REIN: Flexible Mesh Generation from Point Clouds

Rangel Daroya Rowel Atienza Rhandley Cajote
University of the Philippines Diliman
Electrical and Electronics Engineering Institute
{rangel.daroya, rowel, rhandley.cajote}@eee.upd.edu.ph

Abstract

3D reconstruction from sparse point clouds is a challenging problem. Existing methods interpolate from point clouds to produce meshes, but the performance decreases with the number of points. To address this, we propose an algorithm that looks at the global structure while reconstructing the surface one vertex at a time. Experimental results on ShapeNet and ModelNet10 show 81.5% Chamfer Distance and 14% Point Normal Similarity average improvement compared to Ball Pivoting Algorithm (BPA) and Poisson Surface Reconstruction (PSR). Qualitatively, the generated meshes have a closer similarity to the ground truth. Results on ShapeNet Patched illustrate significant improvement in mesh quality compared to BPA and PSR. The code is available at https://github.com/rangeldaroya/rein.

1. Introduction

Representations of 3D objects have seen a lot of applications in several computer vision tasks for allowing high-fidelity modelling in construction planning, risk assessment of infrastructures, and restoration projects [2, 4]. Works have developed systems that can represent objects in 3D [8, 40, 34, 26, 14] either in the form of point clouds [14, 1], occupancy grids [8, 33], or meshes [34, 26, 15, 30, 36]. Point clouds are easily extracted from depth images or sensors by projecting each pixel in 3D space according to the corresponding depth values. However, point clouds can be sparse and lack information to accurately model objects. Sparsity of point clouds can be observed in some low cost Light Detection And Ranging (LiDAR) sensors [35, 39, 12], and in some Structure from Motion (SfM) [31, 33] outputs. When few images are available to reconstruct a 3D object, sparse point clouds result from SfM.

Despite existing works that try to overcome challenges in point cloud and occupancy grid representations [8, 40, 33], meshes can prove to be more efficient [34, 26]. Meshes are represented in a continuous space by the vertex locations and the interconnections or edges to create the 3D object.

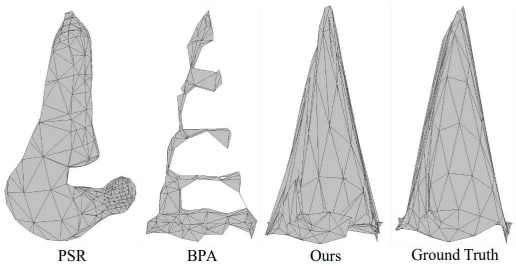

Figure 1. Comparison of surface reconstruction outputs from different algorithms.

Additional information on the object structure is obtained from the surfaces formed by the edges between the vertices or points. When given only the eight vertices of a cube, it is hard to see the underlying object. The cube is better depicted once the edges and the faces are in place to form a solid object. Even with sparse points, an object can be fully represented by forming the necessary faces. For this reason we focus on reconstructing surfaces from sparse point clouds to form meshes.

Although the current methods Ball Pivoting Algorithm (BPA)[3] and Poisson Surface Reconstruction (PSR)[17, 16] demonstrate ability to create meshes from point clouds, they struggle reconstructing sparse inputs. BPA is limited by the proximity of points as basis for forming surfaces. When the given points are sparse, BPA is unable to connect distant points. PSR determines the surface by casting the problem as a spatial Poisson problem. The formed surfaces are closed and continuous, but the algorithm fails to capture sharp edges and high curvatures.

The irregularity of meshes can make them challenging to form and process. Unlike images or occupancy grids that have regular grid-like structures, meshes have vertices embedded in continuous space and have varying face sizes. Most works simplify the mesh generation process by deforming an existing mesh with predefined interconnections such as a sphere or ellipsoid [15, 34, 30]. However, mesh flexibility is limited and the representation efficiency is reduced when the surface regularity is enforced. While

Scan2Mesh [10] developed a way to form meshes without restricting the interconnections, their work is limited to a predefined number of points and relies on fully connected graphs to predict edges. Polygen [23] presented a different method of generating meshes, but their mesh generation is conditioned on other object representations such as images and voxels.

Rather than fixing interconnections or relying on static fully connected graphs, we use a dynamic processing scheme. We propose to predict edges in a sequential manner. Inspired by Graph RNN [29], we present REIN: Recurrent Edge Inference Network. REIN builds a mesh from a point cloud using a bottom-up approach. The point cloud is organized into a queue of points. Meshes are produced by introducing one point at a time from the queue. Features extracted by GraphRNN aids REIN in modelling the distribution of mesh interconnections. The sequential nature takes advantage of the accumulated learned information at every time step, and provides feedback for future predictions. Inclusion of the point cloud latent vector gives information about the global structure throughout the process.

To summarize, our main contribution is a flexible mesh generation method that sequentially predicts edges with an encoded point cloud representation. We demonstrate that our proposed network, REIN, achieves significant improvement compared to other surface reconstruction algorithms: Ball Pivoting Algorithm (BPA) and Poisson Surface Reconstruction (PSR). Experimental results show that REIN achieved 77% average Chamfer Distance reduction compared to BPA, and 86% average Chamfer Distance reduction compared to PSR on the ShapeNet and ModelNet10 datasets. Point normal similarity also improved by 14% compared to BPA and PSR. Figure 1 illustrates some qualitative results.

