Efficient, sparse representation of manifold distance matrices for classical scaling

Javier S. Turek
Intel Labs, Hillsboro, Oregon
javier.turek@intel.com

Alexander G. Huth
Departments of Computer Science & Neuroscience, The University of Texas at Austin
huth@cs.utexas.edu

Abstract

Geodesic distance matrices can reveal shape properties that are largely invariant to non-rigid deformations, and thus are often used to analyze and represent 3-D shapes. However, these matrices grow quadratically with the number of points. Thus for large point sets it is common to use a low-rank approximation to the distance matrix, which fits in memory and can be efficiently analyzed using methods such as multidimensional scaling (MDS). In this paper we present a novel sparse method for efficiently representing geodesic distance matrices using biharmonic interpolation. This method exploits knowledge of the data manifold to learn a sparse interpolation operator that approximates distances using a subset of points. We show that our method is 2x faster and uses 20x less memory than current leading methods for solving MDS on large point sets, with similar quality. This enables analyses of large point sets that were previously infeasible.

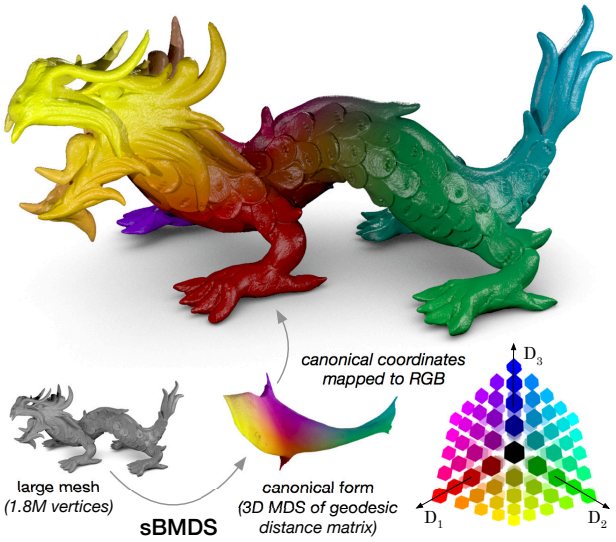

Figure 1. *Sparse Biharmonic Multidimensional Scaling* (sBMDS) was used to obtain the canonical form for a mesh with 1.8M vertices. MDS is impossible on the full distance matrix (1.8M x 1.8M, 26 TB), but easy using a sparse approximation (50,000 landmarks, 20.9 GB). Here the 3-D canonical coordinates found by MDS are are mapped into RGB colors on the original mesh.

1. Introduction

Distance matrices have many important roles in machine learning, graphics, and computer vision. Yet with growing data, it is becoming impossible to store or process full distance matrices. For n points, $\mathcal{O}\left(n^2\right)$ space is required to store the distance matrix, and, depending on the type of data and distance metric, as much as $\mathcal{O}\left(n^2 \log n\right)$ time can be required to compute it. Many algorithms that operate on distance matrices execute in $\mathcal{O}\left(n^3\right)$ time. With growing n these requirements quickly become intractable.

One solution is to work with a low-rank approximation of the distance matrix. While the best rank k approximation is given by the Singular Value Decomposition (SVD), its $\mathcal{O}\left(n^3\right)$ computation is impractical for large matrices. An alternative is the Nyström method [17], which computes a low-rank approximation by subsampling l columns from the distance matrix. Nyström has been studied theoretically [7, 12] and empirically [4, 13, 22], but it is a generic method that does not take advantage of any knowledge about the geometry of the point set.

For the specific problem of approximating geodesic distance matrices computed from 3-D meshes, methods such as spectral MDS (SMDS) [1] and fast MDS (FMDS) [21] have been proposed. These methods were designed to compress large geodesic distance matrices so that they can be analyzed using multidimensional scaling (MDS). The key insight offered by SMDS and FMDS that is not exploited by methods such as Nyström is that the geometry of the

distance matrix closely mirrors the geometry of the underlying point set. If the point set lies along some manifold, then rows of the distance matrix will lie along a higher-dimensional projection of that same manifold. Intuitively this follows from the fact that nearby points on a manifold will have similar distances to other points in the manifold.

Here we explore a novel method, biharmonic matrix approximation (BHA), and its sparse form sBHA. Like SMDS and FMDS, our method uses biharmonic interpolation to approximate the full distance matrix from a few samples. However, we improve upon those methods by noting that most of the values in the biharmonic interpolation operator are very close to zero, and thus the operator can be represented sparsely with little to no effect on accuracy of the approximation. Sparsification also increases the efficiency of algorithms that use the approximate distance matrix, such as MDS. Our method thus makes it possible to approximate and operate on extremely large point sets where existing methods would suffer poor performance due to memory constraints (Figure 1).

2. Background

Given a set of n points from a manifold $\mathcal{M}$ embedded in $\mathbb{R}^d$, $\mathcal{X} = \{\mathbf{x_i}\}_{i=1}^n$ with $\mathbf{x_i} \in \mathcal{M}$, we define the geodesic distance matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ as

$$\mathbf{K}_{i,j} = d_{\mathcal{M}}\left(\mathbf{x_i}, \mathbf{x_j}\right), \tag{1}$$

where $\mathbf{K}_{i,j}$ represents the element in row i and column j of the matrix $\mathbf{K}$, and $d_{\mathcal{M}}(\cdot,\cdot)$ is the geodesic distance, or the length of the shortest path between two points along the surface of $\mathcal{M}$. Assume that $d_{\mathcal{M}}(\cdot,\cdot)$ is symmetric for a pair of points, i.e., $d_{\mathcal{M}}(\mathbf{x_i}, \mathbf{x_j}) = d_{\mathcal{M}}(\mathbf{x_j}, \mathbf{x_i})$.

2.1. Biharmonic interpolation

Let g be a real-valued function defined on a smooth manifold $\mathcal{M}$ embedded in $\mathbb{R}^d$. The manifold has an associated Laplace-Beltrami Operator (LBO), Δ, that depends on the Riemannian metric of the manifold [20] such that $\Delta g = \text{div}\ (\text{grad}\ g)$.

