Neural Jacobian Fields

Pramook Khungurn

April 18, 2025

This note is written as I read the paper "Neural Jacobian Fields: Learning Intrinsic Mappings of Arbitrary Meshes" by Aigerman et al. [AGK+22].

1 Preliminary

- This paper presents a framework to generate "piecewise linear mappings" of meshes with a neural network.
- In this note, we treat points in $\mathbb{R}^d$ as *row vectors*.
  - So, if $\mathbf{x} \in \mathbb{R}^3$, then
  $$\mathbf{x} = \begin{bmatrix} x^1 & x^2 & x^3 \end{bmatrix}.$$
  - Notice that we use superscripts to denote components of a vector. This is because we will use subscripts for other things.
  - This means that a linear transformation from $\mathbb{R}^3$ to $\mathbb{R}^2$ is encoded with a 3×2 matrix instead of 2×3. This matrix is multiplied to the right instead of to the left.
- This makes it quite convenient to represent a batch of N points and matrices.
  - A batch of N points in $\mathbb{R}^3$ is represented by a matrix $\mathbf{P}$ of size $N \times 3$.
  - A linear transformations from $\mathbb{R}^3$ to $\mathbb{R}^2$ is represented by a matrix M fo size 3×2.
  - If we wish to apply this linear transformation to all points in the batch, we just compute $\mathbf{P}M$.

1.1 Meshes

- Let us formally define what a mesh is.
  - We are given a triangular mesh $\mathcal{S}$ in $\mathbb{R}^3$ with vertices $\mathbf{V}$ and triangles $\mathbf{T}$.
  - The ith triangle in $\mathbf{T}$ is denoted by $\mathbf{t}_i$.
  - The **tangent space** of $\mathbf{t}_i$ is the linear space orthogonal to its normal. It is denoted by T_i.
    - Bascially, this is the plane in $\mathbb{R}^3$ that the triangle lies in.
  - A **frame** of triangle $\mathbf{t}_i$ is an oriented orthonomal basis fo its tangent space. It is denoted by $\mathcal{B}_i$.
    - We don't actually know in which direction the frame points to.
    - I guess the natural direction is the direction of the normal vector.
- Let's talk about how the tangent space is represented numerically.
  - Let us say that the three vertices of $\mathbf{t}_i$ have indices j, k, and l.

- So, the triangle vertices are $\mathbf{v}_j$, $\mathbf{v}_k$, and $\mathbf{v}_l$.
- Let us also say that we designate $\mathbf{v}_j$ is the origin of tangent space T_i.
- So, we have that $T_i = \{a(\mathbf{v}_k - \mathbf{v}_j) + b(\mathbf{v}_l - \mathbf{v}_j) : a, b \in \mathbb{R}\}$.
  - Here, we see that T_i is a set of vectors because $\mathbf{v}_k - \mathbf{v}_j$ and $\mathbf{v}_l - \mathbf{v}_j$ are vectors.
- $\mathcal{B}_i$ is an orthonomal basis of T_i. So, we may say that it is a 2×3 matrix:

$$\mathcal{B}_i = \begin{bmatrix} \boldsymbol{\beta}_{i,1} \\ \boldsymbol{\beta}_{i,2} \end{bmatrix}$$

  where $\boldsymbol{\beta}_{i,1}, \boldsymbol{\beta}_{i,2} \in \mathbb{R}^3$ such that $\|\boldsymbol{\beta}_{i,1}\| = \|\boldsymbol{\beta}_{i,2}\| = 1$ and $\boldsymbol{\beta}_{i,1} \cdot \boldsymbol{\beta}_{i,2} = 1$. Last but not least, $T_i = \{a\boldsymbol{\beta}_{i,1} + b\boldsymbol{\beta}_{i,2} : a, b \in \mathbb{R}\}$.
- Because $\mathcal{B}_i$ is an orthonormal basis of T_i, we have that, for any vector $\mathbf{v} \in T_i$, we can find the coordinates of $\mathbf{v}$ with respect to $\mathcal{B}_i$ as follows:

