MeshWalker: Deep Mesh Understanding by Random Walks

ALON LAHAV, Technion – Israel Institute of Technology
AYELLET TAL, Technion – Israel Institute of Technology

Fig. 1. Classification by MeshWalker. This figure shows classification results as the walk (in green) proceeds along the surface of a camel (4K faces) from SHREC11 [Lian et al. 2011]. The initial point was randomly chosen on the neck. After V/50 steps (left), V being the number of vertices, the system is uncertain regarding the class, and the highest probability predictions are for the flamingo class and for the hand class (out of 30 classes). After continuing the random walk along the body and the front leg for V/7 steps, the probability of being a horse is higher than before, but the camel already has quite a high probability. Finally, after V/2.5 steps (right) and walking also along the hump, the system correctly classifies the model as a camel.

1 INTRODUCTION

The most commonly used representation of surfaces in computer graphics is a polygonal mesh, due to its numerous benefits, including efficiency and high-quality. Nevertheless, in the era of deep learning, this representation is often bypassed because of its irregularity, which does not suit Convolutional Neural Networks (CNNs). Instead, 3D data is often represented as volumetric grids [Ben-Shabat et al. 2018; Maturana and Scherer 2015; Roynard et al. 2018; Sedaghat et al. 2016b] or multiple 2D projections [Boulch et al. 2017; Feng et al. 2018a; Kanezaki et al. 2018; Su et al. 2015; Yavartanoo et al. 2018]. In some recent works point clouds are utilized and new ways to convolve or pool are proposed [Atzmon et al. 2018; Hua et al. 2018; Li et al. 2018; Thomas et al. 2019; Xu et al. 2018].

Despite the benefits of these representations, they miss the notions of neighborhoods and connectivity and might not be as good for capturing local surface properties. Recently, several works have proposed to maintain the potential of the mesh representation, while still utilizing neural networks. FeaStNet [Verma et al. 2018] proposes a graph neural network in which the neighborhood of each vertex for the convolution operation is calculated dynamically based on its features. MeshCNN [Hanocka et al. 2019] defines pooling and convolution layers over the mesh edges. MeshNet [Feng et al. 2019]...
treats the faces of a mesh as the basic unit and extracts their spatial and structural features individually to offer the final semantic representation. LRF-Conv [Yang et al. 2020] learns descriptors directly from the raw mesh by defining new continuous convolution kernels that provide robustness to sampling. All these methods redefine the convolution operation, and by doing so, are able to fit the unordered structure of a mesh to a CNN framework.

We propose a novel and fundamentally different approach, named MeshWalker. As in previous approaches that learn directly from the mesh data, the basic question is how to impose regularity on the unordered data. Our key idea is to represent the mesh by random walks on its surface. These walks explore the local geometry of the surface, as well as its global one. Every walk is fed into a Recurrent Neural Network (RNN), that ‘remembers’ the walk’s history.

In addition to simplicity, our approach has three important benefits. First, we will show that even a small dataset suffices for training. Intuitively, we can generate multiple random walks for a single model; these walks provide multiple explorations of the model. This may be considered as equivalent to using different projections of 3D objects in the case of image datasets. Second, as opposed to CNNs, RNNs are inherently robust to sequence length. This is vital in the case of meshes, as datasets include objects of various granularities. Third, the meshes need not be watertight or have a single connected component; our approach can handle any triangular mesh.

Our approach is general and can be utilized to address a variety of shape analysis tasks. We demonstrate its benefit in two basic applications: mesh classification and mesh semantic segmentation. Our results are superior to those of state-of-the-art approaches on common datasets and on highly non-uniform meshes. Furthermore, when the training set is limited in size, the accuracy improvement over the state-of-the-art methods is highly evident.

Hence, this paper makes three contributions:

1. We propose a novel representation of meshes for neural networks: random walks on surfaces.
2. We present an end-to-end learning framework that realizes this representation within RNNs. We show that this framework works well even when the dataset is very small. This is important in the case of 3D, where large datasets are seldom available and are difficult to generate.
3. We demonstrate the benefits of our method in two key applications: 3D shape classification and semantic segmentation.

2 RELATED WORK

Our work is at the crossroads of three fields, as discussed below.

2.1 Representing 3D objects for Deep Neural Networks

A variety of representations of 3D shapes have been proposed in the context of deep learning. The main challenge is how to re-organize the shape description such that it could be processed within deep learning frameworks. Hereafter we briefly review the main representations; see [Gezawa et al. 2020] for a recent excellent survey.

Multi-view 2D projections. This representation is essentially a set of 2D images, each of which is a rendering of the object from a different viewpoint [Bai et al. 2016; Feng et al. 2018b; Gomez-Donoso et al. 2017; Han et al. 2019; He et al. 2018; Johns et al. 2016; Kalogerakis et al. 2017; Kanezaki et al. 2018; Qi et al. 2016; Sarkar et al. 2018; Su et al. 2015; Wang et al. 2019c; Zanuttigh and Minto 2017]. The major benefit of this representation is that it can naturally utilize any image-based CNN. In addition, high-resolution inputs can be easily handled. However, it is not easy to determine the optimal number of views; if that number is large, the computation might be costly. Furthermore, self-occlusions might be a drawback.

Volumetric grids. These grids are analogous to the 2D grids of images. Therefore, the main benefit of this representation is that operations that are applied on 2D grids can be extended to 3D in a straightforward manner [Brock et al. 2016; Fanelli et al. 2011; Maturana and Scherer 2015; Sedaghat et al. 2016a; Tehapmi et al. 2017; Wang et al. 2019a; Wu et al. 2015; Zhi et al. 2018]. The primary drawbacks of volumetric grids are their limited resolution and the heavy computation cost needed.

