Project in Geometric Computer Vision

By Dekel Matalon

200686756

Sphere Embedding

Problem Definition:

Were given distances of **k** vertices from each other → $D \in R^{k \times k}$

Where $(D)_{ij}$ is the distance between vertices i and j.

We would like to embed all vertices in a **d**-dimensional sphere where the geodesic distance between every 2 vertices on the sphere is close as possible to the distance we're given in matrix D.

i.e, for each vertex we want to find d+1 Euclidean coordinates

$$\{z_1, z_2, \dots, z_d, z_{d+1}\} \in R^{d+1}$$

with the constraint of keeping constant distance **r** from the origin:

$$\sum_{i=1}^{n} {z_i}^2 = r^2$$

Thus, all the embedded points live on **d** dimensional sphere with radius **r**.

Relation between the geodesic distances and the embedded points locations:

Let θ be the angle between the vectors connecting the origin to the points. (We will denote those vectors as $x_1, x_2, \dots, x_k$).

θ is calculated using scalar product between those vectors x_i, x_j

$$cos\theta |x_i| |x_j| = x_i \cdot x_j$$

$$cos\theta \, r^2 = x_i \cdot x_j$$

$$\theta = \arccos(\frac{x_i \cdot x_j}{r^2})$$

The geodesic distance on the sphere is

$$(D_s)_{ij} = \theta r$$

$$(D_s)_{ij} = r \cdot \arccos(\frac{x_i \cdot x_j}{r^2})$$

Using Matrix notations, we get

$$D_s = \mathrm{r} \cdot \arccos\left(\frac{ZZ^t}{r^2}\right)$$

Is a relation between the geodesic distances D_s, and the vertices locations matrix Z.

Embedding Process:

We want to minimize the difference between the given distances and the embedded distances.

So, we present the following Embedding error:

$$E = \|D_s(Z) - D\|_F^2$$

where $D_s(Z) = \mathrm{r} \cdot \arccos\left(\frac{ZZ^t}{r^2}\right)$

Minimizing the error:

$$Z^{opt} = \underset{Z}{argmin} \|D_s(Z) - D\|_F^2$$

Using steepest descent process:

$$Z^{k+1} = Z^k - \lambda \frac{\partial E}{\partial \Phi}$$

where Φ is the parametrization coordinates of the sphere.

$\frac{\partial E}{\partial \Phi}$ Approximated by $\frac{\partial E}{\partial \mathrm{Z}}$ Projected back to the sphere in every step. $\frac{\partial E}{\partial \mathrm{Z}} / \left\|\frac{\partial E}{\partial \mathrm{Z}}\right\|$

λ – must change its value during the process so it will converge.

We will use Line Search method:

raising λ when the embedding error goes up

and lowering it otherwise.

Steepest descent initialization:

We would like to choose Z^0 for the first step of the steepest descent algorithm.

I didn't find any smart way to initialize it, so I used intuition –

I used classical-MDS in order to embed D to a d+1 dimensional Euclidean space, and then projected them to the d-dimensional sphere.

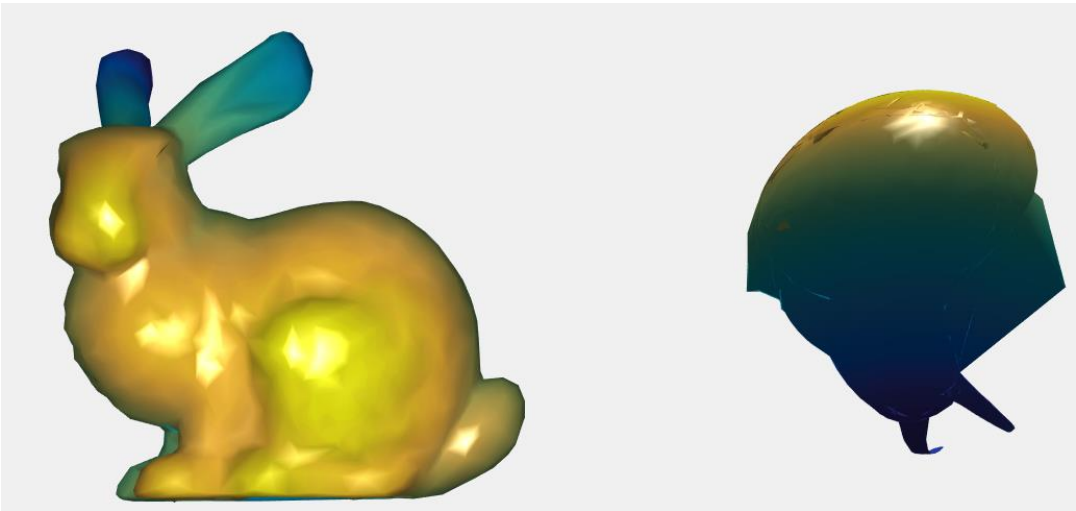
Last thing is to calculate the gradient:

$$\frac{\partial E}{\partial \mathrm{Z}} = 2(D_s - D)(\nabla D_s)^t$$

$$\nabla D_s = \frac{2Z^t}{r^3\sqrt{r^4 - (ZZ^t)^{.2}}}$$

Where " $.^2$ " is elementwise-square of the entries.

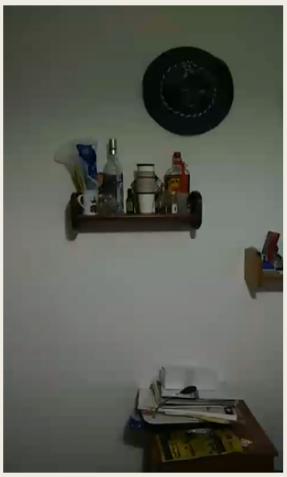
Embedding the bunny:

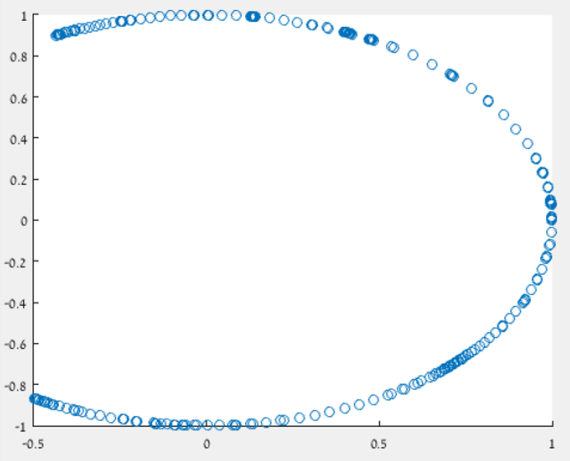


This process can embed the bunny, but present some artifacts.

2 Applications which are using the depicted process:

Application 1





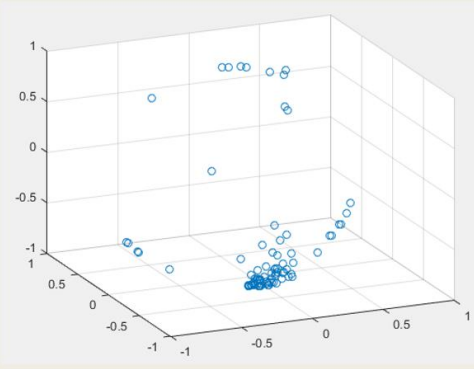
I was taking a video, spinning horizontally in my room. (about 270 degrees)

I built a "frame graph" on which every vertex is a frame and connected to 8 adjacent neighbor frames with distance on the edges:

I was trying to calculate distance using measurements of the affine transformation between the frames (like amount of translation or scale) but found out that it worked better with just calculating distance of histograms.

I used the spherical embedding to get circular angles of the view by embedding graph distances to a 1 dimensional sphere.