$$\mathbf{v}\text{'s coordinate} = \mathbf{v}\mathcal{B}_i^T.$$

- Moreover, for any $\mathbf{x} \in \mathbb{R}^3$, we have that

$$(\mathbf{x} - \mathbf{v}_j)\mathcal{B}_i^T\mathcal{B}_i$$

  is the projection of $\mathbf{x} - \mathbf{v}_j$ onto the plane of the triangle $\mathbf{t}_i$. As a result,

$$\mathbf{v}_j + (\mathbf{x} - \mathbf{v}_j)\mathcal{B}_i^T\mathcal{B}_i$$

  is the point on the plane of the triangle $\mathbf{t}_i$ that is the closest to $\mathbf{x}$.

1.2 Linear Piecewise Mapping

- A **map** or a **mapping**, in our context, is a function $\boldsymbol{\Phi} : \mathbb{R}^3 \to \mathbb{R}^3$.
- We can think of $\boldsymbol{\Phi}$ as being composed of three maps of signature $\mathbb{R}^d \to \mathbb{R}$, one for each component of the output. We can denote the components by Φ^1, Φ^2, and Φ^3. In this way, we have that

$$\boldsymbol{\Phi}(\mathbf{x}) = \begin{bmatrix} \Phi^1(\mathbf{x}) & \Phi^2(\mathbf{x}) & \Phi^3(\mathbf{x}) \end{bmatrix}$$

- A map $\boldsymbol{\Phi}$ is a **piecewise linear mapping** with respect to a mesh $\mathcal{S}$ if the restriction of $\boldsymbol{\Phi}$ to any triangle $\mathbf{t}_i$, denote $\boldsymbol{\Phi}|_{\mathbf{t}_i}$ is affine.
  - This is the most common family of maps used when considering mappings of meshes.
- A piecewise linear mapping is uniquely defined by assigning a new position to one of the vertices.
  - In other words, let $\mathbf{v}_j$ denote the position of the jth vertex. Suppose that we know $\boldsymbol{\Phi}_j$, the image of $\mathbf{v}_j$ under the mapping, for all j.
  - Then, any point inside the triangle is sent to the interpolation of the $\boldsymbol{\Phi}_j$s with barycentric coordinates.
  - As a result, we may encode $\boldsymbol{\Phi}$ as a matrix of the same dimension as $\mathbf{V}$.
  - In this case, $\boldsymbol{\Phi}$ is a matrix of size $|\mathbf{V}| \times 3$.
- Another way to look at the piecewise linear mapping is to see it as a linear combination of basis functions.
  - The basis function we use in this case is called the **hat function**.

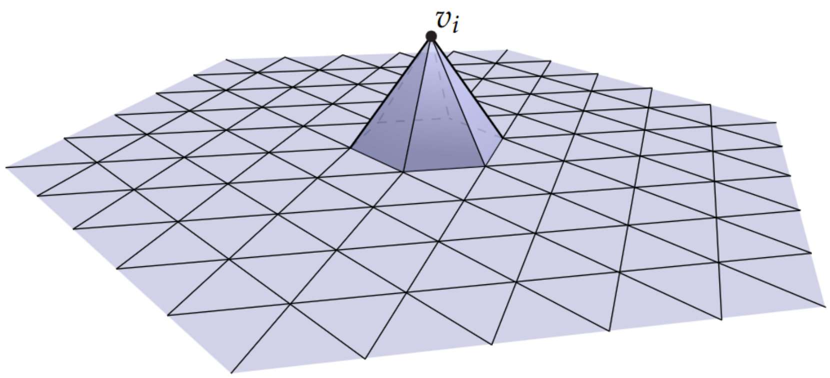

Figure 1: The hat function.

