www.cadanda.com
Computer-Aided Design AND APPLICATIONS

Encoding Partial Point Cloud Neighborhoods for Convolutional Neural Networks

Tathagata Chakraborty[1], Hariharan Krishnamurthy[2]

[1]HCL Technologies, tathagata.chakr@hcl.com
[2]HCL Technologies, hariharan_k@hcl.com

Corresponding author: Tathagata Chakraborty, tathagata.chakr@hcl.com

Abstract. The remarkable success of convolutional neural networks (CNNs) on long standing problems in computer vision has led to the recent resurgence of interest in deep neural networks and their applications to problems in other domains. CNNs have been successfully applied to 2D images, videos, and different types of one-dimensional data. Neural networks like CNNs can progressively learn discriminative hierarchical features, thus capturing the underlying structure of the data very effectively yielding state-of-the-art results on problems in many domains.

It is however difficult to extend CNNs to 3D data without losing some structural information. CNNs have been applied to projections of and patches on 3D models. 3D CNNs have also been used with voxelized 3D data with some success. However, the lack of a natural orientation and order of point data in 3D hinder effective utilization of these deep learning-based techniques. In this paper we describe a simple and easy to implement method for discretely encoding partial point neighborhoods for direct input to traditional CNNs. Our encoding method captures and preserves most of the structural information present in a point neighborhood. We show the usefulness of this technique for accurately predicting point coordinates and other properties. In particular, we train a CNN model with our encoding for patching holes in meshes.

Keywords: Convolutional Neural Networks, Point Cloud, Hole Patching, Mesh Completion
DOI: https://doi.org/10.14733/cadaps.2023.290-305

1 INTRODUCTION

The unexpected success of deep learning methods in image and speech recognition has prompted efforts to obtain success on similar problems in 3D. So the past few years has seen growing interest in extending deep learning methods to 3D data. However, such efforts have been impeded by issues related to the nature of 3D CAD data and the limitations of current deep learning techniques. One-dimensional time-series data and 2D image data both have strong structural properties and natural orientations that can be leveraged by CNNs and

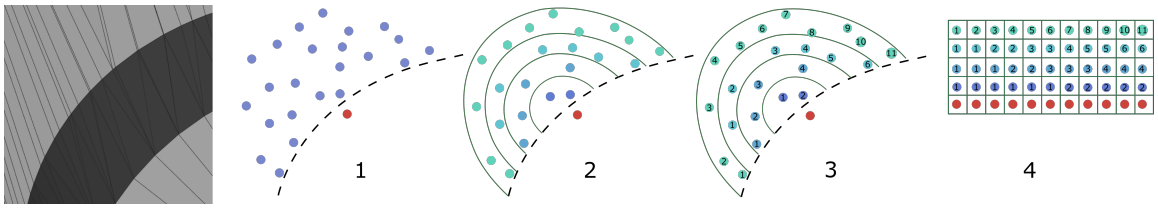

Figure 1: Our method of encoding partial neighborhoods of point clouds comprising: first grouping the neighborhood points (1) into rings (2), ordering the points in the rings (3), and finally oversampling the points in the rings to create a 2D matrix (4) of point coordinates and other properties.

recurrent neural networks (RNNs). On the other end, it is generally expected that the underlying structure, if any, of high dimensional data will be discovered automatically by a deep enough neural network trained on sufficient data. 3D data, for e.g. from CAD models, sits uncomfortably between these extremes.

3D meshes and point clouds have valuable structural information that should preferably not be discarded in pre-processing steps. Neural network models like CNNs can explicitly utilize this underlying structure without requiring a large number of samples. However, there are enough degrees of freedom in 3D, particularly in the way the 3D points comprising the models are oriented and ordered, that it becomes difficult to normalize the data for use with such networks. Neither can the structure of the 3D data be completely abandoned as that would restrict its usefulness to global inference tasks such as object recognition and classification.

Each current approach to deep learning in 3D therefore entails different issues of its own. 3D CNNs on voxelized data are both memory and computation intensive. Surface inaccuracies introduced by voxelization also preclude accurate prediction of local surface properties. Techniques where the point data is embedded in a high-dimensional space and then aggregated for pose and order invariance cannot be easily extended for local estimation tasks. Projection and patch-based methods can be used for tasks such as mesh completion, local estimation, and mesh in-painting, but they are not always very accurate. Graph-based methods have been used for shape correspondence and retrieval, but they haven't yet been investigated for local interpolation tasks.

Our method is inspired by a combination of ideas from graph-based [10][12][9] and patch-based methods [15][19][20]. The method works directly on untransformed point clouds and is comparatively simple to implement. The method does not requires data normalization such as the conversion into height maps needed in patch-based methods. Nor does the method require conversion to polar coordinates or any special neural network machinery used in graph-based methods. We encode a point and its neighborhood as a 2D image containing raw point coordinates and properties which can be directly used to train a standard CNN model.

The method can be used to predict and interpolate points and their properties; we illustrate the method in the context of patching holes on smooth surfaces. Our broad approach for patching holes in meshes is based on the advancing front triangulation method described in [22] but we estimate the points using a CNN instead of using the Moving Least-Squares (MLS) technique. An overview of the encoding process is shown in Fig. 1.

The rest of the paper is organized as follows. First we briefly survey some traditional and modern learning-based techniques for patching holes in 3D meshes and then review literature on deep learning techniques for point cloud completion which are potentially adaptable for local estimation tasks. Additionally, we survey some general techniques for deep learning on 3D data. All these surveys are contained in Sec. 2. This is followed by a detailed description of our approach for encoding partial point neighborhoods in Sec. 3. In Sec. 4 we describe the CNN model architecture and discuss issues with creation of training data, training of the model and its use for point prediction. We present the results in Sec. 5 and, and compare them with the results from the more traditional MLS-based technique. We provide conclusions in 6 and indicate some directions for future research.

2 BRIEF SURVEY OF RECENT TECHNIQUES

Mesh patching has been extensively studied and there are numerous techniques applicable to different problems depending on the required outcome. All mesh patching techniques are essentially based on understanding the local neighborhood around the hole. Traditional methods approximate the neighborhood by fitting a low-degree polynomial or a smooth surface over the surrounding points. These methods thus "learn" the local neighborhood on the fly. In contrast, modern methods learn larger regions using more elaborate models having many more tunable parameters. These modern methods must be pre-trained on the hole neighborhood; typically no learning happens while the hole is being patched.

We first present a brief overview of some traditional methods as we use a triangulation technique from a traditional method and our results are compared with that technique. Thereafter we briefly discuss some learning-based techniques for patching holes and completing point clouds, and lastly we broadly survey some techniques for deep learning on 3D data.

