UNIVERSITY OF COPENHAGEN

Skinning of Puppeteered Characters using MeshCNN and Depth Vision

Max Kragballe Nielsen

Department of Computer Science

Motivation

Producing high-fidelity animations of articulated 3D objects is a crucial part of the digital entertainment industry. Traditionally, artists have been manually rigging and tweaking 3D meshes to suit a linear blend skinning (LBS) scheme. Rigging is rather straightforward for simple meshes, but manually painting skinning weights on to a mesh can produce undesirable artifacts, such as volume loss and candy-wrapper effects (See Figure 1). In recent years, skinning methods have been proposed based on differentiable models, such as NeuroSkinning [4] and SNARF [1]. These methods have solved some of the traditional LBS issues, but at the cost of requiring large training data sets.

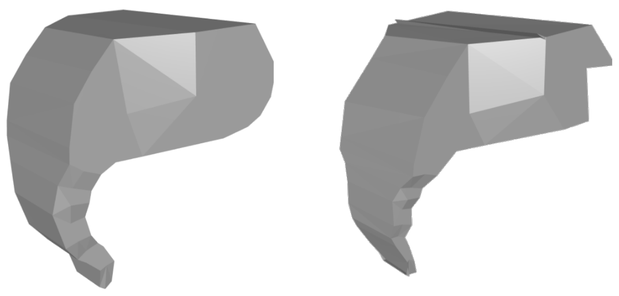

Figure 1: Example of linear blend skinning artifact. *Left:* The elephant head model with proper skinning weights. *Right:* The same model, but with improper skinning weights. Notice how some of the head has been flattened, and how the trunk has lost some of its volume.

Puppeteered Characters

In this work we focus on the case of transferring puppeteered behaviour to a digital representation of the character. Our approach generates skinning weights for a rigged meshed from real data by taking advantage of recent developments in differentiable rendering and Mesh CNNs to ensure the resulting animation is physically realistic.

An example set of puppeteered key frames can be found in Figure 2.

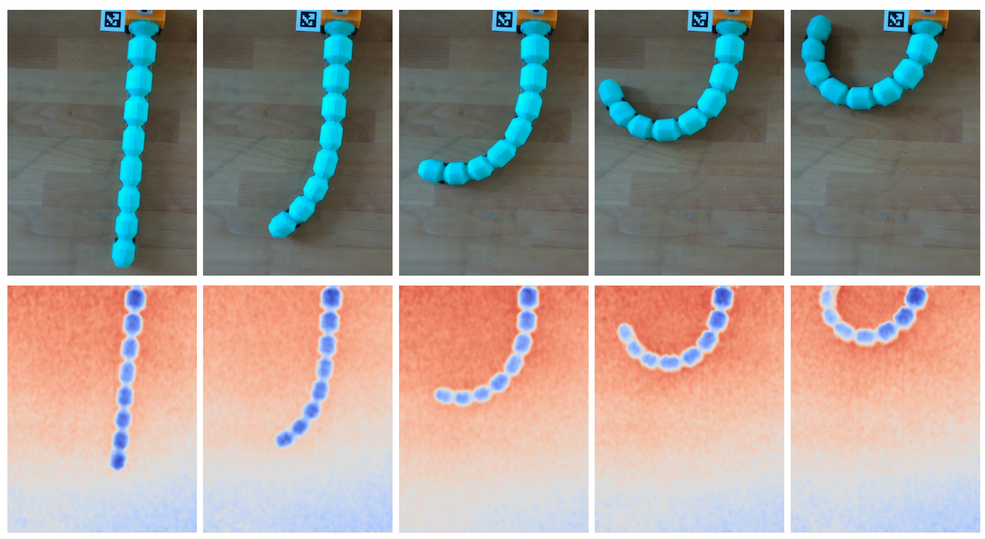

Figure 2: *Top:* Character RGB data. *Bottom:* Corresponding depth map.

Optimizing Skinning Weights

Our method takes as input a rigged polygonal mesh, a target animation - or a depth channel video - and a set of corresponding joint poses. The method can be summarized as,

1. Feeding our mesh through an auto-encoder network using MeshCNN [2], and producing a set of vertex-wise joint weights;

2. Animating the mesh using the provided rig and the joint weight prediction;

3. Rendering the key frames using pyredner [3];

4. Computing the L2-norm between the target animation and the rendered key frames;

5. Updating the weights of the joint weight predictor network;

We repeat this process until we arrive at a set of satisfactory joint weights. See Figure 3 for a visual representation of this process.