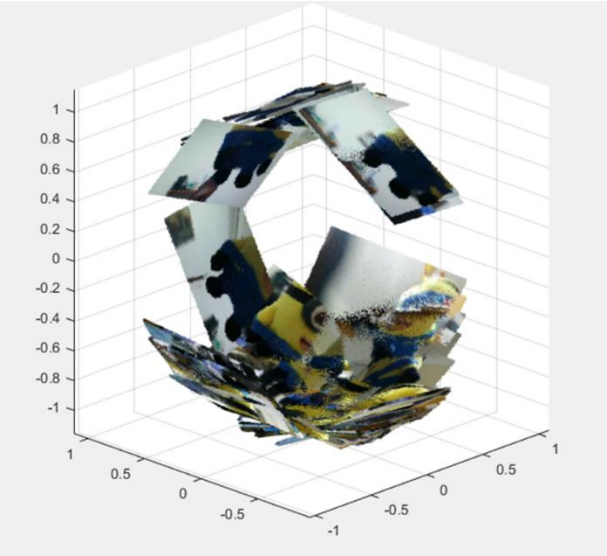
Application2

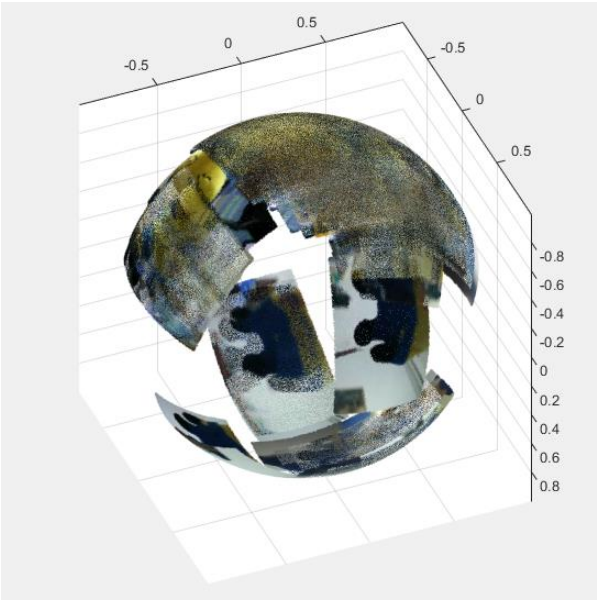


In the following application, I was taking a video of an object in the center of my room from various of angles.

I calculated the distance between every 2 frames (as in the previous application, but now I calculated a full graph)

And embedded it to 2-dimensional sphere.





Those 2 pics presenting the frames sent to their locations which were chose by my spherical-mds alg. And the projected voxels to the sphere.

Using more robust methods for handling numerical errors, better initialization and better scanning equipment, this application could have reconstructed the object as CT scanners do.

Matching shapes without shape reconstruction – photometric stereo based SGMDS

We're given **M** pictures with **N** pixels such that each one of them is taken using different light direction.

In practice, we get $I_{N\times M}$ where $I_{k,l}$is the intensity level for the k'th pixel, using the l'th light source.

and $L_{M\times 3}$, the location of the light sources.

Using photometric stereo equation $V = L^{\dagger} \cdot I^{t}$

Where $L^{\dagger}$ is the Moore-Penrose pseudoinverse of L, and V are the responses that are composed into:

Normals $N_i = \frac{V_i}{\|V_i\|}$ and Albedo levels $\rho_i = \|V_i\|$.

So now when we have the normal for each pixel, we can derive the metric tensor of our shape using the original grid (X,Y) of the picture as our parametric domain.

Using numeric derivatives, we get:

Step in X axis: $D_x^+ S = (1,0, D_x^+ Z)$

Step in Y axis: $D_y^+ S = \left(0,1, D_y^+ Z\right)$

The normal's direction n_d in any point is given by the cross product of the derivatives $n_d = D_x^+ S \times D_y^+ S = (-D_x^+ Z, -D_y^+ Z, 1)$

So we get Z derivatives by dividing normal by their 3rd column values: $Z_x(i) = \frac{n_1(i)}{n_3(i)}$ $Z_y(i) = \frac{n_2(i)}{n_3(i)}$ for every pixel i,

Where $n = (n_1, n_2, n_3)$ are the normal coordinates.

Now we derive the Z values again to get Z_{xx}, Z_{xy}, Z_{yy}

And the **metric tensor:**

$$G = \begin{bmatrix} Z_{xx} & Z_{xy} \\ Z_{xy} & Z_{yy} \end{bmatrix}$$

Next, we would like to calculate the distances between the pixels.