Now suppose that we are given $g(\mathbf{b_i})$ for a set of l points, $\{\mathbf{b_i}\}_{i=1}^l \in \mathcal{M}$, and wish to interpolate $g(\mathbf{u_i})$ at a different set of m points, $\{\mathbf{u_i}\}_{i=1}^m \in \mathcal{M}$. One solution is biharmonic interpolation, which finds a smooth function (i.e. one with continuous second derivative) that passes exactly through $g(\mathbf{b_i})$ for all i. Biharmonic interpolation is accomplished by finding a solution to the biharmonic equation $\Delta^2 g(\mathbf{u}) = 0$, subject to Dirichlet boundary conditions given by $g(\mathbf{b})$. Note that this is equivalent to exactly minimizing the smoothness-preserving energy function defined in [1] and [21].

In the discrete case the biharmonic operator Δ^2 is specified as a sparse matrix $\mathbf{M}$. We assume that the manifold consists exclusively of the data points $\mathbf{b}$ and $\mathbf{u}$, and thus that $\mathbf{M}$ is an $(l+m) \times (l+m) = n \times n$ matrix. We can organize $\mathbf{M}$ so that the first l rows and columns correspond to $\mathbf{b}$, our known data points, and the last m rows and columns correspond to $\mathbf{u}$, the points at which we wish to interpolate g. Thus $\mathbf{M}$ can be split into four submatrices,

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{bb} & \mathbf{M}_{bu} \\ \mathbf{M}_{ub} & \mathbf{M}_{uu} \end{bmatrix}. \tag{2}$$

To find the interpolated values we then solve for $\hat{g}(\mathbf{u})$ in the modified biharmonic equation

$$\begin{aligned} \begin{bmatrix} \mathbf{M}_{bb} & \mathbf{M}_{bu} \\ \mathbf{M}_{ub} & \mathbf{M}_{uu} \end{bmatrix} \begin{bmatrix} g(\mathbf{b}) \\ \hat{g}(\mathbf{u}) \end{bmatrix} &= \mathbf{0}, \\ \mathbf{M}_{ub} g(\mathbf{b}) + \mathbf{M}_{uu} \hat{g}(\mathbf{u}) &= \mathbf{0}, \end{aligned}$$

yielding the solution

$$\hat{g}(\mathbf{u}) = -\mathbf{M}_{uu}^{-1} \mathbf{M}_{ub} g(\mathbf{b}), \tag{3}$$

which is a fully specified, sparse linear system of equations that can be solved using standard methods. Note that $\hat{g}(\mathbf{u})$ is related to $g(\mathbf{b})$ through a linear transformation that is independent of the values in $g(\mathbf{b})$. We can think of this linear transformation, $-\mathbf{M}_{uu}^{-1}\mathbf{M}_{ub}$, as an *interpolation operator*: a transformation that will interpolate any function known on the points $\mathbf{b}$ onto the points $\mathbf{u}$.

2.2. Obtaining the discrete biharmonic operator

The discrete biharmonic operator $\mathbf{M}$ is given as [14]

$$\mathbf{M} = (\mathbf{V} - \mathbf{A})^\top \mathbf{D}^{-1} (\mathbf{V} - \mathbf{A}),$$

where $\mathbf{D}$ is a diagonal matrix containing the "lumped mass" at each point, $\mathbf{A}$ is a weighted adjacency matrix, and $\mathbf{V}$ is a diagonal matrix containing the sum of the adjacency weights for each datapoint $\mathbf{V}_{i,i} = \sum_j \mathbf{A}_{i,j}$ [20].

In some cases there are closed-form solutions for the lumped mass and weighted adjacency matrices. If the point set $\mathcal{X}$ is a triangular mesh embedded in 3-D space, the lumped mass $\mathbf{D}_{i,i}$ is $1/3$ of the total area of the triangles incident on point i, and the adjacency weight

$$\mathbf{A}_{i,j} = \frac{\cot(\alpha_{i,j}) + \cot(\beta_{i,j})}{2},$$

where $\alpha_{i,j}$ and $\beta_{i,j}$ are the angles opposite the edge between points i and j [20, 5]. Similar solutions for $\mathbf{D}$ and $\mathbf{A}$ exist for point clouds [16] and quad meshes [6] in 3-D space. If $\mathcal{X}$ is a generic graph, we set $\mathbf{D} = \mathbf{I}$, and set $\mathbf{A}$ equal to the graph adjacency matrix, where $\mathbf{A}_{i,j} = 1$ if nodes i and j share an edge, and 0 otherwise.

If no graph is given, $\mathbf{D}$ and $\mathbf{A}$ can be estimated. Here it is common to again set $\mathbf{D} = \mathbf{I}$ and then estimate $\mathbf{A}$ using some weighted nearest neighbor algorithm. We will not address this further here, but note that the weights given by Stochastic Neighbor Embedding (SNE) [10] perform particularly well at this problem.

2.3. Multidimensional Scaling

Given a matrix $\mathbf{K}$ with distances between all pairs of n points, Multidimensional Scaling (MDS) methods compute a low-dimensional embedding of these n points. The embedded coordinates $\{\mathbf{z_i}\}_{i=1}^{n} \in \mathbb{R}^m$ are found by minimizing, for all pairs of points, the difference between their Euclidean distances in the embedding $\|\mathbf{z_i} - \mathbf{z_j}\|_2$ and their squared distance $\mathbf{K}_{i,j}^2$ in the original space:

$$\arg\min_{\mathbf{Z}} \|\mathbf{Z}\mathbf{Z}^T + \frac{1}{2}\mathbf{J}\mathbf{E}\mathbf{J}\|_F, \tag{4}$$

where $\mathbf{Z} = [\mathbf{z_1}, \ldots, \mathbf{z_n}]^T \in \mathbb{R}^{n \times m}$ contains the embedded coordinates, $\mathbf{E}_{i,j} = \mathbf{K}_{i,j}^2$ are the squared distances, and $\mathbf{J} = \mathbf{I} - 1/n\mathbf{1}\mathbf{1}^T$ is a centering matrix with $\mathbf{1}$ being a column vector of ones.

In classical MDS the optimal solution to Problem (4) is obtained by first computing the eigenvalue decomposition $\mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$ of the matrix $-1/2\mathbf{J}\mathbf{E}\mathbf{J}$, and then truncating its decomposition to the biggest m eigenvalues, $\mathbf{\Lambda}_m$, and their respective eigenvectors $\mathbf{V}_m$. The embedded points are then computed as $\mathbf{Z} = \mathbf{V}_m\mathbf{\Lambda}_m^{1/2}$.

Solving Problem (4) requires storing the matrix $\mathbf{E}$ in memory, which is prohibitive for more than a few tens of thousands of points. To overcome this limitation, alternative methods use a low-rank approximation of $\mathbf{E}$. This is achieved by subsampling the points and interpolating their distances. Methods such as Landmark MDS (LMDS) [23], SMDS [1], and FMDS [21] propose different solutions to this approximation problem. In what follows, we present an alternative method to approximate $\mathbf{E}$ with a low-rank and *sparse* approximation, simultaneously enabling a smaller memory footprint and faster MDS algorithm for very large numbers of points.

