Surface Representation for Point Clouds

Haoxi Ran¹*  Jun Liu²  Chengjie Wang²
¹Northeastern University  ²Tencent Youtu Lab
{ranhaoxi@gmail.com, juliusliu, jasoncjwang}@tencent.com

Abstract

Most prior work represents the shapes of point clouds by coordinates. However, it is insufficient to describe the local geometry directly. In this paper, we present RepSurf (representative surfaces), a novel representation of point clouds to explicitly depict the very local structure. We explore two variants of RepSurf, Triangular RepSurf and Umbrella RepSurf inspired by triangle meshes and umbrella curvature in computer graphics. We compute the representations of RepSurf by predefined geometric priors after surface reconstruction. RepSurf can be a plug-and-play module for most point cloud models thanks to its free collaboration with irregular points. Based on a simple baseline of PointNet++ (SSG version), Umbrella RepSurf surpasses the previous state-of-the-art by a large margin for classification, segmentation and detection on various benchmarks in terms of performance and efficiency. With an increase of around 0.008M number of parameters, 0.04G FLOPs, and 1.12ms inference time, our method achieves 94.7% (+0.5%) on ModelNet40, and 84.6% (+1.8%) on ScanObjectNN for classification, while 74.3% (+0.8%) mIoU on S3DIS 6-fold, and 70.0% (+1.6%) mIoU on ScanNet for segmentation. For detection, previous state-of-the-art detector with our RepSurf obtains 71.2% (+2.1%) mAP₂₅, 54.8% (+2.0%) mAP₅₀ on ScanNetV2, and 64.9% (+1.9%) mAP₂₅, 47.7% (+2.5%) mAP₅₀ on SUN RGB-D. Our lightweight Triangular RepSurf performs its excellence on these benchmarks as well. The code is publicly available at https://github.com/hancyran/RepSurf.

1. Introduction

Learning from raw point clouds has drawn considerable attention for its advantages in various applications, like autonomous driving, augmented reality, and robotics. However, it can be difficult for the irregularity of point clouds.

To handle irregular points, the pioneering work PointNet [41] adopts point-wise multi-layer perceptrons (MLP) to learn from points independently and utilizes a symmetric function to obtain the global information. PointNet++ [43] further introduces set abstraction (SA) to capture the local information of point clouds. However, both methods learn from standalone points and take no notice of local shape awareness [31].

Local shapes are vital for the learning of point clouds. To learn from the local structural information, some prior works learn from grids [25,53], relations [31,44], or graphs [58,65]. However, these methods learn from shapes indirectly by attaching more ingredients (like Euclidean distances, attention mechanism) or applying various transformations (like graph construction, voxelization). These may lead to complex preprocessing and significant computations. These sophisticated hand-crafted components learn from implicit local shape representations in general. We argue that it may lead to an omission of information when pre-defining the ingredients, or a loss of geometry during transformation.

Taylor Series [52] expresses a local curve by derivatives. We simplify it by considering the second derivative only.

Figure 1. An overview of point cloud classification with RepSurf. Given one point (blue) in the airplane point cloud, we indicate its global position by the coordinate $x_i$. Different from the prior works, we further explicitly describe its local geometry through Triangular RepSurf $t_i$ extracted from the reconstructed triangle or Umbrella RepSurf $u_i$ learned from the reconstructed umbrella surface. By combining positional and geometric information, point representation can be more expressive. After concatenating $x_i$ and $t_i/u_i$ as input, we predict the category of the point cloud via MLPs followed by a pooling operation.
Thus, we can roughly represent the local curve, or what we call the “surface” in 3D point clouds, by its corresponding tangent.

To this end, inspired by Taylor Series, we propose RepSurf (representative surfaces) to explicitly represent the local shape of point clouds (shown in Fig. 1). To complement Cartesian coordinates in a point set with geometric information, we define RepSurf with three properties: discreteness, explicit locality, and curvature sensitivity. These properties allow RepSurf to express local geometry in free collaboration with irregular points. For a simple version of RepSurf, we propose Triangular RepSurf inspired by triangle meshes in computer graphics. We reconstruct a triangle for each point by querying its two neighbors and compute the triangle feature (i.e., normal vector, surface position, normalized coordinate) as RepSurf. To enlarge the perceptive field of RepSurf, we further propose Umbrella RepSurf inspired by umbrella curvature [10]. Umbrella RepSurf can be an extension of Triangular RepSurf since it is computed from the triangles of an umbrella surface. Different from Triangular RepSurf, we reconstruct an umbrella surface after searching $K$ nearest neighbors and sorting the neighbors counterclockwise. For expressive representations, we feed the $K$ triangular features of an umbrella surface into a learnable transformation function followed by aggregation. Moreover, we present several delicate designs (i.e., polar auxiliary, channel de-differentiation) to further improve RepSurf.

Our key contributions are manifold:

- A high-efficiency plug-and-play module based on RepSurf for point cloud models.
- Our method achieves state-of-the-art on numerous point cloud benchmarks.

