Exploration of Deep Mesh Denoising

Amir Barda [* 1] **Mattan Serry** [* 1] **Yotam Erel** [* 1] **Amit H. Bermano** [1]

Abstract

Polygon meshes are a common representation of $3D$ shapes in the domain of computer graphics. Usually, geometry reconstruction technologies such as depth–sensing cameras, acquire other representations, mostly voxels and point clouds. However, in $3D$ objects modeling, these representations are often converted to triangle meshes. This conversion may be lossy, noisy or occluded. In this work, we propose a novel approach for mesh denoising by encoding the local geometry of the mesh edges into a similarity–invariant graph, and applying techniques from recent advancements in Graph Convolutional Networks to learn the denoising task.

1. Introduction

1.1. Triangle Meshes

Polygon meshes are an important representation of 3D objects in computer graphics and geometric modeling. A popular choice for polygon meshes are triangle meshes, in which the planes are triangles in a 3D space. Triangle meshes have the nice trait that each edge has a 1–ring neighborhood of exactly four edges. A triangle mesh is a graph $G = (V, E)$ where the vertices V are coordinates in R^3 and the edges E are pairs of vertices that define a topology. Sometimes a mesh representation includes another set, F, the triplets of vertices that define triangular faces, yet these can be computed directly by searching for cycles of length 3 in E.

1.2. Mesh Representation

There are many options to represent a mesh for machine learning tasks, such as classification and segmentation. The immediate option is to use the graph representation itself, V and E, in matrix forms (or A, an adjacency matrix). This representation is sometimes used in Graph Convolutional Networks. Some disadvantages of this choice is its large memory consumption, the ordered nature of the matrices, and the lack of invariance to similarity transformations of the mesh: translation, rotation, and nonuniform scaling.

1.2.1. Similarity–Invariance

A similarity–invariant compact representation of meshes was proposed in MeshCNN (Hanocka et al., 2019). In this representation, every edge borders two faces, and each face contributes two neighbouring edges. The edge is defined by five scalars: the dihedral angle between the two faces, two inner angles and two edge–length ratios for each face.

1.2.2. Rotation–Invariance

(Yaron Lipman, 2005) introduces a rotation–invariant mesh representation, based on the Laplacian operator and differential geometry, which they named Linear Rotation–Invariant (LRI). The LRI representation calculates a local orthogonal coordinate frame (b_1, b_2, N) for each edge, where the normal is the weighted average normal of incident faces.

$$N_i = \frac{\sum_j A_j \cdot N_j}{\sum_j A_j} \tag{1}$$

where A_j is the area of a face incident to vertex i and N_j is the normal of that face.
The b_1 direction is the projection of an arbitrary incident edge to the plane defined by the vertex v_i and the normal (calculate for v_i using eq. 1), the b_2 direction is defined to create an orthogonal right hand side coordinate frame. After all frames are calculated, the per–edge LRI descriptors can be defined: for each edge, the LRI descriptors are the differences between the two coincident local frames, as in eq. 2 . Note that this encodes the local change in geometry, where smaller differences denote flatter areas in the mesh, and vice versa. In this form each edge has $2 \cdot 3 \cdot 3 = 18$ descriptors: both $\delta_i j$ and $delta_j i$ contain the difference of three 3d vectors.

$$\begin{aligned} \delta_{ij_{b_1}} &= b_{1_i} - b_{1_j} \\ \delta_{ij_{b_2}} &= b_{2_i} - b_{2_j} \\ \delta_{ij_N} &= N_i - N_j \end{aligned} \tag{2}$$

This is redundant, as we can fully restore the local frames using just two of them, because the third is a unit vector which completes to an orthogonal right–hand side system.

*Equal contribution [1]Tel Aviv University, Blavatnik School of Computer Science. Correspondence to: $<>$.

Advanced Machine Learning, Fall '19, Tel Aviv University, Blavatnik School of Computer Science. Copyright 2020 by the authors.

Therefore, to reduce the number of parameters representing a mesh, we implement a reduced form of the LRI descriptors which contain only the b_1 and N frames. This reduces the number of descriptors for each edge in the mesh to $2 \cdot 2 \cdot 3 = 12$.

We can now define the equivalent of the Laplacian (topology–encoding matrix) over the local frames. Given the vector of LRI descriptors, we can solve the system using a least square solution to obtain the full mesh geometry.