3. Biharmonic matrix approximation (BHA)

In the proposed method we use biharmonic interpolation to approximate a manifold-structured distance matrix $\mathbf{K}$. We exploit the fact that the manifold structure of $\mathbf{K}$ is similar to that of the underlying point set $\mathcal{X} = \{\mathbf{x_i}\}_{i=1}^{n}$. We assume that $\mathcal{X}$ lives in a manifold $\mathcal{M}$ whose biharmonic operator $\mathbf{M}$ can either be computed directly or approximated as described in Section 2.2.

The first step is to select $\mathbf{b}$, a set of l landmark points from $\mathcal{X}$. Landmarks are selected using an iterative farthest point procedure [11]: the first landmark $\mathbf{b_1}$ is selected at random from $\mathcal{X}$, and the geodesic distance from that point to all the other points, $\mathbf{K}_{\mathbf{b_1},\cdot}$ is computed. Each subsequent landmark is then chosen as the point with the largest minimum distance to any of the current landmarks,

$$\mathbf{b_j} = \underset{\mathbf{x_i}}{\text{argmax}} \left(\min_{t=1}^{j-1} \mathbf{K}_{\mathbf{b_t},\mathbf{x_i}} \right) \tag{5}$$

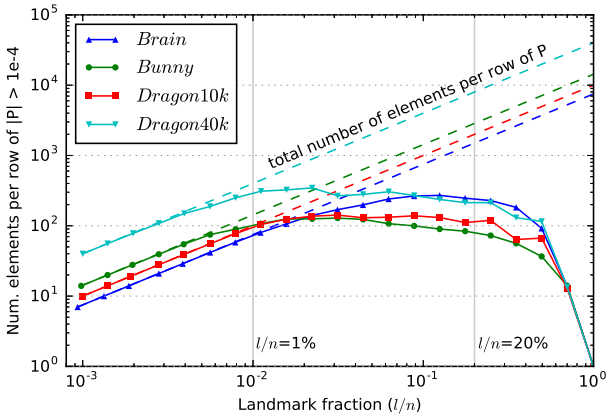

Figure 2. Sparsity of $\mathbf{P}$ as a function of number of landmarks. The number of large elements per row of $\mathbf{P}$ peaks at landmark fractions $l/n < 20\%$ for all point sets.

until l landmarks have been selected.

The second step in BHA is to create an interpolation operator $\mathbf{P}$ that interpolates values from the landmarks $\mathbf{b}$ to the entire manifold by solving Equation 3,

$$\mathbf{P} = \begin{bmatrix} \mathbf{I}_l \\ -\mathbf{M}_{uu}^{-1}\mathbf{M}_{ub} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_l \\ \mathbf{P}_u \end{bmatrix}, \tag{6}$$

where $\{\mathbf{u_i}\} = \mathcal{X} \backslash \{\mathbf{b_i}\}$ is the set of all non-landmark points, and $\mathbf{M}_{uu}$ and $\mathbf{M}_{ub}$ are defined as in Equation (2).

The third step in BHA is to form $\mathbf{W} \in \mathbb{R}^{l \times l}$, the diagonal block from $\mathbf{K}$ that contains geodesic distances between landmark points: $\mathbf{W}_{i,j} = d_{\mathcal{M}}(\mathbf{b_i}, \mathbf{b_j})$ for $1 \leq i, j \leq l$. Note that these values were already computed during the landmark selection procedure, and can be re-used here.

The *Biharmonic Matrix Approximation (BHA)* is then obtained as

$$\tilde{\mathbf{K}}^{BHA} = \mathbf{P}\mathbf{W}\mathbf{P}^{\top}. \tag{7}$$

BHA can be thought of as performing two interpolation operations. First, the product $\mathbf{P}\mathbf{W}$ interpolates geodesic distances from the landmarks to all the points, approximating the $n \times l$ submatrix formed by the columns of $\mathbf{K}$ corresponding to the landmarks. Right-multiplying by $\mathbf{P}^{\top}$ then interpolates each row of $\mathbf{P}\mathbf{W}$ across the entire manifold, giving the full $n \times n$ approximation $\tilde{\mathbf{K}}^{BHA}$.

3.1. From dense to sparse

Analyzing the interpolation operator $\mathbf{P}$ reveals some useful numerical properties. The coefficients in $\mathbf{P}$ define weights that are used to interpolate values in $\mathbf{W}$ onto the non-landmark points $\mathbf{u_i}$. Intuitively, as the number of landmarks grows, the number of data points influenced by each landmark will shrink. In the limit of the number of landmarks approaching the total number of data points, $\mathbf{P} \to I$, and each data point is only influenced by a single landmark. Thus, even though biharmonic interpolation operators are

not theoretically guaranteed to be sparse (i.e. they do not have compact support), empirically most entries in $\mathbf{P}$ tend to be near zero (Figure 2).

Therefore, instead of computing the interpolation operator $\mathbf{P}$ given in Equation (6), we propose to compute a sparse interpolation operator. We note that Equation (6) describes the solution to a system of linear equations to obtain $\mathbf{P}$. We replace the matrix inversion by a sparse coding problem of the form

$$\begin{aligned} \tilde{\mathbf{P}}_u &= \arg\min_{\mathbf{P}_u} \|\mathbf{M}_{uu}\mathbf{P}_u + \mathbf{M}_{ub}\|_F^2 \\ &\quad \text{s.t.} \quad \|\mathbf{P}_u^{(i)}\|_0 \leq p \quad \forall i, \end{aligned} \tag{8}$$

where the $\|\mathbf{x}\|_0$ is the ℓ_0-quasinorm that measures the number of non-zeros in a vector $\mathbf{x}$, $\mathbf{P}_u$ is the submatrix of $\mathbf{P}$ corresponding to the non-landmark points $\{\mathbf{u_i}\}$, $\mathbf{P}_u^{(i)}$ is the i^{th} column of $\mathbf{P}_u$, and p is a scalar value. Problem (8) constrains the solution to be a sparse matrix $\tilde{\mathbf{P}}_u$ with at most p non-zeros per column. The sparse interpolation operator $\mathbf{P}_{sparse}$ is obtained by plugging the solution $\tilde{\mathbf{P}}_u$ into Equation (6). The *Sparse Biharmonic Matrix Approximation (sBHA)* is then obtained as

$$\tilde{\mathbf{K}}^{sBHA} = \mathbf{P}_{sparse}\mathbf{W}\mathbf{P}_{sparse}^{\top}. \tag{9}$$