2.1 Traditional Techniques for Patching Holes

There are many traditional hole filling techniques; see [14] for a detailed survey and comparison of methods. Many of these are patch-based techniques where local patches around the hole boundary are considered one-by-one and the interior of the hole is then interpolated from these neighborhood patches. For patching smooth surfaces, an effective method is to fit a bi-quadratic or a bi-cubic polynomial on the neighborhood of a point near the hole boundary. This point is then projected on the polynomial surface to improve the point estimation.

The process of iteratively interpolating from the boundary of the hole by locally fitting surfaces using a least squares approach is known as the MLS technique (see [8][23]). In [22] the MLS technique is used with a boundary triangulation method for filling holes on smooth mesh surfaces.

A Poisson equation can be used to fit a smooth surface in place of a low-degree polynomial as in [33]. Radial basis functions have also been used to interpolate the surface between neighborhood regions [3][29]. Volumetric approaches for patching multiple holes simultaneously are presented in [16]. Iterative subdivision of a coarse initial triangulation with smoothening is studied in [28]. Several variations of these techniques and other approaches are also reported in literature.

2.2 Learning-based Techniques for Patching Holes

More recently, several learning-based techniques have been investigated for patching meshes. In [15], a method for in-painting 3D meshes based on dictionary learning and sparse coding is described. The mesh is first subdivided into small patches, and the patches are converted into height maps by fitting a plane on points in the patch and computing the signed distance of the points from this plane. The height maps are then learned by a sparse coding model. The method tries to approximate the geometric texture in the region of the hole. It can fill holes on non-smooth surfaces.

A similar dictionary-based learning method for patching small holes is proposed in [19], with the dictionary learnt from many different shapes. This work is extended in [20] by using a CNN-based autoencoder. The mesh is sampled into rectangular patches, which are then projected onto locally fitted planes and converted into height maps. The CNN autoencoder is trained by artificially creating holes in these patches and letting the model reconstruct the original patches.

2.3 Point Cloud Completion

Point cloud completion is a recent research area made possible by modern deep learning methods. The aim of point cloud completion is to create dense point clouds from sparse ones; these methods could potentially be adapted for local feature estimation. We therefore include a brief survey here.

In [32] a method for completing a sparse point cloud, Point Completion Network (PCN), is presented based on the PointNet architecture [17]. Here two stacked layers of PointNet are used in an encoder-decoder configuration to predict a coarse point cloud. Patches from the coarse point cloud are then sampled and passed through another decoder to create a denser point cloud. Similarly, a gridding residual network is described in [30], where the input point cloud is converted into a differentiable grid and a 3D convolutional encoder-decoder network is used to learn from the grid. Additional processes are designed to recreate a coarse point cloud and recover local details.

In [2] a point cloud completion method using a Variational AutoEncoder (VAE) is described that learns the latent space of mesh patches across a large number of meshes. This generalizes better to unseen cases, while also allowing the learned model to be pose invariant. In [25] an autoencoder model is trained on complete point clouds to generate large latent features. A different network is then used to train partial point clouds and a Generative Adversarial Network (GAN) is used to help align the generated features. A VAE is used in [13] to create a dense point cloud using a similar coarse-to-fine process.

2.4 General Deep Learning Techniques for 3D

Deep learning in 3D is an active area of research. The focus is largely on object classification, recognition and shape retrieval; these same tasks have also been investigated using traditional techniques. A comparative study of the various local 3D shape descriptors forming the backbone of many traditional techniques is done in [4]. We briefly point to some popular methods of deep learning for 3D object classification and recognition, and describe some techniques of local estimation. For more comprehensive surveys see [6][1][5].

CNNs have been applied to 2D projections of 3D data for 3D object classification leading to state-of-the-art results [21][31][7]. 3D CNNs have been used with voxelized 3D data and octrees for shape analysis and retrieval [27][18][24]. Points in 3D point clouds have been embedded into high-dimensional spaces [17] and high-dimensional feature descriptors of mesh vertices have been learned using graph-based methods [10][12].

The use of deep learning for local estimation and interpolation in 3D meshes has received relatively less attention; this is possibly due to lack of sufficient data for training from a single model among other reasons. Deep neural networks with a large number of parameters require a large amount of training data and self-sampling from a single 3D model may not always be sufficient. In spite of this, CNNs along with GANs have been trained on small patches sampled from a mesh surface for in-painting meshes [20][26].

In [11] a method for consolidating point clouds by learning over a large set of globally distributed data is described. The objective here is to obtain a uniform and dense point cloud with properly oriented normals and enhanced sharp features. The input point cloud is self-sampled globally to create a large number of unique samples from a single object. In [10] a geodesic convolutional neural network which extends the CNN to non-Euclidean manifolds is proposed. The geodesic neighborhood of a point in a triangulated mesh is encoded using polar coordinates and then binned based on the polar angle and distance. Convolutions are then performed by sliding a kernel over the mesh. The authors of [9] propose to encode a neighborhood of points in a mesh by enumerating the points in a spiral. A fixed length feature vector is then created and trained using a RNN.

From a broad perspective, our encoding approach is similar to the work described in [10][9], while our application and model are similar to the ones in [20]. We use the advancing front triangulation method proposed in [22], but replace the MLS with interpolation from a CNN model.

3 ENCODING PARTIAL NEIGHBORHOODS

In this paper we describe a method for discrete encoding of partial neighborhoods of points in point clouds. Here, by a partial neighborhood we mean a neighborhood with a span less than the complete 360° around a point. The restraint to use partial data when complete data is available can be counter-intuitive but it can help

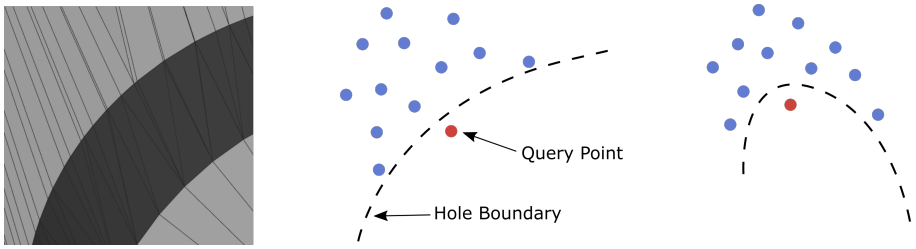

Figure 2: Nature of neighborhood of query point with the advancing front: (left) portion of a small hole to patch, (center) typical neighborhood of a query point (colored red) near the hole boundary (dashed curve), (right) neighborhood (colored blue) on the contracted advancing front.

order the data consistently. The encoding method works with point clouds, triangular meshes and parametric models. Points clouds represent a common form of 3D data, and are also a convenient representation for illustrating our method. In the context of patching meshes, we therefore assume that the faces around the hole boundary can be approximated by a uniformly distributed point cloud; we further assume that such an approximation is available in a data structure suitable for efficient nearest neighbor queries. These are computationally less demanding pre-requisites and are also required by traditional hole patching methods.