2. Related Work

2.1. Surface Reconstruction

Point cloud processing has been done to apply upsampling, segmentation, and classification [21, 27, 28, 1]. These gained traction due to the abundance and availability of point cloud data. The unstructured form of point clouds make processing nontrivial. PointNet [27] addressed the permutation-invariance problem by using a symmetric function that transforms a set of input points to a vector that is invariant to order. They learn to extract global and local properties of each point and its neighbors, enabling them to also predict point normals. This ability to extract point cloud features is used by our work to encode the information from an input point cloud.

Most surface reconstruction algorithms initially estimate the direction of the normal vector per point by taking into account the k nearest neighbors. The estimated normal vectors become the basis for interpolating the points. Two commonly used methods for surface reconstruction are Poisson Surface Reconstruction (PSR) [18, 17] and Ball Pivoting Algorithm (BPA) [3]. BPA and PSR are used to faithfully reconstruct 3D objects from dense point clouds. BPA forms meshes by using a size-varying ball to cluster vertices into triangular faces. PSR extracts an isosurface based on oriented point samples and the Poisson equation. Both PSR and BPA have trouble reconstructing surfaces from sparse point clouds.

Figure 1 illustrates some problems encountered when few points are given to BPA and PSR. PSR forms surfaces in areas where there should be gaps in the mesh, such as the space between the legs of a table. This removes details in the object from sharp contours. The lack of neighbor information to accurately estimate the normal vectors of each point causes erroneous interpolation of the planes and surfaces. BPA produces meshes with gaps or holes when given limited points to reconstruct. The ball-based clustering of points can become ineffective when the vertices are too far apart.

2.2. Learning-Based Mesh Generation

The irregular structure of meshes make them challenging to process. Common CNN architectures are difficult to apply to meshes. Other tools such as Graph Neural Networks (GNNs) [29] and Graph Convolutional Networks (GCNs) [20] are used. Pixel2Mesh [34] uses a GCN-based network to generate a 3D mesh given an RGB image. Their network extracts features from the image to deform an initial ellipsoid mesh. Computations for the mesh deformation are based on the vertices and the adjacency matrices. The number of vertices are preset, focusing on deforming existing primitives or surfaces similar to AtlastNet [15] and other works [30, 36]. Polygen [23] tries to improve on AtlasNet by generating faces instead of using existing surfaces, but their work relies on other object representations such as images or occupancy grids. In contrast, our proposed model, REIN, generates edges from a set of vertices without pre-existing assumptions on the mesh. Edge inference also enables the number of mesh vertices to be flexible.

A similar work is Scan2Mesh [10] which focuses on predicting full mesh structures from incomplete range scans. Aiming to predict both the vertex locations and the interconnections, they use fully connected graphs. GNN is used to simultaneously classify relevant edges. The initial prediction of edges is refined with a dual graph to predict the faces. Although their work can predict full meshes, their process can quickly consume memory. The memory usage increases with increasing number of vertices up to 400 points. Their network also produces meshes with a predetermined number of vertices. Our work aims to incrementally predict edges without having to rely on fully connected

graphs. Incremental processing allows our output to have a more flexible and dynamic structure. Meshes during our training and prediction can have varying number of vertices.

2.3. Generative RNN

GraphRNN [38] utilizes RNN to generate graph structures by reformulating the problem as sequential predictions [6, 24]. Given a set of nodes, edges are generated one at a time, resulting in graphs with the same distribution as the input. As a node is introduced, the network predicts connections with existing nodes in the graph. The network's recurrent nature allows for flexible interconnection generation. Similar works [24, 32] take advantage of the sequential nature of data to determine the current state, aiding the network for future predictions.

Our work is most similar to Graph RNN, utilizing a recurrent network to generate edges. Instead of generating graphs, however, we generate meshes. In addition to inferring edges from points, our network also defines faces. Given a point cloud, we predict interconnections between points by introducing vertices one at a time. The latent vector of the input point cloud is also incorporated in our network using an autoencoder. The latent vector aids the RNN by introducing more information about the whole structure at every time step.

3. Methodology

We define a mesh $\mathcal{M}$ as a set of vertices, edges, and faces: $\mathcal{M} = \{\mathcal{V}, \mathcal{E}, \mathcal{F}\}$. We propose to create meshes from a set of points by utilizing their sequential dependence. The latent vector representation of a point cloud is obtained from a PointNet-based [27] autoencoder. Most of the information about the input points should be accessible from the latent vector, from which we can infer the general structure of the target object.

Since we are aiming for a bottom-up approach, a small section of the input point cloud is examined at a time, instead of all at once. A per section examination is done by introducing one point at a time from the input. Edges are predicted as connections between a newly introduced vertex and the vertices that have already been introduced in previous time steps. Predictions are based on the current status of the partially predicted mesh and the latent vector obtained from the point cloud. Sequential dependence is used to predict the interconnections and provides continuous feedback for future predictions.