Sparse coding is a combinatorial problem, so its solution is typically approximated using greedy methods or convex relaxations such as OMP [19] or LASSO [24]. However, in cases such as this where most entries in $\mathbf{P}$ are already very close to zero, we can use the much cheaper Thresholding method [8] to approximate Problem (8). Noting that the matrix $\mathbf{M}_{uu}$ is square and invertible, this method chooses the p biggest elements in magnitude from the dense solution for each column of $\mathbf{P}$. Each column of $\mathbf{P}_{sparse}$ is solved separately, requiring $\mathcal{O}(n)$ memory to store one dense column of $\mathbf{P}$ at a time, and $\mathcal{O}(n)$ runtime complexity to find the biggest p elements. Typically, we are interested in p_{row}, the average number of non-zeros per row. The relation between p and a p_{row} parameter is given as $p = {}^{(n-l)p_{row}}/_{l}$.

How big must p_{row} be in order for $\mathbf{P}_{sparse}$ to be a good approximation of $\mathbf{P}$? If a large p_{row} is required, then there are no memory or runtime benefits of using $\mathbf{P}_{sparse}$. However, since most elements in $\mathbf{P}$ are close to zero, it seems likely that p_{row} does not need to be very large. Although we provide no theoretical proof, our empirical evaluation in Section 4 suggests that p_{row} can be considered a small constant number. $p_{row} \leq 50$ seems to work well for any acceptable number of landmarks for large point sets.

3.2. Runtime and space complexity

To analyze the runtime and space complexity of BHA we divide the method into two steps: a preprocessing step that computes the biharmonic operator $\mathbf{M}$, and a step that computes the interpolation operator $\mathbf{P}$ and the diagonal block matrix $\mathbf{W}$ for the landmarks.

In the preprocessing step, we compute the lumped mass matrix $\mathbf{D}$, the adjacency matrix $\mathbf{A}$ and the sum of adjacency weights $\mathbf{V}$ in order to obtain the biharmonic operator $\mathbf{M}$. Matrices $\mathbf{D}$ and $\mathbf{V}$ are diagonal and can be computed with one pass over the adjacency matrix $\mathbf{A}$, which is also sparse. When the manifold $\mathcal{M}$ is given as a mesh with at most t neighbors for each point, the adjacency matrix is given and there is no need for more computation or space than is required by these matrices, yielding a total runtime complexity $\mathcal{O}\left(nt^2\right)$ and space $\mathcal{O}\left(nt^2\right)$ to compute and store $\mathbf{M}$. Practically, t has a small value up to tens of neighbors per data point, such that still $t^2 \ll n$.

In the low-rank approximation step we draw a set of landmark points and then compute the interpolation operator $\mathbf{P}$ and the matrix $\mathbf{W}$. Finding landmarks using the farthest point procedure requires $\mathcal{O}(lf)$ time, where f is the complexity of evaluating one row of the distance matrix $\mathbf{K}$. Computing $\mathbf{P}$ requires solving the system of linear equations described in (3). There are l right hand side vectors and the system has $n - l$ equations. As $\mathbf{M}$ is a sparse matrix with $\mathcal{O}(n)$ non-zeros (assuming $t \ll n$), Equation (3) can be solved with sparse linear solvers in $\mathcal{O}(T_P nl)$, where T_P is the number of iterations needed to converge to an specific accuracy. Alternatively, we can compute the sparse Cholesky decomposition of $\mathbf{M}_{uu}$ and then solve for the l right hand side vectors, having a runtime complexity $\mathcal{O}\left(n^{1.5} + nl\right)$. In the dense case, computing $\mathbf{P}$ requires $\mathcal{O}(nl)$ space. In the sparse case, using p_{row} non-zeros per row with $p_{row} < l$ and practically constant, the space is reduced to $\mathcal{O}(np_{row})$ non-zeros. Computing sparse columns adds no cost. Computing the submatrix $\mathbf{W}$ of $\mathbf{K}$ requires $\mathcal{O}\left(l^2\right)$ space and $\mathcal{O}(lf)$ time, but $\mathbf{W}$ can re-use the distances computed while selecting landmarks. Thus sBHA has runtime complexity of $\mathcal{O}\left(nt^2 + T_P nl + lf\right)$ and space complexity of $\mathcal{O}\left(\max\{np_{row} + l^2, nt^2\}\right)$.

3.3. Classical Scaling with BHA

After computing the low-rank approximation $\tilde{\mathbf{E}}$ of the matrix $\mathbf{E}$ of squared distances, one can solve Classical Scaling by extracting the eigenvalues of $-{}^1/_2\mathbf{J}\tilde{\mathbf{E}}\mathbf{J} = -{}^1/_2\mathbf{J}\mathbf{P}\mathbf{W}\mathbf{P}^T\mathbf{J}$. We remind the reader that our aim is to compute the m biggest eigenvalues and respective eigenvectors without computing a prohibitive $n \times n$ matrix. When using BHA, we follow the method proposed by [21]. We dub this implementation *Biharmonic Multidimensional Scaling (BMDS)*. First we compute the QR decomposition $\mathbf{QR} = \mathbf{JP}$, where $\mathbf{R} \in \mathbb{R}^{l \times l}$. Then, we compute the eigen-decomposition of the matrix $-{}^1/_2\mathbf{R}\mathbf{W}\mathbf{R}^T$ given by $\tilde{\mathbf{V}}\Lambda\tilde{\mathbf{V}}^T$. The embedding is computed as $\mathbf{Z} = \mathbf{Q}\tilde{\mathbf{V}}_m\mathbf{\Lambda}_m^{1/2}$. Overall runtime complexity for this method is $\mathcal{O}\left(nl^2 + l^3\right)$.

The above solution computes a QR decomposition that requires $\mathcal{O}(nl)$ memory to store matrix $\mathbf{Q}$. This solution works, but its memory usage can be prohibitive when

n is very large. Alternatively, we propose *sparse BMDS (sBMDS)*, which uses the Lanczos method [18] to compute only the needed m eigenvalues and eigenvectors. The Lanczos method requires only matrix-vector multiplications, avoiding the storage of big matrices [18]. This multiplication is very fast as the interpolation matrix $\mathbf{P}_{sparse}$ is sparse, the matrix $\mathbf{J}$ is a sum of identity and a rank-one matrix, and $\mathbf{W}$ is small compared to n. Overall, computing the m eigenvalues and eigenvectors with such matrix-vector multiplications has a runtime complexity of $\mathcal{O}\left(mn + mnp_{row} + ml^2 + m^2\right)$ with a small additional space for a vector of length $\mathcal{O}(n)$.