* For each Vertex j, we define the hat function φ_j to be a piecewise linear function on each triangle where $\varphi_j(\mathbf{v}_j) = 1$ and $\varphi_j(\mathbf{v}_k) = 0$ for all $k \leq j$. See Figure 1.

– So, the piecewise linear mapping $\mathbf{\Phi}$ can be written as:

$$\mathbf{\Phi}(\mathbf{x}) = \sum_{j=1}^{|\mathbf{V}|} \mathbf{\Phi}_j \varphi_j(\mathbf{x}).$$

– Remember that $\mathbf{\Phi}$ has three component functions. Each function can also be written as a linear combination of the hat functions as well.

$$\Phi^1(\mathbf{x}) = \sum_{j=1}^{|\mathbf{V}|} \Phi^1_j \varphi_j(\mathbf{x}), \qquad \Phi^2(\mathbf{x}) = \sum_{j=1}^{|\mathbf{V}|} \Phi^2_j \varphi_j(\mathbf{x}), \qquad \Phi^3(\mathbf{x}) = \sum_{j=1}^{|\mathbf{V}|} \Phi^3_j \varphi_j(\mathbf{x})$$

1.3 Gradients

- Let us define a **scalar field** to be a function of signature $\mathbb{R}^3 \to \mathbb{R}$.
- A scalar field is **piecewise linear** if, when restricted to each triangle in a mesh, it is an affine function.
- We also know that any piecewise linear scalar field can be written as a linear combination of hat functions.
- We have that, for a piecewise linear mapping $\mathbf{\Phi}$, its component functions Φ^1, Φ^2, and Φ^3 are piecewise linear scalar fields.
- Consider a piecewise linear scalar field f. It is meaningful to talk about its gradient

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \nabla_1 f(\mathbf{x}) \\ \nabla_2 f(\mathbf{x}) \\ \nabla_3 f(\mathbf{x}) \end{bmatrix}.$$

Here, because a vector is a row vector, so a gradient should be a column vector.

- Let $f|_{\mathbf{t}_i}$ denote f restricted to Triangle $\mathbf{t}_i$. From definition, $f|_{\mathbf{t}_i}$ is an affine function. As a result, $\nabla f|_{\mathbf{t}_i}$ is a constant vector. In particular,

$$\nabla f|_{\mathbf{t}_i} = (f_k - f_j)\frac{\mathbf{v}_j - \mathbf{v}_l}{A(\mathbf{t}_i)} + (f_l - f_j)\frac{\mathbf{v}_k - \mathbf{v}_j}{A(\mathbf{t}_i)}$$

  where j, k, l are indices of the vertices of $\mathbf{t}_i$, and $A(\mathbf{t}_i)$ is the area of $\mathbf{t}_i$.

- Because f can be encoded as a $|\mathbf{V}| \times 1$ matrix, we can encode ∇f with a $3|\mathbf{T}| \times 1$ matrix.
- ∇ is a linear operator, so it can be encoded with a $3|\mathbf{T}| \times |\mathbf{V}|$ matrix.
- The `grad` function from the libigl library [JP25] computes this matrix.

1.4 Poisson Problem on a Mesh

- Recall that a **Poisson problem** is the following mathmatical problem.
  - We are given a domain $\Omega \subseteq \mathbb{R}^3$.
  - We are also given a vector field $\mathbf{g} : \Omega \to \mathbb{R}^3$.
  - We are to find a scalar field $f : \Omega \to \mathbb{R}$ that minimizes the following energy:

$$\int_\Omega \|\nabla f(\mathbf{x}) - \mathbf{g}(\mathbf{x})\|^2 \,\mathrm{d}\mathbf{x}.$$

  In other words, we are asked to find a scalar field whose gradient fits the given vector field the best.