In an advancing front technique, the hole is patched by progressively interpolating from the boundary and towards the interior of the hole. We refer to a point to be interpolated near the boundary of the hole as the query point. The neighborhood of a query point in the advancing front of the hole boundary then typically looks like that shown in Fig. 2. In most cases, the neighboring points are largely on one side of a plane passing through the query point (Fig. 2 center), but as the front advances and the hole correspondingly contracts in size the neighborhood of points tends to surround the query point from all sides (Fig. 2 right). Importantly, throughout most of the patching process the neighborhoods of the query point remain partial.

During training a query point is selected at random from the support surface of the hole, and the partial neighborhood as seen along a random direction in the surface around the query point is considered. In each iteration of the advancing front method, the current boundary is first subdivided into small segments. Query points are then determined by moving a small distance from the mid-point of each boundary segment in a direction tangent to the hole surface. The details of the method are as described in [22].

3.1 Grouping into Rings

We first identify the points in a sufficiently large neighborhood of the query point. The neighboring points are then grouped into several "rings" based on their distance from the query point (see Fig. 3). We took the consecutive ring radii as linearly increasing multiples of the point cloud resolution r (that is, r is the mean distance between two neighboring points in the point cloud) in our experiments. This is also the resolution we use to advance the hole boundary. The ring radii may be differently determined depending on the application.

Each ring contains a number of points approximately proportional to the area of the ring. For example, consider rings with radii $2r$, $3r$, $4r$ and so on. For the first ring with radius $2r$ the area is $\pi(2r)^2 \approx 12r^2$ and hence we expect to find approximately $12/2 = 6$ points in the ring (halved due to the partial nature of the neighborhood). Similarly the next ring has area $\pi(3r)^2 - \pi(2r)^2 \approx 15r^2$ and would typically contain 7 or 8 $(15/2 = 7.5)$ points. Alternatively, one may choose the sequence r, $\sqrt{2}r$, $\sqrt{3}r$, ... as ring radii such that an approximately an equal number of points are enclosed by each annulus; this is shown in the right of Fig. 3

The underlying assumption here is that the neighborhood of a hole is a 2-manifold and the local neighborhood around the query point is Euclidean. This is a reasonable assumption when patching holes on smooth surfaces and most traditional patching methods bank on this assumption. However, this assumption may

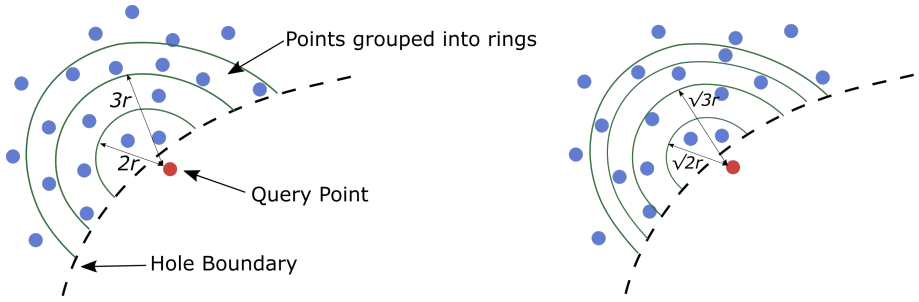

Figure 3: Choice of the sequence of ring radii: (left)rings with linearly increasing radii, (right) non-linearly increasing radii to enclose an equal number of points in each annulus.

not be important when using learning-based methods since we are not trying to fit a smooth surface on the neighborhood. In fact, as we show later, learning-based methods work better on non-smooth surfaces; in this respect, learning methods can be seen as a generalization of traditional hole patching methods to all kinds of surfaces.

3.2 Ordering Point in Rings

The point encodings described in the previous literature [10][12][9] are not rotation invariant. In general, it is impossible to uniquely order a complete $360°$ neighborhood around a point. However, partial neighborhoods can be ordered consistently as follows. We first select a random point P_r from the ring. We denote an estimation of the query point by Q and the estimated normal at this point by N. For each of the rest of the points P_i in the ring, we compute the cross product $C_i = (P_i - Q) \times (P_r - Q)$. Then the dot product $d_i = (P_i - Q) \cdot (P_r - Q) \times sign(N \cdot C_i)$ gives us the signed distance of the rest of the points P_i from the point P_r based on which the points inside a particular ring can be ordered. The computation is visually illustrated in Fig. 4.

The method for encoding partial neighborhoods in point clouds was impelled by the application of the method for patching holes in meshes. One possible reason that such partial encodings, where a unique point ordering is possible, have not yet been investigated much is because applications such as hole filling have not typically not been targeted in deep learning research. However, we note that such an encoding is applicable to other domains apart from hole patching. Our encoding could possibly be used effectively for other tasks such as object segmentation and shape correspondence.

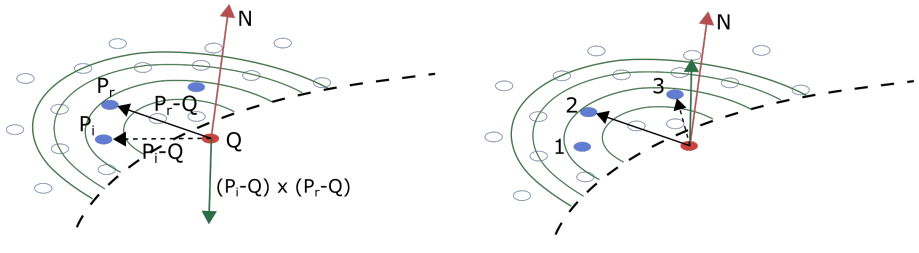

Figure 4: Ordering of points in rings: (left) a negative value from signed distance computation, (right) positive signed distance due to the two vectors' cross product being along the normal; sorted order of points points in the ring also shown.

3.3 Oversampling to Create 2D Images

With the ordered set of points in each ring, we need to oversample the data in each ring next, to obtain fixed sized vectors which are then stacked to form the rows of a 2D "image". Such a 2D image has one more row than the number of rings, where each of the rows, except the last, corresponds to each ring and the additional last row to the query point. The number of columns in the image is set to the number of points in the outermost ring, although a larger or a smaller number can be used depending on the resolution of the point cloud. Typically, each of the smaller inner rings will contain fewer points than required to fill the image rows. Therefore, the points in the rings must be oversampled while preserving their order to artificially "stretch" the data vector and create an equal number of points for each image row. This process is illustrated in Fig. 5.