Sequential Prediction of Edges. We structure the problem as follows: given a point cloud with vertices $\mathcal{V} = \{v_1, v_2, ..., v_n\}$, we need to predict a set of edges $\mathcal{E} = \{\vec{e}_1, \vec{e}_2, ...\vec{e}_n\}$ which defines n edge subsets (denoted by $\vec{e}_i$) that gives the interconnection per vertex. That is, an edge prediction for vertex v_5 is defined as $\vec{e}_5 = \{e_{51}, e_{53}, e_{54}\}$ which implies that vertex v_5 is connected to vertices v_1, v_3, and v_4. The goal is to predict n edge subsets sequentially to define the interconnections for all vertices. Once all edge subsets are defined, faces can be obtained and the mesh is formed. Equation 1 formalizes the sequential edge prediction. Equation 1 states that future predictions on a vertex's interconnections are dependent on predictions of previously introduced vertices:

$$P(\mathcal{E}) = \prod_{i=2}^{n} P(\vec{e}_i | v_1, ..., v_i, \vec{e}_1, ..., \vec{e}_{i-1}) \tag{1}$$

All predicted edges are towards the lower indexed vertices or points. Our formulation assumes that edges in a mesh are undirected. An existing edge e_{12} connecting vertex v_1 to v_2 implies the existence of edge e_{21} connecting vertex v_2 to v_1. Defining edges as undirected simplifies the setup to predict a single edge connecting two vertices, instead of predicting two separate connections for the same two vertices.

Aside from being conditioned on previous edge predictions, we include the point cloud latent vector to give context of the structure. The latent vector allows the relationship between points to be incorporated. Equation 2 shows that the next state of the mesh is determined from the previous state, the previously predicted edges, and the latent vector of the point cloud (z):

$$h_i = f_{trans}(h_{i-1}, \vec{e}_{i-1}, z) \tag{2}$$

The equation for next state supports the previously discussed sequential dependence of the predictions. The state prediction determines the distribution from which the next edges will be based. A mapping function is used to determine the parameter for the distribution. Equation 3 formalizes this:

$$\theta_i = f_{out}(h_i) \tag{3}$$

The distribution of the next edge predictions is determined by the parameter θ_i. Equation 4 shows that the next edge prediction is sampled from the distribution P_{θ_i}:

$$\vec{e}_i \sim P_{\theta_i} \tag{4}$$

The process described above can be repeated for all points to determine the edges of each vertex. The network formulation enables flexible number of input points. For each iteration the edge vector length increases by one. To implement this model, an apt tool to use is a Recurrent Neural Network (RNN). An RNN is capable of executing sequential predictions conditioned on previous states. The RNN's ability of providing future predictions based on past states make it a good candidate and was used in our work to generate edges.

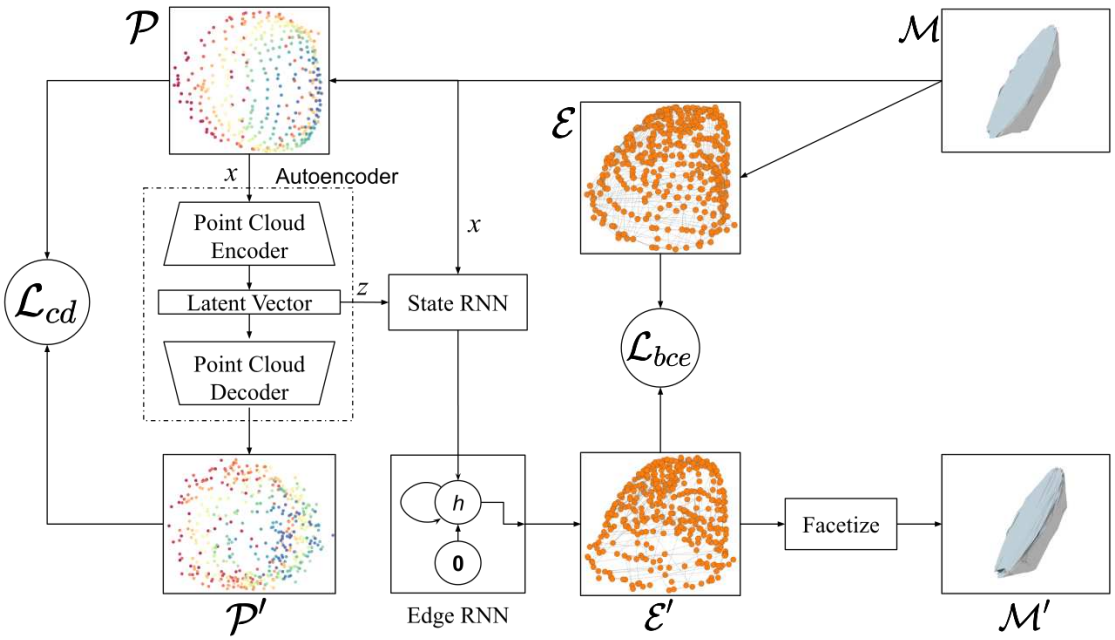

Figure 2. Network architecture used to predict the interconnections and meshes from a set of mesh vertices. Layers used in the architecture are detailed in the Appendix.