3.4. Relationships to other approximation methods

3.4.1 Nyström

The Nyström method [17] is a symmetric matrix approximation that has been successfully applied to machine learning problems on many datasets. The method requires one to sample a subset of l landmark points from the point set and compute the submatrix $\mathbf{C} \in \mathbb{R}^{n\times l}$, which consists of the corresponding l columns of $\mathbf{K}$. The low-rank approximation is then obtained as

$$\tilde{\mathbf{K}}^{Nys} = \mathbf{C}\mathbf{W}^{\dagger}\mathbf{C}^{\top}, \tag{10}$$

where $\mathbf{W} \in \mathbb{R}^{l\times l}$ is the symmetric diagonal block corresponding to the columns and rows of $\mathbf{K}$ for the landmarks, and $\mathbf{W}^{\dagger}$ is its Moore-Penrose pseudoinverse. While BHA may appear structurally similar to Nyström, a critical difference is that BHA does not compute the pseudoinverse $\mathbf{W}^{\dagger}$. This renders BHA more stable than Nyström in situations where $\mathbf{W}$ is (near-)singular. The largest difference between BHA and Nyström is that Nyström does not use any information about the structure of the manifold from which the data is drawn. This hurts Nyström in situations where the manifold is known (*e.g.* a 3-D mesh), since the manifold can be exploited efficiently. When the manifold is unknown, the steps taken to recover it can mean that BHA and other manifold-based methods take longer to set up.

The Nyström approximation requires $\mathcal{O}\left(nl + l^2\right)$ space for storing the matrices $\mathbf{C}$ and $\mathbf{W}^{\dagger}$. The runtime is $\mathcal{O}(lf)$ for the computation of $\mathbf{C}$, where f is the complexity of evaluating one row of the distance matrix $\mathbf{K}$, and $\mathcal{O}\left(l^3\right)$ for computing the pseudoinverse of $\mathbf{W}$.

3.4.2 FMDS

In fast multidimensional scaling (FMDS) [21] the distance matrix is approximated as the symmetrized product of an interpolation matrix, $\mathbf{H} \in \mathbb{R}^{n\times l}$, and a matrix $\mathbf{C} \in \mathbb{R}^{l\times n}$ formed by l rows from $\mathbf{K}$ (similar to Nyström),

$$\tilde{\mathbf{K}}^{FMDS} = \frac{1}{2}(\mathbf{H}\mathbf{C} + \mathbf{C}^{\top}\mathbf{H}^{\top}). \tag{11}$$

The interpolation operator $\mathbf{H}$ is similar but not identical to the BHA interpolation operator $\mathbf{P}$. The difference lies in the fact that $\mathbf{P}$ does exact interpolation, meaning that the values at known points (the landmarks $\mathbf{b}$) must be exactly equal to known values. In contrast, $\mathbf{H}$ allows some small error at the known points, where the amount of error is controlled by a scalar coefficient μ [21]. It is possible to linearly transform $\mathbf{P}$ into $\mathbf{H}$: $\mathbf{H} = \mathbf{P}(\mathbf{M}_{bb} + \mu\mathbf{I}_l + \mathbf{M}_{bu}\mathbf{P}_u)^{-1}\mu$, where $\mathbf{P}_u$ is defined as in Equation 6.

It is important to note that FMDS stores much more of the distance matrix in memory than sBHA. Since it uses such a similar method for interpolation, it is expected that FMDS will yield better performance with the same number of landmarks, albeit using much more memory. Also, the symmetrization step required by FMDS can be expensive for large meshes, whereas SMDS and sBHA are fundamentally symmetric.

3.4.3 SMDS

In spectral multidimensional scaling (SMDS) [1] the distance matrix is also approximated using interpolation, but the dimensionality of the problem is reduced by working in the spectral domain formed by the eigenspace of the LBO. First, sparse eigenvalue decomposition is used to extract the first m eigenvectors $\mathbf{\Phi} \in \mathbb{R}^{n\times m}$ and eigenvalues $\mathbf{\Lambda}$ of the LBO. Then l landmarks are selected and a smooth interpolation operator $\mathbf{H} \in \mathbb{R}^{m\times l}$ is computed. Finally distances are computed among the landmarks and stored in the symmetric matrix $\mathbf{W} \in \mathbb{R}^{l\times l}$ as in BHA. The approximation is then given as:

$$\tilde{\mathbf{K}}^{SMDS} = \mathbf{\Phi}\mathbf{H}\mathbf{W}\mathbf{H}^{\top}\mathbf{\Phi}^{\top}. \tag{12}$$

SMDS has several advantages: the resulting matrix is always symmetric, and working in the eigenspace of the LBO reduces dimensionality significantly. However, computing the eigenvectors is extremely costly in runtime, rendering this method generally less useful than FMDS.

3.4.4 Other methods

Other related methods include the constant time geodesic (CTG) approximation [25] and SpectroMeter (SM) method [15]. CTG uses a geometric approach to "unfold" landmark distances to the entire surface, and thus requires only 3 dimensions but also involves a nonlinear operation (taking the minimum across possible paths). SM uses a spectral decomposition of the Laplace-Beltrami operator (LBO) to rapidly approximate operations used in the heat method for computing geodesics [5], and to interpolate distances, similarly to SMDS.

