MeshWalker: Deep Mesh Understanding by Random Walks

Authors: Alon Lahav, Ayellet Tal

Technion

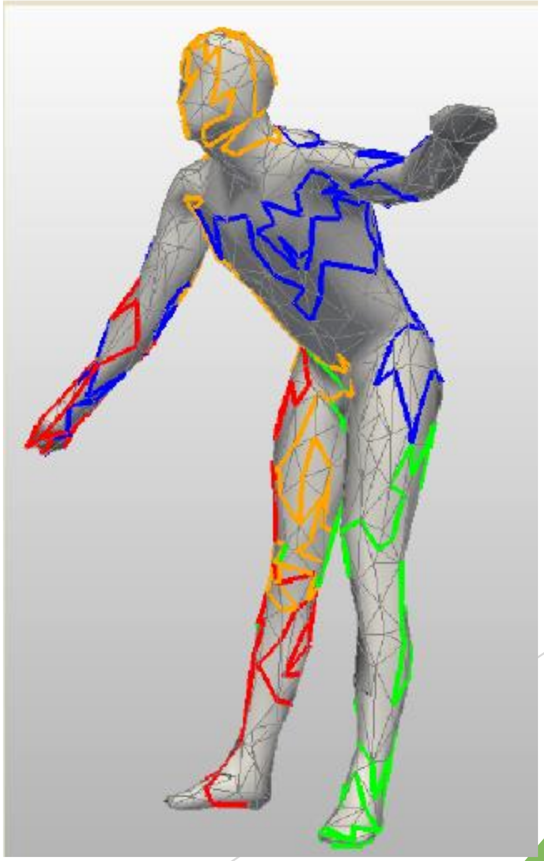
SIGGRAPH ASIA 2020

Presented by: Itay Levy

Goals

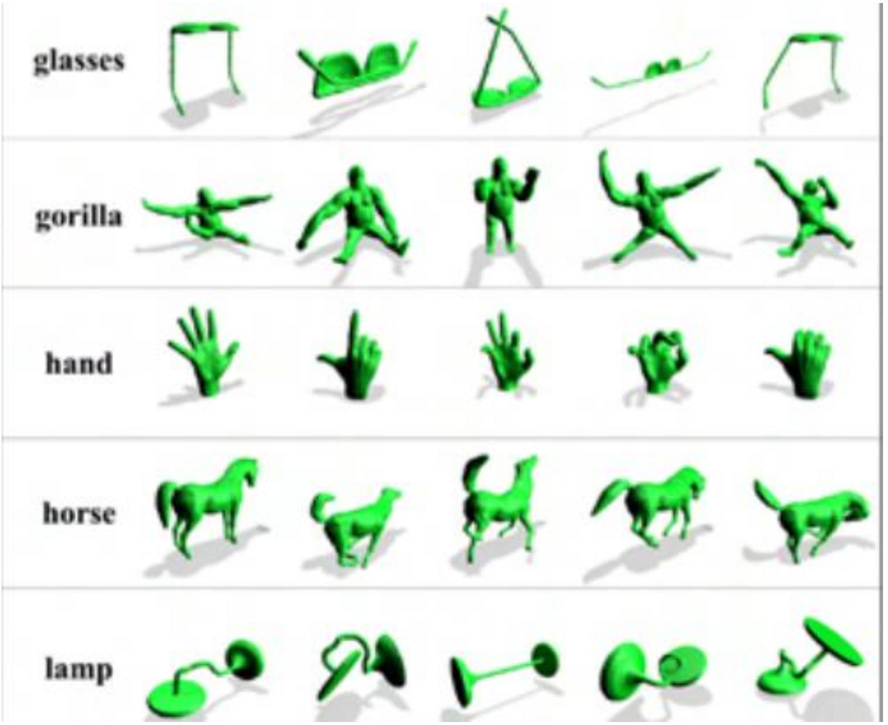
- Finding a mesh representation for DL

- Utilizing it for mesh analysis tasks



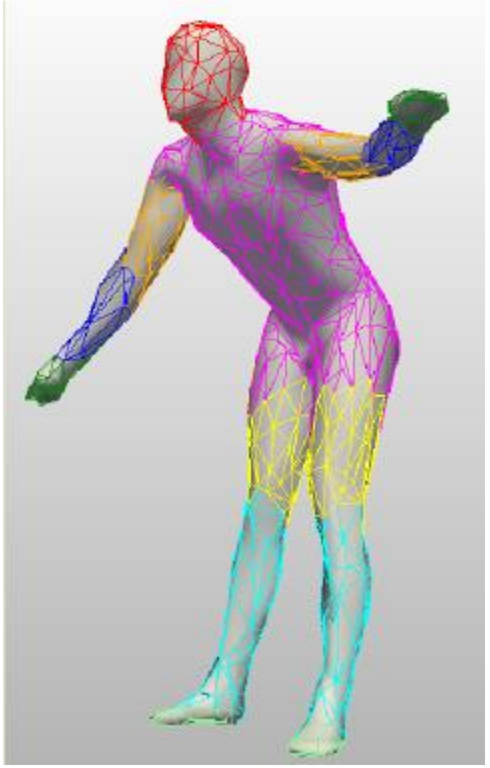
Tasks

Shape classification



Classify a mesh it into one of pre-defined classes

Semantic Segmentation



Determine for every vertex, the segment it belongs to

Benefits of working with meshes

- Most popular representation of 3D shapes in computer graphics
- Adaptive
- Notion of neighborhoods and connectivity

Applying DL to meshes is hard!

- Unordered data

```
v 0.513448 -0.044450 -0.056321
v -0.811377 0.265665 -0.022433
v -0.802689 0.314672 -0.061038
v 0.336913 -0.373204 -0.169275

f 146 142 174