An interesting way of doing this is designing a fast-marching method on the grid, in which the costs of moving up/down are $|Z_x|$ and moving left/right are $|Z_y|$.

So we got the distances of the shape: D, and now we can perform a comparison between our object, and other objects using the MDS family, so the comparison is isometry invariant.

For a simple MDS of our distances maps, we can compare them using the second moments of the canonical forms.

Notice that this whole process we didn't actually have to reconstruct the 3D shape of our object.

Next I will describe the usage of GMDS & SGMDS:

Embed our distance map to the others, without constructing their normal forms.

GMDS:

The generalized MDS using the Gromov-Hausdorff distance that describe distance between 2 metric spaces (S, d_S) and (Q, d_Q):

$$d_{\mathrm{GH}}(S, Q) = \frac{1}{2} \min_{C} \max_{\substack{(s,q) \in C \\ (s',q') \in C}} \left| d_S(s, s') - d_Q(q, q') \right|$$

Where C is a set of matched points between the 2 surfaces.

- Every point has a counterpart in C.

We use function $p: S \times Q \rightarrow [0,1]$ as a soft correspondence function, indicating the level of correspondence between the points.

We will denote:

- ψ as the map from S to Q
- $\phi = \psi^{-1}$ is the inverse map, mapping points from Q to S.
- da_s and da_q are area elements of the points.

The distance maps of S and Q are:

$$d_S(\phi(q), s) = \int_S d_S(s', s) p(q, s') da_{s'}. \qquad d_Q(q, \psi(s)) = \int_Q d_Q(q, q') p(q', s) da_{q'}.$$

The distance of the two maps with the correspondence p:

$$\int_{S,Q} \left(d_S(\phi(q), s) - d_Q(q, \psi(s)) \right)^2 da_s da_q$$

We will plug the 2 distance maps inside the last equation, and discrete it, and demand minimality of the term:

$$\min_{\mathbf{P}} \|\mathbf{P}\mathbf{A}_S\mathbf{D}_S - \mathbf{D}_Q\mathbf{A}_Q\mathbf{P}\|_{S,Q}$$

s.t.

$$\mathbf{P}\mathbf{A}_S\mathbb{1} = \mathbb{1},$$
$$\mathbf{P}^T\mathbf{A}_Q\mathbb{1} = \mathbb{1},$$

$$\|\mathbf{F}\|_{S,Q} = \operatorname{trace}\left(\mathbf{F}^T\mathbf{A}_Q\mathbf{F}\mathbf{A}_S\right)$$

Is a weighted Forbenius norm

Where P are the correspondence levels (as function p), and A are the area elements.

Its solution can be done using 3 gradient descent methods to repeatedly minimize the stress of embedding S on Q, the stress of embedding Q on S, and the mutual correspondence.

Conclusion: we can compare between S and Q using the embedding error of this process.

<u>SGMDS:</u>

SGMDS is designed to lower the dimensionality of the permutation matrix P, using the spectral decomposition of the Laplace Beltrami of the surfaces.

We get Φ_S and Φ_Q the eigenvectors of the spectral decomposition for surfaces S and Q.

For the correspondence function p, we can use projection estimation on Φ_S vector field as follows:

$$p(s,q) = \sum_i \left\langle p(s,q), \phi_i^S(s) \right\rangle_S \phi_i^S(s) = \sum_i \alpha_i^S(q)\phi_i^S(s)$$

Where α_i^S is the coefficient of the eigenvector ϕ_i^S.

Now we will project those coefficients over Φ_Q:

$$\alpha_i^S(q) = \sum_j \left\langle \alpha_i^S(q), \phi_j^Q(q) \right\rangle_Q \phi_j^Q(q) = \sum_j \alpha_{ij}\phi_j^Q(q).$$

By assigning it to the previous equation we get:

$$p(s,q) = \sum_i \sum_j \alpha_{ij}\phi_j^Q(q)\phi_i^S(s)$$

In matrix notation: $\mathbf{P} = \mathbf{\Phi}_Q \boldsymbol{\alpha} \mathbf{\Phi}_S^T$

where α is the coefficients matrix of the two projections.