2. Method

2.1. Human Segmentation

Out experiments are evaluated on the human body segmentation dataset proposed by (Maron et al., 2017). The dataset consists of 370 training models from SCAPE (Anguelov et al., 2005), FAUST (Bogo et al., 2014), MIT (Vlasic et al., 2008) and Adobe Fuse (ado, 2020), and the test set is 18 models from SHREC07 (Giorgi et al., 2007) humans dataset. The 3D meshes include humans in various poses, with the same topological properties: 752 vertices, 2252 edges and 1500 faces. Note The meshes are all manifolds with genus zero, and no self intersecting faces. This dataset also consists of segmentation data, but we are only concerned with reconstructing denoised meshes, so the segmentation labels are discarded. This dataset was selected because we heavily rely on the consistent topology and properties of the meshes mentioned. We permit only a fixed size number of edges as input, and solve the ambiguous neighbour and ordering problem when the meshes are manifolds as mentioned in (Hanocka et al., 2019).

2.2. Mesh Preparation

2.2.1. Global Translation and Rotation

Reconstructing a mesh using the LRI feature representation requires selecting some arbitrary vertex as the origin, and an arbitrary global frame of reference. All other vertices are positioned relative to this origin, and relative to this frame of reference. This implies that upon reconstructing the mesh, it will be arbitrarily be translated and rotated. It follows that determining how good a reconstruction is using a loss function such as MSE between vertex positions of the clean and noisy mesh becomes a problem. To counter this, the first stage of processing a mesh is to extract its LRI representation, and then reconstruct it. This ensures the clean mesh and the noisy meshes are positioned and oriented in the same manner.

2.2.2. Global Scaling

Let v be the $\|V\| \times 3$ array describing vertices V of a mesh. We define the following operator on v:

$$\delta(v) = \sqrt[6]{(det(cov(v^T))} \tag{3}$$

Operator (3) encodes a property of v, such that when dividing v by it, the determinant of its co–variance matrix is exactly unity:

$$\forall v, det(cov(\frac{v^T}{\delta(v)})) = 1 \tag{4}$$

We utilize Formula (4) to normalize all meshes' vertices by dividing them with $\delta(v)$. This enables all of our meshes to have some shared property of normality. In particular, it forces global scale invariance. It also has the effect of decreasing sensitivity to outliers. For example, let mesh M fit exactly in a unit sphere and let mesh M' be a copy of M, but one of its vertices was wrongly measured and is extremely far from the mesh. Suppose our method of normalization was scaling to fit to a unit sphere, then M is unaffected, but M' (which is almost equal to M) would be extremely down–scaled. In our method, however, assuming large $\|V\|$, $\delta(v)$ is almost unaffected, and the meshes will remain similar after the normalization. In our experiments, we used couples of "clean" and noised meshes for training. This normalization enabled us to noise all meshes with the same noise distribution, without worrying that smaller meshes were more affected by the noise.

2.2.3. Noise Application

We applied additive white Gaussian noise (AWGN) with variance 0.1 for noised meshes and variance 0.01 for target (relatively clean) meshes. The selection for the noisy meshes was made by applying the highest variance that was observed to preserve the integrity of the mesh (almost no self intersections), and for the target mesh taking a variance of one order of magnitude smaller. The noise was added to the vertices positions prior to extracting LRI features.

2.3. Network Architecture

The network architecture we used was a fully convolutional encoder–decoder style architecture made with the following convolutional layers filter sizes: $32 \times 32 \times 64 \times 64 \times 128 \times 128 \times 256 \times 256$ (encoder), $256 \times 256 \times 128 \times 128 \times 64 \times 64 \times 32 \times 32$ (decoder). The individual layers were re–purposed from the original implementation presented in (Hanocka et al., 2019), by removing the residual blocks and the pooling layers, and changing the activation layers from ReLU to tanh. Instance normalization layers were used between every stack of identical convolutional layers.