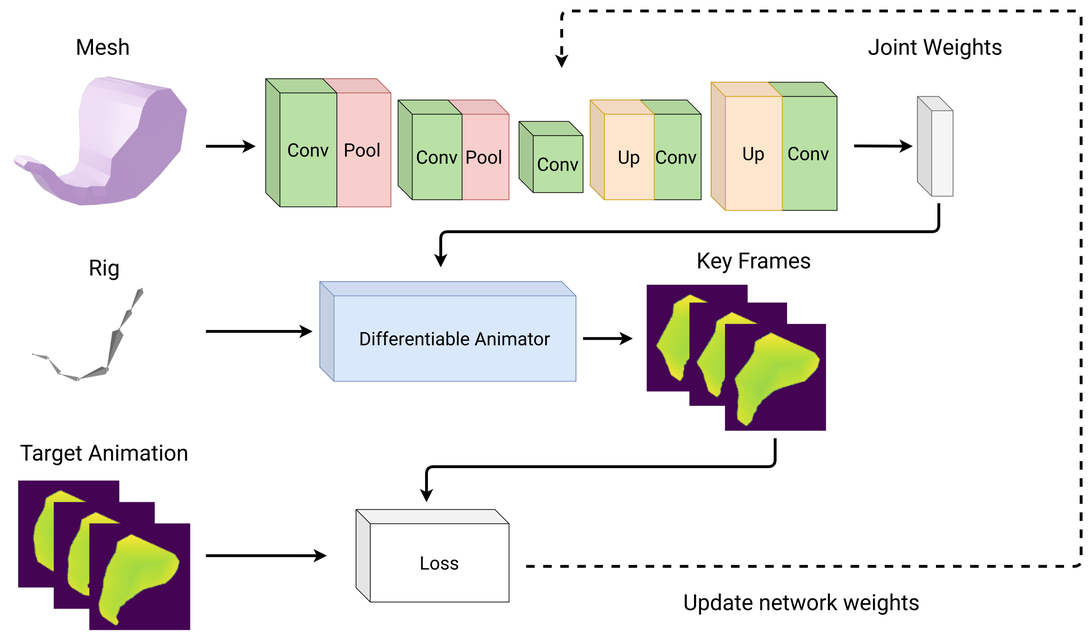

Figure 3: Method overview.

Preliminary Results

In Figure 4, the distribution of vertex-wise joint weights has been painted on to our soft trunk. For this visualization, each joint is assigned a single color, which means that

Notice how the amount of joints influencing each segment decreases over time, as should be expected for this kind of character.

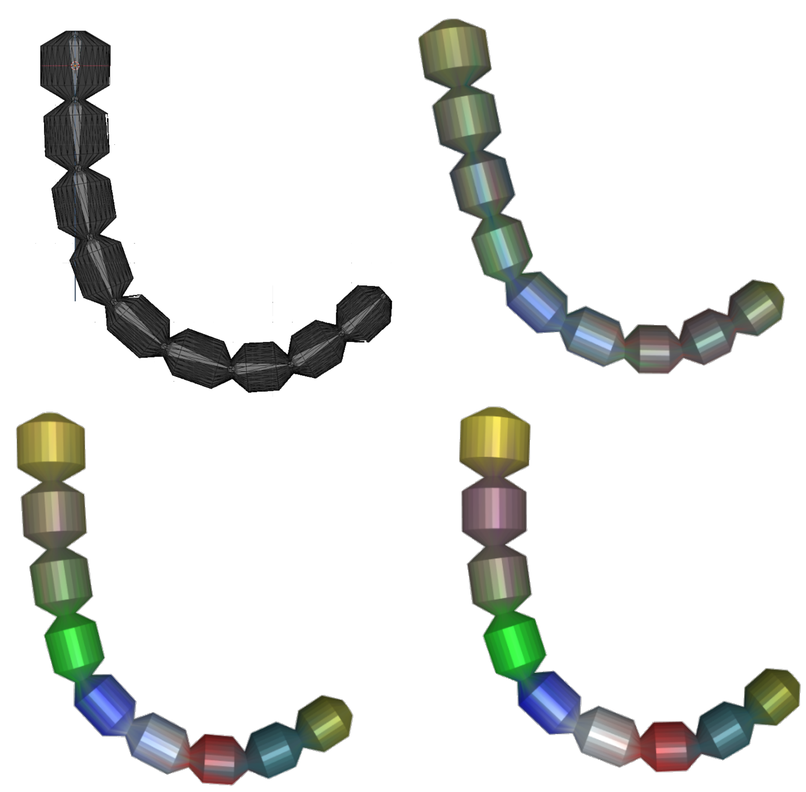

Figure 4: Evolution of skinning weights over time. From top-left to bottom-right we have the skeleton of the mesh and the weight-distribution after $k = (5, 15, 50)$ iterations.

Further Work

For now, we have been able to generate joint weights for a rigged and animated character. Thus, we require the user to animate a skeleton to fit the input data. This work flow can become quite time-consuming, so we want to extent our method to generate the skeleton at the same time as the skinning weights.

References

[1] Xu Chen, Yufeng Zheng, Michael J Black, Otmar Hilliges, and Andreas Geiger. Snarf: Differentiable forward skinning for animating non-rigid neural implicit shapes. *arXiv preprint arXiv:2104.03953*, 2021.

[2] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[3] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

[4] Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. Neuroskinning: Automatic skin binding for production characters with deep graph networks. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.