2. Related Work

2.1. Learning on Point Clouds

Multi-view methods [9, 14–16, 42, 62] or voxel-based methods [7, 21, 34] describe 3D objects with multiple views (i.e., converting 3D shape to 2D images [51] and lattice space [50]) or by voxelization (Oc-tree based networks O-CNN [56] and OctNet [45], efficient submanifold sparse convolution [13]). However, these transformation methods may lead to significant computations as well as a loss of shape information due to occlusion or lower resolution.

Point-based methods [11, 18, 19, 23, 24, 29, 32, 36] have recently attracted great attention to directly process point clouds. PointNet [41] learns from global information through multi-layer perceptrons and max-pooling operation. PointNet++ [43] introduces set abstraction to capture the features from the local point sets, and farthest point sampling (FPS) to uniformly downsample between two set abstractions. Recent works explore local aggregator via convolutions [17, 27, 28, 30, 37, 53, 57, 59, 64, 66, 71, 73], relations [44, 61, 67, 74], and graphs [58, 65, 75]. PointCNN [25] applies traditional convolution on point clouds after transforming neighboring points to the canonical order. RS-CNN [31] predefines geometric relations between points and their neighbors for local aggregation. DGCNN [58] computes the local graphs dynamically to extract geometric information. However, the methods are commonly based on some assumptions of implicit local geometry, which may result in missing geometric information in the input.

2.2. Detection on Point Clouds

Some early methods detect 3D objects by convolution after converting point clouds to 2D grids [5, 22, 26, 63, 68] or 3D voxels [49, 76]. Recent works focus on 3D detection of raw point clouds [4, 6, 33, 35, 38, 40, 46, 47, 72]. VoteNet [39] adopts PointNet++ as the backbone for feature extraction and designs a component to group points corresponding to the voted centroids. [33] removes the hand-crafted operation of grouping by introducing Transformers [55].

2.3. Graphics-related Surface Representation

In computer graphics, triangle meshes are commonly adopted to represent 3D models. To obtain meshes from point clouds, previous works propose various methods for surface reconstruction. Ball-Pivoting Algorithm [3] forms a triangle if a specific-radius ball touches three points without containing other points. [20] defines the spatial Poisson formulation for surface reconstruction.

Curvature can further present the local geometry on 3D point clouds. [70] estimates the local curvature of the point cloud surface by Least Square Fitting. [10] constructs an umbrella surface based on the homogeneous neighbors and calculates the umbrella curvature through the neighbors’ normal vectors and unit direction vectors.

3. Surface Representation

In this section, we first reveal the background for the design of our Representative Surfaces (RepSurf) in Sec. 3.1. Secondly, we introduce several properties of RepSurf as inspiration in Sec. 3.2. Next, we propose two variants of RepSurf, Triangular and Umbrella RepSurf in Sec. 3.3 and Sec. 3.4, respectively. Finally, we implement RepSurf on PointNet++ (SSG version) and provide several exquisite designs to further improve the performance of RepSurf.
3.1. Background

Local shapes are essential to represent point clouds. Prior works learn from shapes indirectly by utilizing extra ingredients or through different transformations. These operations may give some hints to express the local sets of point clouds, but cannot reflect the local shapes explicitly. We argue that the additional information leads to significant computations but contributes little to point cloud representations. Some may even cause the loss of geometric information. Therefore, we have to rethink on how to represent the local geometry.

We can describe a very local part centered on point \((t, f(t))\) of a 2D curve \(f(\cdot)\) by Taylor series [52]:

\[
f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(t)}{n!} (x - t)^n, \quad |x - t| < \epsilon
\]

To simplify the calculation, we approximate this equation by:

\[
f(x) \simeq f(t) + f'(t)(x - t),
\]

where \((t, f(t))\) is the global position on curve \(f(\cdot)\), and the first derivative \(f'(t)\) can intuitively indicate the local orientation near point \((t, f(t))\). To further express the local curve (Fig. 2 left), we represent the local orientation by its corresponding tangent:

\[
a_i (x - x_i) + b_i (y - y_i) = 0 \Rightarrow
\]

\[
a_i x + b_i y - (a_i x_i + b_i y_i) = 0,
\]

where \(x_i = t, y_i = f(t)\), and \(\frac{a_i}{b_i} = -f'(a)\). \((a_i, b_i)\) is the normal vector of the tangent, where \(a_i^2 + b_i^2 = 1\). To conclude, a rough description of the local curve can be defined as:

\[
c_i = (x_i, y_i, a_i, b_i, a_i x_i + b_i y_i).
\]

3.2. Properties of RepSurf

PointNet [41] is inspired by three main properties of point sets in \(\mathbb{R}^{N \times 3}\) from an Euclidean space: 1) unordered, 2) interaction among points and 3) invariance under transformations. It can handle the unordered point sets and alleviate the problem from rigid transformations. However, the ability to interact among points is still underexplored.