Point clouds. This representation consists of a set of 3D points, sampled from the object’s surface. The simplicity, close relationship to data acquisition, and the ease of conversion from other representations, make point clouds an attractive representation. Therefore, a variety of recent works proposed successful techniques for point cloud shape analysis using neural networks [Atzmon et al. 2018; Guerrero et al. 2018; Li et al. 2018; Liu et al. 2019; Qi et al. 2017a,b; Wang et al. 2019d; Williams et al. 2019; Xu et al. 2019; Zhu et al. 2019]. These methods attempt to learn a representation for each point, using its neighbors (Euclidean-wise) either by multi layer perceptions or by convolutional layers. Some also define novel pooling layers. Point cloud representations might fall short in applications when the connectivity is highly meaningful (e.g. segmentation) or when the salient information is concentrated in small specific areas.

Triangular meshes. This representation is the most widespread representation in computer graphics and the focus of our paper. The major challenge of using meshes within deep learning frameworks is the irregularity of the representation—each vertex has a different number of neighbors, at different distances.

The pioneering work of [Masci et al. 2015] introduces deep learning of local features and shows how to make the convolution operations intrinsic to the mesh. In [Poulenard and Ovsjanikov 2018] a new convolutional layer is defined, which allows the propagation of geodesic information throughout the network layers. FeaSt-Net [Verma et al. 2018] proposes a graph neural network in which the neighborhood of each vertex for the convolution operation is calculated dynamically based on its features. Another line of works exploits the fact that local patches are approximately Euclidean. The 3D manifolds are then parameterized in 2D, where standard CNNs are utilized [Boscaini et al. 2016; Ezuz et al. 2017; Haim et al. 2019; Henaff et al. 2015; Maron et al. 2017; Sinha et al. 2016]. A different approach is to apply a linear map to a spiral of neighbors [Gong et al. 2019; Lim et al. 2018], which works well for meshes with a similar graph structure.

Two approaches were recently introduced: MeshNet [Feng et al. 2019] treats faces of a mesh as the basic unit and extracts their spatial and structural features individually, to offer the final semantic representation. MeshCNN [Hanocka et al. 2019] is based on a very unique idea of using the edges of the mesh to perform pooling and
convolution. The convolution operations exploit the regularity of edges—having 4 edges of their incidental triangles. An edge collapse operation is used for pooling, which maintains surface topology and generates new mesh connectivity for further convolutions.

2.2 Classification
Object classification refers to the task of classifying a given shape into one of pre-defined categories. Before deep learning methods became widespread, the main challenges were finding good descriptors and good distance functions between these descriptors. According to the thorough review of [Lian et al. 2013], the methods could be roughly classified into algorithms employing local features [Johnson and Hebert 1999; Liu et al. 2006; Lowe 2004; Ovsjanikov et al. 2009; Sun et al. 2009], topological structures [Hilaga et al. 2001; Sundar et al. 2003; Tam and Lau 2007], isometry-invariant global geometric properties [Jain and Zhang 2007; Mahmoudi and Sapiro 2009; Reuter et al. 2005], direct shape matching, or canonical forms [Bronstein et al. 2006; Elad and Kimmel 2003; Mémoli 2007; Mémoli and Sapiro 2005].

Many of the recent techniques already use deep learning for classification. They are described in Section 2.1, for instance [Bronstein et al. 2011; Ezuz et al. 2017; Feng et al. 2019; Hanocka et al. 2018; Kipf and Welling 2016; Li et al. 2018; Liu et al. 2019; Perozzi et al. 2014; Qi et al. 2017a,b; Thomas et al. 2019; Veličković et al. 2017; Wang et al. 2019b].

2.3 Semantic segmentation
Mesh segmentation is a key ingredient in many computer graphics tasks, including modeling, animation and a variety of shape analysis tasks. The goal is to determine, for the basic elements of the mesh (vertex, edge or face), to which segment they belong. Many approaches were proposed, including region growing [Chazelle et al. 1997; Katz et al. 2005; Koschan 2003; Lavoué et al. 2005; Sun et al. 2002; Zhou and Huang 2004], clustering [Attene et al. 2006; Gelfand and Guibas 2004; Katz and Tal 2003; Shlafman et al. 2002], spectral analysis [Alpert and Yao 1995; Gotsman 2003; Liu and Zhang 2004; Zhang et al. 2005] and more. See [Attene et al. 2006; Rodrigues et al. 2018; Shamir 2008] for excellent surveys of segmentation methods.

Lately, deep learning has been utilized for this task as well. Each proposed approach handles a specific shape representation, as described in Section 2.1. These approaches include among others [Guo et al. 2015; Haim et al. 2019; Hanocka et al. 2018; Li et al. 2018; Maron et al. 2017; Qi et al. 2017a,b,b; Yang et al. 2020].

3 MESHWALKER OUTLINE
Imagine an ant walking on a surface; it will "climb" on ridges and go through valleys. Thus, it will explore the local geometry of the surface, as well as the global terrain. Random walks have been shown to incorporate both global and local information about a given object [Grady 2006; Lai et al. 2008; Lovász et al. 1993; Noh and Rieger 2004]. This information may be invaluable for shape analysis tasks, nevertheless, random walks have not been used to represent meshes within a deep learning framework before.