3.1. Network Architecture

Our work uses concepts borrowed from graph generation to build a set of vertices and edges. A graph's nodes and edges are analogous to a mesh's vertices and edges. Aside from vertices and edges, meshes are defined by a set of faces. There are thus two parts to this work: (1) edge prediction from the given vertices and (2) face generation from the given edges. Figure 2 shows the network overview.

The latent vector representation of the point cloud is used to incorporate information about the global structure of the object. To make sure the latent vector contains enough information about the input point cloud, Chamfer Distance (CD) is used on the input and output of the autoencoder. Equation 5 defines CD:

$$\mathcal{L}_{cd} = \sum_{a} \min_{b} ||a-b||_2^2 + \sum_{b} \min_{a} ||a-b||_2^2 \qquad (5)$$

Points a of point cloud P and points b of point cloud P' are compared in the equation. Edges are predicted using a two stage RNN. To predict edges, vertices are introduced one at a time. Binary cross entropy loss ($\mathcal{L}_{bce}$) is used to compare the predicted edges and the ground truth. The faces are deduced from the edge prediction by defining a face on any three vertices connected by edges.

3.1.1 Edge Prediction

Two sets of RNNs are used to predict the interconnections. We denote these two sets of RNNs as the *State RNN* and *Edge RNN*. The *State RNN* is used to encode the state of the current graph given the nodes and interconnections (see Equation 2). The *Edge RNN* is used to predict the sequence of connections given the current state encoded by the *State RNN* (see Equations 3 and 4). The *State RNN* and *Edge RNN* work together to predict the final structure of the mesh. Figure 2 illustrates the pipeline.

Breadth-First search (BFS) is used to determine the sequence of node introduction to reduce the complexity of the network. The possible interconnections a node can form is reduced with BFS. Without BFS, at each iteration during training, REIN would have to predict the connections from a vertex to all other $n-1$ vertices in the input point cloud with n points. Since a queue for the node introduction order is preset, REIN would only need to predict the interconnection between the current vertex and the previously introduced vertices. The number of possible edge predictions per time step is significantly reduced. The starting node for BFS is determined randomly, and the sequence of node introduction is determined from there.

The *State RNN* continuously encodes the graph given the sequence of edge predictions and the latent representation of the point cloud input. The latent vector gives the network information about the global structure. The local structure can be inferred from the node coordinates. As nodes are added to the graph, the *State RNN* updates the encoded information and initializes the *Edge RNN*. The output of the *State RNN* serves as the hidden state of the *Edge RNN*.

The final prediction is in the form of an adjacency ma-

trix. In the matrix, a value of 1 indicates the presence of a connection, and 0 otherwise. The network works under the assumption that the existence of edge e_{12} implies the existence of edge e_{21}. Since edges are undirected, the network only needs to predict edges e_{ij} where $i > j$.

Gated Recurrent Units (GRUs) [7] were used to both encode the mesh structure and to predict the sequence of edges in *State RNN* and *Edge RNN*. Supervised training was done with binary cross entropy ($\mathcal{L}_{bce}$) as the loss function between the predicted adjacency matrix and the ground truth. Figure 2 illustrates this.

A possible limitation on the proposed network architecture is the required memory for processing large point clouds. Having sequential dependence requires the network to keep track of all previously introduced points. As the number of input points increase, the required memory will also increase. Due to resource limitations, an upper bound on the number of vertices that can be processed at a time is set. However, the limitation in processing is overcome by reconstructing by parts. Instead of processing a large number of points at a time, subsets of the input point cloud are introduced one at a time. Section 5.3 discusses reconstruction by parts in more detail.

3.1.2 Face Generation

Facetizing is done by going through all the existing vertices and edges predicted by the network. It is assumed that all faces are triangular, consistent with the structure of the ground truth meshes used for training. Aiming for a triangular mesh, a face is formed whenever three vertices are connected by three predicted edges. The orientation of the surface normals of the faces are dependent on the order of the vertices. The order is determined by sorting the indices of the vertices.

4. Data Generation

All training data were obtained from ShapeNet [5] and ModelNet10 [37]. Eight object classes were taken from ShapeNet based on previous mesh generation works [10, 11], following the same train/test split. All ten object classes from ModelNet10 were used with the included train/test split.

Meshes from the datasets were preprocessed with the V-HACD library [22] to obtain the convex hulls of the objects that will serve as the ground truth. This is similar to how ground truth meshes were generated from Scan2Mesh [10]. Convex hulls remove ambiguities for internal planes or structures, and result to watertight meshes. A watertight mesh is also required to use BFS for node ordering.

V-HACD library was configured such that the number of vertices is at most 500 vertices by specifying the resolution. Given the convex hulls of the objects, three dataset configurations were implemented to benchmark the performance of the network under different conditions: (1) Original Convex Hull Vertices (Hull), (2) Butterfly Subdivided Meshes (Butterfly) [13], and (3) Midpoint Subdivided Meshes (Midpoint) [25].