In 3D computer graphics, triangle meshes are a common representation of 3D models. Regularly, a triangle mesh consists of a set of triangles connected by their common edges or corners. Thus, triangles can flexibly present continuous and sophisticated 3D shapes for this characteristic. However, triangle meshes may not match the data structure of point clouds due to irregularity. A direct conversion from point cloud to triangle mesh may lead to significant computation as well as loss of point cloud characteristics (like flexibility from unorderness, scalability from the nature of sets). Therefore, we design our RepSurf inspired by the following properties:

- **Discreteness.** Ideally, RepSurf should be a set to collaborate with the related point set. It means that each of \(N\) points has a corresponding RepSurf feature.

- **Explicit Locality.** Unlike prior works describing local structure by learning (implicit locality), RepSurf shows the explicit locality of a part of point clouds numerically.

- **Curvature Sensitivity.** Coordinates can hardly depict the local shapes of 3D point clouds. RepSurf should be eligible to intuitively highlight edges and local shapes. An illustration is shown in Fig. 3.
3. Umbrella RepSurf

3.3. Triangular RepSurf

Denote a point set as \( X = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^{N \times 3} \). Analogous to a 2D curve in Sec 3.1, we define a 3D tangent surface (Fig. 2 right) by point-normal equation. Given a normal vector \( v_i = (a_i, b_i, c_i) \) and a point \( x_i = (x_i, y_i, z_i) \), the surface can be defined as:

\[
\begin{align*}
    a_i(x - x_i) + b_i(y - y_i) + c_i(z - z_i) &= 0 \Rightarrow \quad a_i x + b_i y + c_i z - (a_i x_i + b_i y_i + c_i z_i) = 0. \\
\end{align*}
\]

We define the surface position as \( p_i = a_i x_i + b_i y_i + c_i z_i \), with the range of \([-\sqrt{3} r, \sqrt{3} r]\). \( r \) means the edge length of a cube exactly covering the point set. For example, we utilize the normalized point clouds within the range of \([-1, 1]\) as input, so \( r = 1 \) here. Note that \( p_i \) can also express the directed distance between the origin and the surface. Then, we compute \( v_i \) by cross product. However, the computed \( v_i \) is unoriented — \( v_i \) can be pointing either inside or outside of the surface. To handle this problem, prior works [2] adopt some time-costing methods. Considering efficiency, we simplify this case by keeping \( a_i \) positive and augmenting the normals by instance-level random inverse with a probability of 50\%. Thus, we define Triangular RepSurf as:

\[
    t_i = (a_i, b_i, c_i, p_i). \tag{6}
\]

We define a set of Triangular RepSurf as \( T = \{t_1, \ldots, t_n\} \subseteq \mathbb{R}^{N \times 4} \). To feed point clouds into models, we replace \( X \) with our re-computed centroids \( X' \) of the triangles. Then the input can be the concatenation of \( X' \) and \( T \). A simple illustration and the implementation of Triangular RepSurf is presented in Fig. 1 and Algorithm 1, respectively.

3.4. Umbrella RepSurf

Triangular RepSurf is a lightweight method to represent the local geometry of a point cloud. However, due to limited perceptive field, it may also lead to unstable local representations. To handle this drawback, we expand the perceptive field by proposing Umbrella RepSurf inspired by umbrella curvature [10].

---

**Algorithm 1 Pytorch-Style Pseudocode of Triangular RepSurf**

```python
# B: batch size, N: number of points
# points: coordinates of a point set
pairs = kNN(points, k=2)-points # [B,N,2,3]
centroids = mean(pairs, dim=2) # [B,N,3]
normals = cross_product(pairs) # [B,N,3]
features = pooling(features, dim=2) # [B,N,3+C]
features = MLPs(features, out_channel=C) # [B,N,K,3+C]
features = concat([centroids, normals, positions], dim=2) # [B,N,7]
return out
```

**Algorithm 2 Pytorch-Style Pseudocode of Umbrella RepSurf**

```python
# B: batch size, N: number of points
# K: number of neighbors, C: output channels
# points: coordinates of a point set
neighbors = kNN(points, k=K)-points # [B,N,K,3]
edges = sort_by_clock(neighbors) # [B,N,K,2]
edges = unsqueeze(edges, dim=2) # [B,N,K,1,2]
pairs = concat([edges, edges.roll(-1, 2)], dim=2) # [B,N,K,2,3]
positions = sum(normals*centroids, dim=2)/sqrt(3)
normals = random_inverse(normals) # [B,N,K,3]
normals = normals*pos_mask # [B,N,K,3]
normals = normals/norm(normals, dim=-1) # [B,N,K,3]
normals = cross_product(pairs) # [B,N,K,3]
edges = unsqueeze(neighbors, dim=2) # [B,N,K,1,3]
pairs = concat([edges, edges.roll(-1, 2)], dim=2) # [B,N,K,2,3]
centroids = mean(pairs, dim=3) # [B,N,K,3]
normals = cross_product(pairs) # [B,N,K,3]
features = concat((centroids, normals, positions), dim=2) # [B,N,3+C]
features = MLPs(features, out_channel=C) # [B,N,3+C]
features = pooling(features, dim=2) # [B,N,7]
return out
```