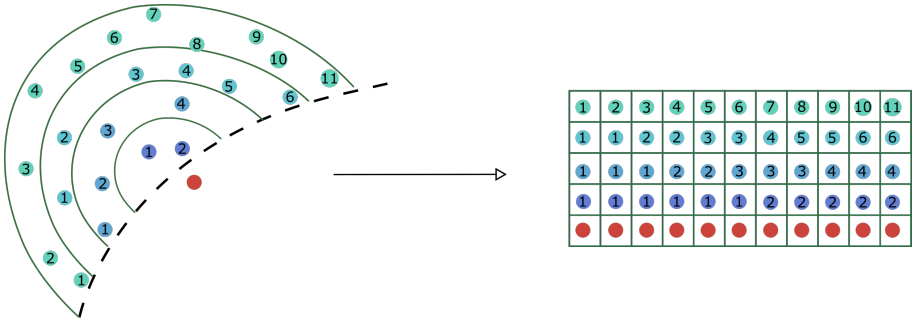

Figure 5: Creation of "images" from ordered points in rings: (left) ordered points in each ring, (right) encoding by oversampling points to fill the columns in the image.

In Fig. 5 we can see 4 rings in the point neighborhood and $4+1=5$ rows in the image. The query point is indicated in red and the rest of the points in each ring are numbered from 1 indicating their clockwise order within the ring. There are 11 points in the last ring, and let's consider 11 columns for the 2D image. In the top row of the image, the points from the fourth, the outermost ring, are placed in the order in which they occur since there are exactly 11 points available. In the second row from the top many points from the third ring are duplicated, since there are only 6 points in the ring. Similarly in the third row from the top, points from the second ring are repeated thrice; a similar repetition happens in the fourth row. In the fifth and last row an estimate of the query point is repeated throughout (i.e. 11 times).

Oversampling the ordered point data in the partial neighborhoods to create 2D images allows us to directly use standard CNN architectures without having to invent special convolutional kernels and other neural-network machinery. Such a 2D image representation also preserves much of the structure present in both the circular and radial directions around the query point. The only structural information loss is due to the nature of the point distribution inherent in the approximation of the neighborhood as a point cloud which may not be uniform with respect to our discretization scheme in many cases.

3.4 Data in the 2D Images

The exact data stored in these 2D images will depend on the type of point property to be estimated. Here some creativity will often be required and some inductive bias should necessarily creep in before the CNN model can learn from the data and interpolate well. We tried several configurations of input data and combinations before we found the set of data, i.e. point properties to be stored as data, that works reasonably well for interpolating point coordinates.

In our first attempt we used the unmodified x, y and z coordinates of the neighborhood points as data to create three 2D images, similar to (r, g, b) data in input images with CNNs. The label to learn was the correct

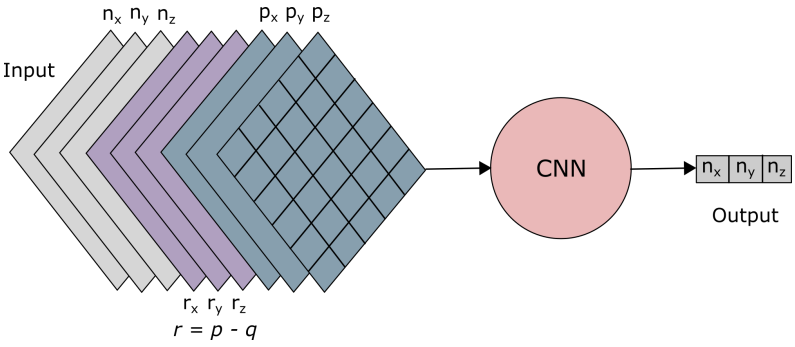

Figure 6: Input and output data of the network - here n represents normals, q the query point, p the absolute coordinate, and r the relative coordinate of the neighboring points.

(x, y, z) coordinates of the query point. It was very difficult to train this network. One reason was that the correct coordinates were often very close the query point's coordinates which were present in the bottom row of the 2D image. This resulted in a very small error and a correspondingly shallow gradient during training. The network therefore quickly learnt to assign a high weight to the bottom row and a very low weight to everything else, and failed to capture the underlying surface structure. We tried to remedy this by normalizing the point coordinates using various techniques but with little success.

Our first breakthrough involved using the normals of the points instead of the point coordinates of the neighborhood points and trying to predict the correct normal of the query point. With this change, the network training improved significantly. Normal data by its very nature is normalized and is thus easier to learn. Second the normals for the initially estimated query point and the accurate normal were often sufficiently different. This allowed for a proper gradient based optimization of the network. After estimation the query point was pivoted about the edge opposite to the point to align its normal with the predicted normal. However, this procedure was still not accurate enough to patch larger holes smoothly. A few wrong predictions could waylay the interpolation as errors would magnify with the advancing front.

Two additional elements were required before sufficiently accurate results could be obtained. Some error in estimation could be attributed to the slightly different placements of the points in the sampled point neighborhoods. To account for this variation, we added another three layers to the 2D image to store the relative point coordinates of the points in the neighborhood with respect to the query point. Lastly we also added the original point coordinate data for the neighboring points. This is required as the network must ultimately learn several smooth functions over the supporting surface of the hole. However, absolute coordinate data is needed by the network to be able to decide which function to use at which location on the surface. The structure of the complete input data and label for our network is illustrated in Fig. 6.

4 CNN MODEL ARCHITECTURE AND TRAINING

4.1 Model Architecture

Once the input data has been converted to 2D images, standard CNN model architectures, that are proven to work in other domains, can be used for training and prediction. For hole patching we use a simple CNN with 5 convolutional layers with 3×3 filters followed by 5 fully connected layers. Our input is a set of 9-layered 2D images, each 8×24 in size ($7 + 1 = 8$ rings each containing 24 point data). The small size of the input image limits the number of convolution layers we can use without extra padding in each convolution step, since each convolution pass reduces the size of the image. Padding would not be preferable for hole patching because it would dilute the point data in the rest of the image and significantly reduce the prediction accuracy.

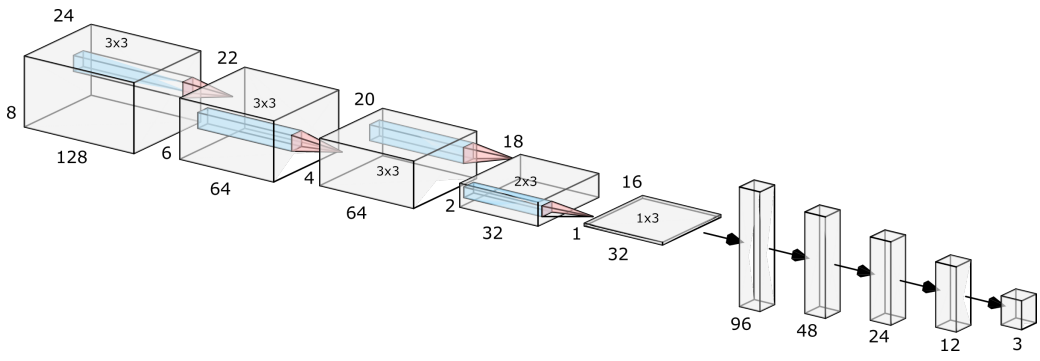

Figure 7: CNN network architecture for 2D images of size 8×24 created from the encoding.