The Hull setting involved training and evaluating the network on the original convex hull vertices of the meshes. The Butterfly and Midpoint configurations were applied to add more points to the set of meshes belonging to Hull. Adding more points aims to approach a more uniform point distribution compared to Hull. The Meshlab [9] implementation of the normal estimation for PSR requires at least 300 vertices, making the Butterfly and Midpoint preprocessing configurations necessary for more test samples. Subdivisions were applied on the meshes until at least 300 vertices are reached. Meshlab parameters used for subdivision are included in the Appendix. Figure 3 illustrates the data processing of the convex hulls.

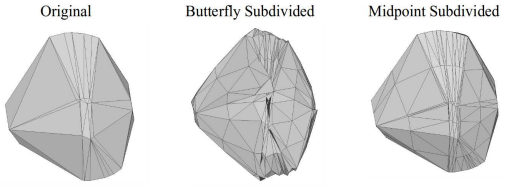

Figure 3. Illustration of preprocessing methods applied on the meshes to generate different datasets.

5. Experiments

The network was trained using GeForce GTX 1080 Ti. The training set was filtered to include meshes with at most 500 vertices. There were at least 16,000 training meshes from ShapeNet, and at least 5,000 training meshes from ModelNet10. Data augmentation was applied to ModelNet10 to compensate for the smaller mesh samples. Random jitter, rotation, and translation were applied to the meshes. Adam optimizer [19] was used with a learning rate of 10^{-5} for the autoencoder, and a learning rate of 0.003 for the *State RNN* and *Edge RNN*. A batch size of 1 was used for all the training setups. The facetize process is done separately and is not part of the training.

5.1. Evaluation of REIN

The test set was restricted to meshes with 300 to 500 vertices. The lower bound in the number of vertices was limited by the implementation of Meshlab's normal estimation for PSR. The parameters used can be found in the Appendix. Scan2Mesh [10] was not considered due to the current unavailability of their processed dataset.

The network was trained and tested exclusively for each of the three ShapeNet datasets (Hull, Butterfly, Midpoint).

The models used for each of the ModelNet10 datasets were pretrained on the ShapeNet dataset due to the lack of sample meshes in ModelNet10. The network hyperparameters are fixed for all dataset variants. The evaluation time for a mesh is approximately 160 ms – 100 ms to predict edges and 60 ms to generate faces. For comparison, BPA, on average, can generate a mesh in 190 ms, and PSR in 220 ms.

Chamfer Distance. The Chamfer Distances between point clouds were used as the basis in evaluating the autoencoder network's outputs. The autoencoder was constructed such that it adequately reconstructs the input. Results of the final mesh were also compared using CD which measures the average point to point distance between meshes. The CD is measured between meshes by uniformly sampling 2,048 points from the predicted mesh and 2,048 points from the ground truth mesh similar to [10]. This is done to compare the surfaces of the generated meshes.

Given two meshes $\mathcal{M}_A$ and $\mathcal{M}_B$, the CD is computed as the distance from $\mathcal{M}_A$ to $\mathcal{M}_B$ and $\mathcal{M}_B$ to $\mathcal{M}_A$. The distance from $\mathcal{M}_A$ to $\mathcal{M}_B$ is obtained using the distance of each point a in $\mathcal{M}_A$ to its corresponding closest point b in $\mathcal{M}_B$, and taking the sum. The average of the two way CD between the meshes serves as the loss of our autoencoder. CD is also the metric we use for comparing the ground truth and the predicted mesh. CD is formalized in Equation 5.

Point Normal Similarity. Vertex normals serve as the basis in predicting the faces. The normals are implicity obtained with the prediction of the interconnections and faces. To measure the accuracy of the surface normals in the predicted mesh, point normals are compared between algorithms. Cosine similarity is used between the predicted and ground truth point normals. Similar to CD, 2,048 point normals were sampled from the mesh surfaces for comparison.

Comparison of Algorithms. Table 1 shows the comparison between our network and commonly used surface reconstruction methods. Both CD and point normal similarity (PNS) were used as the basis in evaluating REIN. The results illustrate that REIN can predict mesh structures more accurately from the same input points. The ShapeNet results show REIN outperforming BPA and PSR in terms of CD and PNS. The difference is more significant in the Hull dataset. The number of points in meshes belonging to the Hull dataset are lower, resulting in poor BPA and PSR performance. The ModelNet10 results show REIN mostly outperforming BPA and PSR on ModelNet10 dataset. PSR performed slightly better for PNS in the Hull dataset of ModelNet10, but the difference in PNS is outweighed by the large difference in CD. REIN is also shown to have consistent performance across the different datasets (CD and PNS are within the same range).

BPA and PSR are expected to have better performance in surface reconstruction with increasing number of vertices. Table 1 shows general improvement for BPA and PSR on Butterfly and Midpoint compared to Hull. Test meshes from ModelNet10 were grouped based on the number of vertices. The average CD of the three algorithms were computed for each group. Figure 4 shows the average CD computed with the specified number of vertices. BPA and PSR significantly improve with increasing number of vertices. Similarly, the performance of REIN improves with increasing vertices.