f 105 95 114
f 121 114 126
f 126 134 121
f 168 162 185
f 76 84 104
f 174 161 167
```

Example of .obj file

Applying DL to meshes is hard!

- Unordered data
- Irregularity & non-uniformity
  - Each vertex has a different number of neighbors, at different distances
- Small datasets
  - Hard to obtain clean data

Other Approaches

(Almost) all use CNN

Transforming the input

- Multi-view 2D projections [Su et al. 2015]
- Voxel grid [Maturana and Sherer 2015]

Or

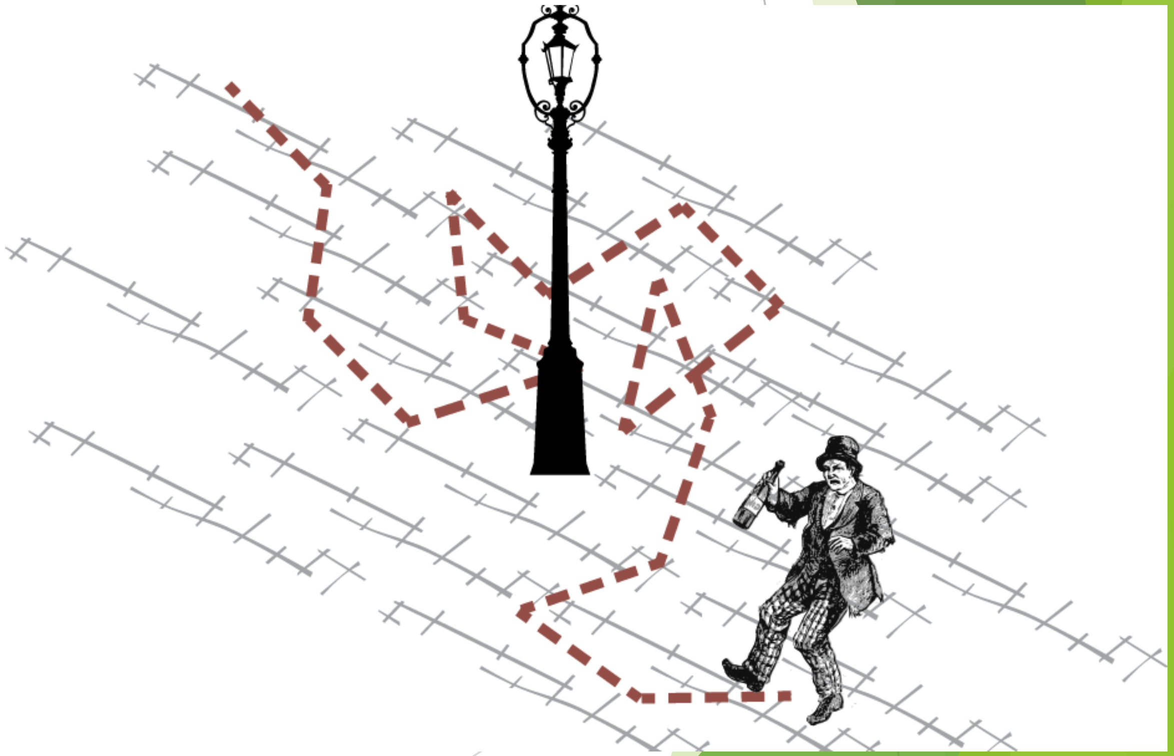
Redefining the basic operations (convolution & pooling)

- FeaStNet [Verma et al. 2018]
- MeshCNN [Hanocka et al. 2019]

MeshWalker - Key Ideas

Random walks on the mesh's surface

- Explore the mesh's local and global geometry
- Impose regularity on the mesh
- Root cause for data efficiency



Feeding the walk to RNN

Methods

What is a walk?