- It can be shown that the optimal function f^* satisfies the following **Poisson equation**:

$$\Delta f^*(\mathbf{x}) = \nabla \cdot \mathbf{g}(\mathbf{x})$$

  for all $\mathbf{x} \in \Omega$. Here, Δ is the Laplace operator:

$$\Delta f(\mathbf{x}) = \nabla \cdot \nabla f(\mathbf{x}) = \sum_{i=1}^{3} \frac{\partial^2 f(\mathbf{x})}{(\partial x^i)^2} = \sum_{i=1}^{3} \nabla_i \nabla_i f(\mathbf{x}) = \sum_{i=1}^{3} \nabla_{i,i} f(\mathbf{x}),$$

  and $\nabla\cdot$ is the divergence operator

$$\nabla \cdot \mathbf{g}(\mathbf{x}) = \sum_{i=1}^{3} \frac{\partial \mathbf{g}^i(\mathbf{x})}{\partial x^i} = \sum_{i=1}^{3} \nabla_i g^i(\mathbf{x})$$

- We note that f^* is unique up to a constant shift. This because

$$\nabla(f + c) = \nabla f$$

  for all constant $c \in \mathbb{R}$.

- On a mesh, the Poisson problem takes on a different guise.
  - We are given a mesh $\mathcal{S}$.
  - We are given a vector field $\mathbf{g}$, that is supposed to encode the gradient of a piecewise linear scalar field.
    * This is encoded as a $3|\mathbf{T}| \times 1$ matrix.

– We are supposed to find a piecewise linear scalar field f that minimizes the following energy

$$\int_{\mathcal{S}} \|\nabla f(\mathbf{x}) - \mathbf{g}(\mathbf{x})\|^2 \, \mathrm{d}\mathbf{x} = \sum_{i=1}^{|\mathbf{T}|} A(\mathbf{t}_i) \|(\nabla f)_i - \mathbf{g}_i\|^2.$$

where $(\nabla f)_i$ and $\mathbf{g}_i$ denote the values of the corresponding vector fields at triangle $\mathbf{t}_i$.

- We can show that, the optimal piecewise linear scalar field f^* must satisfy the equation

$$\nabla f^* = \nabla \cdot \mathbf{g}.$$

Here, $\nabla\cdot$ is a matrix that represents the divergence operator. It is a $|\mathbf{V}| \times 3|\mathbf{T}|$ matrix that is equal to

$$\nabla \cdot = \nabla^T \mathcal{A}$$

where $\mathcal{A}$ is a $3|\mathbf{T}| \times 3|\mathbf{T}|$ diagonal matrix that contains the area $A(\mathbf{t}_i)$ of the triangle $\mathbf{t}_i$ in the rows associated with the triangle $\mathbf{t}_i$. Δ is the $|\mathbf{V}| \times |\mathbf{V}|$ matrix that represents the Laplacian operator. It is equal to

$$\Delta = \nabla^T \mathcal{A} \nabla.$$

It can be shown that this is equal to the "cotangent Laplacian" that is widely used in discrete differential geometry [Cra25].

- So, we can find f^* by computing

$$f^* = \Delta^{-1} \nabla^T \mathcal{A} \mathbf{g}.$$

- However, the above equation doesn't quite work because Δ is a not a full-rank matrix because, if you add the columns up, you will get a zero vector. So, any linear solver would complain.

- The way to solve this is to do the following.

  1. Trim Δ down to a $(|\mathbf{V}|-1) \times (|\mathbf{V}|-1)$ matrix and $\Delta^T \mathcal{A}$ by lobbing off its 1st row and 1st column.
  2. Trim $\nabla^T \mathcal{A}$ to a $(|\mathbf{V}| - 1) \times 3|\mathbf{T}|$ by lobbing off its 1st row.
  3. Compute $f^* = \Delta^{-1} \nabla^T \mathcal{A} \mathbf{g}$ with a standard linear solver.
  4. The output is a $|\mathbf{V} - 1| \times 1$ matrix. To get a full response, just add an extra row of 0 at the beginning.