Denote the number of neighbors as \( K \), the centroids and triangular features of the neighbor triangles as \( X' = \{x'_{i}, \ldots, x'_{iK}\} \subseteq \mathbb{R}^{K \times 3} \) and \( T_j = \{t_{j1}, \ldots, t_{jK}\} \subseteq \mathbb{R}^{K \times 4} \). In [10], the unsigned scalar of umbrella curvature is defined as:

\[
    u_i = \sum_j^K n_{ij} = \sum_j^K \frac{|x'_{ij} \cdot n_i|}{|x'_{ij}|}, \tag{7}
\]

where \( n_i \) is the given normal vector of the \( i \)-th point. However, \( n_i \) is commonly unknown in the point set \( X \). This makes umbrella curvature unpractical in the real scenes. Furthermore, we argue that a scalar curvature cannot fully express the local geometry. In this case, we propose Umbrella RepSurf to express the local geometry without any given normals. Moreover, different from umbrella curvature which is defined based on homogeneous neighbors, our Umbrella RepSurf can handle heterogeneous neighbors for its position sensitivity. An illustration is shown in Fig. 4. The Umbrella RepSurf \( u_i \) of point \( x_i \) is defined as:

\[
    u_i = A (\{T_i (\{x'_{ij}, t_{ij}\}) \}, \forall j \in \{1, \ldots, K\}), \tag{8}
\]

where \( A \) is an aggregation function (i.e., summation), \( T \) is a transformation function, and \( x'_{ij} \) is the normalized coordinate according to its centroid \( x_i \). To calculate \( t_{ij} \), we construct adjacent triangles counterclockwise from 0° (x-axis) to 359° on the xy-plane. Thus, the number of triangles in a umbrella surface is exactly \( K \). Note that, to keep local consistency of the normals’ orientation, we compute these normals by counterclockwise cross product. (An example when reconstructing a umbrella surface unorderedly in Fig. 4.) To simplify the definition of the global normals’ orientation, different from Triangular RepSurf, we keep \( a_{ij} \) of \( t_{ij} \) positive and the orientation of other normals changes accordingly. Therefore, though the orientation is consistent
Figure 4. Examples of reconstructed umbrella surfaces. We present each surface with a regular view (above) and a top view (below). From left to right, we show two surfaces reconstructed from homogeneous neighbors, one from heterogeneous neighbors, and one reconstructed without sorting.

locally, the normals can be unoriented from a global perspective. Similar to Triangular RepSurf, we augment the normals of an umbrella surface $\mathbf{n}_i$ by random inverse. Instead of a predefined transformation function, we adopt a learnable function (a combination of linear functions and non-linearity) for $T$. The implementation of Umbrella RepSurf is shown in Algorithm 2.

3.5. Implementation

We implement our RepSurf on the single-scale grouping (SSG) version of PointNet++ [43] in a simple manner of concatenation. For each set abstraction, we input RepSurf along with point features. An illustration of the input flow is shown in Fig. 5. Furthermore, we propose two designs to further improve our RepSurf.

**Polar auxiliary.** For simplicity, previous point-based models widely adopt Cartesian coordinates as input. However, they cannot fully express the relationships between a centroid and its neighbors. Unlike Cartesian coordinate system, the polar coordinate systems present a point coordinate by distance and angles according to the origin. The polar systems (i.e., Spherical system, Cylindrical system) can be a supplement for its distance and direction sensitivity. In this paper, we explore a practical application of the polar systems for point-based models. We take Spherical system as an example. After querying the neighbors of a point $x_i$, we re-define the position of the $j$-th neighbor by including its spherical position:

$$x'_ij = (x'ij, y'ij, z'ij, \rho_{ij}, \theta_{ij}, \phi_{ij}), \quad (9)$$

where $x'ij, y'ij, z'ij$ are the values of three dimensions of the normalized Cartesian coordinate. $\rho_{ij} = \sqrt{x'ij^2 + y'ij^2 + z'ij^2}$, $\theta_{ij} = \arccos \frac{z'ij}{\rho_{ij}}$, $\phi_{ij} = \arctan2(y'ij, x'ij)$. For more details of the implementation on polar auxiliary, please refer to the supplementary material.

**Channel de-differentiation.** Inspired by [69], we observe that different types of inputs (i.e., coordinate, normal vectors, point features) have significant differences in data distribution. In order to process different inputs equally and to train our models stably, we explore solutions for de-differentiation along the channel dimension. In this paper, we apply Post-CD (performing batch normalization after linear function) to our method. For more details of the implementation on channel de-differentiation, please refer to the supplementary material.