Walk Generation

- Get random starting point
- Iteratively adding vertices
  - next vertex ← randomly chosen from the **unvisited** adjacent vertices
  - If none exist, go back
  - If stuck, jump to a new random vertex

What is a walk?

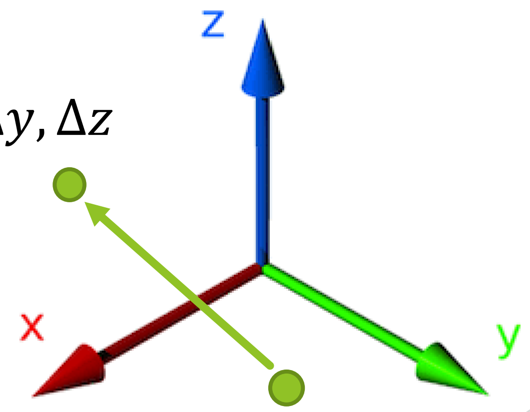
Walk Generation

- Get random starting point
- Iteratively adding vertices
  - next vertex ← randomly chosen from the **unvisited** adjacent vertices
  - If none exist, go back
  - If stuck, jump to a new random vertex

Walk representation

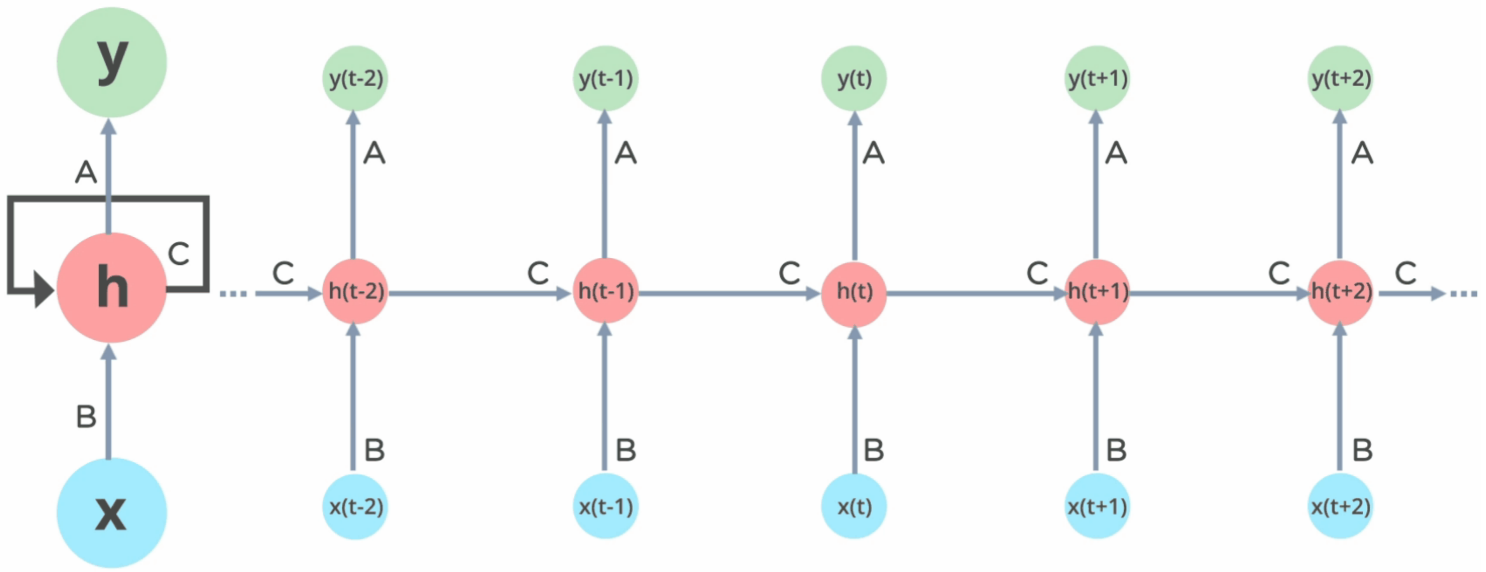
- 3D translation

$\Delta x, \Delta y, \Delta z$



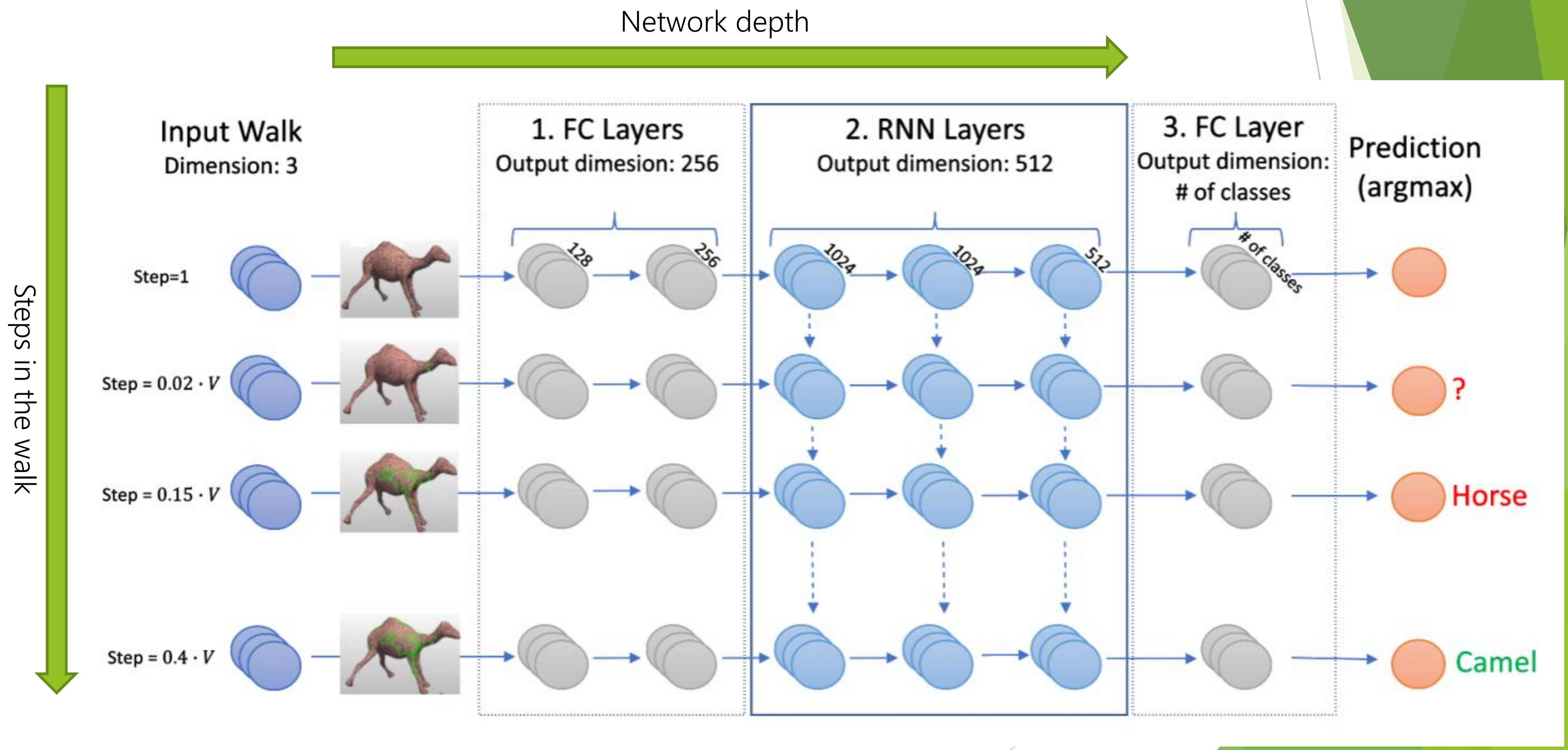
Recurrent Neural Network (RNN)