Given a polygonal mesh, we propose to randomly walk through the vertices of the mesh, along its edges, as shown in Fig. 2(a). In our ant analogy, the longer the walk, the more information is acquired by the ant. But how shall this information be accumulated? We propose to feed this representation into a Recurrent Neural Network (RNN) framework, which aggregates properties of the walk. This aggregated information will enable the ant to perceive the shape of the mesh. This is particularly beneficial for shape analysis tasks that require both the 3D global structure and some local information of the mesh, as demonstrated in Fig. 2(b-c).

Algorithm 1 describes the training procedure of our proposed MeshWalker approach. A defining property of it is that the same piece of algorithm is used for every vertex along the walk (i.e., each vertex the ant passes through). The algorithm iterates on the following: A mesh is first extracted from the dataset (it could be a
Walks provide a novel way to organize the mesh data. A walk is a sequence of vertices (not necessarily adjacent), each of which is associated with basic information.

Walk generation. We adopt a very simple strategy to generate walks, out of many possible ones. Recall that we are given the first vertex \( v_{ij} \) of a walk. Then, to generate the walk \( w_{ij} \), the other vertices are iteratively added, as follows. Given the current vertex of the walk, the next vertex is chosen randomly from its adjacent vertices (those that belong to its one-ring neighbors).

If such a vertex does not exist (as all the neighbors already belong to the walk), the walk is tracked backwards until an un-visited neighbor is found; this neighbor is added to the walk. In this case, the length of the walk is set by default to \( V/2.5 \), where \( V \) is number of vertices.

Walk representation. Once the walk \( w_{ij} \) is determined, the representation \( x_{ij} \) of this walk should be defined; this would be the input to the RNN. Each vertex is represented as the 3D translation from the previous vertex in the walk \((ΔX, ΔY, ΔZ)\). This is inline with the deep learning philosophy, which prefers end-to-end learning instead of hand-crafted features that are separated from a classifier. We note that we also tried other representations, including vertex coordinates, normals, and curvatures, but the results did not improve.

Walks at inference time. At inference, several walks are being used for each mesh. Each walk produces a vector of probabilities to belong to the different classes (in the case of classification). These vectors are averaged to produce the final result. To understand the importance of averaging, let us consider the walks on the camel in Fig. 1. Since walks are generated randomly, we expect some of them to explore atypical parts of the model, such as the legs, which are similar to horse legs. Other walks, however, are likely to explore unique parts, such as the hump or the head. The average result will most likely be the camel, as will be shown in Section 5.

4.2 Learning from walks

Once walks are defined, the next challenge is to distillate the information accumulated along a walk into a single descriptor vector. Hereafter we discuss the network architecture and the training.

Network architecture. The model consists of three sub-networks, as illustrated in Fig. 3. The first sub-network is given the current vertex of the walk and learns a new feature space, i.e. it transforms the 3D input feature space into a 256D feature space. This is done by two fully connected (FC) layers, followed by an instance normalization [Ulyanov et al. 2016] layer and ReLu as nonlinear activation; both empirically outperform other alternatives.

The second sub-network is the core of our approach. It utilizes a recurrent neural network (RNN) whose defining property is being able to “remember” and accumulate knowledge. Briefly, a recurrent neural network [Cho et al. 2014; Graves et al. 2008; Hochreiter and Schmidhuber 1997] is a connectionist model that contains a self-connected hidden layer. The benefit of self-connection is that the ‘memory’ of previous inputs remains in the network’s internal state, allowing it to make use of past context. In our setting, the RNN gets as input a feature vector (the result of the previous sub-network), learns the hidden states that describe the walk up to the current vertex, and outputs a state vector that contains the information gathered along the walk.

Another benefit of RNNs, which is crucial in our case, is not being confined to fixed-length inputs or outputs. Thus, we can use the model to inference on a walk of a certain length, which may differ from walk lengths the model was trained on.

To implement the RNN part of our model, we use three Gated Recurrent Unit (GRU) layers of [Cho et al. 2014]. Briefly, the goal of an GRU layer is to accumulate only the important information from the input sequence and to forget the non-important information.

Formally, let \( x_t \) be the input at time \( t \) and \( h_t \) be the hidden state at time \( t \); let the reset gate \( r_t \) and the update gate \( z_t \) be two vectors, which jointly decide which information should be passed from time \( t-1 \) to time \( t \). To realize GRU’s goal, the network performs the following calculation, which sets the hidden state at time \( t \). Its final content is based on updating the hidden state in the previous time (the update gate \( z_t \) determines which information should be passed)
where will later be found to be irrelevant.

⊙ This effectively allows the hidden state to drop any information that

That is, when the reset gate is close to 0, the hidden state ignores

the previous hidden state and resets with the current input only.

5 parameters in our case, whereas GRU.

over LSTM due to its simplicity and its smaller computational re-

quirements. For comparison, LSTM would require 16.8M trainable

parameters in our case, whereas GRU uses 12.7M. Furthermore, the

inference time is smaller—for instance, a single 100-steps walk takes

5mSec using LSTM and 3mSec using GRU.

The third sub-network in Fig. 3 predicts the object class in case of

classification, or the vertex segment in case of semantic segmen-
tation. It consists of a single fully connected (FC) layer on top of the

state vector calculated in the previous sub-network. More details on

the architectures & the implementation are given in Section 6.

Loss calculation. The Softmax cross entropy loss is used on the

output of the third part of the network. In the case of the classifica-
tion task, only the last step of the walk is used as input to the loss

function, since it accumulates all prior information from the walk.