The number of convolutional and fully connected layers can be chosen by experiment, and depends on the training time and computational resources available. In general, larger networks would require more data to train effectively. There is no systematic method for arriving at the best configuration of layers either, and one must try different combinations of layers and use the model that performs the best. The architecture we use is illustrated in the Fig. 7. Note the lack of any max-pooling layers as the initial 8×24 images are too small already. There are also no normalization layers since the network is not very deep and so may not suffer much from vanishing gradients; that said, further experiments would be required to confirm if a form of normalization or regularization would result in improved performance or accuracy.

4.2 Training the Model

Learning the support surface of a hole for patching it is an unsupervised learning task. However in most learning-based approaches the problem of patching holes is transformed into a supervised task by creating artificial holes on patches sampled from the support surface. The network is then trained to recreate the original patches from the ones with the holes. We take a slightly different approach. In our method an approximation of the point to be estimated is already included in the input to the model (as the last row of the 2D image), and the model learns to predict the correct value of the point normal from this approximation and the neighborhood data. Therefore, during training we need to create such artificial approximations to train the CNN model.

We first identify the support surface of the mesh hole using edge-angle-based heuristics. The support surface is then approximated with a dense uniform point cloud. A random point is selected from this point cloud. With the random point as center, a random partial neighborhood with an angle span between $180°$ to $270°$ is selected. We group the points in this neighborhood into 7 uniformly spaced rings with the randomly selected point as the center. Sometimes, when the selected random point is near the boundary of the surface or the hole, the neighborhood may not contain enough points in all the rings. We use heuristics to discard such sparse neighborhoods and only consider dense neighborhoods for training.

In order to create the artificial point and normal approximation we first find the three nearest non-collinear points to the randomly selected center point. We then use the projection of the center point on the plane defined by the three points as the initial estimate of the query point and the normal of the plane as the approximate normal. The approximate point and normal are then used to fill in the bottom row of the point encoding image, and the original center point normal is used as the ground truth or label to be predicted.

In the results presented below, around 2000 random points and their neighborhoods were sampled from the support surface of the hole for training. Of these around $300 - 400$ sparse neighborhoods had to be discarded in each case. The model was trained with a learning rate of 10^{-4} and a batch size of 256 for 1000 epochs with

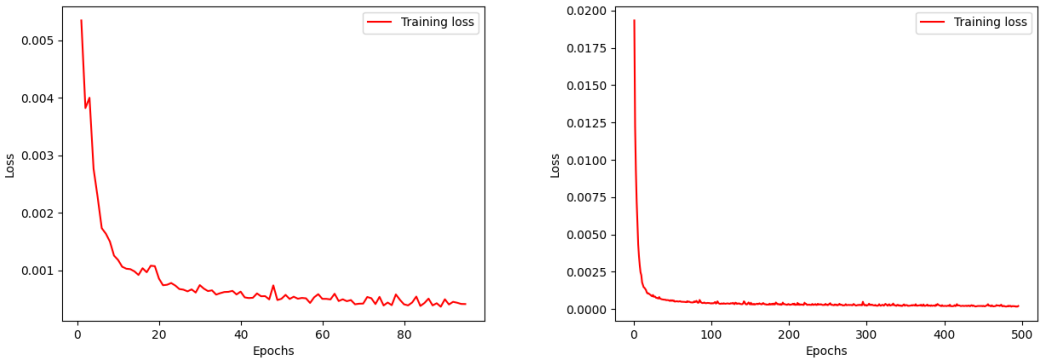

Figure 8: Trend of training loss with epochs: (left) loss (mean squared error) over 100 epochs, (right) loss over 500 epochs.

mean squared error as the loss function. The model took about an hour to train on a laptop-grade CPU. The training time would be reduced to a few minutes on a modern GPU. Slightly better results are obtained with a lower learning rate of 10^{-5}, but this rate required around 5000 epochs to train, and increased the total training time by a factor of $7-8$. Once training is complete, the time required to patch the hole is within a couple of minutes and is comparable to the time taken when using MLS. Since in the case of MLS no pre-training is required, a CNN-based approach garners little to no benefits over MLS when patching holes in smooth surfaces. However, as discussed in the results in section 5, a CNN-based method like ours is better able to patch holes spanning non-smoothly connected surfaces and approximate the surface texture when patching rough surfaces. Techniques which are MLS-based or fit smooth surfaces on the other-hand cannot be used at all in such cases.

Fig. 8 shows the learning graphs for an example support surface over 100 and 500 epochs. These graphs are fairly typical for deep learning models. For hole patching, the loss stops decreasing beyond $700-1000$ epochs. One reason for requiring so many training epochs is that the input data has only around 1500 distinct data points. Even so our network (see Fig. 7) with over $185,915$ tunable parameters can learn to generalize effectively instead of simply memorizing the input data. This is due to the various regularization effects implicit in the input data and the architecture of the CNN model. In our case we have used the mean squared error as the loss function, although the cosine distance loss works equally well. The latter makes more sense since it measure the angle difference between the predicted and correct point normals.

In deep learning, it is standard practice to train the network on a training set and validate it on a smaller validation set. A validation set ensures that the trained model generalizes to unseen data well. For the hole patching application, we decided to forego the use of a validation set. Since the amount of training data is often very small, every bit of additional data helps improve the accuracy of the model.

4.3 Point Prediction and Triangulation

We follow the boundary triangulation scheme described in [22]. The idea is to loop around the boundary of the hole repeatedly, closing sharp concave corners and creating new triangles on the other edges. For this, the boundary loop of the hole is first split into smaller resolution-sized edges. In every iteration we first loop around this segmented boundary to find sharp concave corners and eliminate such corners by joining the end points of the edges adjoining the concave corner to create a new triangle. The hole boundary is also updated. It is noted that this triangulation step requires no interpolation.

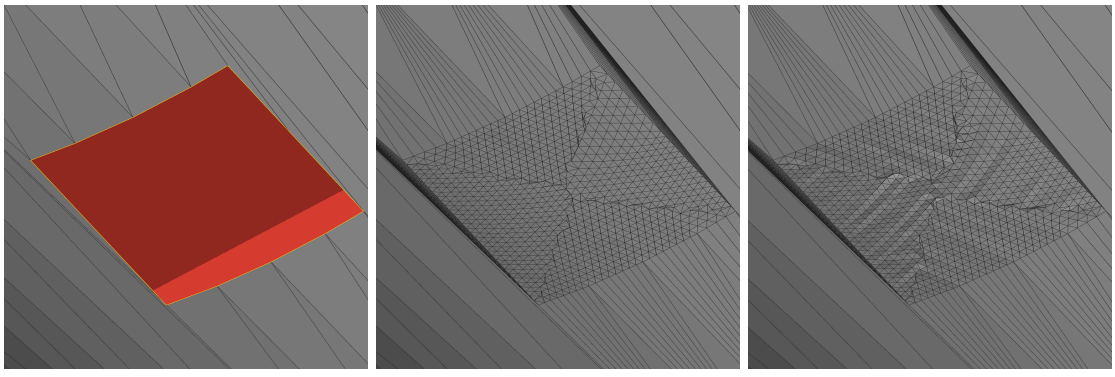

Figure 9: Results with small hole on a smooth surface: (left) hole to patch, (centre) results using MLS, and (right) result from our CNN model.