4. Experiments

We evaluate both of our Triangular RepSurf and Umbrella RepSurf on three main tasks: classification, segmentation, and detection. Furthermore, we conduct ablation studies to assess the effectiveness of our designed modules. Please refer to the supplementary material for more experimental details.

4.1. Classification

3D object classification is a basic task to prove the effectiveness of methods. We perform experiments on ModelNet40 [60], a human-made object dataset, and ScanObjectNN [54], a dataset retrieved from the real scenes.

**Human-made Object Classification.** ModelNet40 [60] contains 9843 training models and 2468 test models, divided into 40 categories. In Tab. 1, we compare our Triangular RepSurf (RepSurf-T) and Umbrella RepSurf (RepSurf-U) with prior methods. Equipped with RepSurf-T and RepSurf-U, the performance of PointNet++ (SSG version) is considerably boosted by 3.7% and 4.1%. For a fair comparison with other methods [31, 61, 64], we apply the strategy of multi-scale inference from [31] for further improvement. Though the results on ModelNet40 tend to be saturated, our RepSurf-U achieves 94.7%, surpassing CurveNet [61] by a large margin of 0.5%. In addition, RepSurf-U is 5.4× and 4.0× faster than CurveNet in terms of training and inference speed, respectively.

**Real-world Object Classification.** For the saturation of ModelNet40, we further verify our RepSurf on the hardest variant (PB_T50_RS variant) of ScanObjectNN [60], a
### Table 1. Performance of classification on ModelNet40 and ScanObjectNN. We evaluate different methods in terms of overall accuracy (OA, %), mean per-class accuracy (mAcc, %), number of parameters (#Params), FLOPs, training speed (duration per input sample), and inference speed (duration per input sample). We consider OA the principle evaluation metric. **Bold** means the result outperforms prior state-of-the-art method on corresponding dataset. **Green** means an improvement from our RepSurf compared with the original model. **Green**: with normal vector.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>ModelNet40 OA</th>
<th>ModelNet40 mAcc</th>
<th>ScanObjectNN OA</th>
<th>ScanObjectNN mAcc</th>
<th>#Params</th>
<th>FLOPs</th>
<th>Train Speed</th>
<th>Infer Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet [41]</td>
<td>1k pnts</td>
<td>89.2</td>
<td>86.0</td>
<td>68.2</td>
<td>63.4</td>
<td>3.47M</td>
<td>0.45G</td>
<td>1.76ms</td>
<td>0.81ms</td>
</tr>
<tr>
<td>DGCNN [58]</td>
<td>1k pnts</td>
<td>92.9</td>
<td>90.2</td>
<td>78.1</td>
<td>73.6</td>
<td>1.82M</td>
<td>2.43G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RS-CNN† [31]</td>
<td>1k pnts</td>
<td>93.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.38M</td>
<td>1.16G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>KPConv [53]</td>
<td>~7k pnts</td>
<td>92.9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>14.3M</td>
<td>-</td>
<td>218.7ms</td>
<td>543.7ms</td>
</tr>
<tr>
<td>PointASNL [67]</td>
<td>1k pnts*</td>
<td>93.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>10.1M</td>
<td>1.80G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Grid-GCN [65]</td>
<td>1k pnts*</td>
<td>93.1</td>
<td>91.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PointTrans. [74]</td>
<td>1k pnts*</td>
<td>93.7</td>
<td>90.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MVTN [15]</td>
<td>multi-view</td>
<td>93.8</td>
<td>92.0</td>
<td>-</td>
<td>-</td>
<td>4.24M</td>
<td>1.78G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PConv† [61]</td>
<td>1k pnts</td>
<td>93.9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.44M</td>
<td>1.68G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RPNet [44]</td>
<td>1k pnts*</td>
<td>94.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.70M</td>
<td>3.90G</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CurveNet† [61]</td>
<td>1k pnts</td>
<td>94.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.14M</td>
<td>0.66G</td>
<td>22.04ms</td>
<td>12.34ms</td>
</tr>
<tr>
<td>PointNet++† [43]</td>
<td>1k pnts</td>
<td>90.7</td>
<td>88.4</td>
<td>77.9</td>
<td>75.4</td>
<td>1.475M</td>
<td>0.77G</td>
<td>2.75ms</td>
<td>1.98ms</td>
</tr>
<tr>
<td>RepSurf-T (ours)</td>
<td>1k pnts</td>
<td>94.0</td>
<td>13.3</td>
<td>91.1</td>
<td>12.7</td>
<td>84.1</td>
<td>16.2</td>
<td>81.2</td>
<td>15.8</td>
</tr>
<tr>
<td>RepSurf-T† (ours)</td>
<td>1k pnts</td>
<td>94.2</td>
<td>13.5</td>
<td>91.3</td>
<td>12.9</td>
<td>84.3</td>
<td>16.4</td>
<td>81.6</td>
<td>16.2</td>
</tr>
<tr>
<td>RepSurf-U (ours)</td>
<td>1k pnts</td>
<td>94.4</td>
<td>13.7</td>
<td>91.4</td>
<td>13.0</td>
<td>84.3</td>
<td>16.4</td>
<td>81.3</td>
<td>15.9</td>
</tr>
<tr>
<td>RepSurf-U† (ours)</td>
<td>1k pnts</td>
<td>94.7</td>
<td>14.0</td>
<td>91.7</td>
<td>13.3</td>
<td>84.6</td>
<td>16.7</td>
<td>81.9</td>
<td>16.5</td>
</tr>
<tr>
<td>RepSurf-U‡ (ours)</td>
<td>1k pnts</td>
<td>94.7</td>
<td>14.0</td>
<td>91.7</td>
<td>13.3</td>
<td>84.6</td>
<td>16.7</td>
<td>81.9</td>
<td>16.5</td>
</tr>
<tr>
<td>RepSurf-U‡ (ours)</td>
<td>1k pnts</td>
<td>94.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.14M</td>
<td>0.66G</td>
<td>22.04ms</td>
<td>12.34ms</td>
</tr>
</tbody>
</table>