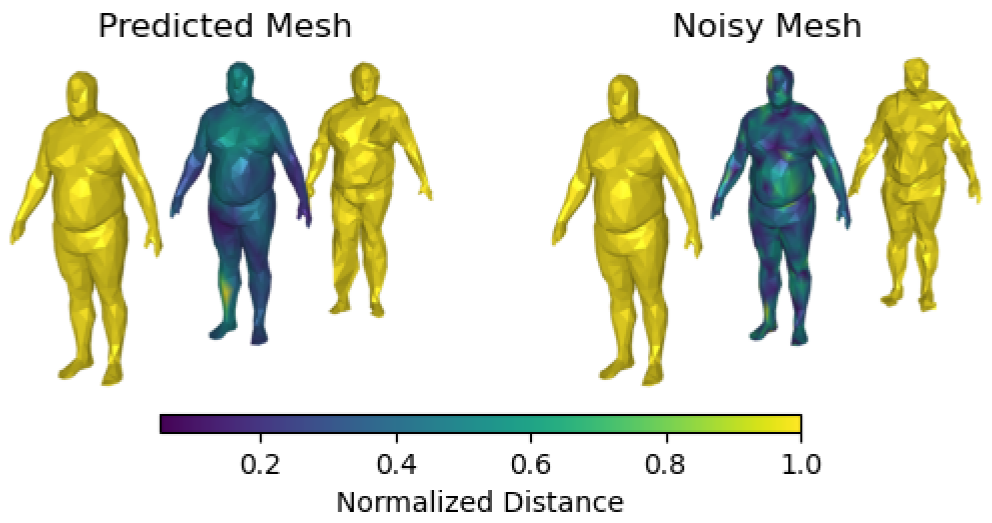

Figure 1. A result from the test set when learning to globally denoise a mesh using LRI features. Left: three meshes are presented (from left to right): 1. The original mesh, 2. the original mesh with color indicating distance of the denoised vertices from it, 3. the denoised mesh as predicted by the network. Right: This time the prediction is replaced with a randomly noised mesh with same distribution the network was trained on. The network is observed to perform some sort of smoothing of the noise. Similar results were observed for other test meshes.

2.4. Evaluation Metric

We chose the E_α evaluation metric to score our results, which is the average angular error over the entire test set. It is defined as the dot product between the normals of the faces in the target mesh and predicted mesh. The E_v metric is also used to make some conclusions. It is defined as the mean squared error between vertices positions over the entire test set.

3. Experiments

3.1. LRI Features Learning

In these experiments, the LRI feature vector representing a mesh is used both as the data and the label to the network in a fully supervised fashion. Since we significantly altered the backbone of our network (Hanocka et al., 2019), the main goal of learning to denoise a mesh is split into smaller, simpler tasks that assure the network is creating a faithful internal representation of our meshes. Each smaller task reveals insights as to how to approach the next:

- Learning the identity function
- Learning mesh augmentations
- Learning to globally denoise a mesh

3.1.1. Identity Function Learning

As another sanity test, we inserted many meshes (processed into feature vectors) from the training set with the label being the feature vector itself: $\{(x, y)|x = y\}$ where $x, y \in \mathbb{R}^{|E|\times 12}$ since there are twelve features per edge of the mesh. We wanted to observe how the network behaves when it is forced to learn the identity function. This is an important step for the denoising task, since a good starting point to denoise a mesh would be the original noisy mesh. In other words, initializing the network weights in later steps such that identity is returned is very useful.

3.1.2. Mesh Augmentations Learning

In this experiment, we wish to verify to some degree that the network is able to cope with our augmentation of choice described in 2.2.3. This will be advantageous because meshes derived from real data often include noise and errors for which our network will be robust against. We create many noisy meshes from a single clean original mesh, where ”clean” and ”noisy” meshes are created as described in 2.2.3. Since all target noisy meshes are derived from a single underlying clean mesh, it is easy to see that a well performing solution in terms of loss minimization would be to output the original clean mesh. Thus, we tested to what extent can the network learn to output the original clean mesh, and if learning converges. The results imply the network has no problem learning the underlying mesh geometry to a satisfactory degree, as can be seen in Figure(2). This is no surprise since learning this is equivalent to learning the distribution of the noise applied to the data, which was shown to be feasible in many prior works and is a common method used to reduce over–fitting.

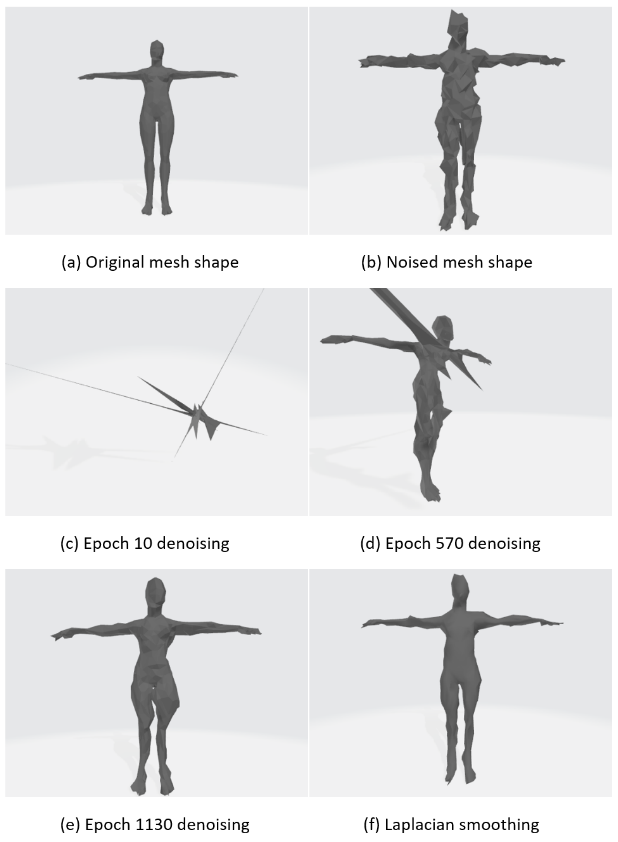

Figure 2. Learning to cope with augmentations. (a) Original mesh from SHREC. A slightly noised version of it is the target for the training. (b) Noised mesh, as input to the model. (c) Denoising result in an early step of the training, no shape features learned yet. (d) Denoising result in an intermediate step of the training, some shape features are learned. (e) Denoising result in an advanced step of the training, shape features are learned and mesh denoising is visible. (f) Mesh denoising with Laplacian smoothing filter (Herrmann, 1976) on *(b)*. Fine features like fingers are blurred.