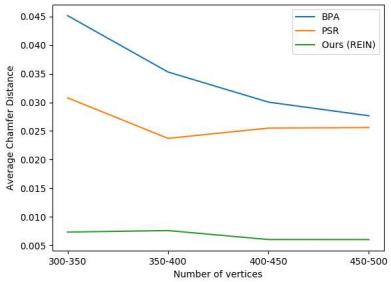

Figure 4. Average CD performance comparison of BPA, PSR, and REIN with increasing number of mesh vertices. The evaluation was done on ModelNet10.

5.2. Qualitative Results

Figure 5 shows sample mesh outputs from datasets ModelNet10 Midpoint and ShapeNet Butterfly. The figure shows that BPA is unable to form a complete surface. Most BPA predictions barely capture the object, and have large holes in the structures. In curved areas where more points are present, PSR can closely reconstruct the surface. PSR predictions fail to capture sharp object features and produce closed and curved objects. This illustrates the challenges BPA and PSR encounter when input point clouds are not dense. Both algorithms are unable to properly interpolate between points, resulting in lacking mesh predictions.

Predictions from REIN can capture sharp edges and surface details better than BPA and PSR for objects in ModelNet10 and ShapeNet. REIN's method of sequential edge prediction gives more freedom in the representation, allowing for both sharp and curved contours to be present. Our network can accept flexible number of points as input, and can reconstruct meshes with varying point distributions. Continuous feedback from previous predictions and incorporating the latent vector of the global structure improves the generation process. The distribution of the mesh interconnections are better captured by REIN.

From our experiments, Meshlab's normal estimation is unable to process surfaces with less than 300 points. REIN can work even with smaller number of vertices. Figure 5 shows some results for meshes with 8 to 300 vertices.

5.3. Additional Experiments

Aside from single hull datasets, ShapeNet meshes with multiple hulls were also examined. Multiple hulls are used

Table 1. Results of REIN compared to BPA and PSR on the ShapeNet and ModelNet10 datasets. BPA and PSR meshes were generated with Meshlab computed normals.

ShapeNet						
	Chamfer Distance (↓)			Point Normal Similarity (↑)		
Algorithm	Hull	Butterfly	Midpoint	Hull	Butterfly	Midpoint
BPA	0.0052	0.0075	0.0177	0.6224	0.8143	0.6995
PSR	0.0871	0.0227	0.0200	0.5795	0.7762	0.7367
Ours (REIN)	**0.0003**	**0.0033**	**0.0028**	**0.8317**	**0.8181**	**0.8313**
ModelNet10						
	Chamfer Distance (↓)			Point Normal Similarity (↑)		
Algorithm	Hull	Butterfly	Midpoint	Hull	Butterfly	Midpoint
BPA	0.0088	0.0106	0.0573	0.7210	0.8062	0.6032
PSR	0.0292	0.0224	0.0292	**0.8273**	0.7938	0.7583
Ours (REIN)	**0.0050**	**0.0056**	**0.0073**	0.8259	**0.8285**	**0.8288**

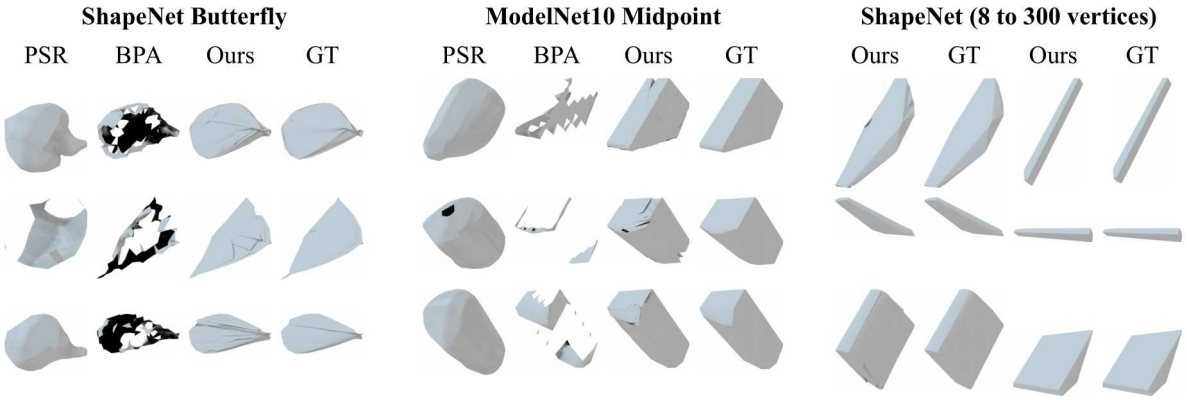

Figure 5. Sample results of our network on meshes from different datasets.

to preserve the shape of the original objects. Two variations were considered for multiple hulls: (1) Wrapped Meshes (Wrapped), and (2) Patched Meshes (Patched). The Wrapped dataset was generated by wrapping the closest surface around the mesh. Blender remeshing was used to create a single continuous surface around the meshes. REIN was retrained on the Wrapped dataset and evaluated separately (the same ShapeNet train/test split was used). No retraining was done for the Patched dataset. The network trained on ShapeNet Wrapped was applied on the test split of the Patched dataset.