†: single-scale grouping (SSG) version, †: multi-scale inference from [31], †: w/ normal vector, o: PointNet++ (SSG) with double channels and deeper networks.

### Table 2. Performance of semantic segmentation on S3DIS (evaluation by 6-fold or on Area 5) and ScanNet V2. We evaluate different methods in terms of mIoU (mIoU, %), mean per-class accuracy (mAcc, %), number of parameters (#Params), FLOPs, and training speed (duration per input sample). We consider OA the principle evaluation metric. **Bold** means the result outperforms prior state-of-the-art method on corresponding dataset. **Green** means an improvement from our RepSurf compared with the original model. **Green**: with normal vector.

<table>
<thead>
<tr>
<th>Method</th>
<th>S3DIS 6-fold</th>
<th>S3DIS Area-5</th>
<th>ScanNet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mIoU</td>
<td>mAcc</td>
<td>OA</td>
</tr>
<tr>
<td>PointNet [41]</td>
<td>47.6</td>
<td>66.2</td>
<td>78.5</td>
</tr>
<tr>
<td>PointWeb [73]</td>
<td>66.7</td>
<td>76.2</td>
<td>87.3</td>
</tr>
<tr>
<td>KPConv [53]</td>
<td>70.6</td>
<td>79.1</td>
<td>-</td>
</tr>
<tr>
<td>PointASNL [67]</td>
<td>68.7</td>
<td>79.0</td>
<td>88.8</td>
</tr>
<tr>
<td>PConv† [61]</td>
<td>69.3</td>
<td>78.6</td>
<td>-</td>
</tr>
<tr>
<td>RPNet [44]</td>
<td>70.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PointTrans. [74]</td>
<td>73.5</td>
<td>81.9</td>
<td>90.2</td>
</tr>
<tr>
<td>PointNet++† [43]</td>
<td>59.9</td>
<td>66.1</td>
<td>87.5</td>
</tr>
<tr>
<td>RepSurf-U (ours)</td>
<td>74.3</td>
<td>82.6</td>
<td>90.8</td>
</tr>
</tbody>
</table>

†: single-scale grouping (SSG) version, †: w/ normal vector.

Semantic Segmentation on S3DIS. S3DIS [1] contains 271 scenes from 6 indoor areas. Each point is categorized into 15 classes. In Tab. 1, our RepSurf-T and RepSurf-U achieve 84.3% and 84.6%, outperforming prior state-of-the-art MVTN [15] by 1.5% and 1.8%, with around 1.8× fewer parameters and 1.2× fewer FLOPs. **Bold** means the result outperforms prior state-of-the-art method on corresponding dataset. **Green** means an improvement from our RepSurf compared with the original model.