Next we loop around the boundary once more and next to all the boundary edges we create a new point on the perpendicular bisector of this edge (at a distance of $\sqrt{3/4}r$ from the edge) in the direction of the hole interior. Using this point as the query point, we find the encoding of the neighborhood using our method and find the predicted normal using the CNN model. The new point is then "pivoted" about its base edge to align its normal with the predicted normal. The boundary is again updated. Before a new triangle is added to the mesh, we additionally check to ensure that these new triangles do not intersect the neighboring triangles of the mesh. If an intersection is found, the triangle is discarded. These two steps are repeated till the complete hole is patched. Sometimes a small patch may be left behind near the center of the hole, which may be triangulated using a simple polygon triangulation algorithm.

5 RESULTS AND DISCUSSION

Figures 9, 10 and 11 show some results of patching meshes using our CNN model alongside the results obtained using the traditional MLS method, where a bi-quadratic surface is fitted locally. In Fig. 9 it is seen that results from the CNN model (right) are not as smooth as that obtained using MLS (center). This slight amount of error may be unavoidable in the case of the CNN model and may be inherent in the pseudo uniform nature of the point cloud approximation of the surrounding surface, and limited by the complexity of the CNN model. A deeper CNN model with many more parameters which is also trained on more samples from a denser point cloud may help to reduce this error. However, the improvement would also entail a longer training time and a computationally-costlier prediction function at runtime.

The triangulation is similar in both cases since the same triangulation algorithm and code are used. A resolution of $0.5mm$ is used for both the point cloud approximation and the triangulation (that is, the average edge length in the triangulation is $0.5mm$) for all the results. Fig. 10 shows the results on a larger hole, and again we see that the results from the CNN model are not as smooth as that obtained using the traditional MLS method. The undulations in the result from the CNN though are very small and between $0.1 - 0.2mm$. A post-processing smoothening step can be used on top of the CNN results to obtain patches that are as good as those obtained from MLS.

Fig. 11 shows the results on a hole spanning a faceted surface where the meeting regions have been filleted with a $5mm$ radius. In this case we see that the MLS leads to a more undulating surface as the algorithm tries to fit a bi-quadratic surface near the edge between the fillet and the flat surface. The CNN model on the other hand performs much better when patching such holes and is better able to maintain the flat regions in the patch. The reason for the lesser accuracy of MLS is that it uses only a very small region of the point

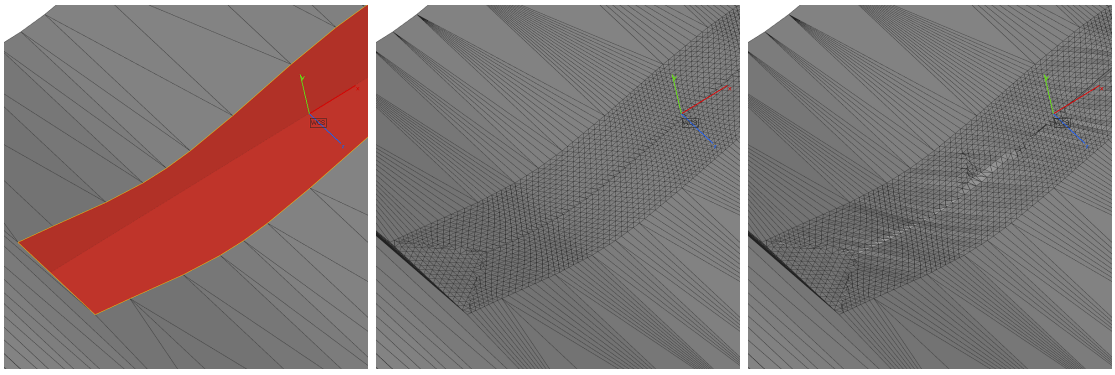

Figure 10: Results with a large hole on a smooth surface: (left) hole to patch, (centre) results using MLS, and (right) result from our CNN model.

neighborhood when estimating a point and cannot account for sudden changes in the curvature of the surface. A CNN model, on the other hand, can learn much larger regions of the model during training, and can learn the varying underlying structure of a non-smooth surface. In fact, the more faceted and noisy the surface the better the CNN model is likely to perform compared to the MLS. This suggests that these techniques are somewhat complementary in nature and could possibly be used synergistically for patching different regions of a large complicated mesh, like those acquired from 3D scanning methods.

Some triangulation errors can be observed in the Figures 10 and 11 which are due to problems with the robustness of the triangulation algorithm as briefly described in the previous section; we hope that the errors in triangulation will not detract the quality of the interpolation results from the CNN model. The lack of a robust triangulation algorithm for noisy point clouds also prevents the illustration of the hole patching results of the CNN model on a more noisy surface at this time. We hope to remedy these shortcomings in future work.

5.1 Benefits Over Traditional Techniques

In general, most traditional techniques like MLS try to approximate the point neighborhood by fitting smooth surfaces on the points in the neighborhood. These techniques therefore are unable to preserve the geometric texture of the surface to be patched. CNN-based and other learning based methods, on the other hand, can learn the geometric texture of the surface and reproduce it in the region of the hole (see [15], [19] and [20]). In this paper we have shown that with proper encoding and training, CNN-based models can also patch smooth surfaces with good accuracy, although it would take a relatively long time to train such a model.

To get some intuition behind the appropriate applications of traditional techniques and learning-based methods we consider 2D cross-sections of different surfaces, as shown in Fig. 12. Fig. 12 (a) shows the cross-section of a low curvature surface. Such surfaces are easily patched using traditional techniques like MLS, because locally a large neighborhood n_a of the surface can be approximated well by a bi-quadratic surface. Fig. 12 (b) shows a smooth undulating surface with multiple high curvature regions compared to that in (a). Such surfaces can also be patched using traditional techniques like MLS, albeit a much smaller neighborhood n_b must be selected and more densely sampled to fit the bi-quadratic surface. It is often difficult to determine an appropriate size of the neighborhood especially in regions where the curvature changes quickly. Additionally, for surfaces that combine low curvature regions like (a) with higher curvature portions like (b) it can be impossible to determine a single fixed-size neighborhood for fitting the bi-quadratic surfaces. In faceted surfaces like Fig. 12 (c), where the curvature changes abruptly from 0 to infinity, traditional techniques based