  Note that, for the last step, 0 is the only one value that would work. (Think about the $(|\mathbf{V}|-1) \times (|\mathbf{V}|-1)$ version of Δ being a submatrix of the full Δ.)

- In other words, given a vector field that is supposed to be the gradient of some piecewise linear scalar field over a mesh, we can find a piecewise linear scalar field whose gradient is the closest to tha vector field by solving a linear equation.

1.5 Jacobians

- We learned earlier that a piecewise linear mapping $\mathbf{\Phi}$ is made up of three piecewise linear scalar fields Φ^1, Φ^2, and Φ^3.

- Let us take a look at the derivative of $\mathbf{\Phi}$, denoted by $\nabla \mathbf{\Phi}$ and commonly called the **Jacobian** of $\mathbf{\Phi}$.

$$\nabla \mathbf{\Phi}(\mathbf{x}) = \begin{bmatrix} \nabla_1 \Phi^1(\mathbf{x}) & \nabla_1 \Phi^2(\mathbf{x}) & \nabla_1 \Phi^3(\mathbf{x}) \\ \nabla_2 \Phi^1(\mathbf{x}) & \nabla_2 \Phi^2(\mathbf{x}) & \nabla_2 \Phi^3(\mathbf{x}) \\ \nabla_3 \Phi^1(\mathbf{x}) & \nabla_3 \Phi^2(\mathbf{x}) & \nabla_3 \Phi^3(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \nabla \Phi^1(\mathbf{x}) & \nabla \Phi^2(\mathbf{x}) & \nabla \Phi^3(\mathbf{x}) \end{bmatrix}.$$

So, the Jacobian of $\mathbf{\Phi}$ is the stack of gradient vectors of its component functions, which are scalar fields over the mesh.

- Again, if we restrict $\mathbf{\Phi}$ to a triangle $\mathbf{t}_i$, we have that $\nabla\Phi^1|_{\mathbf{t}_i}(\mathbf{x})$, $\nabla\Phi^2|_{\mathbf{t}_i}(\mathbf{x})$, $\nabla\Phi^3|_{\mathbf{t}_i}(\mathbf{x})$ are constant vectors, which we will denote by $(\nabla\Phi^1)_i$, $(\nabla\Phi^2)_i$, and $(\nabla\Phi^3)_i$. So,

$$\nabla\mathbf{\Phi}|_{\mathbf{t}_i}(\mathbf{x}) = \begin{bmatrix}(\nabla\Phi^1)_i & (\nabla\Phi^2)_i & (\nabla\Phi^3)_i\end{bmatrix}.$$

  As a result, $\nabla\mathbf{\Phi}|_{\mathbf{t}_i}(\mathbf{x})$ is a constant, which we will denote by $(\nabla\mathbf{\Phi})_i$.

- Thus, we can represent the Jacobian of piecewise linear mapping $\mathbf{\Phi}$ by a $|\mathbf{T}| \times 3 \times 3$ tensor or, to agree with the convention in the previous section, a $3|\mathbf{T}| \times 3$ matrix.

- As a result, given field of 3×3 matrices $\mathbf{M}$ over a mesh, which is represent by assigning a 3×3 matrix to each triangle, we can find a piecewise linar mapping Φ^* such that its Jacobian matrices are closes to $\mathbf{M}$ by computing

$$\mathbf{\Phi}^* = \Delta^{-1}\nabla^T\mathcal{A}\mathbf{M}.$$

  This is equivalent to solving the problems for each component functions $(\Phi^*)^1$, $(\Phi^*)^2$, $(\Phi^*)^3$ independently.

- Let's see what we have so far.
  - If we have a piecewise linear mapping $\mathbf{\Phi}$, we can difinite compute its Jacobian matrices $\mathbf{M} = \nabla\mathbf{\Phi}$, which is an assignment of each triangle to a 3×3 matrix.
  - On the other hand, if we have $\mathcal{M}$, which is a stack of 3×3 matrices where there is one for each triangle, then we can find a piecewise linear mapping Φ^* such that $\nabla\Phi^*$ is the closest to M.