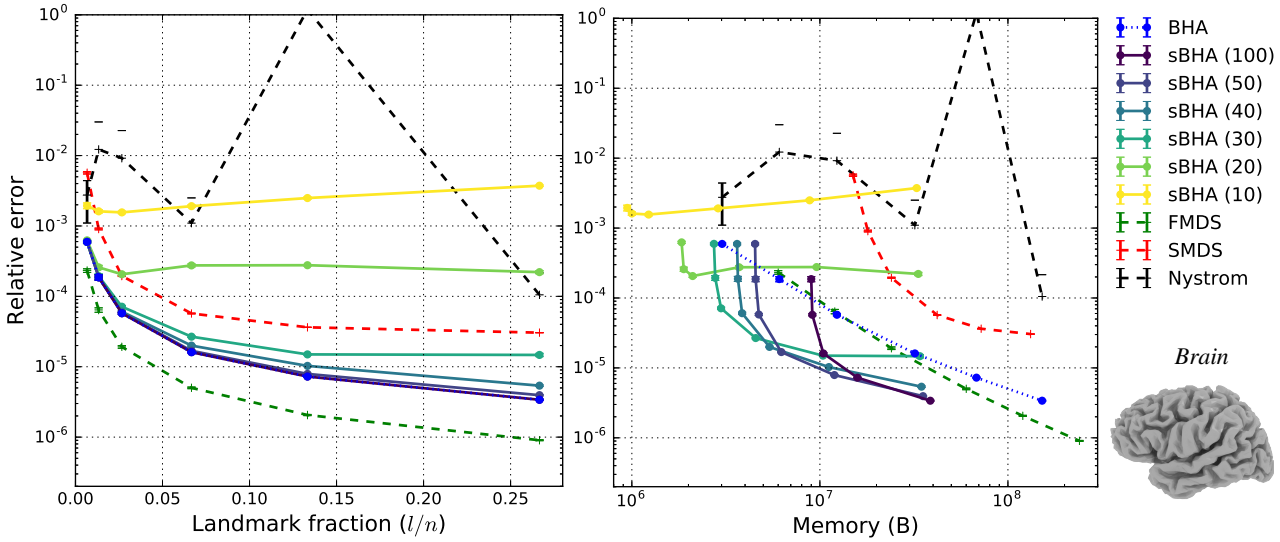

Figure 3. Geodesic distance approximation error for BHA and sBHA with $p_{row} = 10 \ldots 100$ (our methods) vs. FMDS, SMDS, and Nyström on *Brain* with numbers of landmarks l ranging from 1% to 25% of the total number of points $n = 7,502$. Each experiment was repeated 10x. **(Left)** Error vs. Landmark fraction l/n. FMDS has the lowest error for each number of landmarks, while Nyström suffers from numerical instability. There is little difference between BHA and sparse sBHA with $p_{row} = 100$ or $p_{row} = 50$, but sparser solutions suffer. **(Right)** Error vs. size of the approximation in memory (bytes). sBHA strongly outperforms FMDS, using 3-4x less memory to achieve the same error rate.

4. Experimental results

We empirically evaluate BHA and sBHA, and compare to Nyström [17], FMDS [21], SMDS [1], and LMDS [23] in terms of matrix approximation accuracy, MDS stress, memory usage and runtime. We used four point sets: *Brain* is a 3-D mesh reconstruction of one human cortical hemisphere with $n = 7,502$ vertices. *Bunny* is a 3-D mesh with $n = 14,290$ (Stanford Computer Graphics Laboratory). *Dragon* is a 3-D mesh with $n = 1,804,693$ (Stanford Computer Graphics Laboratory). Using quadric decimation in meshlab [3] we created 8 downsampled *Dragon* meshes with $n = 5,000 \ldots 750,000$. *TOSCA* [2] is a set of 148 3-D meshes from 12 categories. Within each category the same mesh appears in different poses.

All experiments were run on a system with two Intel Xeon E5-2699v4 processors (44 cores in total) with 128 GB of physical memory and running Linux. Code[1] was implemented in Python with optimized and parallelized NumPy and SciPy modules. BHA, sBHA, FMDS, and SMDS were implemented with sparse Cholesky decomposition from Scikit-Sparse. The FMDS smoothing parameter was set to $\mu = 50$ as recommended in [21]. For SMDS $m = 200$ eigenvectors were used, as recommended in [1]. Geodesic distances were computed using the approximate geodesics in heat method [5] implemented in pycortex [9]. All reported memory usage is the final memory consumption of the distance matrix approximation, not counting memory used for intermediate steps.

[1] http://github.com/alexhuth/BHA

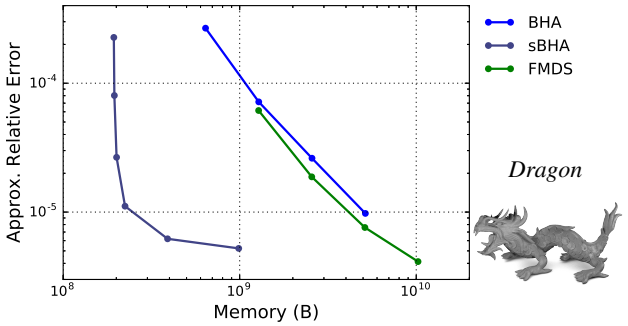

Figure 4. Geodesic distance approximation error for BHA and sBHA with $p_{row} = 50$ (our methods) vs. FMDS on *Dragon320k*, a large mesh with $n = 320,003$ vertices, using landmark fractions $l/n = 0.0008 \ldots 0.031$. Error is plotted vs. size of the approximation in memory (bytes). sBHA performs extremely well, using about 20x less memory than FMDS to achieve an error of 1 in 100,000.

4.1. Distance matrix approximation

Here we evaluate the approximation performance and memory usage of BHA and sBHA versus other methods. We compute low-rank approximations of the geodesic distance matrix for *Brain* using BHA, sBHA with the number of non-zeros per row p_{row} from 10 to 100, FMDS,

SMDS, and Nyström and then compare to the actual distance matrix. Relative approximation error was measured as $\varepsilon(\tilde{\mathbf{K}}) = \|\tilde{\mathbf{K}}-\mathbf{K}\|_F^2/\|\mathbf{K}\|_F^2$. With each method we varied the number of landmarks l between roughly 1% ($l = 50$) and 25% ($l = 2,000$) of the total number of points n.

Comparing performance as a function of the number of landmarks (Figure 3, left) shows that FMDS has the lowest error for each value of l. BHA performs identically to sBHA with $p_{row} = 100$ and is only slightly better than sBHA with $p_{row} = 50$. Sparser solutions perform worse. Nyström performs very badly due to numerical instability of the pseudoinverse $\mathbf{W}^\dagger$.

However, plotting performance as a function of the size of the approximation in memory (Figure 3, right) shows that sBHA uses 3-4x less memory to achieve the same level of performance as FMDS.

The same experiment was performed on *Bunny* (Supplemental Figures) with similar results for BHA, sBHA, and Nyström. However, FMDS and SMDS with default parameter settings performed much worse on *Bunny*, both failing to beat BHA for any given number of landmarks. This suggests that FMDS and SMDS may be more sensitive to hyperparameters than (s)BHA.