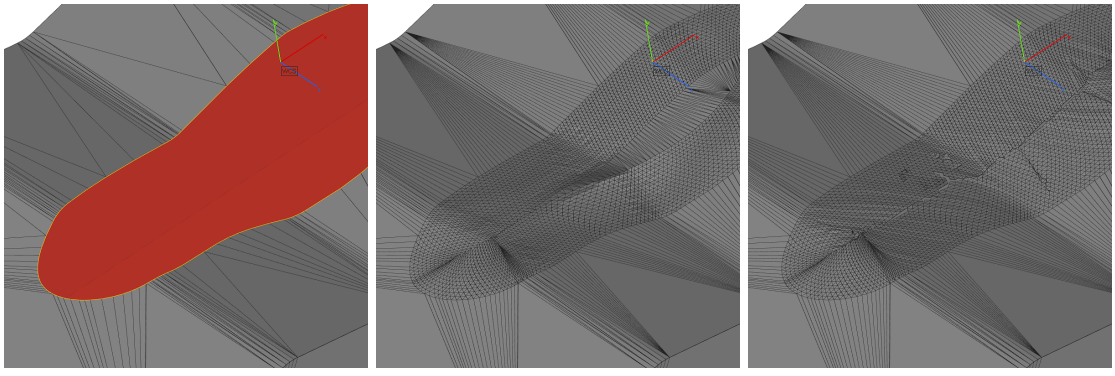

Figure 11: Results for a hole in a filleted meeting region: (left) hole to patch, (centre) results using MLS, and (right) result from our CNN model.

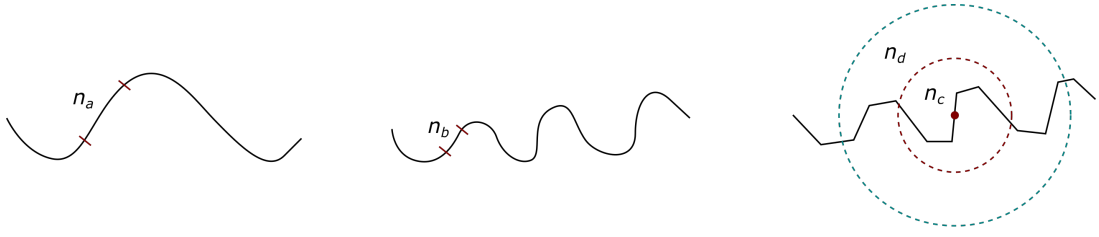

Figure 12: Effectiveness of hole patching approaches under different smoothness levels: (a) a low curvature surface, (b) a surface which a higher curvature, (c) a faceted surface

of surface fitting no longer work well, since it becomes impossible to fit surfaces approximating the area around the sharp edges. It is in such cases that learning-based methods like CNN models can outperform traditional methods, since they are able to learn and interpolate across a much more complicated surface. It must be noted that even in a learning-based method one must choose an appropriately sized neighborhood to train the model. However, here one can err on the conservative side and choose a large neighborhood like n_d for training. This makes it possible to patch a surface without having to increase the sampling density, which can increase the patching time and cause error magnification when using front propagation strategies.

6 CONCLUSIONS

In this paper we describe a simple and easy to implement method for encoding partial neighborhoods of points for use with CNNs and other deep learning techniques. The encoding allows us to capture the local structural information without the loss incurred due to aggregation and normalization used for the sake of orientation and order invariance in other techniques. Instead of considering full rectangular or circular patches we suggest sampling partial patches from the mesh to train a neural network model. For patching holes in meshes, we sample patches which occur in a region sized between a semicircle and a three-quarter circle in size from the support surface of the hole. Choosing a partial neighborhood allows us to order the points in the neighborhood consistently and prevents structural information loss in the input data fed into the neural network. The method is designed to predict and interpolate local mesh properties, and can be adapted for use with different deep learning models. We believe that the idea of partial sampling and ordering can be fruitfully

applied and extended to other domains.

Creativity is still required in choosing the right kind of input data to enable the network to learn the underlying surface structure. For example, it took some ingenuity and much experimentation to discover a method to accurately predict point coordinates by first predicting the normal instead of directly predicting the point coordinates. The normal data is by definition normalized and this enables us to quickly train the model without requiring a lot of data. The results using the CNN model are not as smooth as those by a more traditional technique like MLS especially for smooth surfaces. However, for non-smooth surfaces a CNN-based model outperforms MLS and other traditional techniques which were not designed for non-smooth surfaces.

Our encoding approach opens up several avenues for future exploration. It would be interesting to see how well sequence models like RNNs and transformer networks work with such an encoding. RNNs and transformer networks would allow for a variable number of rings in our scheme and would thus enable better patching near hole boundaries where a sufficiently-sized support surface available. We plan to experiment with larger neighborhoods and other measures such as geodesic distance for grouping the points. Training of the neural networks with more oriented neighborhoods (for example, where the neighborhoods are all aligned with the average normal of the curve of the hole's boundary nearest to it) and using pre-trained networks for transfer learning are likely to lead to better results and faster training times.

ACKNOWLEDGEMENTS

This research was conducted as part of the CAMWorks project. We are thankful to Nitin Umap, Baburaj Iyer, Vivek Govekar and Swadhin Bhide for giving us the opportunity and time to conduct the present research. We would also like to thank our colleagues from the CAMWorks team for their support, help and ideas.

Tathagata Chakraborty, http://orcid.org/000-0000-2752-2533
Hariharan Krishnamurthy, http://orcid.org/0000-0002-3059-4132

REFERENCES

[1] Ahmed, E.; Saint, A.; Shabayek, A.E.R.; Cherenkova, K.; Das, R.; Gusev, G.; Aouada, D.; Ottersten, B.: A survey on deep learning advances on different 3d data representations. arXiv preprint arXiv:1808.01462, 2018.

[2] Badki, A.; Gallo, O.; Kautz, J.; Sen, P.: Meshlet priors for 3d mesh reconstruction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2849–2858, 2020. http://doi.org/10.1109/CVPR42600.2020.00292.

[3] Chen, C.Y.; Cheng, K.Y.; Liao, H.Y.M.: A sharpness dependent approach to 3d polygon mesh hole filling. In Eurographics (Short Presentations), 13–16, 2005.

[4] Guo, Y.; Bennamoun, M.; Sohel, F.; Lu, M.; Wan, J.; Kwok, N.M.: A comprehensive performance evaluation of 3d local feature descriptors. International Journal of Computer Vision, 116(1), 66–89, 2016. http://doi.org/10.1007/s11263-015-0824-y.

[5] Guo, Y.; Wang, H.; Hu, Q.; Liu, H.; Liu, L.; Bennamoun, M.: Deep learning for 3d point clouds: A survey. IEEE transactions on pattern analysis and machine intelligence, 43(12), 4338–4364, 2020. http://doi.org/10.1109/TPAMI.2020.3005434.

[6] Ioannidou, A.; Chatzilari, E.; Nikolopoulos, S.; Kompatsiaris, I.: Deep learning advances in computer vision with 3d data: A survey. ACM Computing Surveys (CSUR), 50(2), 1–38, 2017. http://doi.org/10.1145/3042064.