  Moreover, if we do $\mathbf{\Phi} \to \mathbf{M} \to \mathbf{\Phi}^*$, we have that $\mathbf{\Phi}$ and $\mathbf{\Phi}^*$ only differs by a translation.

- As a result, it follows that we can encode a piecewise linear transformation by its Jacobian matrices.
  - Of course, we have to figure out the missing translation somehow, but we will worry about that later.

1.6 Restricting Jacobians to Tangent Spaces

- Recall that Jacobians are linear transformation such that

$$\mathbf{\Phi}(\mathbf{x}+\mathbf{h}) \approx \mathbf{\Phi}(\mathbf{x}) + \mathbf{h}\nabla\mathbf{\Phi}(\mathbf{x})$$

  when $\mathbf{h}$ is small enough.

- When we restrict $\mathbf{\Phi}$ to the triangle $\mathbf{t}_i$, we have that $\mathbf{\Phi}|_{\mathbf{t}_i}$ is an affine function, and so the approximation above becomes exact.

$$\mathbf{\Phi}|_{\mathbf{t}_i}(\mathbf{x}+\mathbf{h}) = \mathbf{\Phi}|_{\mathbf{t}_i}(\mathbf{x}) + \mathbf{h}\nabla\mathbf{\Phi}|_{\mathbf{t}_i}(\mathbf{x}) = \mathbf{\Phi}|_{\mathbf{t}_i}(\mathbf{x}) + \mathbf{h}(\nabla\mathbf{\Phi})_i.$$

- We typically understand the small vector $\mathbf{h}$ as a vector in the tangent space of $\mathbf{x}$. In other words,

$$\mathbf{h} = \mathbf{b}\mathcal{B}_i$$

  where $\mathbf{b} \in \mathbb{R}^2$ and $\mathcal{B}_i$ is the matrix of basis vectors of the tangent space T_i of triangle $\mathbf{t}_i$.

- Thus,

$$\mathbf{\Phi}|_{\mathbf{t}_i}(\mathbf{x}+\mathbf{b}\mathcal{B}_i) = \mathbf{\Phi}|_{\mathbf{t}_i}(\mathbf{x}) + \mathbf{b}\mathcal{B}_i(\nabla\mathbf{\Phi})_i.$$

- We can view $\mathcal{B}_i(\nabla\mathbf{\Phi})_i$ as a restriction of the Jacobian $(\nabla\mathbf{\Phi})_i$ so that it operates on vectors in the tangent space of the triangle $\mathbf{t}_i$. This matrix is a 2×3 matrix.
- In particular, let us say that we have the stack $\mathcal{B}$ of the basis matrices of the tangent spaces, and this is a $|\mathbf{T}| \times 2 \times 3$ tensor. Suppose that $\nabla\mathbf{\Phi}$ is represented by a $|\mathbf{T}| \times 3 \times 3$ tensor. Then, the restrictions of the Jacobian matrices to the tangent spaces are given by

$$\texttt{bmm}(\mathcal{B}, \mathbf{\Phi})$$

  where `bmm` denotes the batch matrix multiplication function, implemented in PyTorch and other deep learning frameworks.