To test on a larger problem we also ran BHA, sBHA, and FMDS on *Dragon320k*, a mesh with $n = 320,003$. The full distance matrix was too large to fit in memory, so error was estimated using a random subset of 3000 rows. Figure 4 shows that sBHA strongly outperforms FMDS on this large mesh. To achieve an error of one part in 100,000, sBHA uses roughly 20x less memory than FMDS (200 MB vs. 4 GB). This factor is much larger than for *Brain*, suggesting that the efficiency gains of sBHA over FMDS grow with the size of the problem.

4.2. Obtaining canonical forms using MDS

FMDS, SMDS, and LMDS were designed specifically for applying MDS to large problems. Using MDS to embed a geodesic distance matrix into 3 dimensions gives the *canonical form* $\mathbf{Z} \in \mathbb{R}^{n\times 3}$, a representation of the dataset that is largely invariant to nonrigid deformations (Figure 5). Comparing canonical forms can reveal whether two meshes are different poses of the same model, and, if so, which parts match one other.

The quality of an MDS solution can be evaluated by computing stress (Equation 4), which measures how different the global geometries of the original and embedded datasets are. We compare MDS results from BMDS and sBMDS with $p_{row} = 50$ to FMDS, SMDS, LMDS, and normal MDS.

We first compare stress after applying MDS to *Brain* (Figure 6). As in the matrix approximation experiment, FMDS outperforms all other methods when using the same number of landmarks l, but sBMDS is much more efficient,

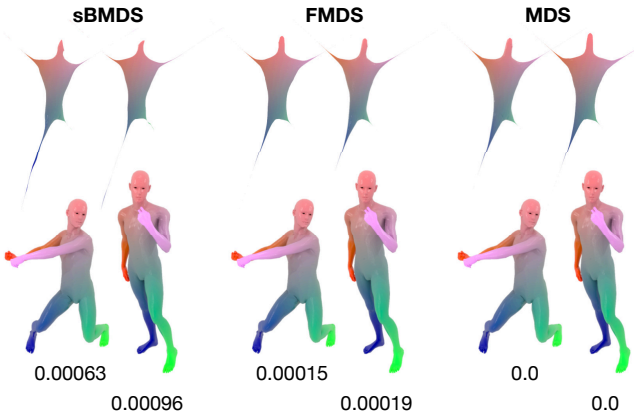

Figure 5. **(Top)** Canonical forms obtained using sBMDS (our method; using $l = 100$ landmarks and $p_{row} = 50$), FMDS (using $l = 100$ landmarks), and MDS on *david-1* and *david-2* from *TOSCA* and then aligned using ICP. **(Middle)** Canonical coordinates are mapped to RGB colors on the original meshes. Parts have the same coordinates (and color) despite pose differences. sBMDS yields nearly identical solutions while using much less memory and time than the other methods. **(Bottom)** Final stress of each approximate MDS solution − stress of the exact solution.

achieving the same stress while using much less memory. LMDS performs the worst for most landmark fractions. Comparisons on *Bunny* were similar to the matrix approximation results (Supplemental Figures).

We next compared scalability of the six methods in terms of runtime and memory usage by testing on *Dragon* meshes having 5,000 to 320,000 vertices. All tests used $l = 1,000$ landmarks. Runtime (Figure 7, left) was best for LMDS, followed closely by sBMDS, BMDS, and FMDS. SMDS was much slower, taking twice as long to embed *Dragon160k* as sBMDS took to embed the larger *Dragon320k*. All approximate methods were much faster than normal MDS, which could only be run on meshes up to 40,000 vertices due to memory constraints. Memory usage (Figure 7, right) scales linearly with mesh size for all methods, but the total memory used varies wildly. sBMDS used by far the least memory, followed by SMDS. FMDS used the most memory.

Finally we studied the quality of the canonical forms obtained using each MDS method. Following [21] we compared 61 meshes coming from 5 different categories (cat, david, horse, lioness, centaur) in *TOSCA* [2]. We first obtained a canonical form for each mesh using each method. Examples are shown in Figure 5. Iterated closest point (ICP) was then used to find the distance between each pair of canonical forms, and these distances were submitted to a second stage MDS. The resulting embeddings (Supplemental Figures) clearly cluster according to category for each approximation method.

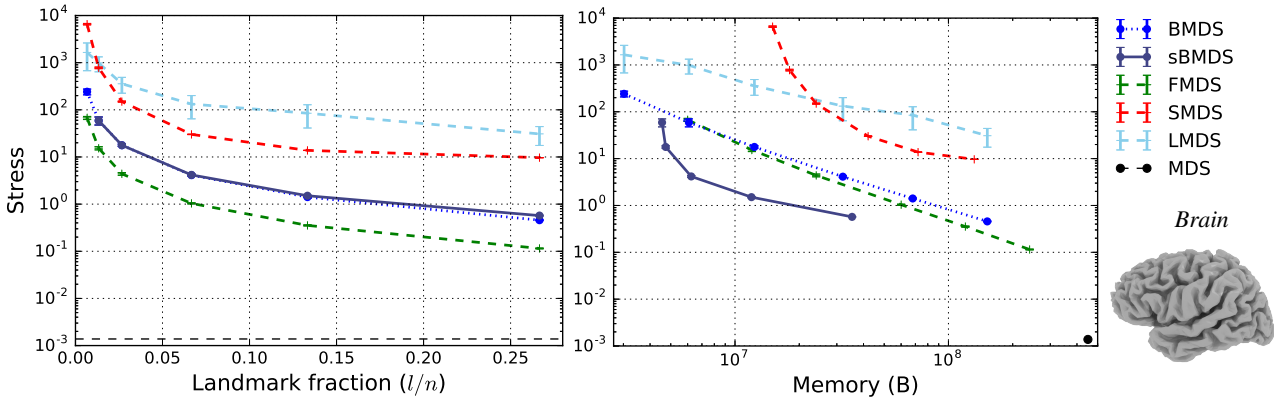

Figure 6. MDS stress for BMDS and sBMDS with $p_{row} = 50$ (our methods) versus FMDS, SMDS, LMDS, and normal MDS after embedding the geodesic distance matrix for *Brain* into 3-D. Numbers of landmarks l ranged from 1% to 25% of the total number of points $n = 7,502$. Each experiment was repeated 10x. **(Left)** Stress vs. Landmark fraction l/n. FMDS has the lowest stress for each number of landmarks. **(Right)** Stress vs. size of the approximation in memory. sBHA outperforms other methods, using less memory to attain the same stress.