In Fig. 3, this is the bottom-right orange component.

In the case of the segmentation task, each vertex has its own

predicted segment class. Each of the orange components in Fig. 3
classifies the segment that the respected vertex belongs to. Since

at the beginning of the walk the results are not trustworthy (as

the mesh is not yet well understood), for the loss calculation in the

training process we consider the segment class predictions only for

the vertices that belong to the second half of the walk.

5 APPLICATIONS: CLASSIFICATION & SEGMENTATION

MeshWalker is a general approach, which may be applied to a variety of

applications. We demonstrate its performance for two fundamental

tasks in shape analysis: mesh classification and mesh semantic

segmentation. Our results are compared against the reported SOTA

results for recently-used datasets, hence the methods we compare

against vary according to the specific dataset.

5.1 Mesh classification

Given a mesh, the goal is to classify it into one of pre-defined classes.

For the given mesh we generate multiple random walks. These walks

are run through the trained network. For each walk, the network

and on its candidate memory content \( \tilde{h}_t \):

\[
\tilde{h}_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t,
\]

where \( \odot \) is an element-wise multiplication. Here, \( \tilde{h}_t \) is defined as:

\[
\tilde{h}_t = \tanh \left( W^{(h)} x_t + U^{(h)} h_{t-1} \odot r_t + b^{(h)} \right).
\]

That is, when the reset gate is close to 0, the hidden state ignores

the previous hidden state and resets with the current input only.

This effectively allows the hidden state to drop any information that

will later be found to be irrelevant.

Finally, the reset gate \( r_t \) and the update gate \( z_t \) are defined as:

\[
\begin{align*}
   z_t &= \sigma \left( W^{(z)} x_t + U^{(z)} h_{t-1} + b^{(z)} \right), \\
   r_t &= \sigma \left( W^{(r)} x_t + U^{(r)} h_{t-1} + b^{(r)} \right),
\end{align*}
\]

where \( \sigma \) is a logistic Sigmoid function, \( W^{(h)}, W^{(z)}, W^{(r)}, U^{(h)}, U^{(z)} \)

and \( U^{(r)} \) are trainable weight matrices and \( b^{(h)}, b^{(z)}, b^{(r)} \) are train-

able bias vectors. The initial hidden state \( h_0 \) is set to 0.

GRU outperforms a vanilla RNN, due to its ability to both remember

the important information along the sequence and to forget unimportant content. Furthermore, it is capable of processing long

sequences, similarly to the Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber 1997]. Being able to accumulate information

from long sequences is vital for grasping the shape of a 3D model,

which usually consists of thousands of vertices. We chose GRU

over LSTM due to its simplicity and its smaller computational re-

quirements. For comparison, LSTM would require 16.8M trainable

parameters in our case, whereas GRU uses 12.7M. Furthermore, the

inference time is smaller—for instance, a single 100-steps walk takes

5mSec using LSTM and 3mSec using GRU.

The network consists of three components: The first component (FC layers) changes the feature space; the second component (RNN layers) aggregates the information along the walk; and the third component (an FC layer) predicts the outcome of the network. For classification, the prediction of the last vertex of the walk is considered and Softmax is applied to its resulting vector (the bottom-right orange circle, classified as a camel). For segmentation (not shown in this figure), the network is similar. However, Softmax is applied to each of the resulting vectors of the vertices (the orange circles in the right column); each vertex is classified into a segment.

Fig. 3. Network architecture. The network consists of three components: The first component (FC layers) changes the feature space; the second component (RNN layers) aggregates the information along the walk; and the third component (an FC layer) predicts the outcome of the network. For classification, the prediction of the last vertex of the walk is considered and Softmax is applied to its resulting vector (the bottom-right orange circle, classified as a camel). For segmentation (not shown in this figure), the network is similar. However, Softmax is applied to each of the resulting vectors of the vertices (the orange circles in the right column); each vertex is classified into a segment.
predicts the probability of this mesh to belong to each class. These prediction vectors are averaged into a single prediction vector. In practice we use 32 walks; Section 6 will discuss the robustness of MeshWalker to the number of walks.

To test our algorithm, we applied our method to three recently-used datasets: SHREC11 [Lian et al. 2011], engraved cubes [Hanocka et al. 2019] and ModelNet40 [Wu et al. 2015], which differ from each other in the number of classes, the number of objects per class, as well as the type of shapes they contain. As common, the accuracy is defined as the ratio of correctly predicted meshes.

**SHREC11.** This dataset consists of 30 classes, with 20 examples per class. Typical classes are camels, cats, glasses, centaurs, hands etc. Following the setup of [Ezuz et al. 2017], we split the objects in each class into 16 (/10) training examples and 4 (/10) testing examples.

Table 1 compares the performance, where each result is the average of the results of 3 random splits of 16/4 or of 10/10. When the split is 10 objects for training and 10 for testing, the advantage of our method is apparent. When 16 objects are used for training and only 4 for testing, we get the same accuracy as that of the current state-of-the-art. In Section 6.1 we show that indeed the smaller the training dataset, the more advantageous our approach is.

Table 1. **Classification on SHREC11 [Lian et al. 2011].** Split-16 and Split-10 are the number of training models per class (out of 20 models in the class). In both cases our method achieves state-of-the-art results, yet it is most advantageous for a small training dataset (Split-10). (We have not found point cloud-based networks that were tested on SHREC11).