“Remember" and accumulate knowledge over the entire walk



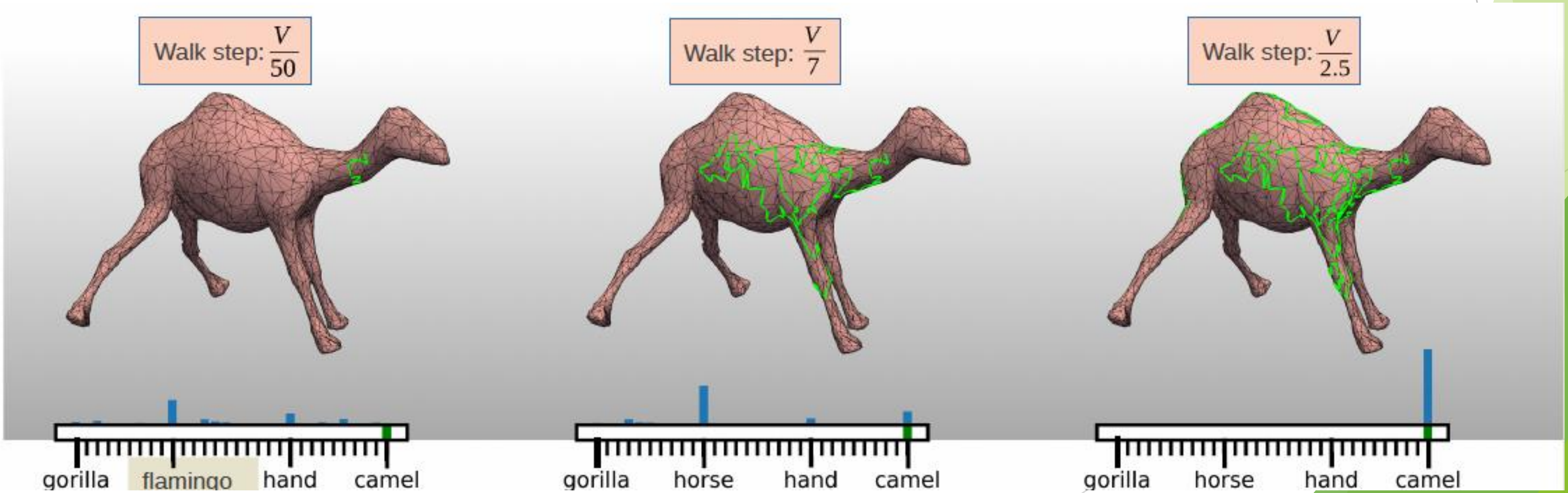
RNNs are not confined to fixed-length inputs

Architecture



Prediction

- Each walk produces a vector of probabilities to belong to the different classes
  - Easily parallelable!
- These vectors are averaged to produce the final result



Handling symmetries

Inherent Invariance

- **Vertex Ordering** – choosing starting point at random
- **Translation** – walk representation
- **Different Triangulations –** random walks vary greatly anyhow

Handling symmetries

Inherent Invariance

- **Vertex Ordering** – choosing starting point at random
- **Translation** – walk representation
- **Different Triangulations –** random walks vary greatly anyhow

Remedies

- **Rotation** - data augmentation

  Adding diversity by rotating the models
- **Scaling** – normalization to unit-sphere
- **Mesh Resolution** - mesh simplification as pre-processing

More details

Softmax Cross Entropy Loss

- Classification
  - only the last step of the walk
- Segmentation
  - Each step, starting from the second half of the walk

Accuracy

- Edge based segmentation
  - the node label with the higher prediction is chosen

Results

Classification

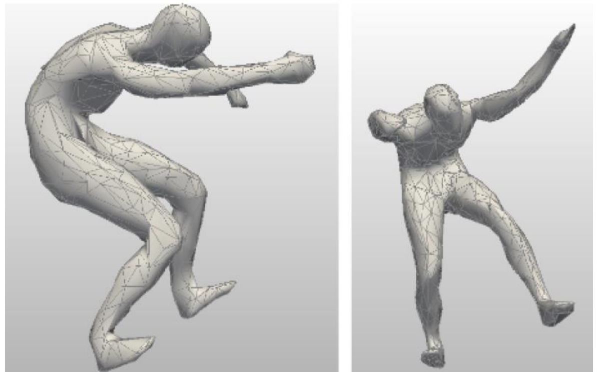


Table 1. **Classification on SHREC11 [Lian et al. 2011].** Split-16 and Split-10 are the number of training models per class (out of 20 models in the class). In both cases our method achieves state-of-the-art results, yet it is most advantageous for a small training dataset (Split-10). (We have not found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

Method	Input	Split-16	Split-10
MeshWalker (ours)	Mesh	**98.6%**	**97.1%**
MeshCNN [Hanocka et al. 2019]	Mesh	**98.6%**	91.0%
GWCNN [Ezuz et al. 2017]	Mesh	96.6%	90.3%
SG [Bronstein et al. 2011]	Mesh	70.8%	62.6%