[7] Kalogerakis, E.; Averkiou, M.; Maji, S.; Chaudhuri, S.: 3d shape segmentation with projective convolutional networks. In proceedings of the IEEE conference on computer vision and pattern recognition, 3779–3788, 2017. http://doi.org/10.1109/CVPR.2017.702.

[8] Levin, D.: The approximation power of moving least-squares. Mathematics of computation, 67(224), 1517–1531, 1998. http://doi.org/10.1090/S0025-5718-98-00974-0.

[9] Lim, I.; Dielen, A.; Campen, M.; Kobbelt, L.: A simple approach to intrinsic correspondence learning on unstructured 3d meshes. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops, 0–0, 2018. http://doi.org/10.1007/978-3-030-11015-4_26.

[10] Masci, J.; Boscaini, D.; Bronstein, M.; Vandergheynst, P.: Geodesic convolutional neural networks on riemannian manifolds. In Proceedings of the IEEE international conference on computer vision workshops, 37–45, 2015. http://doi.org/10.1109/ICCVW.2015.112.

[11] Metzer, G.; Hanocka, R.; Giryes, R.; Cohen-Or, D.: Self-sampling for neural point cloud consolidation. ACM Transactions on Graphics (TOG), 40(5), 1–14, 2021. http://doi.org/10.1145/3470645.

[12] Monti, F.; Boscaini, D.; Masci, J.; Rodola, E.; Svoboda, J.; Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model cnns. In Proceedings of the IEEE conference on computer vision and pattern recognition, 5115–5124, 2017. http://doi.org/10.1109/CVPR.2017.576.

[13] Pan, L.; Chen, X.; Cai, Z.; Zhang, J.; Zhao, H.; Yi, S.; Liu, Z.: Variational relational point completion network. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 8524–8533, 2021. http://doi.org/10.1109/CVPR46437.2021.00842.

[14] Pérez, E.; Salamanca, S.; Merchán, P.; Adán, A.: A comparison of hole-filling methods in 3d. International Journal of Applied Mathematics and Computer Science, 26(4), 885–903, 2016. http://doi.org/10.1515/amcs-2016-0063.

[15] Perez, L.J.F.; Calla, L.A.R.; Montenegro, A.A.: Dictionary learning-based inpainting on triangular meshes, 2018.

[16] Podolak, J.; Rusinkiewicz, S.: Atomic volumes for mesh completion. In Symposium on Geometry Processing, 33–41. Citeseer, 2005.

[17] Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, 652–660, 2017. http://doi.org/10.1109/CVPR.2017.16.

[18] Qi, C.R.; Su, H.; Nießner, M.; Dai, A.; Yan, M.; Guibas, L.J.: Volumetric and multi-view cnns for object classification on 3d data. In Proceedings of the IEEE conference on computer vision and pattern recognition, 5648–5656, 2016. http://doi.org/10.1109/CVPR.2016.609.

[19] Sarkar, K.; Varanasi, K.; Stricker, D.: Learning quadrangulated patches for 3d shape parameterization and completion. In 2017 International Conference on 3D Vision (3DV), 383–392. IEEE, 2017. http://doi.org/10.1109/3DV.2017.00051.

[20] Sarkar, K.; Varanasi, K.; Stricker, D.: 3d shape processing by convolutional denoising autoencoders on local patches. In 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), 1925–1934. IEEE, 2018. http://doi.org/10.1109/WACV.2018.00213.

[21] Su, H.; Maji, S.; Kalogerakis, E.; Learned-Miller, E.: Multi-view convolutional neural networks for 3d shape recognition. In Proceedings of the IEEE international conference on computer vision, 945–953, 2015. http://doi.org/10.1109/ICCV.2015.114.

[22] Tekumalla, L.S.; Cohen, E.: A hole-filling algorithm for triangular meshes. School of Computing, University of Utah, UUCS-04-019, UT, USA, 2, 2004.

[23] Wang, J.; Oliveira, M.M.: A hole-filling strategy for reconstruction of smooth surfaces in range images. In 16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2003), 11–18. IEEE, 2003.

[24] Wang, P.S.; Liu, Y.; Guo, Y.X.; Sun, C.Y.; Tong, X.: O-cnn: Octree-based convolutional neural networks

for 3d shape analysis. vol. 36, 1–11. ACM New York, NY, USA, 2017. http://doi.org/10.1145/3450626.3459787.

[25] Wang, X.; Ang, M.H.; Lee, G.H.: Point cloud completion by learning shape priors. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 10719–10726. IEEE, 2020. http://doi.org/10.1109/IROS45743.2020.9340862.

[26] Wang, X.; Xu, D.; Gu, F.: 3d model inpainting based on 3d deep convolutional generative adversarial network. IEEE Access, 8, 170355–170363, 2020. http://doi.org/10.1109/ACCESS.2020.3024288.

[27] Wang, Y.; Xie, Z.; Xu, K.; Dou, Y.; Lei, Y.: An efficient and effective convolutional auto-encoder extreme learning machine network for 3d feature learning. Neurocomputing, 174, 988–998, 2016. http://doi.org/10.1016/j.neucom.2015.10.035.

[28] Wei, M.; Wu, J.; Pang, M.: An integrated approach to filling holes in meshes. In 2010 International Conference on Artificial Intelligence and Computational Intelligence, vol. 3, 306–310. IEEE, 2010. http://doi.org/10.1109/AICI.2010.302.

[29] Wu, X.J.; Wang, M.Y.; Han, B.: An automatic hole-filling algorithm for polygon meshes. Computer-Aided Design and Applications, 5(6), 889–899, 2008. http://doi.org/10.3722/cadaps.2008.889-899.

[30] Xie, H.; Yao, H.; Zhou, S.; Mao, J.; Zhang, S.; Sun, W.: Grnet: Gridding residual network for dense point cloud completion. In European Conference on Computer Vision, 365–381. Springer, 2020. http://doi.org/10.1007/978-3-030-58545-7_21.

[31] Xie, Z.; Xu, K.; Shan, W.; Liu, L.; Xiong, Y.; Huang, H.: Projective feature learning for 3d shapes with multi-view depth images. In Computer Graphics Forum, vol. 34, 1–11. Wiley Online Library, 2015. http://doi.org/10.1111/cgf.12740.

[32] Yuan, W.; Khot, T.; Held, D.; Mertz, C.; Hebert, M.: Pcn: Point completion network. In 2018 International Conference on 3D Vision (3DV), 728–737. IEEE, 2018. http://doi.org/10.1109/3DV.2018.00088.

[33] Zhao, W.; Gao, S.; Lin, H.: A robust hole-filling algorithm for triangular mesh. The Visual Computer, 23(12), 987–997, 2007. http://doi.org/10.1007/s00371-007-0167-y.