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>Split-16</th>
<th>Split-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshWalker (ours)</td>
<td>Mesh</td>
<td>98.6%</td>
<td>97.1%</td>
</tr>
<tr>
<td>MeshCNN [Hanocka et al. 2019]</td>
<td>Mesh</td>
<td>98.6%</td>
<td>91.0%</td>
</tr>
<tr>
<td>GWCNN [Ezuz et al. 2017]</td>
<td>Mesh</td>
<td>96.6%</td>
<td>90.3%</td>
</tr>
<tr>
<td>SG [Bronstein et al. 2011]</td>
<td>Mesh</td>
<td>70.8%</td>
<td>62.6%</td>
</tr>
</tbody>
</table>

**Cube engraving.** This dataset contains 4600 objects, with 3910/690 training/testing split. Each object is a cube "engraved" with a shape at a random face in a random location, as demonstrated in Fig. 4. The engraved shape belongs to a dataset of 23 classes (e.g., car, heart, apple, etc.), each contains roughly 20 shapes. This dataset was created in order to demonstrate that using meshes, rather than point clouds, may be critical for 3D shape analysis.

Table 2 provides the results. It demonstrates the benefit of our method over state-of-the-art methods.

Table 2. **Classification on Cube Engraving [Hanocka et al. 2019].** Our results outperform those of state-of-the-art algorithms.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshWalker (ours)</td>
<td>Mesh</td>
<td>98.6%</td>
</tr>
<tr>
<td>MeshCNN [Hanocka et al. 2019]</td>
<td>Mesh</td>
<td>92.16%</td>
</tr>
<tr>
<td>PointNet++ [Qi et al. 2017b]</td>
<td>Point cloud</td>
<td>64.26%</td>
</tr>
</tbody>
</table>

**ModelNet40.** This commonly-used dataset contains 12,311 CAD models from 40 categories, out of which 9,843 models are used for training and 2,468 models are used for testing. Unlike previous datasets, many of the objects contain multiple components and are not necessarily watertight, making this dataset prohibitive for some mesh-based methods. However, such models can be handled by MeshWalker since as explained before, if the walk gets into a dead-end during backtracking, it jumps to a new random location.

Table 3 shows that our results outperform those of mesh-based state-of-the-art methods. We note that without 5 classes that are cross-labeled (desk/table & plant/flower-pot/vase) our method’s accuracy is 94.4%. The table shows that multi-views approaches are excellent for this dataset. This is due to relying on networks that are pre-trained on a large number of images. However, they might fail for other datasets, such as the engraved cubes, and do not suit other shape analysis tasks, such as semantic segmentation.

Table 3. **Classification on ModelNet40 [Wu et al. 2015].** MeshWalker is competitive with other mesh-based methods. Multi-view methods are advantageous for this dataset, possibly due to relying on pre-trained networks for image classification and to naturally handling multiple components and non-watertight models, which characterize many meshes in this dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshWalker (ours)</td>
<td>mesh</td>
<td>92.3%</td>
</tr>
<tr>
<td>MeshNet [Feng et al. 2019]</td>
<td>mesh</td>
<td>91.9%</td>
</tr>
<tr>
<td>SNGC [Haim et al. 2019]</td>
<td>mesh</td>
<td>91.6%</td>
</tr>
<tr>
<td>KPConv [Thomas et al. 2019]</td>
<td>point cloud</td>
<td>92.9%</td>
</tr>
<tr>
<td>PointNet [Qi et al. 2017a]</td>
<td>point cloud</td>
<td>89.2%</td>
</tr>
<tr>
<td>RS-CNN [Liu et al. 2019]</td>
<td>point cloud</td>
<td>93.6%</td>
</tr>
<tr>
<td>RotationNet [Kanazaki et al. 2018]</td>
<td>multi-views</td>
<td>97.3%</td>
</tr>
<tr>
<td>GVCNN [Feng et al. 2018b]</td>
<td>multi-views</td>
<td>93.1%</td>
</tr>
<tr>
<td>3D2SeqViews [Han et al. 2019]</td>
<td>multi-views</td>
<td>93.4%</td>
</tr>
</tbody>
</table>

**5.2 Mesh semantic segmentation**

Shape segmentation is an important building block for many applications in shape analysis and synthesis. The goal is to determine, for every vertex, the segment it belongs to. We tested MeshWalker on two datasets: COSEG [Wang et al. 2012] and human-body Segmentation [Maron et al. 2017].
Given mesh, multiple random walks are generated (in practice, \(32 \times \#\) segment classes; see the discussion in Section 6). These walks are run through the trained network, which predicts the probabilities of belonging to the segments. Similarly to the training process, only vertices of the second half of each walk are considered trustworthy. For each vertex, the predictions of the walks it belongs to are averaged. Then, as post-processing, we consider the average prediction of the vertex neighbors and add this average with 0.5 weight. Finally, the prediction for each vertex is the argmax.

Formally, let \(\mathcal{W}\) be the set of walks performed on a mesh. Let \(P_i^v\) be the vector that is the Softmax output for vertex \(v\) from walk \(i\) (if walk \(i\) does not visit \(v\), \(P_i^v\) is set to a 0-vector). Let \(\partial^\text{ring}\) be the list of the vertices adjacent to \(v\) and \(N_v\) be the size of this list. The predicted label, \(l_v\), of vertex \(v\) is defined as (where argmax finds the maximum vector entry):

\[
l_v = \arg\max_i \left( \sum_{i \in \mathcal{W}} P_i^v + \frac{1}{2N_v} \sum_{e \in \partial^\text{ring}} \sum_{i \in \mathcal{W}} P_i^e \right). \tag{5}
\]

We follow the accuracy measure proposed in [Hanocka et al. 2019]: Given the prediction for each edge, the accuracy is defined as the percentage of the correctly-labeled edges, weighted by their length. Since MeshWalker predicts the segment of the vertices, if the predictions of the endpoints of the edge agree, the edge gets the endpoints’ label; otherwise, the label with the higher prediction is chosen. The overall accuracy is the average over all meshes.