4.2. Segmentation

Scene segmentation can be more challenging due to outliers and noise. We evaluate our RepSurf on two large-scale scene datasets, S3DIS [1] and ScanNet V2 [8].
Table 3. Performance of object detection on ScanNet V2 and SUN RGB-D. We evaluate different methods in terms of mAP@0.25, mAP@0.5, number of parameters (#Params), and inference speed (duration per input sample). Bold means the result outperforms prior state-of-the-art method on corresponding dataset. Green means an improvement from our RepSurf compared with the original model. We test the speed of all methods with one NVIDIA Titan-XP GPU and four cores of Intel Xeon @2.50GHz CPU.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>ScanNet V2</th>
<th>SUN RGB-D</th>
<th>#Params</th>
<th>Infer Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mAP@0.25</td>
<td>mAP@0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VoteNet [39]</td>
<td>PointNet++</td>
<td>62.9</td>
<td>39.9</td>
<td>59.1</td>
<td>35.8</td>
</tr>
<tr>
<td>ImVoteNet [38]</td>
<td>PointNet++</td>
<td>-</td>
<td>-</td>
<td>63.4*</td>
<td>-</td>
</tr>
<tr>
<td>H3DNet [72]</td>
<td>PointNet++</td>
<td>64.4</td>
<td>43.4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>H3DNet [72]</td>
<td>4×PointNet++</td>
<td>67.2</td>
<td>48.1</td>
<td>60.1</td>
<td>39.0</td>
</tr>
<tr>
<td>3DETR [35]</td>
<td>Transformer</td>
<td>65.0</td>
<td>47.0</td>
<td>59.1</td>
<td>32.7</td>
</tr>
<tr>
<td>BRNet [6]</td>
<td>PointNet++</td>
<td>66.1</td>
<td>50.9</td>
<td>61.1</td>
<td>43.7</td>
</tr>
<tr>
<td>GroupFree6,256</td>
<td>PointNet++</td>
<td>67.3</td>
<td>48.9</td>
<td>63.0</td>
<td>45.2</td>
</tr>
<tr>
<td>GroupFree6,256</td>
<td>RepSurf-T</td>
<td>68.4↑↑1.1</td>
<td>50.3↑↑0.4</td>
<td>63.9↑↑0.9</td>
<td>45.6↑↑0.4</td>
</tr>
<tr>
<td>GroupFree6,256</td>
<td>RepSurf-U</td>
<td>68.8↑↑1.5</td>
<td>50.5↑↑0.6</td>
<td>64.3↑↑1.3</td>
<td>45.9↑↑0.7</td>
</tr>
<tr>
<td>GroupFree12,512</td>
<td>PointNet++2</td>
<td>69.1</td>
<td>52.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GroupFree12,512</td>
<td>RepSurf-T2</td>
<td>70.4↑↑1.3</td>
<td>54.6↑↑1.8</td>
<td>64.2</td>
<td>47.1</td>
</tr>
<tr>
<td>GroupFree12,512</td>
<td>RepSurf-U2</td>
<td>71.2↑↑2.1</td>
<td>54.8↑↑2.0</td>
<td>64.9</td>
<td>47.7</td>
</tr>
</tbody>
</table>

*: w/ RGB as input, Model²: Model with doubled channels for each MLP, 4×PointNet++: four individual PointNet++ (SSG) in [72], GroupFreeⁿ,ⁿ: GroupFree model [33] with a n-layer decoder and b object candidates.

Semantic Segmentation on ScanNet. ScanNet V2 [8] consists of 1513 indoor training point clouds and 100 test point clouds. It marks each point with 21 categories. In Tab. 2, the performance of RepSurf-U exceeds prior state-of-the-art KPConv [53] by 1.6%. Moreover, our method has 14.3× fewer parameters compared with KPConv.

4.3. Detection

3D detection can further prove the superiority of our method at the application level. We conduct experiments on two widely adopted 3D object detection datasets: ScanNet V2 [8] and SUN RGB-D [48]. We adopt a powerful method [33] for pipeline and replace the backbone with our RepSurf to perform all experiments on this task. Our experiments are mainly based on the codebase¹¹ of [33] as well.

Detection on ScanNet. ScanNet V2 [8] can be adopted for 3D detection as well, consisting of 1513 indoor scenes and 18 object classes. We follow the standard evaluation protocol in [39] by utilizing mean Average Precision under the thresholds of 0.25 (mAP@0.25) and 0.5 (mAP@0.5), without considering the orientation of bounding boxes. As shown in Tab. 3, with almost no increase in computational cost (~0.01M parameters and ~1ms inference speed), our RepSurf-U boosts the performance of previous state-of-the-art [33] by 2.1% mAP@0.25 and 2.0% mAP@0.25.

Detection on SUN RGB-D. SUN RGB-D [48] is a single-view RGB-D dataset for 3D scene analysis, including around 5K indoor RGB and depth images. Following [39], we adopt mean Average Precision on 10 most common categories for evaluation. In Tab. 3, RepSurf-


<table>
<thead>
<tr>
<th>type</th>
<th>X-computed</th>
<th>w/ pi</th>
<th>w/ inverse</th>
<th>acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>given</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>94.08</td>
</tr>
<tr>
<td>given</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>93.39</td>
</tr>
<tr>
<td>given</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>93.95</td>
</tr>
<tr>
<td>triangular</td>
<td>pre</td>
<td>✓</td>
<td>✓</td>
<td>93.57</td>
</tr>
<tr>
<td>triangular</td>
<td>post</td>
<td>✓</td>
<td>✓</td>
<td>93.62</td>
</tr>
<tr>
<td>triangular</td>
<td>post</td>
<td>✓</td>
<td>✓</td>
<td><strong>94.02</strong></td>
</tr>
<tr>
<td>umbrella</td>
<td>pre</td>
<td>✓</td>
<td>✓</td>
<td>93.06</td>
</tr>
<tr>
<td>umbrella</td>
<td>post</td>
<td>✓</td>
<td>✓</td>
<td>93.90</td>
</tr>
<tr>
<td>umbrella</td>
<td>post</td>
<td>✓</td>
<td>✓</td>
<td><strong>94.46</strong></td>
</tr>
</tbody>
</table>

¹¹https://github.com/zeliu98/Group-Free-3D
responding tangent. However, empirical results show that post-computed works better than pre-computed. We additionally test on the designs of surface position and random inverse, both of which slightly improve RepSurf.