Now that we have the Permutation matrix P, we can solve the correspondence problem by projecting the distances matrices to the eigenspaces as well:

$\boldsymbol{\beta}_S = \boldsymbol{\Phi}_S^T \mathbf{A}_S \mathbf{D}_S \mathbf{A}_S \boldsymbol{\Phi}_S$ where $\tilde{\mathbf{D}}_Q \mathbf{A}_Q \mathbf{P} = \boldsymbol{\Phi}_Q \boldsymbol{\beta}_Q \boldsymbol{\alpha} \boldsymbol{\Phi}_S^T$

And continue solving like GMDS as it reduced to spectral values:

$$\|\mathbf{P}\mathbf{A}_S \tilde{\mathbf{D}}_S - \tilde{\mathbf{D}}_Q \mathbf{A}_Q \mathbf{P}\|_{S,Q} = \|\boldsymbol{\alpha}\boldsymbol{\beta}_S - \boldsymbol{\beta}_Q \boldsymbol{\alpha}\|_F^2$$

The degradation of the mapping is reduced more quickly for using the eigenvectors which are corresponding to higher eigenvalues because of they hold more information.

Thus the degradation of the mapping of the surface is reduced quickly at the beginning (when we used eigenvectors that hold more information) and slowly as we use a lot of eigenvectors.

3D reconstruction from a rotating camera

Given a point cloud of **n** points in R^3, centered about the origin, and camera that looks towards the origin and rotating about the Z axis with distance $R \gg r$ where r is the diameter of the point cloud, and R is the radius of the camera's circular trajectory, we are asked to reconstruct the point cloud using the video frames captured by the camera.

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{Calibration Matrix K}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The 3d points are projected to the frame as follows:

On the right, are the homogenous coordinates of a point in the point cloud

And on the left, are the homogenous coordinates of a point on the frame.

The global scaling get its value by the values of the focal point (fx,fy).

R – is a rotation matrix of the object (actually the inverse of the rotation of the camera)

And **t** is the translation. Here t = 0 because the object isn't translating, and the camera is just rotating.

Let (v_1, v_2) be 2 points in the point cloud, and (p_1, p_2) their correspondences on the frame. We get:

$$\|K(R_1 v_1 - R_2 v_2)\|$$

Now, we choose the camera location to be the average angle described in those 2 rotation matrices (R_1, R_2) to be the initial angle view of the camera and add a rotation term $R_a$of the camera around its own axis for the two points to be aligned vertically in the frame.

So now
$\|K(R_a v_1 - R_a v_2)\| = \|KR_a(v_1 - v_2)\| = \|K(v_1 - v_2)\| = \|p_1 - p_2\|$

Where R_a is reduced after the second equal sign because rotation matrices are orthonormal and therefore their norm equals 1.

So we got – the global scale is $\|K\|_F$.

On the projected frames, we get that this distance is given when the projected distance in the frame is maximal.

A suggested process is to evaluate the distance in the video frames for every 2 points in the cloud, and build a distances matrix $\widehat{D}$, and measure the error by taking the real distances D, and perform isometric comparison between the two distances matrices using MDS.

This should give pretty good results when we overlook occlusions.

In a case of occlusions, we can get the optimal values in $\widehat{D}$ only if the mesh is convex, and we are not bounded to a plane with our camera. simply because the occlusion will interfere with discovering the maximal distance between points, for example – points that are not appearing even in 1 frame with each other. The two conditions (convexity and unbounded to a plane) can settle it. Because there will always be an angel from where we can find the maximal projected distance between 2 points.

We can find occluded points using computer graphics Z-buffer algorithms – if the pixel is used with a lower z value, then we mark the point as an occluded point.

Ways to deal with reconstructing a scanned object with occlusions:

1. Present a threshold for removing points with too many occluded frames.
2. Try to guess distance of points that are occluded in too many frames by multiplying their distances from other points
3. For points which are not live together in any frame – find distance with shortest path algorithm.