Figure 6 shows the results from ShapeNet Wrapped with one mesh for each ShapeNet class. Figure 7 shows results from ShapeNet Patched with one mesh from each class except the car class. The car class was excluded due to the large number of vertices per cluster. Results in Figures 6 and 7 show BPA forming surfaces where vertices are present but is unable to cover the whole object. PSR forms closed surfaces around the whole object but overestimates the boundaries. These characteristics limit the application of BPA and PSR. REIN performs better when applied on small sections of the mesh at a time. Information about past interconnections and vertices are better captured when few vertices are processed at a time.

The evaluation of REIN on the Patched dataset illustrates mesh generation for point clouds with more than 500 vertices. The original point cloud was clustered into several parts. Each part of the point cloud was processed by REIN. Parts are then merged together after obtaining the mesh for each cluster. This shows potential for the network to extend to more vertices.

6. Limitations and Future Work

We demonstrated the ability of our network to generate meshes with 8 to 500 vertices. The upper bound on the number of vertices is caused by limitations of the GPU memory. However, this was overcome by reconstructing by parts. Dense point clouds can be processed with more efficient edge representation or with more GPU memory. Memory use can be more efficient by exploring the sparse matrix representation of the mesh edges instead of using dense matrices. REIN also experiences challenges generating faces from edge predictions. REIN is prone to predicting non-manifold surfaces when edge predictions are imprecise, since face generation assumes a flat continuous surface for accurate results. Future work can explore incorporating a learning-based face prediction network by taking as input the edge probabilities.

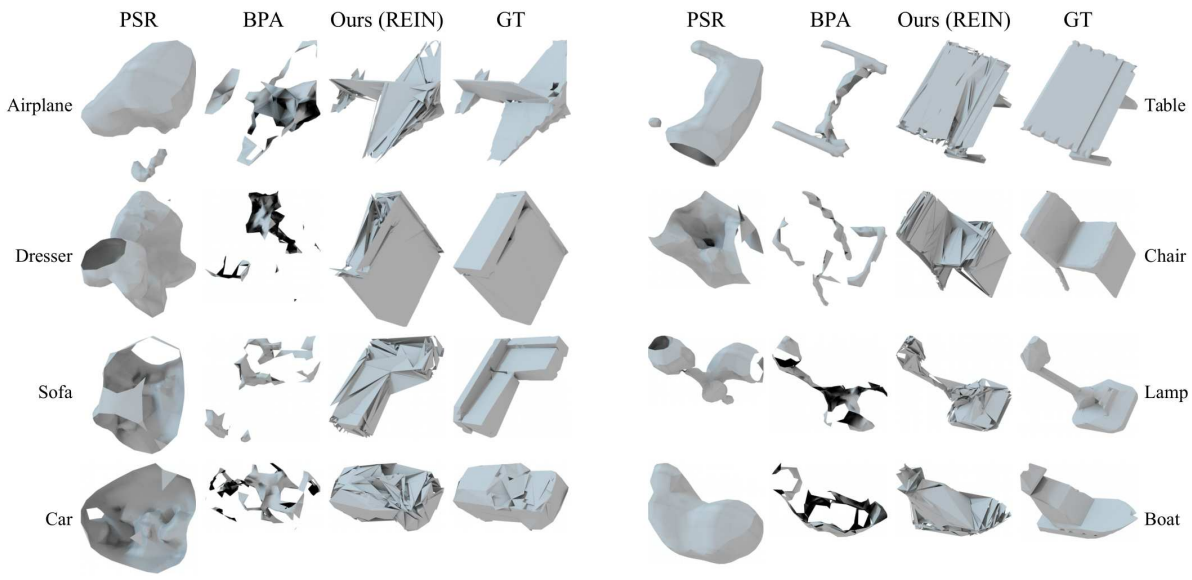

Figure 6. Results of our network on ShapeNet Wrapped compared with BPA, PSR, and ground truth (GT).