3.1.3. Global Denoising Learning

Using the insights gathered from the other tests, we designed a training scheme for the entire dataset. Meshes were fed in batches into the network, each batch consisted of 8 pairs prepared as discussed in section (2.2), and training was stopped when validation loss stopped decreasing. We obtained $E_\alpha = 0.00047$ on the noisy test set using our network, while Laplace Smoothing ((Herrmann, 1976)) on the same noisy set yielded $E_\alpha = 0.00050$. These results strengthen the idea that LRI features are minimizing normal distances better than the classical approach. Visually, the network seemed to ”smooth” the noise as can be seen in Fig(1). The results appear heavily distorted in areas of sharp local changes in the mesh, even when the LRI feature vectors are quite close to the target mesh LRI features. This can be explained by the fact the LRI features in these areas are extremely sensitive to small changes. Also, when measuring the MSE on vertex position the networks output doesn't seem to be better than the random noisy test set (since LRI features are less descriptive of this). We therefore concluded that learning to minimize the MSE on the LRI feature vectors as a representative of the mesh is ultimately not an ideal choice when solving a global noise task. This method is also quite constrained by the fact it only allows calculating a loss that forces a one–to–one mapping between edges in the clean mesh and edges in the noisy mesh (thereby not allowing changes in topology at all).

3.2. 3D Vertices Learning

In this experiment we used the vertices positions themselves as the labels for training instead of the LRI feature vector representation. The inputs to the network are still LRI feature vectors, but we reconstruct the mesh as part of the training process. In other words, The auto–grad tree used to perform back–propagation included our reconstruction algorithm (which is differentiable by design). Note this method offers great benefits over learning directly on LRI features: it enables us to use other losses which are much more suitable to describe meshes (see supplementary for examples). Unfortunately, learning did not converge. The reason for this might be due to the sheer size of the back–prop tree that now includes the reconstruction algorithm as the first step. Our implementation of the LRI reconstruction uses substantially more operators than normal neural networks, and partial derivatives might diminish or suffer from numerical instability when they finally reach the network weights in the chain rule.

4. Discussion

The LRI feature vector representation of meshes opens up new opportunities to explore learning tasks on meshes because of its unique local properties. Reconstructing the mesh using the LRI features boils down to solving a least squares problem which has a closed form solution. This means that the reconstruction is differentiable, and by using it, we have demonstrated a neural network can learn to denoise an unseen mesh by changing its geometry to fit with a prior distribution.

In future work, we plan on extending these results for more elaborated tasks such as mesh super resolution and mesh generation. These sort of tasks require handling differentiable topology changes by the network and are thus highly compatible with our method.

References

Adobe fuse 3d characters, adobe. 2016. `https://www.mixamo.com`, 2020.

Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., and Davis, J. SCAPE: shape completion and animation of people. *ACM Trans. Graph.*, 24(3):408–416, 2005. doi: 10.1145/1073204.1073207. URL `https://doi.org/10.1145/1073204.1073207`.

Bogo, F., Romero, J., Loper, M., and Black, M. J. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 3794 –3801, Columbus, Ohio, USA, June 2014.

Giorgi, D., Biasotti, S., and Paraboschi, L. Shape retrieval contest 2007: Watertight models track. *SHREC competition*, 8(7), 2007.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. Meshcnn: A network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):90:1–90:12, 2019.

Herrmann, L. R. Laplacian-isoparametric grid generation scheme. *Journal of the Engineering Mechanics Division*, 102(5):749–907, 1976.

Maron, H., Galun, M., Aigerman, N., Trope, M., Dym, N., Yumer, E., G Kim, V., and Lipman, Y. Convolutional neural networks on surfaces via seamless toric covers. *ACM Transactions on Graphics*, 36(4):71, 2017.

Vlasic, D., Baran, I., Matusik, W., and Popović, J. Articulated mesh animation from multi-view silhouettes. *ACM Trans. Graph.*, 27(3):1–9, August 2008. ISSN 0730-0301. doi: 10.1145/1360612.1360696. URL `https://doi.org/10.1145/1360612.1360696`.

Yaron Lipman, Olga Sorkine, D. L. D. C.-O. Linear rotation-invariant coordinates for meshes. In *Proceedings of ACM SIGGRAPH 2005*, pp. 479–487. ACM Press, 2005.