**Human-body segmentation.** The dataset consists of 370 training models from SCAPE [Anguelov et al. 2005], FAUST [Bogo et al. 2014], MIT [Vlasic et al. 2008] and Adobe Fuse [Adobe 2016]. The test set consists of 18 humans from SHREC’07 [Giorgi et al. 2007]. The meshes are manually segmented into eight labeled segments according to [Kalogerakis et al. 2010].

There are two common measures of segmentation results, according to the correct classification of faces [Haim et al. 2019] or of edges [Hanocka et al. 2019]. Tables 4 and 5 compare our results to those of previous works, according to the reported measure and the type of objects (simplified or not). Since our method is trained on simplified meshes, to get results on the original meshes, we apply a simple projection to the original meshes jointly with boundary smoothing, as in [Katz and Tal 2003]. In both measures, MeshWalker outperforms other methods. Fig. 5 presents qualitative examples where the difference between the resulting segmentations is evident.

**Table 4. Human-body segmentation results on [Maron et al. 2017].** The accuracy is calculated on edges of the simplified meshes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>Face Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshWalker</td>
<td>Mesh</td>
<td>94.8%</td>
</tr>
<tr>
<td>MeshCNN</td>
<td>Mesh</td>
<td>92.3%</td>
</tr>
</tbody>
</table>

**Table 5. Human-body segmentation results on [Maron et al. 2017].** The reported results are on the original meshes; For MeshCNN, the results shown are ours. Our results outperform those of state-of-the-art algorithms.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input</th>
<th>Face Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeshWalker (ours)</td>
<td>Mesh</td>
<td>92.7%</td>
</tr>
<tr>
<td>MeshCNN [Hanocka et al. 2019]</td>
<td>Mesh</td>
<td>89.0%</td>
</tr>
<tr>
<td>LRF-Conv [Yang et al. 2020]</td>
<td>Mesh</td>
<td>89.9%</td>
</tr>
<tr>
<td>SNGC [Haim et al. 2019]</td>
<td>Mesh</td>
<td>91.3%</td>
</tr>
<tr>
<td>Toric Cover [Maron et al. 2017]</td>
<td>Mesh</td>
<td>88.0%</td>
</tr>
<tr>
<td>GCNN [Masci et al. 2015]</td>
<td>Mesh</td>
<td>86.4%</td>
</tr>
<tr>
<td>MDGCNN</td>
<td>Mesh</td>
<td>89.5%</td>
</tr>
<tr>
<td>[Poulenard and Ovsjanikov 2018]</td>
<td>Mesh</td>
<td>89.5%</td>
</tr>
<tr>
<td>PointNet++ [Qi et al. 2017b]</td>
<td>Point cloud</td>
<td>90.8%</td>
</tr>
<tr>
<td>DynGraphCNN [Wang et al. 2019d]</td>
<td>Point cloud</td>
<td>89.7%</td>
</tr>
</tbody>
</table>
we use at least 16 walks per mesh, a walk whose length is 0.

Though the exact length depends both on the task in hand and on

This result is consistent across all categories and datasets. Table 8

ACM Trans. Graph., Vol. 39, No. 6, Article 263. Publication date: December 2020.

Segmentation results on COSEG [Wang et al. 2012]. Our

Table 6. Segmentation results on COSEG [Wang et al. 2012]. Our

Table 7. Analysis of the training dataset size (COSEG segmentation). “Full” training is 170, 255 and 240 shapes for tele-aliens, vases and chairs, respectively. As expected, the larger the dataset, the better the results. However, even if the training dataset is very small, our results are good.

Table 8. Analysis of the training dataset size (human-body segmentation). As before, the performance of our method degrades gracefully with the size of the training set. We note that the results of MeshCNN are not reported in their paper, but rather the results of new runs of their system.

Table 9 shows that the more walks, the better the accuracy. However, even very few walks result in very good accuracy. In particular, on SHREC11, even with a single walk the accuracy is 90.8%. For the Engraved-Cubes dataset, more walks are needed, since the model is engraved on a single cube facet, which certain walks might not get to. Even in this difficult case, 4 walks already achieve 92.1% accuracy. We note that the STD is between 2.5% for a single walk to 0.4% for 32 walks. As expected, the more walks used, the more stable the results are and the smaller the STD is.

Number of walks. How many walks are needed at inference time? Table 9 shows that the more walks, the better the accuracy. However, even very few walks result in very good accuracy. In particular, on SHREC11, even with a single walk the accuracy is 90.8%. For the Engraved-Cubes dataset, more walks are needed, since the model is engraved on a single cube facet, which certain walks might not get to. Even in this difficult case, 4 walks already achieve 92.1% accuracy. We note that the STD is between 2.5% for a single walk to 0.4% for 32 walks. As expected, the more walks used, the more stable the results are and the smaller the STD is.

Robustness. We use various rotations within data augmentation, hence robustness to orientations. In particular, to test the robustness to rotation, we rotated the models in the Human-body segmentation dataset and in SHREC11 classification dataset 36 times for each axis,

COSEG segmentation. This dataset contains three large classes: aliens, vases and chairs with 200, 300 and 400 shapes, respectively. Each category is split into 85%/15% train/test sets. Fig. 6 presents some qualitative results, where it can be seen that our method performs very well. Table 6 shows the accuracy of our results, where the results of the competitors are reported in [Hanocka et al. 2019]. Our method achieves state-of-the-art results for all categories.