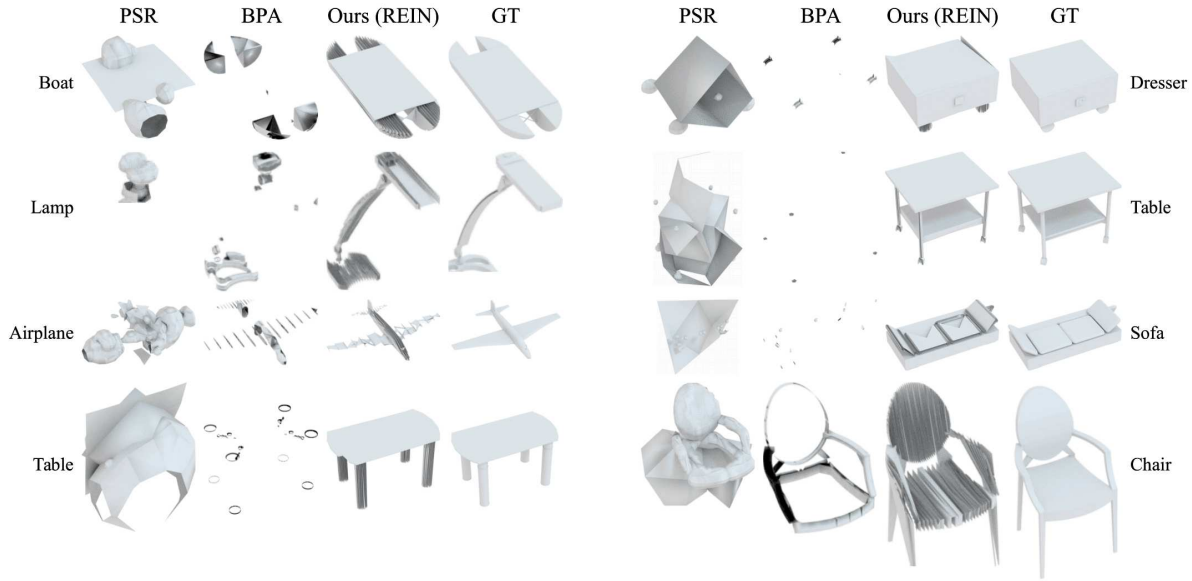

Figure 7. Results of our network on ShapeNet Patched compared with BPA, PSR, and ground truth (GT).

7. Conclusion

We presented REIN, an RNN-based network that can generate meshes with varying number of vertices by sequentially predicting the interconnections. We proposed an architecture that combines extracting a latent vector representation of a point cloud, and combined it with a localized feature predictor using an RNN. REIN was shown to perform better when constructing by parts as shown from the ShapeNet Patched dataset. Our network generates meshes from point clouds better than BPA and PSR for surface reconstruction based on quantitative and qualitative results.

8. Acknowledgements

We thank Joel Casimiro and Aldrin Michael Nilles for their support in creating the figures and the supplementary video of this work. This work was funded by CHED-PCARI Project IIID-2016-005 (AIRSCAN Project).

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017. 1, 2

[2] F Agliardi, GB Crosta, and P Frattini. Integrating rockfall risk assessment and countermeasure design by 3d modelling techniques. *Natural Hazards and Earth System Sciences*, 9(4):1059, 2009. 1

[3] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Cláudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics*, 5(4):349–359, 1999. 1, 2

[4] Fabio Bruno, Stefano Bruno, Giovanna De Sensi, Maria-Laura Luchi, Stefania Mancuso, and Maurizio Muzzupappa. From 3d reconstruction to virtual reality: A complete methodology for digital archaeological exhibition. *Journal of Cultural Heritage*, 11(1):42–49, 2010. 1

[5] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 5

[6] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018. 3

[7] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 5

[8] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016. 1

[9] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136, 2008. 5

[10] Angela Dai and Matthias Nießner. Scan2mesh: From unstructured range scans to 3d meshes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5574–5583, 2019. 2, 5, 6

[11] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5868–5877, 2017. 5

[12] Iván del Pino, Víctor Vaquero, Beatrice Masini, Joan Solà, Francesc Moreno-Noguer, Alberto Sanfeliu, and Juan Andrade-Cetto. Low resolution lidar-based multi-object tracking for driving applications. In *Iberian Robotics conference*, pages 287–298. Springer, 2017. 1

[13] Nira Dyn, David Levine, and John A Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM transactions on Graphics (TOG)*, 9(2):160–169, 1990. 5

[14] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, volume 2, page 6, 2017. 1

[15] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018. 1, 2

[16] Hugues Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *Visualization'99. Proceedings*, pages 59–510. IEEE, 1999. 1

[17] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992. 1, 2

[18] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006. 2

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[20] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017. 2

[21] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018. 2

[22] Khaled Mammou. V-hacd, 2013. 5

[23] Charlie Nash, Yaroslav Ganin, SM Eslami, and Peter W Battaglia. Polygen: An autoregressive generative model of 3d meshes. *arXiv preprint arXiv:2002.10880*, 2020. 2

[24] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016. 3

[25] Jˆrg Peters and Ulrich Reif. The simplest subdivision scheme for smoothing polyhedra. 1996. 5

[26] Jhony K Pontes, Chen Kong, Sridha Sridharan, Simon Lucey, Anders Eriksson, and Clinton Fookes. Image2mesh: A learning framework for single image 3d reconstruction. *ACCV*, 2018. 1

[27] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 2, 3

[28] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 2

[29] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008. 2

[30] Edward J Smith, Scott Fujimoto, Adriana Romero, and David Meger. Geometrics: Exploiting geometric structure for graph-encoded objects. *arXiv preprint arXiv:1901.11461*, 2019. 1, 2

[31] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM transactions on graphics (TOG)*, volume 25, pages 835–846. ACM, 2006. 1

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 3

[33] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2088–2096, 2017. 1

[34] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. *ECCV*, 2018. 1, 2

[35] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8445–8453, 2019. 1

[36] Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. Deep geometric prior for surface reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10130–10139, 2019. 1, 2

[37] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 5

[38] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018. 3

[39] Jesus Zarzar, Silvio Giancola, and Bernard Ghanem. Efficient tracking proposals using 2d-3d siamese networks on lidar. *arXiv preprint arXiv:1903.10168*, 2019. 1

[40] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. Octree-based fusion for realtime 3d reconstruction. *Graphical Models*, 75(3):126–136, 2013. 1