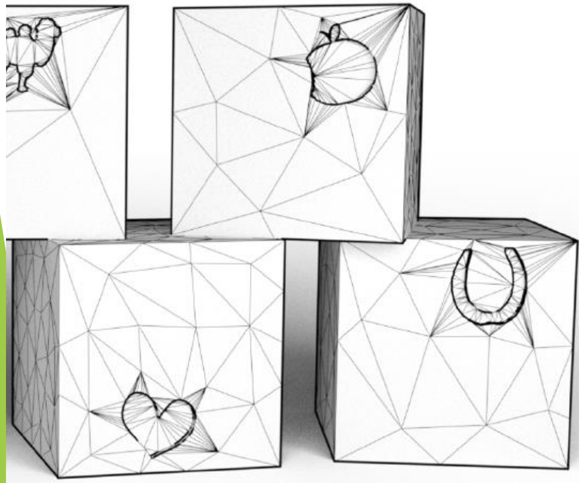


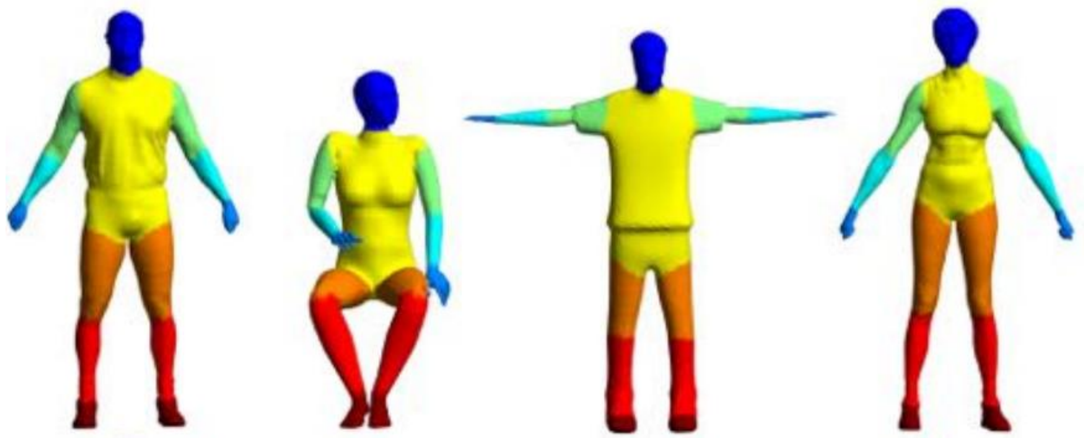
Table 2. **Classification on Cube Engraving [Hanocka et al. 2019].** Our results outperform those of state-of-the-art algorithms.

Method	Input	accuracy
MeshWalker (ours)	Mesh	**98.6%**
MeshCNN [Hanocka et al. 2019]	Mesh	92.16%
PointNet++ [Qi et al. 2017b]	Point cloud	64.26%