6 EXPERIMENTS

6.1 Ablation study

Size of the training dataset. How many training models are needed in order to achieve good performance? In the 3D case this question is especially important, since creating a dataset is costly. Table 7 shows the accuracy of our model for the COSEG dataset, when trained on different dataset sizes. As expected, the larger the dataset, the better the results. However, even when using only 4 shapes for training, the results are pretty good (80%). This outstanding result can be explained by the fact that we can produce many random walks for each mesh, hence the actual number of training examples is large. This result is consistent across all categories and datasets. Table 8 shows a similar result for the human-body segmentation dataset.

Walk length. Fig. 1 has shown that the accuracy of our method depends on the walk length. What would be an ideal length for our system to “understand” a shape? Fig. 7 analyzes the influence of the length on the task of classification for SHREC11. As expected, the accuracy increases with length. However, it can be seen that when we use at least 16 walks per mesh, a walk whose length is 0.15V suffices to get excellent results. Furthermore, there is a trade-off between the number of walks we use and the length of these walks. Though the exact length depends both on the task in hand and on the dataset, this correlation is consistent across datasets and tasks.

Fig. 1 has shown that the accuracy of our method increases with length. However, it can be seen that when we use at least 16 walks per mesh, a walk whose length is 0.15V suffices to get excellent results. Furthermore, there is a trade-off between the number of walks we use and the length of these walks. Though the exact length depends both on the task in hand and on the dataset, this correlation is consistent across datasets and tasks.

Method Vases Chairs Telealiens Mean
MeshWalker (ours) 98.7% 99.6% 99.1% 99.1%
MeshCNN 97.3% 99.6% 97.6% 98.2%
PointNet++ 94.7% 98.9% 79.1% 90.9%
PointCNN [Li et al. 2018] 96.4% 99.3% 97.4% 97.7%

Fig. 6. Qualitative results of segmentation for meshes from COSEG [Wang et al. 2012].

Fig. 6. Qualitative results of segmentation for meshes from COSEG [Wang et al. 2012].

Fig. 6. Qualitative results of segmentation for meshes from COSEG [Wang et al. 2012].
walks are better than a single 0.6V-length walk. This is because they explore walks. As the walk begins, using many walks is not beneficial since the RNN has not accumulated enough information yet. However, after e.g. 0.3V, two walks are better than a single 0.6V-length walk. This is because they explore different mesh regions.

Table 9. Number of walks analysis. The accuracy improves with the number of walks per shape (demonstrated on 2 datasets).

<table>
<thead>
<tr>
<th># Walks</th>
<th>SHREC11 Acc</th>
<th>Eng.Cubes Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>98.3%</td>
<td>97.6%</td>
</tr>
<tr>
<td>16</td>
<td>97.8%</td>
<td>97.4%</td>
</tr>
<tr>
<td>8</td>
<td>97.8%</td>
<td>95.3%</td>
</tr>
<tr>
<td>4</td>
<td>95.5%</td>
<td>92.1%</td>
</tr>
<tr>
<td>2</td>
<td>95.0%</td>
<td>84.8%</td>
</tr>
<tr>
<td>1</td>
<td>90.8%</td>
<td>77.1%</td>
</tr>
</tbody>
</table>

Fig. 7. Walk length analysis. The accuracy increases with walk length, for classification on SHREC11. Here, the X axis is number of vertices along the walk, normalized by number of mesh vertices. This figure illustrates trade-off between the number of walks we use and the length of these walks. As the walk begins, using many walks is not beneficial since the RNN has not accumulated enough information yet. However, after e.g. 0.3V, two walks are better than a single 0.6V-length walk. This is because they explore different mesh regions.

by increments of $10^0$. For each of these rotated versions of the datasets we applied the same testing as before. For both datasets, there was no difference in the results. Furthermore, the meshes are normalized, hence robustness to scaling.

Our approach is inherently robust to different triangulations, as random walks (representing the same mesh) may vary greatly anyhow. Specifically, we generated a modified version of the COSEG segmentation dataset by randomly perturbing 30% of the vertex positions, realized as a shift towards a random vertex in its 1-ring. The performance degradation is less than 0.1%.

6.2 Implementation

Mesh pre-processing: simplification & data augmentation. All the meshes used for training are first simplified into roughly the same number of faces [Garland and Heckbert 1997; Hoppe 1997] (Mesh-Processing procedure in Algorithm 1). Simplification is analogous to the initial resizing of images. It reduces the network capacity required for training. Moreover, we could use several simplifications for each mesh as a form of data augmentation for training and for testing. For instance, for ModelNet40 we use 1K, 2K and 4K faces. The meshes are normalized into a unit sphere, if necessary.

In addition, we augment the training data and add diversity by rotating the models. As part of batch preparation, each model is randomly rotated in each axis prior to each training iteration.

t-SNE analysis. Does the network produce meaningful features? Fig. 8 opens the network’s “black box” and shows the t-SNE projection to 2D of the multi-dimensional features after each stage of our learning framework, applied to the human-body segmentation task. Each feature vector is colored by its correct label.

In the input layer all the classes are mixed together. The same behavior is noticed after the first two fully-connected layers, since no information is shared between the vertices up to this stage. In the next three GRU layers, semantic meaning evolves: The features are structured as we get deeper in the network. In the last RNN layer the features are meaningful, as the clusters are evident. This visualization demonstrates the importance of the RNN hierarchy.