2 Method

2.1 The Big Picture

- The goal of the paper is to create a neural network that consumes a mesh and produces a piecewise linear mapping of the mesh.
- We know what the output looks like. It is $\mathbf{\Phi}$, which assigns each vertex of the mesh to a point in $\mathbb{R}^3$. However, we will make our life easier by saying that the position of the new mesh can be arbitrary in space; i.e., we remove the translation of the mesh.
- With the relaxed requirement, we can encode a piecewise linear mapping with the field of 3×3 Jacobian matrices over the triangles.
- The paper chooses to predict the Jacobian field instead of the new positions of each vertex. Once the Jacobian field has been predicted, one can get a position assignment by solving Poisson's equation.
- The main goal of the paper is to make sure that **how the prediction gets done should be agnostic to the triangulation of the mesh.**
  - This means that the triangles in the mesh should not be processed as a whole.
- As a result, the paper proposes a network that processes each triangle independently.
  - It takes in information regarding the centroid of the triangle.
    - Position.
    - Normal vector.
    - Wave-Kernel signature (WKS) [ASC11] of size 50, computed by averaging the WKSs of the three triangle vertices.
  - In order for the network to have access to global information about the mesh, it also takes in a **global code $\mathbf{z}$**, which is the same for all triangles.
    - We will talk about the global codes later.
  - The network then should predict a 3×3 matrix, which serves as the Jacobian matrix of the predicted piecewise linear mapping.
- The global code can contains information about many things.
  - The shape of the input mesh.
    - The paper uses the encoding by PointNet [QSMG17].
      - We sample 1024 points on the mesh.

      - We create a tensor of point information. Each entry corresponds to a point and has the following fields: (1) the point's 3D position, (2) the point's normal vector, and (3) the point's Wave-Kernel signature [ASC11] of size 50.
      - We feed the above tensor to PointNet to get an encoding.
  - The shape of the output mesh. This can be encoded in the same way as the shape of the input mesh.
  - It can also incorporate conditioning information $\mathbf{y}$ such as the desired pose of the output mesh, the joint angles of the skeleton, desired position to which mesh should bend to, and so on.

- So, in the end, we need to train the following networks
  - A **mesh encoder** E that maps the shape of the input mesh and other conditioning information (shape of the output mesh, desired pose, etc.) to a **global code** $\mathbf{z}$.
  - A **mapper** M that takes in the global code $\mathbf{z}$ and centroid features $\mathbf{c}_i$ and output the Jacobian of that triangle J_i.

- Note that the above two networks do not have to know how many triangles there are in the mesh or how the triangles are connected.
  - The mesh encoder only process samples on the surface of the mesh.
  - The mapper processes each triangles independently. It only knows the centroid of the triangle, not the vertices that the triangles are connected to.

2.2 The Specifics

- The training data is a collection of tuples $\{\mathcal{S}, \mathbf{\Psi}, \mathbf{y}\}$ where
  - $\mathcal{S}$ is a mesh, which has the following information.
    * The vertex positions $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \dots)^T$.
    * The triangles $\mathbf{T} = (\mathbf{t}_1, \mathbf{t}_2, \dots)^T$.
    * Information about surface samples $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2, \dots)^T$ as required by PointNet.
    * Information about centroids $\mathbf{C} = (\mathbf{c}_1, \mathbf{c}_2, \dots)^T$.
    * The matrix of its gradient operator ∇.
    * Its mass matrix $\mathcal{A}$.
    * The matrix of its divergence operator $D = \nabla^T \mathcal{A}$.
    * The matrix of its Laplacian operator $L = \nabla^T \mathcal{A} \nabla$.
      - This matrix prefactored to allow for fast linear solving.
  - $\mathbf{\Psi}$ is the groundtruth mapping, which is basically an assignment of each vertex in $\mathcal{S}$ to a new position.
  - $\mathbf{y}$ is a conditioning information.

- Given a mesh $\mathcal{S}$, the prediction is conducted as follows.
  - We first use the encoder to produce the latent code: $\mathbf{z} = E(\mathbf{S}, \mathbf{y})$.
  - For each triangle $\mathbf{t}_i$ in the mesh, we compute a 3 Jacobian matrix $J_i = M(\mathbf{c}_i, \mathbf{z})$.
    * Let us denote the stack of J_is by $\mathbf{J}$.
  - We then compute a linear piecewise mapping by solving Poisson's equation:

$$L\mathbf{\Phi} = D\mathbf{J}.$$

  - We return $\mathbf{\Phi}$ as the output.