Semantic Segmentation

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

Method	Edge Accuracy
MeshWalker	**94.8%**
MeshCNN	92.3%

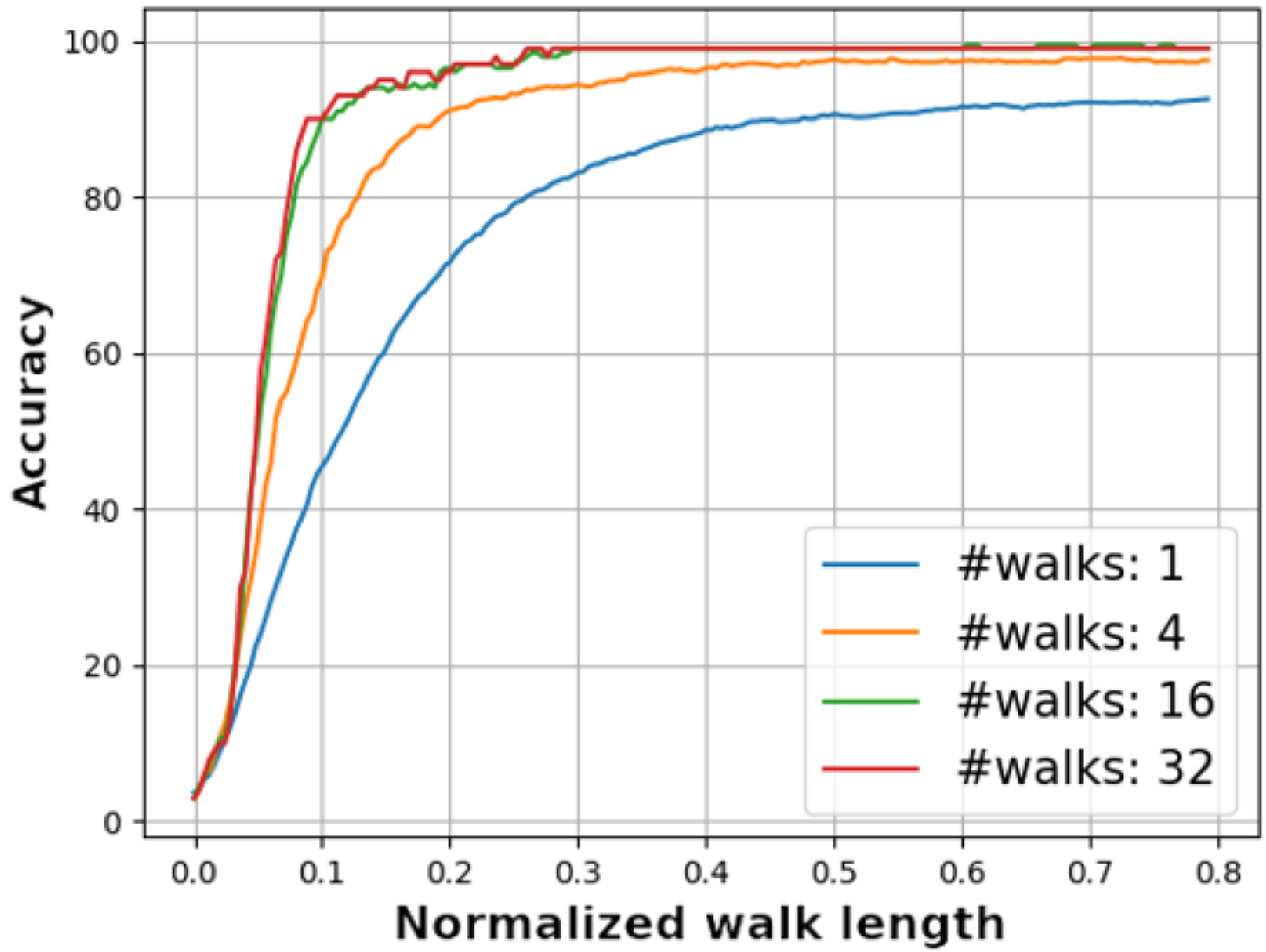


Results stability

Number of walks	Standard deviation
1	2.5%
32	0.4%

Ablations

Effect of walks amount & length

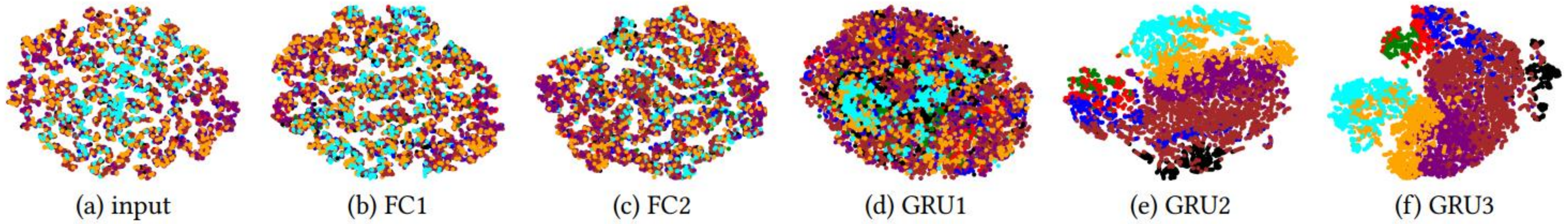


Data-efficiency

Human body segmentation performance

#Training	*MeshWalker*	MeshCNN
381 (Full)	**94.8%**	92.3%
16	**92.0%**	55.7%
4	**84.3%**	48.3%
2	**80.8%**	42.4%

