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering." in *Advances in Neural Information Processing Systems, 14*, 2001, pp. 585–591. 3

[11] M. Brand, "Charting a manifold," in *Advances in Neural Information Processing Systems, 15*, S. T. S. Becker and K. Obermayer, Eds. Cambridge, MA: MIT Press, 2003, pp. 961–968. 3

[12] P. Vincent and Y. Bengio, "Manifold parzen windows," in *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press, 2003, pp. 825–832. 3

[13] D. Donoho and C. Grimes, "Hessian eigenmaps: locally linear embedding techniques for high dimensional data," *Proc. of National Academy of Sciences*, vol. 100, no. 10, pp. 5591–5596, 2003. 3

[14] Y. Bengio and M. Monperrus, "Non-local manifold tangent learning," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 129–136. 3

[15] E. Levina and P. J. Bickel, "Maximum likelihood estimation of intrinsic dimension." in *NIPS*, 2004. 3

[16] Z. Zhang and H. Zha, "A domain decomposition method for fast manifold learning," in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006. 3

[17] S. Roweis, "Em algorithms for PCA and SPCA," in *Advances in Neural Information Processing Systems*, M. I. Jordan, M. J. Kearns, and S. A. Solla, Eds., vol. 10. The MIT Press, 1998. [Online]. Available: http://citeseer.ist.psu.edu/roweis98em.html 8

[18] V. V. Raghavan and S. K. M. Wong, "A critical analysis of vector space model for information retrieval," *Journal of the American Society for Information Science*, vol. 37, no. 5, pp. 279–287, 1986. 32

[19] M. Porter, "An algorithm for suffix stripping," in *Program*, vol. 14(3), 1980, pp. 130–137. 32

[20] M. Breitenbach and G. Z. Grudic, "Clustering through ranking on manifolds," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM Press, 2005, pp. 73–80. [Online]. Available: http://doi.acm.org/10.1145/1102351.1102361 35

[21] A. Demiriz, K. Bennett, and M. Embrechts, "Semi-supervised clustering using genetic algorithms," *Artificial Neural Networks in Engineering*, 1999. [Online]. Available: citeseer.ist.psu.edu/demiriz99semisupervised.html 35

[22] M. H. C. Law, N. Zhang, and A. K. Jain, "Nonlinear manifold learning for data stream," Department of Computer Science and Enginering, Michigan State University, Tech. Rep. MSU-CSE-03-5, 2004. [Online]. Available: citeseer.ist.psu.edu/691968.html 36