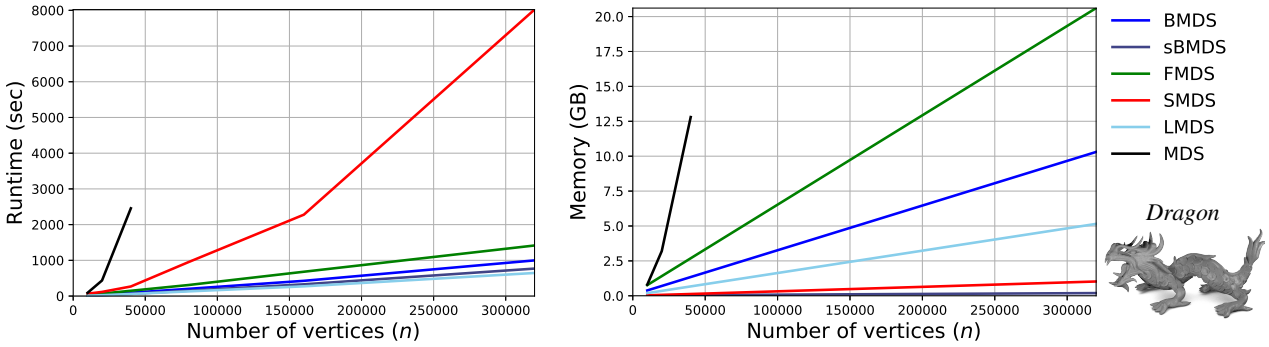

Figure 7. Scalability of BMDS and sBMDS with $p_{row} = 50$ (our methods) versus other approximations SMDS, FMDS, and LMDS, as well as normal MDS. Each algorithm was used to embed geodesic distance matrices for *Dragon* meshes with $n = 5000 \ldots 320,000$ vertices into 3 dimensions. **(Left)** Runtime versus mesh size. **(Right)** Memory usage versus mesh size.

4.3. Extremely large mesh

To demonstrate that our sparse methods, sBHA and sBMDS, can be applied to extremely large problems we used sBMDS to obtain the canonical form for *Dragon1800k*, a mesh with 1.8 million vertices. Using $l = 50,000$ landmarks the approximate geodesic distance matrix is only 20.9 GB, three orders of magnitude smaller than the full 26 TB matrix. The canonical form clearly distinguishes the extremities of the mesh (Figure 1), suggesting that sBMDS was successful at recovering the canonical form.

5. Conclusions

The sBHA method described here offers a sparse alternative to approximation methods like FMDS, SMDS, and Nyström. Sparsity allows sBHA to be extremely efficient in both memory and evaluation time. Results show that sBHA can be used for the same applications and can yield equally accurate approximations using 20x less memory than other methods. The greatest value of sBHA is for scaling to very large problems where the accuracy of current methods is limited by memory.

One key improvement to (s)BHA would be a way to automatically select the number of landmarks that gives an efficient but accurate approximation. Another way forward could be to split the difference between FMDS and sBHA by saving more of the precomputed distances than sBHA does but fewer than FMDS. Ultimately it will also be important to study the theoretical properties of this method and provide bounds on approximation error. Nevertheless, these results show that sBHA can be extremely beneficial in some settings, and is immediately applicable.

References

[1] Y. Aflalo and R. Kimmel. Spectral multidimensional scaling. *Proceedings of the National Academy of Sciences*, 110(45):18052–18057, 2013. 1, 2, 3, 5, 6

[2] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Calculus of nonrigid surfaces for geometry and texture manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):902–913, Sept 2007. 6, 7

[3] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference*, volume 2008, pages 129–136, 2008. 6

[4] C. Cortes, M. Mohri, and A. Talwalkar. On the impact of kernel approximation on learning accuracy. In Y. W. Teh and M. Titterington, editors, *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pages 113–120, 2010. 1

[5] K. Crane, C. Weischedel, and M. Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics*, 32(3):10, 2013. 2, 5, 6

[6] M. Desbrun, E. Kanso, and Y. Tong. Discrete differential forms for computational modeling. In *Discrete differential geometry*, pages 287–324. Springer, 2008. 2

[7] P. Drineas and M. W. Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.*, 6:2153–2175, Dec. 2005. 1

[8] M. Elad. *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer Publishing Company, Incorporated, 1st edition, 2010. 4

[9] J. S. Gao, A. G. Huth, M. D. Lescroart, and J. L. Gallant. Pycortex: an interactive surface visualizer for fMRI. *Frontiers in Neuroinformatics*, 9(September):1–12, 2015. 6

[10] G. Hinton and S. Roweis. Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840, 2002. 2

[11] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, May 1985. 3

[12] S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the nyström method. *J. Mach. Learn. Res.*, 13:981–1006, Apr. 2012. 1

[13] M. Li, J. T. Kwok, and B.-L. Lu. Making large-scale nyström approximation possible. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 631–638, Haifa, Israel, Jun. 2010. Omnipress. 1

[14] Y. Lipman, R. M. Rustamov, and T. A. Funkhouser. Biharmonic distance. *ACM Transactions on Graphics (TOG)*, 29(3):27, 2010. 2

[15] R. Litman and A. M. Bronstein. Spectrometer: Amortized sublinear spectral approximation of distance on graphs. In *Int. Conf. on 3D Vision (3DV) 2016*, pages 499–508, Oct. 2016. 5

[16] Y. Liu, B. Prabhakaran, and X. Guo. Point-based manifold harmonics. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1693–1703, 2012. 2

[17] E. J. Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930. 1, 5, 6

[18] B. N. Parlett. *The symmetric eigenvalue problem*, chapter 13 Lanczos Algorithms, pages 287–321. SIAM, 1998. 5

[19] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 40–44, Nov. 1993. 4

[20] M. Reuter, S. Biasotti, D. Giorgi, G. Patanè, and M. Spagnuolo. Discrete LaplaceBeltrami operators for shape analysis and segmentation. *Computers & Graphics*, 33(3):381–390, Jun. 2009. 2

[21] G. Shamai, Y. Aflalo, M. Zibulevsky, and R. Kimmel. Classical scaling revisited. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2255–2263, 2015. 1, 2, 3, 4, 5, 6, 7

[22] A. Talwalkar, S. Kumar, M. Mohri, and H. Rowley. Large-scale svd and manifold learning. *Journal of Machine Learning Research*, 14:3129–3152, 2013. 1

[23] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, Dec. 2000. 3, 6

[24] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. 4

[25] S.-Q. Xin, X. Ying, and Y. He. Constant-time all-pairs geodesic distance query on triangle meshes. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 31–38, New York, NY, USA, 2012. 5