T-SNE of internal layers



Colored by human-body segmentation labels

Limitations & Derivative Works

Limitations

Many iterations till convergence

- 500K - Much more than MeshCNN

Limitations

Many iterations till convergence

- 500K - **Much more than MeshCNN**

Handling large meshes

- long walks → time and memory issues

Walk generation heuristic is quite simplistic

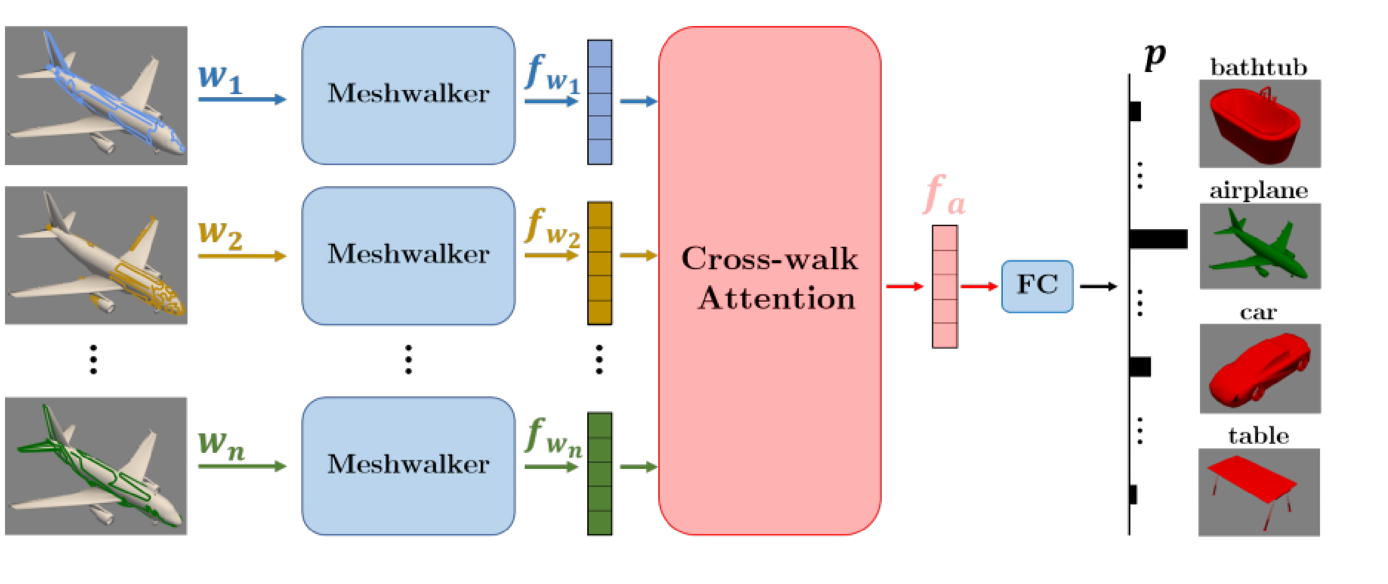
- Decide where to go based on all the information gathered so far

Bad generalization to higher resolution meshes at test time

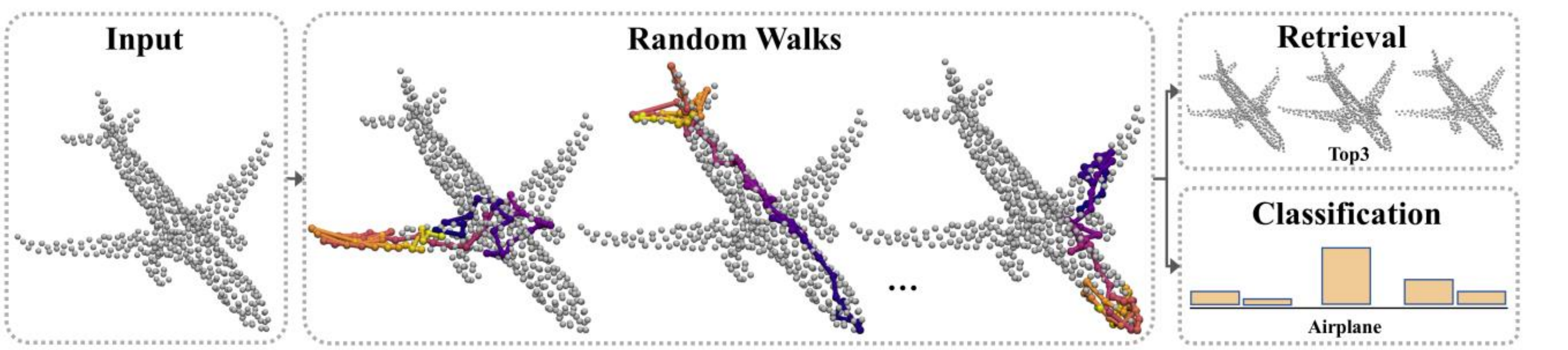
- Despite RNN's length extrapolation ability

Derivative Works

- AttWalk
  - [Ben Izhak et al. 2021]
  - Attention instead of simple averaging



- CloudWalker
  - [Mesika et al. 2021]



Conclusion

Main benefits

Simplicity

Works well for extremely small datasets

- We can produce many random walks for each mesh

Can handle “dirty” triangular meshes

- Mesh need not be watertight or have a single connected component

Conclusion

- Random walks are used to represent the mesh
- Accumulating walk knowledge using RNN
- Data-efficient and parallelizable framework for mesh analysis