- Training uses the following loss.

$$\mathcal{L} = \lambda_{\text{vertex}}\mathcal{L}_{\text{vertex}} + \mathcal{L}_{\text{jacobian}}$$

  - $\mathcal{L}_{\text{vertex}}$ is L2 loss between the predicted vertex positions and the ground-truth ones.
    * The paper uses the formula

$$\mathcal{L}_{\text{vertex}} = \sum_{j=1}^{|\mathbf{V}|} m(\mathbf{v}_j)\|\mathbf{\Phi}_j - \mathbf{\Psi}_j\|^2.$$

      where $m(\mathbf{v}_j)$ is the "lumped mass" around Vertex $\mathbf{v}_j$.
    * However, in the code, there is no lumped mass term. The formula is just:

$$\mathcal{L}_{\text{vertex}} = \sum_{j=1}^{|\mathbf{V}|} \|\mathbf{\Phi}_j - \mathbf{\Psi}_j\|^2$$

    * Since the predicted mapping in unique up to translation, it is translated so that the center of mass are the same.
  - $\mathcal{L}_{\text{vertex}}$ is the L2 loss between the Jacobian matrices, restricted to the tangent space of each traingle.
    * The paper uses the formula

$$\mathcal{L}_{\text{vertex}} = \sum_{i=1}^{|\mathbf{T}|} A(\mathbf{t}_i)\|\mathcal{B}_i J_i - \mathcal{B}_i(\nabla\Psi)_i\|^2.$$

    * However, again, the code just drops the scaling factor.

$$\mathcal{L}_{\text{vertex}} = \sum_{i=1}^{|\mathbf{T}|} \|\mathcal{B}_i J_i - \mathcal{B}_i(\nabla\Psi)_i\|^2.$$

  - The weight λ is set to 10 in the paper.

- The mapping network M.
  - This is a 5-layer fully connected MLP with ReLU activation and group norm after each leayer.
  - Hidden layers are of size 128.
  - The first layer depends on the size of $\mathbf{z}$.
  - The last layer outputs 9 numbers.
  - We add the identity matrix to output of the last layer so that the MLP still producees a valid matrix when it outputs the zero matrix (which it is likely to do right after initialization).

- The global code $\mathbf{z}$.
  - It may contain raw conditioning information such as the pose of the output mesh.
  - To encode shape of a mesh, the paper uses a PointNet [QSMG17].
    * It receives 1024 points sampled uniformly on the mesh, along with their normals and WKS of size 50.
    * The PointNet is modified to use group normalization.

    * It is trained together with the mapping network.

- Training.
  - The optimizer is Adam.
  - Learning rate is first 10^{-3} until loss plateaus. Then, it is reduced to 10^{-4} and trained until the loss plateuas again.
- Factoring the Laplacian L.
  - The paper says it uses LDL decomposition implemented by SciPy's SuperLU decomposition.
  - However, the code uses `CholeskySolverD` from the `cholespy` library.

References

[AGK+22] N. Aigerman, K. Gupta, V. G. Kim, S. Chaudhuri, J. Saito, and T. Groueix, *Neural jacobian fields: Learning intrinsic mappings of arbitrary meshes*, 2022, `arXiv:2205.02904 [cs.GR]`.

[ASC11] M. Aubry, U. Schlickewei, and D. Cremers, *The wave kernel signature: A quantum mechanical approach to shape analysis*, 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2011, pp. 1626–1633.

[Cra25] K. Crane, *Discrete Differential Geometery: An Applied Introduction*, `https://www.cs.cmu.edu/~kmcrane/Projects/DDG/paper.pdf`, 2025, Accessed: 2025-04-16.

[JP25] A. Jacobson and D. Panozzo, *libigl: A simple C++ geometry processing library*, `https://libigl.github.io/`, 2025, Accessed: 2025-04-16.

[QSMG17] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *Pointnet: Deep learning on point sets for 3d classification and segmentation*, 2017, `arXiv:1612.00593 [cs.CV]`.