**Design of RepSurf block.** Shown in Tab. 5, we explore the design of Umbrella RepSurf in terms of input, transformation function $T$, and aggregation function $A$. Empirically, a combination of normal vector, surface position, normalized coordinate and the corresponding polar coordinates outperforms other combinations. Furthermore, prohibition of batch norm, usage of bias for the first layer, sum-pooling, and three-layer MLP perform better than other options.

**Group size.** We explore the group size of Umbrella RepSurf in terms of both accuracy and speed (ms per sample):

<table>
<thead>
<tr>
<th>$k$</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PN2</strong></td>
<td>93.53</td>
<td>93.63</td>
<td>94.36</td>
<td><strong>94.46</strong></td>
<td>94.32</td>
<td>94.20</td>
<td>94.32</td>
</tr>
<tr>
<td><strong>time</strong></td>
<td>0.50</td>
<td>0.46</td>
<td>0.49</td>
<td>0.48</td>
<td>0.58</td>
<td>0.51</td>
<td>0.78</td>
</tr>
</tbody>
</table>

We test the speed of Umbrella RepSurf block only. When $k=2$, Umbrella RepSurf will degenerate to a learnable version of Triangular RepSurf. There is almost no difference in speed when $k$ is in the range of $[2, 12]$. For a trade-off between performance and speed, we consider $k=8$ an ideal choice. Furthermore, when we study on larger group sizes (i.e., 24), a vanishing gradient problem exists. We argue that larger umbrella surfaces may become more indistinguishable and lead to the problem, but this is still an open issue.

**Polar auxiliary.** We study on the design of our polar auxiliary in different versions:

<table>
<thead>
<tr>
<th>$k$</th>
<th>w/o aux.</th>
<th>w/ $\rho$</th>
<th>w/ cylinder</th>
<th>w/ sphere</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>acc.</strong></td>
<td>93.97</td>
<td>94.12</td>
<td>90.15</td>
<td>93.89</td>
</tr>
</tbody>
</table>

Here $\rho$, a part of spherical polar auxiliary, means the distance between a centroid and its neighbors. We discuss that

**Channel de-differentiation.** We test the design of channel de-differentiation (CD) on three versions of PointNet++, including the original (vanilla), Triangular RepSurf (triangular), and Umbrella RepSurf (umbrella):

<table>
<thead>
<tr>
<th>$k$</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PN2</strong></td>
<td>vanilla</td>
<td>93.15</td>
<td>92.70</td>
<td>10.36</td>
<td>94.08</td>
<td>10.93</td>
<td></td>
</tr>
<tr>
<td><strong>PN2</strong></td>
<td>triangular</td>
<td>93.22</td>
<td>92.49</td>
<td>10.73</td>
<td>94.02</td>
<td>10.80</td>
<td></td>
</tr>
<tr>
<td><strong>PN2</strong></td>
<td>umbrella</td>
<td>93.50</td>
<td>92.63</td>
<td>10.87</td>
<td><strong>94.46</strong></td>
<td>10.96</td>
<td></td>
</tr>
</tbody>
</table>

Here Pre-CD means that batch normalization performs before linear function, and Post-CD is the opposite. We argue Post-CD performs better than Pre-CD, since Pre-CD may blur the original semantics of external input (i.e., coordinates, RepSurf features).

5. Discussion

**Limitation.** Though simple and effective, RepSurf may suffer from noises while surface reconstruction due to the noise-sensitive algorithm kNN. Furthermore, we argue that Umbrella RepSurf may be vulnerable to extremely messy points. Thus, when we query more neighbors of a point, in general the distribution of its neighbors would become messy and results in a distorted surface. An example of bad case is shown in Fig. 6.

**Conclusion.** We present two variants of RepSurf, Triangular and Umbrella RepSurf, to explore the surface representation on point clouds. We evaluate our simple baseline on various tasks, including shape classification, scene segmentation and detection. The evaluation results show its astonishing efficiency and performance, superior to the previous state-of-the-art on different benchmarks.

We hope our work can inspire the community and evoe the rethinking on the explicit representation of point clouds. We believe that RepSurf deserves further exploration for different fields (i.e., autonomous driving) or on larger-scale point clouds, since RepSurf is eligible to handle numerous background points in the real scenes. RepSurf may also be helpful for point cloud sampling by its ability on geometry sensitivity. It would be worthy of solving the above limitations of RepSurf as well.
References


[52] Brook Taylor. Methodus incrementorum directa et inversa. Imnys, 1717. 1, 3


18952