Fig. 9 reveals another invaluable property of our walks. It shows the t-SNE visualization of walks for classification of objects from 5 categories of SHREC11. Each feature vector is colored by its correct label; its shape (rectangle, triangle etc) represents the object the walk belongs to. Not only clusters of shapes from the same category clearly emerge, but also walks that belong to the same object are grouped together! This is another indication to the quality of our proposed features.

Computation time. Training takes between 5 hours (for classification on SHREC11) to 12 hours (for segmentation on human-body), using GTX 1080 TI graphics card. At inference, a 100-step walk, which is typical for SHREC11, takes about 4 milliseconds. When we use 32 walks per shape, the running time would be 128 milliseconds. Remeshing takes e.g. 4.6 seconds from 400K faces to 1.5K or 0.85 from 100K face to 1.5K faces. We note that our method is easy to parallelize, as every walk could be processed on a different processor, which is yet another benefit of our approach.

Training configurations. We implemented our network using TensorFlow V2. The network architecture is given in Table 10. The source code is available on https://github.com/AlonLahav/MeshWalker.

Table 10. Training configuration

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex description</td>
<td>3</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>128</td>
</tr>
<tr>
<td>Instance Normalization</td>
<td>128</td>
</tr>
<tr>
<td>ReLU</td>
<td>128</td>
</tr>
<tr>
<td>Fully Connected</td>
<td>256</td>
</tr>
<tr>
<td>Instance Normalization</td>
<td>256</td>
</tr>
<tr>
<td>ReLU</td>
<td>256</td>
</tr>
<tr>
<td>GRU</td>
<td>1024</td>
</tr>
<tr>
<td>GRU</td>
<td>1024</td>
</tr>
<tr>
<td>GRU</td>
<td>512</td>
</tr>
<tr>
<td>Fully Connected</td>
<td># of classes</td>
</tr>
</tbody>
</table>
Optimization: To update the network weights, we use Adam optimizer [Kingma and Ba 2014]. The learning rate is set in a cyclic way, as suggested by [Smith 2017]. The initial and the maximum learning rates are set to $10^{-6}$ and $5 \cdot 10^{-4}$ respectively. The cycle size is $20k$ iterations.

Batch strategy: Walks are grouped into batches of 32 walks each. For mesh classification, the walks are generated from different meshes, whereas for semantic segmentation each batch is composed of 4 walks on 8 meshes.

Training iterations: We train for 60k, 60k, 460k, 200k, 200k iterations for SHREC11, COSEG, human-body segmentation, engraved-cubes and ModelNet40 datasets, respectively. This is so since for the loss to converge fast, many of the walks should cover the salient parts of the shape, which distinguish it from other classes/segments. When this is not the case, more iterations are needed in order for the few meaningful walks to influence the loss. This is the case for instance in the engraved cubes dataset, where the salient information lies on a single facet.

6.3 Limitations

Fig. 10 shows a failure of our algorithm, where parts of the hair were wrongly classified as a torso. This is the case since the training data does not contain enough models with hair to learn from. In general, learning-based algorithms rely on good training data, which is not always available.

Another limitation is handling large meshes. The latter require long walks, which in turn might lead to run-time and memory issues. In this paper, this is solved by simplifying the meshes and then projecting the segmentation results onto the original meshes. (For classification, this is not a concern, as simplified meshes may be used).

7 CONCLUSION

This paper has introduced a novel approach for representing meshes within deep learning schemes. The key idea is to represent the mesh by random walks on its surface, which intuitively explore the shape of the mesh. Since walks are described by the order of visiting mesh vertices, they suit deep learning.

Utilizing this representation, the paper has proposed an end-to-end learning framework, termed MeshWalker. The random walks are fed into a Recurrent Neural Network (RNN), that “remembers” the walk’s history (i.e. the geometry of the mesh). Prior works indicated that RNNs are unsuitable for point clouds due to both the unordered nature of the data and the number of vertices used to represent a shape. Surprisingly, we have shown that RNNs work extremely well for meshes, through the concept of random walks.

Our approach is general, yet simple. It has several additional benefits. Most notably, it works well even for extremely small datasets. e.g. even 4 meshes per class suffice to get good results. In addition, the meshes are not required to be watertight or to consist of a single component (as demonstrated by ModelNet40 [Wu et al. 2015]); some other mesh-based approaches impose these conditions and require the meshes to be manifolds.

Last but not least, the power of this approach has been demonstrated for two key tasks in shape analysis: mesh classification and
mesh semantic segmentation. In both cases, we present state-of-the-art results. An interesting question for future work is whether there are optimal walks for meshes, rather than random walks. For instance, are there good starting points of walks? Additionally, reinforcement learning could be utilized to learn good walks. Exploring other applications, such as shape correspondence, is another intriguing future direction. Another interesting practical future work would be to work on the mesh as is, without simplification as pre-processing.

ACKNOWLEDGMENTS
We gratefully acknowledge the support of the Israel Science Foundation (ISF) 1083/18 and PMRI – Peter Munk Research Institute – Technion.

REFERENCES


An interesting question for future work is whether there are optimal walks for meshes, rather than random walks. For instance, are there good starting points of walks? Additionally, reinforcement learning could be utilized to learn good walks. Exploring other applications, such as shape correspondence, is another intriguing future direction. Another interesting practical future work would be to work on the mesh as is, without simplification as pre-processing.


